O'REILLY®

Third Edition

Designing Interfaces

Patterns for Effective Interaction Design

Jenifer Tidwell,
Charles Brewer &
Aynne Valencia

O'REILLY®

Designing Interfaces

Designing good application interfaces isn't easy now that companies need to create compelling, seamless user experiences across an exploding number of channels, screens, and contexts. In this updated third edition, you'll learn how to navigate through the maze of design options. By capturing UI best practices as design patterns, this best-selling book provides solutions to common design problems.

You'll learn patterns for mobile apps, web applications, and desktop software. Each pattern contains full-color examples and practical design advice you can apply immediately. Experienced designers can use this guide as an idea sourcebook, and novices will find a road map to the world of interface and interaction design.

- Understand your users before you start designing
- Build your software's structure so it makes sense to users
- Design components to help users complete tasks on any device
- Learn how to promote wayfinding in your software
- Place elements to guide users to information and functions
- Learn how visual design can make or break product usability
- Display complex data with artful visualizations

Jenifer Tidwell has been researching user interface patterns since 1997 and designing and building complex applications and web interfaces since 1991.

Aynne Valencia is design director for the San Francisco Digital Services and associate professor at California College of the Arts.

Charlie Brewer is a director of product design with deep experience in SaaS software, platforms, and marketplaces.

"Designing Interfaces has been a staple on the shelves of UX designers for years, and this new edition is no exception. This new edition brings important and relevant updates to the collection and belongs in every designer's library or on desks as it will again become the go-to reference for best practices and patterns of interaction going forward. This book is a must-have update!"

—Erin Malone
Principal, Experience Matters Design
Chair, IXD BFA Program,
California College of the Arts

DESIGN

US $59.99 CAN $79.99
ISBN: 978-1-492-05196-1

Twitter: @oreillymedia
facebook.com/oreilly

THIRD EDITION

Designing Interfaces

Patterns for Effective Interaction Design

Jenifer Tidwell, Charles Brewer, and Aynne Valencia

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Designing Interfaces
by Jenifer Tidwell, Charles Brewer, and Aynne Valencia

Printed in Canada.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Jennifer Pollock
Development Editor: Angela Rufino
Production Editor: Christopher Faucher
Copyeditor: Octal Publishing, LLC
Proofreader: Carol Keller

Indexer: Ellen Troutman
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

November 2005: First Edition
December 2010: Second Edition
January 2020: Third Edition

Revision History for the Third Edition
2019-12-18: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781492051961* for release details.

978-1-492-05196-1

[TI]

Table of Contents

Preface to the Third Edition

"The more things change, the more they stay the same."

We're approaching the 15-year anniversary of the original publication of this book, *Designing Interfaces*. And it's been 10 years since the second edition. It's worth looking at what's changed and what hasn't, and what it means for interface design and people who interact with software.

Since then, the big change is that technology and software accelerated their growth and spread in an ever-increasing way. This trend is not stopping. Today, we interact with software in almost every aspect of our daily lives; for work, leisure, communicating, shopping, learning, and more. The list of devices and things with software smarts and internet connectivity is exploding: cars, smart speakers, televisions, toys, watches, homes. Screen sizes and types vary, and an explosion of interfaces that are primarily gesture or voice are found in consumer products. Globally, more than half the population of the planet now accesses the internet. Finally, software is becoming more powerful, more analytical, more predictive, more able to offer smarter insights and operate more independently. In a phrase, it's becoming more like us.

Interface design, like everything else, changes to keep up with the changing times. A third edition that tries to be the comprehensive design guide for all the possible interfaces in this increasing complexity would be enormous and never finished.

Why We Wrote This Book

When the opportunity arose to create the third edition of *Designing Interfaces*, we were excited for a number of reasons. First, we couldn't help but be impressed that we have seen this title regularly on our colleagues' desks and shelves over the years, a constant companion. Indeed, *Designing Interfaces* has been a lasting source of information and inspiration for many designers for a decade and a half. It's a privilege to have a hand in updating this stalwart text.

The timing was right, too. The speed of change in technology and our digital lives has increased dramatically. Design is undergoing rapid change, as well. *Designing Interfaces* needed to be updated. That meant a twin opportunity of bringing forward what makes *this book* special and then sharpening and freshening the focus of the book.

The vision we said "Yes!" to is this: we see the need for a new guidebook to design. It would help make sense of this new state of software design. We wanted to write a guidebook that would have broad appeal, one that would be kept on hand by designers and teams of all stripes, from novice to seasoned. Although it's no longer possible for a single tome to be a guide to all emerging digital channels and specializations, we still wanted a guidebook that would speak to the "home base" of interaction design as we understand it today. For this reason, we decided to focus this third edition on screen-based interaction design for web and mobile. What we removed is outlined in just a moment. Finally, we wanted to write a guidebook that offers a unique point of view. What makes *Designing Interfaces* unique and relevant is obviously its design patterns. We added some patterns of our own, specifically those aspects of human cognition and behavior that influence our design work. We hope we have a guidebook that brings design patterns to a new audience.

Design Patterns Remain Relevant

We asked ourselves, "How is design and *Designing Interfaces* relevant?" The answer is the design patterns. Design pattern s come from the ways people perceive and use software. Human senses and psychology don't change, and these patterns work with these, not against them. The patterns are evergreen also because they are based on the tasks–big and small–that people want to do with software. People will always want to use screens to search for things; enter data; create, control, or manipulate digital objects; manage money and payments; and send and receive information, messages, and files to other people. Design patterns form the building blocks for UIs for any screen. What's more, thinking in patterns and looking for emergent ones is very much in line with how software and interaction design is carried out today.

Software Is Systems Now

Now more than ever, designers, entrepreneurs and developers, and companies have an effective toolset to design and build great software.

The design and software world has evolved into a systems, components, and modules approach. Starting from scratch to design or code something entirely new is not the norm anymore. There are numerous user interface (UI) toolkits and frameworks that allow you to create screen-based interfaces that work across many screen sizes quickly. These component libraries should be regarded as a way to rapidly get to a solid base. They are not a ceiling for design innovation: they are the floor.

The services and middleware that power software are increasingly an integration of separately owned and operated services, as well. Why develop your own registration system when you can sign up with Google, Facebook, and others? Why develop your own analytics and report-building software when you can integrate your choice of robust, customizable business intelligence platforms? Why host your own mobile platform when you have Amazon Web Services? The same is true for all of your HR processes, all of your IT infrastructure. We increasingly assemble, rather than create *de novo*.

Focus: Screen-Based, Web, and Mobile

We chose to focus on screen-based design for the web and mobile devices because that is the majority of what's out there today. Screens are not going away. There will be more screens. In fact, the complexity of what we need to show on these screens is increasing. This will test our skills as designers and builders even more. We'll always need people to design these interfaces.

We reworked the visual design and interaction design chapters to focus on the foundational theories and practices that drive great design. The rest of the book is the discussion of the patterns and how they can be applied.

We've updated the patterns and examples and provided explanations that show how they are relevant today.

What's Not in This Edition

We deliberately did not go into a number of newly emerged and still-emerging areas. Not because they're unimportant; rather, it's because they have their own still-evolving patterns and present special design challenges. Already they represent distinct domains of design. Design books that focus on these unique new fields are here now. To go into the specifics of these areas, look for a design book that specializes in that domain.

Voice

We talk to our phones, our cars, and our smart music speakers at home to make software work for us. We have conversations with the machine. To find out more about designing for voice, we recommend *Designing Voice User Interfaces: Principles of Conversational Experiences* by Cathy Pearl (O'Reilly, 2017).

Social media

Social media has evolved beyond being a way for friends and family to stay connected. It is a communication, discussion, and interaction layer that is present in almost all software. It has revolutionized business communication and productivity. For

more on this, see *Designing Social Interfaces: Principles, Patterns, and Practices for Improving the User Experience* by Christian Crumlish and Erin Malone (O'Reilly, 2015).

Streaming digital television

What we call television is now streaming digital video for entertainment on the screen or device of our choice. The interfaces for this are evolving beyond searching and browsing. TV is an app now, too, with access to all the features and power of our devices. You can read more on this in *Designing Multi-Device Experiences: An Ecosystem Approach to User Experiences across Devices* by Michal Levin (O'Reilly, 2014).

Augmented reality/virtual reality/mixed reality

Interface and software are becoming a layer on top of the physical world or a fully immersive world of its own. Goggles, glasses, and other devices are allowing us to mix the digital world in with what we see in front of us. To learn more, read *Creating Augmented and Virtual Realities: Theory and Practice for Next-Generation Spatial Computing* by Erin Pangilinan et al. (O'Reilly, 2019).

Chatbots and conversational design

Software assistants that seem to be human now talk to us every day via voice, messaging, and chat. These chatbots understand and respond to a conversation in a highly natural-seeming way. Powered by software that recognizes patterns in data and speech, and then learning and improving, chatbots are able to take over the handling of simple information requests and carrying out basic tasks for almost any business or situation. To achieve this capability, designers must create the source data domain and the conversation frameworks and scenarios that make the chatbot learn and become useful. To learn more about designing bots and conversations, check out *Designing Bots* by Amir Shevat (O'Reilly, 2017).

Natural user interfaces—gesture-based interfaces (beyond touch)

This evolving area of design focuses on using the body to interact with technology. Interfaces that you can touch, hold, squeeze or wave at; experiences that you can trigger by the movement of the hands, feet, or by moving around in space.

Who This Book Is For

We hope *Designing Interfaces* reaches current and new audiences. We created it to be of interest and value to many different people. It's for design beginners, mid-career practitioners and managers, seasoned professionals, and executives. It's for people who want to learn, to get a refresher, and to get inspiration and a new point of view. It's for teams, classes, and individuals. It's for interaction designers, information

architects, product designers, UX/UI designers, product managers, developers, QA engineers, strategists, managers, leaders, consultants, teachers, students, and anyone who is interested in designing better software.

How This Book Is Organized

This book now has 12 chapters—some new, some substantially updated. The chapters themselves mostly follow a standard two-part structure.

Introduction and Design Discussion

The first half of each chapter is focused on introducing the topic and then expanding on it. This includes a discussion of the theory and practice of design as it relates to the chapter topic. Design principles, guidelines, and recommended best practices are reviewed. This sets the context for the second half of the chapter.

The Patterns

The patterns are concrete bundles of components and functions that help make software more usable and more useful. The second half of each chapter is dedicated to a selection of software design patterns. The selection is not an exhaustive list. There are many more out there. Each pattern is broken down into the following structure:

What

A definition of the design pattern.

Use when

This section covers the likely scenarios for use. The context for using the pattern is explained, along with any special considerations or exceptions.

Why

This section analyzes the purpose and benefits of the design pattern. This includes who might benefit and what kind of benefits can be intended or expected.

How

This section speaks most closely to the design of the pattern itself and the means of implementing it. What you should do to use the pattern well and when it is effective are outlined.

Examples

This last section provides a series of screenshots from different web and mobile properties that illustrate the selected pattern. Each example is described and analyzed.

Conclusion

We believe more people than ever are designing and building software. The tools are there. We want a guidebook for this new state of software design that makes it easy to understand and easy to implement. We wanted to write the handbook for web and mobile screen design that we would like to have at hand on our desks, a guide to give early-career designers, and a reference for product managers, engineers, and executive management. We hope that it becomes a useful reference that gives a common vocabulary for designing interfaces.

We see that consumer software experiences are an ever-present part of our lives now. We're spending more time than ever using software interfaces. They should make life easier, not more difficult.

Although new modes, devices, and formats are rapidly emerging, screens are with us now and will be with us for a long time. We will be typing, tapping, and touching screens to get jobs done and to entertain ourselves, to find something, to buy something, and learn something. We hope the principles and examples in this book will give you the knowledge and confidence to use these proven patterns to create great products and services, great design, and great experiences for everyone.

— Jenifer Tidwell, Charlie Brewer, and Aynne Valencia

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates pattern names, new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

O'Reilly Online Learning

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, conferences, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, please visit *http://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/designing-interfaces-3e*.

Email *bookquestions@oreilly.com* to comment or ask technical questions about this book.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

We would like to thank our technical reviewers, Erin Malone, Kate Rutter, Frances Close, Christy Ennis Kloote, Matthew Russell, and George K. Abraham.

Thank you to Christian Crumlish, for connecting us to this opportunity.

And a big thank you to Angela Rufino and Jennifer Pollock from O'Reilly.

Aynne Valencia: I would like to acknowledge the stellar faculty of IxD@CCA, and the students I have had at SFSU, CCA, and GA over the years: you are a constant source of inspiration and give me abundant hope for the future. And mostly, I thank my husband, Don Brooks, for being my travel partner and for always defaulting to yes.

Charlie Brewer: I would like to thank the following people: the people at O'Reilly, for the chance to try something new professionally; the product team at SpaceIQ, for schedule flexibility; and especially my family and friends who have encouraged me in this effort.

Chapter 1
Designing for People

This book is almost entirely about the look and behavior of applications, web apps, and interactive devices. But this first chapter is the exception to the rule. No screenshots here, no layouts, no navigation, no diagrams, no visuals at all.

Why not? After all, that's probably why you picked up the book in the first place.

In this first chapter, we outline the purpose and outcomes of understanding how people use software. Specifically, you'll get a sense for what is critical to people when it comes to designing websites, applications and interfaces:

- Overall goals in using your site or application
- The breakdown of tasks in carrying out those goals
- How they think about the particular subject or domain
- The language they use to think and talk about this subject
- How skilled or unskilled they are in doing the work
- Their attitudes toward the subject

Good interface design doesn't begin with pictures. It begins with an understanding of people: what they're like, why they use a given piece of software, and how they might interact with it. The more you know about them and the more you empathize with them, the more effectively you can design for them. Software, after all, is merely a means to an end for the people who use it. The better you satisfy those ends, the happier those users will be.

A framework for achieving this is described here. It covers four areas. These are not strict rules or requirements for creating great designs. But having a plan for how you will inform yourself and your team in each area will give you confidence that your work is based on real insights into valuable problems to solve for your target

customers. Decide for yourself what level of time and effort is appropriate for your project or company. Revisiting these areas regularly will keep key insights at the top of your mind and help focus everyone's effort, especially user interface (UI) design, on creating great outcomes for people.

The four-part structure for understanding design for people is this:

Context
: Who is your audience?

Goals
: What are they trying to do?

Research
: Ways to understand context and goals

The Patterns
: Cognition and behavior related to interface design

Context

The first major element and the first step in designing for people is to understand the human context for your design intention. Interaction design starts with defining and understanding the people who will use your design. Specifically, ground your design decisions in understanding what they may want to do and what they bring to the interaction in terms of expectations, knowledge of relevant subjects or information domains, and their skill level with software.

Know Your Audience

There's a maxim in the field of interface design: "You are not the user."

So, this chapter talks about people. It covers a few fundamental ideas briefly in this introduction, and then discusses some patterns that differ from those in the rest of the book. They describe human behaviors—as opposed to system behaviors—that the software you design might need to support. Software that supports these human behaviors better helps users achieve their goals.

Interactions Are Conversations

Each time someone uses an application, or any digital product, they carry on a conversation with the machine. It might be literal, as with a command line or phone menu, or tacit, like the "conversation" an artist has with their paints and canvas—the give and take between the craftsperson and the thing being built. With social software, it can even be a conversation by proxy. Whatever the case, the UI mediates that conversation, helping users achieve whatever ends they had in mind.

Here are the key points:

- There are two participants in the conversation: the person and the software.
- There is a constant back and forth exchange of information.
- The exchange is a series of requests, commands, reception, processing, and response.
- The human in the conversation needs continuous feedback from the interface that confirms that things are working normally, inputs are being processed, and that they are proceeding satisfactorily toward the goal of the moment.
- For this feedback loop to work, the software—which can't be as spontaneous and responsive as a real human (at least not yet)—should be designed to mimic a conversation partner. It should be understandable to its partner, it should indicate it's active (when it's "listening"), and it should be obvious when it's responding. Another layer on this is having some anticipated next steps or recommendations, in the same way a considerate person might be helpful to another.

As the UI designer, then, you get to script that conversation, or at least define its terms. And if you're going to script a conversation, you should understand the human's side as well as possible. What are the user's motives and intentions? What "vocabulary" of words, icons, and gestures does the user expect to employ? How can the application set expectations appropriately for the user? How do the user and the machine finally end up communicating meaningfully with each other?

Match Your Content and Functionality to Your Audience

Before you start the design process, consider your overall approach. Think about how you might design the interface's overall interaction style—its personality, if you will.

When you carry on a conversation with someone about a given subject, you adjust what you say according to your understanding of the other person. You might consider how much they care about the subject, how much they already know about it, how receptive they are to learning from you, and whether they're even interested in the conversation in the first place. If you get any of that wrong, bad things happen—the person might feel patronized, uninterested, impatient, or utterly baffled.

This analogy leads to some obvious design advice. The subject-specific vocabulary you use in your interface, for instance, should match your users' level of knowledge; if some users won't know that vocabulary, give them a way to learn the unfamiliar terms. If they don't know computers very well, don't make them use sophisticated widgetry or uncommon interface-design conventions. If their level of interest might be low, respect that, and don't ask for too much effort for too little reward.

Some of these concerns permeate the entire interface design in subtle ways. For example, do your users expect a short, tightly focused exchange about something very specific, or do they prefer a conversation that's more of a free-ranging exploration? In other words, how much openness is there in the interface? Too little, and your users feel trapped and unsatisfied; too much, and they stand there paralyzed, not knowing what to do next, unprepared for that level of interaction.

Therefore, you need to choose how much freedom your users have to act arbitrarily. At one end of the scale might be a software installation wizard: the user is carried through it with no opportunity to use anything other than Next, Previous, or Cancel. It's tightly focused and specific, but quite efficient—and satisfying, to the extent that it works and is quick. At the other end might be an application such as Excel, an "open floorplan" interface that exposes a huge number of features in one place. At any given time, the user has hundreds of things that they could do next, but that's considered good, because self-directed, skilled users can do a lot with that interface. Again, it's satisfying, but for entirely different reasons.

Skill Level

How well can your audience use your interface now and how much effort are your users willing to spend to learn it?

Some of your customers might use it every day on the job—clearly, they'd become an expert user over time. But they will become increasingly unhappy with even small dissatisfactions. Maybe they'll use it sometimes and learn it only well enough to get by (Satisficing). Difficulties in usage can be tolerated more. Maybe they'll use it only once. Be honest: can you expect most users to become intermediates or experts, or will most users remain perpetual beginners?

Software designed for intermediate-to-expert users include the following:

- Photoshop
- Excel
- Code development environments
- System-administration tools for web servers

In contrast, here are some things designed for occasional users:

- Kiosks in tourist centers or museums
- Windows or macOS controls for setting desktop backgrounds
- Purchase pages for online stores
- Installation wizards

- Automated teller machines

The differences between the two groups are dramatic. Assumptions about users' tool knowledge permeate these interfaces, showing up in their screen-space usage, labeling, and widget sophistication, and in the places where help is (or isn't) offered.

The applications in the first group have lots of complex functionality, but they don't generally walk the user through tasks step by step. They assume that users already know what to do, and they optimize for efficient operation, not learnability; they tend to be document centered or list driven (with a few being command-line applications). They often have entire books and courses written about them. Their learning curves are steep.

The applications in the second group are the opposite: restrained in functionality but helpful about explaining it along the way. They present simplified interfaces, assuming no prior knowledge of document- or list-centered application styles (e.g., menu bars, multiple selection, etc.). "Wizards" frequently show up, removing attention-focusing responsibility from the user. The key is that users aren't motivated to work hard at learning these applications—it's usually just not worth it!

Now that you've seen the extremes, look at the applications in the middle of the continuum:

- Microsoft PowerPoint
- Email clients
- Facebook
- Blog-writing tools

The truth is that most applications fall into this middle ground. They need to serve people on both ends adequately—to help new users learn the tool (and satisfy their need for instant gratification) while enabling frequent-user intermediates to get things done smoothly. Their designers probably knew that people wouldn't take a three-day course to learn an email client. Yet the interfaces hold up under repeated usage. People quickly learn the basics, reach a proficiency level that satisfies them, and don't bother learning more until they are motivated to do so for specific purposes.

You might someday find yourself in tension between the two ends of this spectrum. Naturally you want people to be able to use your design "out of the box," but you might also want to support frequent or expert users as much as possible. Find a balance that works for your situation. Organizational patterns in Chapter 2, such as *Help Systems*, can help you serve both constituencies.

Goals: Your Interface Is Just a Means to Their Ends

Everyone who uses a tool—software or otherwise—has a reason for using it. These are their goals. Goals could be outcomes such as these:

- Finding some fact or object
- Learning something
- Performing a transaction
- Controlling or monitoring something
- Creating something
- Conversing with other people
- Being entertained

Well-known idioms, user behaviors, and design patterns can support each of these abstract goals. User experience (UX) designers have learned, for example, how to help people search through vast amounts of online information for specific facts. They've learned how to present tasks so that it's easy to walk through them. They're learning ways to support the building of documents, illustrations, and code.

Ask Why

The first step in designing an interface is to learn what its users are really trying to accomplish. Filling out a form, for example, is almost never a goal in and of itself—people only do it because they're trying to buy something online, renew their driver's license, or install software. They're performing some kind of transaction.

Asking the proper questions can help you connect user goals to the design process. Users and clients typically speak to you in terms of desired features and solutions, not of needs and problems. When a user or client tells you that they want a certain feature, ask why they want it—determine their immediate goal. Then, to the answer of this question, ask "why" again. And again. Keep asking until you move well beyond the boundaries of the immediate design problem.[1]

Design's Value: Solve the Right Problem, and Then Solve It Right

Why should you ask these questions if you have clear requirements? Because if you love designing things, it's easy to get caught up in an interesting interface design problem. Maybe you're good at building forms that ask for just the right information,

1 This is the same principle that underlies a well-known technique called *root-cause analysis*. But root-cause analysis is a tool for fixing organizational failures; here, we use its "five why's" (more or less) to understand everyday user behaviors and feature requests.

with the right controls, all laid out nicely. But the real art of interface design lies in solving the right problem, defined as helping the user achieve their goal.

So, don't get too fond of designing that form. If there's any way to finish the transaction without making the user go through that form at all, get rid of it altogether. That gets the user closer to their goal, with less time and effort spent on their part (and maybe yours, too).

Let's use the "why" approach to dig a little deeper into some typical design scenarios:

- Why does a mid-level manager use an email client? Yes, of course—"to read email." But, why do they read and send email in the first place? To converse with other people. Of course, other means can achieve the same ends: the phone, a hallway conversation, a formal document. But apparently, email fills some needs that the other methods don't. What are they, and why are they important to this manager? The convenience of choosing when to send or respond? Privacy? The ability to archive a conversation? Social convention? What else?
- A father goes to an online travel agent, types in the city where his family will be taking a summer vacation, and tries to find plane ticket prices on various dates. He's learning from what he finds, but his goal isn't just to browse and explore different options. Ask why. His goal is actually a transaction: to buy plane tickets. Again, he could have done that at many different websites, or over the phone with a live travel agent. How is this site better than those other options? Is it faster? Friendlier? More likely to find a better deal?
- Sometimes, goal analysis really isn't enough. A snowboarding site might provide information (for learning), an online store (for transactions), and a set of video clips (for entertainment). Suppose that someone visits the site for a purchase, but they get sidetracked into the information on snowboarding tricks—they have switched goals from accomplishing a transaction to browsing and learning. Maybe they'll go back to purchasing something, maybe not. And does the lifestyle and entertainment part of the site successfully entertain both the 12-year-old and the 35-year-old? Will the 35-year-old go elsewhere to buy their new board if they don't feel at home there, or do they not care? It's useful to expand your goal framework to include an understanding of the specific business purchase cycle. Your snowboarding customer will have different goals at different stages of this cycle. Alternately, you might want to design how you could foster a long-term loyalty between the brand and the customer. This could be done via content and functionality that fosters an identity, builds a community, and celebrates a lifestyle.

It's deceptively easy to model users as a single faceless entity—"The User"—walking through a set of simple use cases, with one task-oriented goal in mind. But that won't necessarily reflect your users' reality.

To do design well, you need to take many "softer" factors into account: expectations, gut reactions, preferences, social context, beliefs, and values. All of these factors could affect the design of an application or site. Among these softer factors, you might find the critical feature or design factor that makes your application more appealing and successful.

So, be curious. Specialize in finding out what your users are really like, and what they really think and feel.

Research: Ways to Understand Context and Goals

Research is the starting point in design for understanding people. Empirical discovery is the only really good way to obtain this information. Qualitative research such as one-on-one interviews gives you the basis for understanding your audience's expectations, vocabulary, and how they think about their goals or structure their work. You can often detect patterns in what you're hearing. These are your signals for guiding the design. Quantitative research such as a survey can give numerical validation or disqualification to your quant findings.

To start a design, you'll need to characterize the kinds of people who will be using your design (including the softer factors just mentioned), and the best way to do that is to go out and meet them.

Each user group is unique, of course. The target audience for, say, a new mobile phone app will differ dramatically from the target audience for a piece of scientific software. Even if the same person uses both, their expectations for each are different—a researcher using scientific software might tolerate a less-polished interface in exchange for high functionality, whereas that same person might stop using the mobile app if they find its UI to be too difficult to use after a few days.

Each user is unique, too. What one person finds difficult, the next one won't. The trick is to figure out what's generally true about your users, which means learning about enough individual users to separate the quirks from the common behavior patterns.

Specifically, you'll want to learn the following:

- Their goals in using the software or site
- The specific tasks they undertake in pursuit of those goals
- The language and words they use to describe what they're doing
- Their skill at using software similar to what you're designing
- Their attitudes toward the kind of thing you're designing, and how different designs might affect those attitudes

I can't tell you what your particular target audience is like. You need to find out what they might do with the software or site, and how it fits into the broader context of their lives. Difficult though it may be, try to describe your potential audience in terms of how and why they might use your software. You might get several distinct answers, representing distinct user groups; that's OK. You might be tempted to throw up your hands and say, "I don't know who the users are," or, "Everyone is a potential user." But that doesn't help you focus your design at all—without a concrete and honest description of those people, your design will proceed with no grounding in reality.

This user-discovery phase will consume time and resources early in the design cycle, especially if you don't really have a handle on who your audience is and why they might use your designs. It's an investment. It's worth it because the understanding you and the team gain gives long-term payback in better designs: solving the correct problems, and fit for purpose.

Fortunately, lots of books, courses, and methodologies now exist to help you. Although this book does not address user research, following are some methods and topics to consider.

Direct Observation

Interviews and onsite user visits put you directly into the user's world. You can ask users about what their goals are and what tasks they typically do. Usually done "on location," where users would actually use the software (e.g., in a workplace or at home), interviews can be structured—with a predefined set of questions—or unstructured, in which you probe whatever subject comes up. Interviews give you a lot of flexibility; you can do many or a few, long or short, formal or informal, on the phone or in person. These are great opportunities to learn what you don't know. Ask why. Ask it again.

Case Studies

Case studies give you deep, detailed views into a few representative users or groups of users. You can sometimes use them to explore "extreme" users that push the boundaries of what the software can do, especially when the goal is a redesign of existing software. You can also use them as longitudinal studies—exploring the context of use over months or even years. Finally, if you're designing custom software for a single user or site, you'll want to learn as much as possible about the actual context of use.

Surveys

Written surveys can collect information from many users. You can actually get statistically significant numbers of respondents with these. Because there's no direct human contact, you will miss a lot of extra information—whatever you don't ask about, you won't learn about—but you can get a very clear picture of certain aspects

of your target audience. Careful survey design is essential. If you want reliable numbers instead of a qualitative "feel" for the target audience, you absolutely must write the questions correctly, pick the survey recipients correctly, and analyze the answers correctly—and that's a science.

You can find some further guidelines for writing effective survey questions here:

- Writing Good Survey Questions
- Questionnaire Design Tip Sheet

Personas

Personas aren't a data-gathering method, but they do help you figure out what to do with your data after you have it. This is a design technique that models the target audiences. For each major user group, you create a fictional person, or "proto-person," who captures the most important aspects of the users in that group: what tasks they're trying to accomplish, their ultimate goals, and their experience levels in the subject domain and with computers in general. Personas can help you stay focused. As your design proceeds, you can ask yourself questions such as, "Would this fictional person really do X? What would she do instead?"

Design Research Is Not Marketing Research

You might notice that some of these methods and topics, such as interviews and surveys, sound suspiciously like marketing activities. They are closely related. Focus groups, for example, can be useful, but be careful. In group settings, not everyone will speak up, and just one or two people might dominate the discussion and skew your understanding. There is also the very robust marketing practice of market segmentation. It resembles the definition of target audiences used here, but market segments are defined by demographics, psychographics, and other characteristics. Target audiences from a UI design perspective are defined by their task goals and behaviors.

In both cases, the whole point is to understand the audience as best you can. The difference is that as a designer, you're trying to understand the people who use the software. A marketing professional tries to understand those who buy it.

It's not easy to understand the real issues that underlie users' interactions with a system. Users don't always have the language or introspective skill to explain what they really need to accomplish their goals, and it takes a lot of work on your part to ferret out useful design concepts from what they can tell you—self-reported observations are usually biased in subtle ways.

Some of these techniques are very formal, and some aren't. Formal and quantitative methods are valuable because they're good science. When applied correctly, they help

you see the world as it actually is, not how you think it is. If you do user research haphazardly, without accounting for biases such as the self-selection of users, you might end up with data that doesn't reflect your actual target audience—and that can only hurt your design in the long run.

But even if you don't have time for formal methods, it's better to just meet a few users informally than to not do any discovery at all. Talking with users is good for the soul. If you're able to empathize with users and imagine those individuals actually using your design, you'll produce something much better.

The Patterns: Cognition and Behavior Related to Interface Design

The patterns that follow are some of the most common ways people think and behave as it relates to software interfaces. Even though individuals are unique, people in general behave predictably. Designers have been doing site visits and user observations for years; cognitive scientists and other researchers have spent countless hours watching how people do things and how they think about what they do.

So, when you observe people using your software, or doing whatever activity you want to support with new software, you can expect them to do certain things. The behavioral patterns that follow are often seen in user observations. The odds are good that you'll see them too, especially if you look for them.

Note

For all of you pattern enthusiasts: these patterns aren't like the others in this book. They describe human behaviors—not interface design elements—and they're not prescriptive, like the patterns in other chapters. Instead of being structured like the other patterns, these are presented as small essays.

Again, an interface that supports these patterns well will help users achieve their goals far more effectively than interfaces that don't support them. And the patterns are not just about the interface, either. Sometimes the entire package—interface, underlying architecture, feature choice, documentation, everything—needs to be considered in light of these behaviors. But as the interface designer or interaction designer, you should think about these as much as anyone on your team. You might be in a better place than anyone to advocate for the users.

- *Safe Exploration*
- *Instant Gratification*
- *Satisficing*

- *Changes in Midstream*
- *Deferred Choices*
- *Incremental Construction*
- *Habituation*
- *Microbreaks*
- *Spatial Memory*
- *Prospective Memory*
- *Streamlined Repetition*
- *Keyboard Only*
- *Social Media, Social Proof, and Collaboration*

Safe Exploration

> "Let me explore without getting lost or getting into trouble."

When someone feels like they can explore an interface and not suffer dire consequences, they're likely to learn more—and feel more positive about it—than someone who doesn't explore. Good software allows people to try something unfamiliar, back out, and try something else, all without stress.

Those "dire consequences" don't even need to be very bad. Mere annoyance can be enough to deter someone from trying things out voluntarily. Clicking away pop-up windows, reentering data that was mistakenly erased, suddenly muting the volume on one's laptop when a website unexpectedly plays loud music—all can be discouraging. When you design almost any kind of software interface, make many avenues of exploration available for users to experiment with, without costing the user anything.

This pattern encompasses several of the most effective usability guidelines, based on research, as identified by usability expert Jakob Nielsen:[2]

- Visibility of system status
- Match between the system and the real world
- User control and freedom

2 Nielsen, Jakob. "10 Usability Heuristics for User Interface Design," *Nielsen Norman Group*, 24 Apr. 1994. *www.nngroup.com/articles/ten-usability-heuristics*.

Here are some examples of what "Safe Exploration" is like:

- A photographer tries out a few image filters in an image-processing application. They then decide that they don't like the results, and click Undo a few times to get back to where they were. Then they try another filter, and another, each time being able to back out of what they did. (The pattern named *Multilevel Undo*, in Chapter 8, describes how this works.)
- A new visitor to a company's home page clicks various links just to see what's there, trusting that the Back button will always get them back to the main page. No extra windows or pop ups open, and the Back button keeps working predictably. You can imagine that if a web app does something different in response to the Back button—or if an application offers a button that seems like a Back button, but doesn't behave quite like it—confusion might ensue. The user can become disoriented while navigating, and might abandon the app altogether.

Instant Gratification

> "I want to accomplish something now, not later."

People like to see immediate results from the actions they take—it's human nature. If someone starts using an application and gets a "success experience" within the first few seconds, that's gratifying! They'll be more likely to keep using it, even if it becomes more difficult later. They will feel more confident in the application, and more confident in themselves, than if it had taken a while to figure things out.

The need to support instant gratification has many design ramifications. For instance, if you can predict the first thing a new user is likely to do, you should design the UI to make that first thing stunningly easy. If the user's goal is to create something, for instance, create a new canvas, put a call to action on it, and place a palette next to it. If the user's goal is to accomplish some task, point the way toward a typical starting point.

This also means that you shouldn't hide introductory functionality behind anything that needs to be read or waited for, such as registrations, long sets of instructions, slow-to-load screens, advertisements, and so on. These are discouraging because they block users from finishing that first task quickly.

To summarize, anticipate their need, provide an obvious entry point, provide value to the customer first before asking for something valuable (email address, a sale) in return.

Satisficing

> "This is good enough. I don't want to spend more time learning to do it better."

When people look at a new interface, they don't read every piece of it methodically and then decide, "Hmmm, I think this button has the best chance of getting me what I want." Instead, a user will rapidly scan the interface, pick whatever they see first that might get them what they want, and try it—even if it might be wrong.

The term *satisficing* is a combination of satisfying and sufficing. It was coined in 1957 by the social scientist Herbert Simon, who used it to describe the behavior of people in all kinds of economic and social situations. People are willing to accept "good enough" instead of "best" if learning all the alternatives might cost time or effort.

Satisficing is actually a very rational behavior after you appreciate the mental work necessary to "parse" a complicated interface. As Steve Krug points out in his book *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability* (New Riders, 2014), people don't like to think any more than they need to—it's work! But if the interface presents an obvious option or two that the user sees immediately, they'll try it. Chances are good that it will be the right choice, and if not, there's little cost in backing out and trying something else (assuming that the interface supports Safe Exploration).

This means several things for designers:

- Use "calls to action" in the interface. Give directions on what to do first: type here, drag an image here, tap here to begin, and so forth.
- Make labels short, plainly worded, and quick to read. (This includes menu items, buttons, links, and anything else identified by text.) They'll be scanned and guessed about; write them so that a user's first guess about meaning is correct. If he guesses wrong several times, he'll be frustrated, and you'll both be off to a bad start.
- Use the layout of the interface to communicate meaning. Chapter 4 explains how to do so in detail. Users "parse" color and form on sight, and they follow these cues more efficiently than labels that must be read.
- Make it easy to move around the interface, especially for going back to where a wrong choice might have been made hastily. Provide "escape hatches" (see Chapter 3). On typical websites, using the Back button is easy, so designing easy forward/backward navigation is especially important for web apps, installed applications, and mobile devices.
- Keep in mind that a complicated interface imposes a large cognitive cost on new users. Visual complexity will often tempt nonexperts to satisfice: they look for the first thing that might work.

Satisficing is why many users end up with odd habits after they've been using a system for a while. Long ago, a user might have learned Path A to do something, and even though a later version of the system offers Path B as a better alternative (or maybe it was there all along), they see no benefit in learning it—that takes effort, after all—and keep using the less-efficient Path A. It's not necessarily an irrational choice. Breaking old habits and learning something new takes energy, and a small improvement might not be worth the cost to the user.

Changes in Midstream

> "I changed my mind about what I was doing."

Occasionally, people change what they're doing while in the middle of doing it. Someone might walk into a room with the intent of finding a key that they had left there, but while there, they find a newspaper and start reading it. Or they might visit Amazon.com to read product reviews, but end up buying a book instead. Maybe they're just sidetracked; maybe the change is deliberate. Either way, the user's goal changes while they're using the interface you designed.

This means designers should provide opportunities for people to do that. Make choices available. Don't lock users into a choice-poor environment with no connections to other pages or functionality unless there's a good reason to do so. Those reasons do exist. See the patterns called *Wizard* (Chapter 2) and *Modal Panel* (Chapter 3) for examples.

You can also make it easy for someone to start a process, stop in the middle, and come back to it later to pick up where they left off—a property often called reentrance. For instance, a lawyer might start entering information into a form on an iPad. Then, when a client comes into the room, the lawyer turns off the device, with the intent of coming back to finish the form later. The entered information shouldn't be lost.

To support reentrance, you can make dialogs and web forms remember values typed previously, and they don't usually need to be modal; if they're not modal, they can be dragged aside on the screen for later use. Builder-style applications—text editors, code development environments, and paint programs—can let a user work on multiple projects at one time, thus letting them put any number of projects aside while they work on another one. For more information, see the *Many Workspaces* pattern in Chapter 2.

Deferred Choices

> "I don't want to answer that now; just let me finish!"

This follows from people's desire for instant gratification. If you ask a task-focused user unnecessary questions in the process, they might prefer to skip the questions and come back to them later.

For example, some web-based bulletin boards have long and complicated procedures for registering users. Screen names, email addresses, privacy preferences, avatars, self-descriptions…the list goes on and on. "But I just wanted to post one little thing," says the user. Why not allow them to skip most of the questions, answer the bare minimum, and come back later (if ever) to fill in the rest? Otherwise, they might be there for half an hour answering essay questions and finding the perfect avatar image.

Another example is creating a new project in a video editor. There are some things that you do need to decide up front, such as the name of the project, but other choices—where on the server are you going to put this when you're done? I don't know yet!—can easily be deferred.

Sometimes, it's just a matter of not wanting to answer the questions. At other times, the user might not have enough information to answer yet. What if a music-writing software package asked you up front for the title, key, and tempo of a new song, before you've even started writing it? (See Apple's GarageBand for this bit of "good" design.)

The implications for interface design are simple to understand, though not always easy to implement:

- Don't accost the user with too many upfront choices in the first place.
- On the forms that they do need to use, clearly indicate required versus optional fields, and don't make too many of them required. Let them move on without answering the optional ones.
- Sometimes you can separate the few important questions or options from others that are less important. Present the short list; hide the long list.
- Use *Good Defaults* (Chapter 10) wherever possible, to give users some reasonable default answers to start with. But keep in mind that prefilled answers still require the user to look at them, just in case they need to be changed. They have a small cost, too.
- Make it possible for users to return to the deferred fields later, and make them accessible in obvious places. Some dialog boxes show the user a short statement, such as "You can always change this later by clicking the Edit Project button." Some websites store a user's half-finished form entries or other persistent data, such as shopping carts with unpurchased items.

- If registration is required at a website that provides useful services, users might be far more likely to register if they're first allowed to experience the website—drawn in and engaged—and then asked later about who they are. Some sites let you complete an entire purchase without registering and then ask you at the end if you want to create a no-hassle login with the personal information provided in the purchase step.

Incremental Construction

> "Let me change this. That doesn't look right; let me change it again. That's better."

When people create things, they don't usually do it all in a precise order. Even an expert doesn't start at the beginning, work through the creation process methodically, and come out with something perfect and finished at the end.

Quite the opposite. Instead, they start with some small piece of it, work on it, step back and look at it, test it (if it's code or some other "runnable" thing), fix what's wrong, and start to build other parts of it. Or maybe they start over, if they really don't like it. The creative process goes in fits and starts. It moves backward as much as forward sometimes, and it's often incremental, done in a series of small changes instead of a few big ones. Sometimes it's top-down; sometimes it's bottom-up.

Builder-style interfaces need to support that style of work. Make it easy for users to build small pieces. Keep the interface responsive to quick changes and saves. Feedback is critical: constantly show the user what the entire thing looks and behaves like while the user works. If the user builds code, simulations, or other executable things, make the "compile" part of the cycle as short as possible, so the operational feedback feels immediate—leave little or no delay between the user making changes and seeing the results.

When creative activities are well supported by good tools, they can induce a state of flow in the user. This is a state of full absorption in the activity, during which time distorts, other distractions fall away, and the person can remain engaged for hours—the enjoyment of the activity is its own reward. Artists, athletes, and programmers all know this state.

But bad tools will keep users distracted, guaranteed. If the user must wait even half a minute to see the results of the incremental change they just made, their concentration is broken; flow is disrupted.

If you want to read more about flow, there are multiple books by researcher Mihaly Csikszentmihalyi. One title is *Flow: The Psychology of Optimal Experience*. (Harper Row, 2009).

Habituation

> "That gesture works everywhere else; why doesn't it work here, too?"

When you use an interface repeatedly, some frequent physical actions become reflexive: pressing Ctrl-S to save a document, clicking the Back button to leave a web page, pressing Return to close a modal dialog box, using gestures to show and hide windows—even pressing a car's brake pedal. The user no longer needs to think consciously about these actions. They've become habitual.

This tendency helps people become expert users of a tool (and helps create a sense of flow, too). Habituation also measurably improves efficiency, as you can imagine. But it can also lay traps for the user. If a gesture becomes a habit and the user tries to use it in a situation when it doesn't work—or, worse, does something destructive—the user is caught short. They suddenly need to think about the tool again (What did I just do? How do I do what I intended?), and they might need to undo any damage done by the gesture.

Millions of people have learned the following keyboard shortcuts based on using Microsoft Word and other word processors:

Ctrl-X
: Cut the selection

Ctrl-V
: Paste the selection

Ctrl-S
: Save the document

These shortcuts are true universals now. Consistency across applications can be an advantage to use in your software design. Just as important, though, is consistency within an application. Some applications are evil because they establish an expectation that some gesture will do Action X, except in one special mode where it suddenly does Action Y. Don't do that. It's a sure bet that users will make mistakes, and the more experienced they are—that is, the more habituated they are—the more likely they are to make that mistake.

Consider this carefully if you're developing gesture-based interfaces for mobile devices. After someone learns how to use their device and gets used to it, they will depend on the standard gestures working consistently on all applications. Verify that gestures in your design all do the expected things.

This is also why confirmation dialog boxes often don't work to protect a user against accidental changes. When modal dialog boxes pop up, the user can easily get rid of them just by clicking OK or pressing Return (if the OK button is the default button). If the dialogs pop up all the time when the user makes intended changes, such as

deleting files, clicking OK becomes a habituated response. Then, when it actually matters, the dialog box doesn't have any effect, because it slips right under the user's consciousness.

Note

I've seen at least one application that sets up the confirmation dialog box's buttons randomly from one invocation to another. You actually must read the buttons to figure out what to click! This isn't necessarily the best way to do a confirmation dialog box—in fact, it's better to not have them at all under most circumstances—but at least this design sidesteps habituation creatively.

Microbreaks

> "I'm waiting for the train. Let me do something useful for two minutes."

People often find themselves with a few minutes of downtime. They might need a mental break while working; they might be in line at a store or sitting in a traffic jam. They might be bored or impatient. They want to do something constructive or entertaining to pass the time, knowing they won't have enough time to get deep into an online activity.

This pattern is especially applicable to mobile devices, because people can easily pull them out at times such as these. The enormous success of the social media technology sector was built in no small part by taking advantage of this. Social and casual gaming, Facebook, Instagram, Snap...all are enjoyed in microbreaks.

Here are some typical activities during microbreaks:

- Checking email
- Reading *Streams and Feeds* (in Chapter 2) such as Facebook or Twitter
- Visiting a news site to find out what's going on in the world
- Watching a short video
- Doing a quick web search
- Reading an online book
- Playing a short game

The key to supporting microbreaks is to make an activity easy and fast to reach—as easy as turning on the device and selecting an application (or website). Don't require complicated setup. Don't take forever to load. And if the user needs to sign in to a

service, try to retain the previous authentication so that they don't need to sign in every time.

For "Stream and Feed" services, load the freshest content as quickly as possible and show it in the first screen the user sees. Other activities such as games, videos, or online books should remember where the user left them last time and restore the app or site to its previous state, without asking (thus supporting reentrance).

If you're designing an email application or anything else for which the user needs to do "housekeeping" to maintain order, give them a way to triage items efficiently. This means showing enough data per item so that they can identify, for instance, a message's contents and sender. You can also give them a chance to "star" or otherwise annotate items of interest, delete items easily, and write short responses and updates.

Long load times deserve another mention. Taking too long to load content is a sure way to make users give up on your app—especially during microbreaks! Make sure the page is engineered so that readable, useful content loads first, and with very little delay.

Spatial Memory

> "I swear that button was here a minute ago. Where did it go?"

When people manipulate objects and documents, they often find them again later by remembering where they are, not what they're named.

Take the Windows, Mac, or Linux desktop. Many people use the desktop background as a place to put documents, frequently used applications, and other such things. It turns out that people tend to use spatial memory to find things on the desktop, and it's very effective. People devise their own groupings, for instance, or recall that "this document was at the upper right over by such-and-such." (Naturally, there are real-world equivalents, too. Many people's desks are "organized chaos," an apparent mess in which the office owner can find anything instantly. But heaven forbid that someone should clean it up for them.)

Many applications put their dialog buttons—OK, Cancel, and so on—in predictable places, partly because spatial memory for them is so strong. In complex applications, people might also find things by remembering where they are relative to other things: tools on toolbars, objects in hierarchies, and so forth. Therefore, you should use patterns such as *Responsive Enabling* (Chapter 4) carefully. Adding items to blank spaces in an interface doesn't cause problems, but rearranging existing controls can disrupt spatial memory and make things more difficult to find. It depends. Try it out on your users if you're not sure.

Many mobile applications and games consist of just a few screens. Often the start screen is designed to be where users spend all of their time. There might not be any apparent navigation. But users learn to swipe left, right, up, or down to get to the

other screens (such as messaging or settings). These other screens are there, just off to the side. Snap is a good example of a mobile app that is designed to use people's spatial memory.

Along with habituation, which is closely related, spatial memory is another reason why consistency across and within a platform's applications is good. People might expect to find similar functionality in similar places. For an example, see the *Sign-In Tools* pattern (Chapter 3).

Spatial memory explains why it's good to provide user-arranged areas for storing documents and objects, such as the aforementioned desktop. Such things aren't always practical, especially with large numbers of objects, but it works quite well with small numbers. When people arrange things themselves, they're likely to remember where they put them. (Just don't rearrange it for them unless they ask!) The *Movable Panels* pattern in Chapter 4 describes one particular way to do this.

Also, this is why changing menus dynamically can sometimes backfire. People get used to seeing certain items on the tops and bottoms of menus. Rearranging or compacting menu items "helpfully" can work against habituation and lead to user errors. So can changing navigation menus on web pages. Try to keep menu items in the same place and in the same order on all subpages in a site.

Incidentally, the tops and bottoms of lists and menus are special locations, cognitively speaking. People notice and remember them more than items in the middle of a list. The first and last items in a list are more likely to be noticed. So, if you want to draw attention to one or two items in a list of items, place them at the beginning or end. Items moved to the middle are less likely to be noticed or remembered.

Prospective Memory

> "I'm putting this here to remind myself to deal with it later."

Prospective memory is a well-known phenomenon in psychology that doesn't seem to have gained much traction yet in interface design. But I think it should.

We engage in prospective memory when we plan to do something in the future, and we arrange some way of reminding ourselves to do it. For example, if you need to bring a book to work the next day, the night before you might put it on a table beside the front door. If you need to respond to someone's email later (just not right now!), you might leave that email on your screen as a physical reminder. Or if you tend to miss meetings, you might arrange for Outlook or your mobile device to ring an alarm tone five minutes before each meeting.

Basically, this is something almost everyone does. It's a part of how we cope with our complicated, highly scheduled, multitasked lives: we use knowledge "in the world" to aid our own imperfect memories. We need to be able to do it well.

Some software does support prospective remembering. Outlook and most mobile platforms, as mentioned earlier, implement it directly and actively; they have calendars, and they sound alarms. Trello is another example; it is a Kanban board of cards. Memory aids that people use can include the following:

- Notes to oneself, like virtual "sticky notes"
- Windows left on-screen
- Annotations put directly into documents (such as "Finish me!")
- Browser bookmarks, for websites to be viewed later
- Documents stored on the desktop rather than in the usual places in the filesystem
- Email kept in an inbox (and maybe flagged) instead of filed away

People use all kinds of artifacts to support passive prospective remembering. But notice that almost none of the techniques in the preceding list were designed with that in mind! What they were designed for is flexibility—and a laissez-faire attitude toward how users organize their stuff. A good email client lets you create folders with any names you want, and it doesn't care what you do with messages in your inbox. Text editors don't care what you type, or what giant bold magenta text means to you; code editors don't care that you have a "Finish this" comment in a method header. Browsers don't care why you keep certain bookmarks around.

In many cases, that kind of hands-off flexibility is all you really need. Give people the tools to create their own reminder systems. Just don't try to design a system that's too smart for its own good. For instance, don't assume that just because a window's been idle for a while, that no one's using it and it should be closed. In general, don't "helpfully" clean up files or objects that the system might think are useless; someone might be leaving them around for a reason. Also, don't organize or sort things automatically unless the user asks the system to do so.

As a designer, is there anything positive you can do for prospective memory? If someone leaves a form half-finished and closes it temporarily, you could retain the data in it for the next time—it will help remind the user where they left off. (See the *Deferred Choices* pattern.) Similarly, many applications recall the last few objects or documents they edited. You could offer bookmark-like lists of "objects of interest"—both past and future—and make those lists easily available for reading and editing. You can implement *Many Workspaces* (Chapter 2), which lets users leave unfinished pages open while they work on something else.

Here's a bigger challenge: if the user starts tasks and leaves them without finishing them, think about how to leave some artifacts around, other than open windows, that identify the unfinished tasks. Another idea: how might a user gather reminders from different sources (email, documents, calendars, etc.) into one place?

Streamlined Repetition

> "I have to repeat this *how* many times?"

In many kinds of applications, users sometimes find themselves needing to perform the same operation over and over again. The easier it is for them, the better. If you can help reduce that operation down to one keystroke or click per repetition—or, better, just a few keystrokes or clicks for all repetitions—you will spare users much tedium.

Find and Replace dialog boxes, often found in text editors (Word, email composers, etc.), are one good adaptation to this behavior. In these dialog boxes, the user types the old phrase and the new phrase. Then, it takes only one Replace button click per occurrence in the entire document. And that's only if the user wants to see or veto each replacement. If they're confident that they really should replace all occurrences, they can click the Replace All button; one gesture does the whole job.

Here's a more general example. Photoshop lets you record "actions" when you want to perform some arbitrary sequence of actions with a single click. If you want to resize, crop, brighten, and save 20 images, you can record those four steps as they're done to the first image, and then click that action's Play button for each of the remaining 19. For more information, see the *Macros* pattern in Chapter 8.

Scripting environments are even more general. Unix and its variants allow you to script anything you can type into a shell. You can recall and execute single commands, even long ones, with a Ctrl-P and Return. You can take any set of commands you issue to the command line, put them in a `for` loop, and execute them by pressing the Return key once.

Or you can put them in a shell script (or in a for-loop in a shell script) and execute them as a single command. Scripting is very powerful, and when complex, it becomes full-fledged programming.

Other variants include copy-and-paste capability (preventing the need to retype the same thing in a million places), user-defined "shortcuts" to applications on operating-system desktops (preventing the need to find those applications' directories in the filesystem), browser bookmarks (so users don't need to type URLs), and even keyboard shortcuts.

Direct observation of users can help you to determine just what kinds of repetitive tasks you need to support. Users won't always tell you outright. They might not even be aware that they're doing repetitive things that could be streamlined with the appropriate tools—they might have been doing it for so long that they don't even notice anymore. By watching them work, you might see what they don't see.

In any case, the idea is to offer users ways to streamline the repetitive tasks that could otherwise be time consuming, tedious, and error prone. For more information, see *Macros* in Chapter 8.

Keyboard Only

> "Please don't make me use the mouse."

Some people have real physical trouble using a mouse. Others prefer not to keep switching between the mouse and keyboard because that takes time and effort—they'd rather keep their hands on the keyboard at all times. Still others can't see the screen, and their assistive technologies often interact with the software using just the keyboard API.

For the sakes of these users, some applications are designed to be "driven" entirely via the keyboard. They're usually mouse-driven, too, but there is no operation that must be done with only the mouse—keyboard-only users aren't shut out of any functionality.

Several standard techniques exist for keyboard-only usage:

- You can define keyboard shortcuts, accelerators, and mnemonics for operations reachable via application menu bars, such as Ctrl-S for Save. See your platform style guide for the standard ones.
- Selection from lists, even multiple selection, is usually possible using arrow keys in combination with modifiers (such as the Shift key), though this depends on which component set you use.
- The Tab key typically moves the keyboard focus—the control that receives keyboard entries at the moment—from one control to the next, and Shift-Tab moves backward. This is sometimes called *tab traversal*. Many users expect it to work on form-style interfaces.
- Most standard controls, even radio buttons and combo boxes, let users change their values from the keyboard by using arrow keys, the Return key, or the space bar.
- Dialog boxes and web pages often have a "default button"—a button representing an action that says "I'm done with this task now." On web pages, it's often Submit or Done; on dialog boxes, OK or Cancel. When users press the Return key on this page or dialog box, that's the operation that occurs. Then, it moves the user to the next page or returns him to the previous window.

There are more techniques. Forms, control panels, and standard web pages are fairly easy to drive from the keyboard. Graphic editors, and anything else that's mostly spatial, are much more difficult, though not impossible.

Keyboard-only usage is particularly important for data-entry applications. In these, speed of data entry is critical, and users can't afford to move their hands off the keyboard to the mouse every time they want to move from one field to another or even one page to another. (In fact, many of these forms don't even require users to press the Tab key to traverse between controls; it's done automatically.)

Social Media, Social Proof, and Collaboration

> "What did everyone else say about this?"

People are social. As strong as our opinions might sometimes be, we tend to be influenced by what our peers say and do. And we are powerfully attuned to seeking approval from others and belonging to a group. We maintain social media identities. We contribute to groups and people we care about.

The advice of peers, whether direct or indirect, influences people's choices when they decide any number of things. Finding things online, performing transactions (Should I buy this product?), playing games (What have other players done here?), and even building things—people can be more effective when aided by others. If not, they might at least be happier with the outcome.

We are much more likely to watch, read, buy, join, share, comment, or take any other action if we see that someone we know has recommended it or done it or is present. This is called *social proof*.

All of these real-world dynamics underpin the massive scale and success of social computing in its many forms. It is fair to say that a social aspect or layer is part of almost all software today. Enabling social dynamics in your software can bring increased engagement, virality, community, and growth.

Here are some examples of social functionality:

User-generated reviews and comments
: These allow individuals to get a sense of the wisdom of the crowd. Reviews can be rated, and participants can gain fame or other rewards for being rated as a good reviewer.

Everything is a social object
: Text posts, images, video, check-ins, almost anything that users create in social media becomes an object that people can virtually gather around. Anything can be shared, rated, have a discussion thread attached to it, and similar activities.

Collaboration
: Business productivity and communication software has been transformed by software that allows people separated by space and time to come together in discussion threads, document reviews, video conferencing, tracking status, live and asynchronous communications, and many other activities.

Social proof motivates people to take action. Social group identity, participation, and recognition are powerfully rewarding to people.

Designing these capabilities into your interface creates the opportunity for social dynamics to increase your audiences' engagement, reward, and growth.

Of the patterns in this book, *Help Systems* (Chapter 2) most directly addresses this idea; an online support community is a valuable part of a complete help system for some applications.

For a deep dive into designing for social media, see *Designing Social Interfaces: Principles, Patterns, and Practices for Improving the User Experience* by Christian Crumlish and Erin Malone (O'Reilly, 2015).

Conclusion

This chapter gave you a tour of the vital context for any successful interaction design: understanding who will be using your software. This is the basis for designs that are fit for purpose and able to be understood easily. To achieve this, ground your design process in the four-part foundation described in this chapter. First, understand the context. This means getting clear on what sorts of people you are designing for, the subject or work domain you are designing for them, and your users' existing skill level. Second, understanding their goals is important. This is the framework of workflows, tasks, and outcomes you will design for. Third, user research is a valuable activity and skill to help you understand users and their goals. We outlined a number of research activities that you can choose from. Finally, we looked at a number of patterns in human behavior, perception, and thinking that are relevant for designing interfaces. These four elements form the foundation of your design process. In Chapter 2, we look at creating a strong organization foundation for your software or app itself.

Chapter 2

Organizing the Content: Information Architecture and Application Structure

Picking up on the concept of building a foundation from Chapter 1, let's now take a look at what an information foundation for your software or app might look like. What this means is designing the information architecture—how the data, content, and functionality are organized in a way that makes sense for the people you're designing for. Specifically, in this chapter we cover the following:

- A definition of information architecture
- How to design the information and task space for comprehension and navigation
- Different methods of organizing content and data for use
- How to organize tools and features for efficient work
- Developing a system of repeatable frameworks or screen types
- Patterns for displaying, accessing, and navigating content and functionality

At this point, you might feel confident that you understand what your users want out of your application or site. You might have written down some typical scenarios that describe how people might use high-level elements of the application to accomplish their goals. You have a clear idea of what value this application adds to people's lives.

It's tempting at this stage to want to go directly to designing the screens and components of your interface, working with colors, typography, language, and layouts. If you're the kind of person who likes to think visually and needs to play with sketches while working out the broad strokes of the design, go for it.

However, to take full advantage of your customer insights and give your design the best chance for success, the next step is to use your understandings to develop your

information architecture. Don't become locked in to specific interface design decisions just yet. Instead, take a step back and think about designing the overall structure and framework of your software so that it makes sense from your user's point of view. Think through the information, the workflows, the language of the site or application, and then organize them so they are easy to learn and easy to use.

This is *information architecture*. Let's break this down by looking at the benefits and scope of information architecture design.

Purpose

The purpose of information architecture is to create the framework for your digital product, service, site, or application to be successful. In particular, we mean successfully understood, learned, and used with a minimum of mental strain or confusion. A critical part of these interactive experiences is that the interface shouldn't get in the way.

The irony of information architecture is that customers only notice it when it's bad. For instance: the organization of the experience makes no sense. The interface is confusing and the screens are frustrating. Customers don't understand the terminology they see on screen. They can't find what they need when they need it. Basically, it gets in the way.

On the other hand, if we've done our jobs properly, our design is invisible. Users don't really notice great information architecture. All users know is that they have a natural, efficient, and pleasant digital experience.

What does that mean in practice?

Stepping back, we can say that the context is people who are trying to do something: find information, watch a video, buy something, sign up for something. In short, they have a task to do. But the people who make the digital product—that's you—can't be there in person. You need to design your app to mimic what a good customer service representative would do:

- Anticipate what they need
- Organize and talk about the information from the customer's point of view
- Offer information in a clear, simple way
- Use words that the customer understands
- Offer clear next steps
- Make it really obvious where you are and what's happening
- Confirm that a task was successfully completed

Definition

Information architecture (IA) is the art of organizing and labeling an information space for optimal understanding and use. Specifically, IA is using your understanding of your users to design the following:

- Structures or categories for organizing your content and functionality
- Different ways that people can navigate through the experience
- Intuitive workflows or multistep processes for getting tasks done
- Labels and language for communicating about this content
- Searching, browsing, and filtering tools to help them find what they're looking for
- A standardized system of screen types, templates, or layouts so that information is presented consistently and for maximum usability

IA encompasses many things: presenting, searching, browsing, labeling, categorizing, sorting, manipulating, and strategically hiding information. Especially if you're working with a new product, this is where you should begin. The goal is for all of this to make sense from the user's point of view so that they can use your site or app successfully.

Designing an Information Space for People

In the same way that construction architects draw up blueprints before a house is actually built, designers—information architects—create a plan for how their information space will be laid out in a usable manner and how people will move around in it and get work done there. In both cases, there is efficiency and value in thinking through how people will use what you are going to build, before you build it.

Approach

It can be helpful to think about an application in terms of its underlying data and tasks. Do this without thinking about the look and feel that they will ultimately take on. Think more abstractly:

- What information and tools do you need to show to the users?
- Based on their expectations and immediate situation, when do you show them?
- How are the information and the tools categorized and ordered?
- What do users need to do with them?

- How many ways can you present those things and tasks? You might need more than one.
- How can you make it usable from their point of view?

These lines of inquiry can help you think more creatively about the IA you're designing.

Separate Information from Presentation

Thinking about IA separately from the visual design is so important that it's worth looking at in more detail. The design challenge becomes more manageable when you tackle it in phases. In fact, it's useful to think of designing as a process of building up layers of design, one on top of the other.

In the same way that software engineers think of applications in three layers—databases; tools and queries; and reports, results, and responses—designers can think of their design has having three layers.

Figure 2-1 gives a schematic representation of this approach. IA is the lowest layer, the foundation. Much like with a physical building, the structure of this foundation will ultimately be invisible, but will shape everything that is built on top of it. In the digital world, we are concerned with creating an IA foundation that has appropriate concepts, labels, relationships, and categories. It lays out the permanent information structure that users can navigate, search through, and manipulate on the upper layers of the experience.

The middle layer is the *functionality and information delivery layer* of your site or app. It is the screens, pages, stories, lists, and cards your users browse, search, and read. It contains the tools they use to search, filter, monitor, analyze, communicate, and create.

The topmost layer is the *presentation layer*: the visual design and editorial system for presenting and rendering. It consists of colors, typography, layouts, graphics, and more. When done well, the presentation layer design creates focus, flow, and clarity.

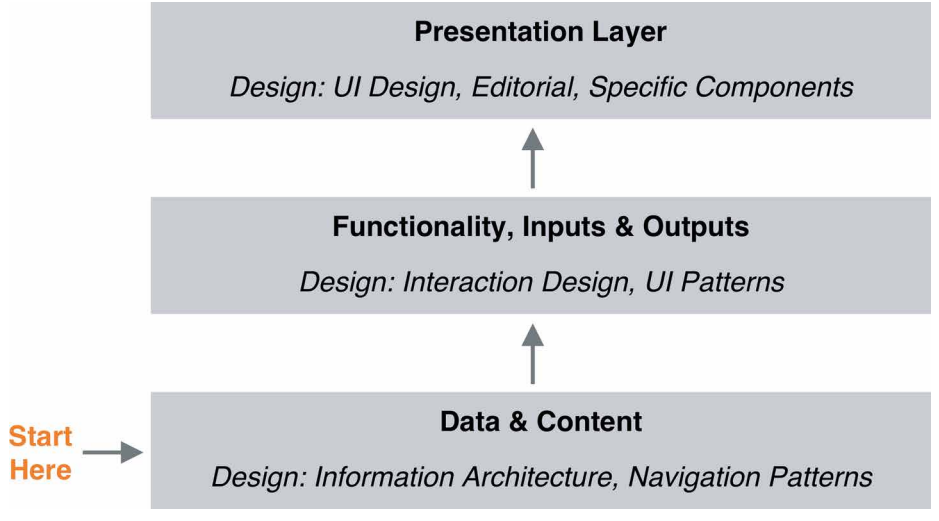


Figure 2-1. *Layered Design: Designing up from the content/data layer to the presentation layer (based on ideas by Jesse James Garrett, The Elements of User Experience: User-Centered Design for the Web and Beyond [New Riders, 2011])*

Mutually Exclusive, Collectively Exhaustive

Your content and tools need to be organized in a way that makes sense to your audience. As you go through the process of organizing the data and content into major categories or sections, consider this useful rule of thumb: *MECE*.

MECE stands for "Mutually Exclusive, Collectively Exhaustive." First, your information architecture should have categories of content that are clearly distinct from one another, with no confusing overlap. Second, "collectively exhaustive" means that taken all together, your organization schema is complete. It accounts for all the information your site or application is supposed to handle, and all of the situations and use cases you are designing for. There's a place to find everything or put anything. Later in the process, your information structure should be able to expand to accommodate new data without things becoming confusing.

These categories will form the foundation of your navigation system, which is discussed in more detail in Chapter 3.

To document and communicate this organization scheme, information architects develop such tools as *site maps* and *content outlines*.

Ways to Organize and Categorize Content

You have probably already used some common organizational methods to organize and categorize the information in your site or app. These are especially useful when planning how to display large amounts of structured data in tables. They also are important for planning for how users will search and browse, filter information, and sort and refine their results. We describe six of these methods in a moment. This list is based on the work of Richard Saul Wurman in his book on information architecture, *Information Anxiety 2* (Que, 2001), and Abbey Covert and Nicole Fenton's book *How to Make Sense of Any Mess* (Covert, 2014). Both make excellent further reading.

Wurman offers a handy way to remember the main methods of organization in the acronym "LATCH." This stands for "Location, Alphabet, Time, Category, Hierarchy." Let's look at these and others in more detail.

Alphabetical

This means organizing lists, names, and any labeled items according to the sequence of the alphabet. You can do this in descending order, from A to Z, or in ascending or reverse order, from Z to A. This can also include numbers if they are part of the name or label, with the numbers "0" and up preceding the alphabetical letters. This is a great default for any list or menu of items.

Number

Organizing by number can include a number of variations. First is according to integer, in which items or the numbers themselves are sorted in ascending or descending order based on the sequence of the number system. The second is by ordinal position: first, second, third, and so on. A third way is by value or total. Things such as financial amounts, discounts, size, rank, priority, and rates of change can be arranged from largest value to smallest, and vice versa. Tabular data uses this pattern heavily.

Time

Chronological order is another useful way to organize content. This is very common in social media feeds, where reverse chronological order is common—the most recent item is first in the list, with older items lower in the list. Information can be organized by date, time, or duration in ascending or descending order. They can also be arranged by frequency—low to high, and the reverse. They can also be arranged by sequence in time: which happened first, or should happen first, as in steps in a process. Tasks are also often split into a sequence of steps (can also be thought of as the aforementioned number).

Location

Location can mean organizing by a geographic or spatial location. There are many systems for specifying geographic location, such as latitude and longitude. Geographic categories are often nested or hierarchical, such as countries containing states, which contain cities (see Hierarchy, next). Location can also be distance from or to some reference point, and ordering based on that. In digital systems, it's critical for users to learn their location in the information space, both overall and on an individual screen.

Hierarchy

Your data might best be represented by containers or parent–child relationships, with the larger item containing a smaller one. Examples of this can be countries which contain states, years which contain months, or transactions which contain purchased items.

Category or Facet

In IA, content can be labeled and then grouped into categories or topics. A category can be thought of as a feature or quality that a set of items has in common. This is a very useful method of organizing because it is so flexible. Often, there is a spectrum or degree implied in the category that can be used to order the items in the set. A simple example is organizing by color.

Advanced organizing schemes use facets. A facet system assigns multiple qualities or categories, with a range of values in each, to each item. A good example of faceted classification in action is Amazon, where customers can use multiple values to narrow down their product search, such as price, availability, and customer rating.

Designing for Task and Workflow-Dominant Apps

IA also includes designing workflows and tasks. Documentation related to this often includes such things as user stories and flow diagrams.

Make Frequently Used Items Visible

The first rule of thumb in designing for tasks or workflow is frequency of use. Tasks, controls, commands, or topics that are repeated often or used frequently should be immediately available to the user, without having to search or browse. On the other hand, controls or information that are not needed very often can be hidden away or accessed only by navigating to it. User settings and help systems are good examples of features that are hidden in standard use but accessible when needed.

"Chunk Up" Jobs into a Sequence of Steps

Sequencing is the second organizing principle for IA for tasks and workflows. This means breaking up a big task or process into a series of steps, to make each individual step less demanding on the user. This is often the structure for a wizard or multistep process that leads the user through a complex task. Plan for communicating to the user where they are in the process.

Design for both novice and experienced users

When chunking a task, consider the level of learning, skill, or mastery that different users may have. Like in computer games, it's useful to consider how first-time users might need a simplified interface or special additional help to assist them. This can take the form of additional instructions, screen overlays, or wizards for complex processes. Many applications and websites devote resources to designing a *new user experience* or *onboarding*, to improve learning and customer retention. People who are unfamiliar with your app or site will particularly appreciate this approach.

On the other end of the skill spectrum, advanced or experienced users can be fast and efficient with complicated interfaces that are densely packed with information and selectors. Offering them "accelerators" such as shortcuts and the ability to customize their interface help them to be efficient. Designing for keyboard-only navigation and input is also valuable here.

Multiple Channels and Screen Sizes Are Today's Reality

It goes without saying that consumers and business users alike now expect to access information, sites, and applications via multiple channels such as desktop, mobile and messaging, and across a multitude of screen sizes and devices. Voice-activated services interfaces don't have screens at all. In designing your IA, consider what channels, modes, and devices your site or application will need to function across. This will drive how your information is organized, segmented, and sequenced.

Display Your Information as Cards

A common pattern in several of the examples that follow is a reliance on a *Cards* pattern. With the majority of all digital interactive experiences being via mobile devices, it makes sense to make the building block of your experience a card that fits on a smaller screen. This small container of information, photos, and other data can work individually or it can be displayed in a list or grid on larger screens. The key is to plan for how to scale down and scale up the experience while still delivering access and control to your information and features.

Designing a System of Screen Types

As mentioned earlier, IA also includes designing a system of screen types. Each screen type has a differentiated function. In this way, the user can learn how to use each screen reliably, even when the content in it changes based on topic, filters, and other selections.

A useful framework for approaching screen types is from Theresa Neil. She developed ideas for application structures in the context of Rich Internet Applications (RIAs). Neil defines three types of structures based on the user's primary goal: information, process, and creation.[1]

This list gives us a framework within which to fit the idioms and patterns that we talk about in this and other chapters.

Now, let's look at pages that serve single important functions. In an application, this might be a main screen or a major interactive tool; in a richly interactive website, it might be a single page, such as Gmail's main screen; in a more static website, it might be a group of pages devoted to one process or function.

Any such page will primarily do one of these things:

Overview
: Show a list or set of things

Focus
: Show one single thing, such as map, book, video, or game

Make
: Provide tools to create a thing

Do
: Facilitate a single task

Most apps and sites do some combination of these things, of course. But consider developing a screen system in which each has a particular organizing principle.

1 "Rich Internet Screen Design" in *UX Magazine*.

Overview: Show a List or Grid of Things or Options

This is what most of the world's home pages, start screens, and content sites seem to do. The digital world has converged on many common idioms for showing lists, most of which are familiar to you:

- Simple text lists
- Menus
- Grids of cards or images
- Search results in list or grid form
- Lists of email messages or other communications
- Tables of data
- Trees, panels, and accordions

These overview screens present information organization challenges. Here are some questions to consider in designing your overview screens:

- How big is the dataset or how long is the list?
- How much space is available to display it?
- Is it flat or hierarchical, and if it is a hierarchy, what kind?
- How is it ordered, and can the user change that ordering dynamically?
- How can the user search, filter, and sort?
- What information or operations are associated with each list item, and when and how should they be shown?

Because lists and grids are so common, a solid grasp of the different ways to present them can benefit any designer. A few patterns for designing an interface around a list are described in this chapter (others are in Chapter 7).

You can build either an entire app or site or a small piece of a larger artifact around one of these patterns. They set up a structure into which other display techniques—text lists, thumbnail lists, and so on—can fit. Other top-level organizations not listed here might include calendars, full-page menus, and search results:

- *Feature, Search, and Browse*. Countless websites that show products and written content follow this pattern. Searching and browsing provide two ways for users to find items of interest, whereas the front page features one item to attract interest.

- *Streams and Feeds*. Blogs, news sites, email readers, and social sites such as Twitter all use a news stream or social stream pattern to list their content, with the most recent updates first in the scrollable list.
- *Grids*. A well-defined interface type for presenting stories, actions, cards, and selectors. It is also used for handling photos and other pictorial documents. It can accommodate hierarchies and flat lists, tools to arrange and reorder documents, tools to operate directly on pictures, launch apps, drill down to details, and so on.

After you've chosen an overall design for the interface, you might look at other patterns and techniques for displaying lists. See Chapter 7 for a thorough discussion.

Focus: Show One Single Thing

The entire point of this screen type is to show or play a single piece of content or functionality, such as an article, map, or video. There might be small-scale tools clustered around the content—scrollers and sliders, sign-in box, global navigation, headers and footers, and so forth—but they are minor. Your design might take one of these shapes:

- A long, vertically scrolled page of flowed text (articles, books, and similar long-form content).
- A zoomable interface for very large, fine-grained artifacts such as maps, images, or information graphics. Map sites such as Google Maps provide some well-known examples.
- The "media player" idiom, including video and audio players.

As you design this interface, consider the following patterns and techniques to support the design:

- *Mobile Direct Access*, to take the user directly into the main function of your app, often using location and time data to generate valuable information without the user providing any input.
- *Alternative Views*, to show the content in more than one way.
- *Many Workspaces*, in case people want to see more than one place, state, or document at one time.
- *Deep-Linked State*, in Chapter 3. With this, a user can save a certain place or state within the content so that they can come back to it later or send someone else a URL.
- Some of the mobile patterns described in Chapter 6, if one of your design goals is to deliver the content on mobile devices.

Make: Provide Tools to Create a Thing

This screen type is for creating or updating digital objects. Most people are familiar with the idioms used by these tools: text editors, code editors, image editors, editors that create vector graphics, and spreadsheets.

At the level of application structure or IA, we can often find the following patterns:

- *Canvas Plus Palette* describes most of these applications. This highly recognizable, well-established pattern for visual editors sets user expectations very strongly.
- Almost all applications of this type provide *Many Workspaces*—usually windows containing different documents, which enable users to work on them in parallel.

Do: Facilitate a Single Task

Maybe your interface's job isn't to show a list of anything or create anything, but simply to get a job done. Signing in, registering, posting, printing, uploading, purchasing, changing a setting—all such tasks fall into this category.

Forms do a lot of work here. Chapter 10 talks about forms at length and lists many controls and patterns to support effective forms. Chapter 8 defines another useful set of patterns that concentrate more on "verbs" than "nouns."

Not much IA needs to be done if the user can do the necessary work in a small, contained area, such as a sign-in box. But when the task gets more complicated than that—if it's long, or branched, or has too many possibilities—part of your job is to work out how the task is structured.

- Much of the time, you'll want to break the task down into smaller steps or groups of steps. For these, a *Wizard* might work well for users who need to be walked through the task.
- A *Settings Editor* is a very common type of interface that gives users a way to change the settings or preferences of something—an application, a document, a product, and so on. This isn't a step-by-step task at all. Here, your job is to give users open access to a wide variety of choices and switches and let them change only what they need, when they need it, knowing that they will skip around.

These four screen types—overview, focus, make, and do—will likely show up in some form in your own system of screen types for your software.

The Patterns

The previous section reviewed information architecture in general terms. Now let's examine specific design patterns related to expressing information architecture in the experience of your software.

- *Feature, Search, and Browse*
- *Mobile Direct Access*
- *Streams and Feeds*
- *Thumbnail Grid*
- *Dashboard*
- *Canvas Plus Palette*
- *Wizard*
- *Settings Editor*
- *Alternative Views*
- *Many Workspaces*
- *Help Systems*
- *Tags*

Feature, Search, and Browse

What

A three-element combination on the main page of the site or app: a featured item, article, or product; a search box (expanded by default, or collapsed); and a list of items or categories that can be browsed.

Use when

Your site offers users long lists of items—articles, products, videos, and so on—that can be browsed and searched. You want to engage incoming users immediately by giving them something interesting to read or watch.

Alternately, your site focuses on enabling searching or transacting. In this case, search is the dominant element on the screen. Featured content and browsing have secondary importance.

Why

These three elements are found together on many sites. Searching and browsing go hand in hand to find desired items. Some people will know what they're looking for and use the search box, whereas others will do more open-ended browsing through the lists and categories you show them.

Featured items are how you engage the user. They're far more interesting than just category lists and search boxes, especially when you use appealing images and headlines. A user who lands on your page now has something to read or experiment with, without doing any additional work at all—and they may find it more interesting than whatever they originally came for.

How

Place a Search box in a prominent location, such as an upper corner, or in a banner across the middle top of the site. Demarcate it well from the rest of the site—use whitespace to set it off, and use a different surrounding background color if necessary.

Alternatively, display Search in a collapsed or compacted state. It still needs to be easy to see and access, but it can be an icon or the label "Search." Selecting this opens the full search field. This pattern saves space on smaller screens.

Set aside *Center Stage* (Chapter 4) for the featured article, product, or video. Very near it, and still at the top of the page, place an area for browsing the rest of the site's content. Most sites show a list of stories, cards, topics, or product categories. These might be links to pages devoted to those categories.

If the category labels open in place to show subcategories, the list behaves like a tree. Some sites, such as Amazon, turn the category labels into menus: when the pointer rolls over the label, a menu of subcategories appears.

Choose the features well. Features are a good way to sell items, advertise specials, and call attention to breaking news. However, they are the front door and also define what your site is about. What will they want to know about? What will capture their attention and hold them at your site?

As the user browses through categories and subcategories, help them "stay found" with the *Breadcrumbs* pattern (Chapter 3).

Content-centric websites. The following three examples demonstrate the classic pattern of *Feature, Search, Browse*. WebMD (Figure 2-2), Yahoo! (Figure 2-3), and Sheknows (Figure 2-4) are news- and content-centric digital publishers. WebMD and Yahoo! have Search at the top as a single large feature. Sheknows offers a variation: two features above a prominent search input.

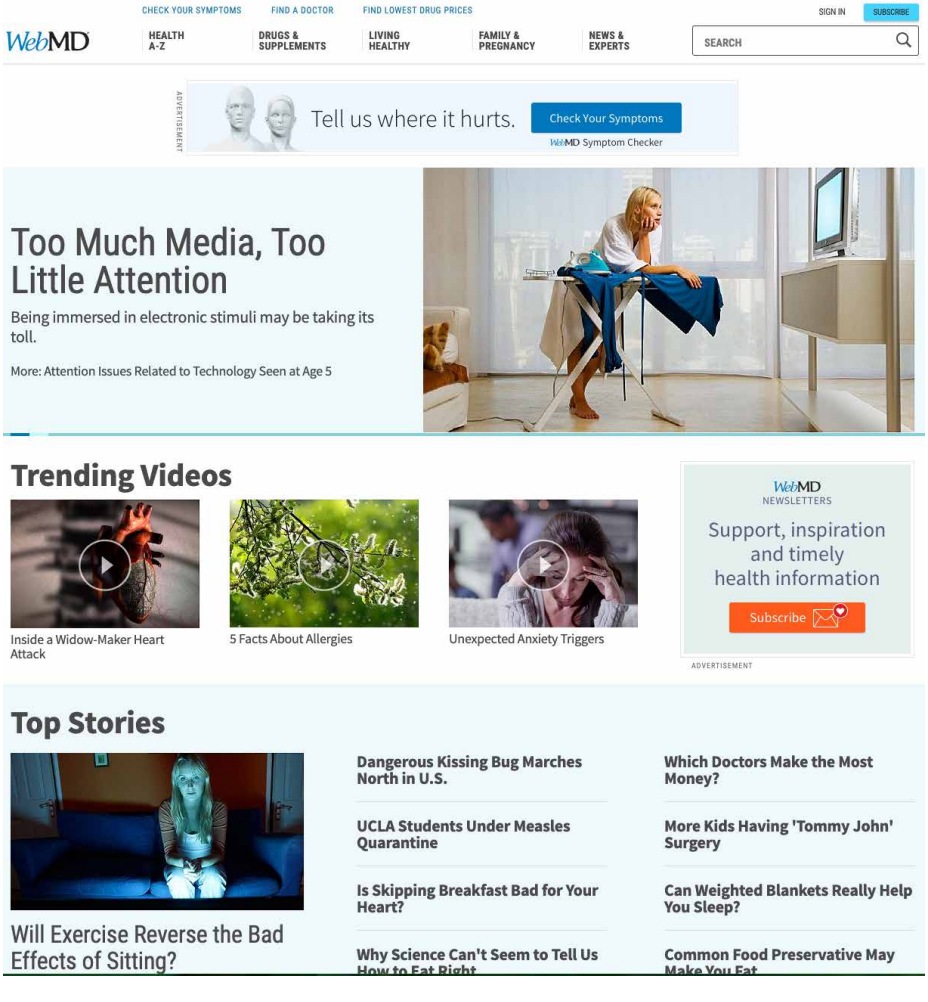


Figure 2-2. *WebMD*

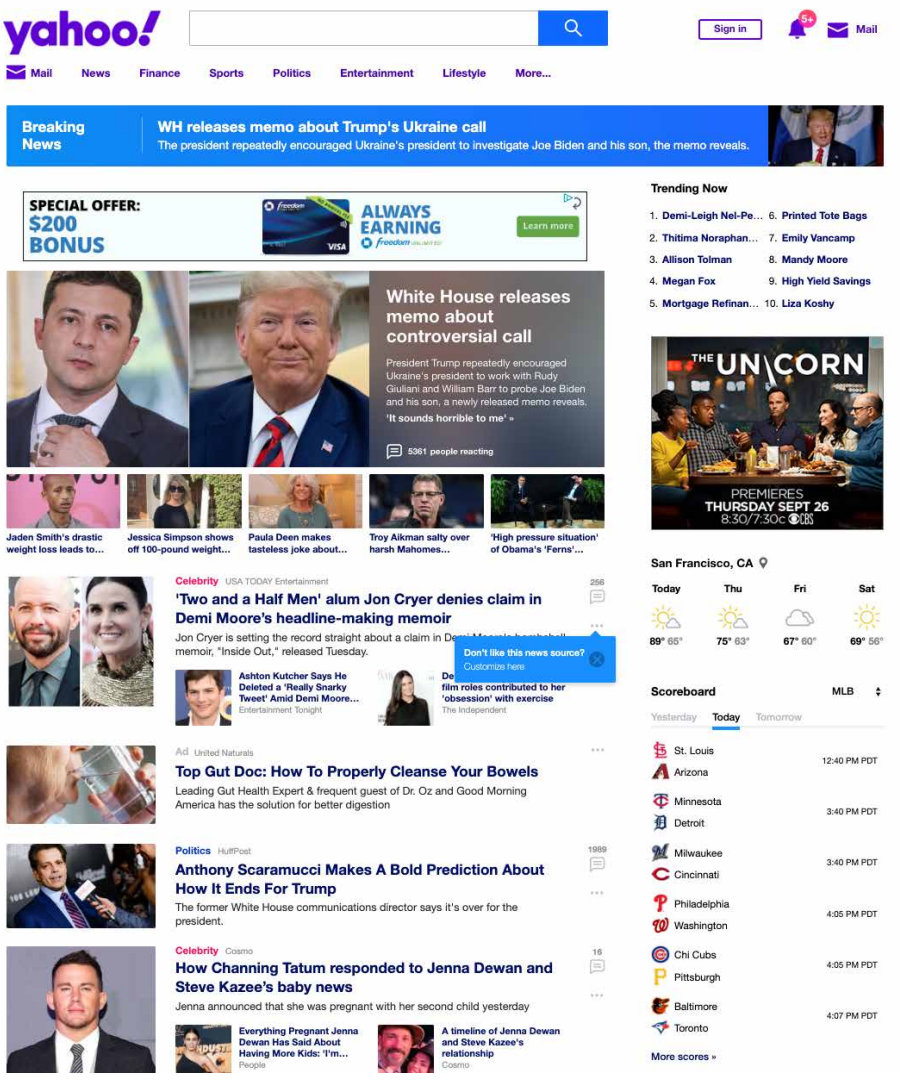


Figure 2-3. *Yahoo!*

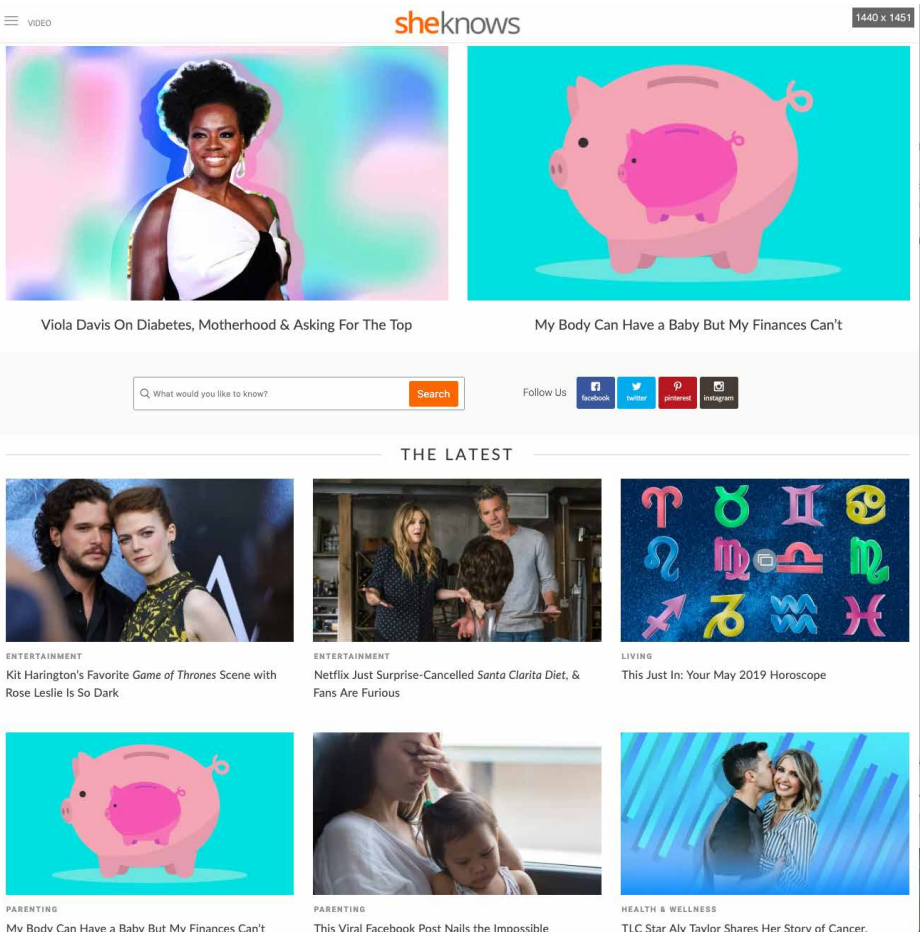


Figure 2-4. *Sheknows*

Commerce- centric websites. Major retailers like Target (Figure 2-5) and Ace Hardware (Figure 2-6) follow the same pattern. Search is at the top, with large features (sales promotions) below. Both sites support browse with grids of cards.

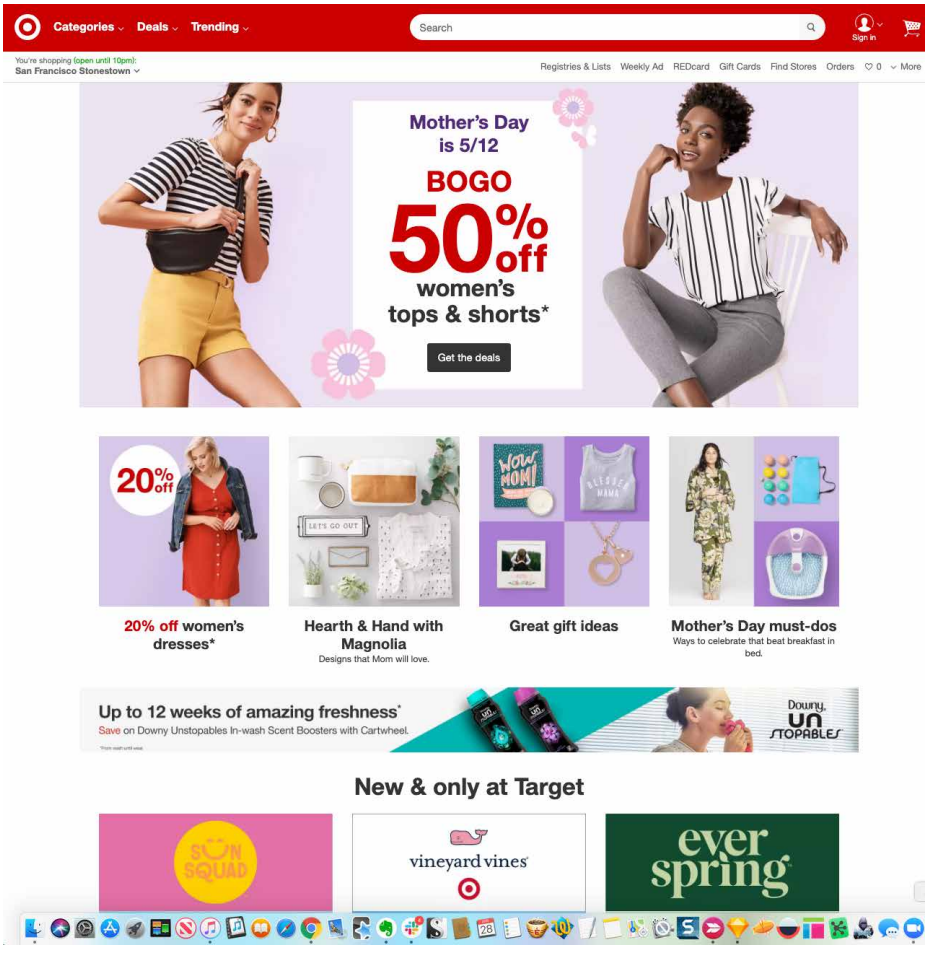


Figure 2-5. *Target*

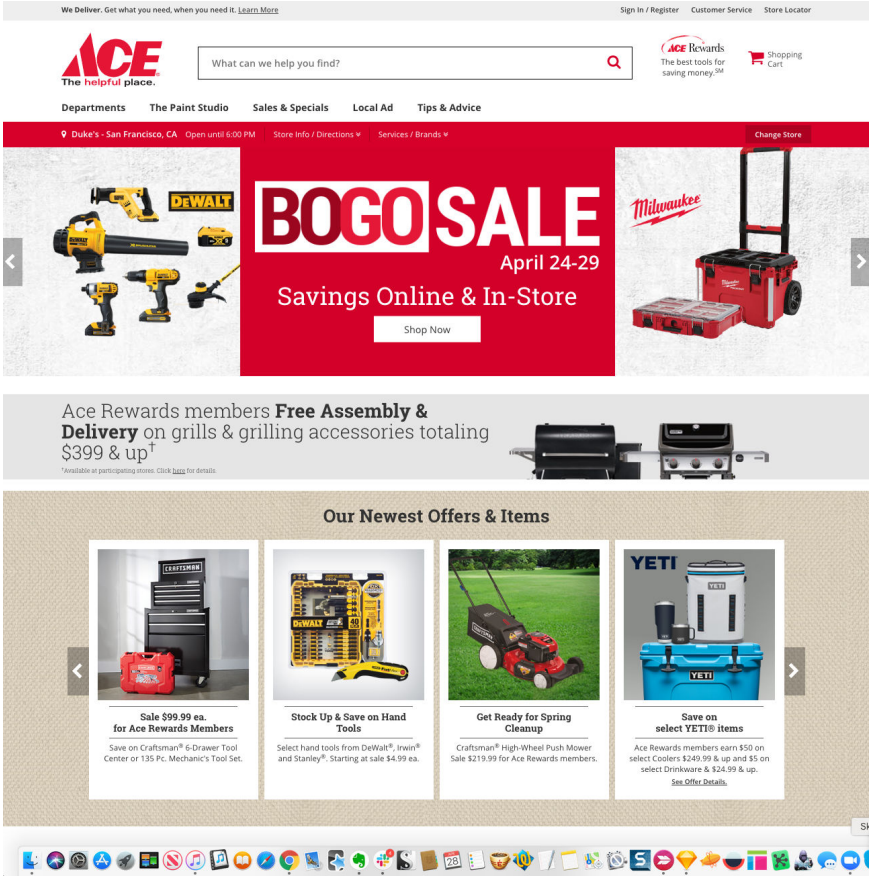


Figure 2-6. *Ace Hardware*

Task-centric websites. These two examples show how *Feature, Search, and Browse* changes when deployed in a context for which getting the user to carry out a task (search) is the priority. For British Airways (Figure 2-7), a large travel search module dominates the entire screen at the top of the page. Below the fold, there is minimal content: A featured article and only three cards for browsing.

Epicurious (Figure 2-8) gives primacy to search, as well. It occupies a screen-width panel at the top of the page. However, the content features and browse cards begin immediately below. Their large size and appetizing photos and titles give them almost equal weight to search. Based on your needs, you can decide which prioritized or balanced approach is best suited for your situation and design your interface accordingly.

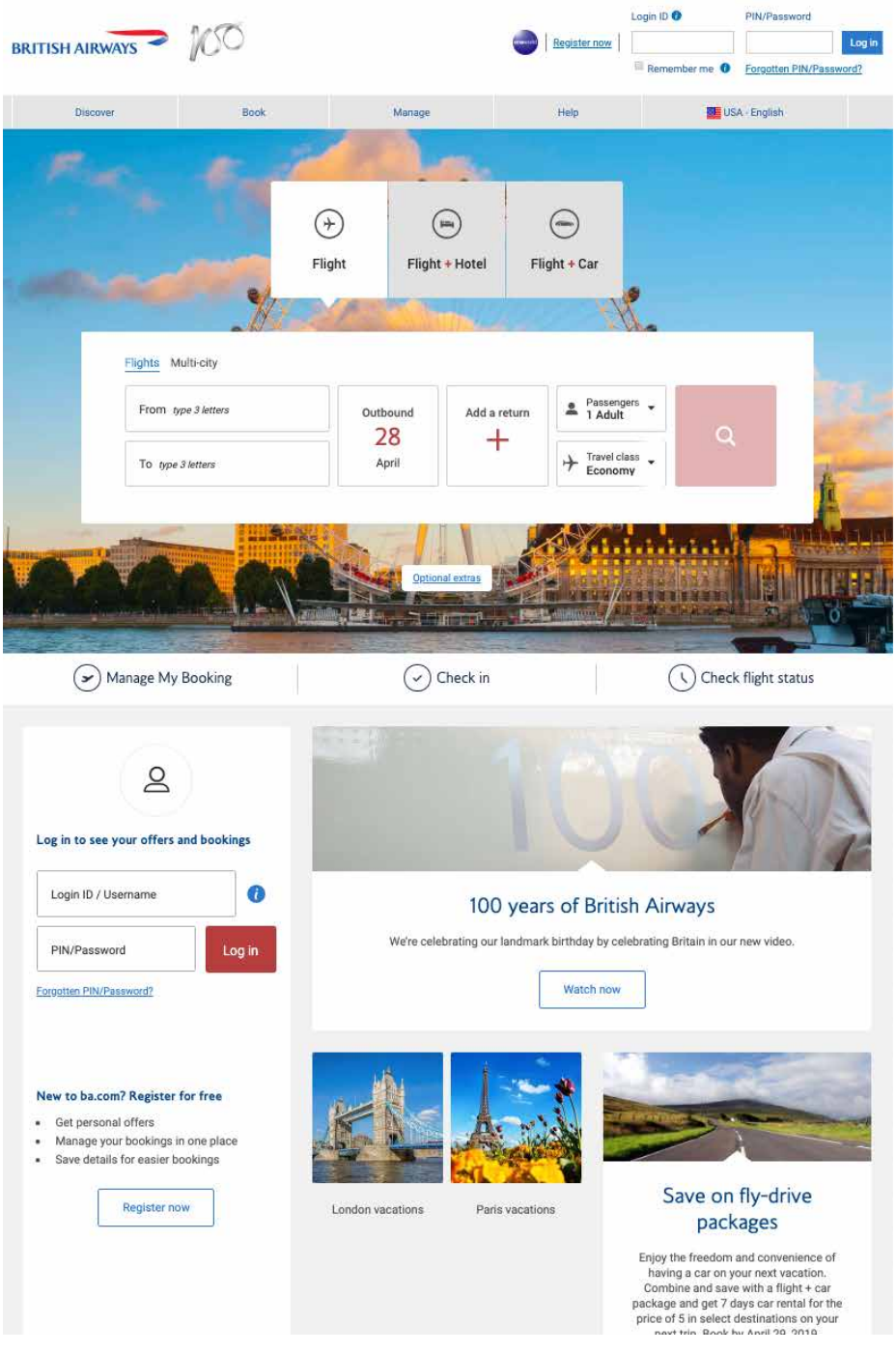


Figure 2-7. *British Airways*

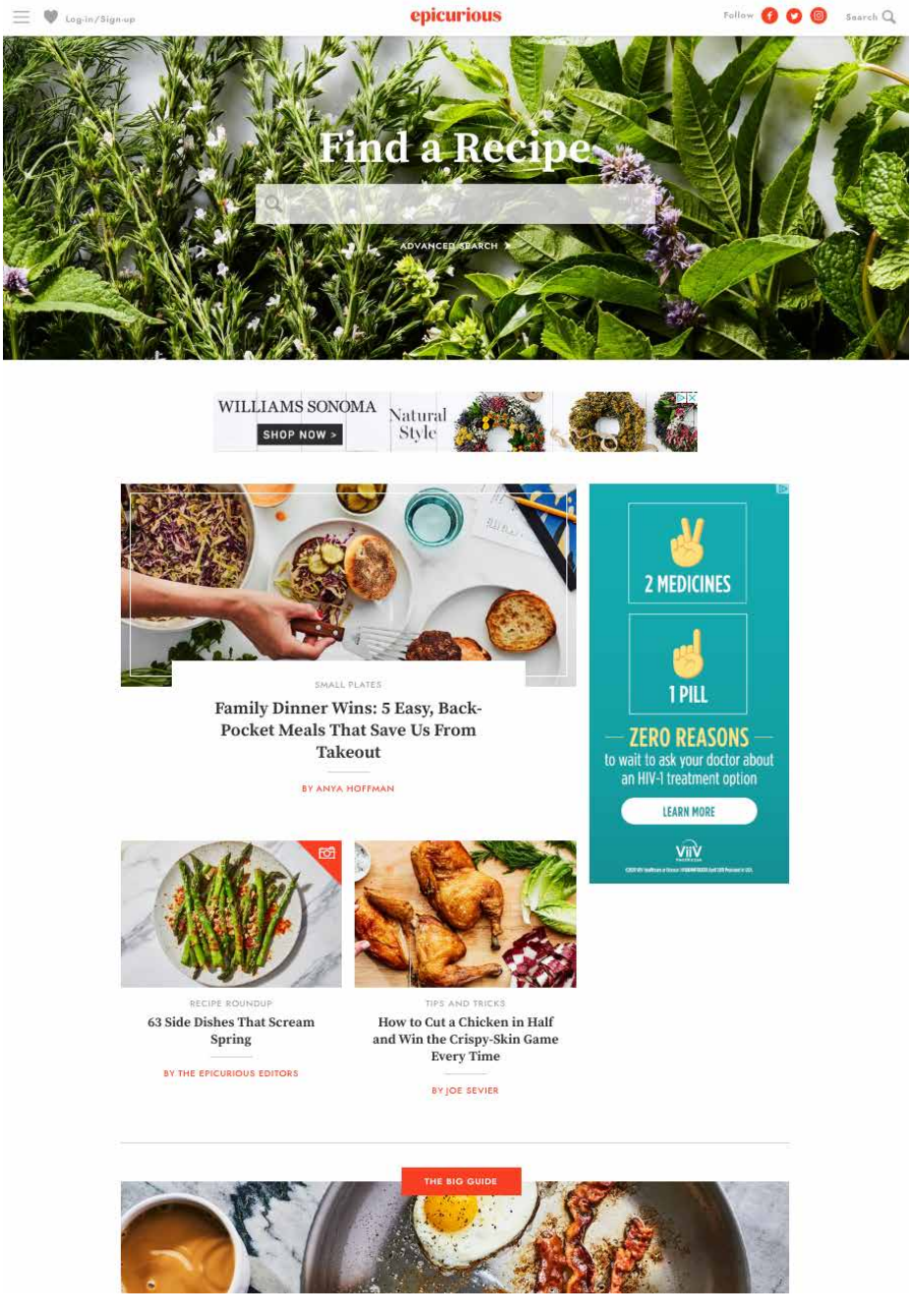


Figure 2-8. *Epicurious*

Search with facets and filters. These examples show how two different websites use multiple categories, or *facets*, to help the user create a targeted search for large datasets. Each facet can have a range of values. Combinations of facets for search or filtering allows for sophisticated searching of large datasets.

Crunchbase (Figure 2-9) is unusual in that it promotes faceted filters as part of its search submittal, implying that the searcher can achieve better, more appropriate search results by using the facets. Featured content is below.

Epicurious (Figure 2-10) and Airbnb (Figure 2-11) both use a more traditional deployment of faceted filters on their search results screens (note the mobile-friendly grid of cards format for search results in both cases). The facets are most relevant for narrowing the search results based on their domain.

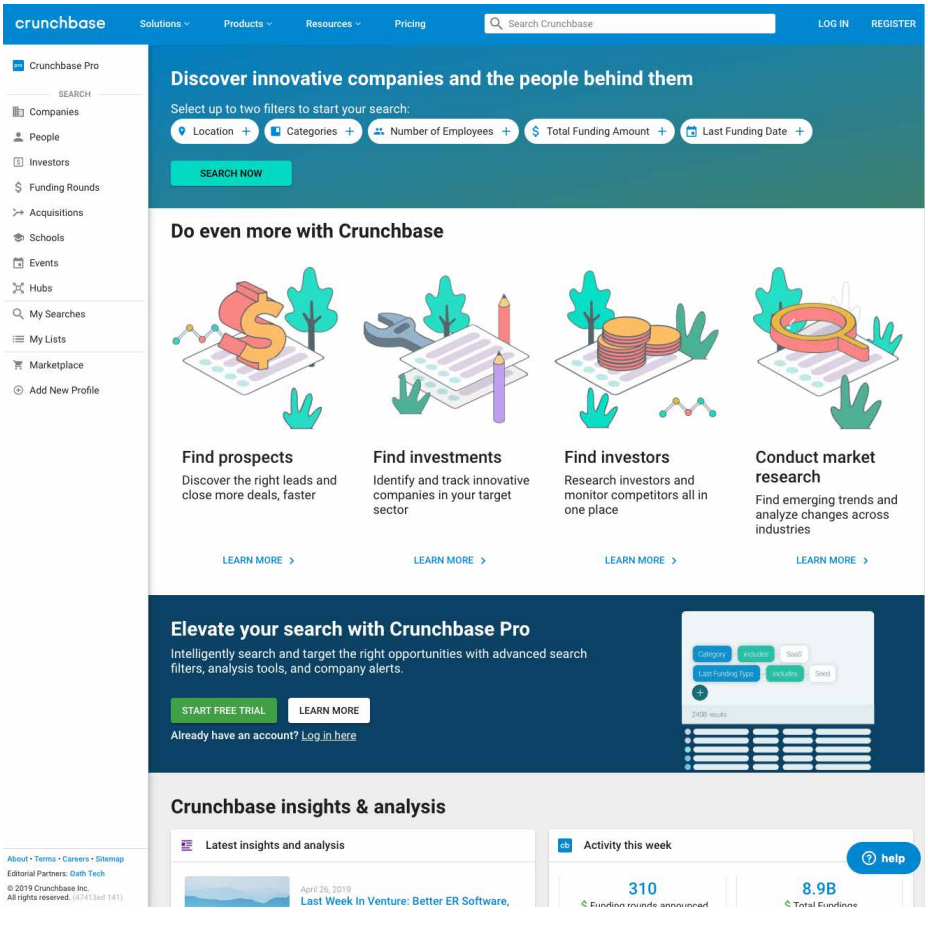


Figure 2-9. *Crunchbase*

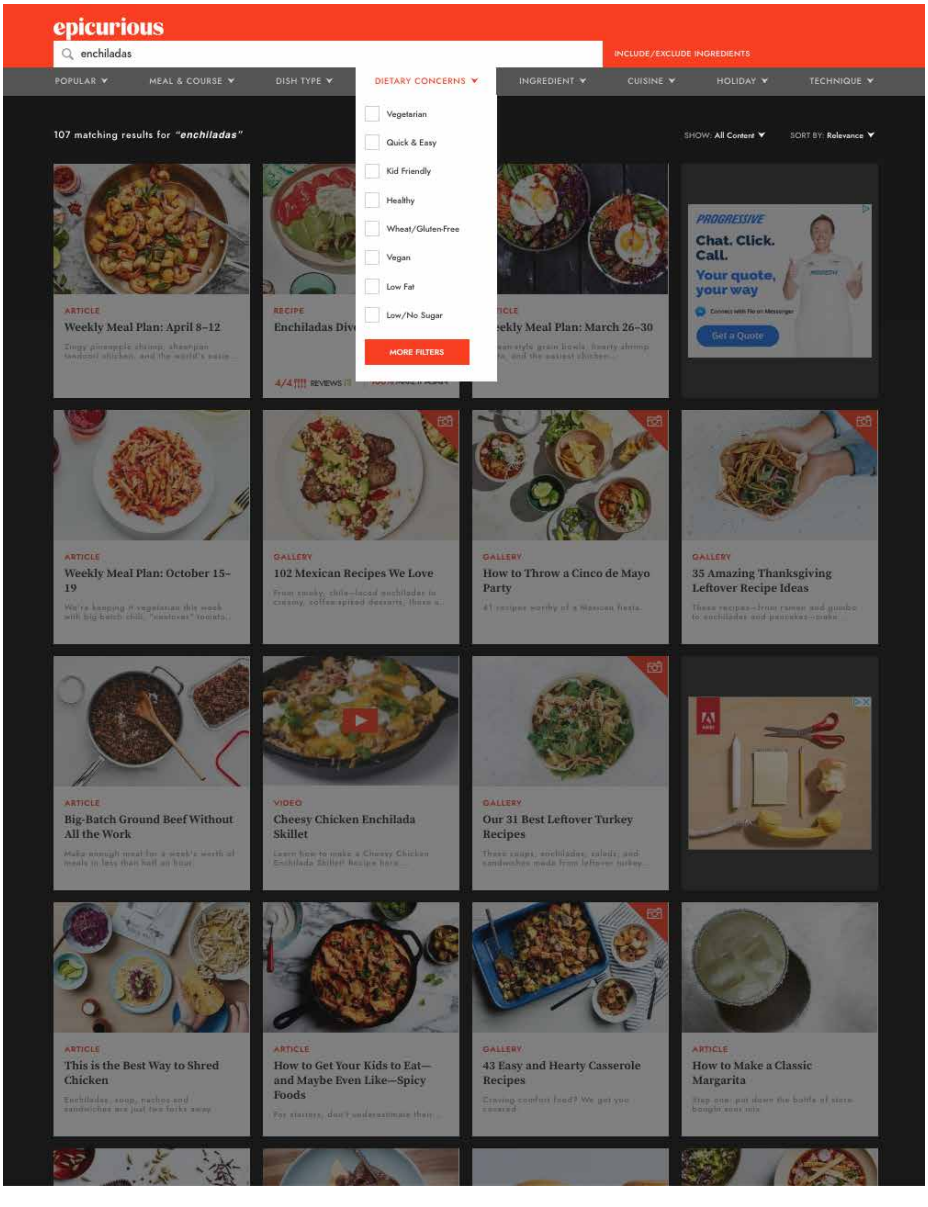


Figure 2-10. *Epicurious search results with faceted filters*

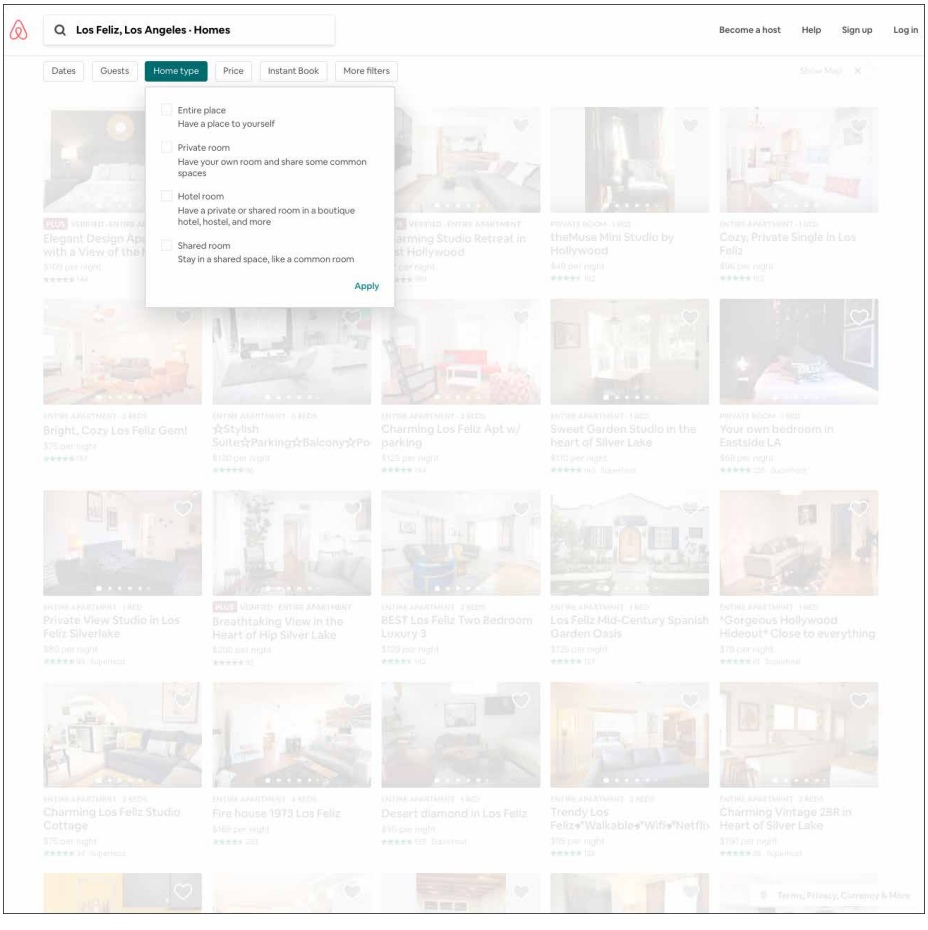


Figure 2-11. *Airbnb search results with faceted filters*

Mobile Direct Access

What

The first screen presents actionable information without requiring any input or action from the user. The app makes assumptions about any settings or queries (such as location or time) related to its primary function, and presents the output for immediate response. Making assumptions about what the user is most likely to do also drives what appears on launch.

Use when

Your mobile app generates value by doing one thing really well, or is used or known for one primary thing.

Why

Starting the user with an immediately actionable mode, choice, or screen provides instant value to the user, and gets them engaged instantly. There is limited ability to enter search or any kind of choice or configuration, but there is useful information available from the device and the primary use cases to make this assumption highly likely to be valuable, even expected.

How

Use live data from the user's mobile device (assuming the user has given permission in the settings). Primarily, look at location and time to generate a meaningful landing screen for the user. Make assumptions about what the user is most likely to be doing with your app, and get them as close to completing the action as possible, with a minimum of input.

Examples

All of these examples are mobile but could be valuable in desktop settings. In the first example, Snap (Figure 2-12, left) lives up to its brand as a photo-centric social media and camera company. When the app is launched, the user-facing camera is automatically turned on, ready to take a selfie. The next three examples all show how the apps use location and time data to return meaningful results with no input from the customer. INRIX ParkMe (Figure 2-12, right), Eventbrite and Weatherbug (Figure 2-13) give useful results this way. ParkMe makes some smart prefill assumptions (park for one hour) to gather price results by default.

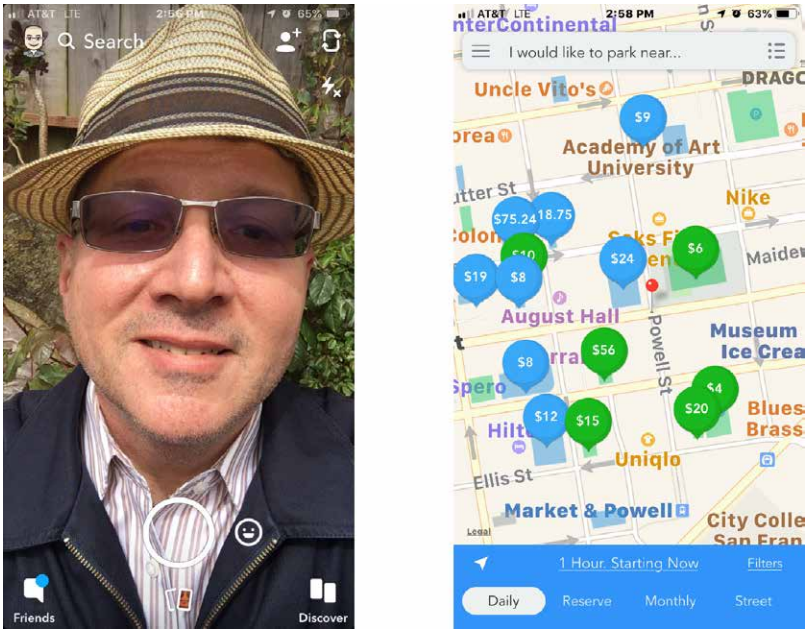


Figure 2-12. *Snap and INRIX ParkMe start screens*

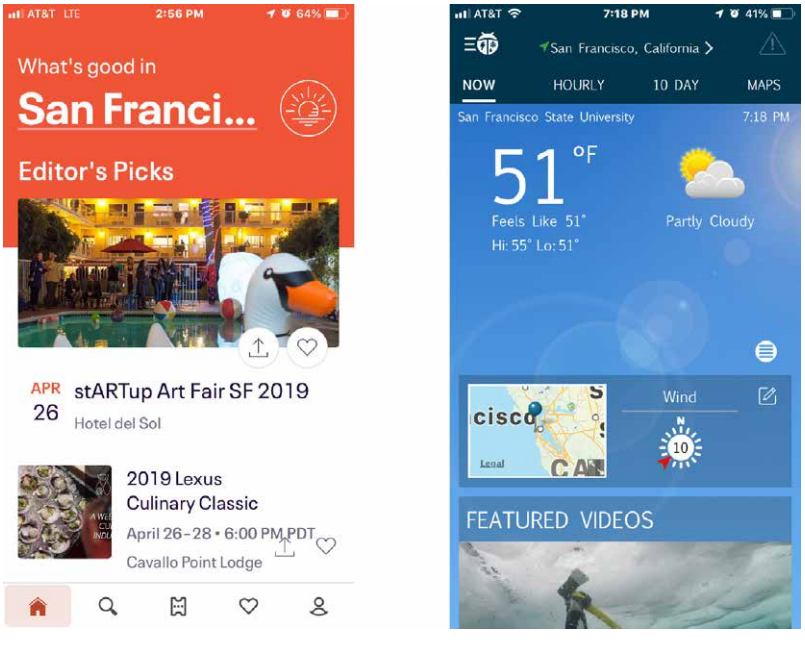


Figure 2-13. *Eventbrite and Weatherbug start screens*

Streams and Feeds

What

A continuously updated series of images, news stories, web articles, comments, or other content presented in a scrollable, vertical (sometimes horizontal) strip or ribbon. The items in the feed are usually presented as "cards," with an image from the article, a headline, introductory copy, and the name of the source with a link.

News/content streams. Content can be first party (publish your own content) or aggregated from third parties. This news stream approach, inspired by social media, is an established, mobile-friendly pattern for publishers. This usually consists of time-stamped items in a reverse chronological list that updates dynamically.

Social streams. Content comes from other members that the user follows (plus options to have editors highlight certain posts, or insert sponsored content). Again, this is usually a list of items in reverse chronological order, updated dynamically.

Business collaboration. Social media technology has now integrated into the way we work online. It is common for companies of all types to use online tools to collaborate. This allows employees and teams from any location to still come together online for discussion and feedback. Social media–style comment feeds are one of the most common patterns. There is often an important difference in how it is displayed: it is the standard in "collaboration-ware" and messaging-based apps for the most recent entry or posting to be at the bottom of the activity feed. Older comments move up as newer comments post at the bottom, where the newest post appears. Discussion channels in Slack are great examples of this model.

Use when

Your site or app has frequently updated content and the user checks it often, often dozens of times a day. Also, use this pattern when you have multiple collaborators on a project, and you need to stay on top of comments and feedback from multiple people. This feedback often comes asynchronously, meaning different people give feedback at different times. This is especially common in distributed or remote teams. For news publishers and aggregators, use one or more source channels, such as own original content, blogs, major news sites, other social site updates, and content partners to deliver timely content to users.

The pure social version can be personal—a user "owns" it, as for social media sites like Twitter or Facebook friends list.

For business collaboration software, use this pattern to allow people to view, comment, and edit a document. The document and the discussion are presented together. Employees can scroll through the comment feed to see the history of the discussion.

Why

Ensures that new content is always appearing first in the list of items in the user's feed. This makes each visit a reward, with something new to see and to scroll through. People can keep up with a news stream easily because the latest items reliably appear first in the list with no effort on the part of the user. This promotes the habit of checking back frequently and spending lots of time reading, following, and interacting with their stream.

People go to many sites or apps each day to keep up with their friends' activities, engage in conversations, or follow topics or blogs of interest. When multiple "news" sources can be blended in one place, it's easier to keep track of it all.

From the perspective of a publisher, such as a news website, publishing your own content in the feed or stream format promotes engagement, return visits, and interaction.

From the perspective of a business, social collaboration software allows employees to be more efficient and to save time. Remote workers and employees in different locations and time zones can still come together asynchronously to get work done.

This pattern supports the *Microbreaks* behavior pattern that we introduced in Chapter 1. A glance at a *Streams and Feeds* application can give a user lots of useful information (or entertainment) with very little time or effort.

How

From its origins as a social media innovation, this pattern has become a common one for any company, app, or site that publishes or aggregates large amounts of content continuously, for social media, and for business collaboration software. The following discussion assumes a chronological ordering for the streams and feeds, but this is only one way. Indeed, nowadays the stream sequence is determined by algorithms, which might be optimizing for engagement, clicks, customer interest, or other parameters. So, consider that more personalized, context-responsive experiences are possible.

List incoming items in reverse chronological order. Display newest items at the beginning of the list without waiting for the user to request an update. Older items are pushed further away by the newer comments or entries. Offer a way for the user to get an immediate update or refresh. Also, they need to be able to scroll or review through the list to get to the older, unreviewed items.

Offer publisher-curated streams that the user can view in addition to their own social stream. Offer advanced users the ability to create custom streams based on topics or curated lists of other members. Others, such as TweetDeck, use *Many Workspaces* to show multiple parallel panels of incoming content.

Information shown with each item might include the following:

What
: For short micro-updates, show the entire thing. Otherwise, show a title, a teaser that's a few words or sentences long, and a thumbnail picture if one is available.

Who
: This might be the person who wrote an update, the blog where an article was posted, the author of said article, or the sender of an email. It could be the coworker who posted a comment or document. Actual names of the people humanizes the interface, but balance this against recognition and authoritativeness—the names of news outlets, blogs, companies, and so forth are important, too. Use both if that makes sense.

When
: Give a date or timestamp; consider using relative times, such as "Yesterday" and "Eleven minutes ago." As the post ages, switch to the traditional date-and-time stamp.

Where
: If an item's source is a website, link to that website. If it comes from one of your organization's blogs, link to that.

  When there's more to an item than can be shown easily in the list display, show a "More" link or button. This is a good pattern for long comments. For news or story cards, allow the reader to click on the card to load the full story as a new screen. You might design a way to show the entire contents of an item within the news stream window. The News Stream is a list, so you can choose among *Two-Panel Selector*, *One-Window Drilldown*, and *List Inlay*. Examples abound of each model.

  Give the user ways to respond immediately to incoming items. Stars, thumbs-up, liking, and favoriting are available in some systems—these all provide low-effort feedback and "handshaking" among people who don't have time to write out long replies. But allow those long replies to be written, too! By placing controls and text fields immediately next to an item, you encourage responsiveness and interaction. This is usually a good thing in social systems.

  As of this writing, *Streams and Feeds* designs for mobile devices are fairly straightforward. Almost all of them devote the full screen to a single list—often an *Infinite List* (Chapter 6) with richly formatted text—and users can drill down to an item by simply tapping or clicking it in the list.

Many *Streams and Feeds* services, including Twitter and Facebook, use the *Infinite List* pattern for both their mobile and full-screen designs. This pattern lets users load a page or two of the most recent updates, and it gives the option of loading more to go "backward in time."

Activity history

Some resources use the term *activity stream* for a very closely related concept: the time-ordered stream of actions (usually social actions) performed by a single entity such as an individual, system, or organization. It is a history of their actions. This is a useful concept, and it doesn't really conflict with a *Streams and Feeds* pattern, which talks about the stream of activities that are of interest to an individual or group of users, not generated by them. A news stream will usually have multiple diverse sources.

Examples

News/content streams. Techcrunch (Figures 2-14 and 2-15) is a great example of a *Streams and Feeds* publisher. The main mobile app and website is a scrollable stream of stories, with the most recent stories first in the list of content. At this level, the reader is provided with just enough information to get the main idea: a photo, a headline, and some introductory copy. If the reader selects the story, they go to the full version, with larger images and full text. This detail page is where the social sharing feature is available, promoting distribution to the reader's own social networks.

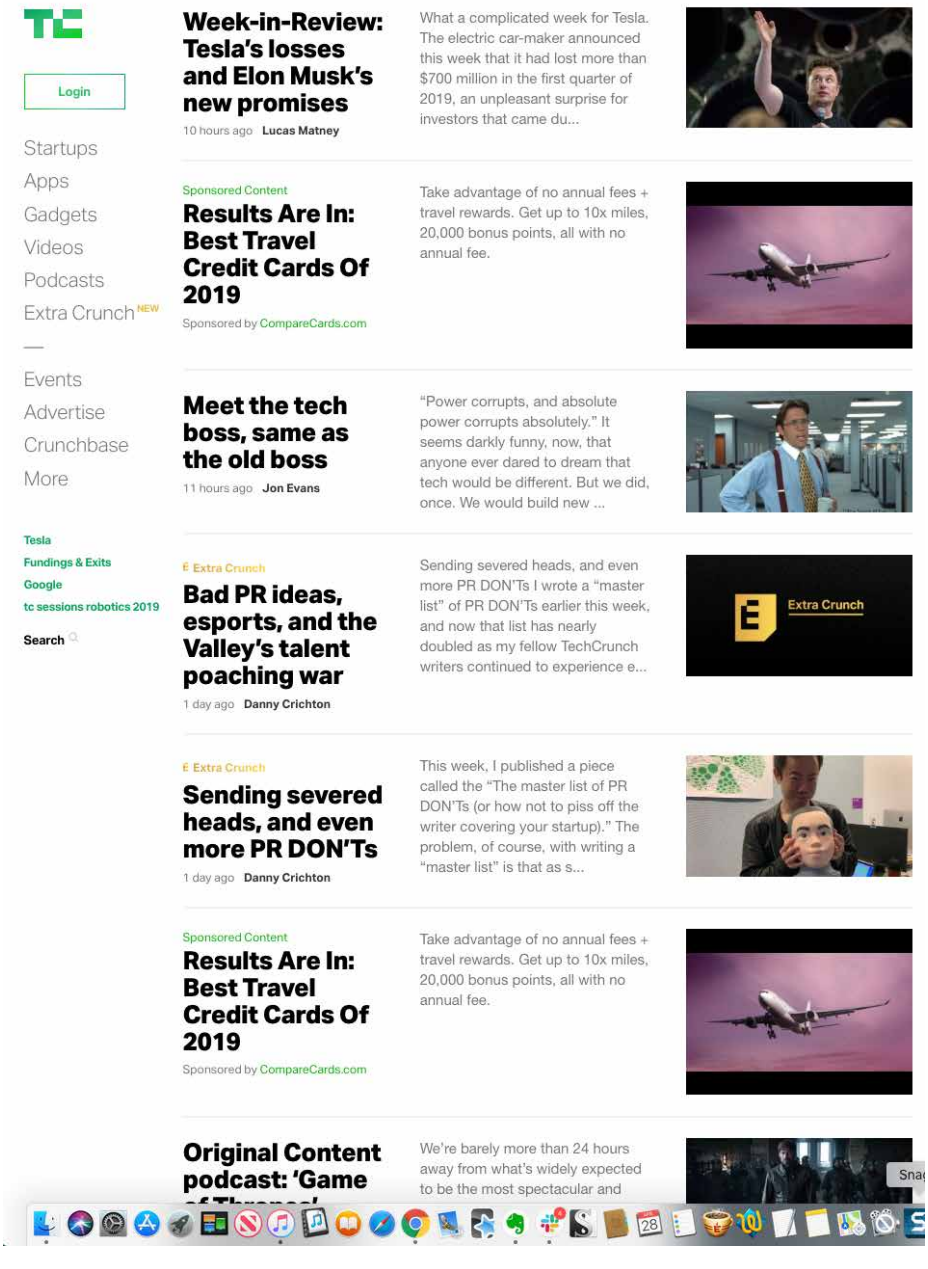


Figure 2-14. *Techcrunch*

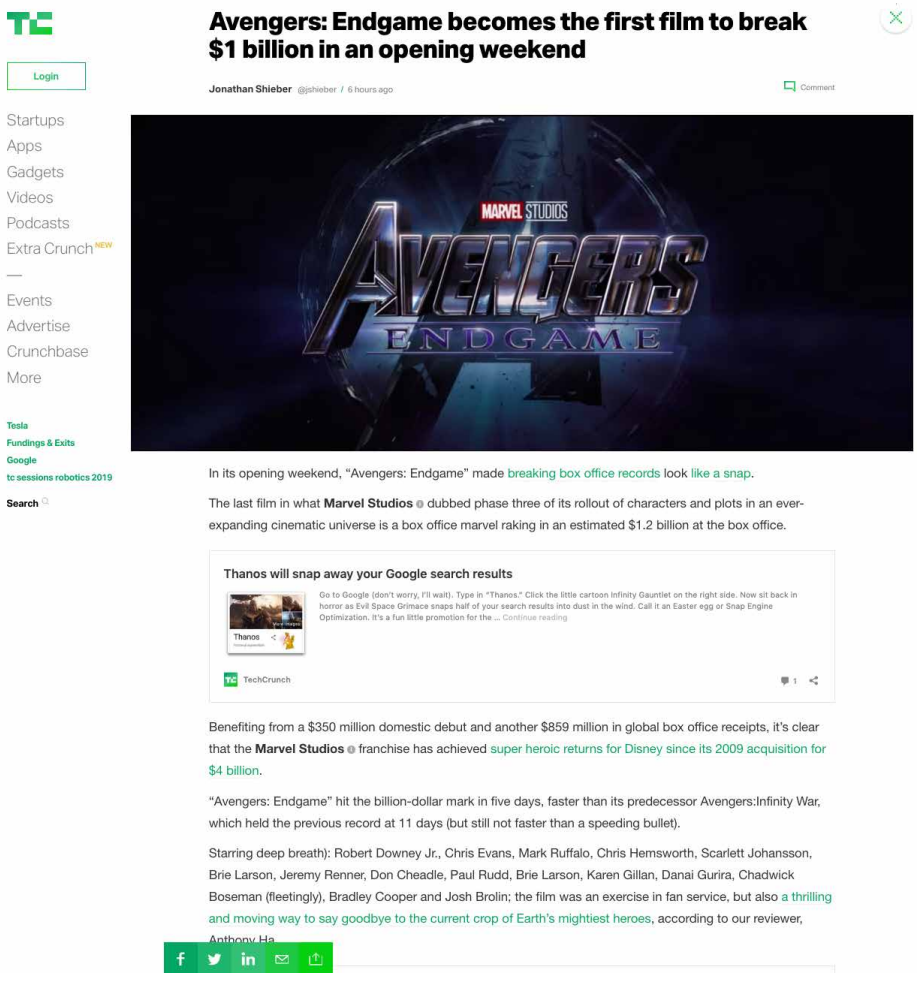


Figure 2-15. *Techcrunch Detail/Individual Article screen*

BuzzFeed News (Figures 2-16 and 2-17) follows exactly the same pattern. Note that this company uses the word "feed" in its name. This shows how important this pattern is to the company's identity and purpose. Again, we see a scrollable stream of stories, with the most recent stories first in the list of stories. BuzzFeed's strong editorial voice is clear in the enticing headlines and reader-challenging questions. Selecting a story loads the detail screen, with the full story, quiz, or gallery of images. The social sharing widget is even more prominent.

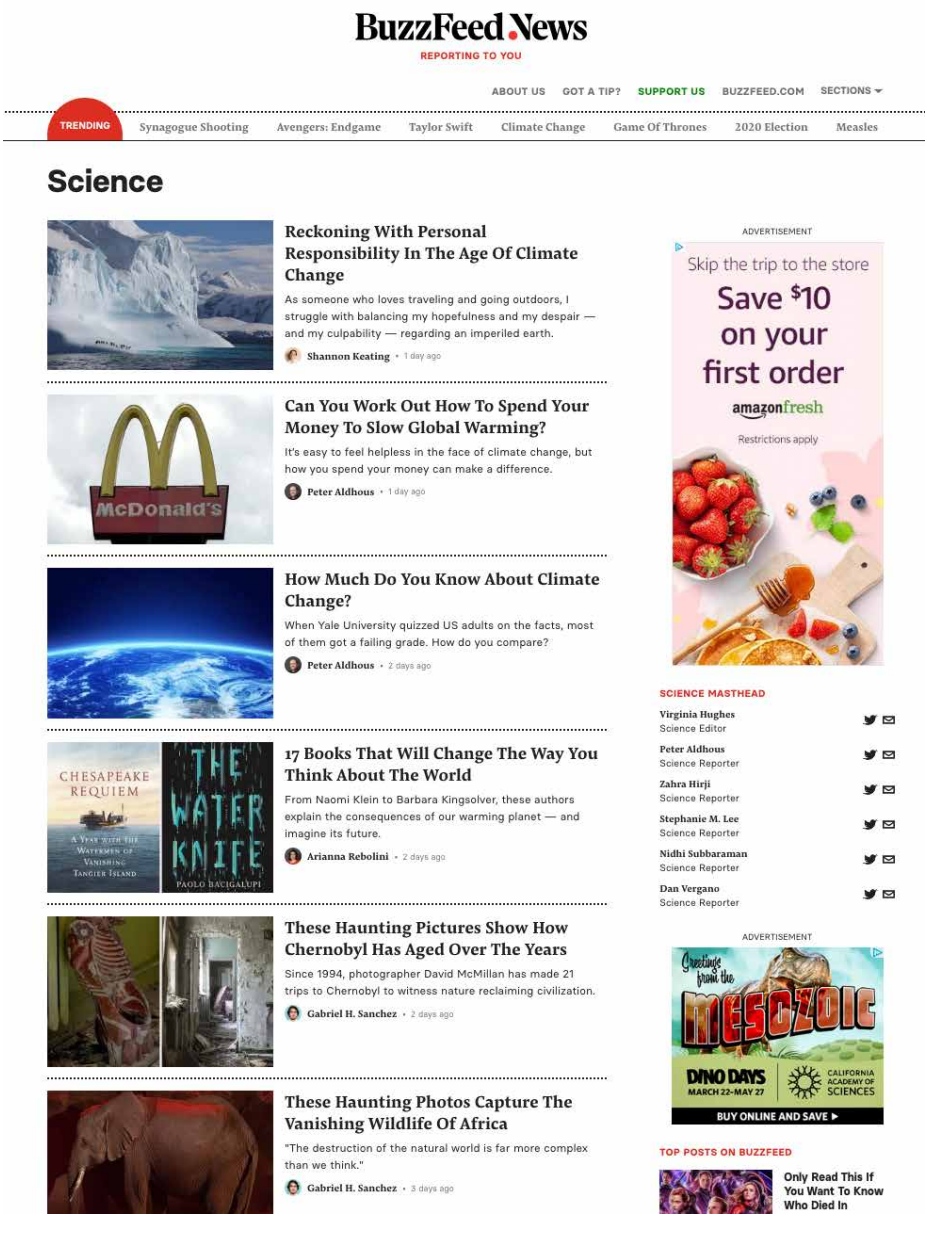


Figure 2-16. *BuzzFeed News*

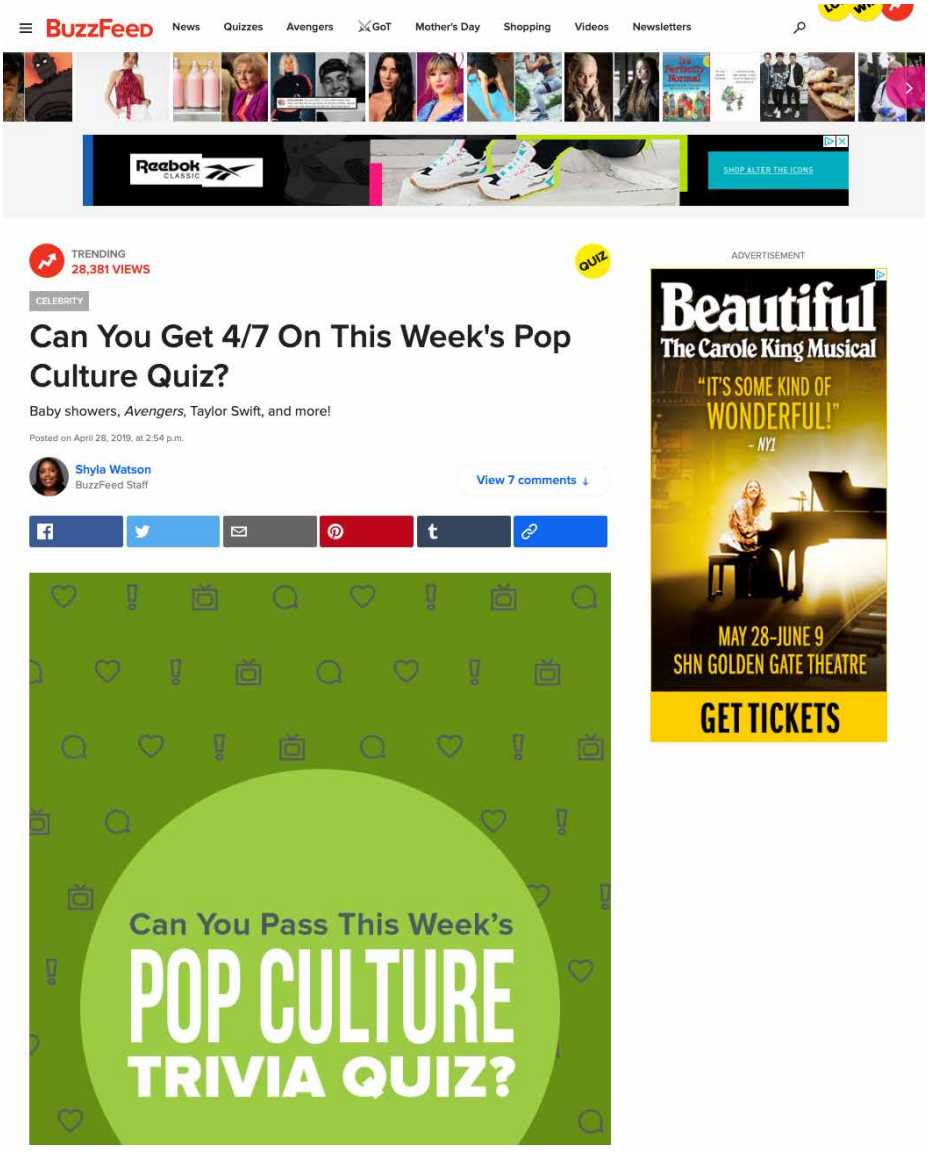


Figure 2-17. *BuzzFeed News Detail/Individual Article screen*

The RealClear family of sites is based around aggregating links to stories published elsewhere on the web. The most active is RealClearPolitics (Figure 2-18). Although this looks like a simple, flat list that you might find in a wiki or encyclopedia, it is a feed. The links are published multiple times each day. These updates, taking a cue from the history of print newspapers, have the labels "Morning Edition" and

"Afternoon Edition." Note how RealClearPolitics mimics the newspapers of the past, with a "morning edition" of curated links, followed by an "evening edition" of new links. The reader can scroll or use a menu selector to review previous days' links. It's an endless feed of a curated, time-based list of stories.

Figure 2-18. *RealClearPolitics*

Flipboard (Figure 2-19) looks like a magazine or picture viewer but is actually a feed reader. It can link to and aggregate from your social media accounts or from popular publishers with feeds. You can also use hashtag keywords to create feeds of matching articles from across all feeds.

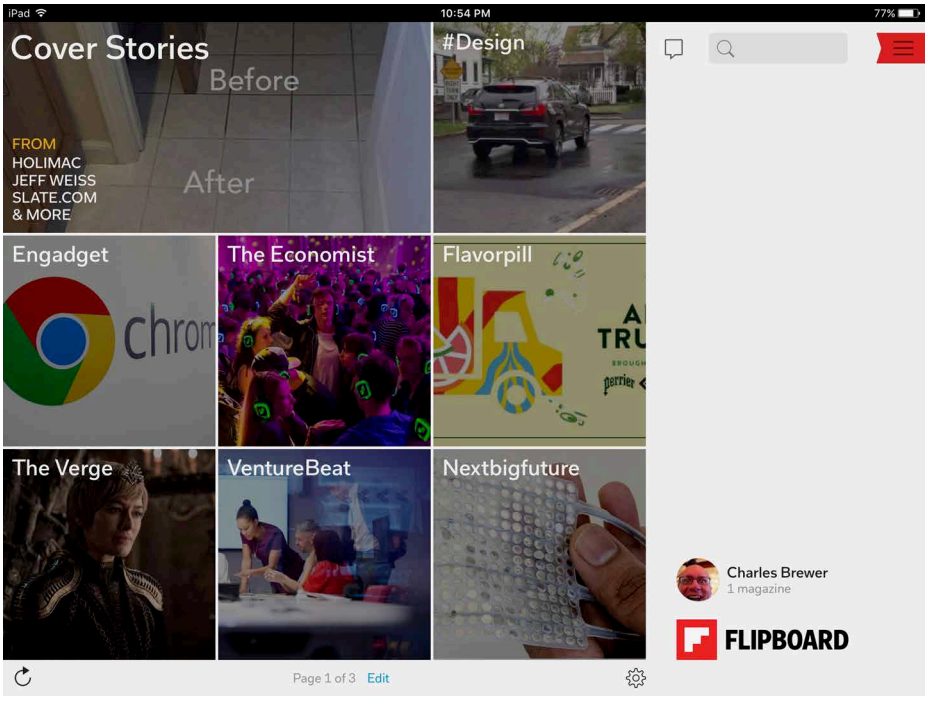


Figure 2-19. *Flipboard start screen*

Flipboard, again, is actually a feed that has rendered the content cards in a variety of sizes, organized into pages like a book (Figure 2-20). The user swipes left to right just like browsing a magazine.

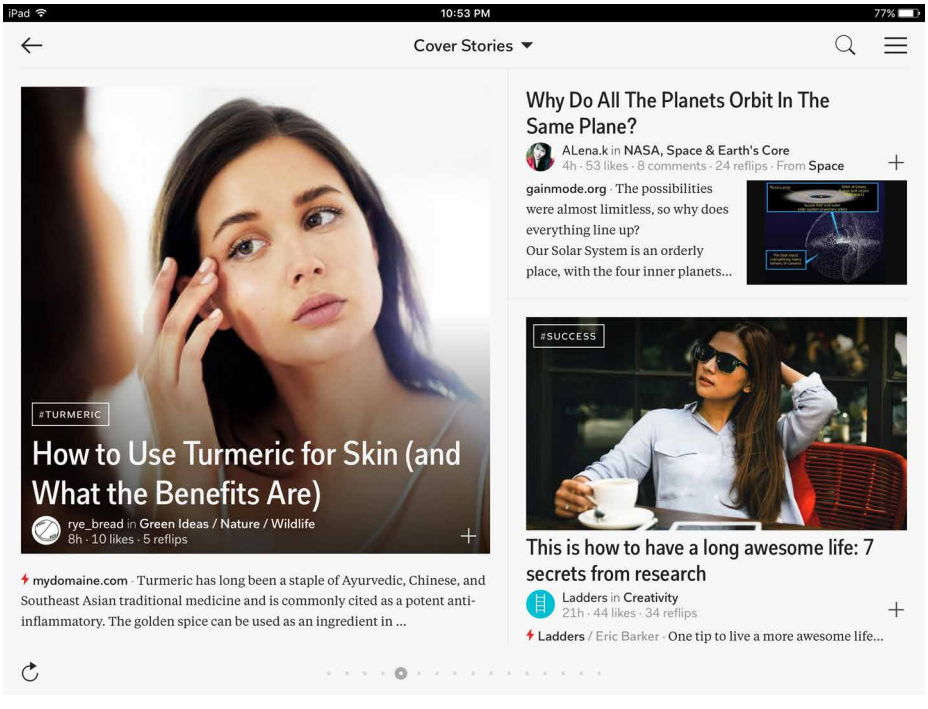


Figure 2-20. *Flipboard news stream*

Social streams. Social streams dominate the consumer internet experience. This doesn't show any signs of ending anytime soon. This is because streams drive engagement. As shown with Twitter (Figure 2-21) and Instagram (Figure 2-22), the social stream is alive and well. The feed can be a mix of content or posts or cards from business and personal networks, as with Twitter. Or it can be (almost) purely a personal social feed, as with Instagram. The feed is the consumption format here: view the image, the comment, and then use the social feedback features to like, share, or comment in turn.

Social networking services, news aggregators, and private communications (such as email) provide plenty of examples of personal *Streams and Feeds*.

Facebook automatically (and unpredictably) switches between a filtered Top Stories view, and a Most Recent view that shows everything. However, Facebook excels at the immediate response. Posting a short comment to a Facebook entry is almost as easy as thinking about it.

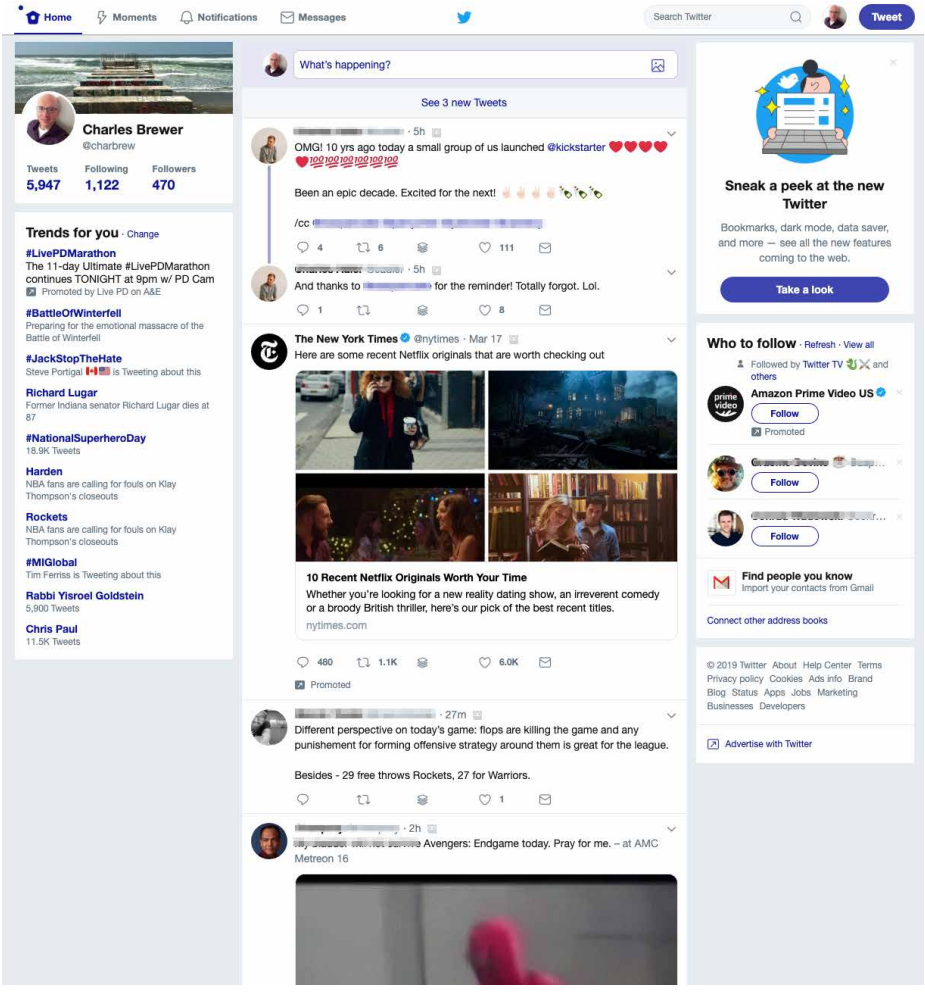


Figure 2-21. *Twitter*

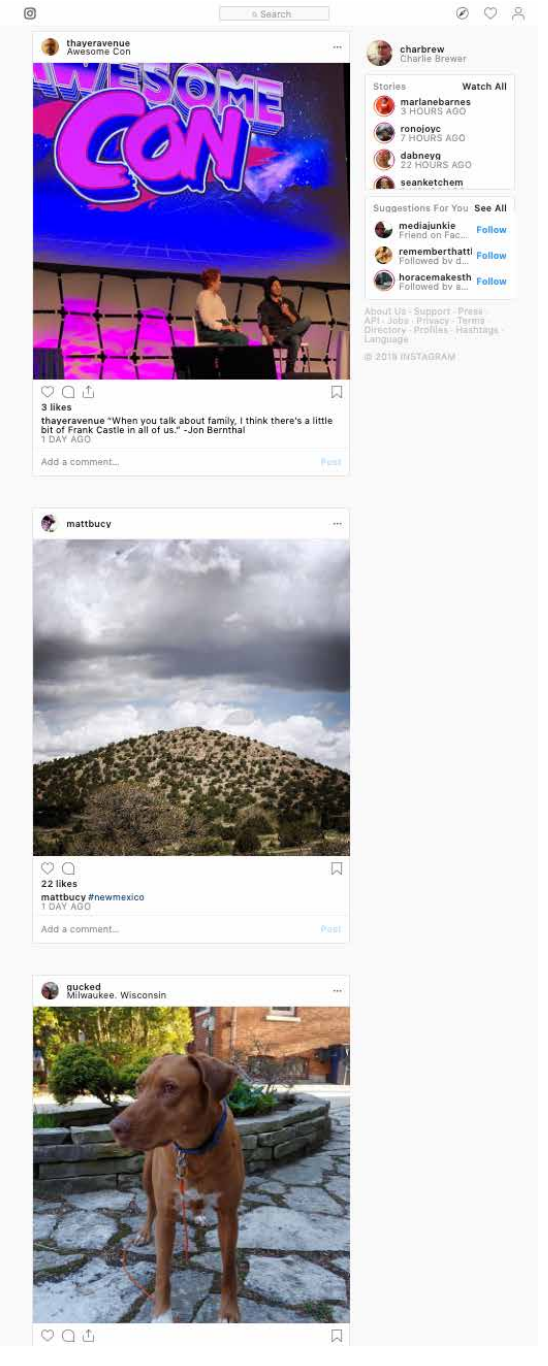


Figure 2-22. *Instagram*

Business collaboration. Social feeds and streams have made the jump from consumer experiences to business. They are the key component for enabling online, distributed and remote work collaboration. They allow employees to work together in a number of ways. They can start or add to threaded discussions organized by topic. Or they can congregate around a digital document that is generated collaboratively. This work can happen in real time or asynchronously. Employees can be in the same geographic location, or distributed across time zones. In Slack (Figure 2-23), the whole platform is built around discussion topics. Within the company "space," employees can start or contribute to group discussions, or start private chat sessions with one or more coworkers. Files can be shared directly in the feed. In Quip (Figure 2-24), a digital document is the anchor. Multiple collaborators work on this document. The social and comment feed next to the document gives a history of the discussion around the document. Both of these approaches are now a standard part of many business applications.

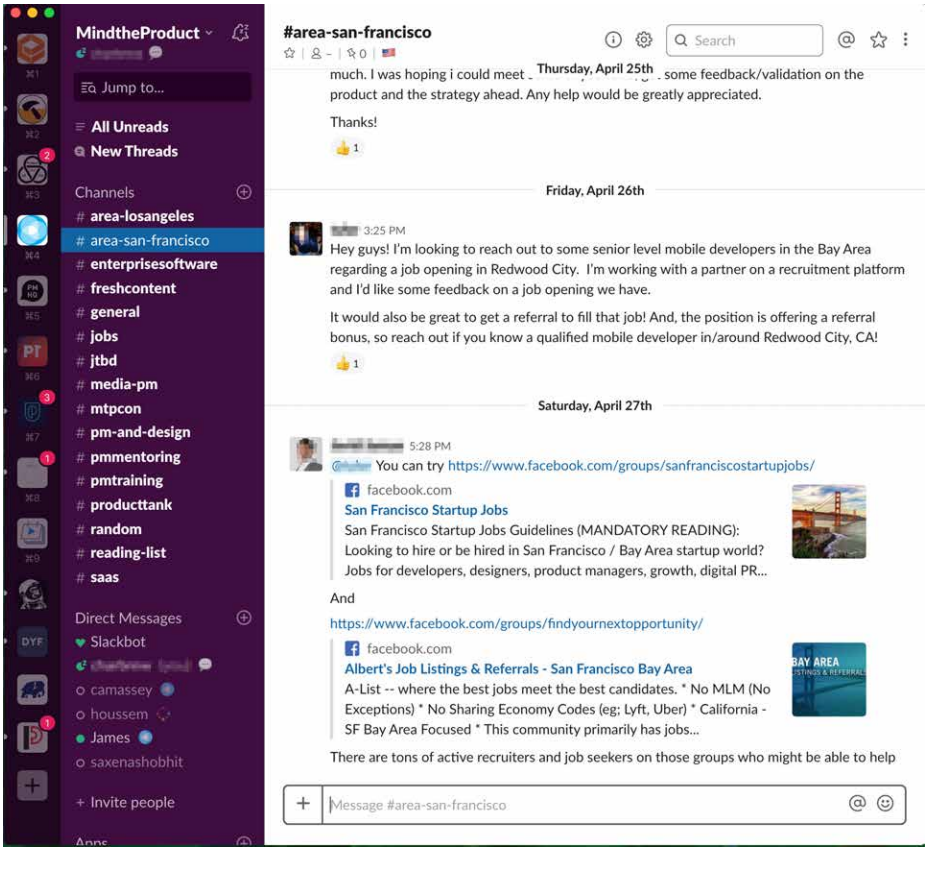


Figure 2-23. *Slack*

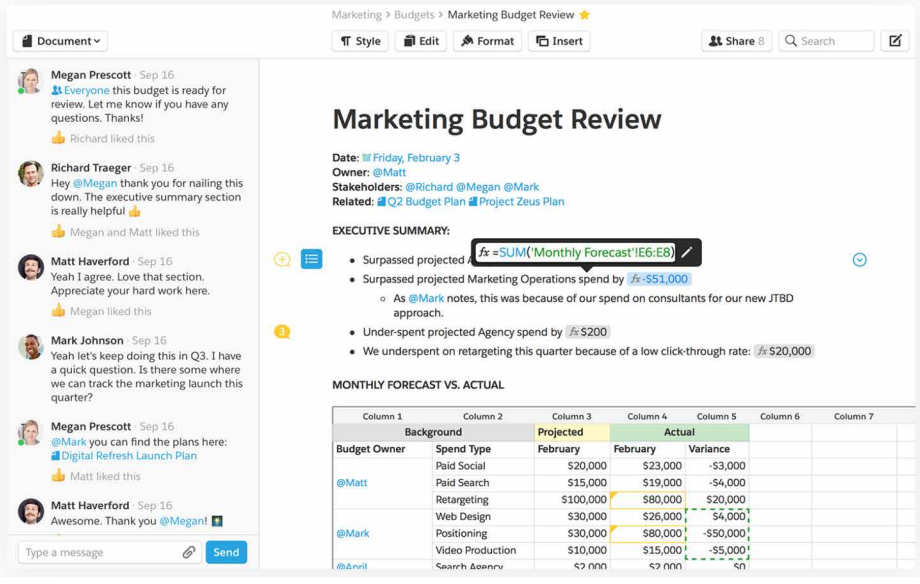


Figure 2-24. *Quip*

Media Browser

What

A "grid of objects" structure is for browsing and selecting from a group of objects. The objects are presented in rows and columns. It uses thumbnails, item views, and a browsing interface, such as a scrolling list. For content-centric sites and applications, it allows an overview of the files, stories, or documents available to read. Media browsers are also common for managing media and editing photos, videos, and other pictorial items.

Why

This is a distinct style of application that is commonly used for mobile and desktop. As soon as someone sees the grid of image and video objects in the media browser, they know what to expect: browse, click to view, set up slideshows or playlists, and so on.

Patterns and other components described elsewhere in this book that often use media browser structures include the following:

- *Grid of Equals*
- *One-Window Drilldown*

- *Two-Panel Selector*
- *Pyramid*
- *Module Tabs* and *Collapsible Panels*
- *Button Groups*
- Trees or outlines
- *Keyboard Only*
- Search box
- Social comments and discussion

How

Set up two principal views: a matrix or grid layout of the items in the list, and a large view of a single item. Users will go back and forth between these. Design a browsing interface and associate it with the *Media Browser* to let users explore a large collection easily.

The media browser. Use this pattern to show a sequence of items. Many apps show a small amount of metadata with each item, such as title or author; but do this with care because it clutters the interface. You might offer a control to adjust the size of the thumbnails. There might also be a way to sort the items by different criteria, such as date, label, or rating, or to filter it and show only the starred items (for instance).

When a user clicks an item, show it immediately in the single-item view. Applications often let the user traverse the grid with the keyboard; for example, with the arrow keys and space bar. (See the *Keyboard Only* pattern in Chapter 1.)

If the user owns the items, offer ways to move, reorder, and delete items at this level in the interface. This implies having a multiple-selection interface, such as Shift-select, checkboxes, or lassoing a group of items with the pointer. Cut, copy, and paste should also work in applications.

You can offer slideshow or playlist functionality to all users at the *Media Browser* level.

The browsing interface. The contents of the *Media Browser* should be driven by a browsing interface that might be complex, simple, or nearly nonexistent, depending on the nature of the application. If needed, interfaces should offer a Search box, either to search an individual user's items or to search all public items (or both). Alternatively, just present a scrollable grid.

Websites that host public collections, such as YouTube and Vimeo, use the entire home page as a browsing interface. Sites such as these often present a balanced view with user-owned content and also public or promoted content.

Private photo and video management interfaces—especially desktop apps such as Apple Photos or iMovie—should let the user browse the filesystem for images stored in different directories. If users can group items into albums, sets, projects, or other types of collections, these should be available in a browsing interface, too. Most also permit favoriting or starring of items.

Adobe Bridge puts filters into its browsing interface; more than 10 properties can be used to slice through a large collection of items, including keywords, modification date, camera type, and ISO.

The single-item view. This displays a full story or document or image using the entire screen so that the user can read it, edit it, or comment/share. This is the detail or full-view screen as a screen on its own. Alternatively, show a large view of the selected image (or a player, for a video). This view can be next to the media browser grid if the window is large, or it might replace the area used by the grid. Display metadata—information about the item—next to it. In practice, this means choosing between a *Two-Panel Selector* and a *One-Window Drilldown*. See Chapter 7 for these list-related patterns.

If the interface is a website or is otherwise web-connected, you might choose to offer social features at this level. Comments, liking or thumbs-up, and sharing might be here. Likewise, users also can tag or label here, either privately or publicly. An "other items you may like" feature is sometimes found in web-based public collections.

Editing features for individual items will reside here, as well. For instance, a photo manager might offer simple functionality such as cropping, color and brightness adjustment, and red-eye reduction. Metadata properties could be edited here, too. If a full editor is too complex to present here, give the user a way to launch a "real" editor. (Adobe Bridge, for example, lets the user launch Photoshop on a photo.) Use *Button Groups* to maintain a simple, comprehensible visual grouping of all these features.

Link the item to the previous and next items in the list by providing "previous" and "next" buttons, especially if you use *One-Window Drilldown* to display the single-item view (which also requires a Back button). See the *Pyramid* navigational pattern in Chapter 3.

Browse a collection of objects. The power of images is that they can carry a huge amount of information and can be recognized quickly. This is why they are so frequently used to represent a collection of objects for browsing and selecting. A grid of images presented with or without written descriptions is a compact, useful pattern for selecting a single item from a large collection. This pattern is universal across mobile, desktop, and large screen user interfaces like Apple TV. Selecting a single item or card from the media browser grid will load the object for direct consumption or it will load a detail screen with a description. In the Kindle reading app for iOS (Figure 2-25), the browser is nothing more than images of the book covers. In Instagram (Figure 2-26), one's profile has very little metadata in favor of a scrolling grid of previously posted images. The layout of images is your Instagram identity and personality. In YouTube (Figure 2-27), Apple TV (Figure 2-28) and LinkedIn Learning (Figure 2-29), full-screen browsers allow users to browse a large number of video assets quickly. Seen here also is a common variation on the square grid layout: a scrolling ribbon—a single height row that scrolls. Because of the huge number of items to browse, all of them group the images into categories for easier comprehension and review. Apple TV takes a minimalist approach. YouTube provides the most information, with each item becoming a "card" with image, title, author, and popularity metrics (not surprising since YouTube is built on social media dynamics).

When you view someone's YouTube channel, you can choose to see either a media browser or, in the single video view, a list beside a video player (the default). Clicking a thumbnail brings you to the page for that video, where detailed information and a discussion are shown. Visitors can browse by looking at playlists, the latest videos added, the most-viewed videos, and the top-rated videos; a Search box is also provided, as it is everywhere.

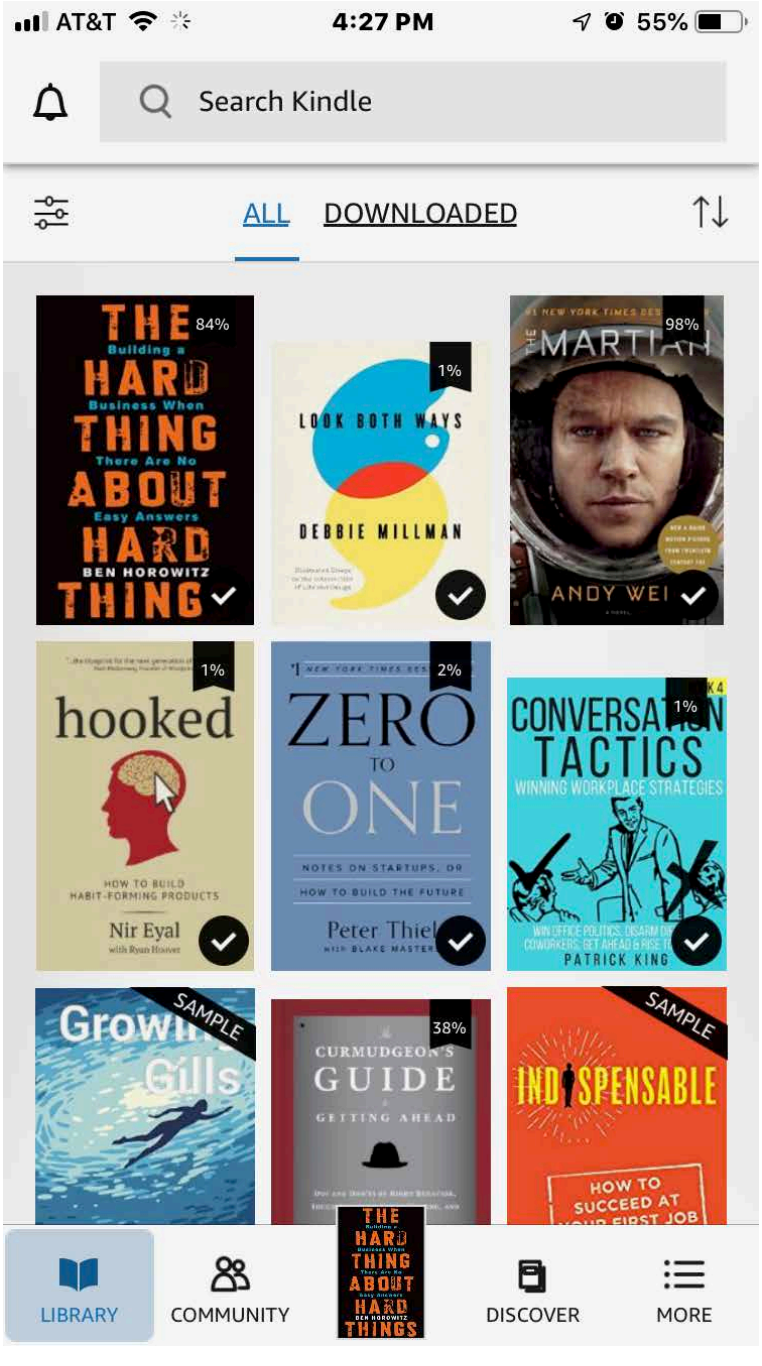


Figure 2-25. *iOS Kindle app*

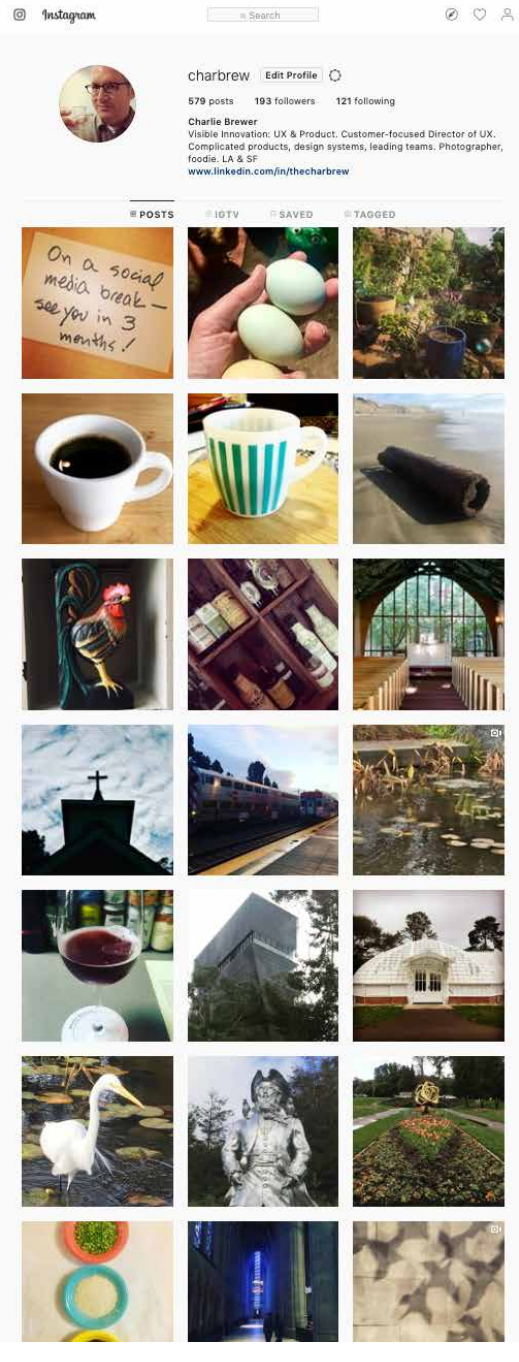


Figure 2-26. *Instagram profile screen*

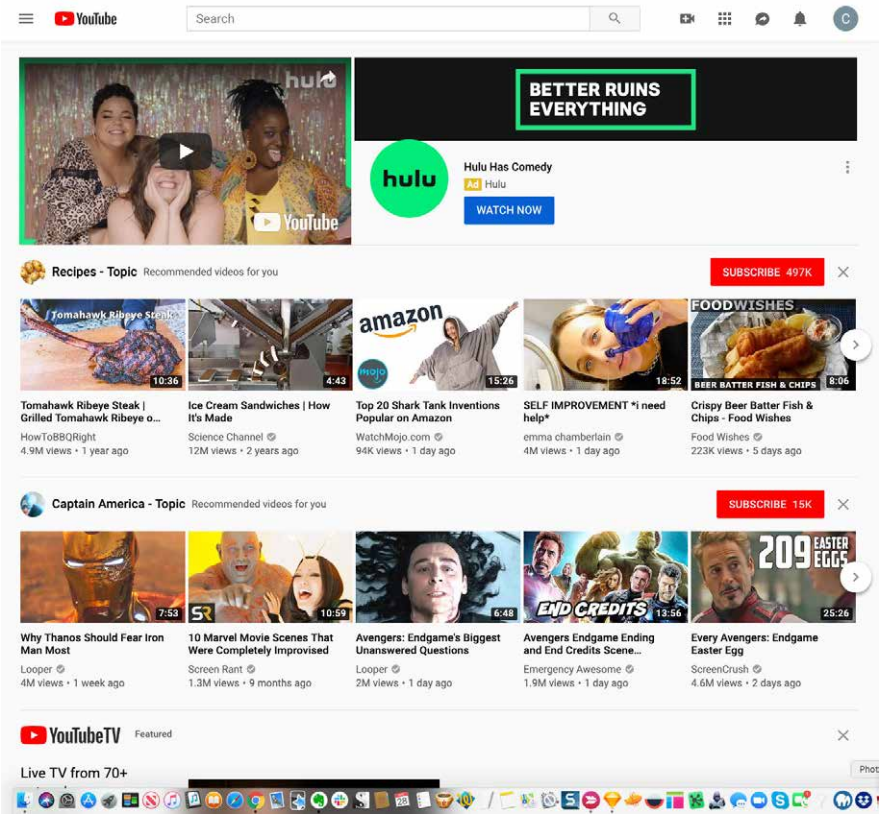


Figure 2-27. *YouTube*

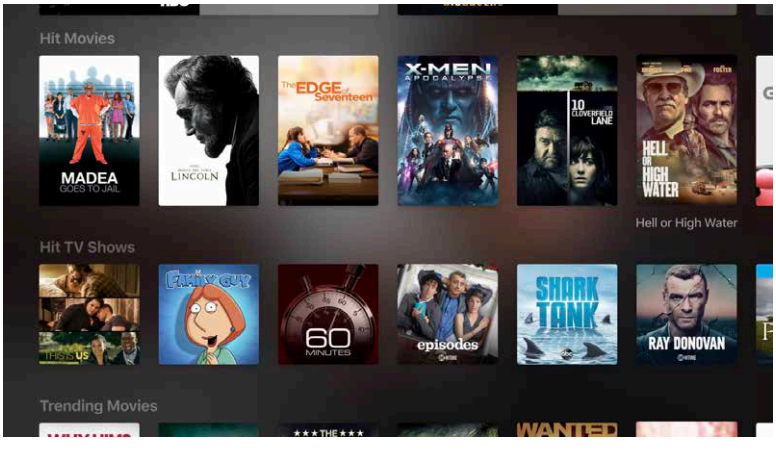


Figure 2-28. *Apple TV*

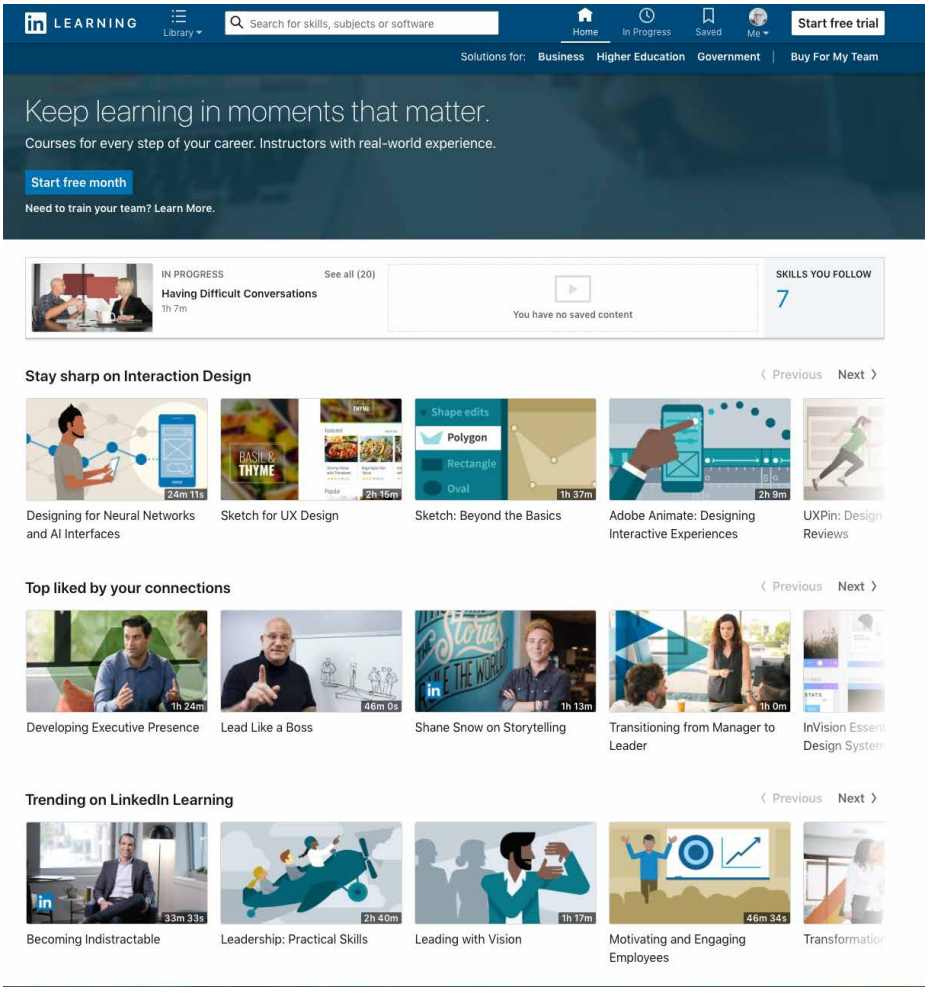


Figure 2-29. *LinkedIn Learning*

Manage and edit media assets. Media and document creators also use this media browser layout to manage assets that are being assembled or processed. Apple Photos (shown in Figure 2-30), Adobe Bridge (Figure 2-31), and Apple iMovie (Figure 2-32) are mobile desktop applications for managing personal collections of images. Their browsing interfaces—all *Two-Panel Selector*—vary in complexity from Apple Photos' very simple design to Adobe Bridge's numerous panels and filters. Apple Photos uses *One-Window Drilldown* to reach the single-item view, whereas Adobe Bridge and Apple iMovie put all three views together on one page. A common variation on the square grid layout is a single-height row that scrolls: a ribbon. In the case of iMovie, it is a timeline. It is the central working palette for creating time-based media such as videos.

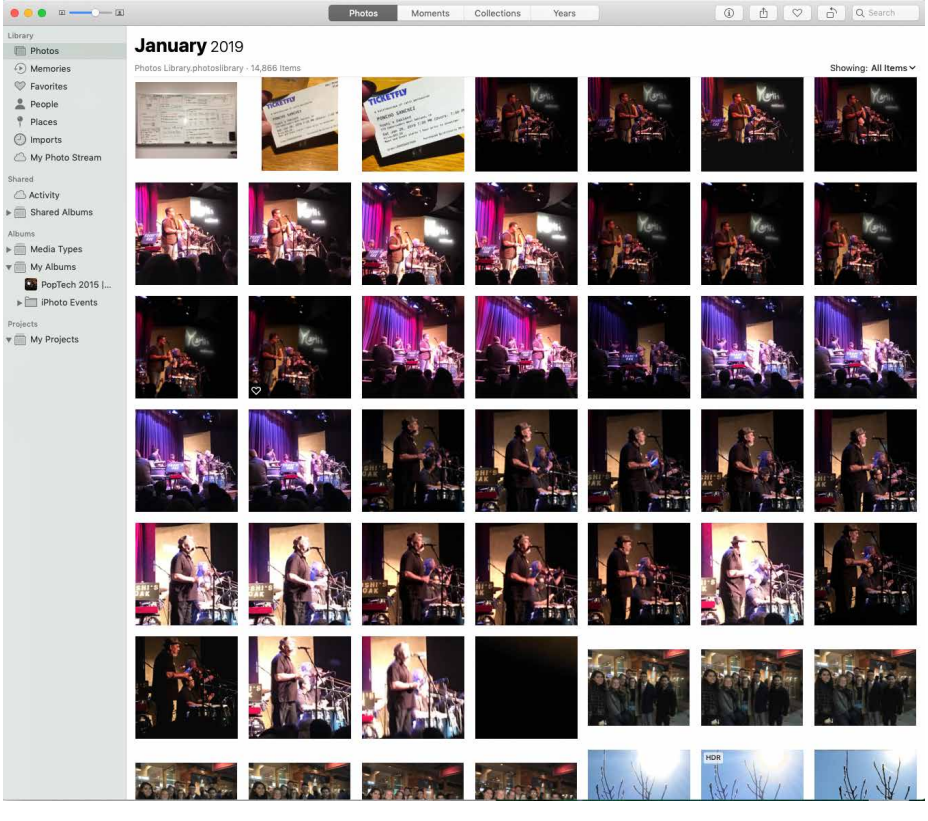


Figure 2-30. *Apple Photos*

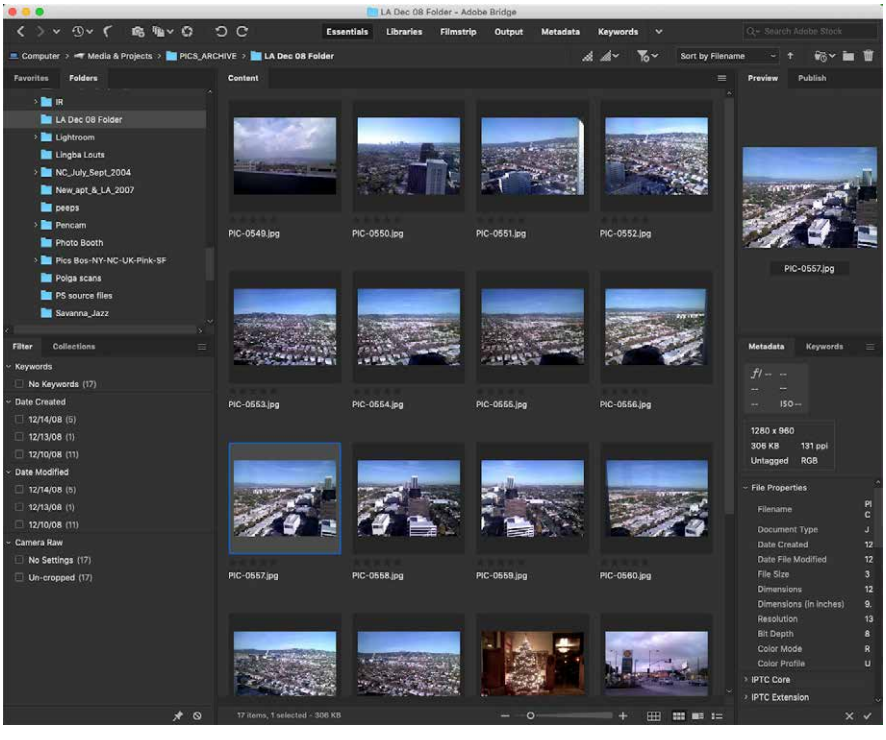


Figure 2-31. *Adobe Bridge*

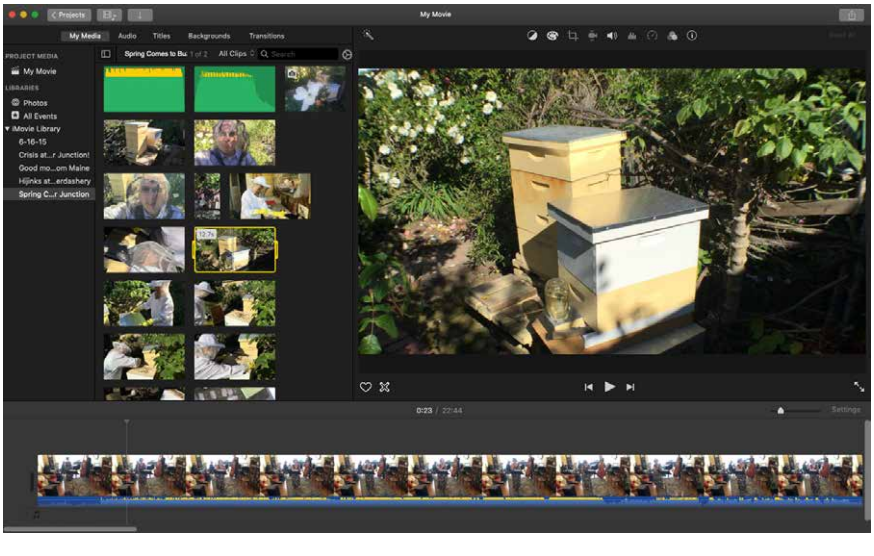


Figure 2-32. *Apple iMovie*

Adobe Acrobat (Figure 2-33), the reader/editor for the popular PDF document format, offers a grid view of document pages. This is used in the edit mode. It allows for rapid reorganization of pages, or selecting pages for deletion, or selecting an insertion point for adding screens.

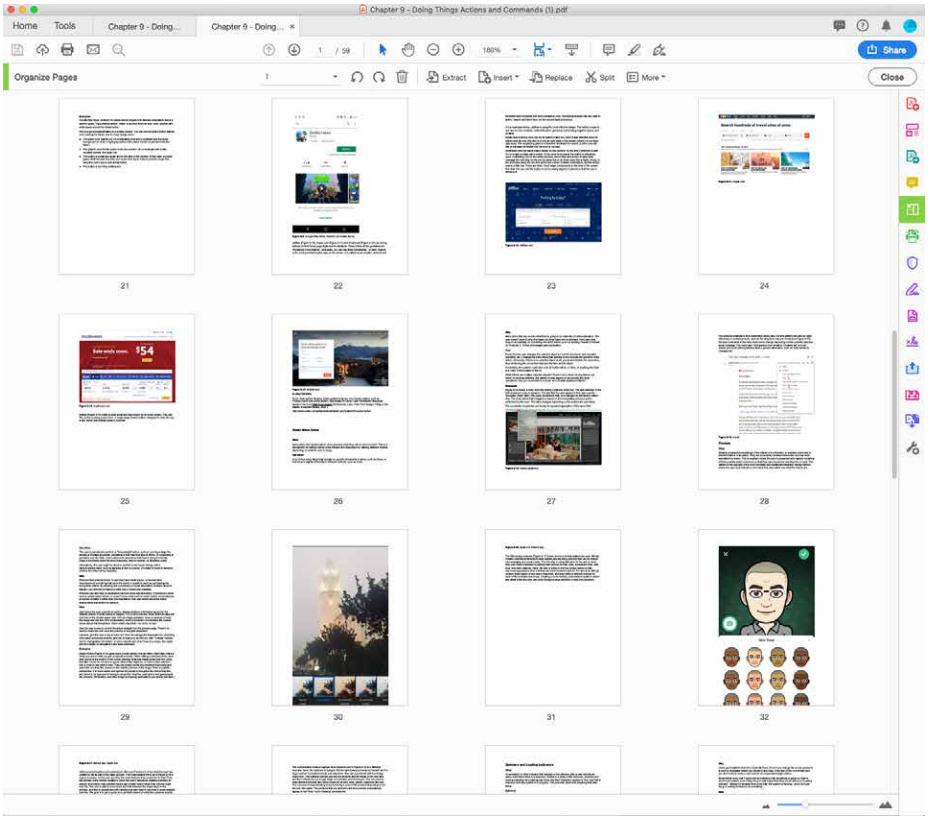


Figure 2-33. *Adobe Acrobat*

Dashboard

What

A dashboard is often the first screen a customer will see when logging in to a consumer or business platform. They are a very common pattern for business information software. Dashboards display key data points in a single information-dense page, updated regularly. They show users relevant, actionable information, charts and graphs, and important messages and links or buttons, often related to key metrics or workflows that are important to the business.

Dashboards are very popular because they solve the need to get a quick update on status, on key information, and on tasks to be done. Your site or application deals with an incoming flow of information from something—web server data, social chatter, news, airline flights, business intelligence information, or financials, for example. Your users would benefit from continuous monitoring of that information.

Why

This is a familiar and recognizable page style. Dashboards have a long history, both online and in the physical world, and people have well-established expectations about how they work: they show useful information, they update themselves, they usually use graphics to display data, and so on.

A dashboard uses many interlocking patterns and components. Many online dashboards use a collection of these patterns found elsewhere in this book:

- *Titled Sections*
- Tabs and *Collapsible Panels*
- *Movable Panels*
- *One-Window Drilldown*
- Lists and tables of various kinds (Chapter 7)
- Information graphics (Chapter 9)
- *Datatips*

How

Determine what information users need or want to see, or what tasks they need to stay on top of. This isn't as simple as it sounds, because you need a researcher's or editor's eye—eliminate confusing or unimportant data, or people won't be able to pick out the parts that matter. Remove, or at least deemphasize, information that doesn't help the user. Feature the most important information or next steps.

Use a good visual hierarchy (Chapter 4) to arrange lists, tables, and information graphics on the page. Try to keep the main information on one page, with little or no scrolling, so people can keep the window on-screen and see everything at a glance. Group related data into *Titled Sections*, and use tabs only when you're confident that users won't need to see the tab contents side by side.

Use *One-Window Drilldown* to let users see additional details about the data—they should be able to click links or graphics to find out more. *Datatips* work well to show individual data points when the pointer rolls over an information graphic.

Choose appropriate and well-designed information graphics for the data you need to show. Gauges, dials, pie charts, and 3D bar charts look nice, but they are rarely the best way to show comparative information at a glance—simple line and bar charts express data better, especially time-based data. When numbers and text are more relevant than graphics, use lists and tables. Row Striping is a common style for multicolumn data tables.

People will try to get actionable information from the dashboard at a glance, without looking closely at every element on the page. So, when you show text, consider highlighting keywords and numbers so that they stand out from surrounding text.

Should your users be able to customize their dashboard displays? Many dashboards do offer customization, and your users might expect it. One way to customize a dashboard page is to rearrange the sections—Salesforce offers *Movable Panels* to users, in addition to choosing which gadgets are shown.

Examples

Salesforce (Figure 2-34) has built a huge software business around answering the need for enterprises large and small to monitor and manage all forms of business processes. Custom-built and customizable dashboards are a central part of this strategy. Some examples of purpose-built dashboards are shown here. Users can build or configure modules according to their needs, and then arrange the standard-sized modules into a grid that suits them best. These can be saved and shared.

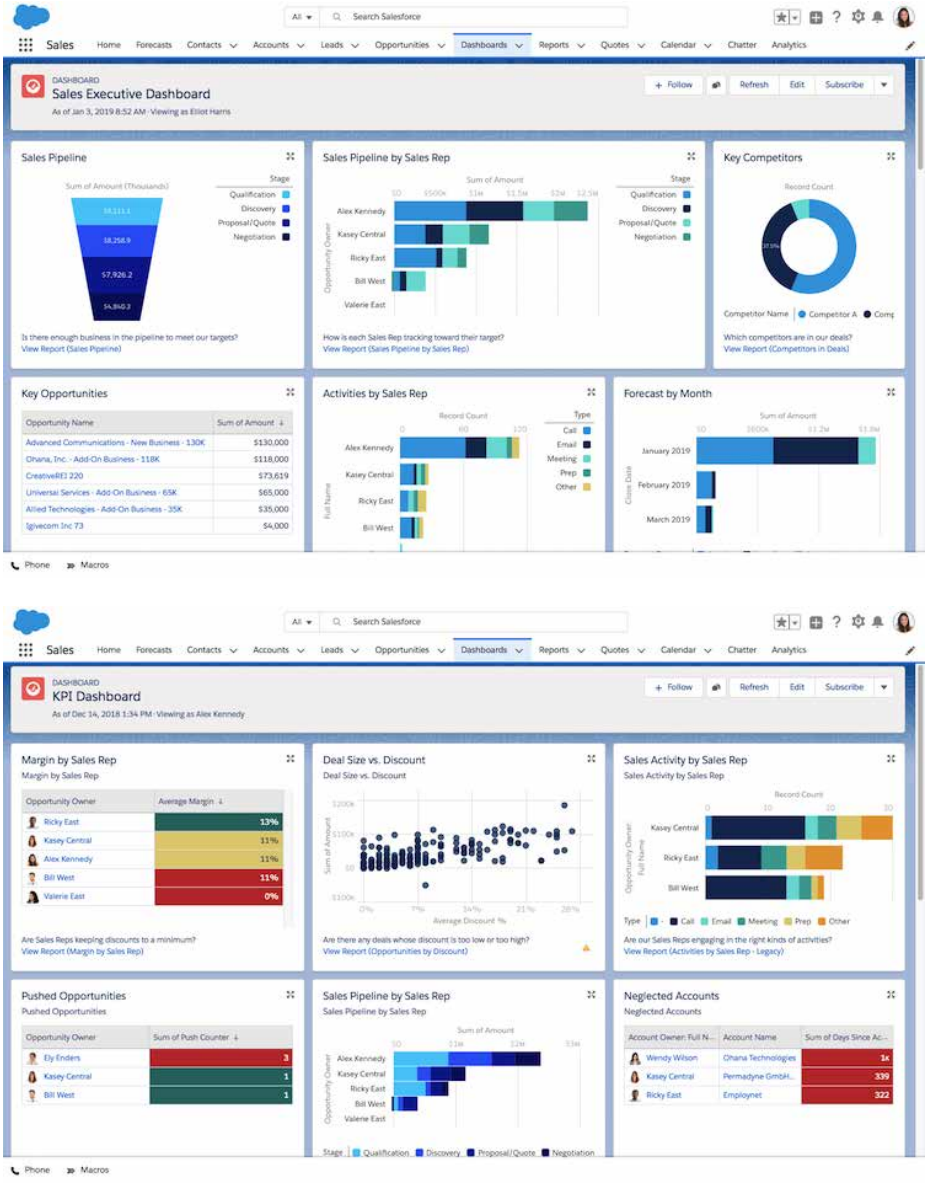


Figure 2-34. *Salesforce Dashboards*

SpaceIQ (Figure 2-35) offers a dashboard on login. This startup follows a classic design pattern of offering key performance indicators, an overview of what is happening now or needs attention, and quick navigation to other key parts of the platform.

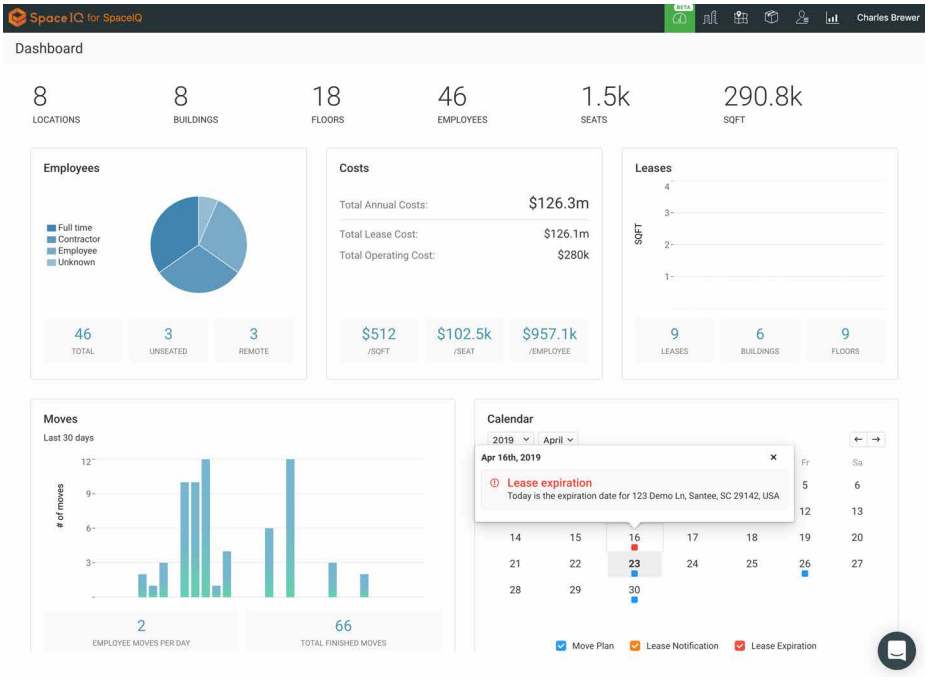


Figure 2-35. *SpaceIQ*

Finally, you might be interested in Stephen Few's book on information dashboards, *Information Dashboard Design: Displaying Data for At-a-Glance Monitoring* (Analytics Press, 2013).

Canvas Plus Palette

What

An application structure defined by a central workspace with containers of tools around it. It consists of a large blank area, or canvas, where a user creates or edits a digital object. Arranged around this open area, at the sides or top or bottom, are grids of tools, called palettes. The tools are represented as icons. The user clicks the palette buttons to create objects on the canvas, or to select a tool to modify objects. The overall effect is a digital workbench or virtual artist's easel. The user selects one tool after another to use on the main object.

Use when

You're designing any kind of graphical editor. A typical use case involves creating new objects and arranging them on some virtual space.

Why

This pair of panels—a palette with which to create things, and a canvas on which to put them—is so common that almost every user of desktop software has seen it. It's a natural mapping from familiar physical objects to the virtual on-screen world. And the palette takes advantage of visual recognition: the most common icons (paintbrush, hand, magnifying glass, etc.) are reused over and over again in different applications, with the same meaning each time.

How

Present a large empty area to the user as a canvas. It might be in its own window, as in Photoshop (Figure 2-18), or embedded in a single page with other tools. The user just needs to see the canvas side by side with the palette. Place additional tools—property panels, color swatches, and so on—to the right or bottom of the canvas, in small palette-like windows or panels.

The palette itself should be a grid of iconic buttons. They can have text in them if the icons are too cryptic; some GUI-builder palettes list the names of GUI components alongside their icons, for instance.

Place the palette to the left or top of the canvas. It can be divided into subgroups, and you might want to use *Module Tabs* or *Collapsible Panels* to present those subgroups.

Most palette buttons should create the pictured object on the canvas. But many builders have successfully integrated other things such as zoom mode and lassoing into the palette.

The gestures used to create items on a palette vary from one application to another. Some use drag-and-drop only; some use a single click on the palette and a single click on the canvas; and some use pressure-sensitive tools, such as a digital pen or simply varying your finger pressure on a touch-sensitive screen and other carefully designed gestures. Usability testing in this area is particularly important because the behaviors of the tools might not be obvious or might be difficult to learn.

Examples

The examples that follow show a variety of canvas and palette patterns. Adobe Photoshop (Figure 2-36) is a classic. All four edges of the app feature palettes, panels and commands that the author uses when working on an image. In Axure RP Pro (Figure 2-37), there are palettes to the left and right of the central drawing pane, where the user creates interactive wireframes for software prototypes. Next is OmniGraffle (Figure 2-38), a vector drawing application for MacOS, which has its tools palette on the left. Even mobile apps have need of this pattern when dealing with large numbers of tools or choices. In iOS Photos (Figure 2-39), the image that the user is editing appears in the central canvas. Below this are three tools palettes that are open-close panels, with multiple editing tools inside each.

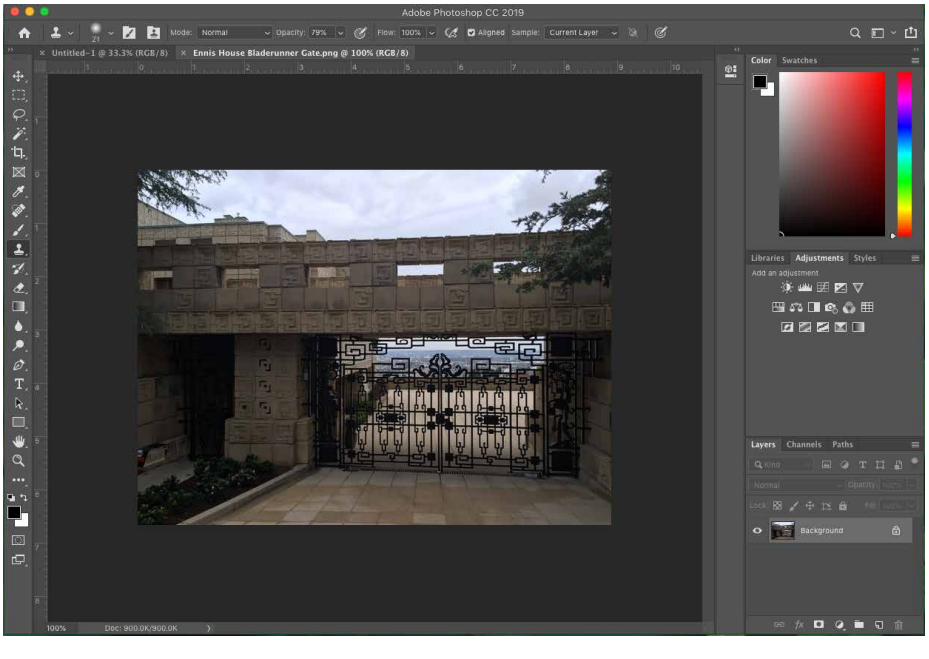


Figure 2-36. *Adobe Photoshop*

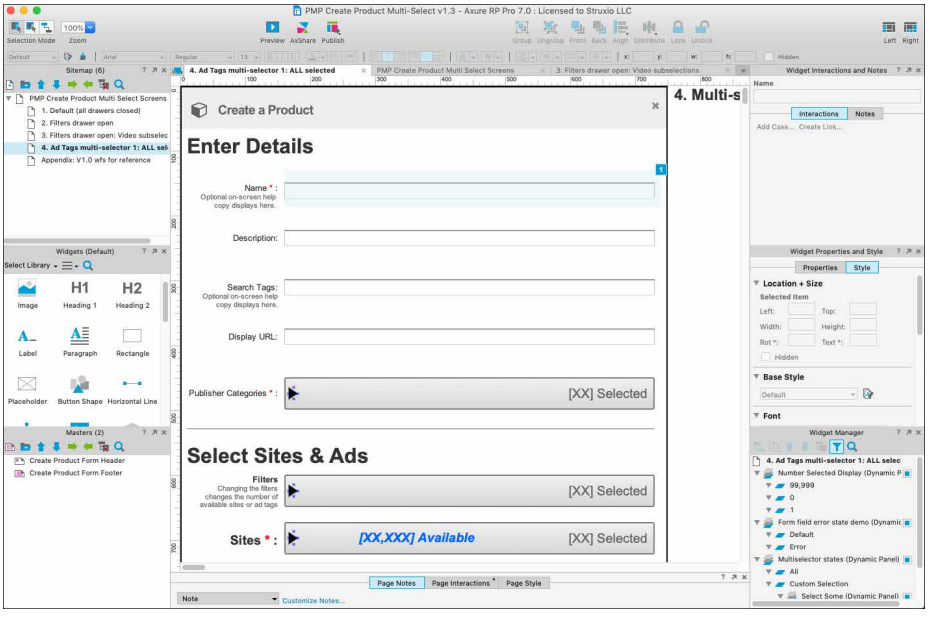


Figure 2-37. *Axure RP Pro*

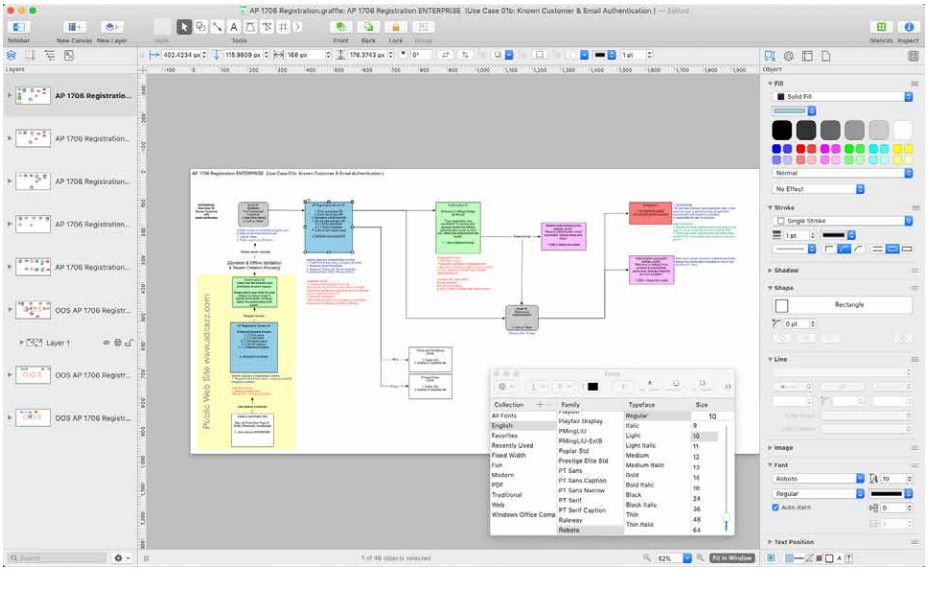


Figure 2-38. *OmniGraffle*

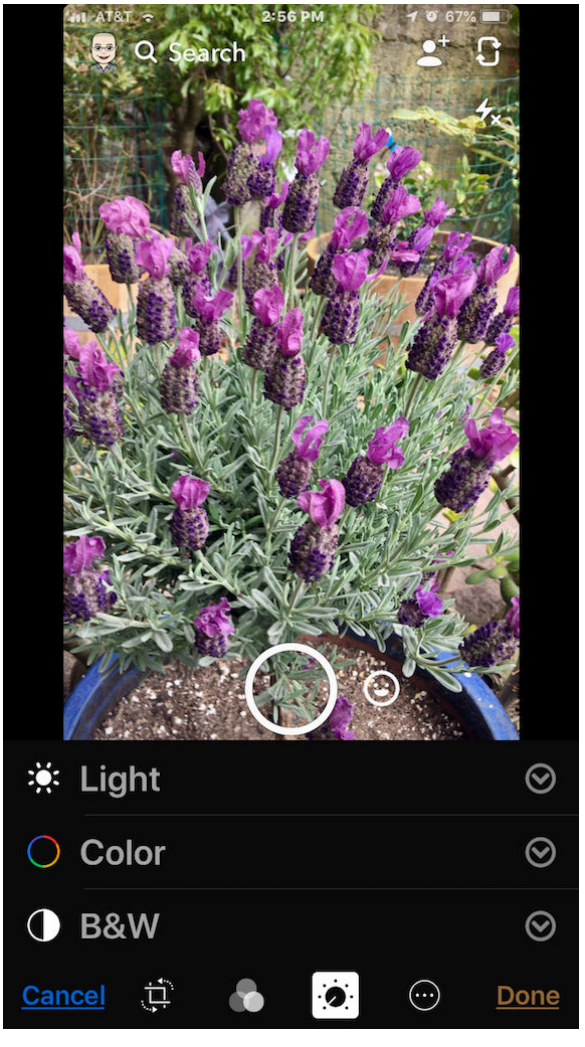


Figure 2-39. *iOS Photos*

Wizard

What

A feature or component that leads the user through the interface step by step to do tasks in a prescribed order.

Use when

You are designing a UI for a task that is long or complicated, and that will usually be novel for users—not something that they do often or want much fine-grained control over (such as the installation of a software package). You're reasonably certain that the designer of the UI will know more than the user does about how best to get the task done.

Tasks that seem well suited for this approach tend to be either branched or very long and tedious—they consist of a series of user-made decisions that affect downstream choices.

The catch is that the user must be willing to surrender control over what happens and when. In many contexts, that works out fine, because making decisions is an unwelcome burden for people doing certain things: "Don't make me think, just tell me what to do next." Think about moving through an unfamiliar airport—it's often easier to follow a series of signs than it is to figure out the airport's overall structure. You don't get to learn much about how the airport is designed, but you don't care about that.

But in other contexts, it backfires. Expert users often find a *Wizard* frustratingly rigid and limiting. This is particularly true for software that supports creative processes such as writing, art, or coding. It's also true for users who actually do want to learn the software; *Wizard* doesn't show users what their actions really do or what application state gets changed as choices are made. That can be infuriating to some people.

Why

Divide and conquer. By splitting up the task into a sequence of chunks, each of which can be dealt with in a discrete "mental space" by the user, you effectively simplify the task. You have put together a preplanned road map through the task, thus sparing the user the effort of figuring out the task's structure—all they need to do is address each step in turn, trusting that if they follow the instructions, things will turn out OK.

But the very need for a Wizard indicates that a task might be too complicated. If you can simplify a task to the point where a short form or a few button clicks can do the trick instead, that's a better solution. (Keep in mind, too, that a Wizard approach is considered a bit patronizing.)

"Chunking" the task. Break up the operations constituting the task into a series of chunks, or groups of operations. You might need to present these groups in a strict sequence, or not; sometimes there is value in breaking up a task into steps 1, 2, 3, and 4 just for convenience.

A thematic breakdown for an online purchase can include screens for product selection, payment information, a billing address, and a shipping address. The presentation order doesn't much matter because later choices don't depend on earlier choices. Putting related choices together just simplifies things for people filling out those forms.

You might decide to split up the task at decision points so that choices made by the user can change the downstream steps dynamically. In a software installation *Wizard*, for example, the user might choose to install optional packages that require yet more choices; if they choose not to do a custom installation, those steps are skipped. Dynamic UIs are good at presenting branched tasks such as this because the user never needs to see anything that's irrelevant to the choices they made.

In either case, the difficult part of designing this kind of UI is striking a balance between the sizes of the chunks and the number of them. It's silly to have a two-step *Wizard*, whereas one comprising 15 steps is tedious. On the other hand, each chunk shouldn't be overwhelmingly large, or you've lost some benefits of this pattern.

Physical structure. *Wizard*s that present each step in a separate page, usually navigated with Back and Next buttons, are the most obvious and well-known implementation of this pattern. They're not always the right choice, though, because now each step is an isolated UI space that shows no context—the user can't see what went before or what comes next. But an advantage of such a *Wizard* is that they can devote each page to that step completely, including illustrations and explanations.

If you do this, allow the user to move back and forth at will through the task sequence. Offer a way for the user to step backward or to otherwise change their mind about an earlier choice. Additionally, many UIs show a selectable map or overview of all the steps, getting some of the benefits of a *Two-Panel Selector*. (In contrast to that pattern, a *Wizard* implies a prescribed order—even if it's merely suggested—as opposed to completely random access.)

If you instead choose to keep all the steps on one page, you could use one of several patterns from Chapter 4:

- *Titled Sections*, with prominent numbers in the titles. This is most useful for tasks that aren't heavily branched because all steps can be visible at once.
- *Responsive Enabling*, in which all the steps are present on the page, but each one remains disabled until the user has finished the previous step.
- *Progressive Disclosure*, in which you wait to show a step on the UI until the user finishes the previous one. This might be the most elegant way to implement a short *Wizard*. It's dynamic, compact, and easy to use.

Good Defaults and Smart Prefills (from Chapter 10) are useful no matter how you arrange the steps. If the user is willing to turn over control of the process to you, the odds are good they're also willing to let you pick reasonable defaults for choices they might not care much about, such as the location of a software installation.

Examples

The Microsoft Office designers have done away with many of its *Wizard*s, but a few remain—and for good reason. Importing data into Excel is a potentially bewildering task. The Import Wizard (Figure 2-40) is an old-school, traditional application *Wizard* that guides the user step by step through the import process. It uses Back/Next buttons, branching, and no sequence map, but it works. Each screen lets you focus on the step at hand, without worrying about what comes next.

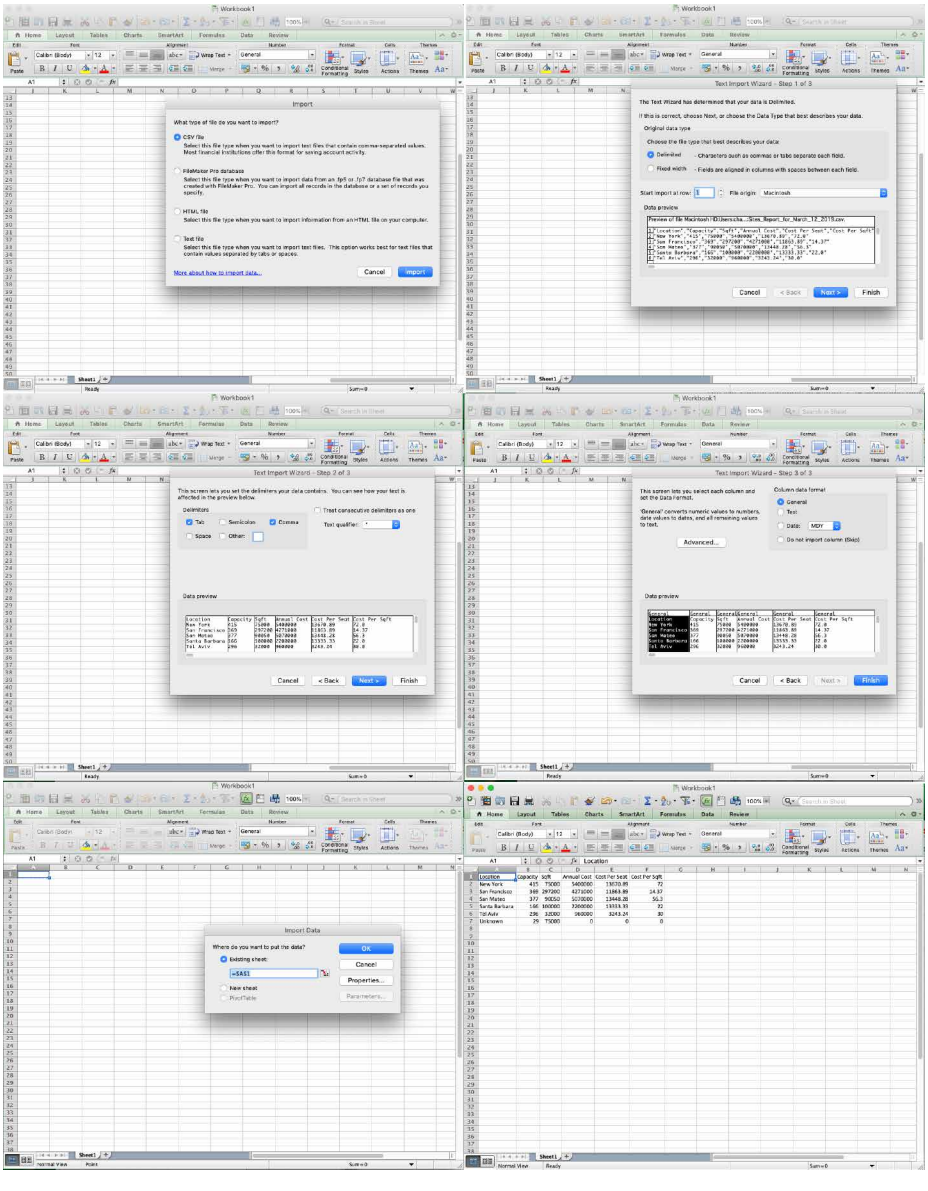


Figure 2-40. *The Microsoft Excel import wizard*

Settings Editor

What

An easy-to-find, self-contained page or window where users can change settings, preferences, or properties. Divide the content into separate tabs or pages if you need to manage large numbers of settings.

Use when

You are designing any of the following applications or tools, or something similar:

- An application that has app-wide preferences.
- An operating system (OS), mobile device, or platform that has system-wide preferences.
- A site or app for which a user must sign in—users will need to edit their accounts and profiles.
- An open-ended tool to create documents or other complex work products. Users might need to change a document's properties, an object within a document, or another item.
- A product configurator, which allows people to customize a product online.

Why

Though both use forms, a *Settings Editor* is distinct from a *Wizard*, and it has very particular requirements. A user must be able to find and edit a desired property without being forced to walk through a prescribed sequence of steps—random access is important.

To aid findability, the properties should be grouped into categories that are well labeled and make immediate sense.

Another important aspect of *Settings Editor* design is that people will use it for viewing existing settings, not just changing them. The design needs to communicate the values of those settings at a glance.

Experienced users have strong expectations for preference editors, account settings, and user profiles being in familiar places and behaving in familiar ways. Break these expectations at your own peril!

How

First, make it findable. Most platforms, both mobile and desktop, have a standard place to find application-wide preferences—follow the conventions, and don't try to

be overly clever. Likewise, websites where people sign in usually put links to account settings and profiles where the username is shown, often in the upper-right or upper-left corner.

Second, group the properties into pages, and give those pages names that make it easy to guess what's on them. (Sometimes all the properties or settings fit on one page, but not often.) Card-sorting exercises with representative users can help you figure out the categories and their names. An outrageously large number of properties might require a three- or four-level hierarchy of groups, but be careful that users don't get frustrated at having to click 53 times to reach commonly needed properties.

Third, decide how to present these pages. Tabs, *Two-Panel Selector*, and *One-Window Drilldown* (Chapter 7) with an extensive page "menu" on the top page seem to be the most common layouts for *Settings Editor*.

The design of the forms themselves deserves an entire chapter. See Chapter 10 for patterns and techniques used in forms.

Finally, should you immediately apply changes that the user makes or offer Save and Cancel buttons? That can depend on the type of settings you're working with. Platform-wide settings seem to be applied immediately when changed; settings on websites mostly use Save buttons; and application settings and preferences can go either way. It might not be a huge usability issue in any case. Follow an established convention if there is one, or see what the underlying technology requires; test it with users if you still have open questions.

Examples

Google (Figure 2-41) and Facebook (Figure 2-42) both use tabs to present the pages of their profile editors. Amazon has one single link for all account-related information: Your Account (see Figure 2-43). This *Menu Page* (Chapter 3) lists account settings alongside order information, credit card management, digital content, and even community and wish-list activity. The clean, tight page organization is terrific—if I have any questions about what's going on with my relationship to Amazon, I know I can find it somewhere on this page.

Google, Facebook, and Amazon have huge settings, preferences, and configuration management issues related to their services. Customers must access these settings from time to time in order to review or change them. All have opted for a strong organization system to categorize their settings and preferences. Google and Facebook use tabs to organize the settings into major categories, with screens in each that are in turn sectioned out with titles and groups of controls to allow for comprehension and relatively easy access. Amazon places its most frequently used settings and configurations at the top of the settings screen, with special formatting as giant buttons. Selecting one allows the user to drill down into the appropriate category of

settings. Below this is a grid of cards, each labeled with its settings category name, and displaying a list of links to each subcategory within. All three use strong information architecture and navigation to bring some understandable structure to a complicated part of their platform. Although it's not painless, the user has a good chance of eventually finding and changing the setting they seek.

Amazon (Figure 2-43) offers an outrageously large number of properties that require a deep hierarchy of pages. The designers mitigated some of the problems, however. For instance, they put a list of shortcuts on the top-level page; these are probably the items users look for most often. They put a search box on the top. And by using lists of items, they show users which items fall into which categories.

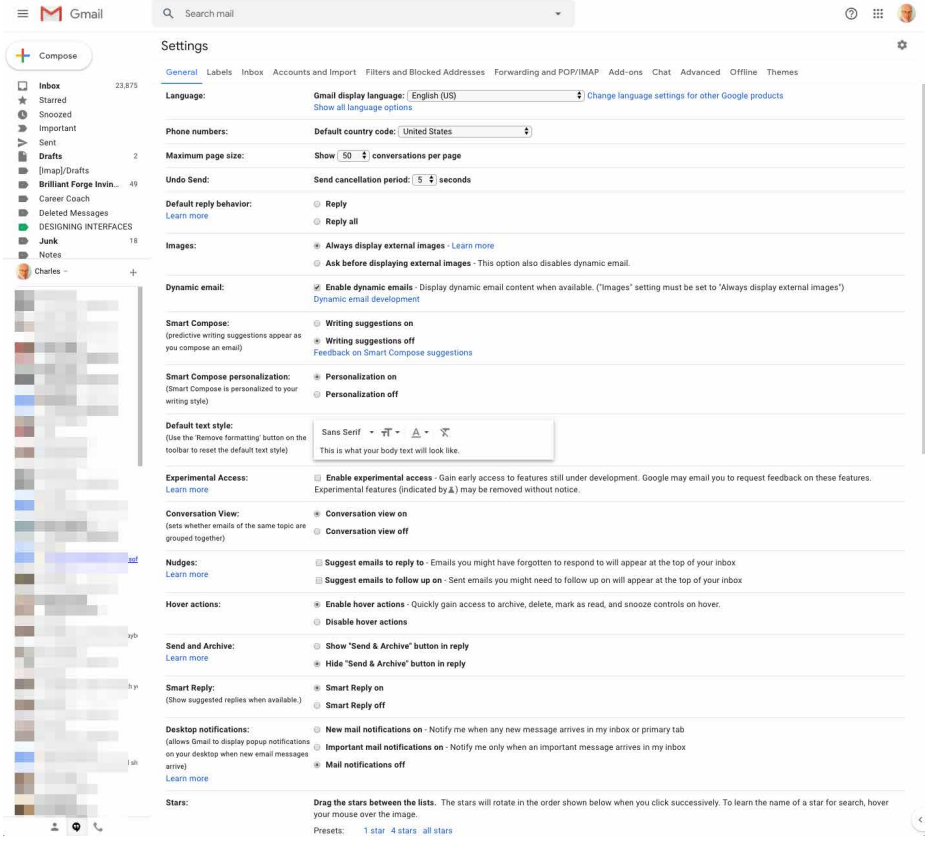


Figure 2-41. *Google*

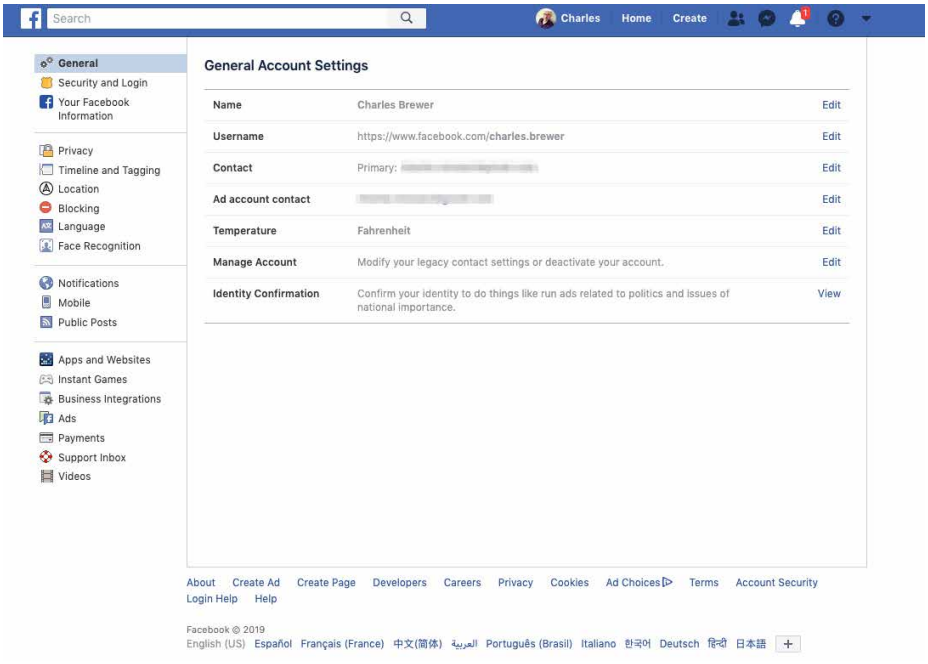


Figure 2-42. *Facebook*

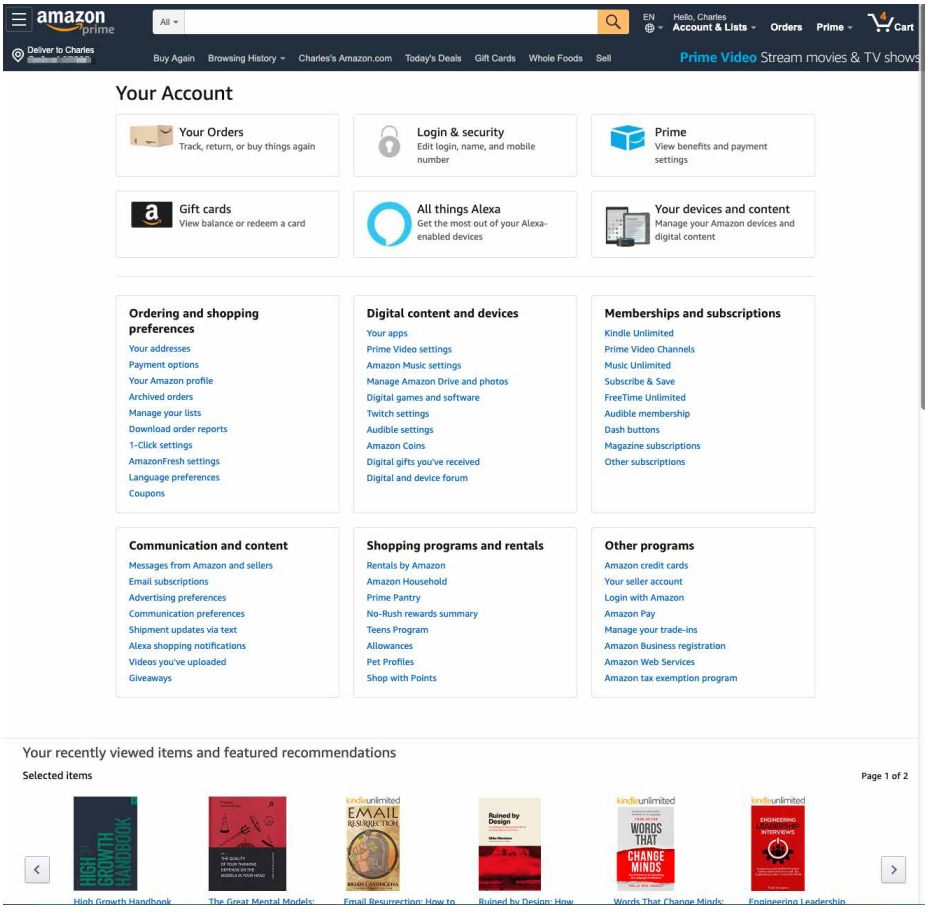


Figure 2-43. *Amazon*

In Apple's mobile OS, iOS, there are many settings (Figure 2-44). Some are for the entire device, and some are for individual apps on the iPhone. Apple has opted for a single scrolling list. Some order is provided by putting the most critical and frequently used settings at the top. The items in the long list are also grouped to help with navigation and selection.

For its desktop OS, macOS, Apple has opted for a panel of categories for its system settings (Figure 2-45). These are marked by sections, icons, and labels to help with understanding the categories and selecting the right one.

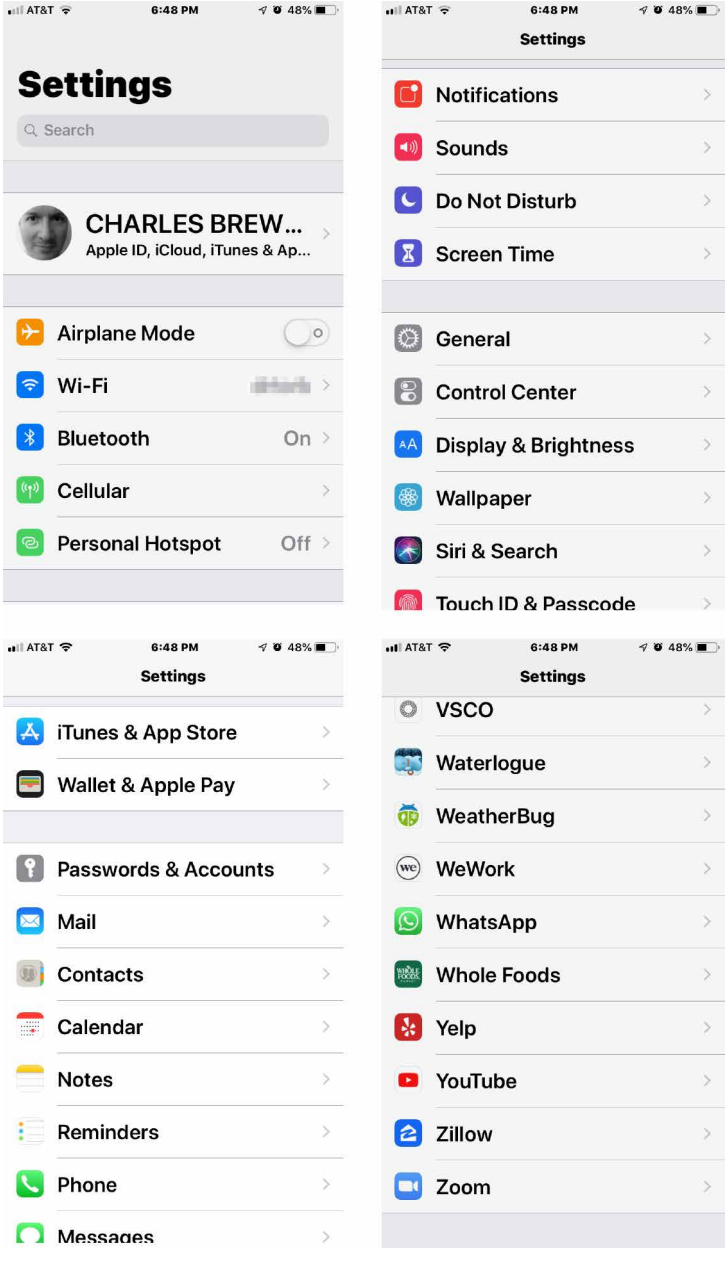


Figure 2-44. *iOS settings*

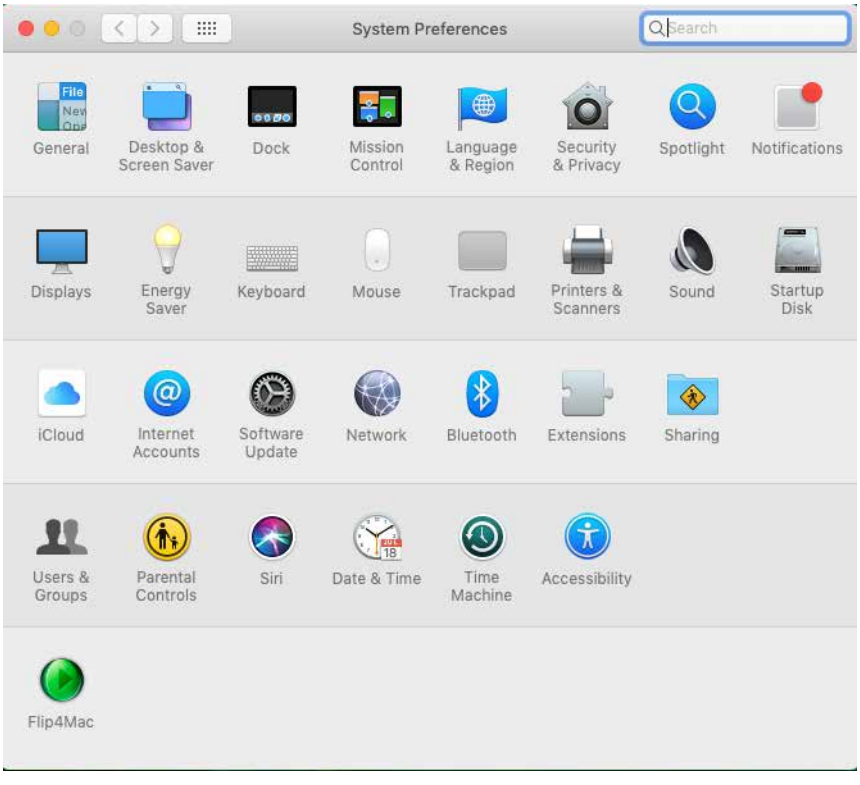


Figure 2-45. *macOS system preferences*

Alternative Views

What

Views or methods of visualizing information in your software or app that are substantially different from one another but offer access to the same information.

Use when

You're building something that views or edits a complex document, list, website, map, or other content. You might face design requirements that directly conflict with one another. You can't find a way to show both feature set A and feature set B at the same time, so you need to design both separately and let the user choose between them.

Why

Try as you might, you can't always accommodate all possible usage scenarios in a single design. For instance, printing is typically problematic for websites because the information display requirements differ—navigation and interactive gizmos should be removed, for instance, and the remaining content reformatted to fit the printer paper.

There are several other reasons for *Alternative Views*:

- Users have preferences with regard to speed, visual style, and other factors.
- A user might need to temporarily view data through a different "lens" or perspective in order to gain insight into a problem. Consider a map user switching between views of street information and topographic information (see Figure 2-31 at the top of the pattern).
- If a user is editing a slideshow or website, for instance, they might do most of their editing while using a "structural" view of the document, containing editing handles, markers for invisible content, layout guides, private notes, and so on. But sometimes they will want to see the work as an end user would see it.

How

Choose a few usage scenarios that cannot easily be served by the application's or site's normal mode of operation. Design specialized views for those scenarios, and present them as alternatives within the same window or screen.

In these alternative views, some information might be added and some might be taken away, but the core content should remain more or less the same. A common way to switch views is to change the rendering of a list; file finders in both Windows

and macOS let users switch from lists to *Thumbnail Grid* to *Tree Table* to *Carousel*, for instance.

If you need to strip down the interface—for use by a printer or screen reader, for instance—consider removing secondary content, shrinking or eliminating images, and cutting out all navigation but the most basic.

Put a "switch" for the mode somewhere on the main interface. It doesn't need to be prominent; PowerPoint and Word used to put their mode buttons in the lower-left corner, which is an easily overlooked spot on any interface. Most applications represent the alternative views with icons. Make sure it's easy to switch back to the default view, too. As the user switches back and forth, preserve all of the application's current state—selections, the user's location in the document, uncommitted changes, undo/redo operations, and so on because losing them will surprise the user.

Applications that "remember" their users often retain the user's alternative-view choice from one use to the next. In other words, if a user decides to switch to an alternative view, the application will just use that view by default next time. Websites can do this by using cookies; desktop applications can keep track of preferences per user; an app on a mobile device can simply remember what view it used the last time it was invoked. Web pages can have the option of implementing *Alternative Views* as alternative CSS pages. This is how some sites switch between ordinary pages and print-only pages, for example.

Examples

Let's look at an effective use of different "modes" or alternative views of data in two different applications. The pattern is to offer search results on a map, giving a spatial or geographic representation, and then to show the same results as a scrollable list, which can be sorted and filtered more easily. One mode is for an overview, and geographic context (closer to or further from me). The other is for reading detailed information about each item.

Yelp, the local business directory platform, is the first example. On iOS, the Yelp mobile app (Figure 2-46) offers the two aforementioned views. The searcher must toggle between the views on the small screen format. The Yelp desktop app (Figure 2-47) has the room to offer the map and list views side by side. This offers more robust exploration and learning interactions, such as highlighting the same venue in the list and on the map.

Zillow, the real estate platform, follows a similar design. On iOS, Zillow offers a map view and a list view of your query results (Figure 2-48). The user must toggle between the views. On the Zillow desktop app (Figure 2-49), the home or rental searcher can view these modes side by side. Note the choice to use photographs of the properties in the list view. This supports rapid scanning of the huge numbers of choices so that the searcher can zero in on properties they consider most attractive.

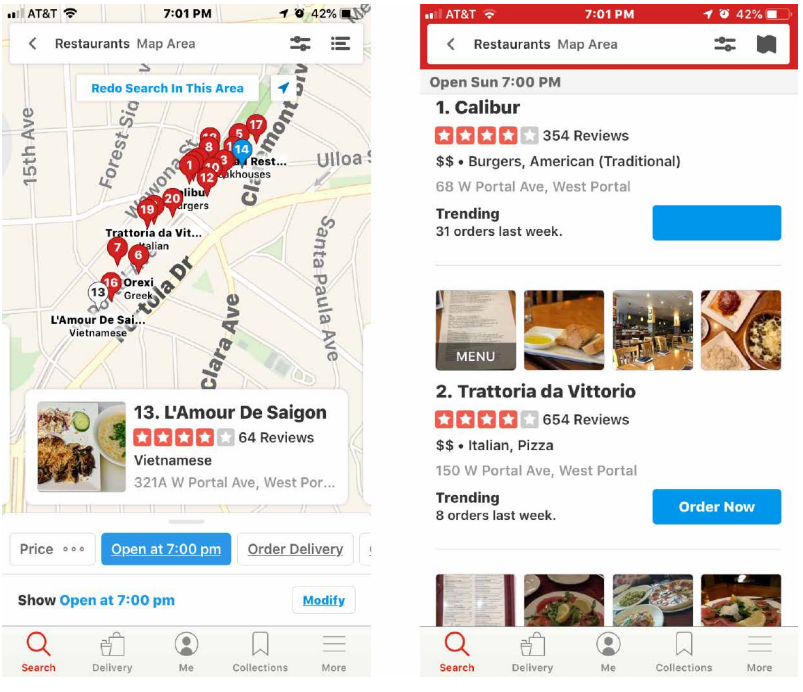


Figure 2-46. *Yelp iOS map and list screens*

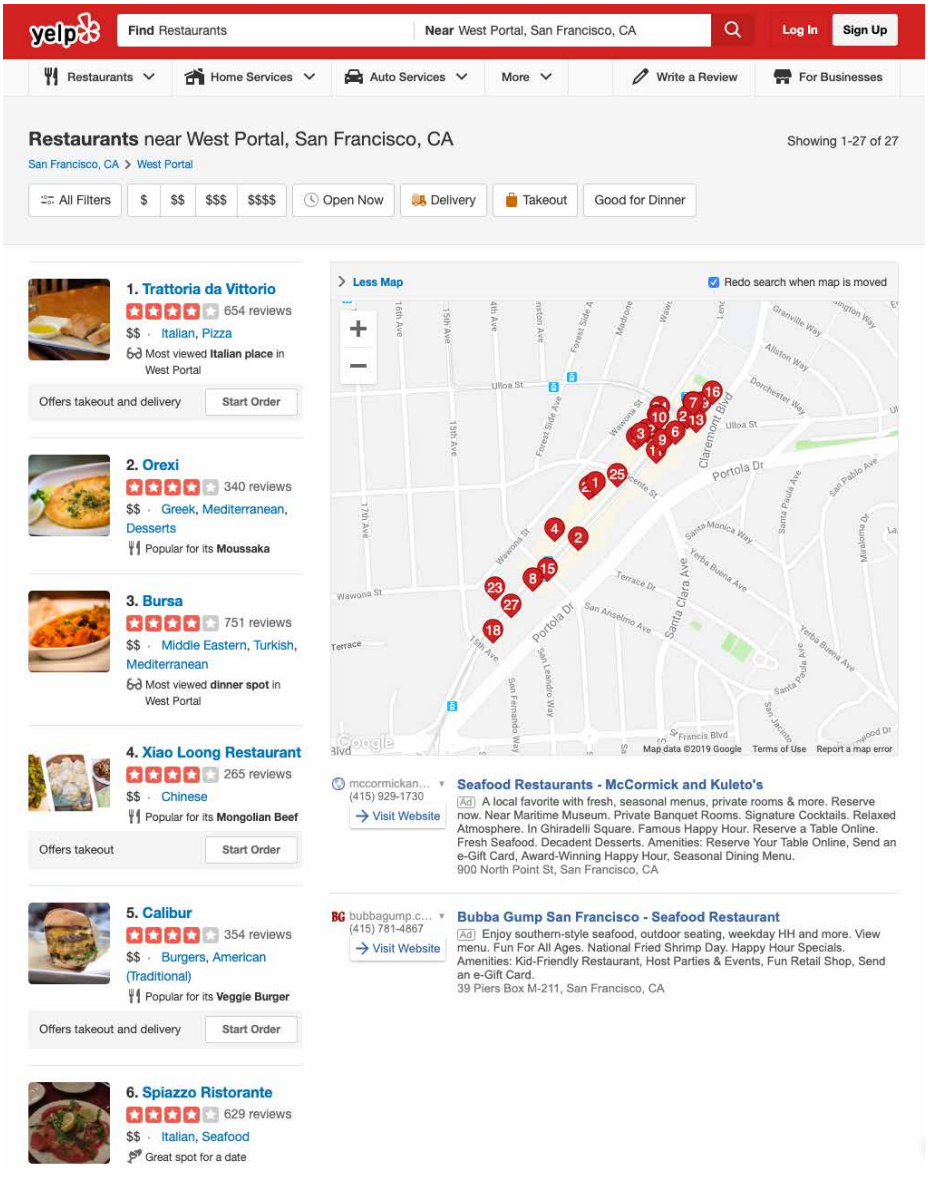


Figure 2-47. *Yelp desktop map and list combination screen*

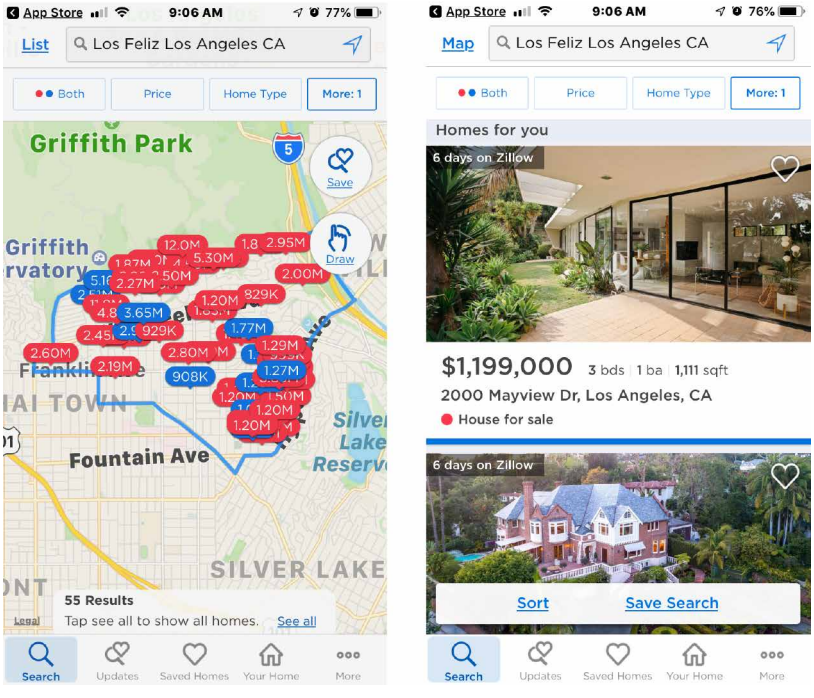


Figure 2-48. *Zillow iOS map and list screens*

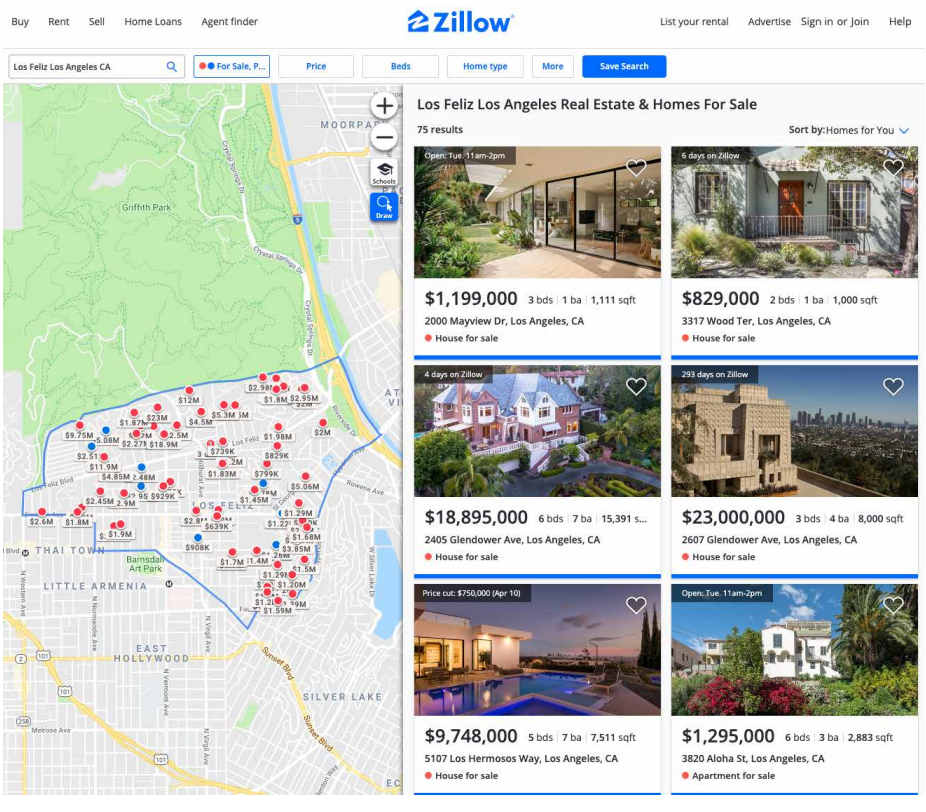


Figure 2-49. *Zillow desktop map and list combination screen*

Two graphic editors, Apple Keynote (Figure 2-50) and Adobe Illustrator (Figure 2-51), show different views of a work product. In the slideshow, the user normally edits one slide at a time, along with its notes, but sometimes the user needs to see all the slides laid out on a virtual table. (Not shown is a third view, in which Keynote takes over the screen and actually plays the slideshow.)

Adobe Illustrator shows an "outline" view of the graphic objects in the document—most useful if you have a lot of complex and layered objects—and the normal, fully rendered view of the artwork. This mode is explicitly for performance reasons. The outline view puts much less demand on the computer processor and so speeds up work considerably.

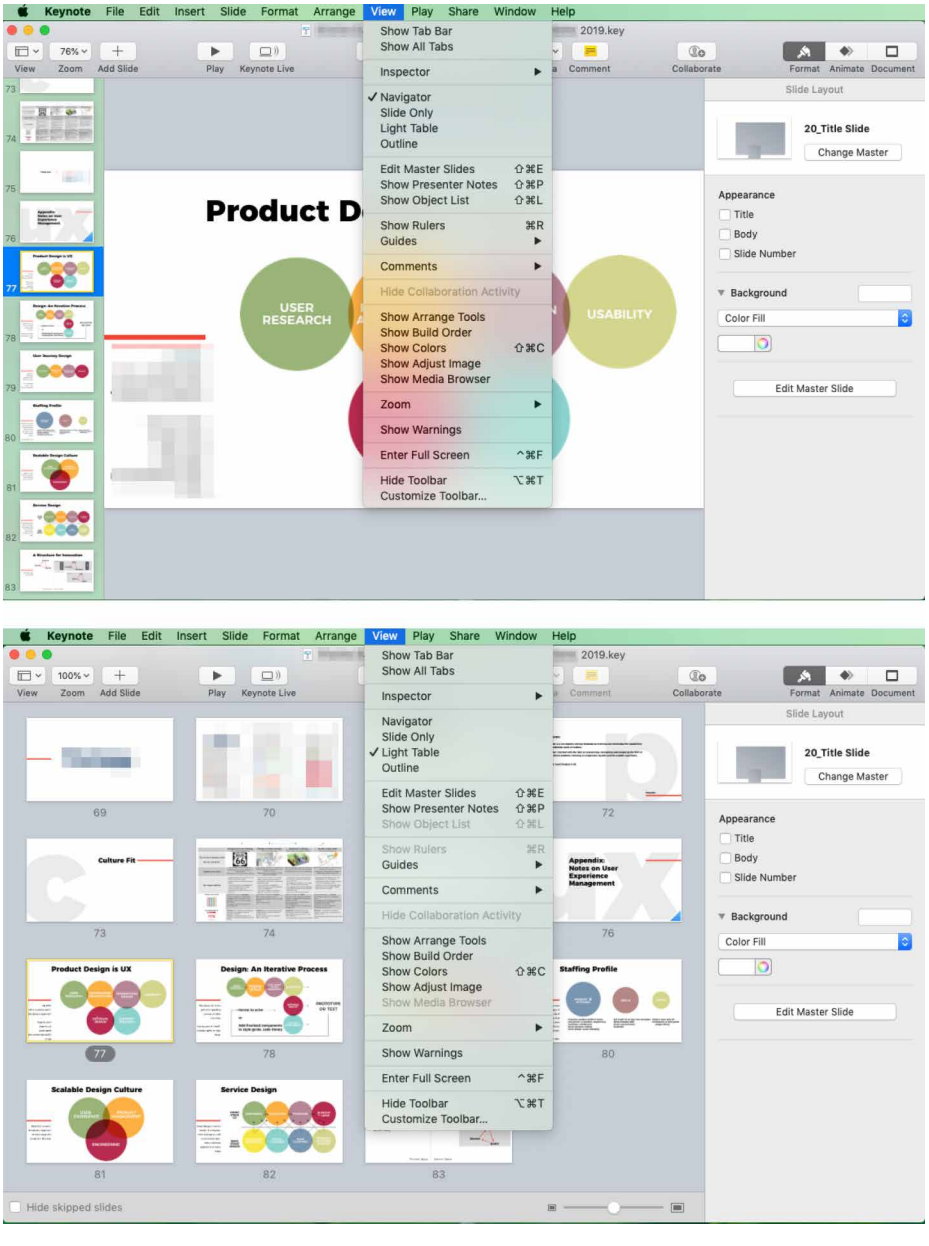


Figure 2-50. *Apple Keynote*

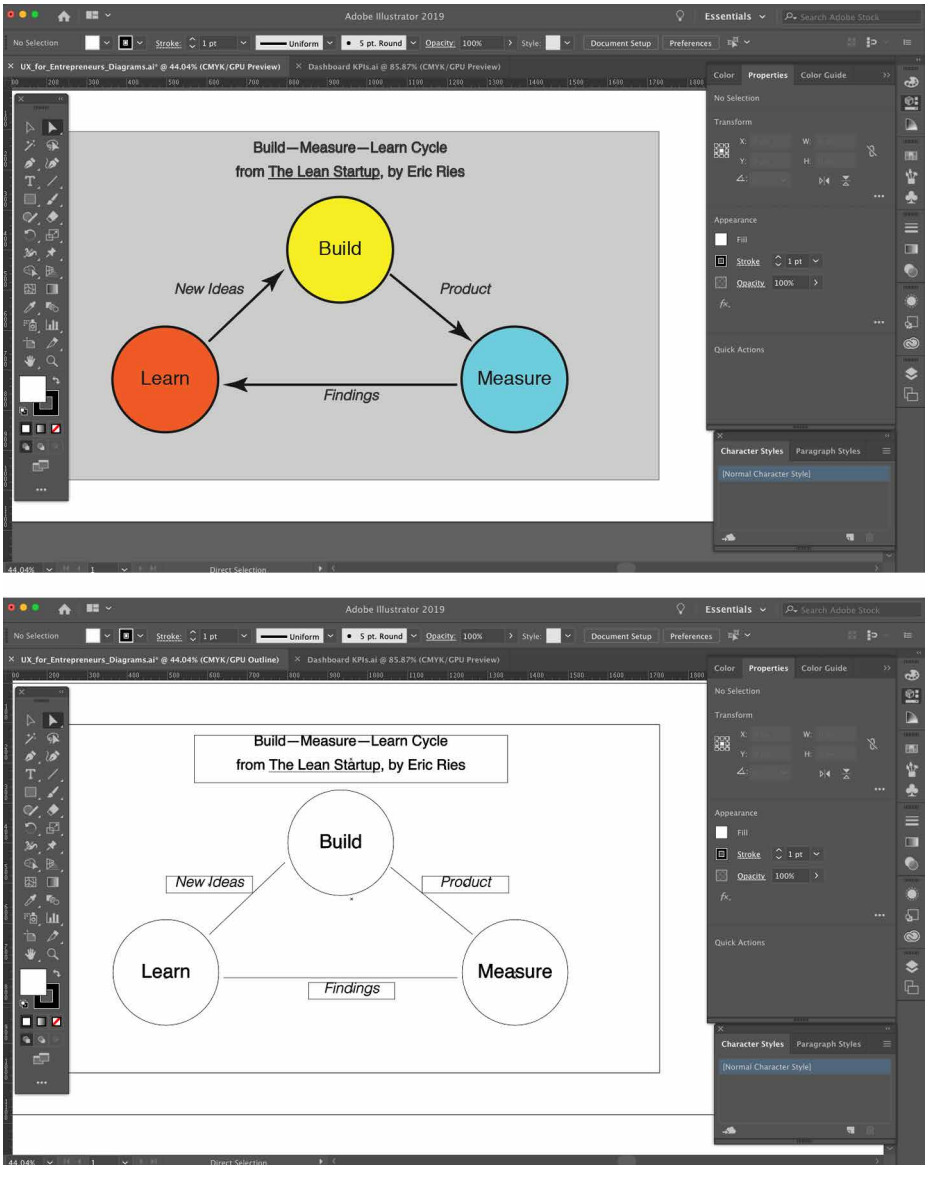


Figure 2-51. *Adobe Illustrator*

Many Workspaces

What

An interface where users can view more than one page, project, file, or context at a time. It can consist of multiple top-level tabs, tab groups, streams/feeds, panels, or windows. Users might have the option to place these workspaces side by side.

Use when

You're building an application that views or edits any type of content—websites, documents, images, or entire projects that include many files. A major aspect of choosing this pattern is the need to have different views or task "modes" available at the same time. For example, people often keep many browser tabs open at the same time so that they can switch between various websites, or compare them. Application developers and media creators often need to see and adjust code or controls in an editor window, and at the same time see the output of their work to see whether they are getting the desired outcomes—either as compiled and executed code, or as a rendered media object.

Designers of conventional websites don't generally need to think about this. All of the major browsers supply perfectly good implementations of this pattern, using tabs and browser windows. Spreadsheet applications such as from Microsoft or Google offer tabbed workspaces to separate a complicated workbook into individual calculation sheets.

Applications whose central organizing structure is a personal news stream might not need *Many Workspaces*, either. Email clients, personal Facebook pages, and so forth show only the one news stream that matters to the user; multiple windows don't add much value. That being said, email clients often let a user launch multiple email messages in different windows. Some Twitter applications can show several filtered streams side by side; for instance, they might show a search-based feed, then a feed from a custom list, and then a feed of popular retweets. (See the TweetDeck example in Figure 2-52.)

Why

People sometimes need to switch rapidly between different tasks in the same project or file, or monitor activity across a large number of real-time feeds.

People multitask. They go off on tangents, abandon trains of thought, stop working on task A to switch to task B, and eventually come back to something they left hanging. You might as well support it directly with a well-designed interface for multitasking.

Side-by-side comparisons between two or more items can help people learn and gain insight. Let users pull up pages or documents side-by-side without having to laboriously switch context from one to another.

This pattern directly supports some Chapter 1 patterns, such as *Prospective Memory* (a user might leave a window open as a self-reminder to finish something) and *Safe Exploration* (because there's no cost in opening up an additional workspace while leaving the original one where it is).

How

Choose one or more ways to show multiple workspaces. Many well-known applications use the following:

- Tabs
- Separate OS windows
- Columns or panels within a window
- Split windows, with the ability to adjust the splitters interactively

If you deal with fairly simple content in each workspace—such as text files, lists, or *Streams and Feeds*—split windows or panels work fine. More complex content might warrant entire tab pages or windows of their own so that a user can see a larger area at once.

The most complicated cases involve development environments for entire coding projects. When a project is open, a user might be looking at several code files, stylesheets, command windows (where compilers and other tools are run), output or logfiles, or visual editors. This means that many, many windows or panels can be open at once.

When users close some web browsers, such as Chrome, the set of workspaces (all open web pages, in tabs and windows) are automatically saved for later use. Then, when the user restarts the browser, their entire set of previously opened web pages is restored, almost as they left it. This is especially nice when the browser or machine has crashed. Consider designing in this feature; it would be a kindness to your users.

Examples

Both TweetDeck (Figure 2-52) and Hootsuite (Figure 2-53) take a multipanel or multistream approach to managing social media feeds.

TweetDeck is a *Streams and Feeds*–type application that can show many streams at once: filtered Twitter feeds, non-Twitter sources, and so on. The example in Figure 2-52 shows several typical TweetDeck columns. This maintains the spirit of a news stream by keeping all the updates visible at once; had these columns been in different tabs or windows, a user wouldn't be able to see all the updates as they happen.

TweetDeck by default allows the user to see multiple streams side by side from within their own account. In the example in Figure 2-52, the user can see their main feed, notifications, and their messages at the same time. Normally, these are hidden behind various navigation tabs and can be viewed only one at a time. Note also that the fourth panel displays a Twitter-curated list of trending hashtags. TweetDeck supports many feed panels open at the same time, which is useful for monitoring other Twitter accounts at the same time.

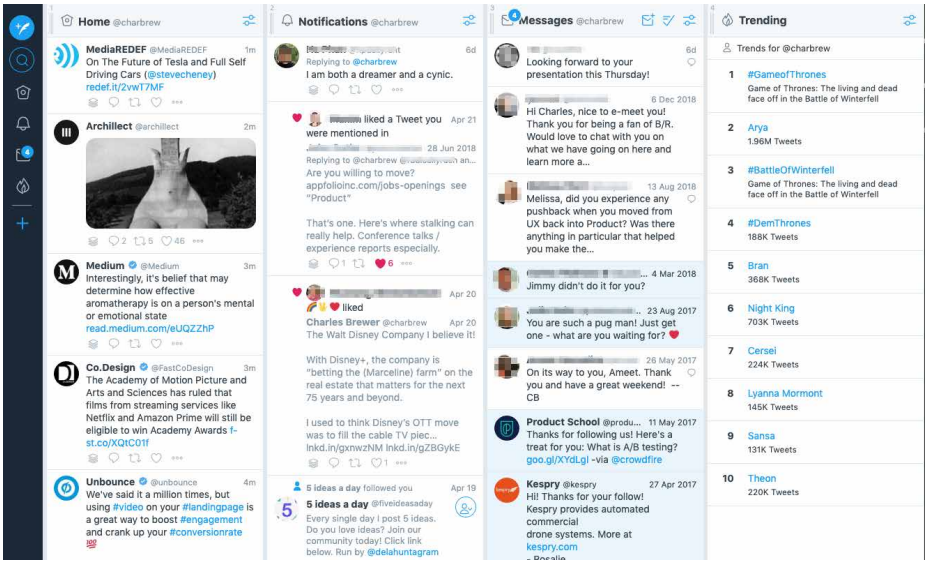


Figure 2-52. *Twitter TweetDeck*

Hootsuite is a social media postings management platform. It's valuable for individuals, businesses, and publishers who want to manage and coordinate their social media accounts in one place. This is useful for pushing out new content or increasing follower and reader interaction across their entire social media ecosystem. In this example (Figure 2-53), the Hootsuite user has set up their Twitter and LinkedIn accounts. With this side-by-side feed view, the user can keep track of activity in both accounts (and many more). Posting and responding to each separate account can be carried out from one multipanel Hootsuite view.

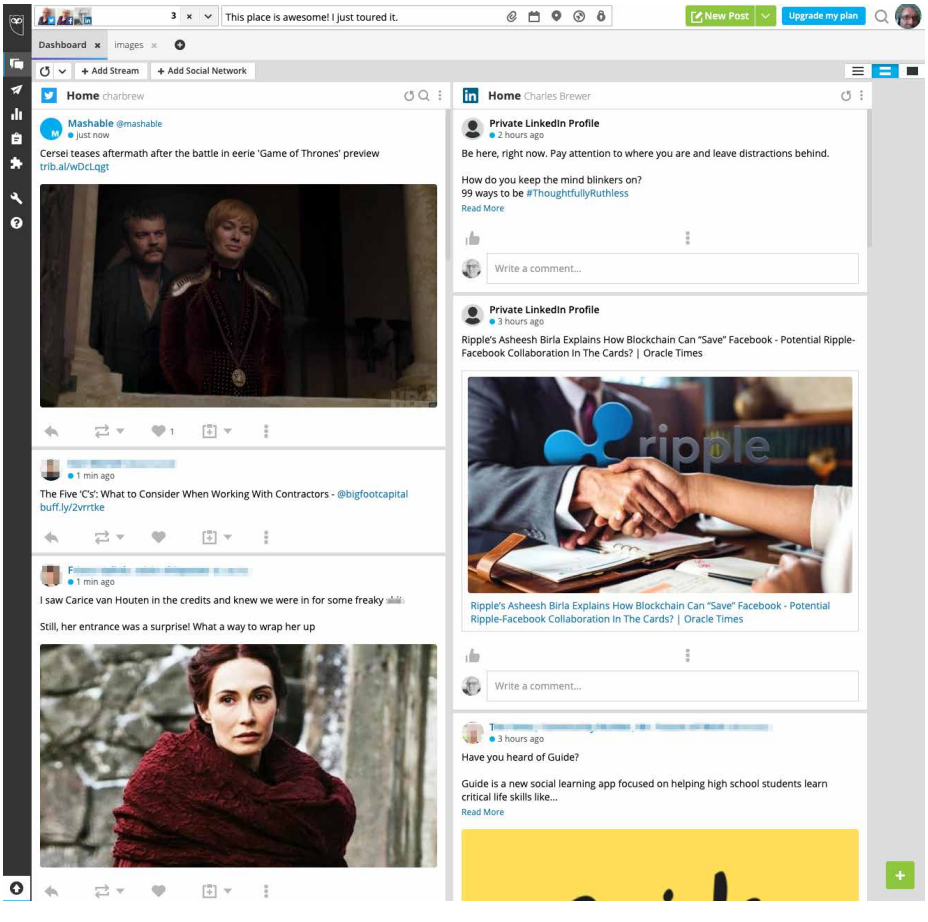


Figure 2-53. *Hootsuite*

Cinema 4D (Figure 2-54) is a tool for creating, rendering, and animating three-dimensional objects. This desktop application uses a multipanel approach to give the user simultaneous views of multiple information and tool spaces. There are panels that show source files and versions of the current work file; a central panel that shows the current 3D object, palettes, toolbars, and panels that have all the controls for working on the 3D object; and a timeline tool that controls the animation of the 3D object.

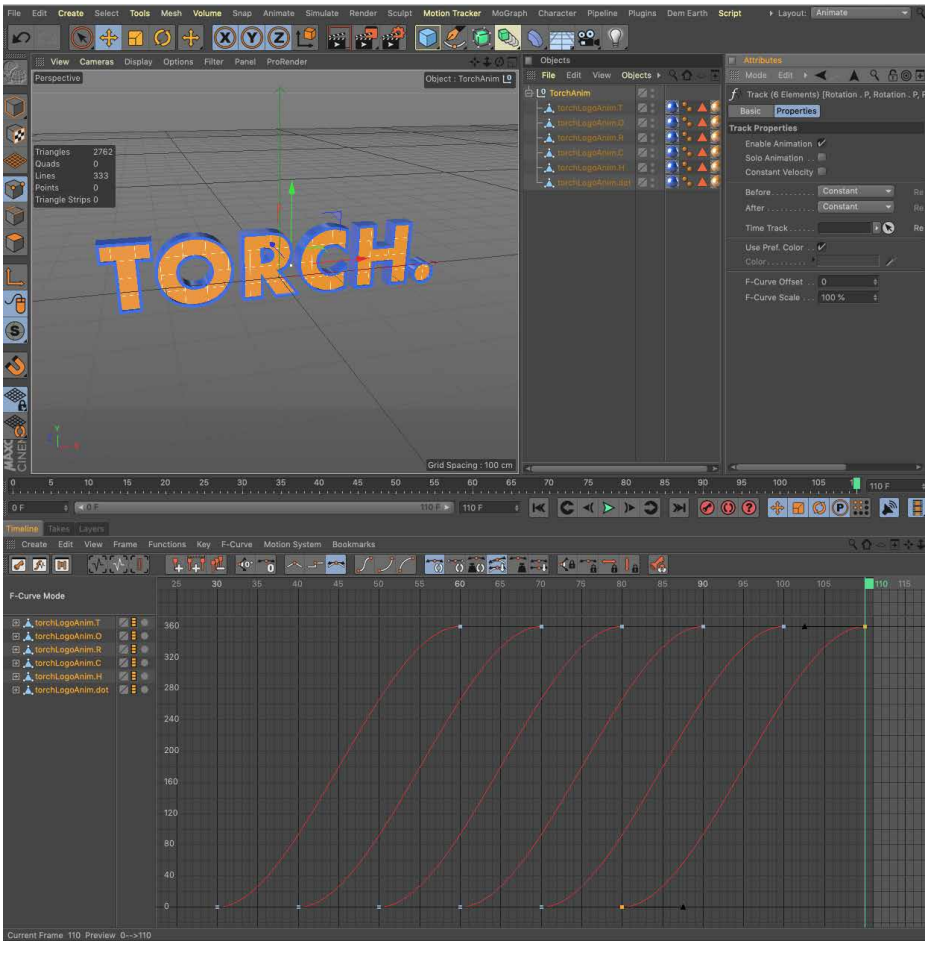


Figure 2-54. *Cinema 4D*

Help Systems

What

Providing labels, explanations, and descriptions of how to use the interface are a fundamental part of making usable software. The goal is to provide assistance, answers, or training to users when they need it—in multiple forms so the it can be accessed in different situations.

Inline/display. Helping your users in an immediate way starts with the copy that displays on the screen by default. The purpose of inline copy is to communicate what the user is looking at and what the purpose of a given section or component is. Additionally, examples of the inputs you might be soliciting from users also helps prevent misformatting. Consider a mix of the following:

- Meaningful headlines and subheaders
- Instructions: a phrase or sentence directly on the screen to help with a particularly tricky interface.
- Labels for form elements
- Prompts or example inputs either in or next to form elements

Tool tips. Tool tips are brief descriptions or explanations of each component on the screen. On desktop web apps, these display when the user hovers over the interface component. Another method is to display a question mark or other icon (or a link) next to a specific component. Tapping or clicking the icon displays a short explanation.

Full help system. This is a fully written out user guide that covers all the major features and functions of your app. This is most common for desktop applications. The help system can include descriptions, glossaries, FAQs, how-to's, videos, and other information. Help systems often reinforce or replicate user training materials, especially for complicated apps. Help systems can be embedded in the app itself, or can be hosted on a separate website.

Guided tours. It is now common to deploy step-by-step guided tours or walkthroughs within your application. Many companies now offer this capability. It usually takes the form of a lightbox or other layer on top of the application itself. These display as a series of pop ups or pointers that take the user through a tour or help them complete a process in a stepwise fashion. These guided tours can be triggered by a variety of events: first time in the app, from the user selecting a "show me how" help option, or by more advanced user behavior analytics and behaviors.

Knowledge base. Many contemporary customer success software platforms include a Quora-style knowledge base. This consists of a database of questions and answers that are built up over time by the users of the system. This can be created or used just by a customer support team. However, it is now common to open up the knowledge base to customers, as well. Often this knowledge base offers the ability to submit a question or topic and see a list of most related questions and answers created by previous users. Knowledge bases are often the core of modern app help systems, and are the first form of customer support–self-support.

Online community. Software that is popular enough or specialized enough to have a significant user base can also be supported by an online community of users. This is an advanced technique where the goal is to build a long-term community of users who will help one another, spread usage, and create a culture that fosters platform growth. Sometimes, these communities form on their own, as groups on social media platforms or forums like LinkedIn, Facebook, or Reddit. Many companies create, host, and moderate their own online user communities to ensure quality discussions. They also create and moderate official groups and communities on social media sites.

Use a mixture of lightweight and heavyweight help techniques to support users with varying needs.

Use when

Every well-designed website or application should have some form of help. Copy on the screen, prompts in form elements, and tool tips are a must. How will you help beginners become experts? Some users might need a full-fledged help system, but you know most users won't take the time to use it. For complicated applications, full training and help documentation is a must.

Why

Users of almost any software artifact need varying levels of support for the tasks they're trying to accomplish. Someone approaching it for the first time ever (or the first time in a while) needs different support than someone who uses it frequently. Even among first-time users, enormous differences exist in commitment level and learning styles. Some people want to watch a tutorial video, some won't; most find tool tips helpful, but a few find them irritating.

Help texts that are provided on many levels at once—even when they don't look like traditional "help systems"—reach everyone who needs them. Many good help techniques put the help texts within easy reach, but not directly in the user's face all the time, so users don't become irritated. However, the techniques need to be familiar to your users. If they don't notice or open a *Collapsible Panels*, for instance, they'll never see what's inside it.

How

A lightweight or simple help approach can include displaying meaningful on-screen copy, such as descriptive headers and on-screen instructions. Rollover tool tips for all controls is another lightweight approach. However, rollovers work only on desktop apps in which the browser can track where the user's mouse pointer is, thus triggering tool tips based on location. Mobile can offer tool tips, as well, but these must be tapped on to access them (there is no hover concept in the mobile interaction design world). This also means any hover tools that you have implemented in a web app must be accessed via a tap action, icon, or menu on mobile.

When the user wants to learn in more detail or as a task in itself, a user guide or online manual is the way to go. Sometimes, this is in your application itself. Other times, it is a separate website or system. This is a more heavyweight technique because it involves creating more content. The benefit is that this help system information has a long life cycle–it provides value for a long time and doesn't need continuous updating. Users can refer to it again and again, and updates need happen only periodically.

You can think of help systems as existing on multiple levels of detail. Some are very close to the user and their task and are meant to maintain focus on completing it. Others are separate experiences in and of themselves and create a separate learning environment, for times when the user wants to focus on learning.

Create help on several levels, including some (but not necessarily all) of the help types in the following list. Think of it as a continuum: each requires more effort from the user than the previous one but can supply more detailed and nuanced information:

- Meaningful headings, instructions, examples, and help text directly on the page, including patterns such as *Input Hints* and *Input Prompt* (both found in Chapter 10). At the same time, try to keep the total amount of text on the page low.
- Prompt text in form fields.
- Tool tips. Use them to show very brief, one- or two-line descriptions of interface features that aren't self-evident. For icon-only features, tool tips are critical; users can take even nonsensical icons in stride if a rollover says what the icon does! (Not that I'd recommend poor icon design, of course.) The disadvantages of tool tips are that they hide whatever's under them and that some users don't like them popping up all the time. A short time delay for the mouse hover—for example, one or two seconds—removes the irritation factor for most people.
- *Hover Tools* (Chapter 8). These can display slightly longer descriptions, shown dynamically as the user selects or rolls over certain interface elements. Set aside areas on the page itself for this rather than using a tiny tool tip.
- Longer help texts contained inside *Collapsible Panels* (see Chapter 4).

- Introductory material, such as static introductory screens, guided tours, and videos. When a new user starts the application or service for the first time, these materials can immediately orient them toward their first steps (see the *Instant Gratification* pattern in Chapter 1). Users might also be interested in links to Help resources. Offer a toggle switch to turn off the introduction—users will eventually stop finding it useful—and offer a way back to it elsewhere in the interface in case a user wants to go back and read it later.
- Help shown in a separate window, often in HTML via browsers, but sometimes in WinHelp or Mac Help (a desktop app for help). The help resource is often an online manual—an entire book—reached via menu items on a Help menu, or from Help buttons on dialog boxes and HTML pages.
- Live technical support, usually via chat email, the web, Twitter, or telephone.
- Online community support. This applies only to the most heavily used and invested software—the likes of Photoshop, Linux, Mac OS X, or MATLAB—but users might consider it a highly valuable resource. Host your own community or use social networking resources for these or more traditional online forums.

Examples

In-screen help: labels and tool tips. The first layer of help that can be designed into your software is the help that appears right in the context of usage: labels, rollover highlights, and tool tips. In Adobe Photoshop's desktop application (Figure 2-55), hovering the mouse over a tool icon (which lacks a label) causes it to change background color and then display a tool tip that explains what the highlighted tool does. Learning through exploration is enabled this way.

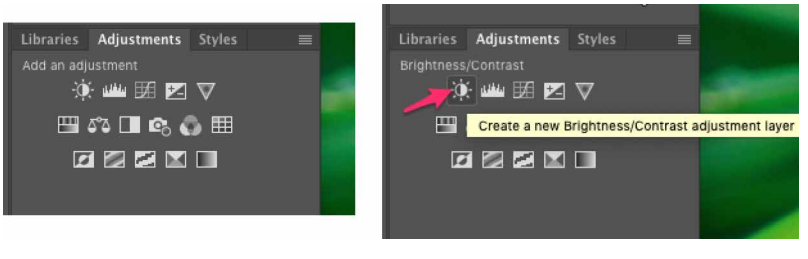


Figure 2-55. *Adobe Photoshop: animated titles as hover tools and tool tips*

Microsoft Excel (Figure 2-56) view controls have permanent labels. The selected tool has a selected appearance and a corresponding label. But if the user hovers over an unselected view button, the label will change to show temporarily what the name of the highlighted (not selected) tool is. Excel also makes extensive use of regular roll-over tool tips, as shown in Figure 2-57, especially for icons that don't have default labels.

Figure 2-56. *Microsoft Excel: animated titles as hover tools*

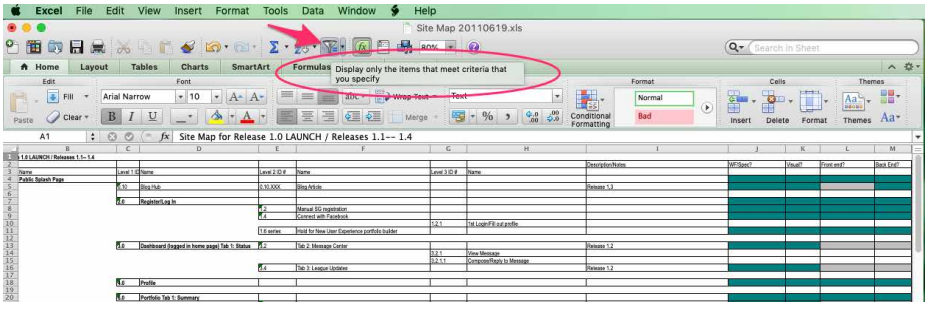


Figure 2-57. *Microsoft Excel: tool tips*

Help systems. Publishing instructional and reference content in the software platform or application itself is a longstanding best practice in usability and customer success. Essentially it means providing a digital version of the user manual for the software that the customer can access as needed. This provides incredible value to users because they have a place to turn to answer their questions that they can access on their own, without having to initiate a customer service request. They also don't need to break from their task at hand to come back later. Also, new users self-train with such materials. From a customer confidence and satisfaction point of view, and considering the savings in reduced customer service requests, deploying a help system in your application (or in a separate website that your software can link to) is a significant benefit.

Adobe Photoshop (Figure 2-58) shows how a designer can access help and tutorials directly in the application. From Photoshop, the user has links to the Adoble Photoshop Help website (Figure 2-59). Microsoft Excel (Figure 2-60) offers the full help system in a separate application window that can be opened while using the app.

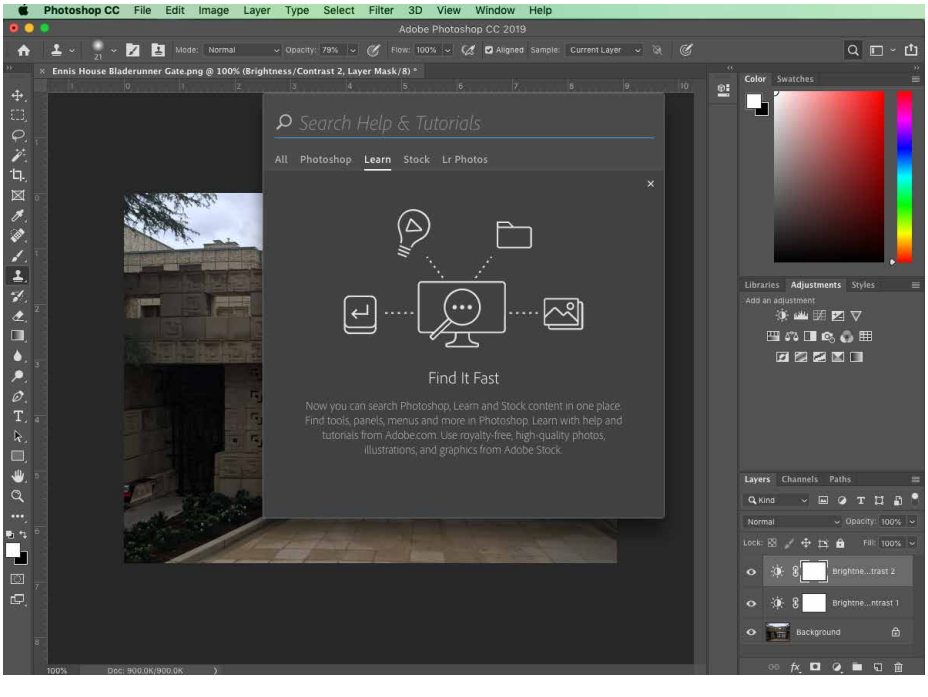


Figure 2-58. *Adobe Photoshop Search and Help*

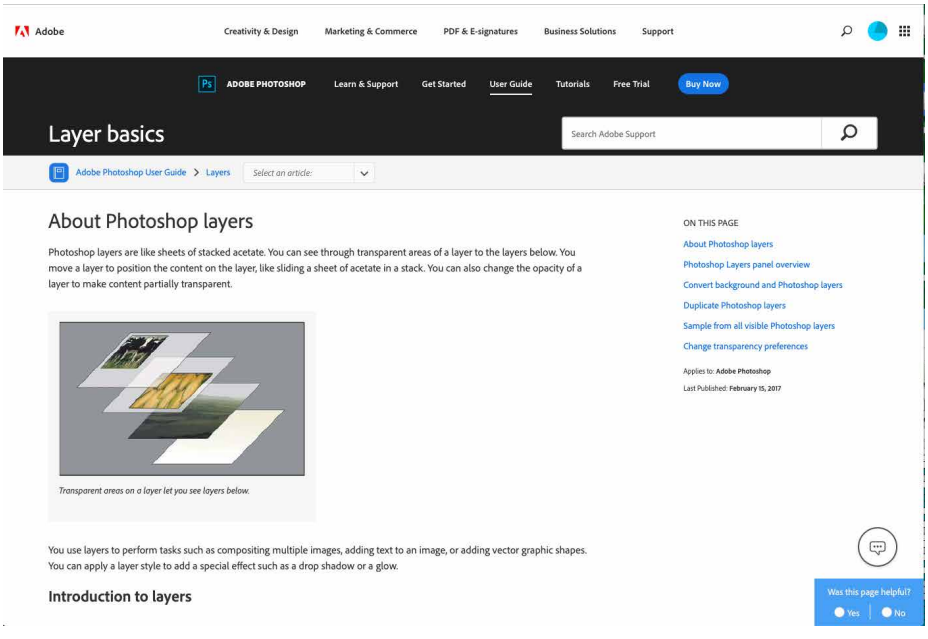


Figure 2-59. *Adobe Photoshop Help (website)*

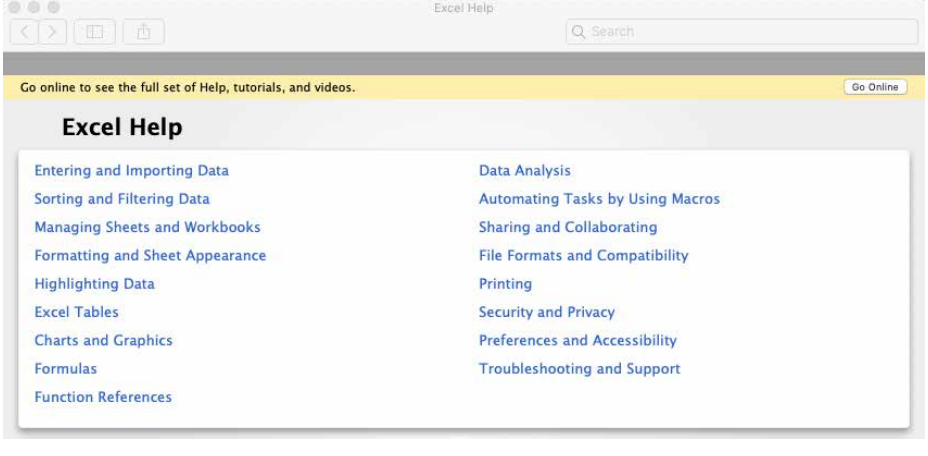


Figure 2-60. *Microsoft Excel Help (in the application itself)*

New user experiences: guided instruction. What is the most natural way in the world for a visitor, a new resident, or a new employee to learn where things are, how the space or campus is laid out, or how to do things? Or, what's the best way for an existing user to get some help in the form of a reminder, a repeat training, or a "refresher"? It's to

ask a more experienced person to give a tour, to be a guide. Many software platforms have taken this approach to onboarding new users. It's possible to create and deploy step-by-step guides for software that gives a new customer or employee a tour of the software or a guide for how to do specific tasks. These often take the form of an overlay on top of the standard interface, or a series of pop ups that point to key screen components. In this way, a new UX can be scripted and developed once and then used by many people after that.

Userlane (Figure 2-61) is a company that provides a user-guide authoring platform that other companies can deploy in their own software. In the example here, using Wikipedia as a demonstration, an instructional panel displays in lightbox mode on top of the regular Wikipedia interface. In this way, the learner is prompted to focus on one step or interface component at a time, which promotes better learning. They can also go at their own pace while still getting a tour of all the most important parts of the software by the end.

Similarly, Pendo (Figure 2-62) offers pop-up user guides as part of its customer engagement platform. The Pendo customer, usually a software company, creates user guides for how to do things in the app. In this example, the learner is in the middle of some task walk-through. The immediate next step, which is to go to Settings, is highlighted at top in an overlay.

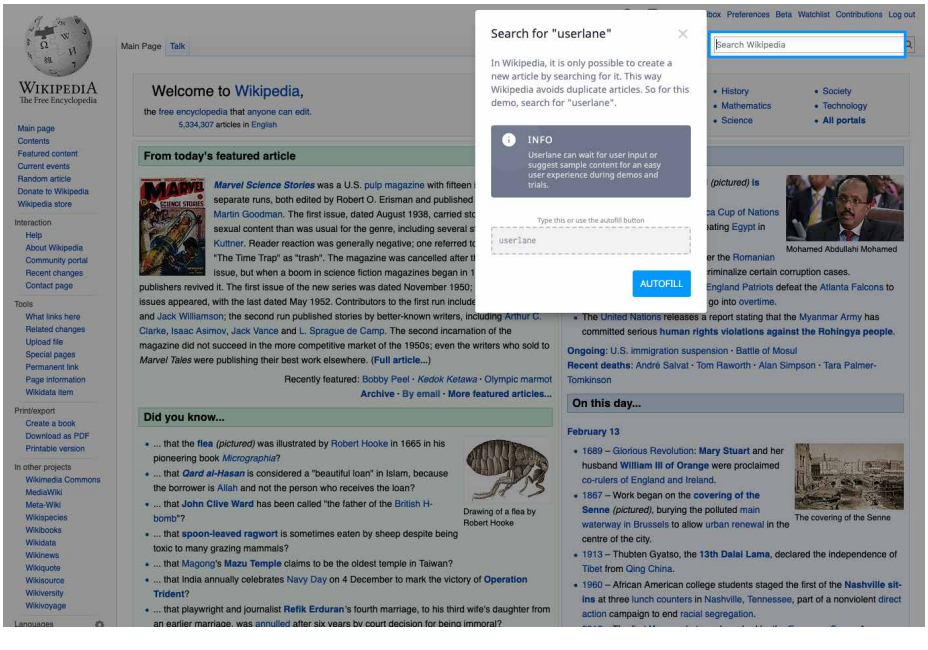


Figure 2-61. *Lightbox mode: instructional panel with highlighted step (Userlane)*

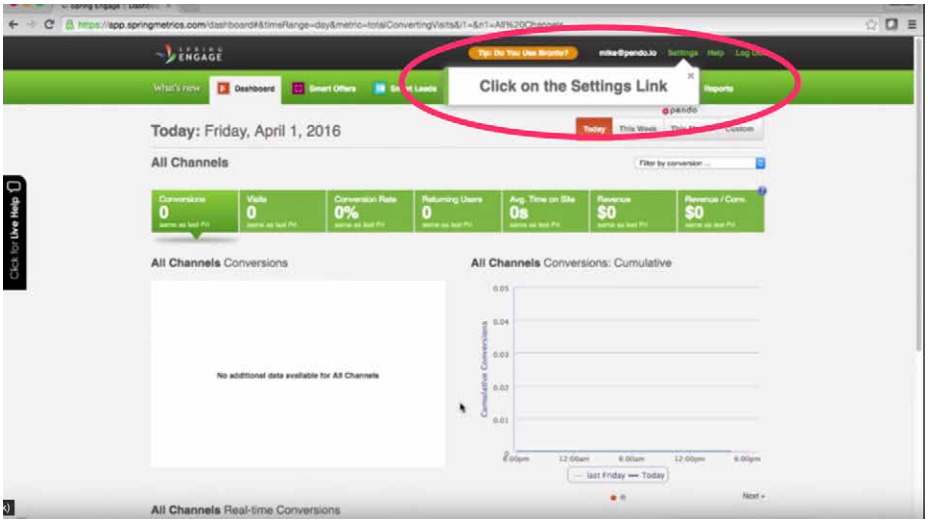


Figure 2-62. *Overlay mode: step-by-step pop ups pointing to UI elements (Pendo)*

Online community. Finally. if all other sources of help are exhausted, a user can turn to the wider user community for advice. We've now moved beyond the realm of software design, per se, but this is still product design—the UX extends beyond the bits installed on users' computers. It includes the interactions they have with the organization, its employees or other representatives, and its website.

The UI design app Sketch is widely used, and it's no surprise that the company wants to bring this community together around this popular tool. The Sketch User community (Figure 2-63) shows how this community is a source of news and learning.

Similarly, Adobe User Forums (Figure 2-64) offer a way for designers and others to discuss issues. Adobe creates the forum. Their customers create the content in the form of discussions, questions and answers, and advice. In this way, knowledge can be spread more easily among members of the community.

Community building like this happens only for products in which users become deeply invested, perhaps because they use the product every day at work or at home. But having an online community of users of some sort is common. It is also a huge advantage for the product. So, it is worth considering how to foster such a community.

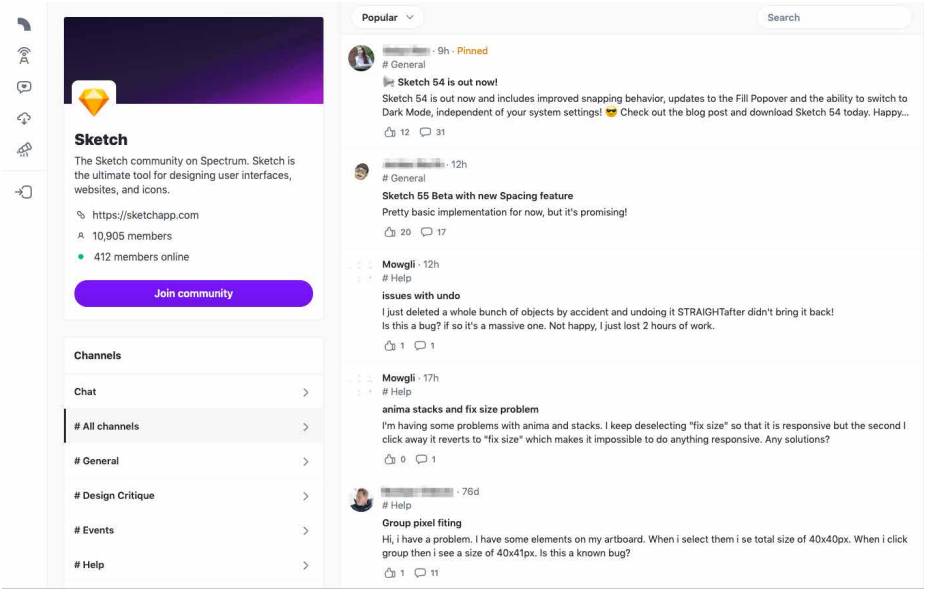


Figure 2-63. *User community as a learning resource; hosted by third party (Sketch community hosted by Spectrum)*

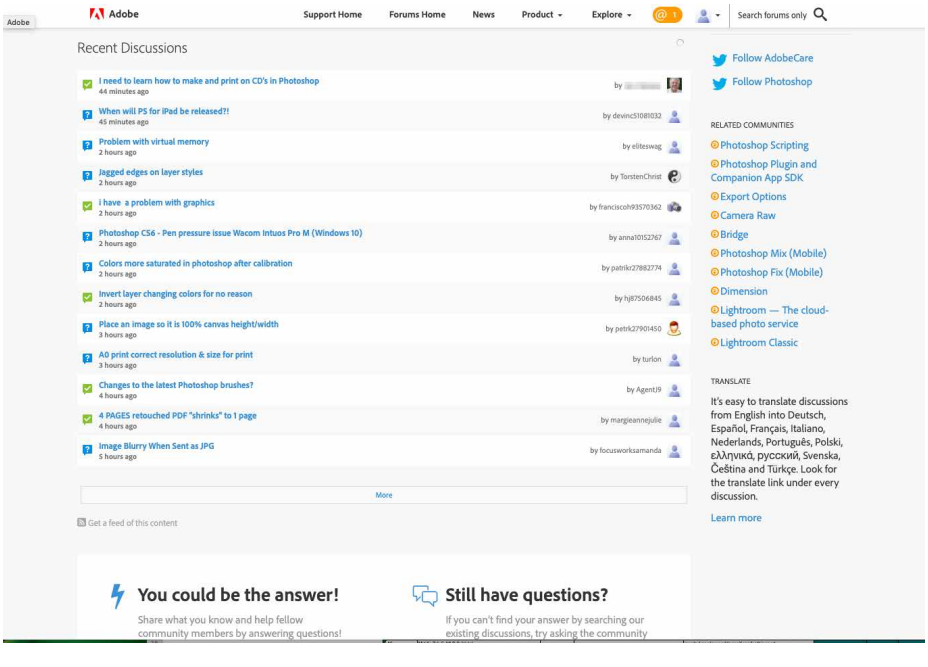


Figure 2-64. *Adobe User Forums*

Tags

What

Tags are a method of classifying and categorizing information by adding labels to it. To put it another way, it is a method of creating or attaching descriptive metadata to an article, a social media post, or any other piece of information. They form an additional kind of faceted classification and navigation system for your app or website: a topic-based one. If you provide or attach these tags, the customer can use them to search and browse content by these facets, or category descriptors.

Another aspect of tags is that they can be provided by the reader or consumer, too. They can be a form of user-generated content. Tags can be a way for users to manage their own data: grouping of content, searching, browsing, sharing, or recall. Multiple tags can be added to the same post or article. With enough tagged content, online consumers can view or browse the content or online activity using tags created by co-users. This is a common feature of discussion boards, blogs and social-enabled apps and sites of all kinds.

Social media has created and spread the near-universal standard of adding the hash or pound symbol "#" to the beginning of a word to mark it specifically as a tag, or "hashtag. " Hashtags create a way for users to link from a single social post or article to a search results feed that contains posts from other people also marked with the same hashtag. This is a powerful way for information to spread rapidly.

Use when

Enable tagging when you want to take advantage of your users' desire to classify, browse, share, and promote content associated with topics of interest to them. Tags are a user-generated classification system. When this is generated by your readers or customers and is accessible to others, it becomes a way for them to discover and navigate to content that's of interest. Apps or platforms that have a large amount of information, such as news publishers or social media sites, will add tags to their content, or allow users to tag their posts. In this way, a homegrown topic navigation and discovery system grows along with the content, a complement to whatever formal navigation and IA scheme you have designed into your product.

Why

The purpose of tagging from a design perspective is twofold. First, it promotes increased usage of your app or website because your users are able to find content, media, and other information that is highly relevant to the tag or keyword of interest at the moment. Even more, if you have users who actively tag content, they are becoming more invested in your product or platform because they are creating their

own content (usually for social media sharing) using your content. Both of these can help to increase usage and retention.

The second reason to tag your content or enable your users to tag is to crowdsource an organic organization scheme for your app or community that would otherwise be too expensive and time consuming to do on your own. These can form quickly around topics of the moment, and they can also be long-lived topics. Over time, a naturally user-centered structure and order can emerge. This can be especially useful if you want to activate or amplify your customers' own interest in finding, reading, sharing, commenting on, and more related to topics that are naturally interesting to them.

How

Create or incorporate software that allows words to be added to content or posts as tags. Furthermore, your search function must be able to search the tagged content and create index or results screens that show all content that is tagged with a particular keyword. Tags or hashtags also should be formatted as links automatically. Selecting the tag generates a search based on that tag, and creates a search results page with most relevant or most recent content also tagged with that term.

Examples

Stack Overflow (Figures 2-65 and 2-66) is a hugely popular question-and-answer discussion board and online community that serves the software developer community. This website consists almost totally of user-generated content in the form of threaded, tagged discussions. In other words, it offers a deep and robust crowd-sourced tag/topic system for finding information. Readers can browse lists of most recent and most popular questions. They also use the tags attached to each post. Participants tag their posts liberally. This allows readers to find discussion threads tagged with the same topic, and closely related topics. Tracking and displaying related tags creates a rich information and navigation browsing ecosystem with high likelihood of readers finding relevant content related to their topic of interest.

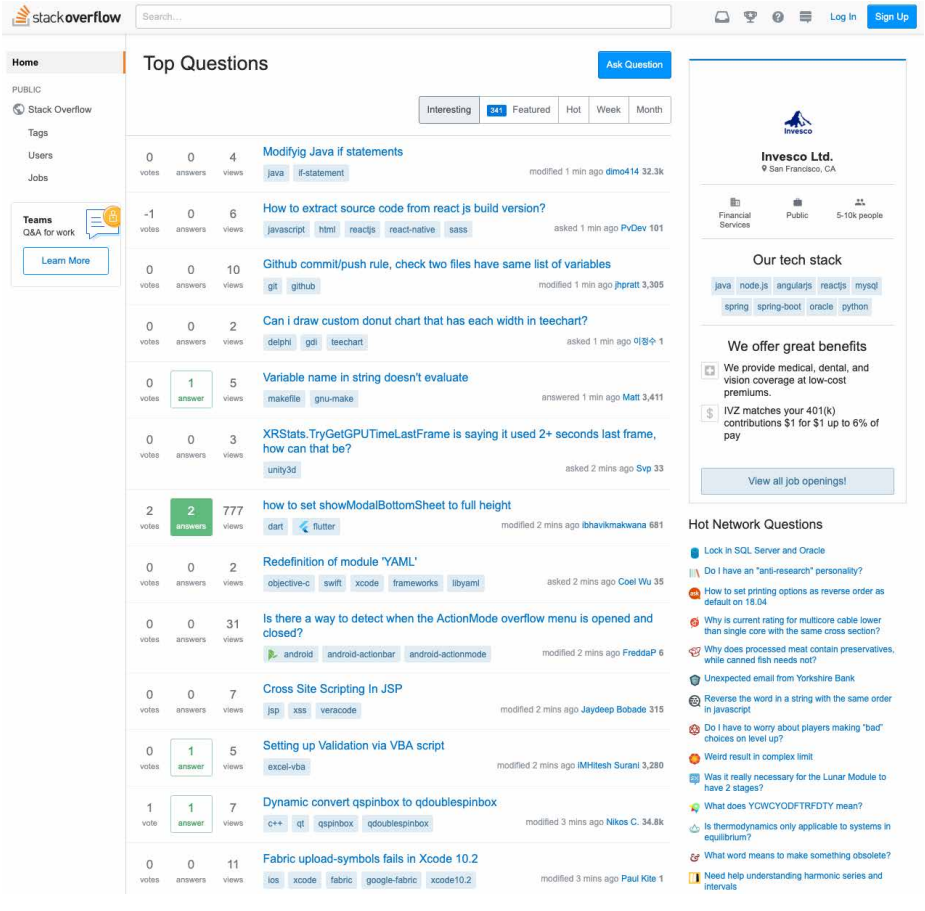


Figure 2-65. *Stack Overflow, showing relevant tags (each highlighted within in a round-cornered rectangle below the subject) for each question*

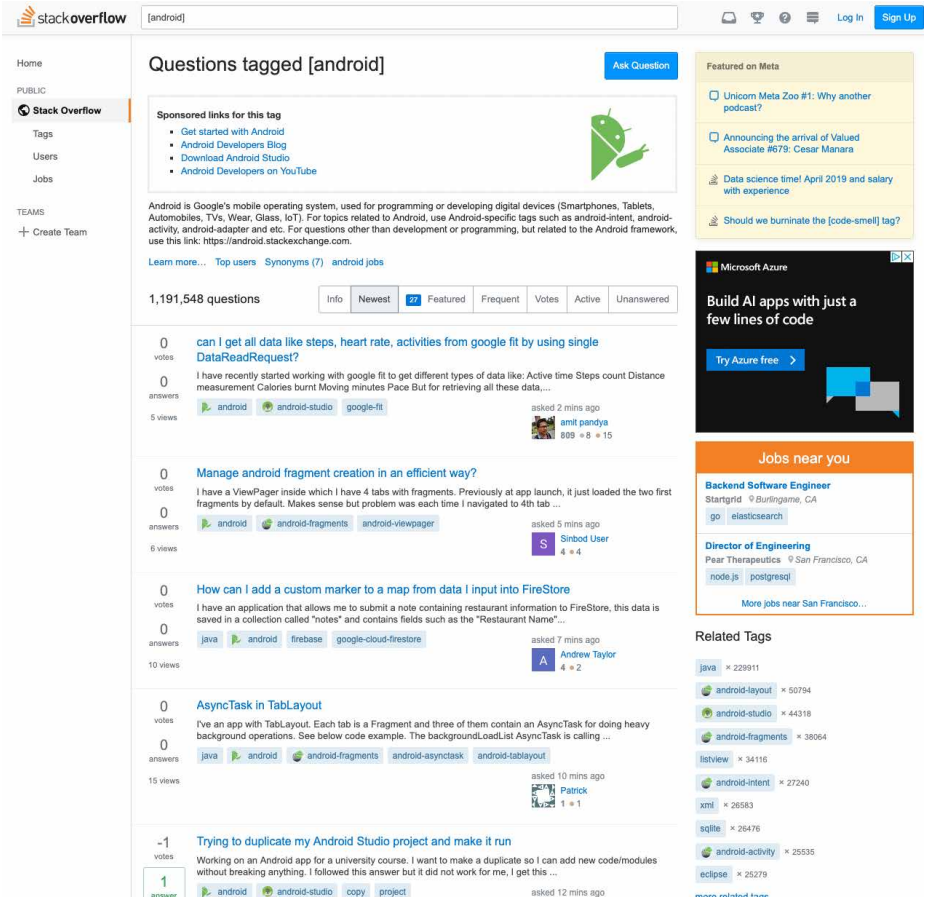


Figure 2-66. *Stack Overflow filtered by tag*

Texas Monthly (Figure 2-67) uses the massively popular Wordpress content management system and web publishing platform to create their online magazines. A prominent feature of Wordpress is the ability to add tags to articles and posts at the time of publication. These tags are displayed along with the main article, and link to other posts within the Wordpress system, which generates additional pages for the reader based on the tag they clicked. In the Texas Monthly example, several tags display at the bottom of an article. The authors have chosen a number of keywords to describe this and other articles. In this example, when the user clicks the tag "Travel," for example, they can see a list of other travel articles that Texas Monthly has published. This promotes increased time on the site, site circulation, and reader engagement.

Figure 2-67. *Texas Monthly article tags (powered by Wordpress)*

Evernote is a personal note-taking and web archiving application for mobile, web, and desktop. Its users save articles, files, presentations and web pages into a central location: the Evernote platform. It allows users to add their own tags when "clipping" a web page to Evernote (Figure 2-68). This helps with categorizing and finding similar saved articles later. There is also a tag index screen where the user can see what tags exist, and how many articles or clippings are tagged with each term (Figure 2-69). Selecting a given tag creates a search results list of clippings with that tag. In addition to free-text searching, Evernote's user-generated tag system creates a powerful method for categorizing and managing large amounts of disparate media and information. Inside the Evernote app itself, the user can search by tag (Figure 2-70).

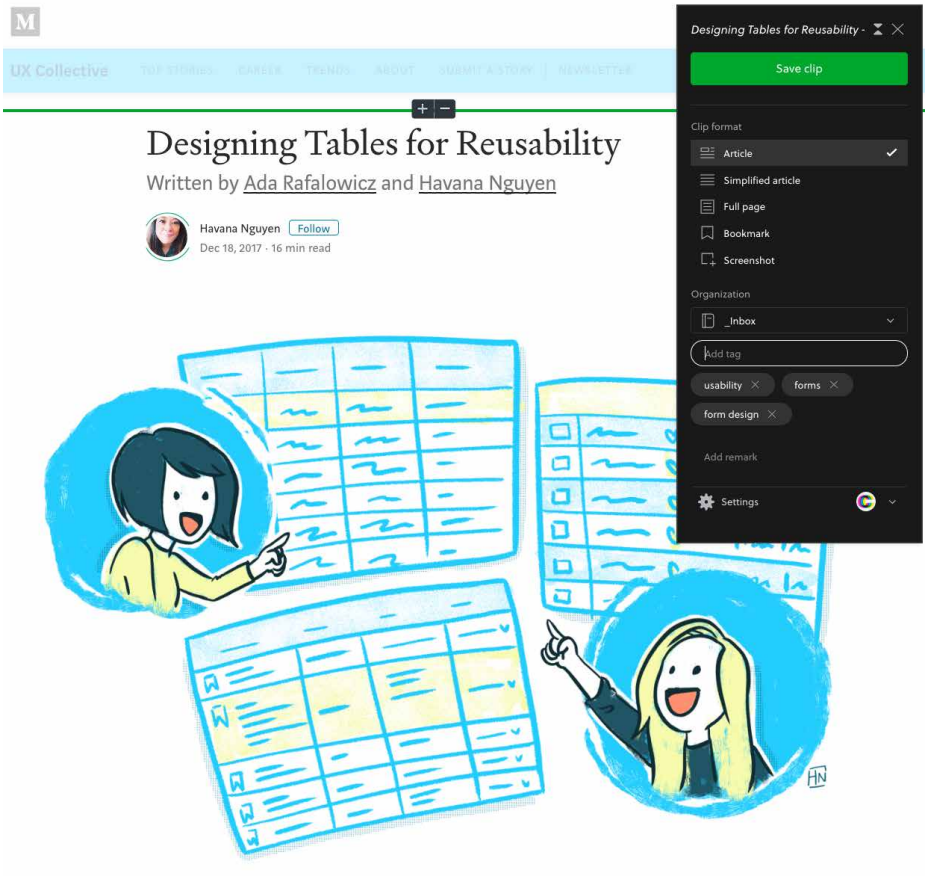


Figure 2-68. *Evernote Screen Clipper offers the ability to add tags*

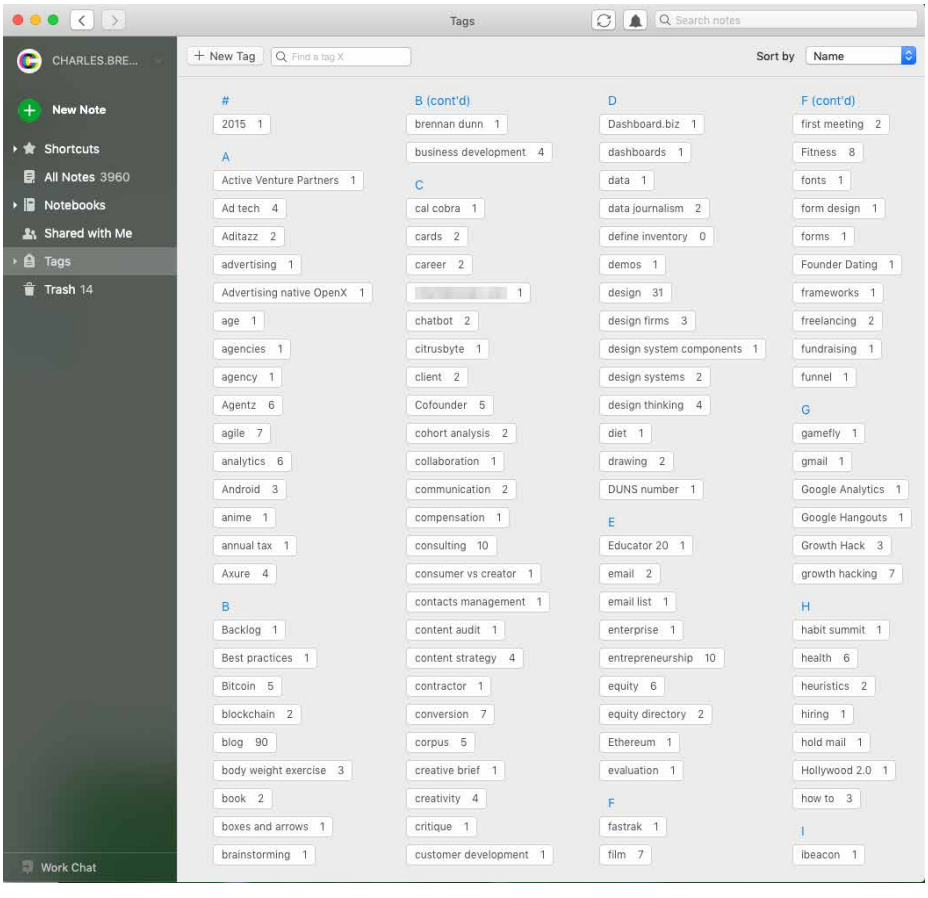


Figure 2-69. *Evernote app with tags browser*

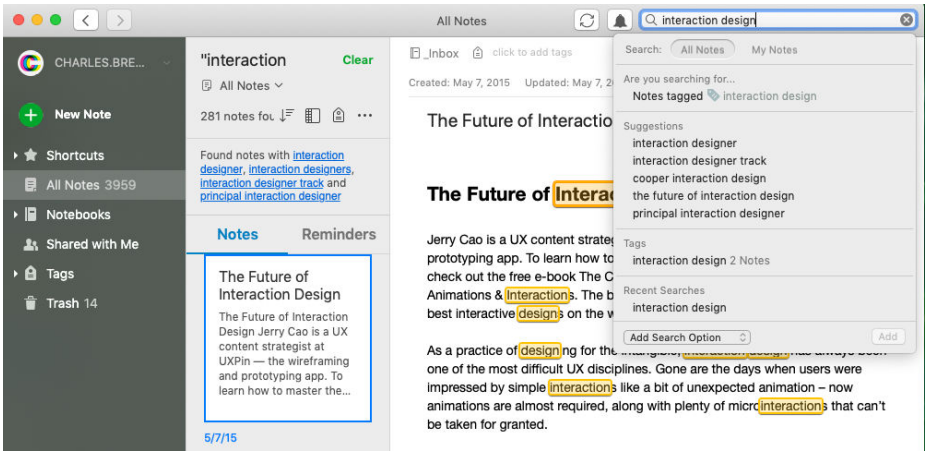


Figure 2-70. *Evernote app with ability to search by tags*

Conclusion

Information architecture, or IA, is a challenging aspect of designing software because it is simultaneously abstract (organization and labeling schemes) and also concrete (navigation and tags, for example). It's important to remember that the IA should be based on the users' vocabulary and mental model of the business or task domain. The IA should be designed to accept new content and data over time, and still serve to organize and hold everything without confusion. So, design for an "evergreen" information architecture. The processes outlined in this chapter will help you do just that: mutually exclusive, collectively exhaustive categorization; using commonly understood organization methods, chunking up content, developing a system of screen types based on usage, and then the patterns in this chapter. It's useful to think of the IA of your software as being the invisible organization scheme, and the various interaction design patterns and widgets as ways of exploring this information space. Search, browse, navigation systems, tags, cross-links, multiple media types, and more might all be necessary to learn and navigate the information architecture fully.

In Chapter 11, we look at how modern UI frameworks and atomic design principles help us to quickly build out these IA widgets at scale.

Chapter 3

Getting Around: Navigation, Signposts, and Wayfinding

The patterns in this chapter deal with the challenges of navigation: How do users know where they are now? Where can they or should they go next? How can they get there from here?

To answer these questions, we look at these important aspects of navigation:

- The purpose of navigation in user experience (UX)
- Methods to promote wayfinding in your software
- Different types of navigation
- How to design navigation
- Patterns of navigation that can be useful

Navigation can be challenging because moving around in a website or application is like commuting: you need to do it to get where you want to go, but it's dull, it's sometimes infuriating, and the time and energy you spend on it just seems wasted. Couldn't you be doing something better with your time, such as playing a game or getting some actual work done?

The best kind of commuting is none at all. Having everything you need right at your fingertips without having to travel somewhere is pretty convenient. Likewise, keeping most tools "within reach" on an interface is handy, especially for intermediate-to-expert users (i.e., people who have already learned where everything is). Sometimes, you do need to put lesser-used tools on separate screens where they don't clutter things up; sometimes, you need to group content onto different pages so that the interface makes sense. All of this is fine as long as the "distances" that a user must travel remain short. So, less is better.

Understanding the Information and Task Space

The purpose of navigation is to help the user know and understand the information space they are in. This includes understanding what tasks they can do, as well. Finally, they need to know how to get around. Navigation helps users know the following:

- The information and tools that are available in terms of subject and scope
- How the content and functionality are structured
- Where I am now
- Where I can go
- Where I came from and how to go back or how to back up

Suppose that you've built a large website or application that you've had to break up into sections, subsections, specialized tools, pages, windows, wizards, and so forth. How do you help users navigate?

Signposts

Signposts are features that help users figure out their immediate surroundings. Common signposts include page and window titles, web-page logos and other branding devices, tabs, and selection indicators. Patterns and techniques such as good global and local navigation links, *Progress Indicator*, *Breadcrumbs*, and *Annotated Scroll Bar*—all described in this chapter—inform users as to where they currently are and, often, where they can go with only one more jump. They help a user to stay "found" and to plan their next steps.

Wayfinding

Wayfinding is what people do as they find their way toward their goal. The term is pretty self-explanatory. But how people actually do it is quite a research subject—specialists from cognitive science, environmental design, and website design have studied it. These common-sense features help users with wayfinding:

Good signage
: Clear, unambiguous labels anticipate what you're looking for and instruct you where to go; signs are where you expect them to be, and you're never left standing at a decision point without guidance. You can check this by walking through the artifact you're designing and following the paths of all the major use cases. Make sure that each point where a user must decide where to go next is signed or labeled appropriately. Use strong "calls to action" on the first pages that a user sees.

Environmental clues
: You'd look for restrooms in the back of a restaurant, for instance, or a gate where a walkway intersects a fence. Likewise, you would look for an "X" close button in the upper right of a modal dialog box and logos in the upper-left corner of a web page. Keep in mind that these clues are often culturally determined, and someone new to the culture (e.g., someone who's never used a given operating system before) might not be aware of them.

Maps
: Sometimes, people go from sign to sign or link to link without ever really knowing where they're going in a larger frame of reference. (If you've ever found your way through an unfamiliar airport, that's probably what you did.) But some people might prefer to have a mental picture of the whole space, especially if they're there often. Also, in badly signed or densely built spaces, such as urban neighborhoods, maps might be the only navigational aids people have.

In this chapter, the *Clear Entry Points* pattern is an example of careful signage combined with environmental clues–the links should be designed to stand out on the page. A *Progress Indicator*, obviously, is a map. *Modal Panel* sort of qualifies as an environmental clue because the ways out of a modal panel take you directly back to where you just were.

I've compared virtual spaces to physical spaces here. But virtual spaces have the unique ability to provide a navigational trump card, one that physical spaces can't (yet) provide: the *Escape Hatch*. Wherever you are, click that link, and you're back to a familiar page. It's like carrying a wormhole with you. Or a pair of ruby slippers.

Navigation

Let's briefly review the different types of navigation that designers commonly use and that users are probably familiar with. The types described here are broadly determined by function. The listing here is not exhaustive. Naming can also vary but we use some of the most common terms: global, utility, associative and inline, related content, tags, and social.

Global Navigation

This is the site or app navigation that is on every main screen. It usually takes the form of menus, tabs, and/or sidebars, and this is how users move around the formal navigational structure of the site. (In an earlier version of this book, global navigation was defined as a pattern. But by now, it's so common and well understood that it really doesn't need to be called out as such anymore.)

Global navigation is almost always shown at the top or left of a web page, sometimes both (called the *inverted L* navigation layout). Rarely, you might find it on the right—but this placement can cause problems with page size and horizontal scrolling.

In the mobile environment, we see two main approaches to global navigation. The first is a navigation bar that sits at the bottom of the screen. It remains in that location as the user goes from screen to screen. It might have an internal left-right scroll, as well, if there are additional navigation items. The second approach is the "hamburger" menu; this is a stack of three horizontal lines. Sometimes, this is a slenderer stack of three dots. Tapping this invokes a panel with the global navigation choices.

Utility Navigation

This is also found on every main screen and contains links and tools related to non-content aspects of the site or application: sign-in, help, print, *Settings Editor* (Chapter 2), language tools, and so on.

When a site's visitors are typically signed-in members, that site might offer a set of utility navigation links in its upper-right corner. Often there is an icon of a human or a tiny photo of the member, if that is available. This is the symbol of the member—their *avatar*—clearly hinting that personal-to-you information is located here. Users tend to look there for tools related to their presence on the site: account settings, user profile, logout, help, and so on. See the *Sign-in Tools* pattern for more. Sometimes, all the utility navigation items are displayed. Often, they are collapsed behind the avatar icon, and the user must click to open it.

Associative and Inline Navigation

These are links in or near the actual content. As the user reads or interacts with the site, these links present options that might be immediately relevant to the user. They tie content together thematically.

Related Content

A common form of associative navigation is a *Related Articles* section or panel. News sites and blogs use this a lot. When a user reads an article, a sidebar or footer shows other articles that talk about similar topics or are written by the same author.

Tags

Tags, user defined and system defined, can help support associative navigation and related articles or links. Tag clouds support topical findability on some sites, especially where the number of articles is very large and the topics fine-grained. (On smaller sites and blogs, they don't work as well.) A more common navigational

technique is to list an article's tags at the end; each tag is a link leading to an entire set of articles that share that tag.

Social

When a site takes advantage of social media integration, even more navigation options come into play. This often takes a number of forms. There might be a news or postings module that displays activity and content shared by your friends. There might be a type of leaderboard or "trending now" component. This provides links to stories and posts that are being shared the most among all users on the social media network. For a closer look at how to use the social graph in your designs, see *Designing Social Interfaces: Principles, Patterns, and Practices for Improving the User Experience* by Christian Crumlish and Erin Malone (O'Reilly, 2015).

Design Considerations

Navigation must be designed. What navigation options are displayed, how they are labeled, and where and when the navigation displays in the UI are all considerations for design. This section briefly discusses design considerations for effective navigation.

Separate the Navigation Design from the Visual Design

It's a good process to separate the design of the navigational model from its visual design. Work out the number and sequence of navigation options that are needed and what it shows by default. Is it necessary for the user to expand the navigation categories to browse the structure, with links to a second or even third levels? Or is that not necessary? Reasoning this way can help you think more flexibly and deliberately about how to design the pages themselves. After you've established that, you then can consider the look and feel. There are conventions regarding visual placement of navigational features. Although it's tempting to be different, there are huge advantages to following common layout patterns.

Cognitive Load

When you walk into an unfamiliar room, you look around. In a fraction of a second, you take in the shape of the room, the furnishings, the light, the ways out, and other clues; very quickly you make some assumptions about what this room is and how it relates to why you walked in. Then, you need to do what you came in to do. Where? How? You might be able to answer immediately—or not. Or maybe you're just distracted by other interesting things in the room.

Similarly, bringing up a web page or opening a window incurs a cognitive cost. Again, you need to figure out this new space: you take in its shape, its layout, its contents, its

exits, and how to do what you came to do. All of this takes energy and time. The "context switch" forces you to refocus your attention and adjust to your new surroundings.

Even if you're already familiar with the web page (or room) you just went into, it still incurs a cost in terms of time for perceiving, thinking, and reorienting yourself to the information space. Not a large cost, but it adds up—especially when you figure in the actual time it takes to display a window or load a page.

This is true whether you're dealing with web pages, application windows, dialog boxes, or device screens. The decisions that users make about where to go are similar—they still need to read labels or decode icons, and the users will still make leaps of faith by clicking links or buttons they're not sure about.

Furthermore, loading time affects people's decisions. If a user clicks through to a page that takes too long to load—or fails to load altogether—they might be discouraged and just close the page before they find what they came for. (So, how many viewers is that sidebar video player costing you?) Also, if a site's pages take a chronically long time to load, users will be less likely to explore that site.

There's a reason why companies like Google work very hard to keep page loads as fast as possible: *latency costs viewers*.

Keep Distances Short

Knowing that there's a cost associated with jumping from page to page, you can understand now why it's important to keep the number of those jumps down. A great rule of thumb is to think about how to keep the number of taps or clicks to get from point A to point B as small as possible. There are several ways you can optimize for this in your navigation design.

Broad global navigation

Design your navigation and your application so that there are more selections at the first, topmost level. Make the site structure as flat as possible; that is, minimize the levels of the site hierarchy. Put access to more screens directly in the global navigation. Put another way, avoid having just a few top-level navigation items if that means users must navigate a lot of category and subcategory menus.

Put frequently accessed items directly in the global navigation

Frequency of use influences the design of your navigation, too. Elevate or raise frequent actions so that they are at the top level of your navigation structure and thus there is direct access. This is independent of where they are in the structure of your site or app.

You can bury infrequently-used tools or content in the site structure. They will require a drilldown in the appropriate submenu. This is true within a single tool or screen, as well. You can hide seldom-used settings or optional steps behind an extra "door : a closed accordion panel or a tabbed panel. As always, experiment with different designs, and usability-test them if you have any doubts.

Bring steps together

Real efficiency gains can come from structuring the application so that common tasks can be accomplished in a single screen. One of the most annoying situations for users is to have a simple or frequent task, but be forced to go into multiple levels of subpages, dialogs, and so forth to perform one step in each place. It's even worse if there is a dependency among the steps. Having to back up because of a missing precondition is wasted time and energy.

Can you design your workflows so that the most common 80% of use cases can be done in one page, without any context switches?

This is difficult to do with some kinds of applications. A certain tool or form can simply be very complicated. First try simplifying and minimizing. Group and segment the screen, but then shorten labels, turn words into pictures, use different form controls that save space, and move instructional copy to tool tips and pop-up panels. Then, look at using progressive disclosure so that only the first step or most-used controls display. Consider *Module Tabs* or an *Accordion* to hide other steps or content by default. This can be revealed automatically as the user works their way through the tool, or it can be optional information that the user must click or tap to view.

A second method is to bring multiple steps, tools, or screens together into a single *Wizard* with multiple steps (we examine this shortly).

Navigational Models

What is the navigational model for your site or app? In other words, how do the different screens (or pages, or spaces) link to one another and how do users move between them?

Now let's look at a few models found in typical sites and apps.

Hub and Spoke

Most often found on mobile devices, this architecture (Figure 3-1) lists all of the major parts of the site or app on the home screen, or "hub." The user clicks or taps through to them, does what they need to do, and comes back to the hub to go somewhere else. The "spoke" screens focus tightly on their jobs, making careful use of space—they might not have room to list all of the other major screens. The iPhone home screen is a good example; the *Menu Page* pattern found on some websites is another.

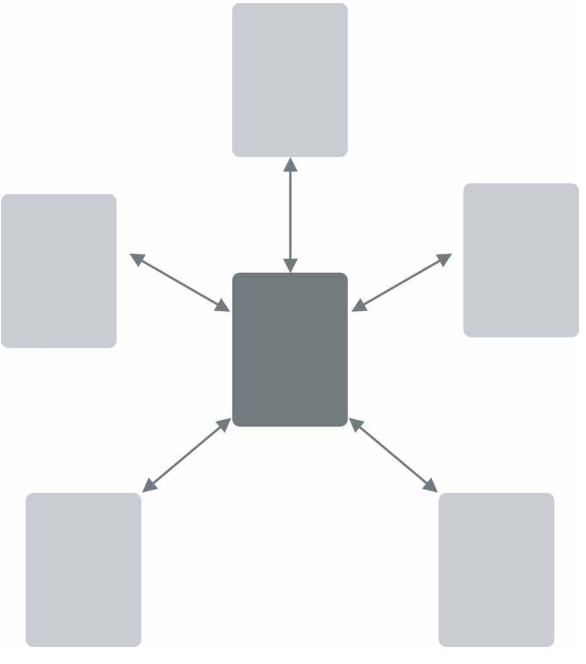


Figure 3-1. *Hub and spoke architecture*

Fully Connected

Many websites and mobile applications follow this model. There's a home page or screen, but it and every other page link to all the others—they each have a global navigation feature, such as a top menu. The global navigation might be a single level (as shown in Figure 3-2, with only five pages), or it might be deep and complex, with multiple levels and deeply buried content but with complete navigation on every screen. As long as the user can reach any page from any other with a single jump, it's fully connected.

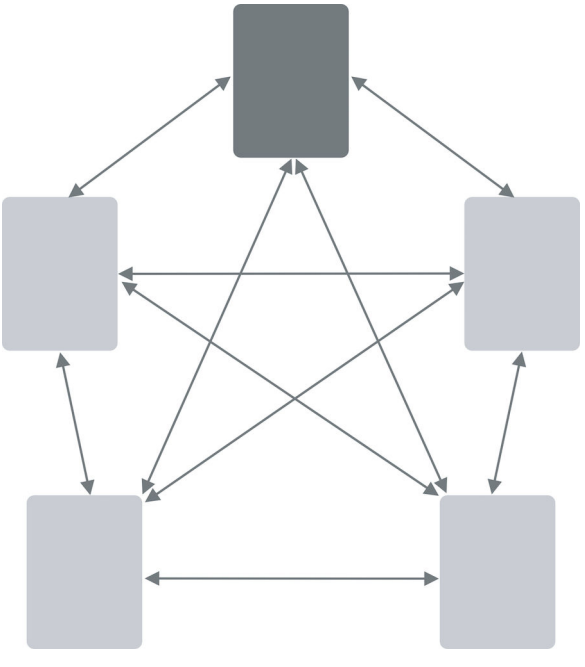


Figure 3-2. *The fully connected model*

Multilevel or Tree

This is also common among websites (see Figure 3-3). The main pages are fully connected with one another, but the subpages are connected only among themselves (and usually to the other main pages, via global navigation). You've seen this on sites that have subpages listed only in sidebars or subtabs—users see these on menus that only show up after they've clicked the link for the main page or category. It takes two or more jumps to get from one arbitrary subpage to another. Using drop-down menus, the *Fat Menus* pattern, or the *Sitemap Footer* pattern with a multilevel site converts it to a fully connected one, which is preferable.

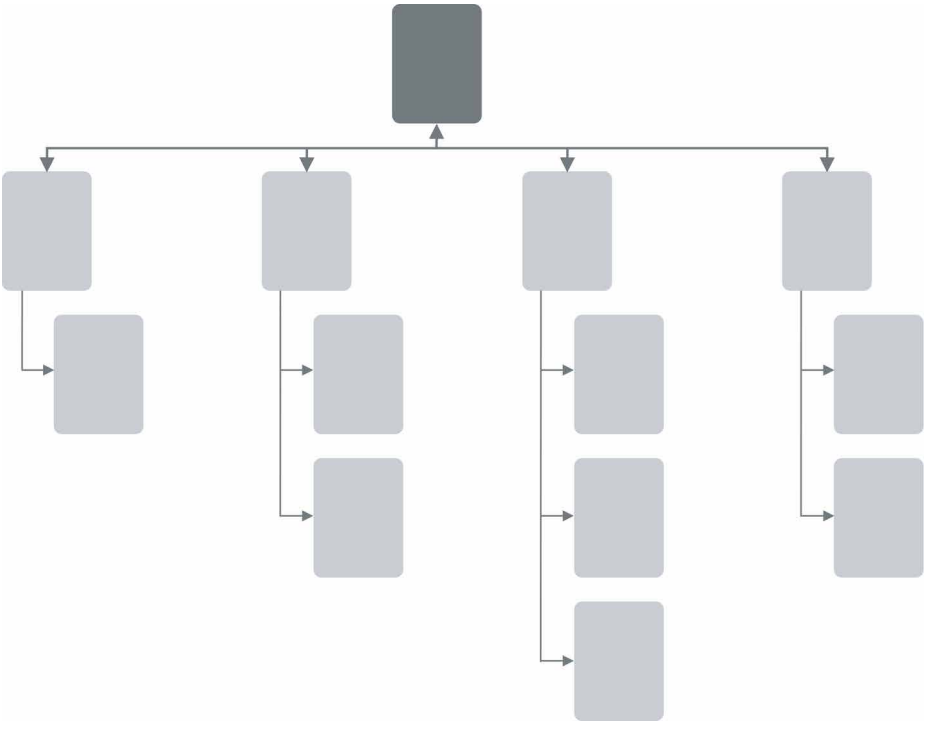


Figure 3-3. *Multilevel navigation*

This pattern is also found in enterprise web software. Companies that offer a suite of web applications often keep these apps separate. This can be because full integration into a unitary platform is not desired, or simply hasn't happened yet, or because the apps are a result of recent or ongoing acquisitions. Another reason is that the components of the suite might be sold separately. The end result is a product set in which each application is an experience on its own, separate from the others. However, the software provider and the customer usually want a single sign on and a single point of access to everything. So, they are tied together at the top, behind a login. There, the user can access each of the applications and manage their account settings.

Step by Step

Slideshows, process flows, and *Wizard* (Chapter 2) lead the user step by step through the screens in a prescribed sequence (Figure 3-4). Back/Next links are prominent on the page. Stepwise navigation can be as simple as designing a search interface that then presents a search engine results page. Ecommerce purchase flows are also a common example. Here, there is a designed path from product page, to shopping cart, to the checkout process (can be multiple screens), and finally the completed transaction confirmation. As a third example, many subscription retail companies (especially clothing, cosmetics, and other personal goods) have a questionnaire or online survey as the first step in the customer onboarding process. The customer steps through a series of questions that establish their preferences for style, budget, sizes, brands, frequency of delivery, and so on.

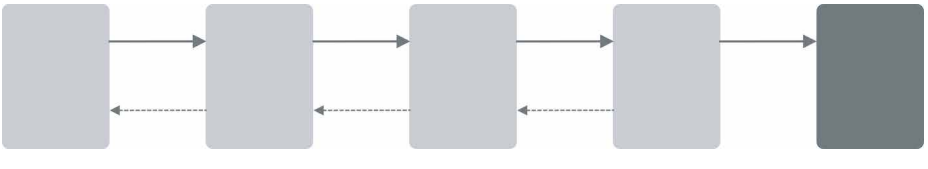


Figure 3-4. *Step-by-step flows*

Pyramid

A variant on the stepwise model, a pyramid uses a hub page or menu page to list an entire sequence of items or subpages in one place (see Figure 3-5). The user picks out any item, jumps to it, and then has the option to use Back/Next links to step through other items in order. They can go back to the hub page anytime. See the *Pyramid* pattern in this chapter for more. This is very common for content sites that publish stories as a gallery of pictures.

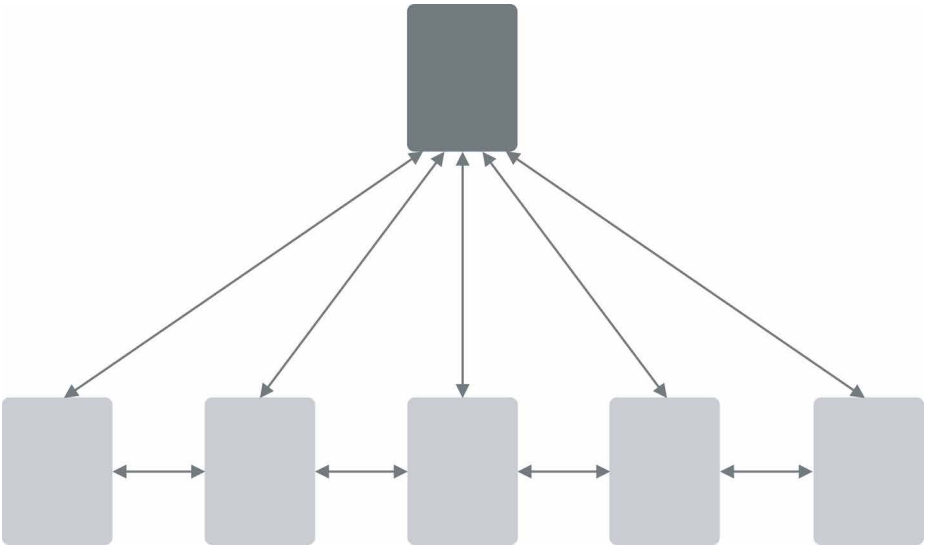


Figure 3-5. *Pyramid*

Some artifacts are best represented as single large spaces, not many small ones. Maps, large images, large text documents, information graphics, and representations of time-based media (such as sound and video) fall into this category. Panning and zooming are still navigation–so offer controls for panning (moving horizontally or vertically), zooming in and out, and resetting to a known position and state.

Figure 3-6 shows an example of pan-and-zoom. Map interfaces are the most common example of this type of navigation.

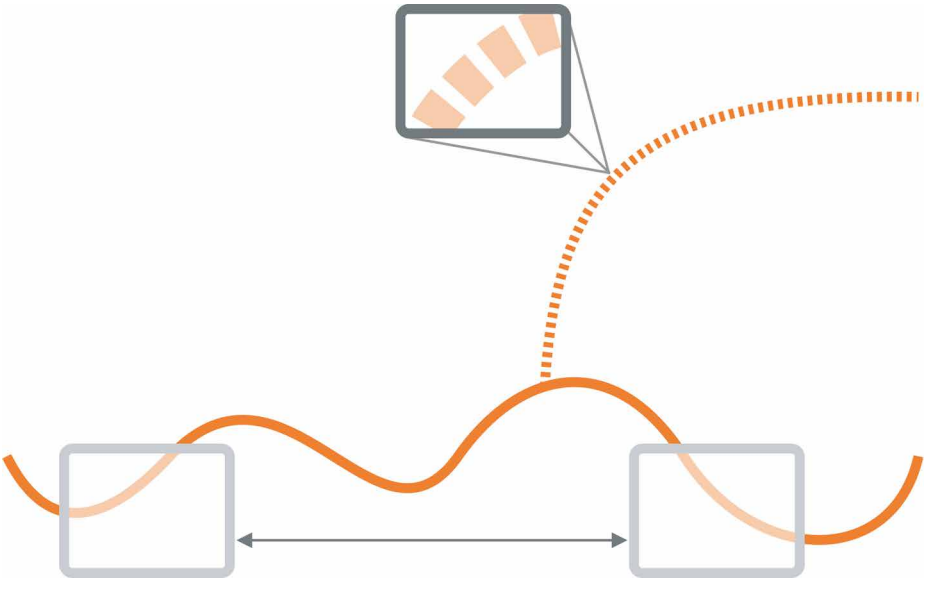


Figure 3-6. *Pan and Zoom*

Flat Navigation

Some types of applications need little or no navigation at all. Consider *Canvas Plus Palette* applications such as Photoshop, or other complex apps such as Excel–these offer tons of tools and functions that are easily reached via menus, toolbars, and palettes. Tools that don't act immediately upon the work can be accessible via *Modal Panel* or step-by-step progressions. These types of applications seem to be qualitatively different from the other navigation styles listed here: the user always knows where they are, but they might not easily find the tools they need because of the sheer number of features available at one time.

There are three things to notice about these navigational models. The first is that they're mix and match–an app or site might combine several of these.

The second thing is universal global navigation and short jumps are good things most of the time. But at other times, a mode with very few navigation options is better. When a user is in the middle of a full-screen slideshow, they don't want to see a complicated global navigation menu. They would rather just focus on the slideshow itself, so Back/Next controls and an *Escape Hatch* are all that's necessary. The presence of full global navigation everywhere is not without cost: it takes up space, clutters the screen, incurs cognitive load, and signals to the user that leaving the page doesn't matter.

Third, all of these mechanisms and patterns can be rendered on screen in different ways. A complex site or app might use tabs, menus, or a sidebar tree view to show the global navigation on each page–that's something you don't need to decide until you begin laying out the page. Likewise, a modal panel might be done with a lightbox or an actual modal dialog–but you can postpone that until you know what needs to be modal and what doesn't.

Visual design can come later in the design progression, after the information architecture (IA) and navigational models.

The Patterns

This chapter talks about several aspects of navigation: overall structure or model, knowing where you are, determining where you're going, and getting there efficiently.

The first set of patterns address the navigational model and are more or less independent of screen layout:

- *Clear Entry Points*
- *Menu Page*
- *Pyramid*
- *Modal Panel*
- *Deep Links*
- *Escape Hatch*
- *Fat Menus*
- *Sitemap Footer*
- *Sign-In Tools*

The next few patterns work well as "You are here" signposts (as can a well-designed global navigation):

- *Progress Indicator*
- *Breadcrumbs*
- *Annotated Scroll Bar*

Progress Indicator, *Breadcrumbs*, and *Annotated Scroll Bar* also serve as interactive maps of the content. *Annotated Scroll Bar* is intended more for pan-and-zoom models than for multiple interconnected pages. Finally, *Animated Transition* helps users stay oriented as they move from one place to another. It's a visual trick, nothing more, but it's very effective at preserving a user's sense of where they are and what's happening. Now, let's begin exploring these patterns.

Clear Entry Points

What

Present only a few main entry points into the interface so that the user knows where to start. For first-time and infrequent users, it removes some of the burden of learning the site. Make them task-oriented or directed at a specific audience type. Use clear calls to action. The *Clear Entry Points* schematic in Figure 3-7 represents this concept.

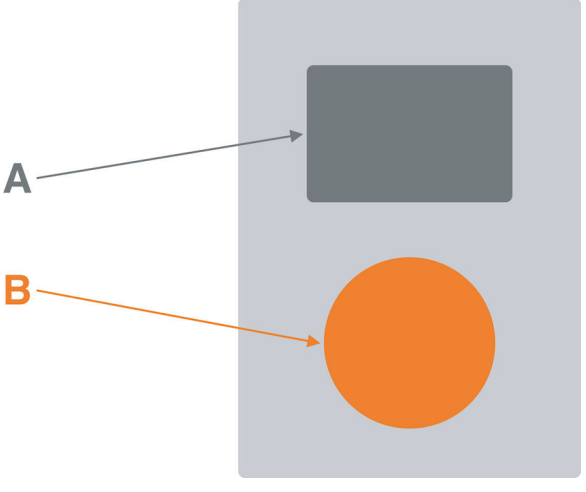


Figure 3-7. *Clear Entry Points schematic*

Use when

You're designing a site or application that has a lot of first-time or infrequent users. Most of these users would be best served by reading a certain piece of introductory text, doing an initial task, or choosing from a very small number of frequently used options.

However, if the purpose is clear to basically everyone who starts it, and if most users might be irritated by one more navigation step than is necessary (like applications designed for intermediate-to-expert users), this might not be the best design choice.

Why

Some applications and websites, when opened, present the user with what looks like a morass of information and structure: lots of tiled panels, unfamiliar terms and phrases, irrelevant ads, or toolbars that just sit there disabled. They don't give the hesitant user any clear guidance on what to do first. "OK, here I am. Now what?"

For the sake of these users, list a few options for getting started. If those options match a user's expectations, they can confidently choose one and begin working-this contributes to immediate gratification. If not, at least they know now what the site or app actually does, because you've defined the important tasks or categories at the outset. You've made the application more self-explanatory.

How

When the site is visited or the application started, present these entry points as "doors" into the main content. From these starting points, guide the user gently and unambiguously into the application until they have enough of a context to continue by themselves.

Collectively, these entry points should cover most of the reasons most users would be there. There might be only one or two entry points, or several; it depends on what fits your design. But you should phrase them with language that first-time users can understand—this is not the place for application-specific tool names.

Visually, you should show these entry points with emphasis proportional to their importance.

On the home page or starting page, most sites will additionally list other navigation links—global navigation, utility navigation, and so on—and these should be smaller and less prominent than the *Clear Entry Points* . They're more specialized, and don't necessarily lead you directly into the heart of the site any more than a garage door leads you directly into the living room. The *Clear Entry Points* should serve as the "front doors."

Examples

Apple's main iPad page (Figure 3-8) needs to do only a few things: identify itself, make the iPad look inviting, and direct the user toward resources for buying one or learning more. The global navigation recedes visually, compared to the strong, well-defined entry point. There is a secondary focus on the row of small schematic diagrams. These are links to the iPad models, too, plus accessories. But above the fold, the users is clearly directed to the iPad Pro. On the rest of the page, there are additional front doors—large promotional images for other iPad models.

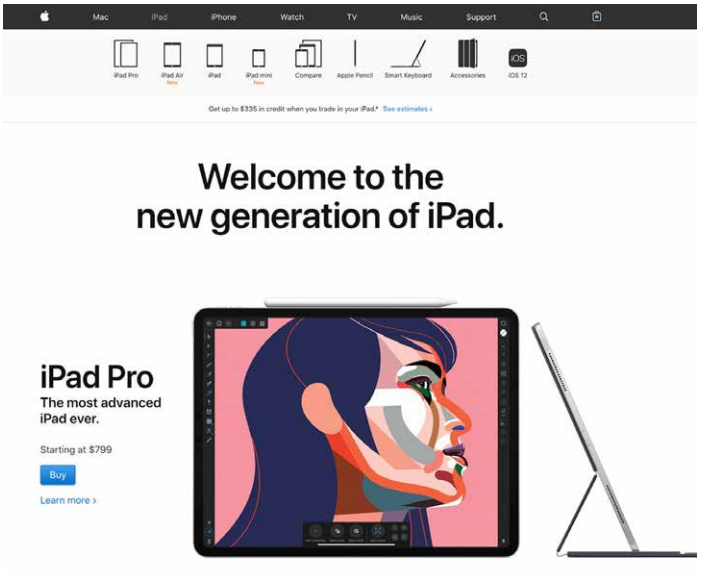


Figure 3-8. *iPad page on Apple's site*

Spotify (Figure 3-9) focuses exclusively on the new customer with its website's landing page. There is a simple and clear call to action in the center of the screen.

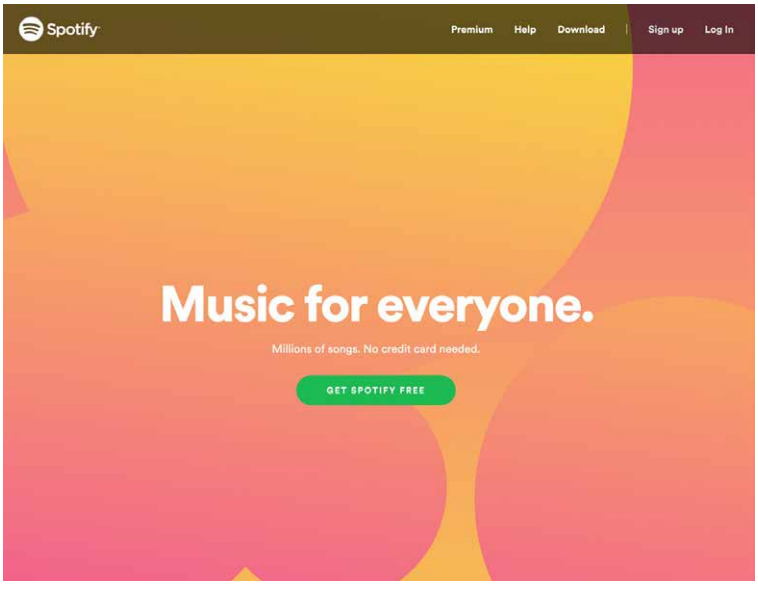


Figure 3-9. *The Spotify landing page—a very clear single door*

Adobe Illustrator and other applications show a startup dialog when the application is started (see Figure 3-10). This orients a new or infrequent user to the possibilities for action. The major actions are creating something new or opening an existing document. Each one is treated twice. On the left, for more experienced or confident users who are ready to get down to business, there are two boldly marked buttons: Create New and Open. Despite being small, their visual treatment makes them pop as entry points into getting started. On the right, there are the same two options, but given a much bigger, more visual and more explanatory approach. "Start a New File Fast" has several choices that represent most likely device and screen sizes. The schematic diagrams for each make them even easier to understand. Below this, in Recent, is a grid of recently opened files, each with a thumbnail image to assist in recognition and recall. This is a good example of designing different entry points to appeal to different users.

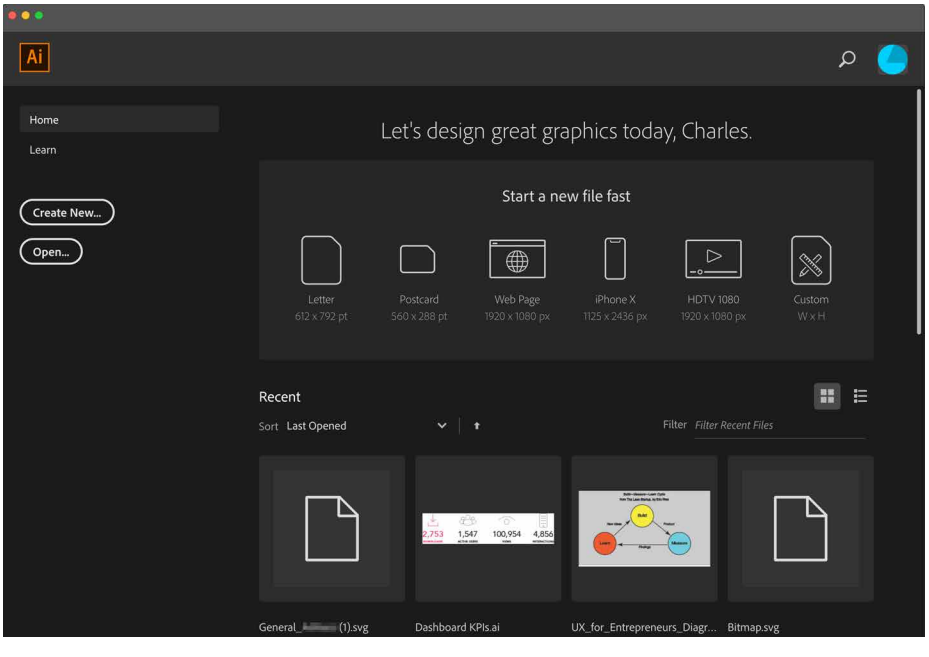


Figure 3-10. *The Adobe Illustrator CC startup dialog*

Prezi (Figure 3-11), like Spotify, is using its entry point on the website to make it easy for potential customers to move toward purchase. In Prezi's case, it realized its innovative presentation software needs demonstration. Differentiating its product is the biggest need, and probably the biggest question in the minds of Prezi's potential buyers. The front door sends the message, "Come in and check it out."

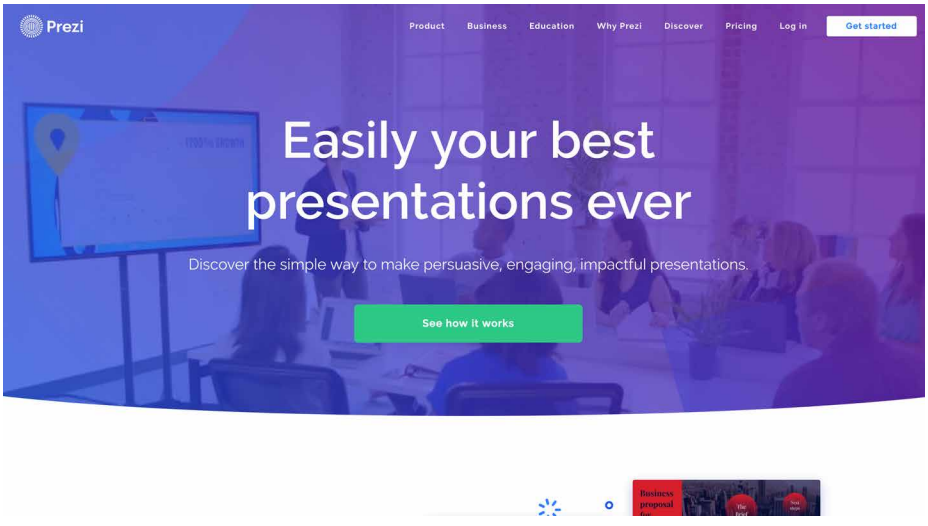


Figure 3-11. *Prezi's landing page*

Tesla (Figure 3-12) offers three entry points as represented by the three Tesla models in the photograph. The primary focus is on the Model 3 (the user interface zooms in on it, a clever move). With the focus on the Model 3, there are two entry points: Customize your own Tesla or browse the cars that are available now.

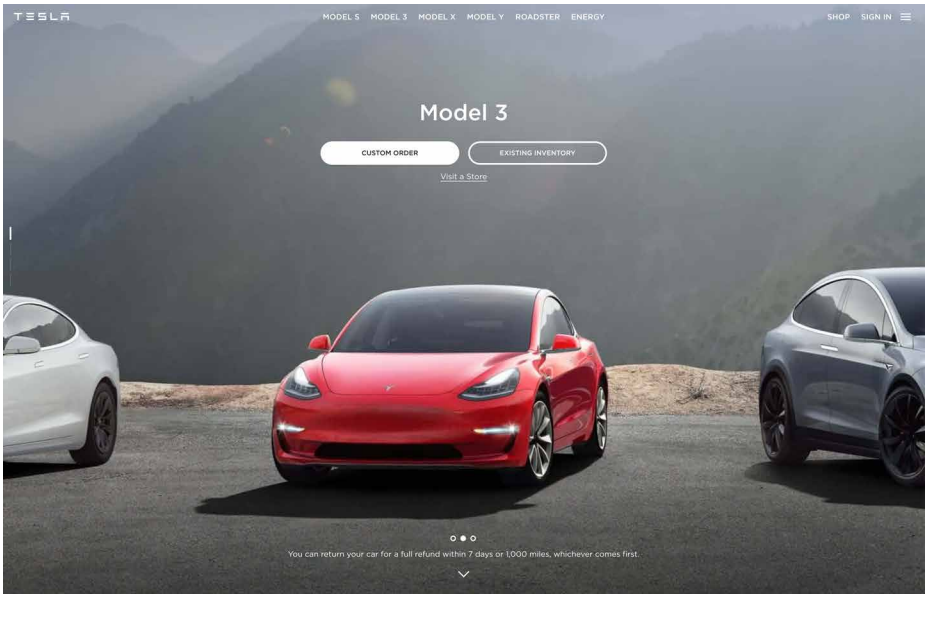


Figure 3-12. *Tesla's landing page*

Menu Page

What

Fill the page with a list of links to content-rich pages in your site or app. Show enough information about each link to enable the user to choose well. Show no other significant content on the page. The venerable Craigslist home page (Figure 3-13) is an example of a highly successful menu page.

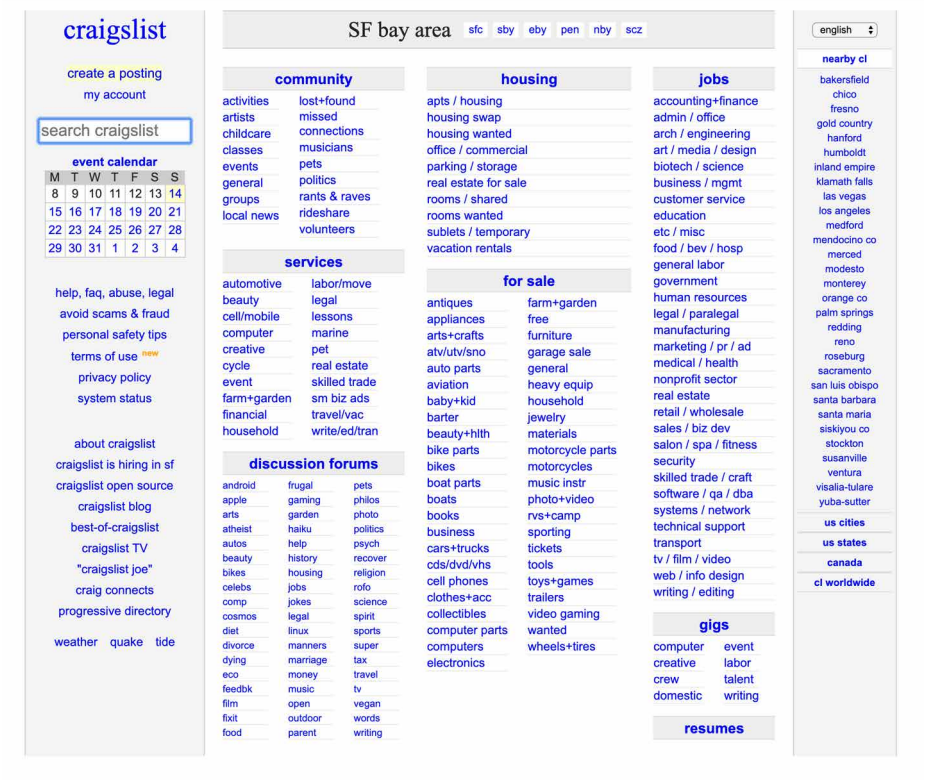


Figure 3-13. *Craigslist*

Use when

You're designing a home page, starting screen, or any other screen whose purpose is to be just a "table of contents"–to show where users can go from here. You might not have room for featured content (such as an article, video, or promotion), or you might simply want to let the user pick a link with no distractions.

Mobile apps and sites often use compact, one-column *Menu Pages* to make the best use of their small screens when they need to display many navigation options.

If your (full-size) site needs to "hook" visitors into staying on the page, it might be better to use some of the page space for promotional items or other interesting content, and a *Menu Page* wouldn't be the best design choice. Likewise, a site that needs to explain its value and purpose should use the space to do that, instead.

It takes some courage to design a *Menu Page* because you must be very confident of the following:

- Visitors know what the site or app is about
- They know what they came for and how to find it
- They are searching for a particular topic or destination and want to locate it as quickly as possible
- They wouldn't be interested in news, updates, or features
- They won't be confused or repelled by the density of your menu page design

Why

With no distractions, users can focus all of their attention on the available navigation options. You get the entire screen (or most of it, anyway) to organize, explain, and illustrate those links, and can thus direct users to the most appropriate destination page for their needs (Figure 3-13).

How

If you're creating a mobile design, *Menu Page* is one of your principal tools for designing sites or apps with many levels of functionality. Keep list labels short, make targets large enough to tap easily (for touch screens), and try not to make hierarchies too deep.

A word of caution regarding menu pages: It is very easy for these to become overwhelming. Consider using them for seldom-used screens such as reference or index pages. In all cases, look for ways to align, group, and label content for easier comprehension.

The remainder of the discussion here applies to full-sized sites and apps.

First, label the links well and provide just enough contextual information for users to decide where to go. This isn't necessarily easy. Visitors might find it very helpful to have a description or teaser with each link, but that could take up a lot of space on the page. Likewise for thumbnail images—they can look great, but how much value do they add?

Look at the San Francisco city government directory screen in Figure 3-14 and the University of California, Berkeley directory screen in Figure 3-15. Visitors to the SF.gov Departments page are given just that: an alphabetical list. Visitors to the UC, Berkeley site already know the meanings of these links—they're the names of academic programs—so the extra information is about the degree offerings, not the definition of the school. The designer is thus able to pack in more links toward the top of the screen. The result is an information-dense, useful page.

On the other hand, the articles in the AIGA resources page (Figure 3-16) do benefit from descriptive text and images. The titles alone aren't necessarily enough to persuade a visitor to click through. (Keep in mind, too, that a user who clicks through and finds that the destination page isn't what they wanted will become frustrated quickly. Make sure your descriptions are accurate and fair!)

Second, consider the visual organization of the list of links. Do they come in categories, or perhaps a two- or three-level hierarchy? Is it ordered by date? Express that organizational scheme in the list.

Third, don't forget a search box.

Finally, reconsider whether you have anything else to say on this page. Home page space, in particular, is quite valuable for drawing in users. Is there an interesting article teaser you can put there? A work of visual art? If such things would annoy more than intrigue, continue designing a pure *Menu Page*.

Examples

The website for the City and County of San Francisco, SF.gov (still being developed as of this writing), takes a portal approach (Figure 3-14). It brings together links to all of the services and departments. The implied majority use case here is that people are searching for a particular department or service within this large governmental organization.

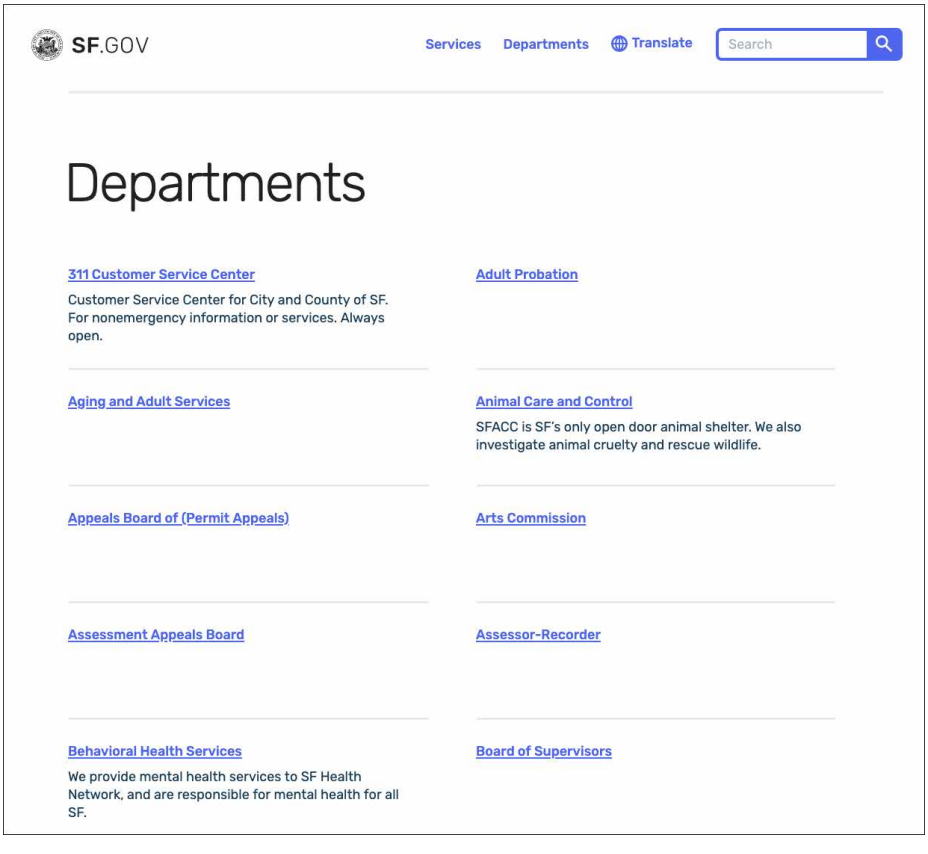


Figure 3-14. *SF.gov*

In the website for the University of California, Berkeley (Figure 3-15), the "Academics" page shows a list of links. When a user reaches this point in the website, they're probably looking for a specific department or resource, not for, say, an explanation of what Berkeley is about. The whole point of this page is to move the visitor along to a page that answers a well-defined need.

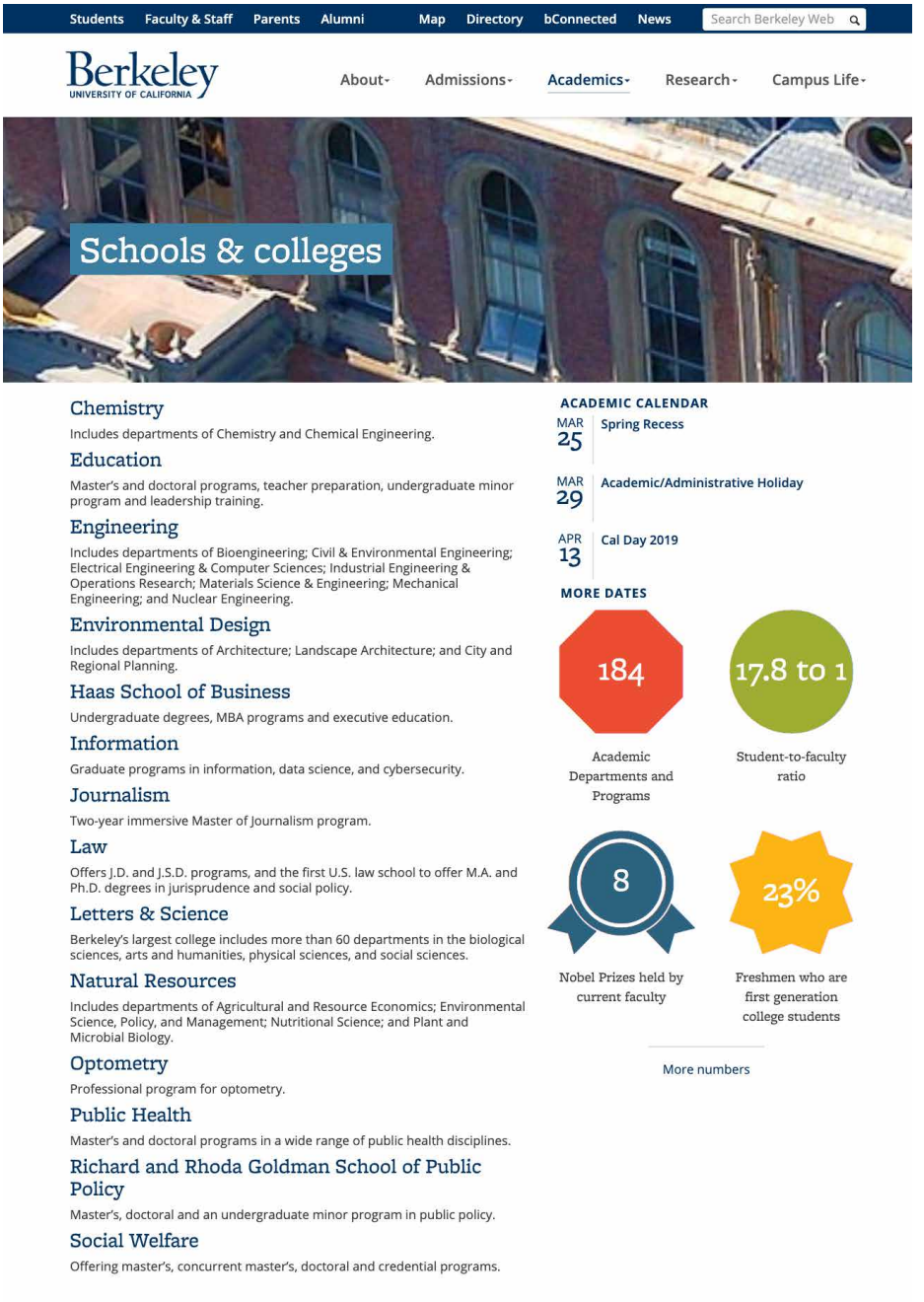


Figure 3-15. *The University of California, Berkeley schools and colleges menu page*

The AIGA website contains many resources for design professionals. The site presents several top-level categories for those resources, as shown in the global navigation, but the landing page for each of those categories is a *Menu Page* (Figure 3-16). The articles are shown with thumbnail images and summary text; the rich format gives the viewer enough of a context to decide whether to invest time in clicking through to the article.

This page is intriguing enough to hook a user on its own, without featuring any particular content at all.

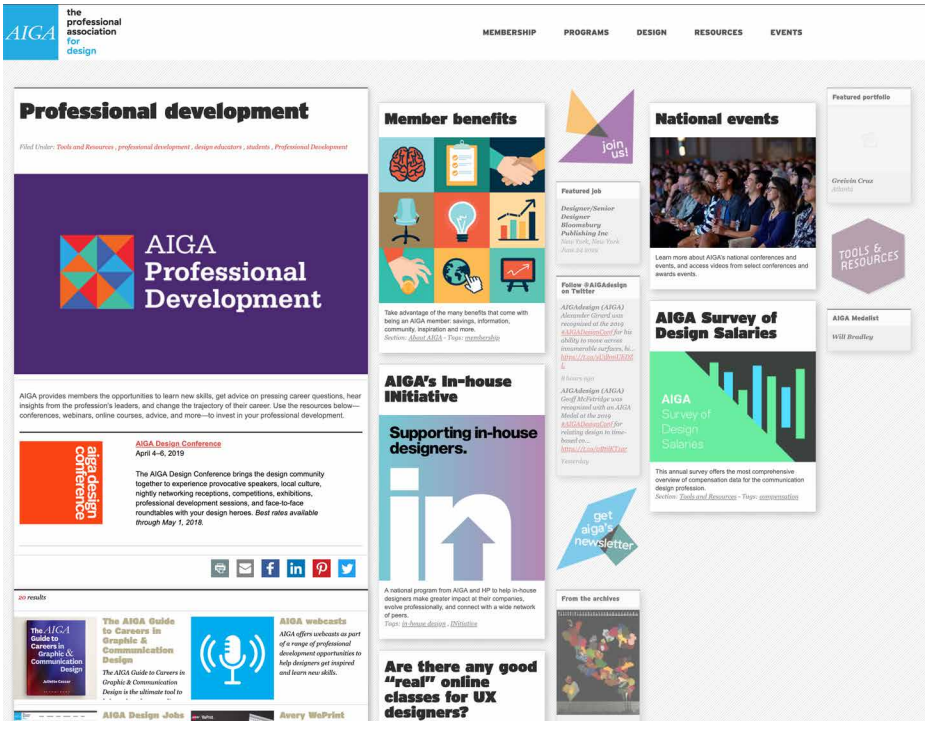


Figure 3-16. *A Menu Page from AIGA's website*

Last, the Museum of Modern Art uses large images and small text on this version of *Menu Page* (Figure 3-17).

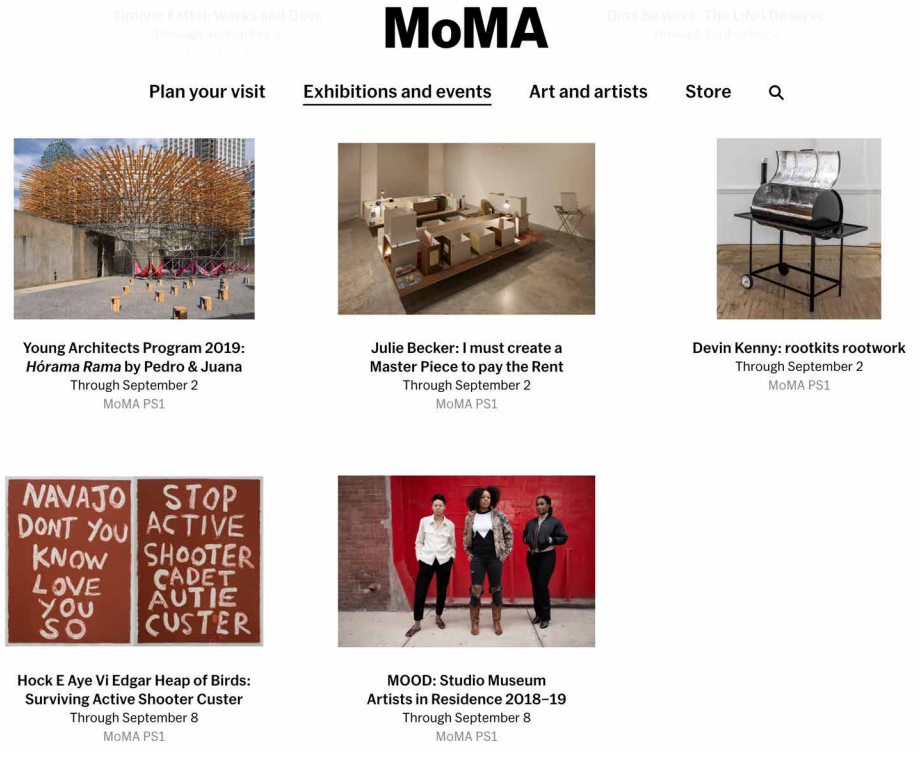


Figure 3-17. *The Museum of Modern Art, New York PS1 exhibit menu page*

Pyramid

What

Link together a sequence of pages with Back/Next links. Create a parent page that links to all of the pages in this sequence, and let the user view them either in sequence or out of order. Figure 3-18 presents this pattern schematically.

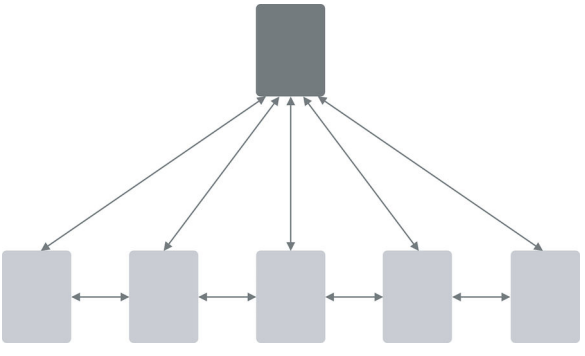


Figure 3-18. *Pyramid schematic*

Use when

The site or application contains a sequence of items that a user would normally view one after another, such as a slideshow, a wizard, chapters in a book, or a set of products. Some users would rather view them one at a time and out of order, however, and they need to be able to pick from a full list of the items.

Why

This pattern reduces the number of clicks it takes to get around. It improves navigation efficiency, and it expresses a sequential relationship among the pages.

Back/Next (or Previous/Next) links or buttons are all well and good. People know what to do with them. But a user doesn't necessarily want to be locked into a page sequence that they can't easily get out of: having gone seven pages in, will they need to click the Back button seven times to get back where they started? That is a recipe for frustration and low usability (lack of user control).

By putting a link back to the parent page on each sequence page, you increase the user's options. You've now have three main navigation options instead of two—Back, Next, and Up. You haven't made it much more complex, but a user who is casually browsing (or one who's changed his mind in midstream) will need far fewer clicks to go where they want to go. It's more convenient for users.

Likewise, chaining together a set of unconnected pages is kind to users who actually want to see all the pages. Without the Back/Next links, they would be "pogo sticking" to the parent page all the time; they might just give up and leave.

How

List all of the items or pages, in order, on the parent page. Render the list in a way that suits the types of items you're dealing with (Chapter 7), such as a *Thumbnail Grid* for photos or a rich text list for articles. A click on an item or link brings the user to that item's page.

On each item page, put Back/Next links. Many sites show a small preview of the next item, such as its title or a thumbnail. In addition, put in an Up or Cancel link to bring the user back to the parent page.

One *Pyramid* variation turns a static linear sequence into a loop by linking the last page back to the first without going back to the parent. This can work, but does the user know that they've looped all the way back around? Do they recognize the first page in the sequence? Not necessarily. If the order of a sequence is important, you should link the last page to the parent page; this signals the user that they've seen all there is to see.

Examples

Facebook's photo album page is a classic *Pyramid* example.The album can be seen in its entirety by scrolling the page (see Figure 3-19). The images are thumbnails. Selecting a photo opens the slideshow, which is organized via the *Pyramid* pattern (Figure 3-20). Scroll right, scroll left, or exit to the grid again are the navigation options.

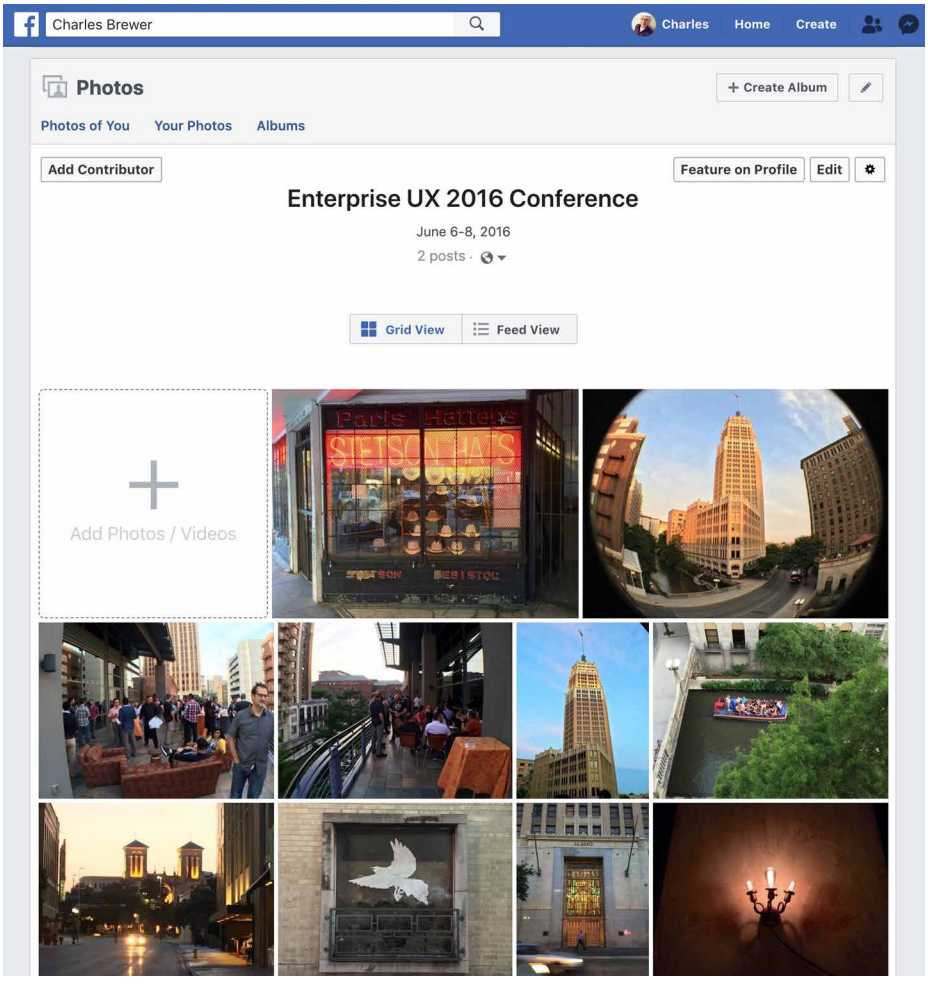


Figure 3-19. *A Facebook photo album*

Figure 3-20. *A child page from the same Facebook feature, showing Back, Next, and Close buttons near the photo*

Modal Panel

What

A screen with no navigation options other than acknowledging its message, completing its form, or clicking the panel away. Modals appear on top of the current screen. Modals are usually invoked by a user action. This can be selecting something or performing some triggering action. Modal panels often show up in a "lightbox" on top of a full screen or page: The screen underneath is visible but everything except the modal is behind a gray layer and is not accessible. This is used for small, focused tasks that require the user's full attention. Modals usually consist of one page, with no other navigation options, until the user finishes the immediate task. Figure 3-21 illustrates the schematic for the *Modal Panel* pattern.

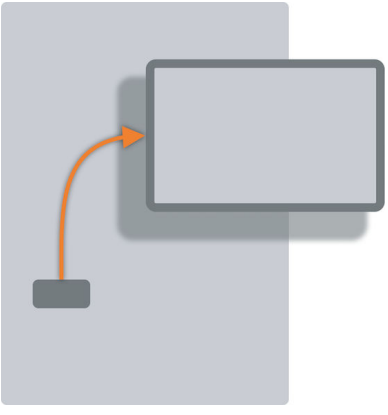


Figure 3-21. *The Modal Panel schematic*

Use when

Modals are great for focusing on a single action or process. They are also great for not losing context while carrying out a quick subtask or detour from a main screen or action, where you want the user to take care of a small task and then get back to the bigger task. They can also work well when the app or site has gotten into a state from which it shouldn't or can't proceed without input from the user. In a document-centric application, for instance, a "save" action might need the user to supply a file-name if one wasn't already given. In other contexts, the user might need to sign in before proceeding, or acknowledge an important message.

If the user simply initiates a minor action that might need further input, try to find a way to ask for that input without a modal panel. You could show a text field right below the button that the user clicked, for example, and leave it "hanging" there until the user comes back to it—there's no need to hold up the entire site or app until that input is given. Let the user do something else and then return to the question at a later time.

Why

A modal panel cuts off all other navigation options from the user. They can't ignore it and go somewhere else in the app or site: they must deal with it here and now. When that's done, the user is sent back to where they were before.

It's an easy model to understand—and to program—though it was overused in applications of past years. A modal panel is disruptive. If the user isn't prepared to answer whatever the modal panel asks, it interrupts their workflow, possibly forcing them to make a decision about something they just don't care about. But when used

appropriately, a modal panel channels the user's attention into the next decision that they need to make. There are no other navigation possibilities to distract the user.

How

In the same space on the screen where the user's attention lies, place a panel, dialog box, or page that requests the needed information. It should prevent the user from bringing up other pages in that application. This panel ought to be relatively uncluttered, in keeping with the need to focus the user's attention onto this new task with minimal distractions.

Remember that this is a navigation-related pattern. You should carefully mark and label the ways out, and there shouldn't be many of them; one, two, or maybe three. In most cases, they are buttons with short, verbish labels, such as "Save" or "Don't save." There is usually a "Close" or "X" button in the upper right. Upon clicking a button, the user should be taken back to the page they came from.

The lightbox effect is a very effective visual presentation of a modal panel. By dimming most of the screen, the designer highlights the bright modal panel and focuses attention on it. (For this to work, the modal panel needs to be large enough for the user to find it effortlessly.)

Some websites use modals for sign-in and registration screens. Retail and other sites that want to prompt sign in/registration only when truly needed (to avoid interrupting the user) are commonly done this way: global and local navigation are stripped out, and all that's left are to perform the sign-in task or exit.

Operating systems and graphical user interface (GUI) platforms usually offer OS-level modal dialog boxes. These are best used in traditional desktop applications—websites should avoid them in favor of lighter-weight overlay techniques, which are easier for the designer to control and less disruptive to the user. OS-level modals usually freeze the UI except for the modal window.

Examples

Airbnb uses a lightbox to draw attention to its login (Figure 3-22). This appears directly over the public website landing page. There are only three ways to deal with it: sign in, register, or click the familiar "X" button in the upper-left corner. This is typical of many lightbox-highlighted modal panels on the web. If Airbnb does not recognize the user's computer (usually because they cleared cookies), the modal panel changes in place to display the two-factor authentication screen.

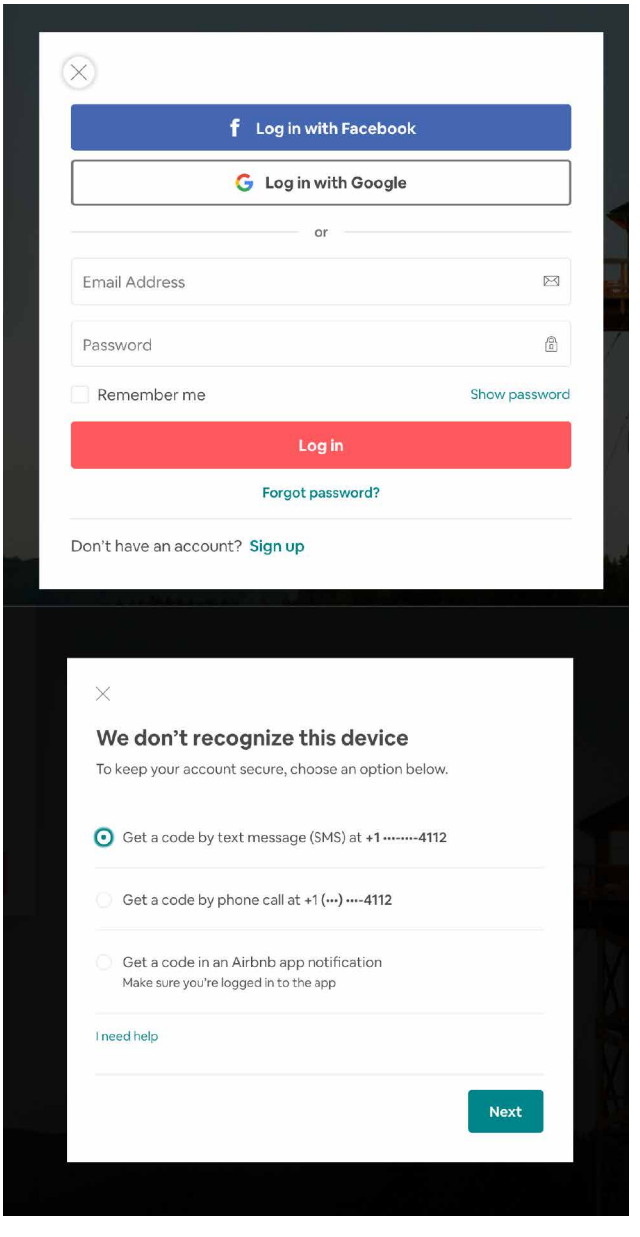


Figure 3-22. *Airbnb login modal panel and security check modal panel*

As mentioned earlier, retailers often delay any sign-in or register step until the last possible moment in the shopping process in order to make the purchase process as uninterrupted as possible. After the user is in the shopping cart, however, it makes sense to require them to sign in. Registered shoppers need to activate their shipping and payment details, and the retailer would like to invite new customers to become registered. B&H Photo is a good example of this pattern (Figure 3-23).

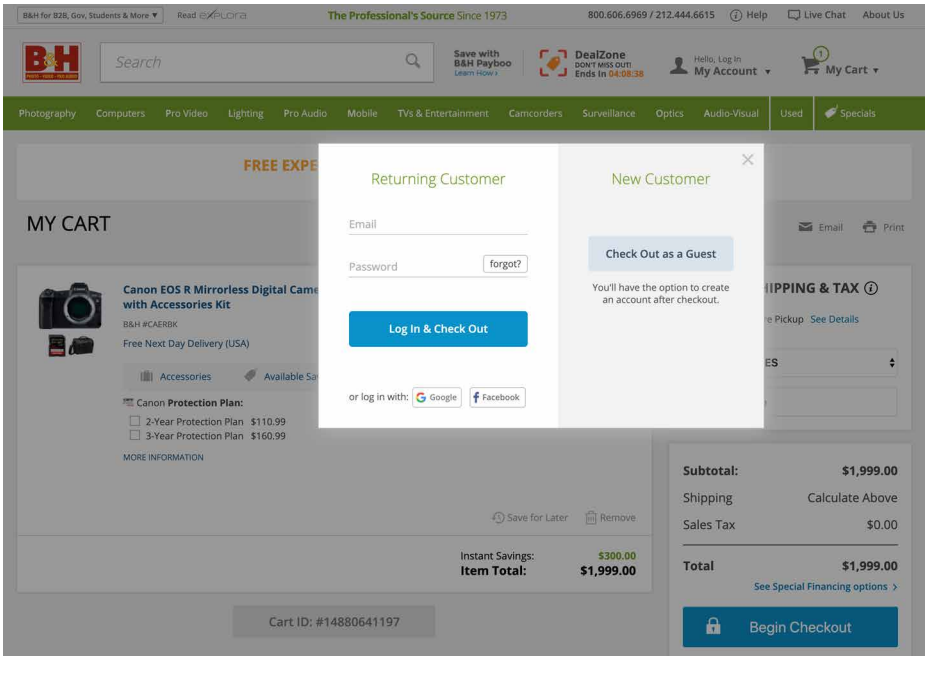


Figure 3-23. *B&H checkout log in modal*

Macy's uses a modal window earlier in the shopping process (Figure 3-24). In its case, it confirms for the shopper that their selected item has been added to their shopping bag for purchase. Macy's takes advantage of this moment to offer additional items that the shopper might be interested in.

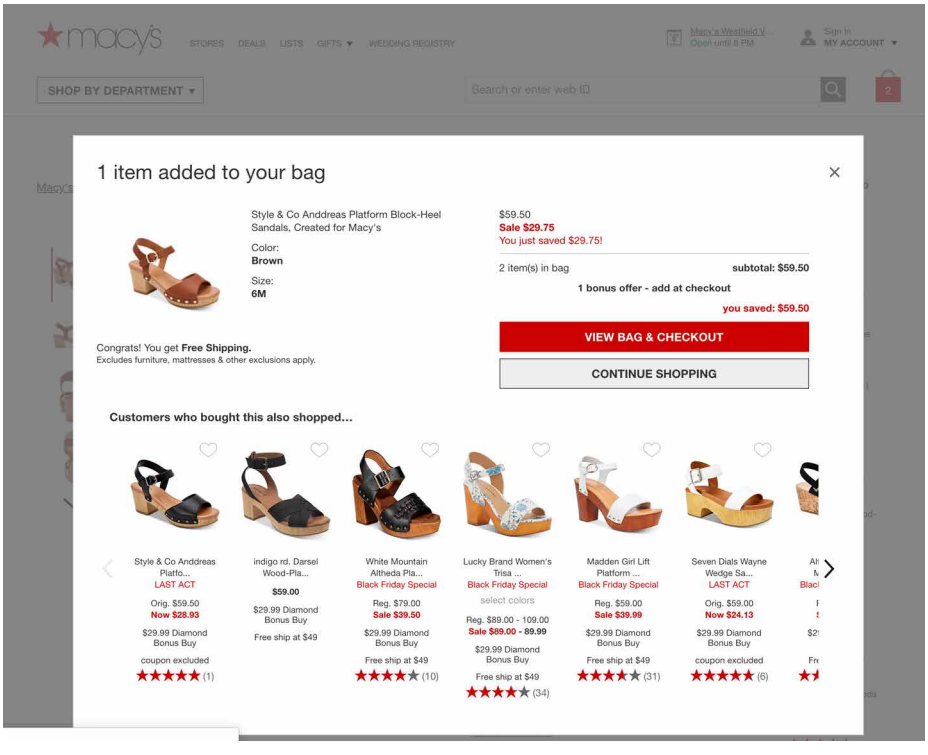


Figure 3-24. *Macy's "Put in Bag" confirmation modal*

Priceline uses a modal to respond to a traveler's lack of activity in a clever way (Figure 3-25). If the customer has searched for a flight or hotel, but doesn't take any further action from the search results page, it's possible they've switched to another task or website or have stepped away. Priceline seeks to reengage with the customer, so after a short time, this modal appears, offering to show even more recent, up-to-date results.

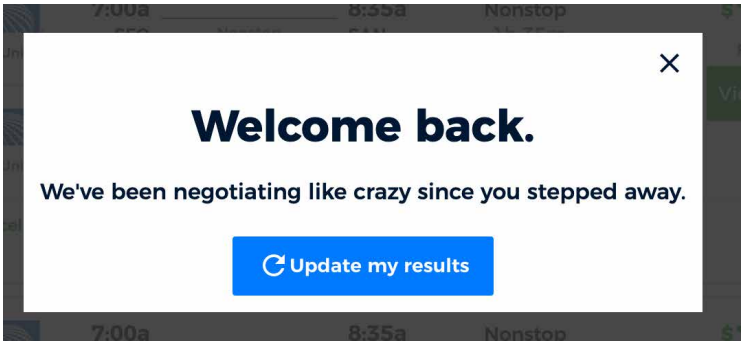


Figure 3-25. *Priceline timeout and reengagement modal*

The "shade" form of a MacOS modal dialog box draws attention to itself as it drops down from the window title bar (animated, of course). These and other application-level modal dialog boxes actually prevent the user from interacting with the rest of the application; thus, the user is forced to finish or dismiss this thread of work before doing anything else (Figure 3-26).

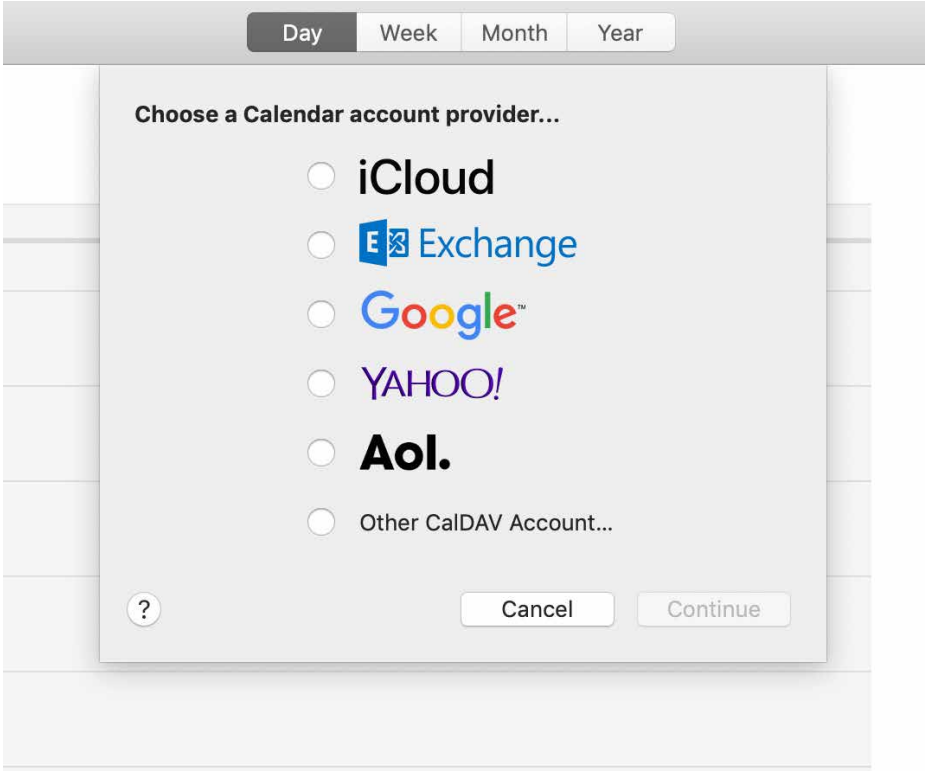


Figure 3-26. *A modal panel in a Mac application*

Deep Links

What

Capture the state of a site or app in a URL or other link that can be saved or sent to other people. When loaded, it restores the state of the app to what the user was seeing. In other words, a deep link should be a method for linking to both a location in your software and also a state, such as being signed in or resuming an incomplete process at the point where the user left off, or retaining information so that the user doesn't need to reenter it. Such bookmarks, permalinks, and deep links are all ways for a user to conveniently navigate to a selected point or state, even if it's deep within a navigational structure. This avoids traversing many links to get to a desired page or state. Mobile OS deep links are a particular method for allowing users to go from app to app on their mobile device without losing information or task context. Figure 3-27 shows how this works schematically.

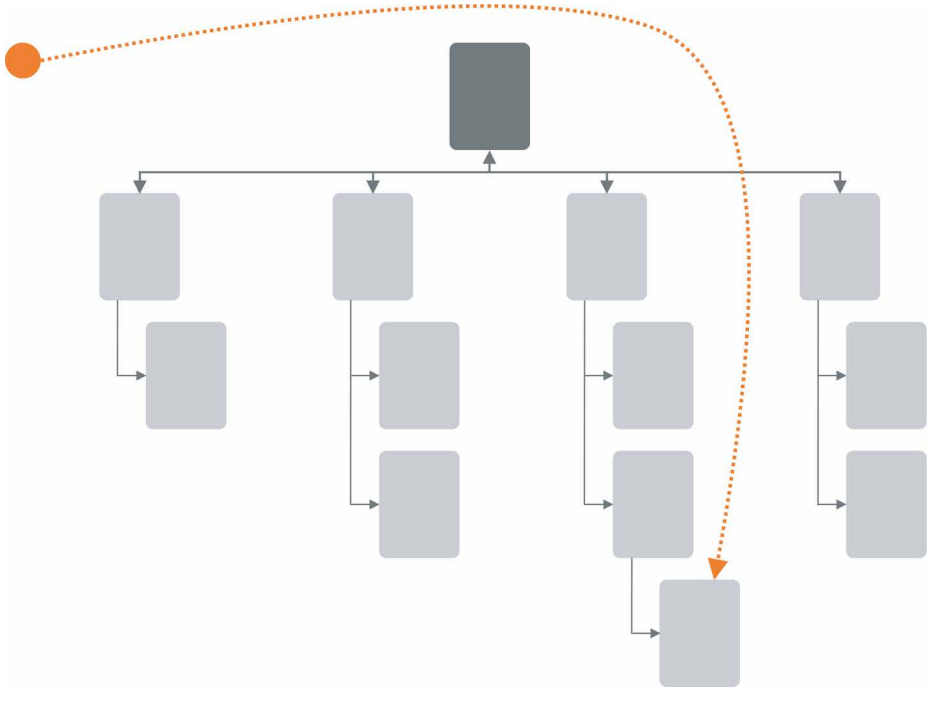


Figure 3-27. *Deep Links state schematic*

Use when

The site or app's content is something specific and interactive, such as a map location and zoom level, book page, video clip, or information graphic. The deep-link sender wants to include a specific desired point or state that might be difficult to find otherwise, or it might take many steps to get there from a typical starting point. The app might have many user-settable parameters or states, such as viewing modes, scales, data layers, and so on—these can add to the complexity of finding a particular point and seeing it in the "right" way.

Why

Deep Links gives the user a way to jump directly to a desired point and application state, thus saving time and work. It behaves like a "deep link" directly into a piece of content on a conventional site—or a permalink to a blog entry—in the sense that you end up with a URL pointing directly to the desired content. But it can be more complex than a permalink, because it can capture both application state and content position.

This pattern is useful for saving a state that the user might want to recreate later, especially if they can "bookmark" it using well-known mechanisms (like browser bookmarks). It's also handy for sharing with other people, and that's where it really shines. A URL representing a deep-linked state can be emailed, tweeted, posted to a social network, discussed in a forum, published in a blog entry, and talked about in any number of ways. It might make a statement, or go viral, or become a "socially mediated object."

How

Track the user's position in the content, and put that into a URL. Track supporting data there as well—comments, data layers, markers, highlighting, and so on—so that reloading the URL will bring it all back.

Consider what other parameters or interface states you might want users to save: zoom levels, magnification, viewing modes, search results, and so on. Not all of these should necessarily be captured, because loading the deep-linked state shouldn't trample on settings that a user doesn't want changed. Work carefully through some usage scenarios to figure this out.

URLs are the best format for saving *Deep Links*: they are universally understood, portable, short, and supported by a vast variety of tools, such as bookmarking services.

As a user moves through the content and changes various parameters, immediately put the updated URL in the browser's URL field so that it can be easily seen and ultimately copied and shared. Not everyone will think to find it there, so you might also design a "Link" feature whose existence informs the user, "Here's how you create a link to this screen." Some sites offer to generate a JavaScript "Embed" fragment that not only captures position and state, but also lets users embed the entire thing into another website.

Deep links in the mobile world have a specific meaning. Both iOS and Android applications can be configured so that public URLs map to corresponding "locations" in the native OS mobile application (rather than the public URL in the mobile browser). This allows any shared links to launch their associated mobile app, with its (usually) more robust controls and performance. Mobile-native applications can also pass deep links from one application to another. For example, the IMDB app might host a link to a movie trailer on a website. Instead of opening the mobile web browser, the link is passed to, say, the YouTube native app on the user's device, which offers more control over playback and interaction.

Examples

One of the nicest features of sharing videos from YouTube is the ability to embed a starting point for the clip directly in the share link (Figure 3-28). YouTube sharing includes the ability to specify starting point in a video. Recipient's video playback will begin here, not at the beginning of the shared video.

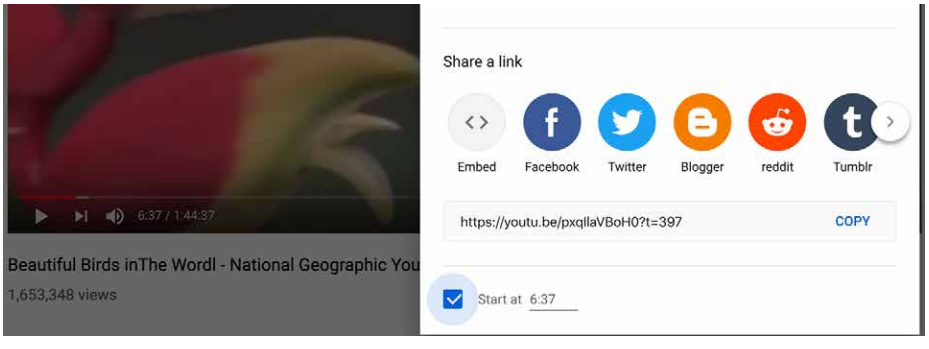


Figure 3-28. *Sharing a YouTube video*

Google Books captures a large amount of state in its URLs (Figure 3-29): the position in the book, the viewing mode (single page, two-up, thumbnails), the presence of toolbars, and even search results. It does not capture magnification level, which makes sense, because that's a very individual setting. The URL as seen in the "Link" tool is actually redundant—the URL shown by the browser itself is exactly the same.

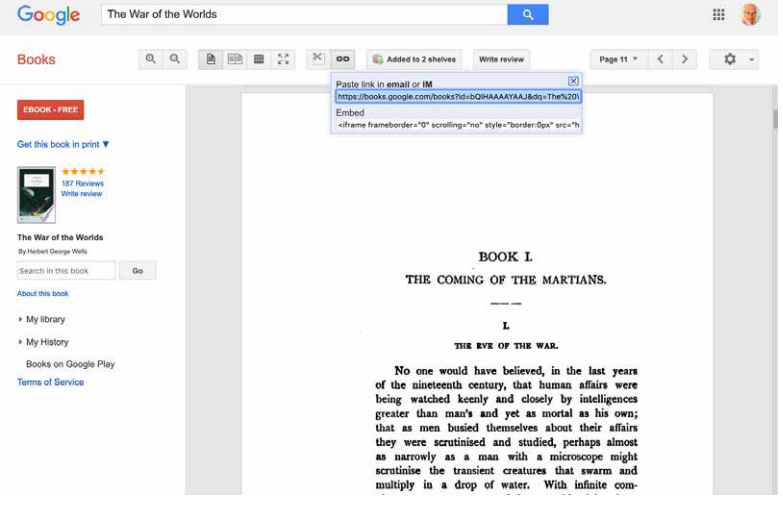


Figure 3-29. *Deep-linked state in Google Books, found in two places: the browser's URL field, and the "Link" feature*

In Apple iOS, the operating system itself checks public URLs against the deep link configuration of all installed native applications. This allows a pass-off from the mobile browser so that the user can view the selected page, song, stream, or video in an installed app (Figure 3-30), and not through the mobile browser, which might offer limited functionality for the destination link. Rerouting to the device-resident app lets the user enjoy more functionality and a more robust viewing experience.

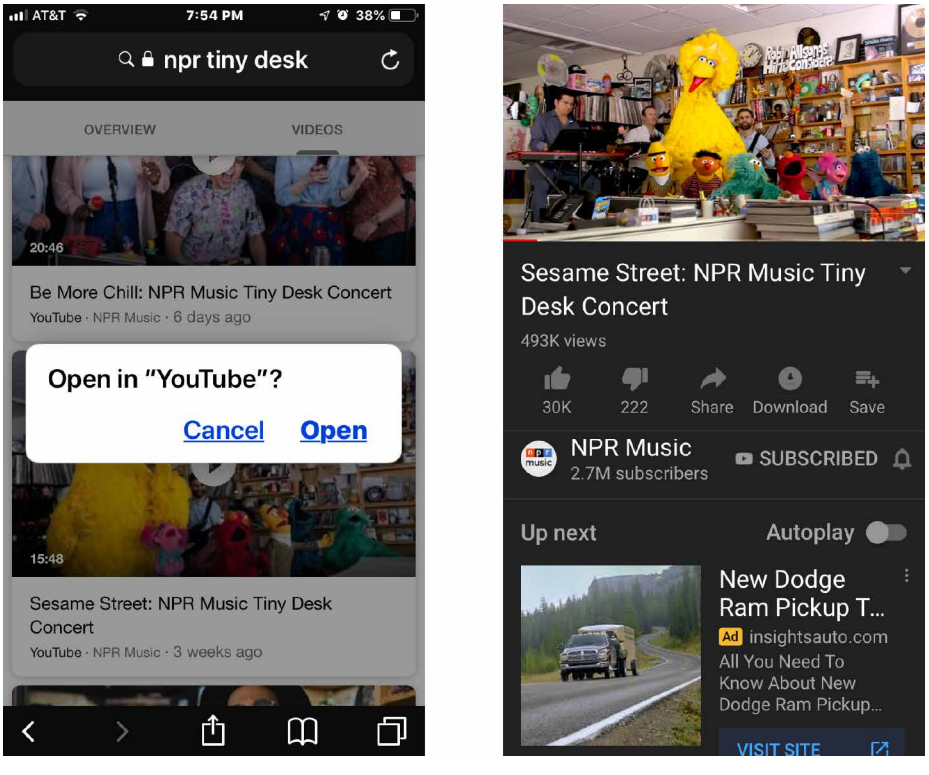


Figure 3-30. *iOS; Deep linking from mobile web to mobile app*

Job listings site Indeed.com has robust searching and filtering tools for job searchers. These parameters are written to the URL, allowing the search to be shared or saved for later, to trigger a refreshed search (Figure 3-31).

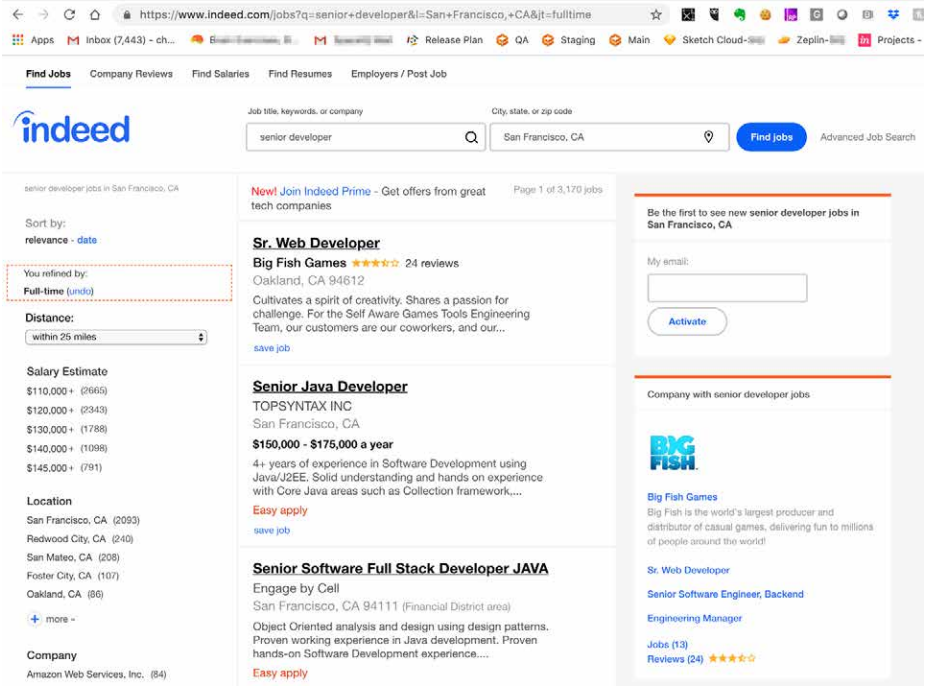


Figure 3-31. *Indeed job search; parameters are written in the URL so that this search can be shared or saved*

Escape Hatch

What

A well-labeled button or link that clearly gets the user out of their current screen and back to a known place. Use these on screens that have limited navigation options. Also use escape hatches for when a user is hopelessly entangled in an app, reaches an error state, or becomes deep-linked into a page that they have no context for understanding. The schematic in Figure 3-32 illustrates this concept.

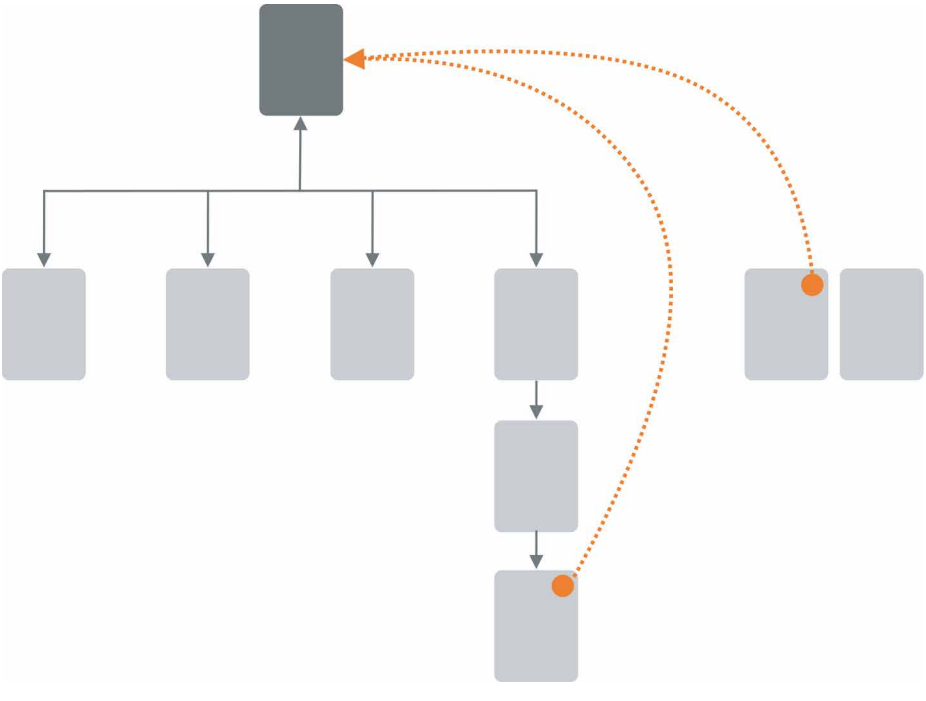


Figure 3-32. *Escape Hatch schematic*

Use when

You have pages that constitute some sort of serial process, such as a wizard, or any pages that lock the user into a limited navigation situation, such as a *Modal Panel*. These might also be pages that users can reach out of context, as they could do via search results.

There are also dead-end screens. For example, HTTP server error screens, such as for Error 404 Page Not Found (there are many of this type of error screens), are a great place to put an escape hatch.

Why

Limited navigation is one thing, but having no way out is quite another! If you give the user a simple, obvious way to escape from a page, no strings attached, they're less likely to feel trapped there. It also prevents people from bailing out completely by closing the application completely.

This is the kind of feature that helps people feel like they can safely explore an app or site. It's sort of like an undo feature—it encourages people to go down paths without feeling like they're committing to them. (See the *Safe Exploration* pattern in Chapter 1.)

Now, if these are pages that users can reach via search results, it's doubly important that escape hatches be put on each page. Visitors can click these to get to a "normal" page that tells them more about where they actually are.

How

Put a button or link on the page that brings the user back to a "safe place." This might be a home page, a hub page in a hub-and-spoke design, or any page with full navigation and something self-explanatory on it. Exactly what it links to will depend upon the application's design.

Examples

Websites often use clickable site logos as home-page links, usually in the upper left of a page. These provide an *Escape Hatch* in a familiar place while helping with branding.

In some dialogs, a Cancel button or the equivalent can serve this purpose. These also let the user say, "I'm done with this; forget I ever started it."

Have you ever called a company—for instance, your bank—and had to work your way through a set of phone menus? They can be long, confusing, and time-consuming. If you find yourself in the wrong menu, you might just hang up and try again from the top. But many phone menu systems have a hidden *Escape Hatch* that they don't tell you about: if you dial "0" at any point, you might be connected to a human operator. Many customers go directly to this hidden shortcut.

Many websites have certain pages that limit navigation options, such as *Modal Panel* and pages without global navigation. The LinkedIn Settings screen is one example. This section of LinkedIn is separate from the main web application. The global navigation is not present. If a user finds themselves here, there are two ways to get back, through two escape hatches. The first is the LinkedIn logo to go back to the home page. The second is a "Go Back to LinkedIn.com" link with the member's own profile picture (see Figure 3-33).

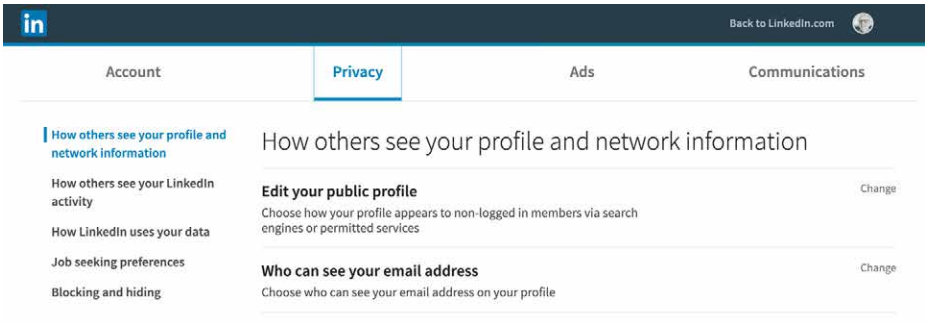


Figure 3-33. *The LinkedIn Settings page, with link and avatar in the upper right as an escape hatch back to LinkedIn*

Helping browsers recover from dead ends is a good use of escape hatches, too. Curbed.com's website offers an escape hatch on its 404 Not found error screens. In the copy is a link to jump to the home page (Figure 3-34). Curbed also offers system status messaging, so if the Curbed website is actually not active, the user would know that.

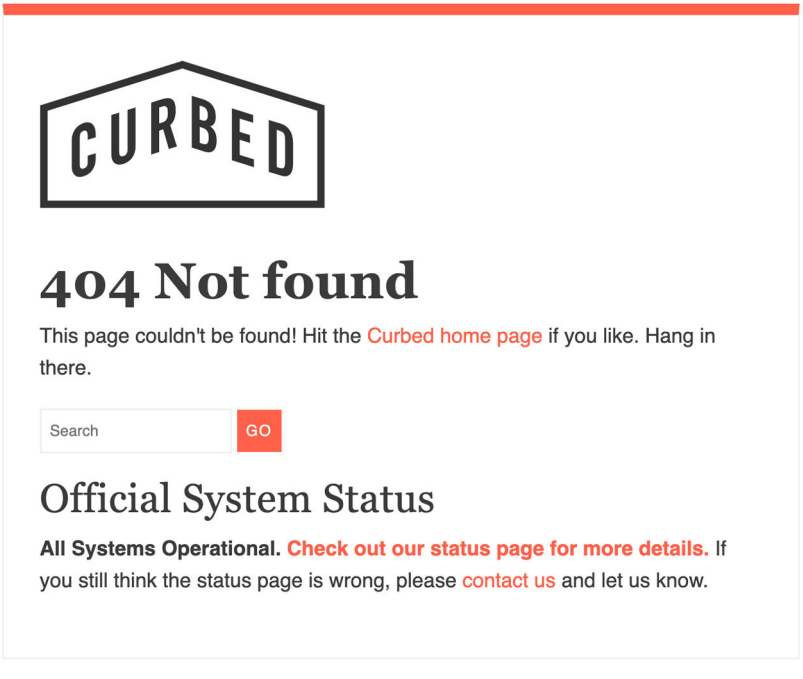


Figure 3-34. *Curbed.com 404 error page with an escape hatch to the home page*

Fat Menus

What

Display a long list of navigation options in drop-down or fly-out menus. Also called "mega-menus." Use these to show all of the subpages in site sections. Organize them with care, using well-chosen categories or a natural sorting order, and spread them out horizontally. You can find an example of this pattern in the "All Microsoft" fat menu on *Microsoft.com* (Figure 3-35).

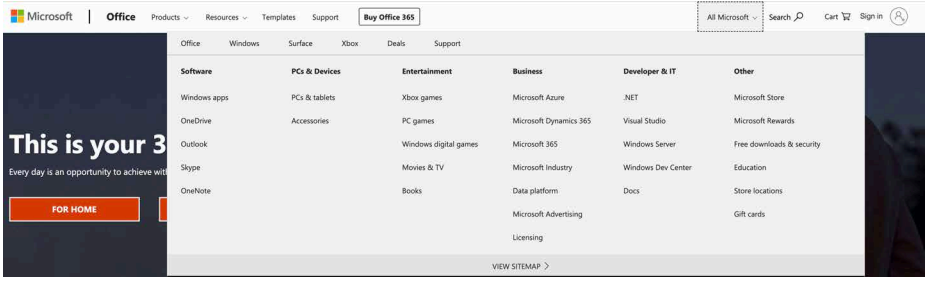


Figure 3-35. *Microsoft's All Microsoft menu*

Use when

The site or app has many pages in many categories, possibly in a hierarchy with three or more levels. You want to expose most of these pages to people casually exploring the site, so they can see what's available. Your users are comfortable with drop-down menus (click to see them) or fly-outs (hover over them with the pointer).

Why

Fat Menus makes a complex site more discoverable. They expose many more navigation options to visitors than they might otherwise find.

By showing so many links on every page, you make it possible for a user to jump directly from any subpage to any other subpage (for most subpages, anyhow). You thus turn a multilevel site—where subpages aren't linked to the subpages in other site sections—into a fully connected site (Figure 3-35).

Fat Menus are a form of progressive disclosure, an important concept in UI design. Complexity is hidden until the user asks to see it. A visitor to a site that uses these can look over the menu headings to get a high-level idea of what's there, and when that user is ready to dive in, they can open up a *Fat Menu* with a gesture. The user isn't shown millions of subpages before they're ready to deal with them.

If you're already using menus in your global navigation, you might consider expanding them to *Fat Menus* if showing more links makes the content more attractive to casual browsers. People won't need to drill down into categories and subcategories of your site hierarchy in order to discover interesting pages—they'll see them there, right up front.

How

On each menu, present a well-organized list of links. Arrange them into *Titled Sections* (Chapter 4) if they fit into subcategories; if not, use a sorting order that suits the nature of the content, such as an alphabetical or time-based list.

Use headers, dividers, generous whitespace, modest graphic elements, and whatever else you need to visually organize those links. And take advantage of horizontal space—you can spread the menu across the entire page if you want. Many sites make excellent use of multiple columns to present categories. If you make the menu too tall, it might go right off the end of the browser page.

The best sites have *Fat Menus* that work stylistically with the rest of the site. Design them to fit well into the color scheme, grid, and so on of the page.

Some menu implementations don't work well with accessibility technology such as screen readers. Ensure that your *Fat Menus* can work with these. If they can't, consider switching to a more static strategy, such as a *Sitemap Footer*.

You can adapt *Fat Menus* for mobile screens if necessary. In that case, you should linearize the columnar layout left to right. That is, the menu is reordered into a single column, with the sections stacked vertically. It's best not to insert this much navigation content into every mobile screen. Instead, consider making this a reference navigation screen that is accessed through a separate mobile navigation scheme.

Examples

Macy's, like most big retailers, has a vast inventory with many categories of items for sale. Browsing and finding a specific category or item of interest can be challenging in these cases. Well-designed fat menus can be a useful solution. Macy's uses a two-part fat menu (Figure 3-36). The shopper first opens the top-level fat menu with the level-one major categories. When they select one of these, a second panel opens that covers the page. A huge amount of level-2 categories are displayed in this second panel.

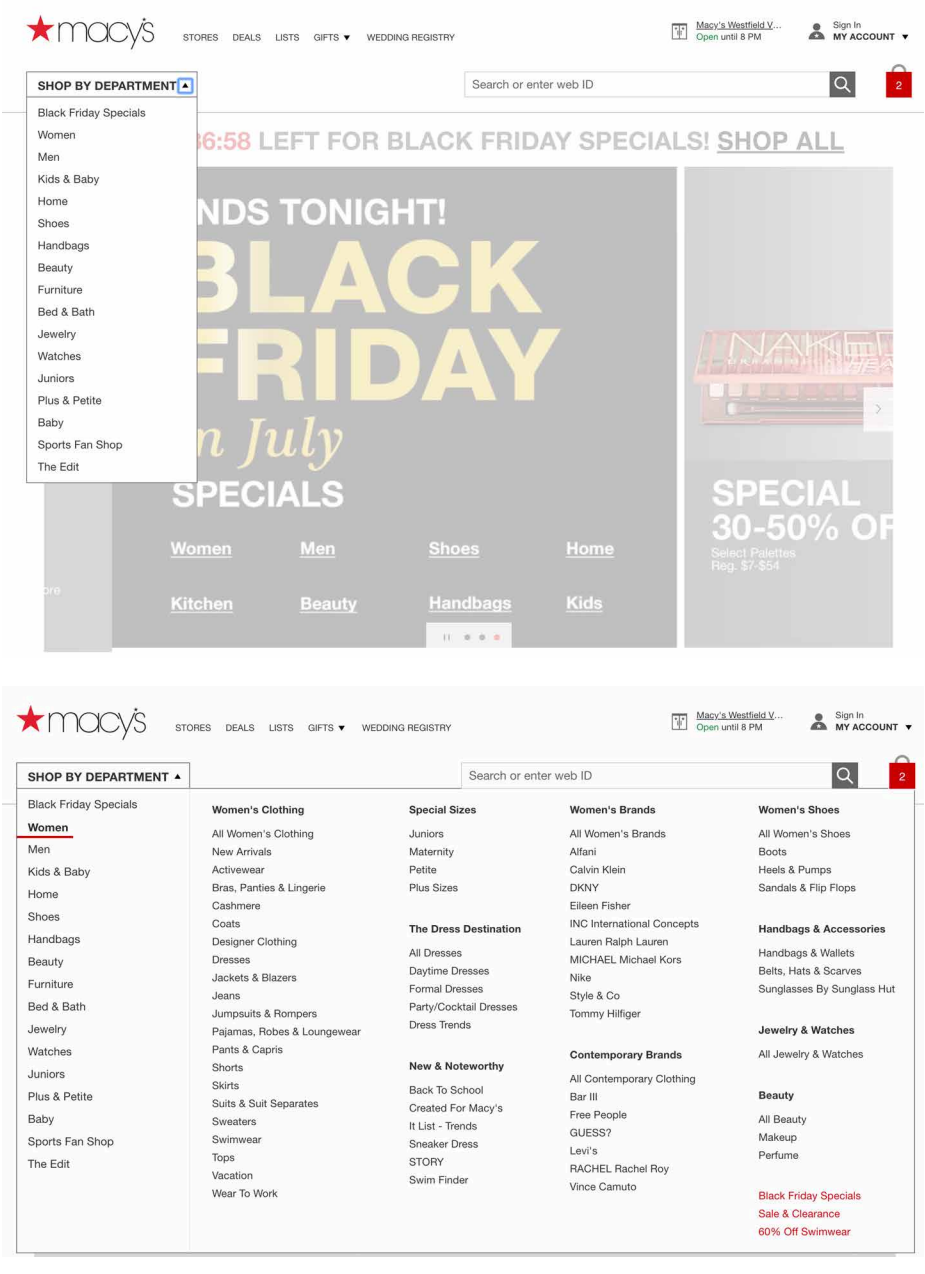


Figure 3-36. *The Macy's two-level fat menu with progressive disclosure*

The *Fat Menus* on the Starbucks website are very well designed (Figure 3-37). Each menu is a different height but the same width and follows a strict common page grid (they're all laid out the same way). The style blends in with the site, and the generous negative space makes it easy to read. Product promotions are worked into the design, but not obnoxiously.

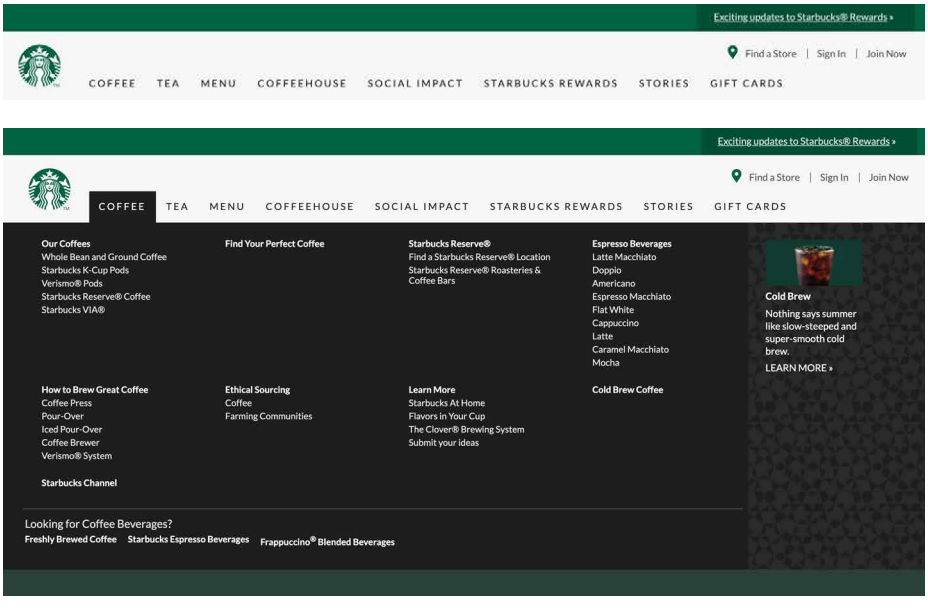


Figure 3-37. *Starbucks coffee menu*

As shown in Figure 3-38, Mashable's fat menus use a hybrid approach. The text menus are off to the left and deemphasized. It takes full advantage of the horizontal space to show featured articles. This is clever—the knowledgeable user can skim a large number of headlines by rolling over the menus.

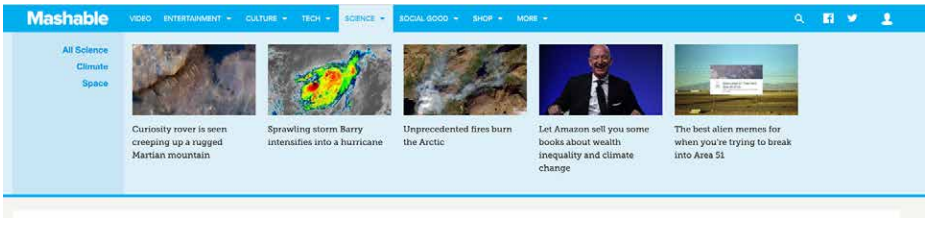


Figure 3-38. *Mashable's Science menu*

The American Red Cross uses *Fat Menus* liberally (Figure 3-39). When the user rolls over any top-level menu item, the resultant *Fat Menus* cover up the top portion of the screen. There is good organization and presentation of topics and links, making this large website structure easily comprehensible. The sections in each *Fat Menu* are organized by most likely questions or use cases.

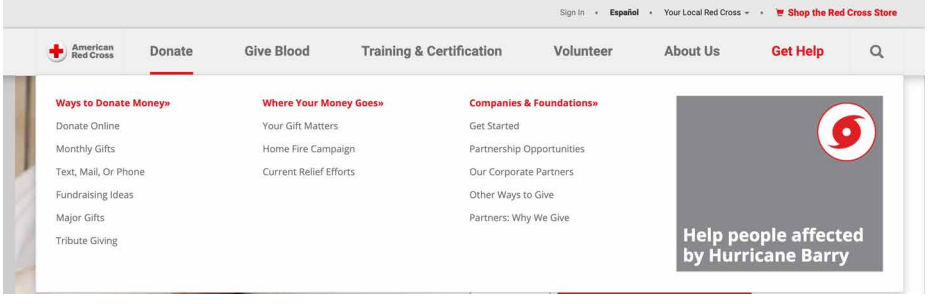


Figure 3-39. *The American Red Cross menu*

WebMD uses an alphabetical sorting order for its list of health topics (Figure 3-40). There is direct access to information on most common conditions, a long list of additional resources, and room for two promoted stories with graphics. The likelihood that the site visitor can find the link they're looking for—and continue to engage—is high.

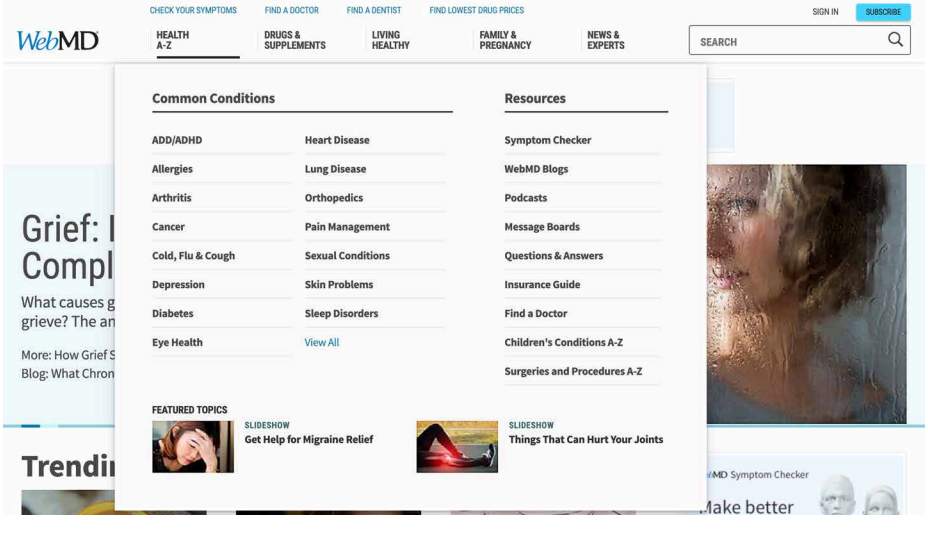


Figure 3-40. *WebMD's Health A–Z menu*

Sitemap Footer

What

A comprehensive directory of links, organized into categories, that provides an at-a-glance review of the full scope of the website, and links to all major sections and pages (Figure 3-41). In other words, the sitemap footer is an index to the website, and could also be a directory to other sites and resources. What is unique about the footer location is that there are no vertical space restrictions, unlike *Fat Menus* at the top of the screen.

Figure 3-41. *Whole Foods footer*

Use when

The site you're designing uses a generous amount of space on each page and you don't have severe constraints on page size or download time. You don't want to take up too much header or sidebar space with navigation.

The site has more than a handful of pages, but not an outrageously large number of categories and "important" pages (things that users will look for). You can fit a reasonably complete site map—at least for pages that aren't in the header—into a strip no taller than about half of a browser window.

There might be a global navigation menu in the page header, but it doesn't show all levels in the site hierarchy—maybe it shows only the top-level categories. You prefer a simple, well-laid-out footer instead of *Fat Menus*, perhaps because of implementation ease or accessibility issues.

Why

Sitemap Footer make a complex site more discoverable. The pattern exposes many more navigation options to visitors than they might otherwise have.

By showing so many links on every page, you make it possible for a user to jump directly from any subpage to any other subpage (or major page, anyhow). You thus turn a multilevel site—where subpages aren't linked to the subpages in other site sections—into a fully connected site. The footer is where the user's attention lands when they read to the end of a page. By placing interesting links there, you entice the user to stay on the site and read more.

Finally, showing users the entire site map gives them a strong sense of how the site is constructed and where they might find relevant features. In complex sites, that could be valuable.

You might find yourself trying to choose between a *Sitemap Footer* design and a *Fat Menus* design. In conventional websites, a *Sitemap Footer* would be easier to implement and debug because it doesn't depend on anything dynamic: instead of showing fly-out menus when the user rolls over items or clicks them, a *Sitemap Footer* is just a set of static links. It's also easier to use with screen readers and it doesn't require fine pointer control, so it wins on accessibility, as well.

On the other hand, the footer might be ignored by busy or casual users who focus only on the page content and the headers. Usability-test if you have any doubts, and watch the click metrics to see if anyone even uses the *Sitemap Footer*.

How

Design a page-wide footer that contains the site's major sections (categories) and their most important subpages. Include utility navigation and tools such as language choice, along with other typical footer information such as copyright and privacy statements.

This might constitute a complete site map for your site, or it might not. The idea is to cover most of what visitors need to find without overloading the header or sidebar navigation. Place a site map into the footer of every page on a site. Treat it as part of the global navigation, complementary to the header.

In practice, what often happens is that the global navigation options at the top of the page reflect a more task-oriented design—it tries to answer visitors' immediate questions regarding "What is this about?" and "Where do I find X right this second?" Meanwhile, the *Sitemap Footer* shows the complete, permanent information architecture of the site itself. This two-part arrangement appears to work well.

If your site deals with content that itself requires complex navigation—such as a large set of products, news articles, music, videos, books, and so on—you could use the top of the page for content navigation and the *Sitemap Footer* for almost everything else.

Here are some features that you can find found in *Sitemap Footer*:

- Major content categories
- Information about the site or organization
- Corporate information, Contact Us, and Careers links
- Partner or sister sites; for example, sites or brands owned by the same company
- Community links such as forums
- Help and support
- Contact information
- Current promotions
- Donation or volunteer information, for nonprofits

Examples

REI's website demonstrates the difference between task-oriented top-of-page global navigation and an effective *Sitemap Footer* (Figure 3-42). Shopping, learning, and travel dominate the header, as they should—these are what most site visitors come for. The footer handles secondary tasks that are nevertheless important: Corporate "about" information, customer support, membership, and so on.

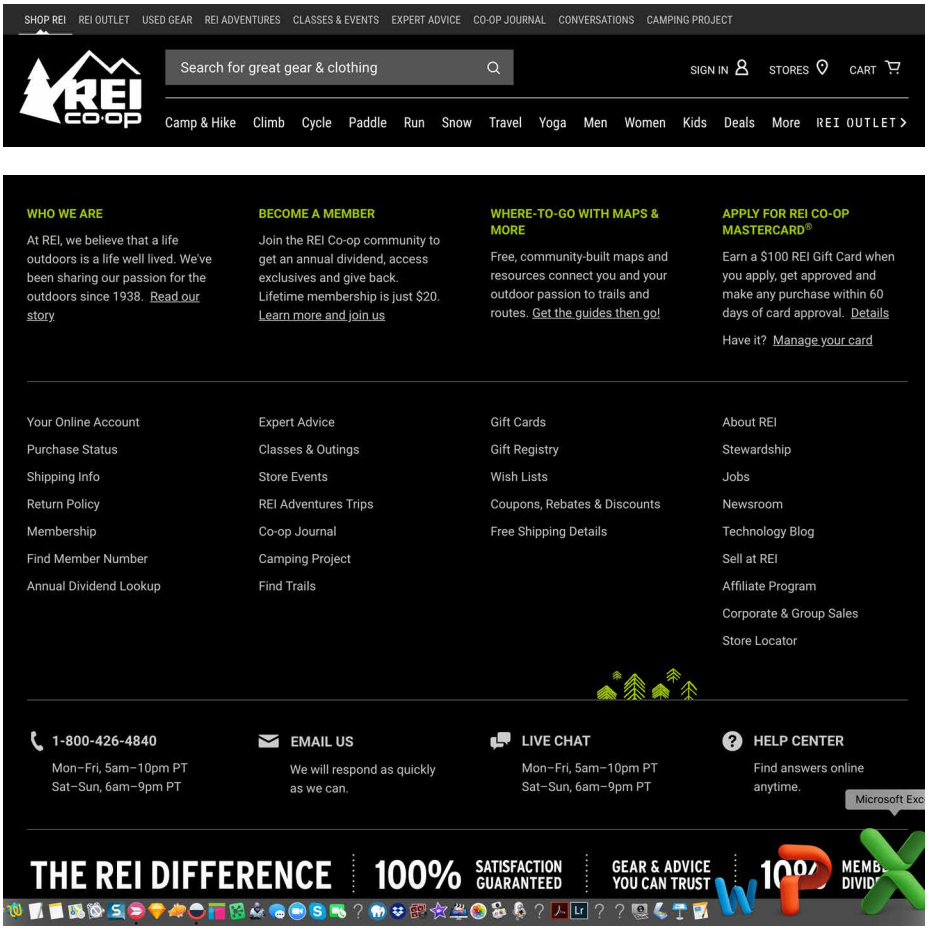


Figure 3-42. *REI header and footer*

The *Los Angeles Times* header and footer shows a similar approach, but for a big publisher. The header menu is organized by the major topics of interest to the news consumer. It maps closely to a traditional newspaper section structure. The footer is different: it's organized around corporate information and links, and secondary audiences such as advertisers and job seekers (Figure 3-43).

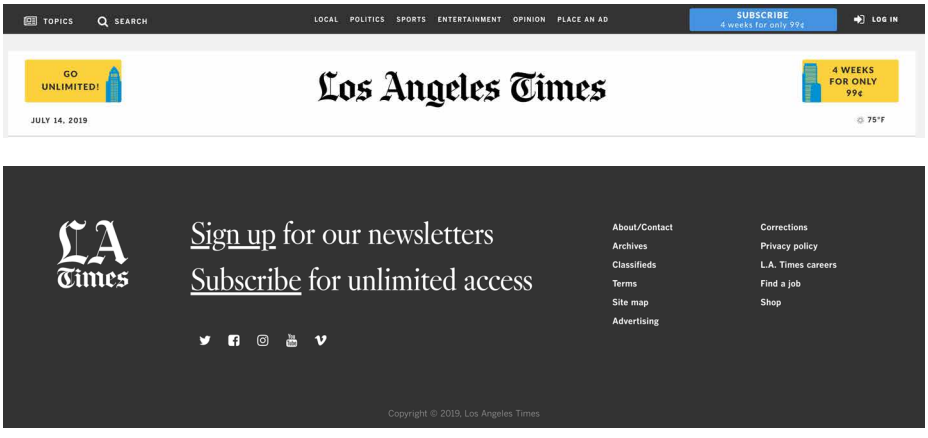


Figure 3-43. *Los Angeles Times header and footer*

The *Wall Street Journal* takes a similar approach in its header and footer (Figure 3-44). There is a robust news topics structure in the header. The footer is dedicated mostly to the business: membership, customer service, and other businesses in the Dow Jones & Company organization. There is additional consumer content featured in the Tools section.

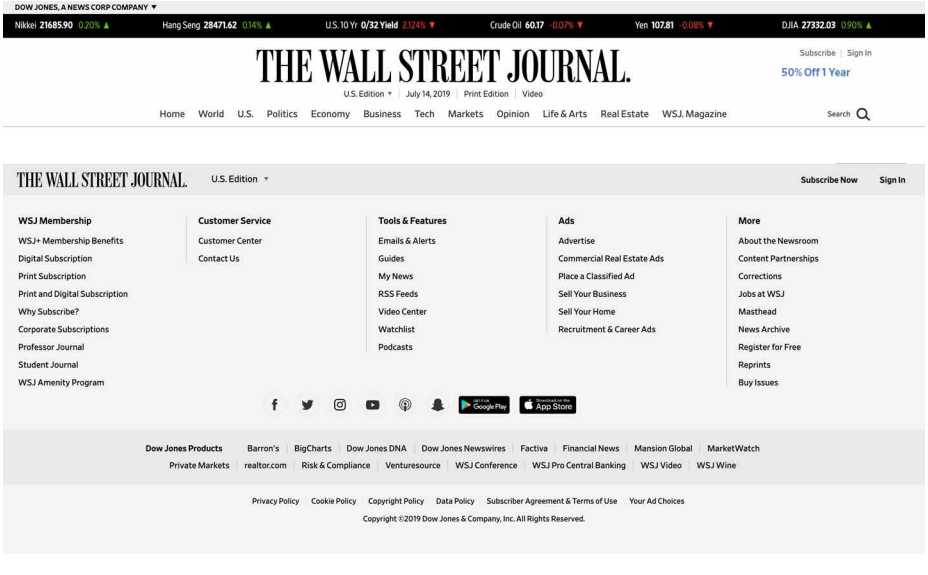


Figure 3-44. *Wall Street Journal footer*

The *New York Times* does not follow this pattern in its footer. It uses this space to offer an expanded view into the information hierarchy of the new content. It's a bigger index that backs up the header navigation (Figure 3-45). There are links to the corporate organization, but they are strongly deemphasized at the very bottom.

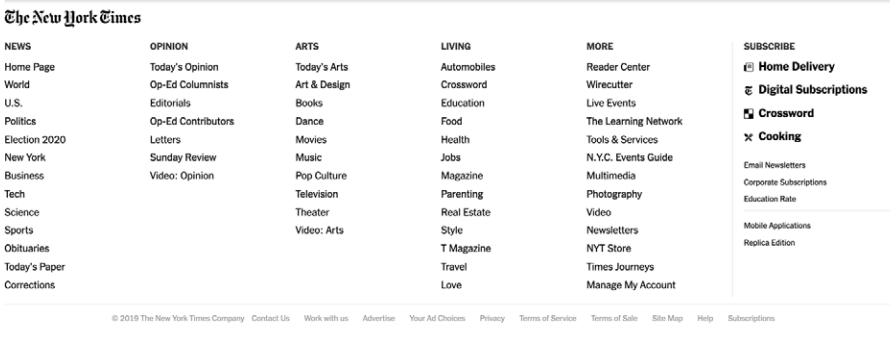


Figure 3-45. *New York Times footer*

Salesforce uses its sitemap footer to recap the three main areas that it expects visitors and customers to be interested in (Figure 3-46). There is a set of links that showcases the company's products, and why customers should care. A second set offers links to the usual corporate information, careers, and investors information. The third is to important related content, such as the company's third-party application marketplace and its annual conference.

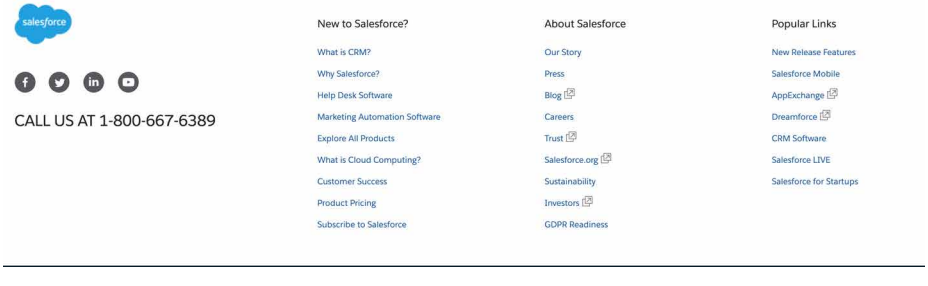


Figure 3-46. *Salesforce footer*

Sign-In Tools

What

Place utility navigation related to a signed-in user's site experience in the upper-right corner. Show tools such as shopping carts, profile and account settings, help, and sign-out buttons.

Use when

Sign-In Tools are useful for any site or service for which users often sign in.

Why

This pattern is purely convention; the upper-right corner is where many people expect such tools to be, so they will often look there. Give users a successful experience by putting these tools where they expect them to be.

How

Reserve space near the upper-right corner of each page for *Sign-In Tools*. Place the user's sign-in name there first (and possibly a small version of their avatar, if it exists), unless the name and avatar are already present elsewhere on the page. Make sure each tool works exactly the same on every page in the site or app.

Cluster together tools such as the following:

- Sign-out button or link (this is important, so make sure it's here)
- Account settings
- Profile settings
- Site help
- Customer service
- Shopping cart
- Personal messages or other notifications
- A link to personal collections of items (e.g., image sets, favorites, or wish lists)
- Home

Don't make this space too large or loud, lest it dominate the page—it shouldn't. This is utility navigation; it's there when a user needs it, but is otherwise "invisible" (well, not literally). For some items, you can use small icons instead of text—shopping carts, messages, and help all have standard visuals you can use, for instance. See the examples in this pattern for some of them.

The site Search box is often placed near the *Sign-In Tools*, although it needs to be in a consistent spot regardless of whether anyone is signed in.

When no user is signed in, this area of the page can be used for a sign-in box—name, password, call to action, and possibly tools for retrieval of forgotten passwords.

Examples

Following are some examples of *Sign-In Tools* from Airbnb (Figure 3-47), Google (Figure 3-48), and Twitter (Figure 3-49). These are visually unobtrusive but findable simply because they're in the correct corner of the page or window. Airbnb surfaces a set of links that relate to membership and signing in: becoming a host, upcoming trips, saved searches in addition to the member sign-in tools drop-down menu. Google and Twitter hide the sign-in tools completely in a drop-down menu. Only the user's profile picture displays as the access by default.

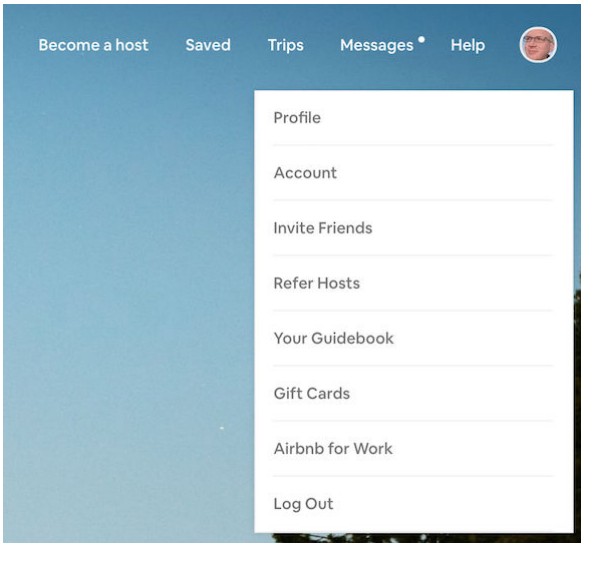


Figure 3-47. *Airbnb sign-in tools*

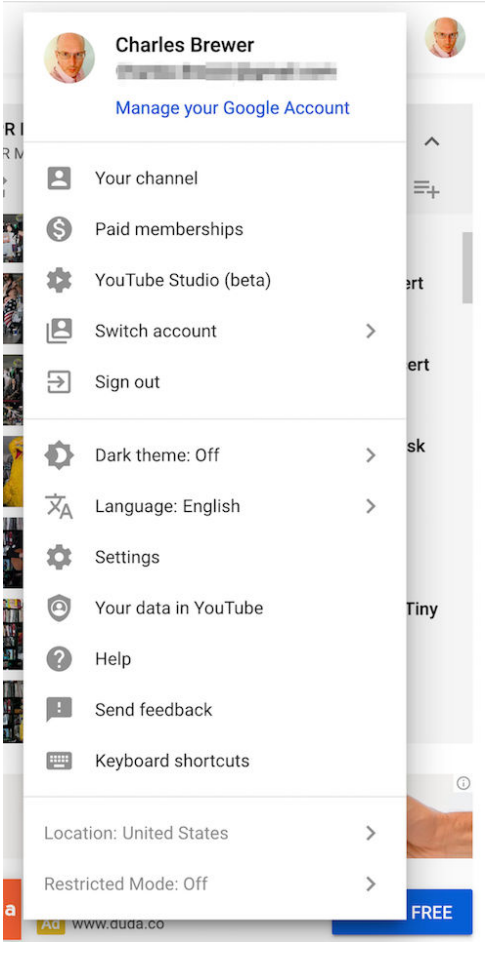


Figure 3-48. *Google sign-in tools*

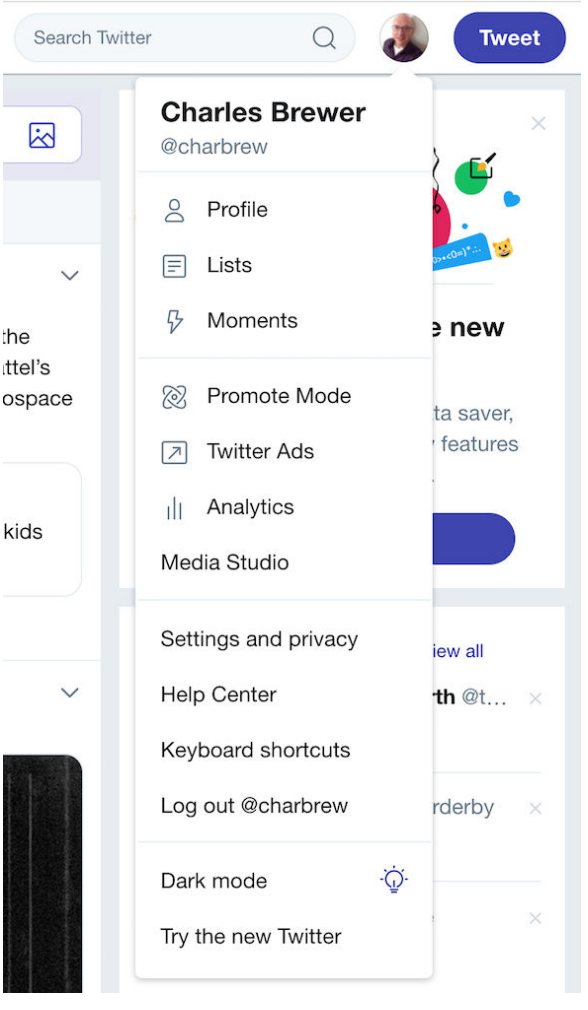


Figure 3-49. *Twitter sign-in tools*

Progress Indicator

What

On each page in a sequence, show a map of all the pages in order to show steps in a process, including a "You are here" indicator. Retailer Menlo Club (Figure 3-50) uses a progress indicator in its check-out process.

Figure 3-50. *Menlo Club checkout progress indicator.*

Use when

You design a written narrative, a process flow, a *Wizard*, or anything else through which a user progresses page by page. The user's path is mainly linear.

If the navigation topology is large and hierarchical (as opposed to linear) you might want to consider using *Breadcrumbs* instead. If you have a large number of steps or items and their order doesn't matter much, this morphs into a *Two-Panel Selector* (Chapter 7).

Why

Progress Indicators indicate to a user how far they've come through a series of steps—and, more important, how far they have yet to go before the process is finished. Knowing this helps them decide whether to continue, estimate how long it will take, and stay oriented.

Progress Indicators also serve as navigational devices. If someone wants to go back to a previously completed step, they can do so by clicking that step in the map of steps.

How

Near an edge of the page, place a small map of the pages in the sequence. Make it one line or column if you can, to keep it from competing visually with the actual page content. Give the current page's indicator some special treatment such as making it lighter or darker than the others; do something similar with the already-visited pages.

For the user's convenience, you might want to put the map near or next to the main navigation controls, usually Back and Next buttons.

How should you label each page's indicator on the map? If the pages or steps are numbered, use the numbers—they're short and easy to understand. But you should

also put the page titles in the map. (Keep the titles short, so the map can accommodate them.) This gives the user enough information to know which pages to go back to and anticipate what information they'll need in upcoming pages.

Examples

The slideshow shown in Figure 3-51 has a simple *Progress Indicator* at the bottom. It's a simple page count, with the current page indicated. The user cannot use it to actually move through the sequence. Users would need to use the Previous and Next arrow buttons on the sides.

Figure 3-51. *National Geographic Kids slideshow with page number progress indicator (center bottom)*

The Vanity Fair slideshow also uses a static page numbering progress indicator (Figure 3-52). The indicator itself does not trigger navigation.

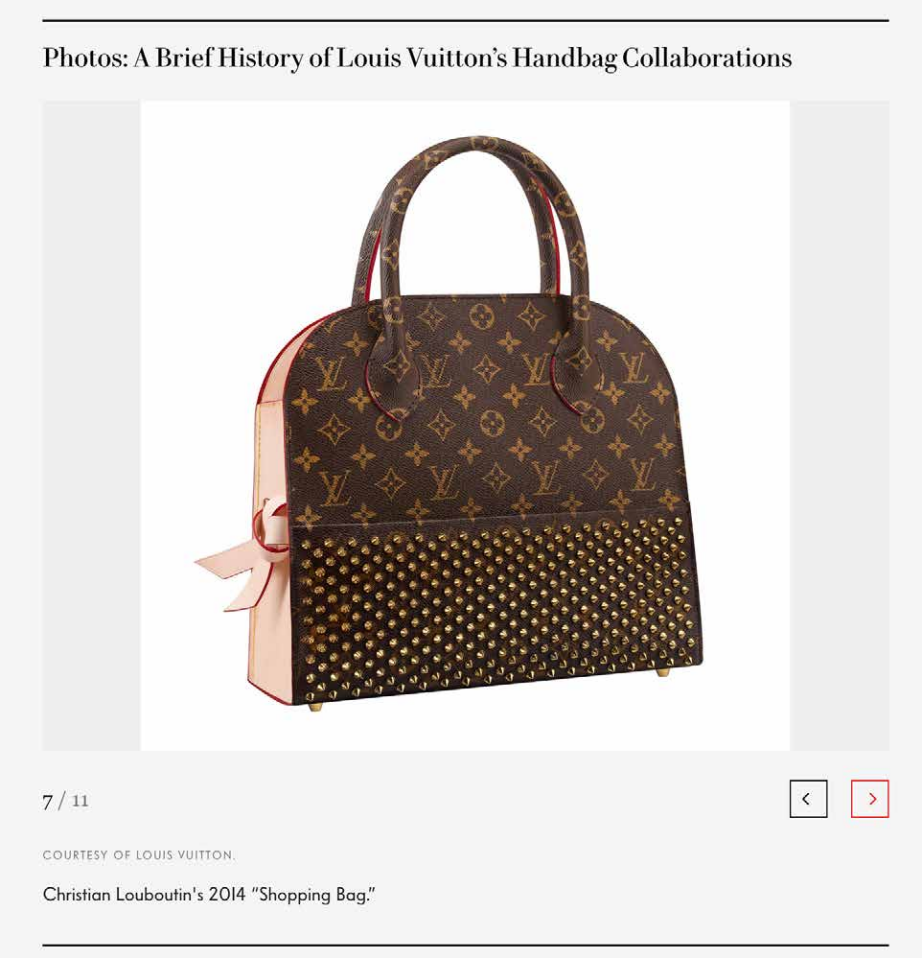


Figure 3-52. *Vanity Fair's slide show with page number progress indicator*

The Mini Cooper product configurator (Figure 3-53) shows a full-featured progress indicator that lets the user move back and forth at will, but organizes the pages in a sequence. The *Progress Indicator* at the top is a critical control for "playing" with the app, for moving among the various pages, and exploring different options.

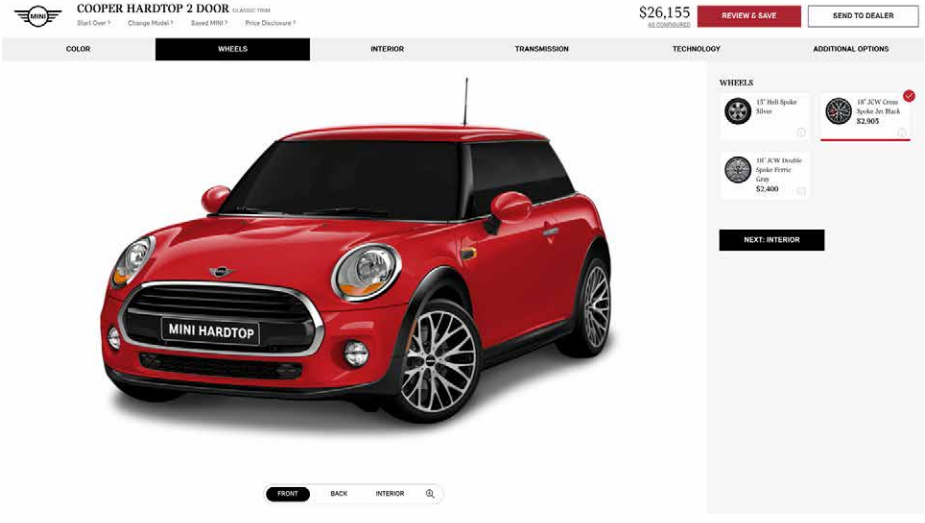


Figure 3-53. *Mini Cooper product configurator, with sequence map across the top*

Ecommerce check-out processes usually have a few, defined steps. The one for B&H Photo (Figure 3-54) has a typical *Progress Indicator* at the top. Its steps are disabled when the user hasn't completed the required earlier step.

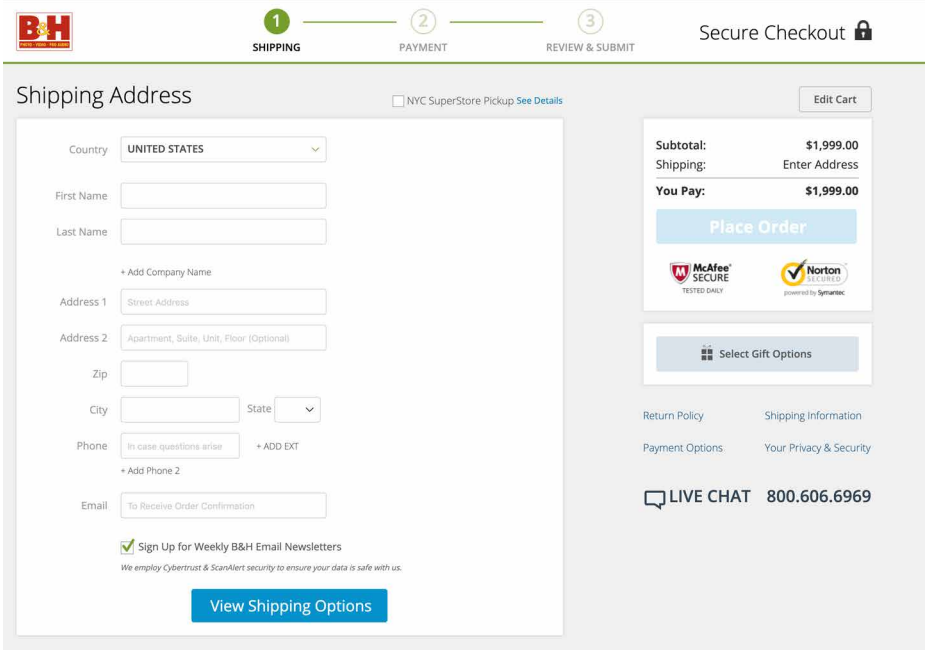


Figure 3-54. *B&H checkout with progress indicator*

Breadcrumbs

What

Breadcrumbs refers to a specific type of navigation that shows the path from the starting screen down through the navigational hierarchy, the content architecture of the site, to the selected screen. The *Breadcrumbs* navigation pattern can be thought of as a series of parent—child links that show the drilldown into the information architecture of the site. The breadcrumbs show where in the content hierarchy the current screen is. Target (Figure 3-55) shows a common use of breadcrumbs on sites with large product directories.

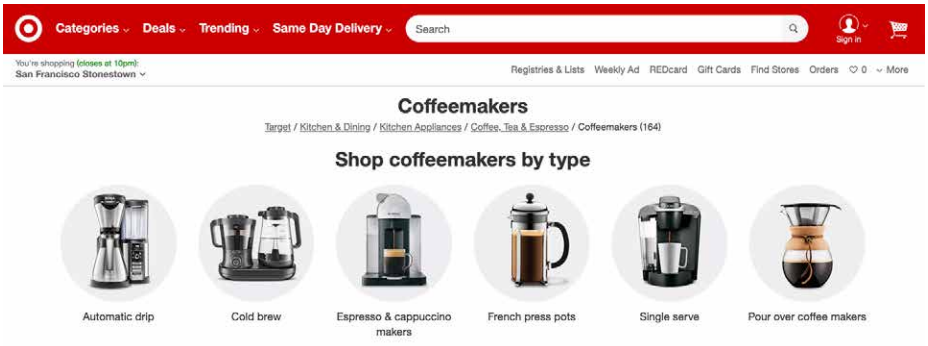


Figure 3-55. *Target breadcrumbs*

Use when

Your application or site has a hierarchical structure with two or more levels. Users move around via direct navigation, browsing, filtering, searching within the site, or deep-linking into it from elsewhere. Global navigation alone isn't sufficient to show a "You are here" signpost, because the hierarchy is too deep or large.

Alternatively, your site or app might have a set of browsing and filtering tools for a large dataset, such as products being sold online. The products are categorized in a hierarchy, but that categorization doesn't necessarily match the way people will look for those products.

Why

Breadcrumbs show each level of hierarchy leading to the current page, from the top of the application all the way down. In a sense, they show a single linear "slice" of the overall map of the site or app.

So, like a *Progress Indicator*, *Breadcrumbs* help a user to pinpoint where they are. This is especially handy if they've jumped abruptly to somewhere deep in the tree, as they would by following search results or a faceted browsing tool. Unlike a *Progress Indicator*, though, *Breadcrumbs* don't indicate to the user where they're headed next. They deal only with the present.

Some texts tell you that *Breadcrumbs*—so named for the Hansel and Gretel story, in which Hansel drops breadcrumbs on a forest trail to mark their way home—are most useful for telling the user how they got to where he is from the top of the site or app. But that's only true if the user has drilled straight down from the top, with no side-tracking, or following other branches, or dead-ends, or searching, or linking directly from other pages…not likely.

Instead, *Breadcrumbs* are best for telling you where you are relative to the rest of the app or site—it's about context, not just history. Look at the Target example in Figure 3-55. Faceted browsing—searching for items with certain characteristics—brought me to this page deep in the Target website. (A keyword search could have done the same.) But now that I'm here, I can see where I am in the product hierarchy and I know what else I can look at. I can use the *Breadcrumbs* to look at all of Target's stand mixers and do some comparison shopping.

Finally, *Breadcrumbs* are usually clickable links or buttons. This turns them into a navigational device in their own right.

How

On each page that is below a certain level in the navigational hierarchy—that is, it is deep in the content or screen or page architecture—show a list of all the parent pages, up to the main or home page. The goal is to see the parent-child relationships or what the "drill down" path is to get to the currently selected screen. Near the top of the page, put a line of text or icons indicating the current level of hierarchy. Start with the top level; to its right, put the next level and so on down to the current page. Between the levels, put a graphic or text character to indicate the parent—child relationship between them. This is usually a right-pointing arrow, triangle, greater-than sign (>), slash (/), or right angle quotes (»).

The labels for each page should be the page titles. Users should recognize them if they've been to those pages already; if not, the titles should at least be self-explanatory enough to tell the user what those pages are about. The labels should be links to those pages.

Some *Breadcrumbs* show the current page as the last item in the chain; some don't. If yours do, make them visually different from the rest of the items because they're not links.

Examples

Samsung makes extensive use of *Breadcrumbs*, especially in its content-dense customer support area of the website. Figure 3-56 shows two uses of breadcrumbs. One is in the expected position in the upper left, just above the central image. It shows where we are in the product support hierarchy. Down below in the main content area, there is a step-by-step "Find your TV" widget that helps the customer narrow in on their specific television. On the left we can see a smaller breadcrumb that plays back where the user is in the product hierarchy.

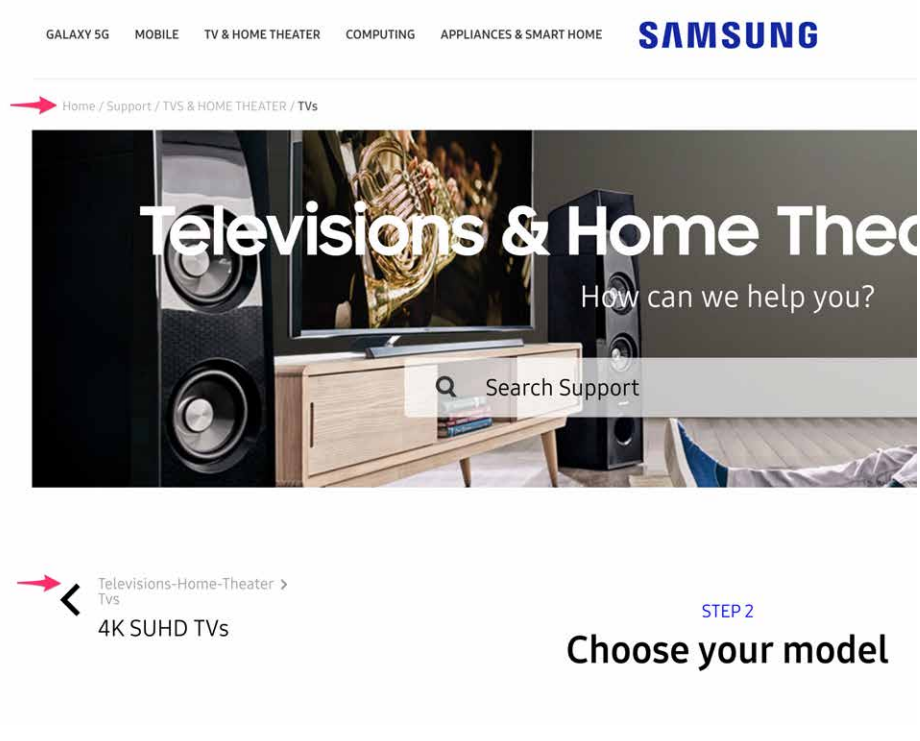


Figure 3-56. *Samsung TV support; showing breadcrumbs twice*

B&H Photo uses a large, prominent breadcrumb component to help the customer see where they are and where they have navigated to in this huge online catalog of products (Figure 3-57).

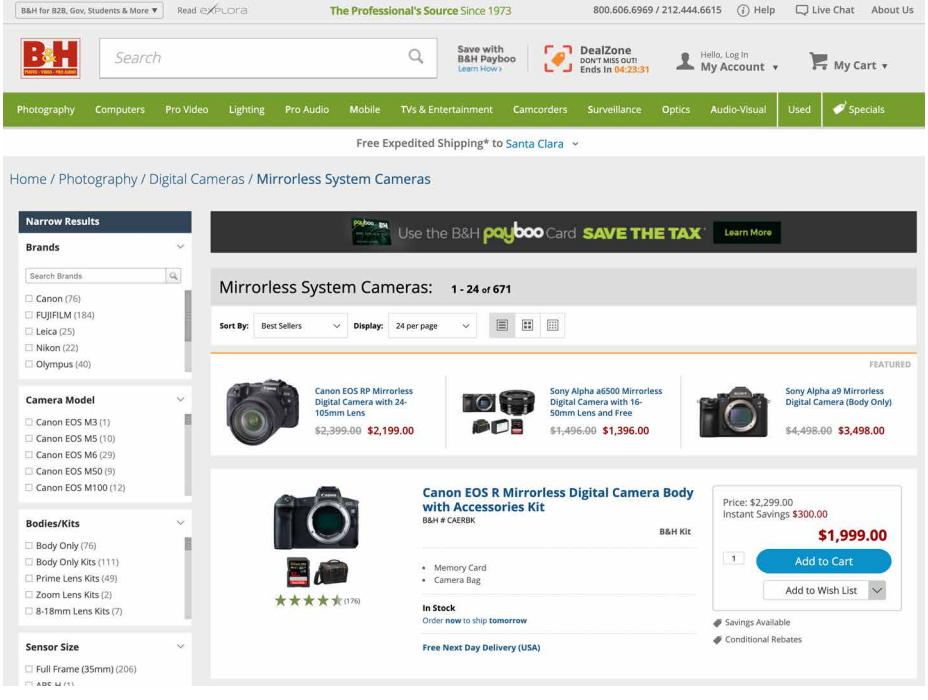


Figure 3-57. *Breadcrumbs on the B&H site*

Figure 3-58 shows an example of *Breadcrumbs* used outside a "page" context. The Chrome developer tools, among many other such tools for software developers, provide a way for users to manage very deep hierarchical structures (in this case, the Document Object Model and nested structural tags in an HTML page). *Breadcrumbs* are invaluable here for letting the you keep track of where you are in that code structure.

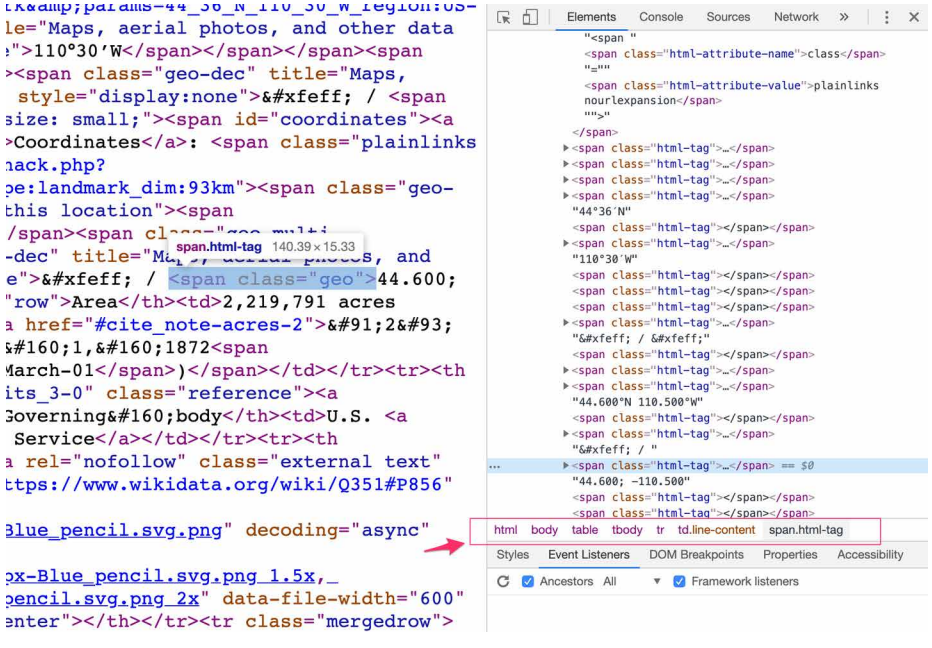


Figure 3-58. *Chrome developer tools*

Annotated Scroll Bar

What

An addition to ordinary scroll bar functionality so that it serves as a notification or as a map of the content in the current document or screen, or as a "You are here" indicator. In the example from Google Docs (Figure 3-59), the pop-up panel attached to the scroll grab bar lets the user see where they are in a multipage document.

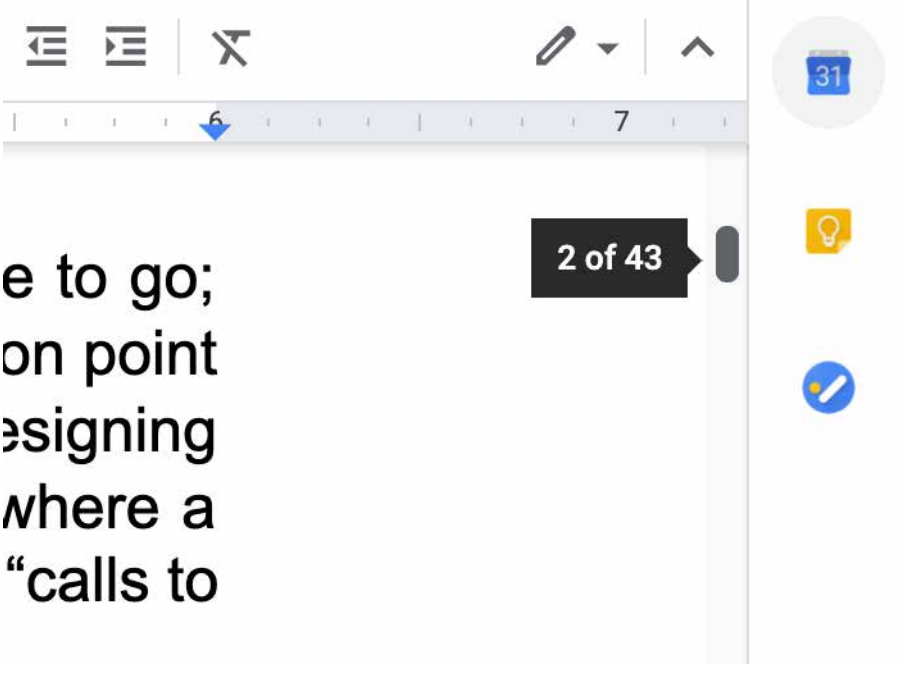


Figure 3-59. *Google Docs scroll bar showing page numbers*

Use when

You're designing a document- or data-centric application. Users will scan this document or graphic for items of note such as specific page numbers or section or chapter titles, or alerts. This can be helpful if your users have trouble keeping track of where they are and where to go next as they scroll.

Why

Even though the user remains within one navigational space as they scroll through the content, signposts are still useful. When scrolling quickly, it's really difficult to

read the text flying by (or impossible, if the screen can't refresh quickly enough), so some other indicator of position is necessary. Even if they stop briefly, the part of the document that they can see might not contain anything by which they can orient themselves, like headers.

Why a scroll bar? Because that's where the user's attention is focused. If you put signposts there, the user will see them and use them as they scroll, rather than trying to look at two different screen areas at once. You can put signposts close to the scroll bar and still get the same effect; the closer, the better.

When the scroll bar shows indicators in the scroll bar track itself, you get something that behaves just like a one-dimensional *Overview Plus Detail* (Chapter 9). The track is the overview; the scrolled window is the detail.

How

Put a position indicator on or near the scroll bar. Either static or dynamic indicators might work—dynamic indicators change as the user scrolls (Figure 3-59). Static indicators are those that don't change from second to second, such as blocks of color in the scroll bar track (see the DiffMerge screenshot in Figure 3-60). Make sure their purpose is clear, though; such things can baffle users who aren't used to seeing graphics in the scroll bar track.

Dynamic indicators change as the user scrolls, and they are often implemented as tool tips. As the scroll position changes, the tool tip shown next to the scroll thumb changes to show information about the content there. This will vary with the nature of the application. Microsoft Word, for instance, puts page numbers and headers in these tool tips.

In either case, you'll need to learn what a user will most likely be looking for and thus what you need to put into the annotations. The content structure is a good starting point. If the content is code, you might show the name of the current function or method; if it's a spreadsheet, show the row number, and so on. Also consider whether the user is currently performing a search—the scroll bar annotation should show where the search results are in the document.

Examples

The DiffMerge application shown in Figure 3-60 visually highlights the differences between two versions of a text file: differing sections are marked in red, and the corresponding section of the scroll bar is highlighted in blue. The scroll bar serves as an overall map, thus making large numbers of file "diffs" easier to comprehend.

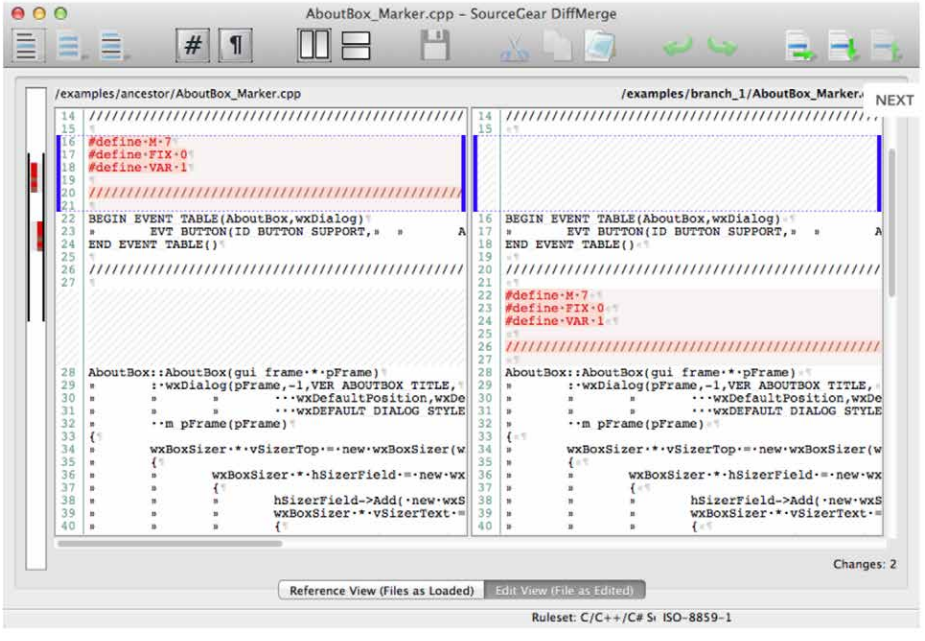


Figure 3-60. *DiffMerge*

Chrome annotates its scroll bar with search results (Figure 3-61). When you search for a word on a web page, Chrome highlights the found words on the page with yellow, and places a yellow indicator in the scroll bar wherever they are found. This way, the user can scroll directly to those points in the document.

In this example, the two instances of the "Find" search word are highighted in the text. In the scroll bar on the right, notice the small hash marks running top to bottom. The search matches are indicated as the first two yellow bars in this list. We can see that there are many more "Find" matches down in the rest of the article because we see many more yellow hash mark displayed down the scroll bar.

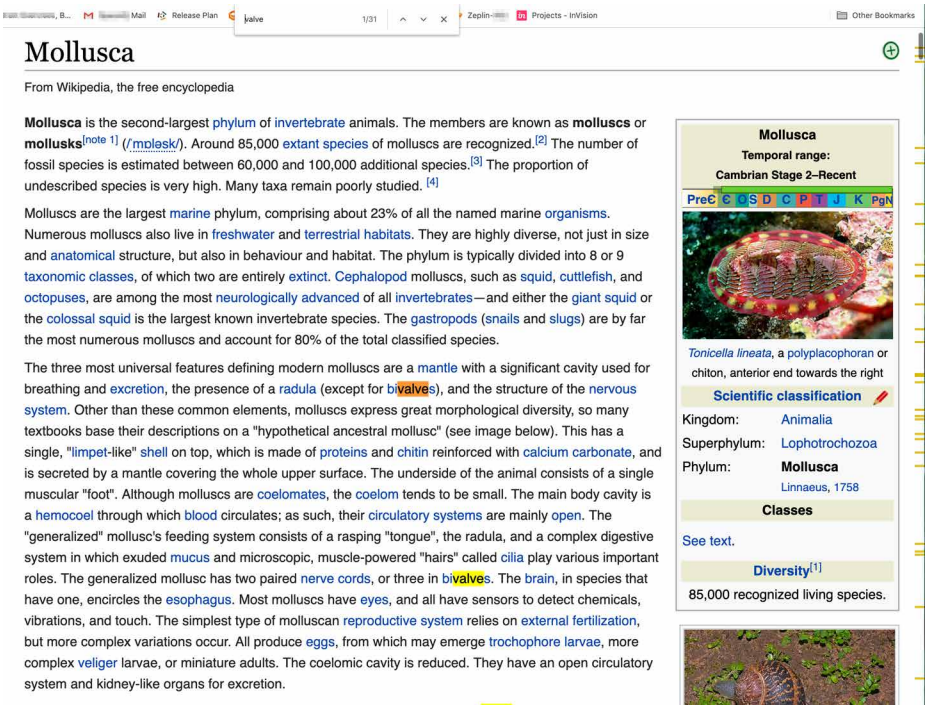


Figure 3-61. *Chrome "Find" results*

Animated Transition

What

Add motion and transformations to the appearance of objects to indicate that an action is happening. Smooth out a startling or dislocating transition with an animation that makes it feel natural. This pattern includes slides, fade ins/fade outs, bounces, zooms, and other animation techniques.

Use when

Interface animations are very popular and common in mobile applications. They are almost a standard for quality interactions on mobile. Some folder, window, and scrolling animations are part of the mobile OS itself. Above and beyond this, when you want to add a clear visual confirmation that a user's input was received, such as a tap on a button or that an action is in progress (such as "screen loading"), or when you simply want to brand your application experience, consider using animations.

Users might need to move through a large virtual space, such as an image or map. They might be able to zoom in to varying degrees, pan or scroll, or rotate the entire thing. This is especially useful for information graphics, such as maps and plots.

Alternatively, the interface might have sections that can be closed and opened again, either by the system or by the user—such as trees with closable parent nodes, standalone windows that open and close, or an interface built with open/close panels. You also might use *Animated Transition* when users jump from one separate page to another.

Animated transitions can also give the user a sense for where the file or object is located in the interface itself—for example, where a launcher icon might be in the macOS launch bar.

Why

All of these transformations can disrupt a user's sense of where they are in the virtual space. Zooming in and out, for instance, can throw off a user's spatial sense when it's done instantaneously, as can rotation and the closing of entire sections that prompts a relayout of the screen. Even scrolling down a long page of text, when it's jumpy, can slow down the reader.

But when the shift from one state to another is visually continuous, it's not so bad. In other words, you can animate the transition between states so that it looks smooth, not discontinuous. This helps keep the user oriented. We can guess that it works because it more closely resembles physical reality—when was the last time you instantly jumped from the ground to 20 feet in the air? Less fancifully, an animated transition gives the user's eyes a chance to track a location while the view changes rather than trying to find the location again after an abrupt change.

It can also give useful UI control and navigation feedback. In Figure 3-62, we see two animations that Apple's macOS uses extensively. The first is rollover magnification of icons in the "dock." This actually helps users understand which icon their mouse is above as they swipe back and forth. The second is the page open/close zoom effect. The document window animates to its parent app icon in the dock, helping the user remember where they have put it away.

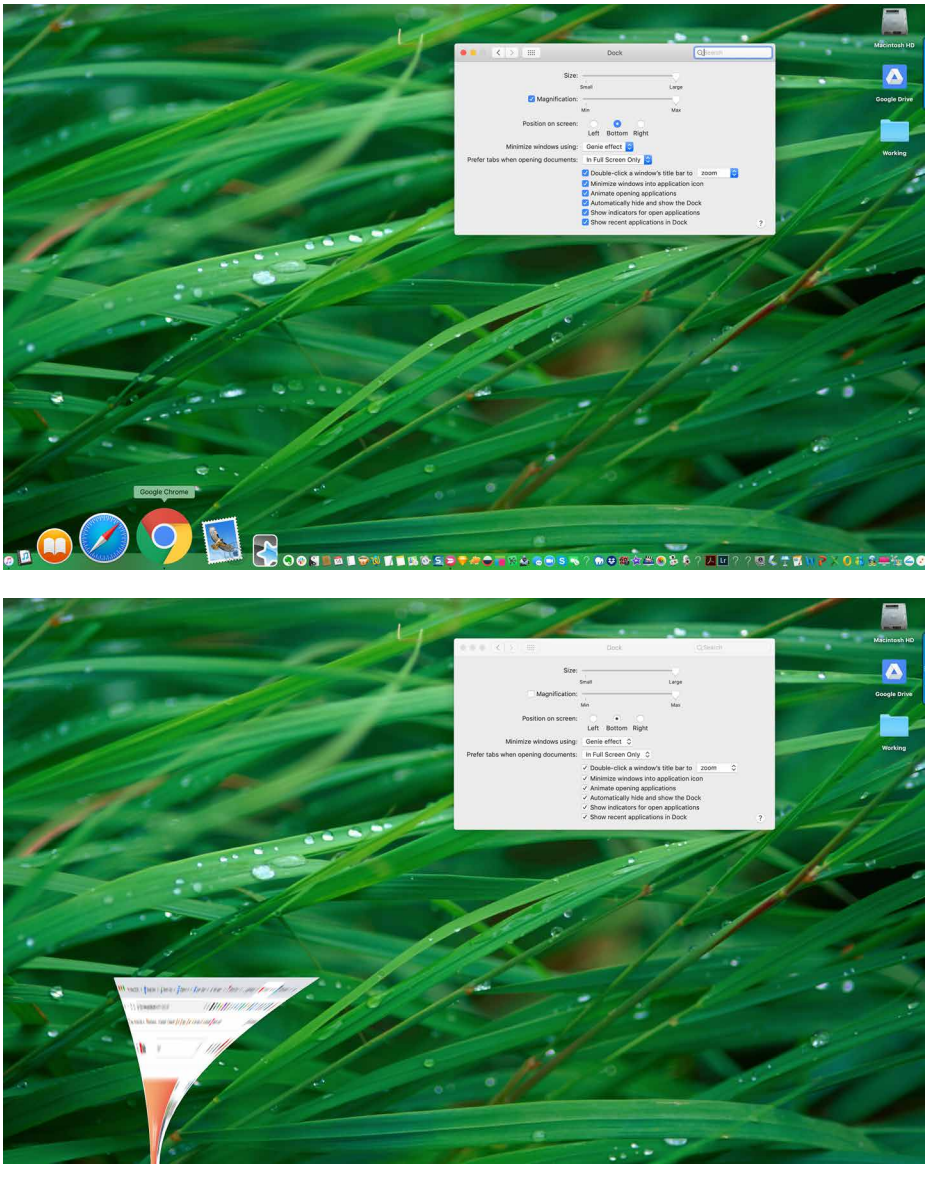


Figure 3-62. *macOS dock magnification and app window transition*

When done well, *Animated Transition* bolsters your application's perception of quality, speed, and liveliness.

How

For each type of transformation that you use in your interface, design a short animation that "connects" the first state with the second state. For zoom and rotate, you might show the in-between zoom or rotate levels; for a closing panel, you might show it shrinking while the other panels expand to take up the space it leaves behind. To whatever extent possible, make it look like something physical is happening.

But this pattern is a double-edged sword. Beware of making the user motion sick! The animations should be quick and precise, with little or no lag time between the user's initiating gesture and the beginning of the animation. Limit it to the affected part of the screen; don't animate the entire window. And keep it short. Research shows that 300 milliseconds might be ideal for smooth scrolling. Test it with your users to see what's tolerable.

If the user issues multiple actions in quick succession, such as pressing the down arrow key many times to scroll, combine them into one animated action. Otherwise, the user might sit there through several seconds' worth of animation as the punishment for pressing the down arrow key 10 times. Again: keep it quick and responsive.

Some of the types of transitions to consider include the following:

- Brighten and dim
- Expand and collapse
- Fade in, fade out, and cross-fade
- Slide
- Spotlight

Examples

Tesla (Figure 3-63) uses a simple zoom in the initial load of its website. The user can see the three Tesla models in overview, and then the image zooms in on the Model 3. In this case, the zoom in is purely an attention-focusing flourish. The user can't actually control the zoom in. However, the user can pan left and right to select the other Tesla models. This panning action is smoothly animated.

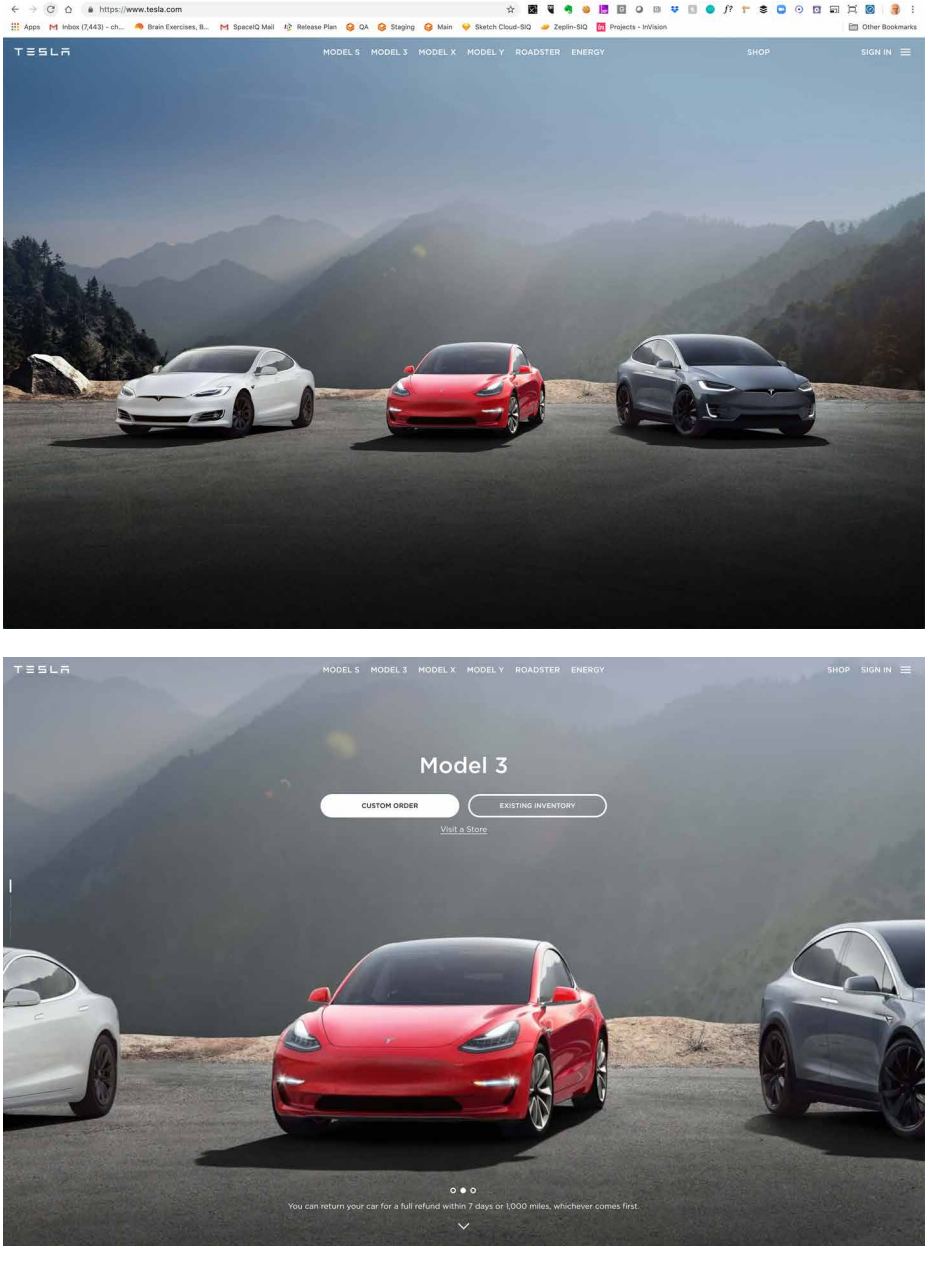


Figure 3-63. *Tesla.com, loading screen "zoom in" animation*

Prezi presentation software makes extensive use of zooming and panning views to create unique and flowing presentations. Figure 3-64 shows a selection of screens from a demonstration. As the user steps through the screens, the presentation zooms in, displays text, pans to the right, and then zooms out again when exiting the section. It makes for an interesting sense of flying through the information space.

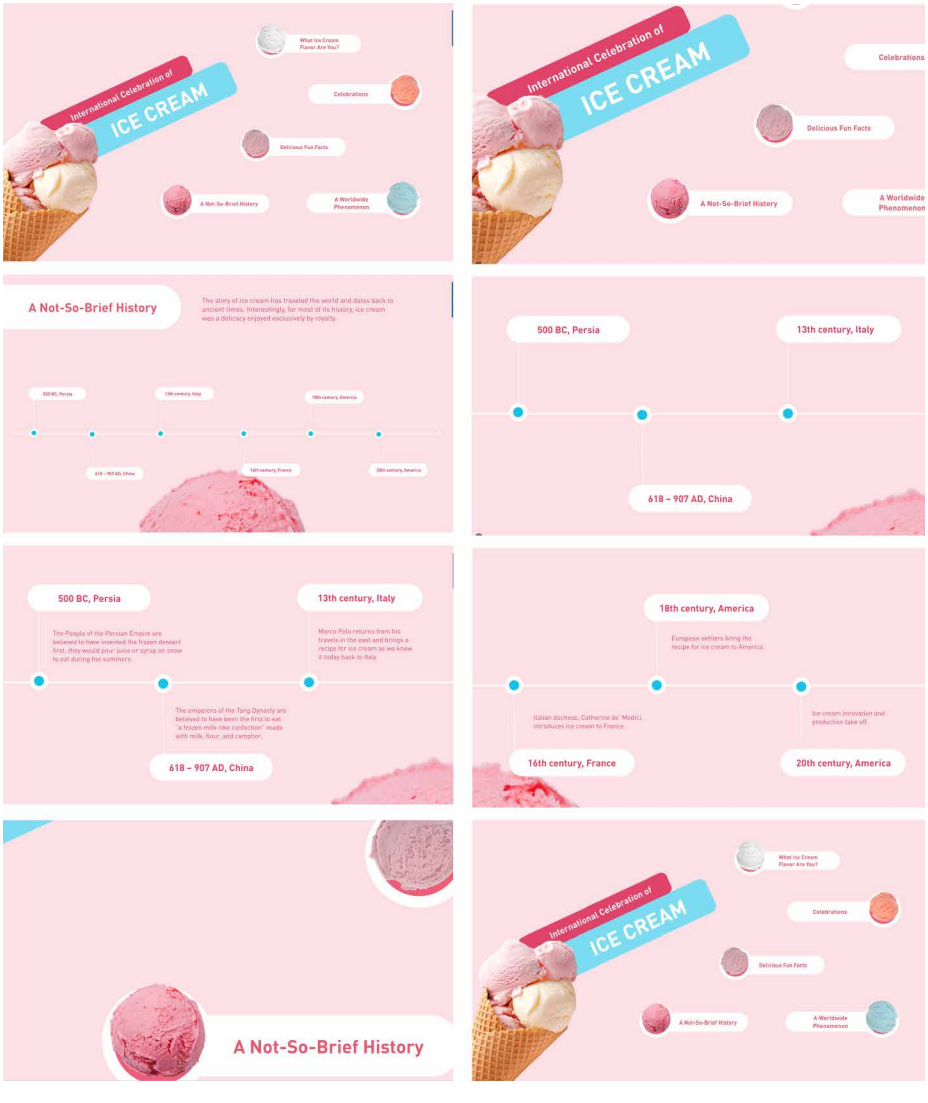


Figure 3-64. *Prezi's unique presentation software features animated "zoom reveals," slides, and zoom-out animations.*

Further Reading

For a much deeper exploration of the value of animation and how to bring it into your interface design, check out the following:

- *Creating Web Animations: Bringing Your UIs to Life* by Kirupa Chinnathambi (O'Reilly, 2017).
- *Transitions and Animations in CSS: Adding Motion with CSS* by Estelle Weyl (O'Reilly, 2016)
- "SVG Animations" in *From Common UX Implementations to Complex Responsive Animation* by Sarah, Drasner (O'Reilly, 2017)

Conclusion

Well-designed navigation and wayfinding in your app or platform can help your users learn your app more quickly, maintain a sense of orientation and context, be confident in using your content and tools, know what to do and where to go, and generally not spend excess time being flustered, confused, or lost. Navigation is a feature of your designs that has a long life cycle —maybe the longest of any feature. If designed well (that is, it makes sense from your user's point of view and fit for purpose, based on the information and tasks they are working with), it will have "evergreen" value. Good navigation and wayfinding help tremendously with new user onboarding, recovering from error states and confusion, getting work done, and generally feeling not blocked but capable. The design approach and the navigation patterns outlined in this chapter will help you to create a UX in your application where getting around takes on a flowing, magic quality of almost vanishing from the user's awareness.

Chapter 4

Layout of Screen Elements

Layout is defined as the particular way elements are arranged. In the case of interface design, these elements are the informational, functional, framing, and decorative parts of the screen. Thoughtful placement of these elements helps to guide and inform your users about the relative importance of the information and functions.

No matter what size screen you are designing for—web, kiosks, or mobile—careful consideration of the placement of content is key to helping the user understand what they need to know and what to do about it.

Often you will hear the term "clean" to describe a screen-based design. A clean layout follows the principles of visual information hierarchy, visual flow, alignment through a grid, and adheres to Gestalt principles.

In this chapter, we define these principles that inform your users what you want them to know and what you want them to do about it.

The Basics of Layout

This section discusses several elements of layout: visual hierarchy, visual flow, and dynamic displays.

Visual Hierarchy

Visual hierarchy plays a part in all forms of design. The most important content should stand out the most, and the least important should stand out the least. A viewer should be able to deduce the informational structure from its layout.

A good visual hierarchy gives instant clues about the following:

- The relative importance of screen elements
- The relationships among them
- What to do next

Visual hierarchy in action

Take a look at the example in Figure 4-1. Can you tell at a glance what is the most important information in Example A? Can you tell what is the most important information in example B? Most people will find Example B the easier layout to understand, even with only rectangles and squares. This is because the arrangement of the screen elements, the relative size, and proportion of the elements imply their importance and guide the viewer on what to pay the most attention to.

Figure 4-1. *A) Example of no visual hierarchy, and B) example of a visual hierarchy*

What Makes Things Look Important?

Size

The size of the headlines and subheads give the viewer a clue about order and importance. Headlines will tend to be bigger and more dramatic because of contrasting size, visual weight, or color. In contrast, a small strip of text at the bottom of the page says quietly, "I'm just a footer," and less important (Figure 4-2).

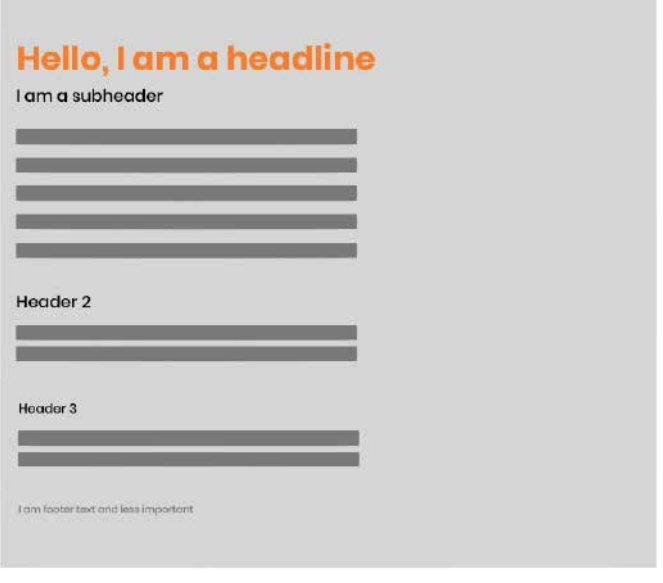


Figure 4-2. *Example of size*

Position

Simply by glancing at the size, position, and colors of the layout in Figure 4-3, you can guess the most important elements of each of the four examples.

Figure 4-3. *Ways to distinguish importance by position, size, or color*

Density

Density refers to the amount of space between elements of the screen. Take a look at the example in Figure 4-4. The left shows a denser layout where content is tightly gathered together. The example on the right has a more open look with content spread evenly apart. The less-dense example will also be slightly more difficult to read and for the viewer to distinguish which elements are related to one another.

Figure 4-4. *Example of more density and less density*

Background color

Adding shade or a background color will draw attention to a block of text and distinguish it from other text. The example on the left in Figure 4-5 has a consistent background for all elements. This implies no distinction and a continuity of the importance of the elements. In comparison, in the example on the right side of the figure, the background and contrast elements in the middle draw the eye immediately to it, implying more importance. Contrast draws attention.

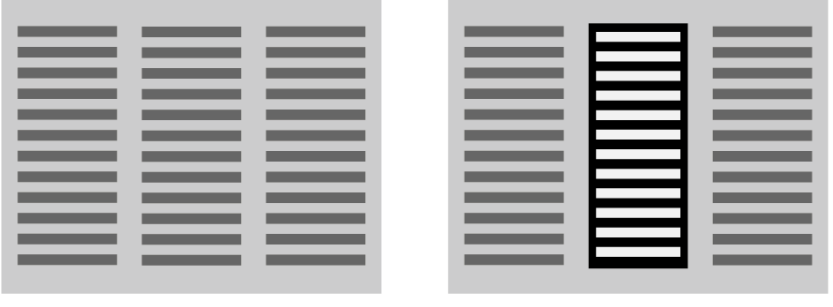


Figure 4-5. *Example of no background color and background color*

Rhythm

Lists, grids, whitespace, and alternating elements such as headlines and summaries can create a strong visual rhythm that irresistibly draws the eye, as shown in Figure 4-6.

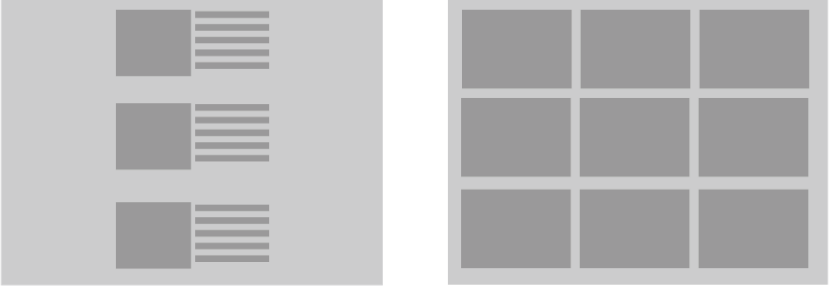


Figure 4-6. *Lists and grids to create a visual rhythm*

Emphasizing small items

To make small items stand out, put them at the top, along the left side, or in the upper-right corner. Give them high contrast and visual weight, and set them off with whitespace. But note that in a text-heavy screen, like most websites, certain controls —especially search fields, sign-in fields, and large buttons—tend to stand out anyway. This is less about raw visual characteristics than meaning: if someone is looking for a Search box, for instance, their eyes will go straight to the text fields on the page. (They might not even read the labels for those text fields.)

Another way to emphasize small items is to use spacing and contrast to distinguish them, as shown in Figure 4-7.

Figure 4-7. *Techniques to bring attention to small items*

Alignment and grid

In digital design, legibility is critical. Helping guide the viewer to information and action is key. Creating a design that is based on a grid (Figure 4-8) allows the designer to focus on the content, assured that the layout will have visual consistency and balance (Figure 4-9). Grids also help multiple designers work on separate but related layouts.

Figure 4-8. *Gridless layout (left), and a layout designed on a grid (right)*

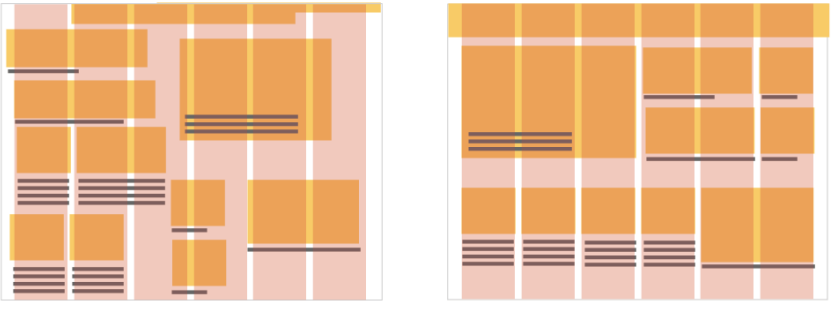


Figure 4-9. *Grid overlaid on the examples in Figure 4-8*

Grids are found in all digital design, and they are fundamental in creating designs that will be responsive and can accommodate dynamic content, which we discuss later.

Grids are composed of margins and gutters (Figure 4-10 and Figure 4-11). Margins are the space around the content and gutters are the spaces between.

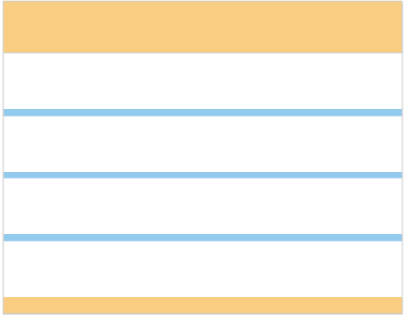


Figure 4-10. *Vertical grid with margins (yellow) and gutters (blue)*

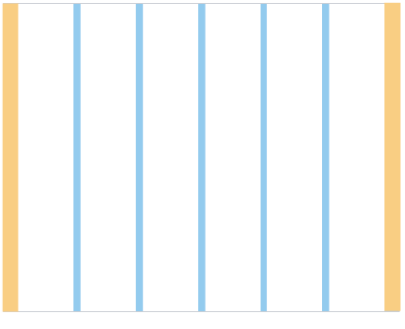


Figure 4-11. *Horizontal grid with margins (yellow) and gutters (blue)*

Using a grid is a good way to ensure that the content will create a visually harmonious composition, and will aid in reducing the cognitive load of your viewer.

Four Important Gestalt Principles

"Gestalt" is a term that comes from a psychological theory that took hold in the 1920s. *Gestalt* is a German word that means "form" or "shape." In design, we often refer to *Gestalt Principles* which refers to a set of rules describing the way humans perceive visual objects. The theory behind grouping and alignment was developed early in the twentieth century by the Gestalt psychologists. They described several layout properties that seem to be hardwired into our visual systems. We look at each one in the following subsections.

Proximity

When you put things close together, viewers associate them with one another. This is the basis for a strong grouping of content and controls on a user interface (UI). Grouped items look related. Conversely, isolation implies a distinction.

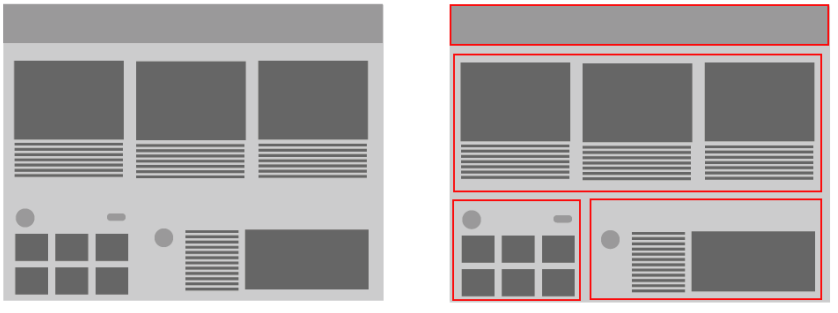


Figure 4-12. *The Gestalt principle of proximity*

Similarity

Items that are similar in shape, size, or color are perceived as related to one another. If you have a few things "of a type" and you want viewers to see them as equally interesting alternatives, give them an identical (and distinctive) graphic treatment.

A list of many similar items, arranged in a strong line or column, becomes a set of peer items to be viewed in a certain order. Align these items very precisely with one another to create a visual line. Examples include bulleted lists, navigation menus, text fields in a form, row-striped tables, and lists of headline and summary pairs.

Are one or more items more "special" than the others like it? Give it a slightly different treatment, such as a contrasting background color, but otherwise, keep it consistent with its peers (see the example on the right in Figure 4-13). Or use a graphic element to disrupt the line along which the items are aligned, such as a bump-out, overlap, or something at an angle.

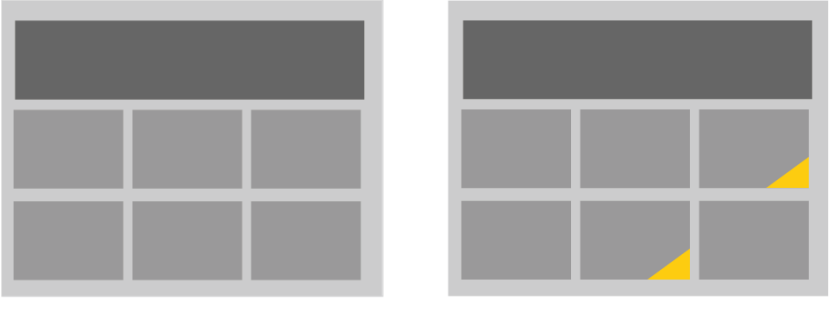


Figure 4-13. *Grouping related peer items (left) and distinguishing two items among peers (right)*

Continuity

Our eyes will naturally follow the perceived lines and curves formed by the alignment of other elements, as demonstrated in Figure 4-14.

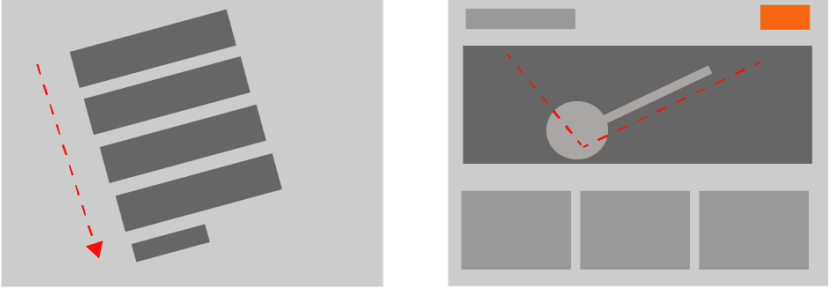


Figure 4-14. *Two examples of visual continuity.*

Closure

The brain will naturally "close" lines to create simple closed shapes such as rectangles and blobs of whitespace, even if they aren't explicitly drawn for us. In the examples in Figure 4-15, you will likely see (clockwise, from upper left) a rectangle, a circle, and two triangles. None of these shapes are actually represented in the image but the eye completes the line in your brain.

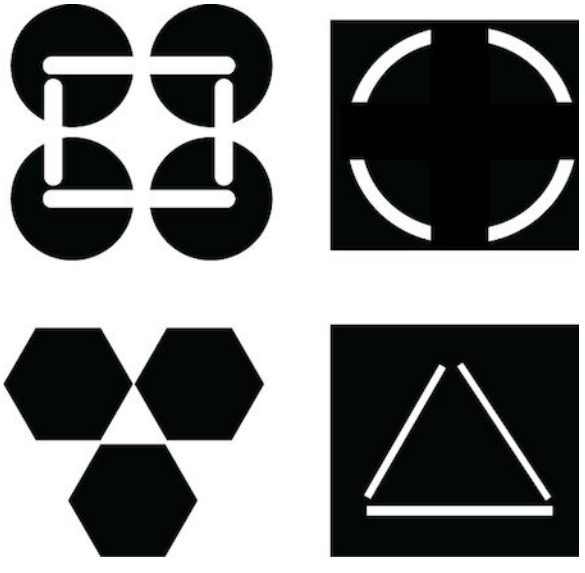


Figure 4-15. *Example of closure*

These principles are best used in combination with one another. Continuity and closure explain alignment. When you align things, you form a continuous line with their edges, and the users will follow that line and assume a relationship. If the aligned items are coherent enough to form a shape—or to form one out of the whitespace or "negative space" around it—closure is also at work, adding to the effect.

Visual Flow

Visual flow deals with the tracks that readers' eyes tend to follow as they scan the page.

It's intimately related to visual hierarchy, of course—a well-designed visual hierarchy sets up focal points on the page wherever you need to draw attention to the most important elements, and visual flow leads the eyes from those into the less important information.

As a designer, you want to be able to control visual flow on a page so that people follow it in approximately the correct sequence. Several forces can work against one another when you try to set up a visual flow. For those raised in Western culture, one is our tendency to read top to bottom and left to right. When faced with a monotonous page of text, that's what you'll do naturally; but if there are strong focal points on the page, they can distract you from the usual progression, for better or for worse.

Focal points are the spots your eyes can't resist going to. You tend to follow them from strongest to weakest, and skillfully designed pages have only a few—too many focal points dilute the importance of each one. A good visual hierarchy uses focal points to pull eyes to the appropriate places in the correct order. The next time you pick up a magazine, look at some well-designed ads and notice what your eyes gravitate toward. The best commercial graphic artists are masters at setting up focal points to manipulate what you see first.

So, how do you create a good visual flow? One simple way is to use implied lines, either curved or straight, to connect elements on the page. This creates a visual narrative for the viewer to follow. In the example in Figure 4-16, the designer who created this older Uber home page used several techniques to guide the eye around the screen; from the use of the blue call to action buttons, to the use of a grid to bring harmony to the composition, to the size of the headline in comparison to the subhead. Pay particular attention to the gaze of the young woman in the photo. Her gaze is pointed to the headline, which uses implied lines to subconsciously guide the viewer's eye around the layout to hit all the of the most important elements of the page.

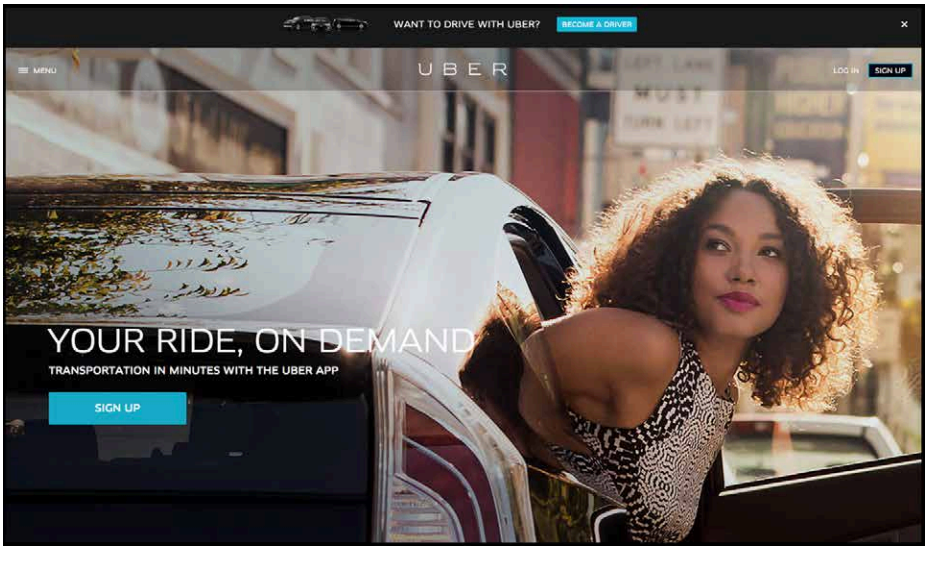


Figure 4-16. *Visual hierarchy in an older Uber home page*

In the examples in Figure 4-17, your eyes will naturally follow the lines. It's not difficult to set up a layout that flows well, but be on your guard against layout choices that work counter to flow. If you want viewers to read a site's story and value proposition, arrange the critical pieces of that narrative along a continuous line, and don't interrupt it with eye-catching extras.

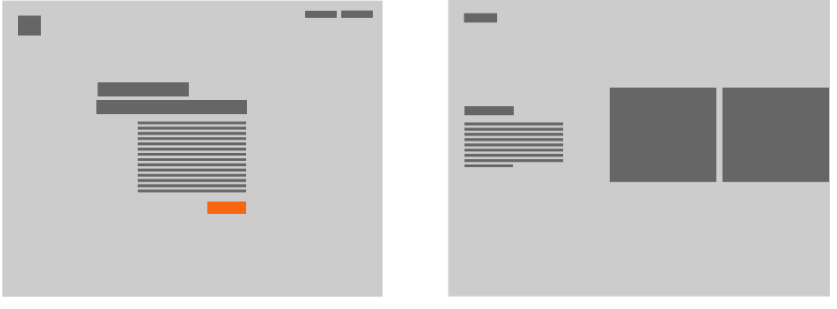


Figure 4-17. *Implied lines for visual flow*

If you're designing a form or set of interactive tools, don't scatter controls all over the page—that just forces the user to work harder to find them. In the example in Figure 4-18, look for where the call to action button is; it should be easy to find because the designer placed it after the text that a user will read first. If you don't care whether they read that text, you can isolate the calls to action with whitespace.

Figure 4-18. *Calls to action*

Figure 4-19 shows a poor example of visual flow and visual hierarchy. How many focal points are there, and how do they compete with one another? Where do your eyes want to go first, and why? What does this page say is important?

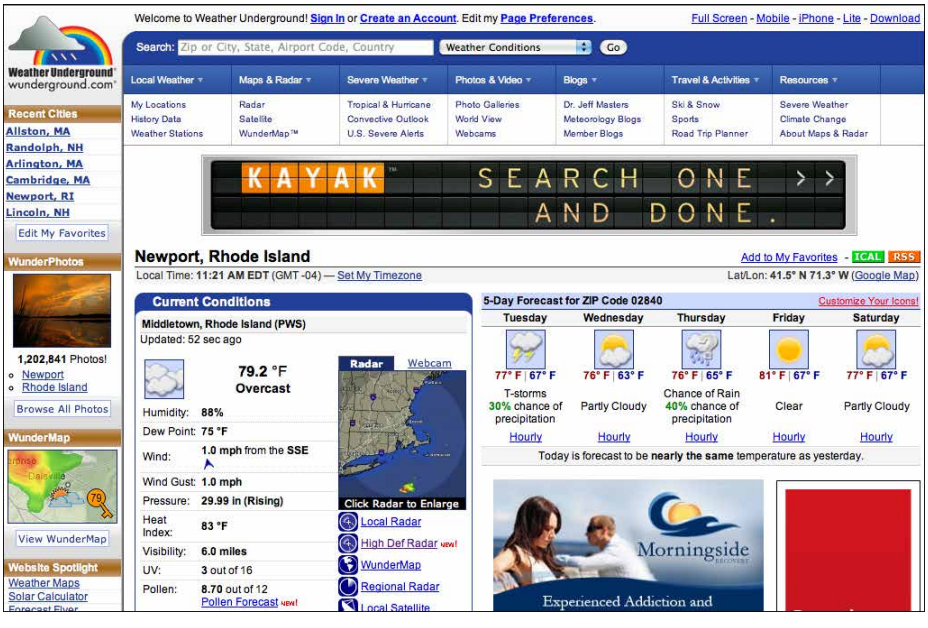


Figure 4-19. *Weather Underground's jumbled visual hierarchy*

Using Dynamic Displays

Everything we've discussed so far applies equally to UIs, websites, posters, billboards, and magazine pages. They deal with static aspects of the layout. Ah, but you have a dynamic computer to work with—and suddenly time becomes another dimension of design.

Just as important, computers permit user interaction with the layout to an extent that most printed matter can't. There are ways in which you can take advantage of the dynamic nature of computer or mobile displays. Consider space usage, for example—even the biggest consumer-grade computer screens have less usable space than, say, a poster or a newspaper page. There are many techniques for using that space to present more content than you can show at one time.

Scroll Bars

Scroll bars let the user move around at will, in one or two dimensions (but refrain from using horizontal scrolling with text, please!). Scroll bars are one very common way of presenting a small "viewport" onto a large thing, such as text, an image, or a

table. Or, if you can carve up the content into coherent sections, you have several options—*Module Tabs*, *Accordions*, *Collapsible Panels*, and *Movable Panels* are patterns that put some layout control into the user's hands, unlike the more static *Titled Sections*.

The following techniques invoke time by letting the user see different content at different times.

Responsive Enabling

To guide a user successfully through a form or process, or to prevent confusion about the user's mental model, a UI might only enable certain functionality when the user completes a specific action. In the example in Figure 4-20, the Mac OS System Preferences provide a typical example of disabling based on a binary choice.

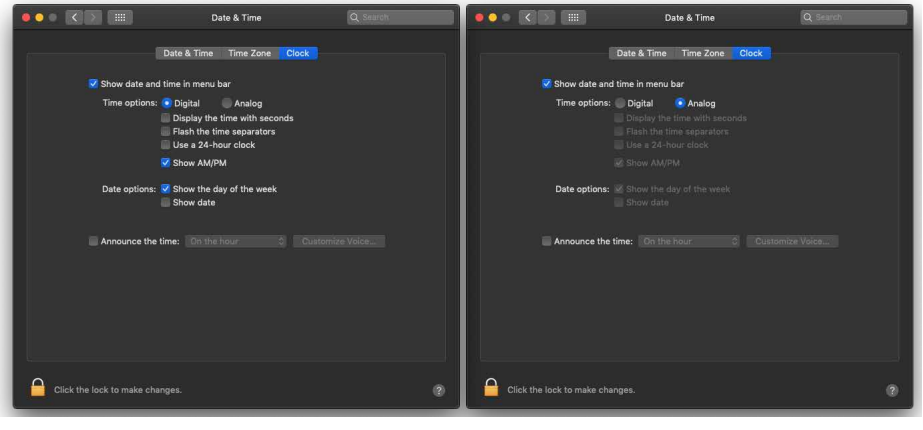


Figure 4-20. *Mac OS System Preferences*

Progressive Disclosure

In some contexts, information is shown only after the user takes a specific action. Moo.com, an online custom business card and print shop (Figure 4-21), uses this technique in its "create a custom product" flow. In the figure, a customer doesn't see the customization options until they click into the editable parts of the card.

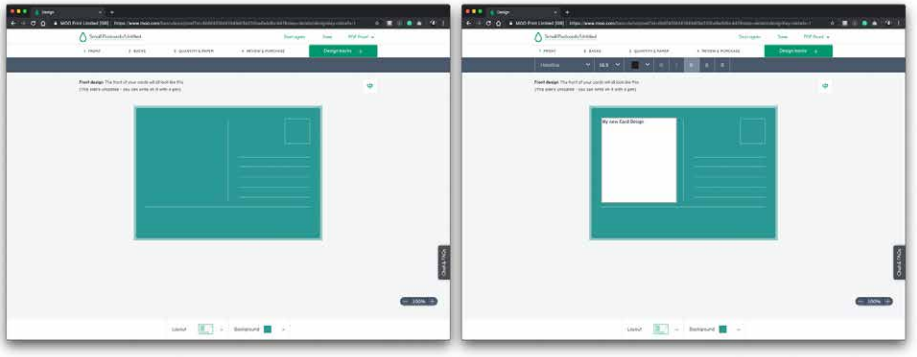


Figure 4-21. *Progressive Disclosure example in Moo.com*

UI Regions

Whether you are designing for web, software, or mobile layouts, you can typically count on having one or more of the following UI regions to work with (see also Figure 4-22):

Header/window title
: This is the topmost region in any given layout, it is used for global branding and global navigation elements in mobile and web design, and you will often see this region used for toolbars and navigation in software. Headers are typically a constant element in the layout of a template, so it's important to carefully choose what you put on this valuable screen space.

Menu or navigation
: This is usually near the top or to the left-hand side of the screen. Note these can also be panels (see below).

Main content area
: This should be where the majority of screen real estate is dedicated. This is where the information content, forms, task areas, and branding for landing experiences is located.

Footers
: This is where secondary and global or redundant navigation resides; it can also contain helpful information like business contact information.

Panels
: Panels can be on the top, on the side or the bottom. They can be persistent or dismissible depending on what functionality you are using on that panel.

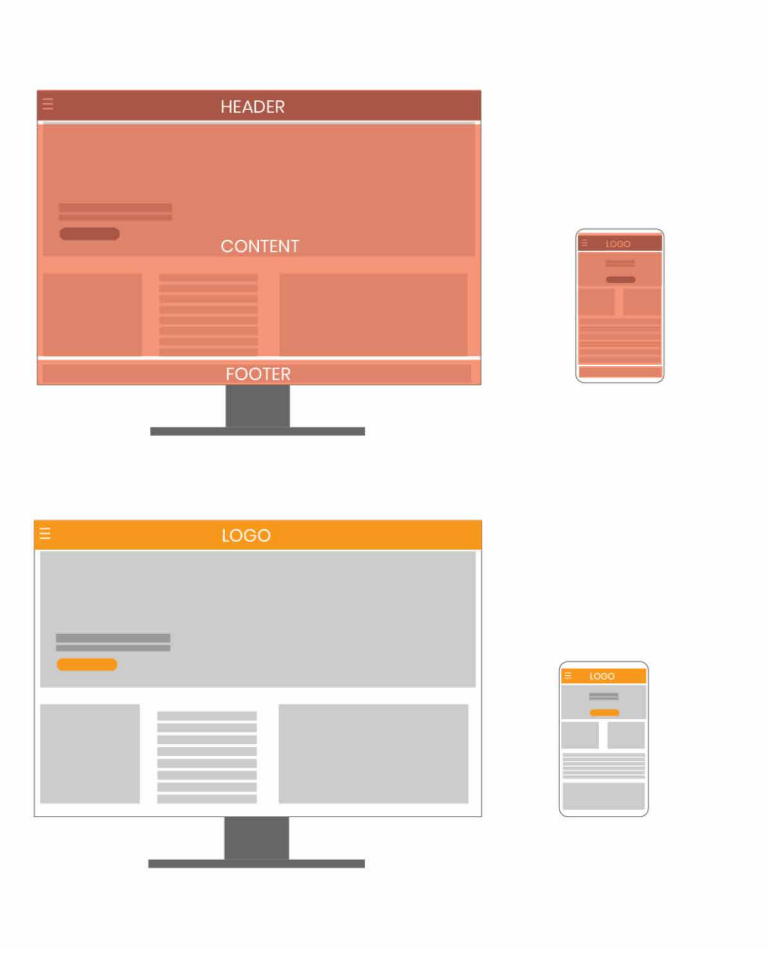


Figure 4-22. *UI regions, web and desktop application*

The Patterns

This chapter's patterns give you specific ways to put all these layout concepts into play. The first three address the visual hierarchy of the entire page, screen, or window, regardless of the type of content you put into that page. You should consider *Visual Framework* fairly early in a project because it affects all of the major pages and windows in an interface.

Layout

The following patterns are most commonly used for desktop and web applications. If you are primarily showing search results, a *Grid of Equals* is a good choice. However, if your application is task or productivity or a creation tool, *Center Stage* might make the most sense. Whatever you choose, make sure that choice is driven by the content you need to show for the user to achieve the desired objective.

- *Visual Framework*
- *Center Stage*
- *Grid of Equals*

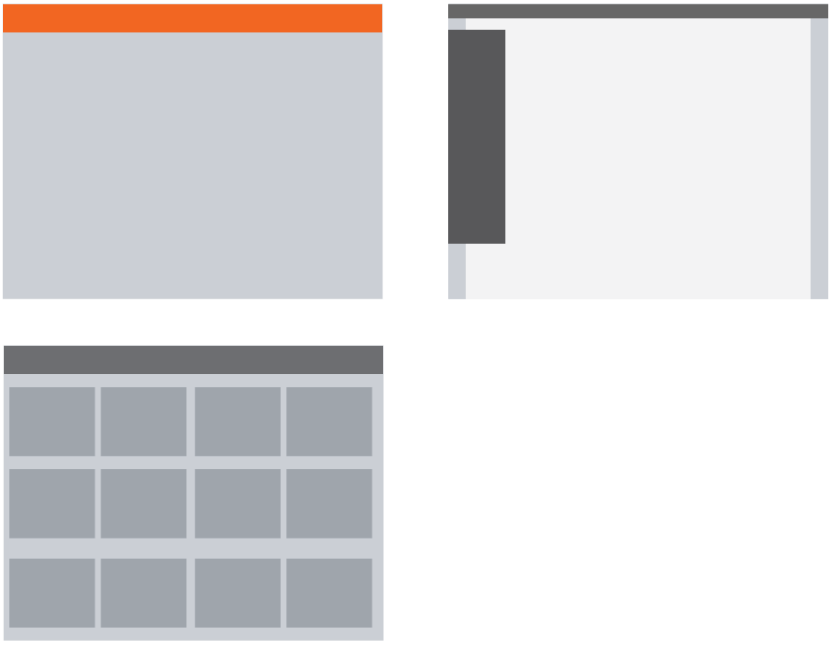


Figure 4-23. *Visual Framework (upper left), Center Stage with a panel (upper right), Grid of Equals (lower left)*

Chunking Information

The next group of patterns represents alternative ways of "chunking" content on a page or window. This is useful when you have more content than you can comfortably put on the screen at one time. These patterns deal with visual hierarchy, too, but

they also involve interactivity, and they can help you choose among the specific mechanisms available in UI toolkits. Here are the patterns we look at in this section:

- *Titled Sections*
- *Module Tabs*
- *Accordion*
- *Collapsible Panels*
- *Movable Panels*

Visual Framework

What

Across an entire app or site, all screen templates share common characteristics to maintain a consistent layout and style. Each page might use the same basic layout, margin, header and gutter size, colors, and stylistic elements, but the design gives enough flexibility to handle varying page content.

Use when

You're building a website with multiple pages or a UI with multiple windows—in other words, almost any complex software. You want it to "hang together" and look like one thing, deliberately designed; you want it to be easy to use and navigate.

Why

When a UI uses consistent color, font, layout, and when titles and navigational aids—signposts—are in the same place every time, users know where they are and where to find things. They don't need to figure out a new layout each time they switch context from one page or window to another.

A strong visual framework, repeated on each page, helps the page content stand out more. That which is constant fades into the background of the user's awareness; that which changes is noticed. Furthermore, adding enough character to the design of the visual framework helps with the branding of your website or product—the pages become recognizable as yours.

How

Draw up an overall look-and-feel that will be shared among all pages or windows. Home pages and main windows are "special" and are usually laid out differently from inner pages, but they should still share certain characteristics with the rest of the site, such as:

Color
: Backgrounds, text colors, accent colors, and other colors

Fonts
: For titles, subtitles, ordinary text, callout text, callout text, and minor text

Writing style and grammar
: Titles, names, content, short descriptions, any long blocks of text, and anything else that uses language.

In a *Visual Framework* design like that of JetBlue, shown in Figure 4-24, all pages or windows share the following, as appropriate:

- "You are here" signposts, such as titles, logos, *Breadcrumbs* trails, global navigation with indicators of the current page, and *Module Tabs*
- Navigational devices, including global and utility navigation, OK/Cancel buttons, Back buttons, Quit or Exit buttons, and navigational patterns such as *Progress Indicator* and *Breadcrumbs* (Chapter 3)
- Techniques used to define *Titled Sections*
- Spacing and alignment, including page margins, line spacing, the gaps between labels and their associated controls, and text and label justification
- The overall layout, or the placement of things on the page, in columns and/or rows, taking into account the margins and spacing issues listed previously

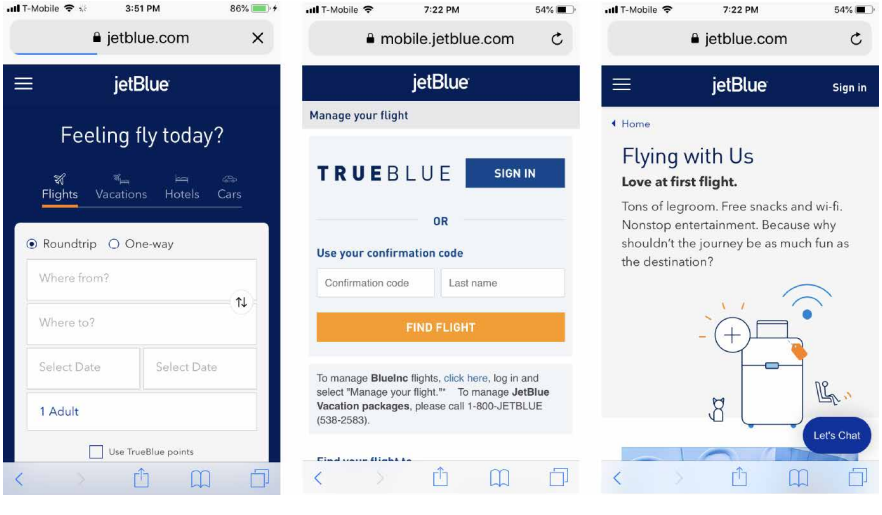


Figure 4-24. *Example of a Visual Framework in JetBlue's mobile website*

Implementation of a *Visual Framework* should force you to separate stylistic aspects of the UI from the content. This isn't a bad thing. If you define the framework in only one place—such as a CSS stylesheet, a Java class, or a visual system library—it lets you change the framework independently from the content, which means that you can more easily tweak and adjust it to get it as you want it. (It's also good software engineering practice.)

Examples

JetBlue's site (Figure 4-25) employs a restricted color palette, a strong header, and consistent use of fonts and curved rectangles in its *Visual Framework*. Even the login page and modal dialogs use these elements; they don't look out of place. Notice the stylistic similarities to the mobile site.

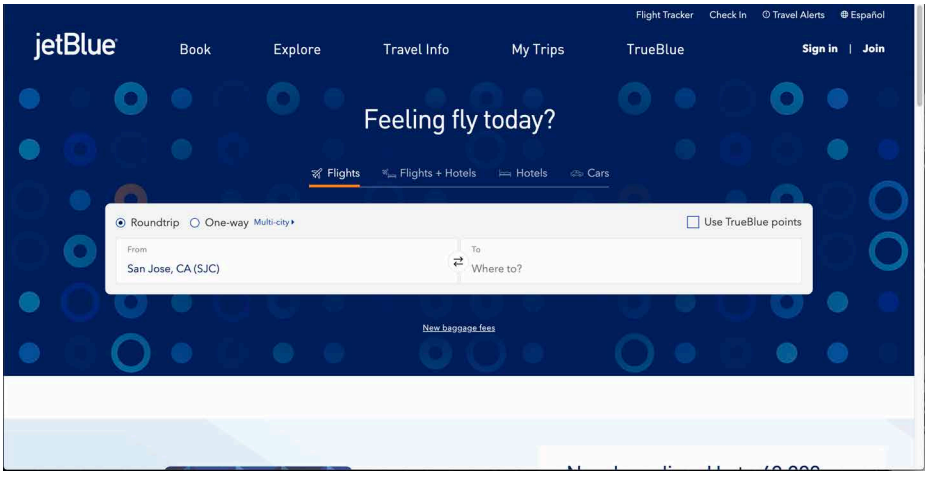


Figure 4-25. *The JetBlue home page*

In the same way, TED's site (Figure 4-26) uses limited color and a layout grid to maintain consistency.

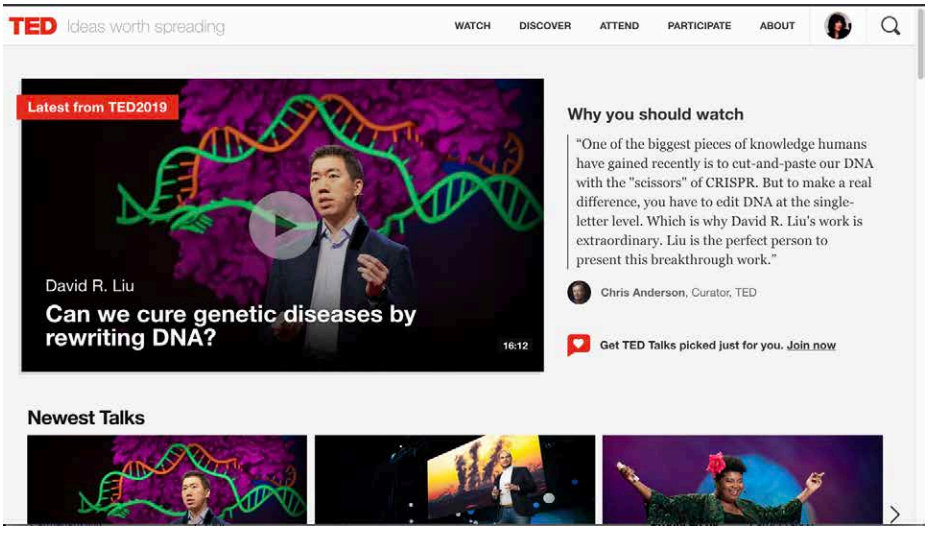


Figure 4-26. *The TED home page*

Center Stage

What

The task at hand is placed front and center at most times in the user experience. This type of layout puts the most important part of the UI into the largest subsection of the page or window, clustering secondary tools and content around it in smaller panels.

Use when

The screen's primary job is to show a single unit of coherent information to the user, let them edit a document, or enable them to perform a certain task. Other content and functions are secondary to this one. Many types of interfaces can use a *Center Stage*—tables and spreadsheets, forms, and graphical editors all qualify. So do web pages that show single articles, images, or features.

Why

The design should guide the user's eyes immediately to the beginning of the most important information (or task) rather than have them wandering over the page in confusion. An unambiguous central entity anchors the user's attention. Just as the

lead sentence in a news article establishes the subject matter and purpose of the article, so the entity in *Center Stage* establishes the purpose of the UI.

After that's done, the user will assess the items in the periphery in terms of how they relate to what's in the center. This is easier for the user than repeatedly scanning the page, trying to figure it out. What comes first? What's the second? How does this relate to that? And so on.

How

Establish a visual hierarchy with the primary content or document dominating everything else. (See the chapter introduction for a discussion of visual hierarchy.) When designing a *Center Stage*, consider these particular factors, though none of them are absolutely required:

Size
: The *Center Stage* content should be at least twice as wide as whatever's in its side margins, and twice as tall as its top and bottom margins. (The user can change its size in some UIs, but this is how it should be when the user first sees it.) Keep "the fold" in mind—when a small screen is used, where does the content cut off at the bottom? Make sure the *Center Stage* still takes up more of the "above-the-fold" space than anything else.

Headlines
: Big headlines are focal points and can draw the user's eye to the top of the *Center Stage*. That happens in print media, too, of course. See the chapter introduction and *Titled Sections* for more.

Context
: What is the primary task of the product? This will inform what the user will expect to see when they open the screen. Is it a graphic editor? A long text article? A map? A filesystem tree?

Notice that I didn't mention one traditional layout variable: position. It doesn't much matter where you put the *Center Stage*—top, left, right, bottom, center; it can be made to work. If it's big enough, it ends up more or less in the center anyway. Note that well-established genres have conventions about what goes into which margins, such as toolbars on top of graphic editors, or navigation bars on the left side of web or mobile screens.

Examples

The Google Docs text editor (Figure 4-27) devotes almost all of its horizontal space to the document being edited; so does Google's spreadsheet editor. Even the tools at the top of the page don't take up a huge amount of space. The result is a clean and balanced look.

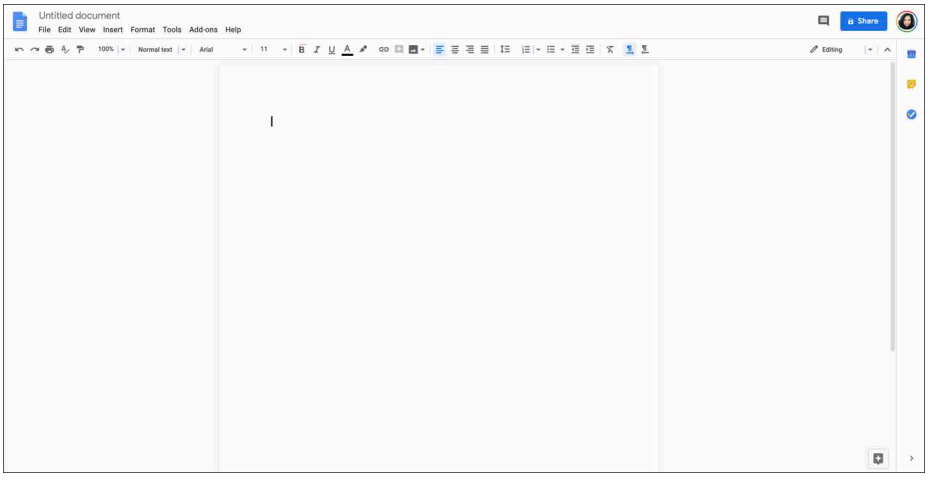


Figure 4-27. *Google Docs*

The Sketch desktop application (Figure 4-28) features a *Center Stage* layout. Upon launching a new blank document or template, the user sees a canvas where they are able to have the visual focus on creating content unencumbered by unneeded content or features.

Because most of the products in the Google Suite are task-based, *Center Stage* is a common framework found in most of those products (Figure 4-29), such as Google Earth, Google Slides, Google Hangout, and Google Sheets.

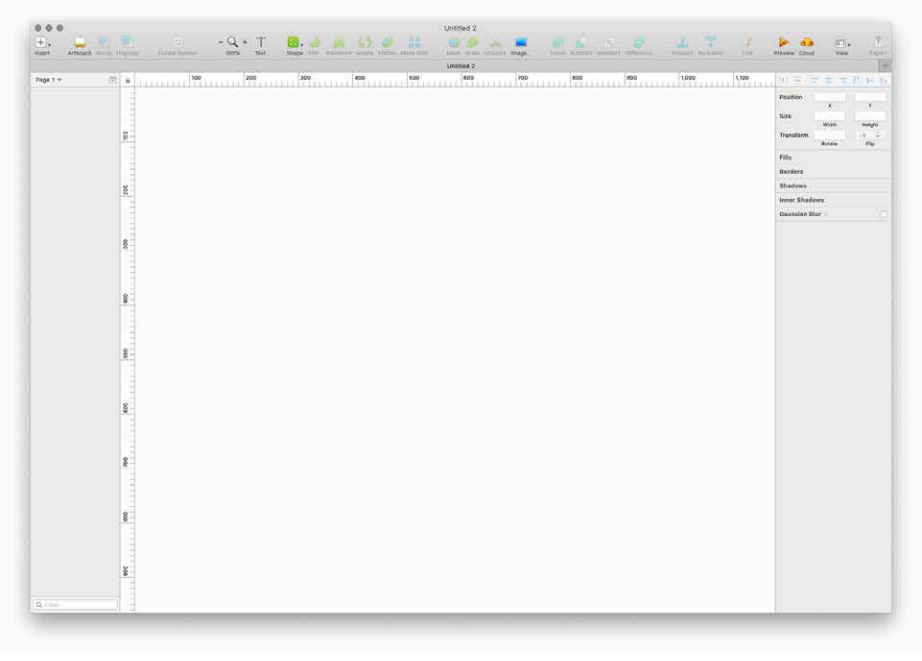


Figure 4-28. *Bohemian Sketch*

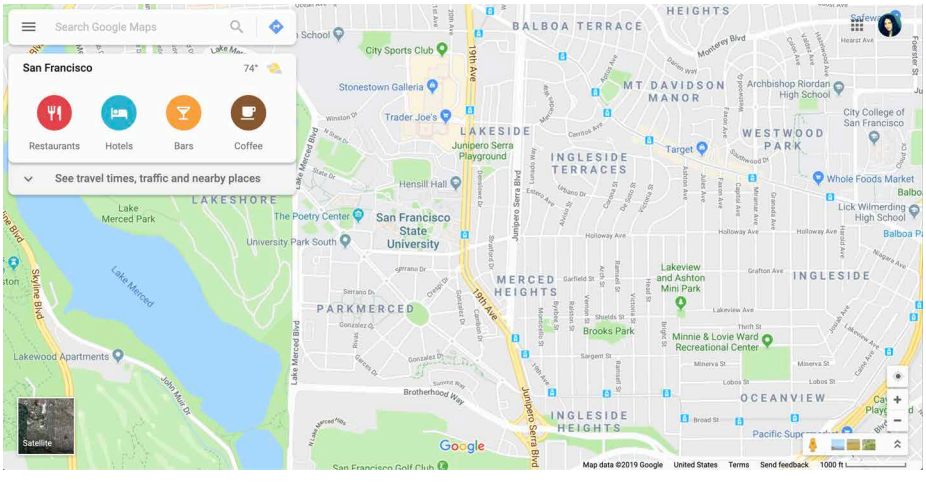


Figure 4-29. *Google Maps*

For more information on Google's design language, go to *https://material.io/design*.

Grid of Equals

What

Arrange content items, such as search results, into a grid or matrix. Each item should follow a common template, and each item's visual weight should be similar. Link to item pages as necessary.

Use when

The page contains many content items that have similar style and importance, such as news articles, blog posts, products, or subject areas. You want to present the viewer with rich opportunities to preview and select these items.

Why

A grid that gives each item equal space announces that they have equal importance. The common template for items within the grid informs the user that the items are similar to one another. Together, these techniques establish a powerful visual hierarchy that should match the semantics of your content.

How

Figure out how to lay out each item in the grid. Do they have thumbnail images or graphics? Headlines, subheads, summary text? Experiment with ways to fit all the right information into a relatively small space—tall, wide, or square—and apply that template to the items you need to display. Arrange content items in a grid or matrix. Each item should follow a common template, and each item's visual weight should be similar.

Now arrange the items in a grid. You could use a single row or a matrix that's two, three, or more items wide. Consider page width as you do this design work—what will your design look like in a narrow window? Will most of your users have large browser windows or use mobile or other devices (Figure 4-30)?

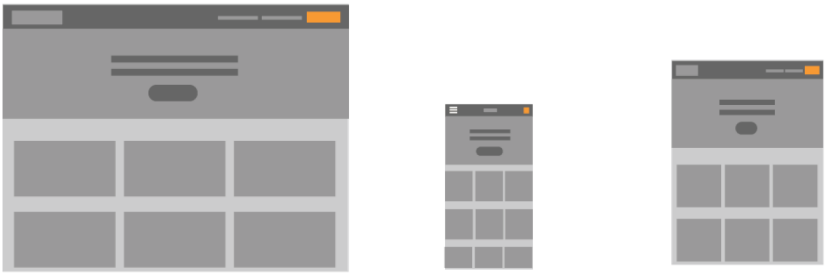


Figure 4-30. *Responsive design example: desktop version (left), mobile version (center), tablet version (right)*

You might choose to highlight grid items statically (to emphasize one item over others) or dynamically, as a user hovers over those grid items. Use color and other stylistic changes, but don't change the positions, sizes, or other structural elements of the grid items.

Examples

In the Hulu example (Figure 4-31), the size and relative importance of each item in the grid is the same and has a consistent interaction behavior.

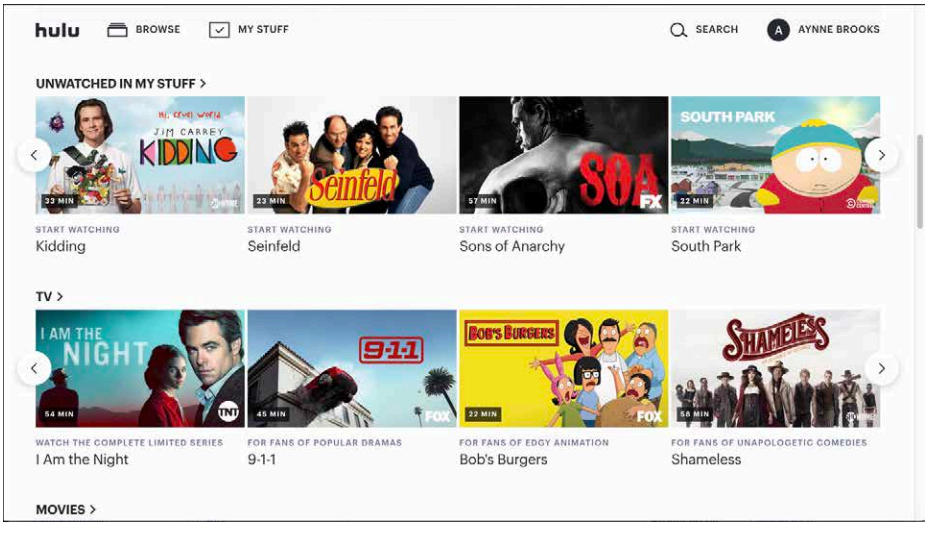


Figure 4-31. *Hulu grid*

In CNN's layout (Figure 4-32) and Apple TV's layout (Figure 4-33), the consistent visual treatment marks these items as peers of one another. An advantage of using grids to display lists of items is that a user of this interface will need to interact with only one grid item to understand how all of the grid items behave.

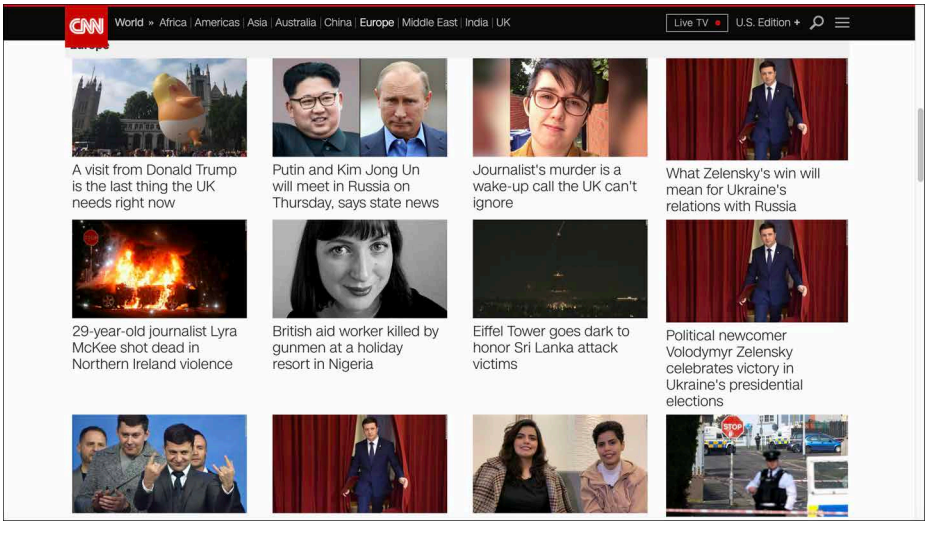


Figure 4-32. *CNN's grid*

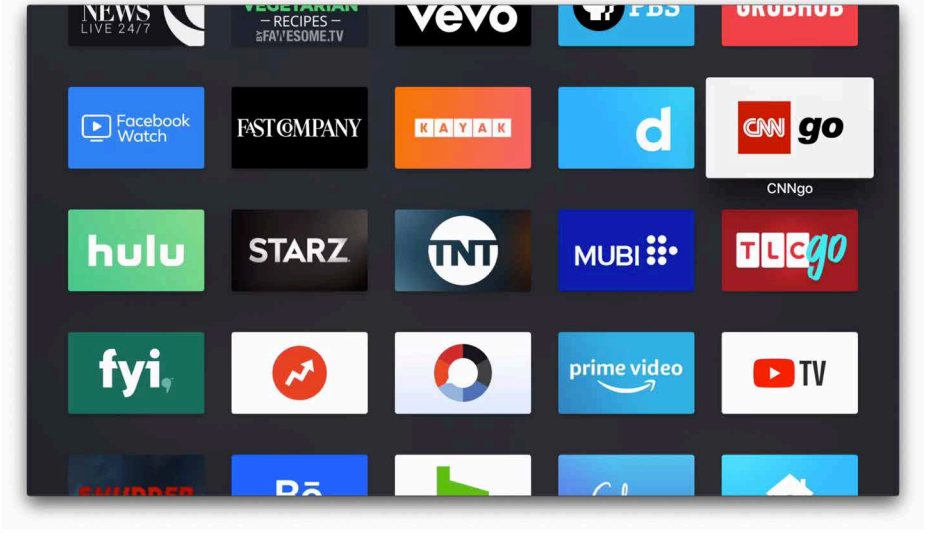


Figure 4-33. *Apple TV's grid*

Titled Sections

What

Define separate sections of content by giving each one a visually strong title, separating the sections visually, and arranging them on the page.

Use when

You have a lot of content to show, but you want to make the page easy to scan and understand, with everything visible. You can group the content into thematic or task-based sections that make sense to the user.

Why

Well-defined and well-named sections structure the content into easily digestible "chunks," each of which is now understandable at a glance. This makes the information architecture obvious. When the user sees a page sectioned neatly into chunks like this, their eye is guided through the page more comfortably.

How

First, get the information architecture right—split up the content into coherent chunks (if it hasn't already been done for you) and give them short, memorable names (see Figure 4-34):

- For titles, use typography that stands out from the rest of the content—bolder, wider, larger point size, stronger color, different font family, outdented text, and so on. See the chapter introduction for more on visual hierarchy.
- Try reversing the title against a strip of contrasting color.
- Use whitespace to separate sections.
- Use blocks of contrasting background color behind the entire section.
- Boxes made from etched, beveled or raised lines are familiar with desktop UIs. But they can get lost (and just become visual noise) if they're too big, too close to one another, or deeply nested.

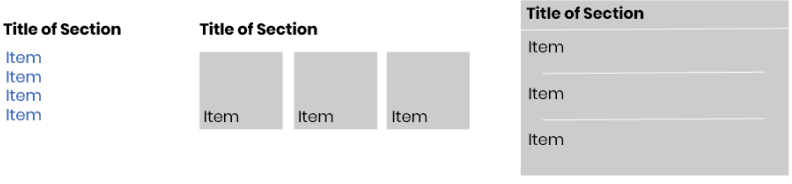


Figure 4-34. *Examples of titled sections*

If the page is still too overwhelming, try using *Module Tabs*, an *Accordion*, or *Collapsible Panels* to hide some of the content.

If you're having trouble giving reasonable titles to these chunks of content, that might be a sign that the grouping isn't a natural fit for the content. Consider reorganizing it into different chunks that are easier to name and remember. "Miscellaneous" categories may also be a sign of not-quite-right organization, though sometimes they're genuinely necessary.

In its account settings page, Amazon (Figure 4-35) shows titles corresponding to levels of the visual hierarchy: the page title, section titles, and subtitles atop lists of links. Note the use of whitespace, boxes, and alignment to structure the page.

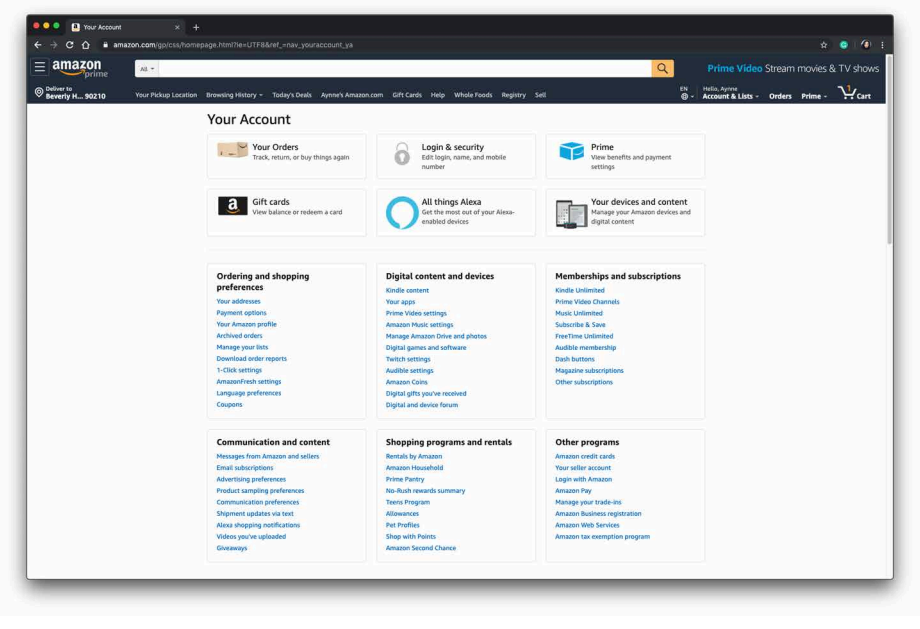


Figure 4-35. *Amazon account settings*

Google's settings also feature *Titled Sections* (Figure 4-36). Some contain functionality; others deep-link to other settings pages.

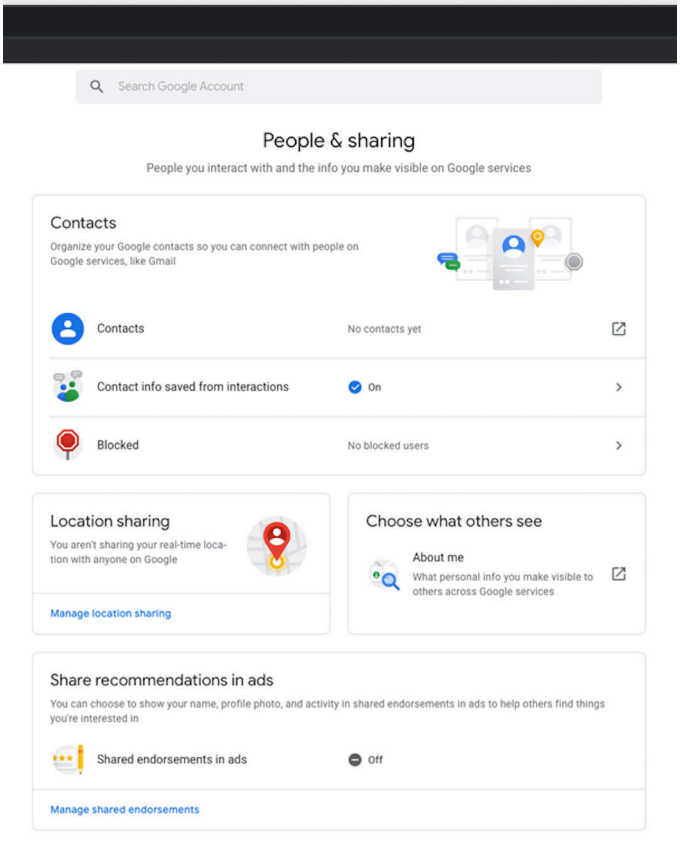


Figure 4-36. *Google account settings*

Module Tabs

What

Put modules of content into small tabbed areas so that only one module is visible at a time. The user clicks or taps on tabs to bring different content to the top.

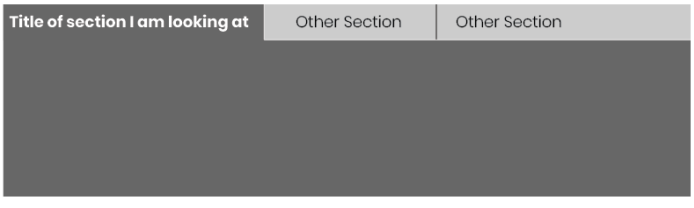


Figure 4-37. *The Module Tabs pattern*

Use when

You have a lot of heterogeneous content to show on the page, possibly including text blocks, lists, buttons, form controls, or images, and you don't have room for everything. Some of the page content comes in groups or modules (or can be sorted into coherent groups). Those modules have the following characteristics:

- Users need to see only one module at a time.
- They are of similar length and height.
- There aren't many modules—fewer than 10; preferably a small handful.
- The set of modules is fairly static; new pages won't be added frequently nor will existing pages change or be removed frequently.
- The modules' contents can be related to or similar to one another.

Why

Grouping and hiding chunks of content can be a very effective technique for decluttering an interface. Tabs work well; so do *Accordions*, *Movable Panels*, *Collapsible Panels*, and simply arranging things into a clean grid of *Titled Sections*.

How

First, get the information architecture right. Split up the content into coherent chunks, if it hasn't already been done for you, and give them short, memorable titles (one or two words, if possible). Remember that if you split up the content incorrectly, users will be forced to switch back and forth between tabs as they compare them or

look for information they can't find. Be kind to your users and test the way you've organized it.

Indicate the selected tab unambiguously, such as by making it contiguous with the panel itself. (Color alone isn't usually enough. If you have only two tabs, make sure it's abundantly clear which one is selected and which one isn't.)

The tabs don't need to be literal tabs, and they don't have to be at the top of the stack of modules. You can put them in a left-hand column, or underneath, or even turned 90 degrees with the text read sideways.

When deployed on web pages, *Module Tabs* tend to be distinct from navigational tabs (those used for global navigation, or separate documents, or for loading new pages). Tabs are useful there, too, of course, but this pattern is more about giving the user a lightweight way to see alternative modules of content within a page.

If there are too many tabs to fit in a narrow space, you could do one of several things: shorten the labels by using an ellipsis (and thus make each tab narrower), or use carousel-like arrow buttons to scroll the tabs. You could also put the tab labels in a left-hand column, instead of putting them on top. Never double-row the tabs.

Examples

Expedia's flight search module (Figure 4-38) breaks up the different types of searches available into tabs. This allows Expedia to feature the options available to the potential customer in a highly discoverable way without sacrificing valuable screen real estate.

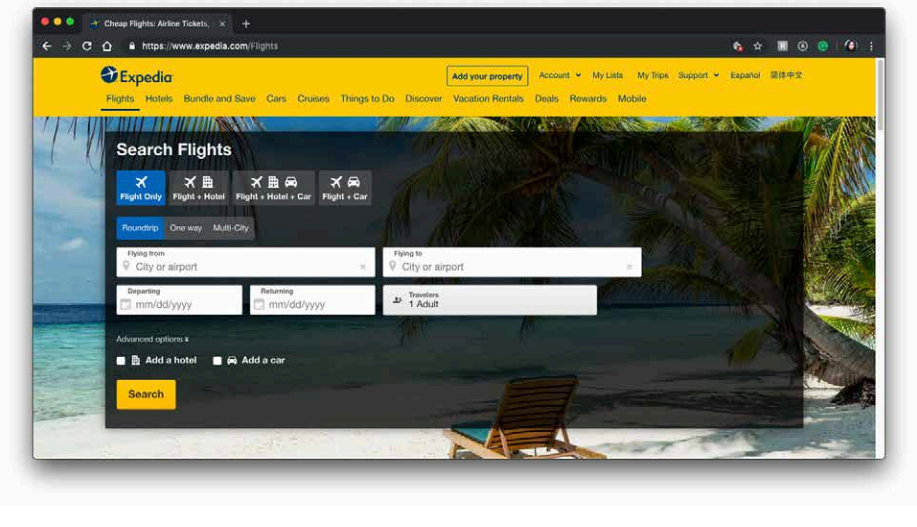


Figure 4-38. *Expedia Search*

MacOS system preferences (Figure 4-39) also use *Module Tabs* to highlight functionality in the most logical place a user might look for it. The tabs are across the top, labeled Battery and Power Adapter.

Figure 4-39. *macOS system preferences*

Accordion

What

Put modules of content into a collinear stack of panels that the user can close and open independently of one another (Figure 4-40).

Figure 4-40. *Examples of accordions*

Use when

You have a lot of heterogeneous content to show on the page, possibly including text blocks, lists, buttons, form controls, or images, and you don't have room for everything. Some of the page content comes in groups or modules (or can be sorted into coherent groups).

Those modules have the following characteristics:

- Users might want to see more than one module at a time.
- Some modules are much taller or shorter than others, but they're all of a similar width.
- The modules are part of a tool palette, a two-level menu, or some other coherent system of interactive elements.
- The modules' contents might be otherwise related or similar.
- You might want to preserve the linear order of the modules.

Also note that when large modules are open or many modules are open, the labels on the bottom of the *Accordion* can scroll off the screen or window. If that's a problem for your users, consider using a different solution.

Why

Accordions have become a familiar interactive element on web pages, almost as familiar as tabs and drop-down menus. (They aren't quite as straightforward to use, however.) Many websites use accordions in their menu systems to manage very long lists of pages and categories.

In general, grouping and hiding chunks of content can be a very effective technique for decluttering an interface. The *Accordion* pattern is part of a toolkit that includes *Module Tabs*, *Movable Panels*, *Collapsible Panels*, and *Titled Sections* to do so.

Accordions can be useful in web-page navigation systems, but they really shine in desktop applications. Tool palettes in particular work well with *Accordions* (and *Movable Panels*, as well, for similar reasons). Because users can open any set of modules and leave them open, *Accordions* help users modify their "living space" in a way that suits them. Yet it's easy to reopen a rarely used module when it becomes needed.

How

Establish the title that you want to name each section. It should be concise but enough information to know what the information under it will be.

Create a visual affordance (an indication that represents how something is meant to be used) such as an arrow or triangle icon to indicate that there is information that can be revealed by clicking or tapping on it.

Allow more than one module to be open at a time. There are differing opinions on this—some designers prefer only one module to be open at a time, and some implementations allow only one (or have a switch that developers can set, at least). But in my experience, especially in applications, it's better to let users open multiple modules at a time. It avoids the abrupt and unexpected disappearance of a previously open module, and allows users to compare content in multiple modules without repeatedly opening and closing modules.

When used in an application or when the user is signed in to a website, an *Accordion* ought to preserve its state of opened and closed modules between sessions. This isn't as important for navigation menus as it is for tool palettes.

Examples

Samsung's FAQs in its Help pages use accordions to display the question and reveal the answer upon clicking on the expand arrow (Figure 4-41). This allows the user to quickly scan the page for the information they are seeking and quickly navigate to another topic if they need to.

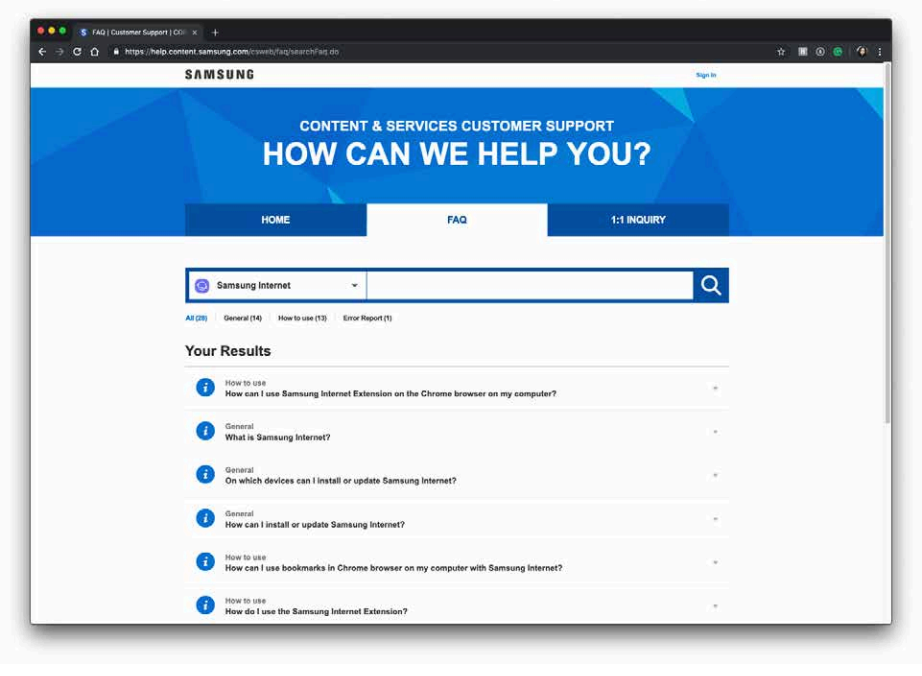


Figure 4-41. *Samsung Help*

Google Chrome's settings page uses expanding accordions to reveal the detailed setting options (Figure 4-42). This enables the user to see all the options "above the fold" and get a better idea of where to click.

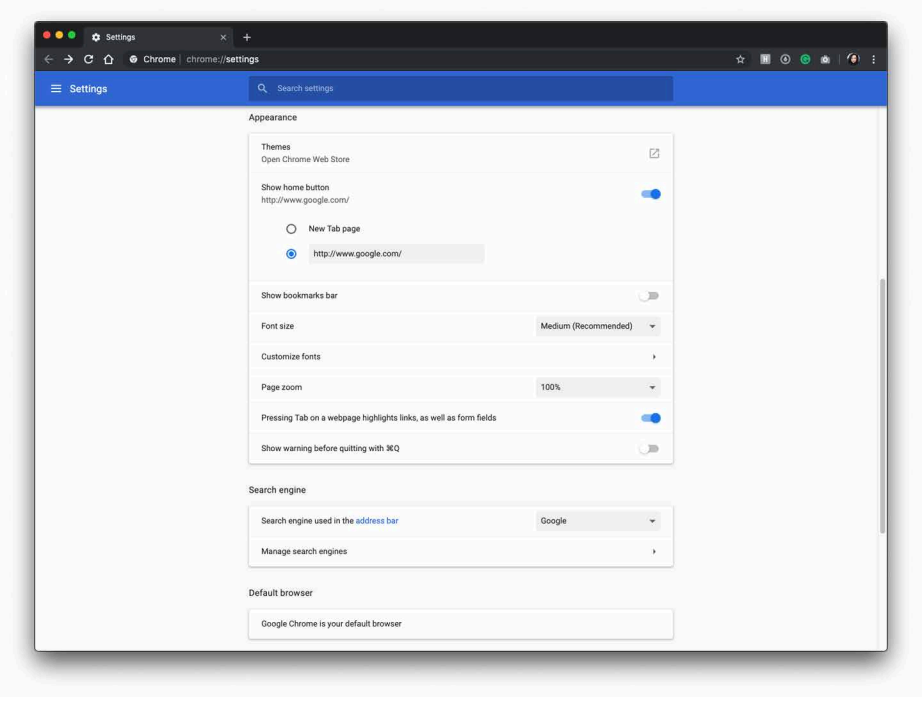


Figure 4-42. *Google Chrome Settings*

Collapsible Panels

What

Put modules of secondary or optional content or functions into panels that can be opened and closed by the user.

Use when

You have a lot of heterogeneous content to show on the page, possibly including text blocks, lists, buttons, form controls, or images or when you have *Center Stage* content that needs to take visual priority.

These modules have the following characteristics:

- Their content annotates, modifies, explains, or otherwise supports the content in the main part of the page.
- The modules might not be important enough for any of them to be open by default.
- Their value can vary a lot from user to user. Some will really want to see a particular module; others won't care about it at all.
- Even for one user, a module might be useful sometimes, but not other times. When it's not open, its space is better used by the page's main content.
- Users might want to open more than one module at the same time.
- The modules have very little to do with one another. When *Module Tabs* or *Accordions* are used, they group modules together, implying that they are somehow related; *Collapsible Panels* do not group them.

Why

Hiding noncritical pieces of functionality or content helps to simplify the interface. When a user hides a module that supports the main content, it simply collapses, giving its space back over to the main content (or to whitespace). This is an example of the principle of Progressive Disclosure (showing hidden content when and where the user needs it). In general, grouping and hiding chunks of content can be a very effective technique for decluttering an interface.

How

Put each supporting module into a panel that the user can open and close via a single click. Label the button or link with the module's name or simply "More," and consider using a chevron, menu icon, or rotating triangle to indicate that more content is hidden there. When the user closes the panel, collapse the space used by that panel

and devote it to other content (such as by moving up the content below it on the page).

Animating the panels as they open and close helps the user create a visual and spatial understanding of how this functions and where to find it in the future.

If you find that most users are opening up a *Collapsible Panel* that's closed by default, switch to open by default.

Examples

The Apple news application (Figure 4-43) uses a left-hand panel as an extensible way for the user to add or remove news and topic channels and a way to navigate through those channels. When the user wants to focus on the content alone, they can tap the panel icon (to the left of "Today," on the center header) to slide the panel away to the left.

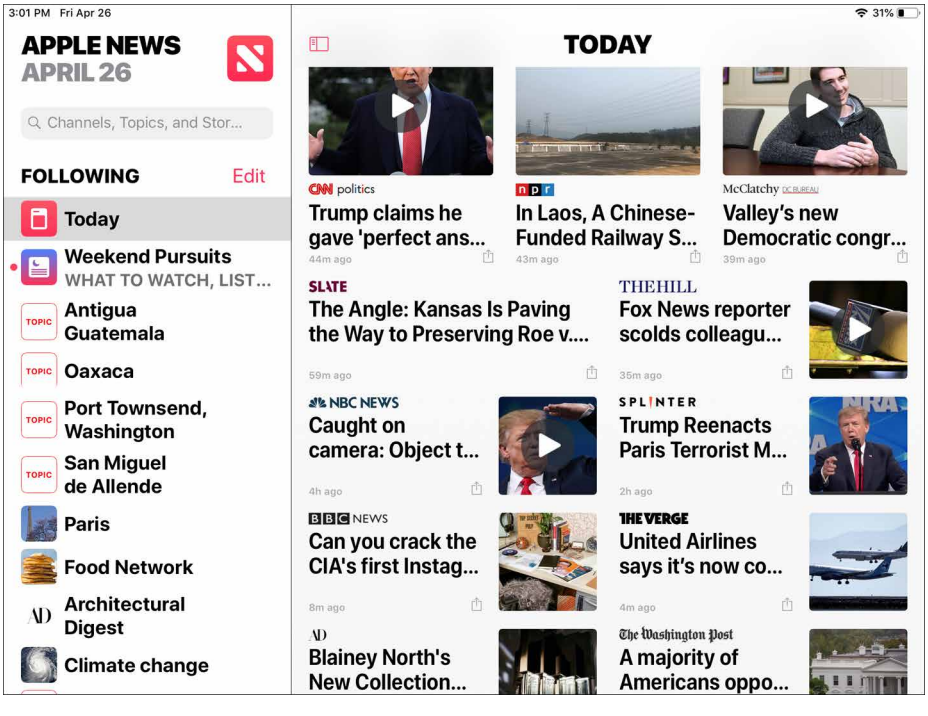


Figure 4-43. *Apple News, iPad version, with navigation panel expanded*

Google Maps (Figure 4-44) allows for the user to continue to see the map while finding directions, once the destination directions are selected, the panel collapses to allow for viewing the directions unobstructed. If the user needs to edit or change or add a stop, they can easily expand the side panel.

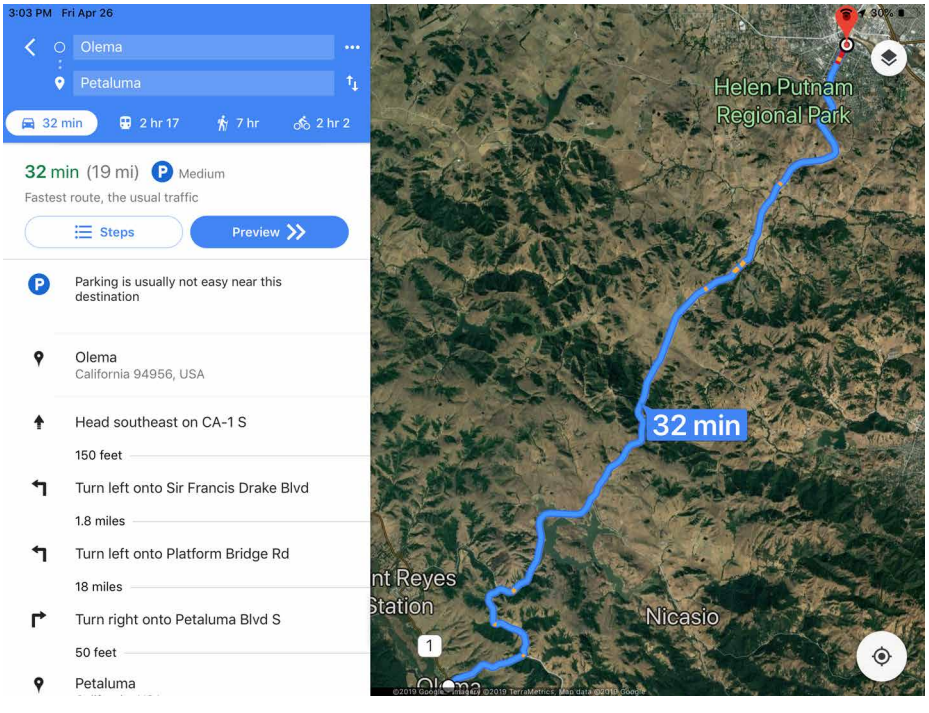


Figure 4-44. *Google Maps, iPad version, showing left-hand panel with direction functionality*

Movable Panels

What

Put modules of content in boxes that can be opened and closed independently of one another. Allow the user to arrange the boxes freely on the page into a custom configuration. You will often see movable panels in the *Center Stage* layout in creator tools like the Adobe Suite (Figure 4-45), and productivity and communication applications like Microsoft Excel and Skype.

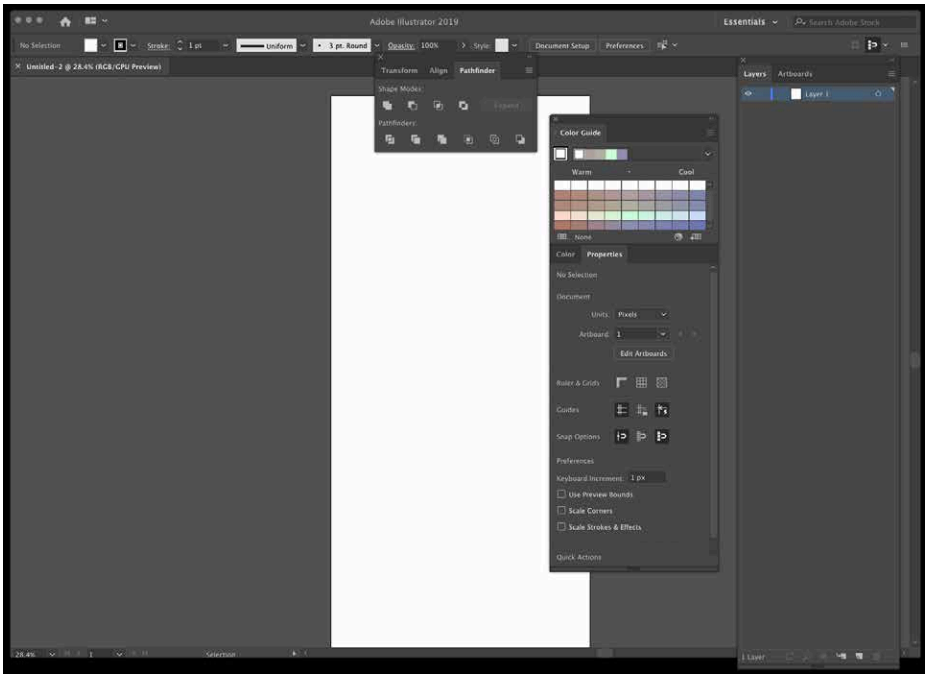


Figure 4-45. *Movable Panels in Adobe Illustrator*

Use when

You're designing either a desktop application or a website that most users sign in to. News portals, dashboards, and canvas-plus-palette apps often use *Movable Panels*. The screen you are designing is a major part of the app or site—something that users see often or for long periods of time. You have a lot of heterogeneous content to show on the page, possibly including text blocks, lists, buttons, form controls, or images.

Modules have some of the following characteristics:

- Users will almost certainly want to see more than one module at a time.
- Their value can vary a lot from user to user. Some people want modules A, B, and C, whereas others don't need those at all and want to see only D, E, and F.
- The modules vary a lot in size.
- Their position on the page isn't terribly important to you, but it might be to users. (By contrast, a page of static *Titled Sections* ought to be arranged with thought given to the importance of page position; important things go to the top, for instance.)
- There are many modules—possibly so many that if all were shown at once, a viewer would be overwhelmed. Either you or the user should pick and choose among them.
- You're willing to let users hide some modules from view altogether (and offer a mechanism to bring them back).
- The modules can be part of the tool palette or some other coherent system of interactive elements.

Why

Different users have different interests. Websites such as dashboards and portals are most useful to people when they can choose the content they see. When they'll be working on something for a while in a desktop application, people like to rearrange their environment to suit their working style. They can place needed tools close to where they work; they can hide things they don't need; they can use *Spatial Memory* to remember where they put things.

Rationally speaking, *Movable Panels* help users get things done more efficiently and comfortably (in the long run—after they've spent time rearranging their environment the way they like it.). But this kind of personalization seems to appeal to people on some other level, too. They might do this on infrequently visited websites that provide some kind of entertainment, for instance. Personalization can increase engagement and buy-in.

Finally, a *Movable Panels* design easily accommodates new modules introduced over time, such as those contributed by third parties.

How

Give each module a name, a title bar, and a default size, and arrange them on the screen in a reasonable default configuration. Let the user move modules around the page at will, via drag-and-drop if possible. Permit each module to be opened and closed with a simple gesture, such as a mouse click on a title bar button.

Depending upon the design you've chosen, you might want to give the user freedom to place these pieces anywhere at all, even if they overlap. Or you might want a predefined layout grid with "slots" where pieces can be dragged and dropped—this lets the screen maintain alignment (and some sense of dignity) without making the user spend too much time fiddling with windows. Some designs use *ghosting*—big drop targets that appear dynamically; for example, dotted rectangles—to show where a dragged module would go when dropped.

Consider letting users remove modules altogether. An "X" button in the title bar is the familiar way to remove one. After a module is gone, how does a user bring it back? Let users add modules—including brand-new ones, perhaps—from a list of available modules that can be browsed and searched.

Chapter 5
Visual Style and Aesthetics

"Never underestimate the power of beauty."

Visual design is more than just "skinning a user interface"; visual design and look-and-feel, done well, can make a digital product stand out. The visual language used in any given interface conveys the attitude and spirit of your brand, and stands as an avatar of it across various touchpoints. Visual design can make or break the usability of a product and trust in your brand.

In 2002, a research group discovered something interesting. The Stanford Web Credibility Project set out to learn what causes people to trust or distrust websites. Much of what they found made intuitive sense: company reputation, customer service, sponsorships, and ads all helped users decide whether a website is credible.

But the most important factor—number one on their list—was the appearance of the website. Users did not trust sites that looked amateurish. Sites that made the effort to craft a nice, professionally designed look made a lot more headway with users, even if those users had few other reasons to trust the site.

What was true then is true now; looking good matters.

True beauty is the combination of the physical form and desired function operating together in harmony. In digital design, it is not enough that each pixel is perfect but it also must add usefulness, understanding, or delight and often a combination of all three.

In this chapter, we go over the core elements of visual design and discuss what makes a visual design aesthetically pleasing.

The Basics of Visual Design

In this chapter, we talk about some of the principles of good visual design:

- Visual hierarchy
- Composition
- Color
- Typography
- Readability
- Evoking a feeling
- Images

Take a look at the four examples in Figure 5-1. They might look like different designs but they all contain mostly the same visual elements. Using only color and changes to the text, they achieve such different impressions—a screen's color scheme can cause you to either smile or cringe, for example. The impression they give are vastly different, although they contain the same content.

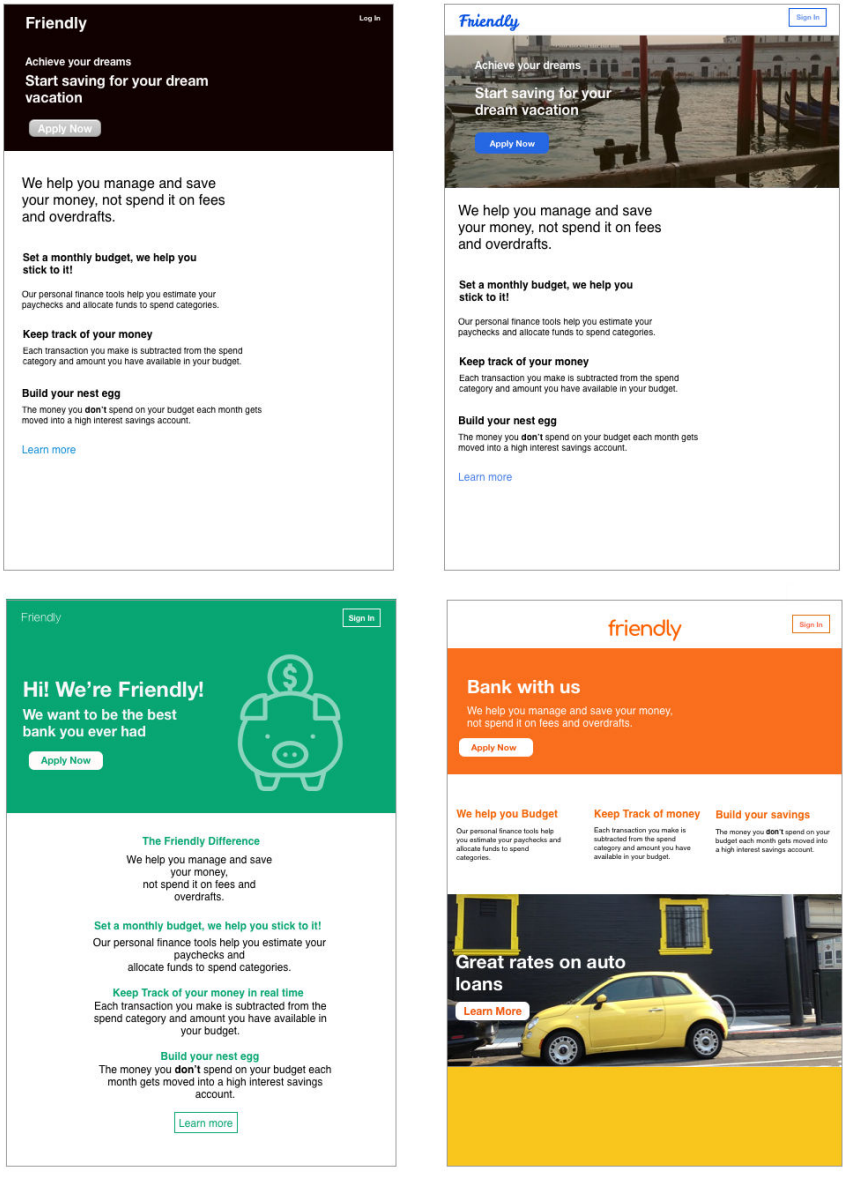


Figure 5-1. *Visual design examples*

Visual Hierarchy

Visual hierarchy refers to presentation of visual elements on any given layout. Let's explore the characteristics of visual hierarchy.

Clarity

How well the design communicates the information the designer is trying to convey.

Actionability

How the user knows what they are supposed to do on any given screen.

Affordance

Affordance means that it looks or behaves like what it does. For example, a button that looks slightly three-dimensional gives a visual cue that it is clickable.

Composition

Composition refers to the arrangement and proportion of a visual design.

Consistency

Visual elements appear in a predictable and uniform visual language. If you employ icons in your interface design that they are always used within your experience to convey the same functionality. The same goes for the use of language in your interface.

Alignment

Nothing is more jarring to a user than moving from screen to screen and having the elements of the screen change for no apparent reason. Make sure the screen elements do not move their place from screen to screen. Text that changes from left, right, or center aligned randomly is also disharmonious to the understandability and legibility of digital product design.

Color

Color is immediate. It's one of the first things you perceive about a design, along with basic forms and shapes. Yet the application of color to art and design is infinitely subtle—master painters have studied it for centuries. We can only scratch the surface here.

When devising a color scheme for an interface, first rule out anything that makes the text difficult to read:

- Always put dark foregrounds against light backgrounds, and vice versa—to test, pull the design into an image tool such as Photoshop and desaturate it (make it grayscale).
- When red or green indicate a critical distinction, make sure to also reinforce the color with a different shape or with text. This is because many color blind people won't be able to see the difference. Statistically, 10% of men have some form of colorblindness, as do about 1% of women.
- Avoid certain color combinations. As an example, bright blue text on a bright red background will fatique the eye. This is because blue and red are complimentary colors, which means they are on opposite sides of the color wheel (Figure 5-2).

Figure 5-2. *A color wheel*

- It is less straining on the eyes to read black text on a white background, however, white backgrounds tend to glow light through them, which can be fatiguing to the eye. So, if you are designing something that will be used on a tablet and you will have lots of blank space around your content or UI elements, try a dark colored background to reduce the backlight glow.

With that out of the way, here are some very approximate rules for color usage.

Warm versus cool

Red, orange, yellow, brown, and beige are considered "warm" colors. Blue, green, purple, gray (in large quantities), and white are considered "cool." See Figure 5-3.

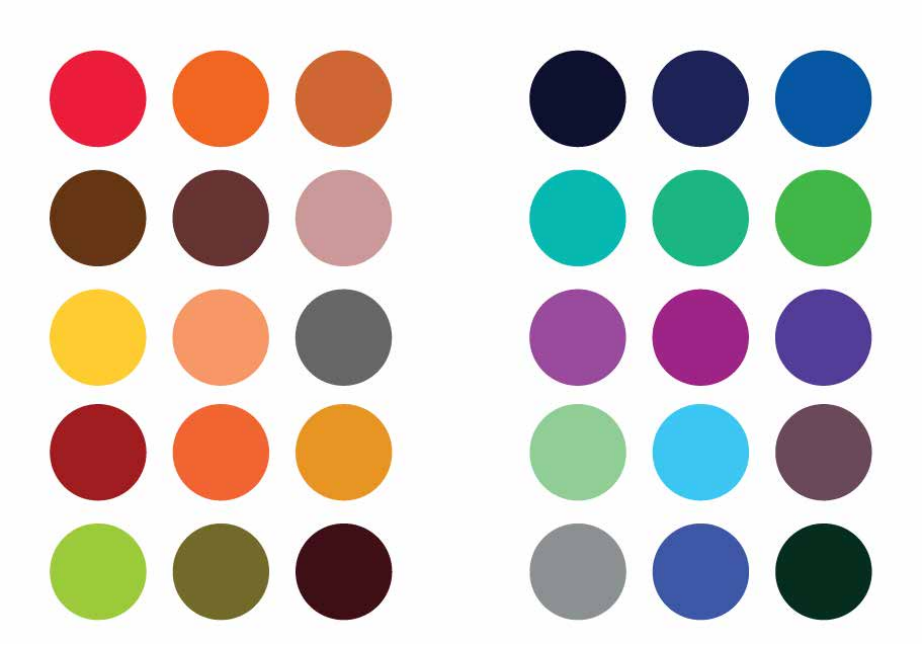


Figure 5-3. *Warm colors versus cool colors*

Dark versus light background

The screens with light backgrounds—white, beige, and light gray—feel very different from those with very dark backgrounds. Light is more typical of computer interfaces (and printed screens); dark screens can feel edgier, more somber, or more energetic, depending on other design aspects. See Figure 5-4.

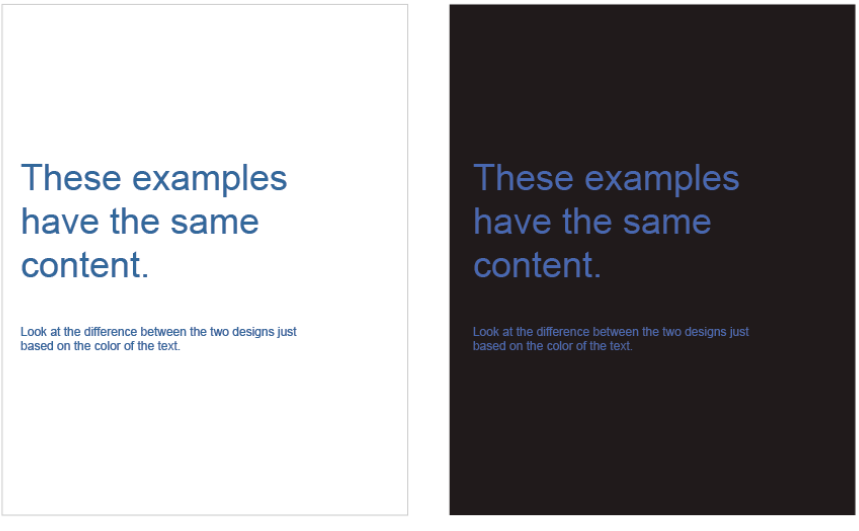


Figure 5-4. *Dark versus light backgrounds*

High versus low contrast

Whether the background is dark or light, the elements on that background might have either high or low contrast against it. Strong contrast evokes tension, strength, and boldness; low contrast is more soothing and relaxing. See Figure 5-5.

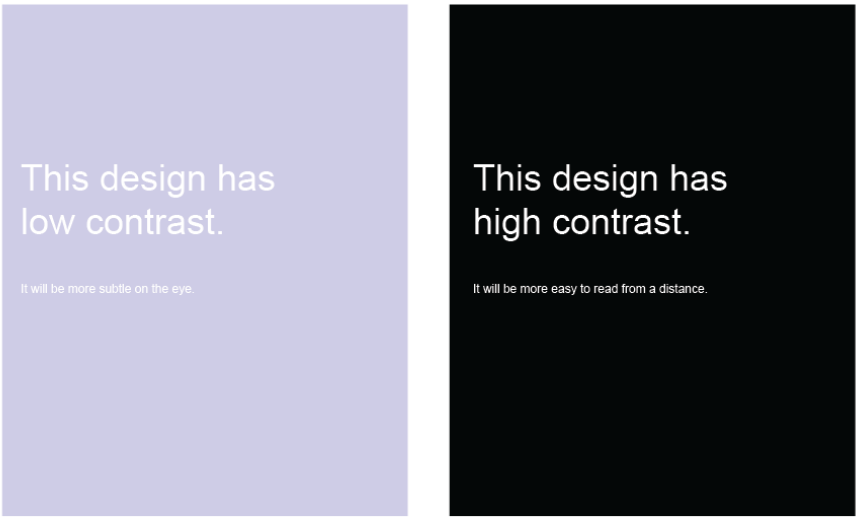


Figure 5-5. *Contrast*

Saturated versus unsaturated

Highly saturated, or pure, colors—brilliant yellows, reds, and greens, for example—evoke energy, vividness, brightness, and warmth. They are daring; they have character. But when overused, they can tire the eyes, so most UI designs use them sparingly; they often choose only one or two. Muted colors, either dark or light (*tones* or *tints*, respectively), make up the bulk of most color palettes. You probably wouldn't want to stare at that pink all day long in a desktop application. See Figure 5-6.

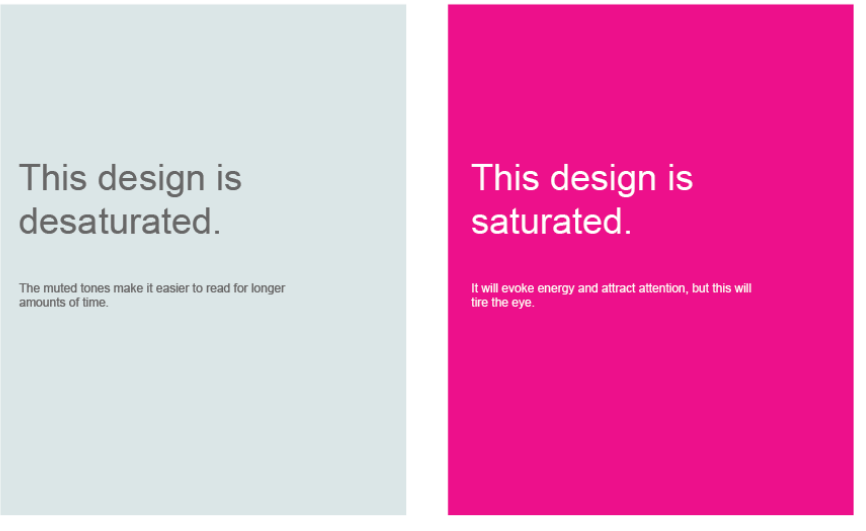


Figure 5-6. *Saturated versus desaturated*

Combinations of hues

When you begin combining colors, interesting effects happen. Two saturated colors can evoke far more energy, motion, or richness than one alone. A screen that combines one saturated color with a set of muted colors directs attention to the saturated color and sets up "layers" of color—the brighter and stronger ones appear closer to the viewer, whereas the grayer and paler colors recede. Strong dimensionality can make a design dramatic. Flatter designs, with more muted or lighter colors, are calmer.

Don't rely on color alone

Color is awesome, but don't rely on color alone to indicate important information. A good example from the real world is a stop sign in the United States (see Figure 5-7). Anywhere you go, the stops signs will always look the same. They are red, they say "STOP" and they are a distinct octagon shape. These create three cognitive cues to indicate what is being asked of the viewer.

Figure 5-7. *Example of a stop sign in the United States*

The same is true in using visual elements in the digital world. Help your users to understand the information you are trying to convey by doubling up with color and shape.

Color references and resources

- *https://www.colorbox.io*
- *https://color.adobe.com/create*
- *http://khroma.co*

Typography

The majority of content on the web, other than cat videos and movies, is textual. Designing for a legible, non–eye straining experience on the digital medium is an art all to itself. Typography is a topic deserving of its own book, and in fact, there are entire categories of books devoted to this topic alone. You can certainly nerd out on typography, as is demonstrated in Figure 5-8. In this section, we introduce some of the most important points to understand typography for digital design.

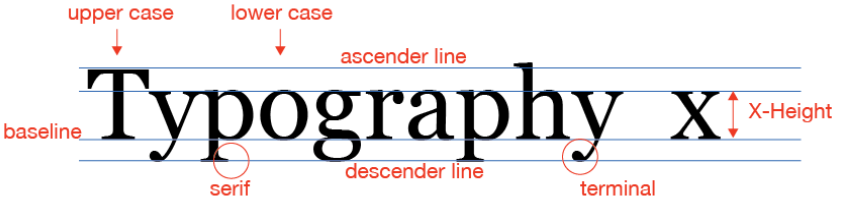


Figure 5-8. *The anatomy of type for digital design*

Types of typefaces

There are a couple of terms that will help in understanding some of the finer points around using type. A *typeface* refers to the name of the particular design of type. For example, Helvetica or Arial or Times New Roman are typefaces. A *font* is a particular size, weight, and style of a typeface; for example, Helvetica Bold 12 pt, or Arial Italic 8pt, or Times New Roman 18 pt. You will often see the word "font" incorrectly used to describe a typeface in digital design. There is no origin story that we have found to explain why that is, but it is helpful to know the difference.

There are several layers of classifications for typefaces, but here are the ones most relevant to digital design.

Serif

Serif typefaces have small lines and curves at the end of the letters (Figure 5-9). They tend to be the most common typefaces for reading dense amounts of text. The serif tends to subtly guide the reader from letter to letter, thereby making the reading experience less taxing on the eye.

Figure 5-9. *Examples of serif typefaces*

Sans serif

San serif fonts (Figure 5-10) do not have lines at the end of the letters. They tend to have a more contemporary look and the letters tend to hold their legibility at smaller sizes, which is one of the main reasons you will see sans serif typefaces used frequently in user interfaces.

Figure 5-10. *Examples of sans serif typefaces*

Display

Display typefaces (Figure 5-11) are typefaces that work well at a very large size; they can be serif or sans serif. Display typefaces are good for establishing a look and feel of a brand, in a headline or for a logo, but they are not appropriate for user interfaces, UI controls, forms, or body copy. Never use a display typeface for large amounts of text as display typefaces can be overwhelming when overused and lose legibility in smaller sizes.

Figure 5-11. *Examples of display typefaces*

Monospace

Monospace fonts (Figure 5-12) have letters that occupy the same space horizontally regardless of the actual width of the character. These typefaces were used in the early days of computers and you will see them also used in the interfaces of LED screens, interfaces where numbers are the primary content and screens that do not have more sophisticated ways of rendering text. Examples where you might see monospaced fonts are interfaces of noncomputer electronic devices, in-dash car displays, and appliance interfaces.

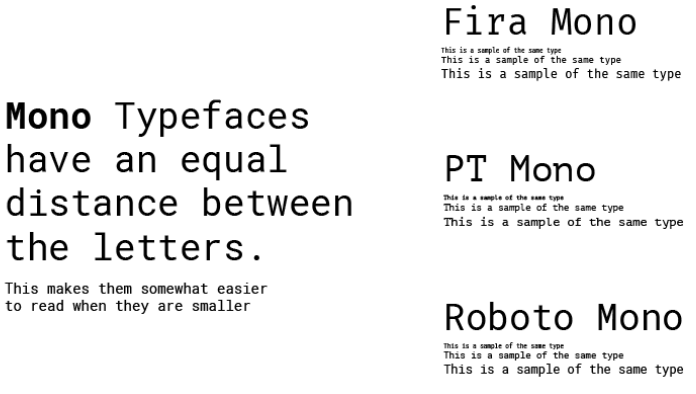


Figure 5-12. *Examples of monospaced typefaces*

Size

Type size is measured by "points," commonly abbreviated as "pt." The smaller the point size, the smaller the type. Generally, you do not want to go below 10 pts for the best on-screen legibility; 12 pts seems to be the most used standard body-text size (see Figure 5-13). For small print items such as copyright information you might have in the footer of a website, 9 pts works well. For experiences that are primarily for reading (such as a news site, or a digital reader), it's best to default to a comfortable point size like 12 pts and allow the user to increase the size as they prefer.

Figure 5-13. *Examples of font sizes*

Leading

Leading (pronounced like the metal) refers to the vertical separation between lines of text, specifically the distance between the baseline (refer back to Figure 5-8) of one line of text and the baseline of the next line. Leading should provide enough space between the lines so the eye flows from line to line and to prevent ascenders and descenders from overlapping, but not so much that the each line feels distant and separated.

Tracking and kerning

Related to leading is *tracking*. Tracking is the horizontal spacing between all letters. Improper tracking can affect readability, particularly when set too tight or too wide (Figure 5-14). But even when set properly, there can still be problems with certain characters. This is when *kerning* comes in handy. Kerning is when a designer manipulates and adjusts the space (tracking) between specific pairs of characters (usually reducing the space). This is needed, for instance, when a letter that takes up a lot of

space, such as a capital "D," and one that takes up much less space, like an "I," are side by side and have an awkward distance between them. Kerning helps make the letter pairs look more balanced and legible. Most typefaces that are optimized for digital design, such as the fonts in Google Fonts or Apple or Microsoft UI typefaces, have already been kerned to look proportional on the screen.

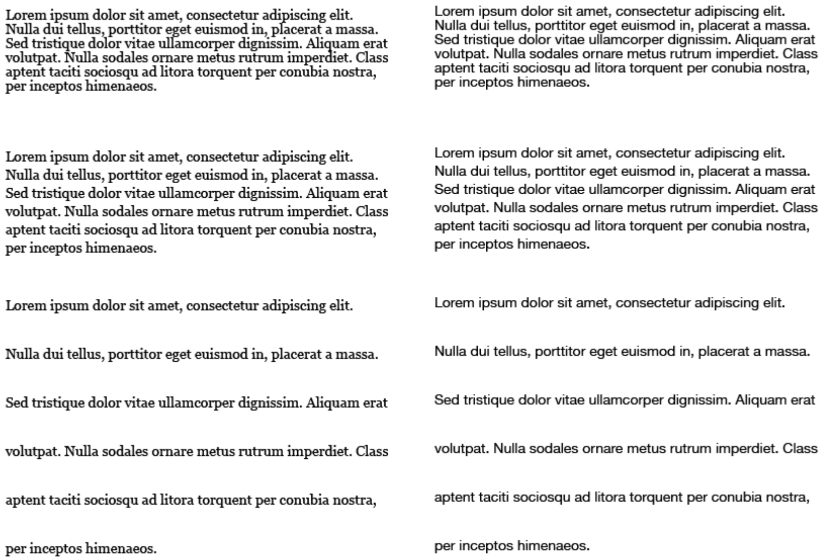


Figure 5-14. *Tracking that is tight, proportional and wide*

Font pairing

Font pairing is combining two typefaces in a design. Combining fonts is an art unto itself, but there are now some sites that help in aiding selection of fonts that look good together. There are numerous subtleties to finding combinations that work well, but there are a couple of quick rules:

- When combining fonts that are in the same typeface family, use a different weight or style (bold, italic) to differentiate the text.
- Never combine two typefaces that are similar. A way to avoid this is to pair a serif with a sans serif type (Figure 5-15).

Figure 5-15. *Examples of type pairs*

Paragraph alignment

Paragraph alignment refers to the imaginary vertical line to which the text of a paragraph aligns. In digital design, there are four options to choose from: left align, center align, right align, or justified. Justified text adjusts the spacing between words so that lines are both left- and right-aligned.

Generally speaking, you cannot go wrong left-aligning the text. As you can see in Figure 5-16, this is a highly readable way to align large amounts of text.

- The center-aligned text will draw the eye because of the whitespace around it, but use it sparingly because it is also more difficult to read.
- Right-aligned text and justified text are not usually used in UI design.

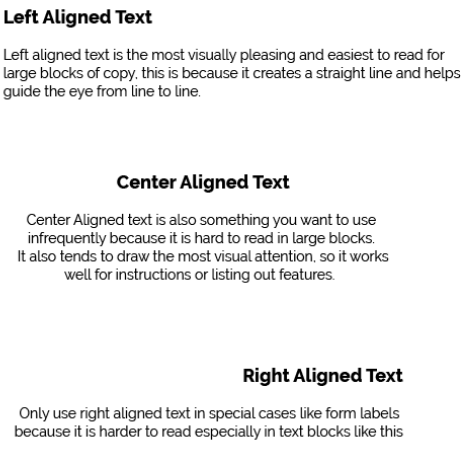


Figure 5-16. *Paragraph alignment*

Numbers

When choosing a typeface, always be sure to see what the numbers and letters look like together. In some typefaces, it is difficult to distinguish the lowercase letter "l" from the number "1" or "0" from the capital letter "O."

Readability

By choosing a typeface for a piece of text, you decide what kind of voice that text is "spoken" in. The voice might be loud or soft, friendly or formal, colloquial or authoritative, hip or old-fashioned.

As with color, readability—the cognitive part—comes first when choosing type. Small text—or what's called "body text" in print and on websites—demands careful consideration. The following characteristics for body text also apply to "label fonts" in graphical user interfaces (GUIs), used to caption text fields and other controls:

- On computer displays, sans serif fonts often work better at very small point sizes, unlike print, for which the serif fonts tend to be more readable as body text. Pixels aren't big enough to render tiny serifs well. (Some serif fonts, such as Georgia, do look passable, though.)
- Avoid italicized, cursive, or otherwise ornamental fonts; they are unreadable at small sizes.
- Highly geometric fonts tend to be difficult to read at small point sizes, as the circular letters (*e, c, d, o*, etc.) are difficult to differentiate. Futura, Universal, and some other mid-twentieth-century fonts are like this.

- All-capital letters is too difficult to read for body text, though it works fine for headlines and short texts. Capital letters tend to look similar and are not as easy for a reader to differentiate.
- Set large amounts of text in a medium-width column when possible—say, around 10—12 English words on average. Don't right justify narrower columns of text; let it be "ragged right."

Evoking a Feeling

Now for the visceral and emotional aspects. Typefaces have distinctive voices—they have different characteristics, textures, and colors on the screen. For instance, some fonts are dense and dark, whereas others are more open (Figure 5-17)—look at the thickness of strokes and the relative sizes of letter openings for clues, and use the "squint test" if you need a fresh and objective look at the font. Some fonts have narrower letters than others, and some font families have "condensed" versions to make them even narrower. The tracking might be distant or close, making the block of text look either more open or more solid.

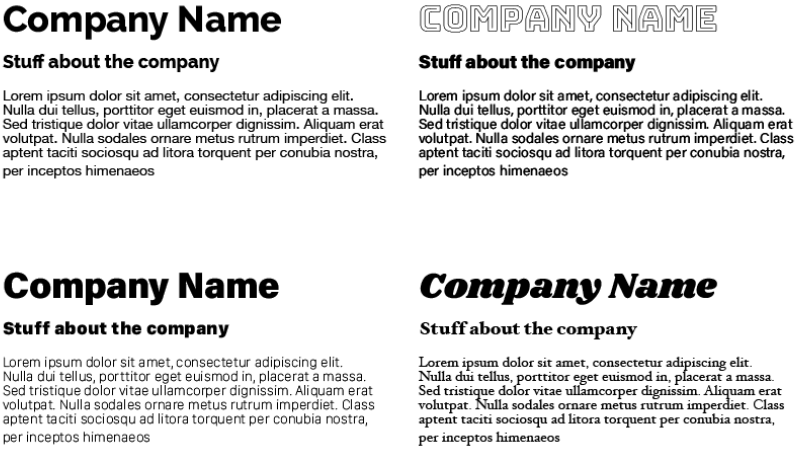


Figure 5-17. *Examples of typefaces*

Serifs and curves add another dimension to font color and texture. Serifs add a level of scale that's much smaller than the letterform itself, and that adds refinement to the font's texture—the thick sans serif fonts look blunt and strong in comparison. The curves and angles used in each letterform, including those that form the serif, combine to form an overall texture. Though it's not always easy to explain why some fonts speak with a formal voice, whereas others speak with an informal voice. Comic Sans and other playful fonts are certainly informal, but so is Georgia, when compared to

Didot or Baskerville. All-caps and capitalized words speak more formally than lowercase; italics speak informally.

Cultural aspects come into play here, too. Old-fashioned fonts, usually with serifs, tend to look—wait for it—old-fashioned, although anything set in Futura (a sans serif font) still looks like it came from a 1963 science textbook. Verdana has been used so much on the web that it's now standard for that medium. And Chicago will always be the original Mac font, no matter what context it's used in.

Spaciousness and crowding

Some designs use plenty of whitespace; others crowd the screen elements together. Spaciousness on the screen gives an impression of airiness, openness, quiet, calmness, freedom, or stateliness and dignity, depending on other design factors.

Crowded designs can evoke urgency or tension under some circumstances. Why? Because text and other graphic elements need to "breathe"—when they're colliding against one another or against the edges or borders of the screen, they cause visual tension, as demonstrated in Figure 5-18. Our eyes want to see margins around things. We become slightly disturbed by designs that shove the headlines directly against the text. Likewise, the compact layout somehow contributes to the busy, industrial feel of the screen, though it doesn't have collisions.

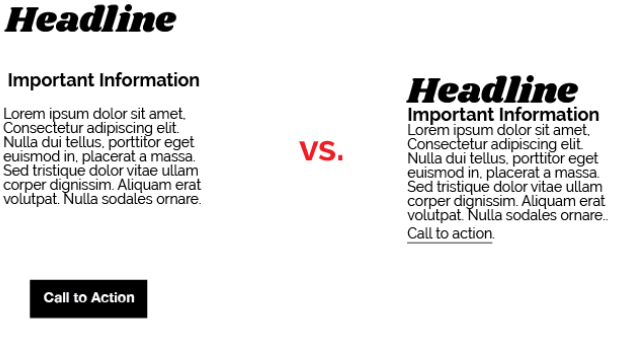


Figure 5-18. *A spacious and crowded visual design*

However, not all crowded designs evoke that kind of tension. Some connote friendliness and comfort. If you give the text and other elements just enough space and reduce the interline spacing (leading) to the smallest amount that is comfortably readable, you might achieve a friendlier and less rarified look.

Angles and curves

A screen composed of straight up-and-down lines and right angles generally looks calmer and more still than a screen containing diagonal lines and nonrectangular shapes. Likewise, a screen with many different angles has more apparent motion than a screen with a single repeated angle on it.

Curves can also add motion and liveliness, but not always. A design made with a lot of circles and circular arcs can be calming and restful. But a curve swooping through a screen sets the whole design in motion, and a few carefully chosen curves in an otherwise rectangular design add sophistication and interest.

In the example in Figure 5-19, Stripe uses angles to create a dynamic and legible design to guide the eye around the design and thus to the important information the designer wants visitors to read.

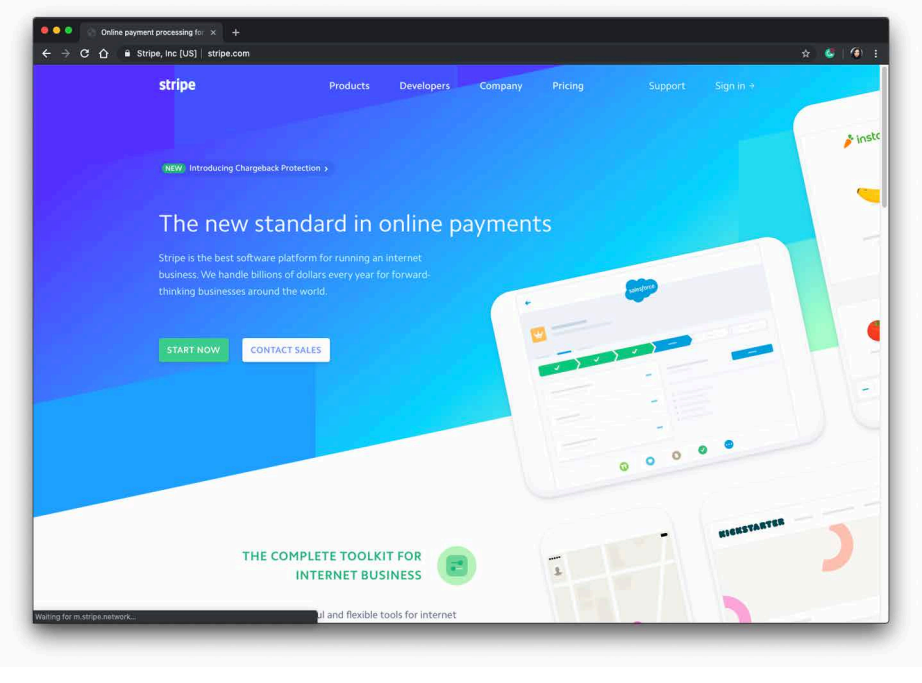


Figure 5-19. *Stripe online payments website*

Wherever two curves intersect, notice what the geometrical tangents to those curves are doing. Are the tangents at right angles? That results in a calmer, more still composition; if they cross at a more acute angle, the design has more tension and apparent motion. (Again, these aren't hard-and-fast rules, but they're generally true.)

When using angles, curves, and nonrectangular shapes, think about where the focal points are: at sharp angles, where lines cross, and where multiple lines converge, for instance. Use these focal points to draw the viewer's eye where you want it to go.

Texture and rhythm

Texture can add richness to visual design. Text forms its own texture, and you can control the look of that texture by choosing good typefaces. For many screens and interfaces, fonts are the most important texture element.

You also can use texture to surround strong visual elements and set them off. Textures add visual interest and depending on what they look like, they can add warmth, richness, excitement, or tension.

The most effective textures in the interface design are subtle, not vivid checkerboards of eye-hurting colors. They use gentle color gradations and very tiny details. When spread over large areas, their impact is greater than you might think. An exception might be in the example in Figure 5-20. Szimpla Kert is a store and cafe in Budapest and the website uses bright colors and visual texture to evoke a celebratory, dynamic look and feel.

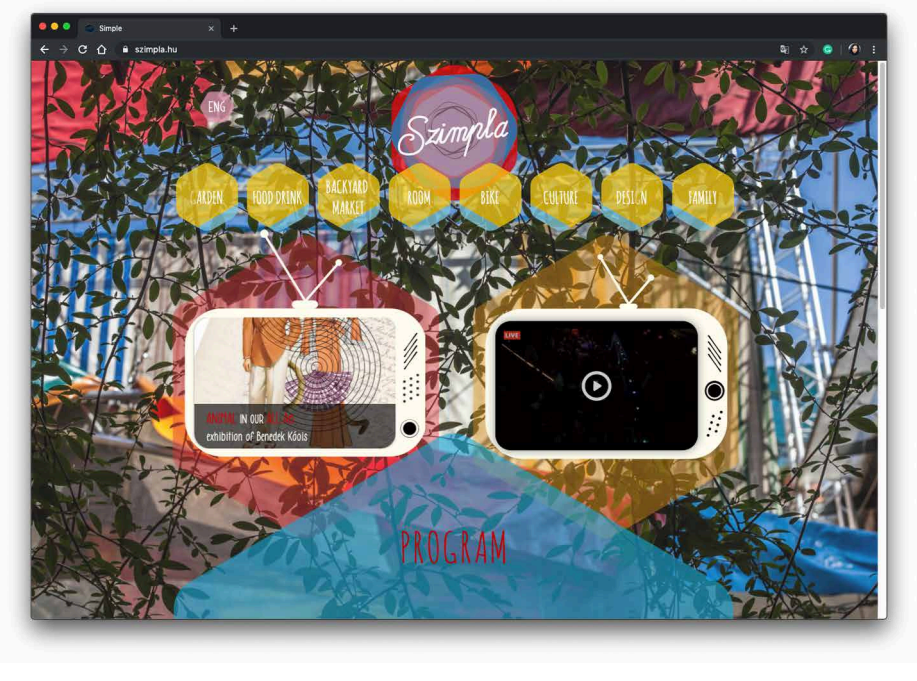


Figure 5-20. *Szimpla Kert website*

Be careful when using textures behind words on a computer screen—it rarely works. All but the subtlest textures interfere with the readability of small text. You can put them behind large text, but watch the way the edges of the letterforms interact with the different colors in the texture; this can visually distort the letters.

Images

Photography

Photography can set the mood of a design. On the web and mobile digital products, photography is one of the most powerful tools to establish how a brand is expressed. A well-placed photo can tell a story in a single glance much more efficiently than words could. Photographs are extraordinary tools for evoking emotional responses.

In most desktop applications and mobile applications, content and ease of use are more important than style. You should use purely decorative images sparingly and with great care on functional GUIs because they tend to be distracting.

Here are a few tips to keep in mind:

- If you use a person's face, be sure to pay attention to where the gaze of the person's eye is directed. Humans tend to look where other humans look. Even when those humans are images on a screen.
- Try to avoid clichés when you can. How many web screens have you seen showing the same happy, smiling faces? Kids flying kites? Competent-looking business people in crisp suits? How about roads winding through beautiful mountain scenery? Sunsets or beaches? Rolling grassy hills under sunny blue skies? Try not to rely on these visual conventions alone to set the tone for your brand.
- Stock art (photographs that you can purchase, sometimes royalty-free) are okay, but for the biggest impact, nothing beats custom photography or designs that have been created by trained art directors and visual designers.

Icons

Icons (Figure 5-21) are graphical representations that serve in place of text to express an idea or denote functionality.

Figure 5-21. *Icons*

Creating icons, like typography, photography, or illustration, is a skill on its own. Icons express complex ideas at a glance and give the user an idea of what actions to expect when they click or tap an item.

- Look for UI conventions you see on the web or from other icons. Applying common conventions found in other designs make it less likely your user will need to relearn what the icon means.
- Make sure your icons all share the same visual style: the same weight, and either filled in or outlined, for example.
- Don't rely on icons alone. Use them sparingly and when you can, use text labels as well for maximum user comprehension.

Icon references and resources

- *https://developer.apple.com/design/human-interface-guidelines/ios/icons-and-images/custom-icons*
- *https://thenounproject.com*
- *https://material.io/tools/icons*

Cultural references

A design might remind you of something cultural—a brand, movies, art style, historical era, a literary genre, or inside joke. A familiar reference can evoke memories or emotions strong enough to trump all these other design factors, though the best designs make cultural references work in concert with everything else.

Obviously, if you make overt cultural references, consider your audience. A 10-year-old will not get a 1970s pop art reference. Chances are good that a young adult in India won't either. But if your audience is sufficiently well defined for you to know that a cultural reference will be familiar to them, it can be a good "hook" to engage a viewer emotionally with your design.

Cultural references are rarely used in functional application designs, but you can see them in applications like QuickBooks, in which some screens are designed to look like checks and bills. They actually move beyond a stylistic treatment and become an interaction metaphor, but the metaphor still is entirely cultural—someone who has never seen a checkbook wouldn't respond in the same way as someone who has.

Repeated visual motifs

Good design has unity: it hangs together as one entity, with each element supporting the others structurally and viscerally. That's a challenging goal to achieve. I can't give you hard-and-fast rules on how to do it; it takes skill and practice.

But one thing that contributes greatly toward visual unity is the repetition of visual elements or motifs. We've already talked about angles and curves; you can use diagonal lines of the same angle, or lines with similar curvature, as repeated elements in a design.

Also, consider typography. Use only one main body text font, though other fonts can work very effectively in small areas such as sidebars or navigation links. (Their contrast to the main font makes them stand out.) If you have several headlines or titled sections, use the same headline font for them. You also can pull smaller graphic elements—line width and color, for instance—out of your fonts into the rest of the design.

Rhythms like these can be powerful design tools. Use them with care, and apply them to groups of comparable things—users will assume that similarity in form means similarity in function.

Visual Design for Enterprise Applications

Those of you who work on consumer-facing digital products might already be familiar with everything discussed so far. People expect websites and mobile applications to have strong graphic styling, and you rarely will find them looking completely plain and neutral.

But what if you work on desktop or enterprise applications? If you try to apply these principles to the controls' look-and-feel—how the controls are drawn—you might not have many choices. Native Windows or Mac applications generally use the standard platform look-and-feel, unless you're willing to work hard to develop a custom one. Enterprise applications should be optimized for workflow and be easy to look at for large stretches of the workday.

Given the situation, you can be forgiven for just using the platform look-and-feel standards and concentrating your graphic design attentions elsewhere.

Even if you do use a neutral look-and-feel for your actual widgets, there still are ways to be creative:

Backgrounds
: Unobtrusive images, gradient fills, and subtle textures or repeated patterns in large background areas can brighten up an interface to an amazing extent. Use them in dialog or screen backgrounds; tree, table, or list backgrounds; or box backgrounds (in conjunction with a box border).

Colors and fonts
: You often can control overall color schemes and fonts in a native-looking UI, too. For instance, you might draw headlines in an unusual font at several point sizes larger than standard dialog text, and maybe even on a strip of contrasting background color. Consider using these if you design a screen layout using the *Titled Sections* pattern (Chapter 7).

Borders
: Borders offer another possibility for creative styling. Again, if you use *Titled Sections* or any other kind of physical grouping, you might be able to change how box borders are drawn.

Images
: In some UI toolkits, certain controls let you replace their standard look-and-feel with custom images on a per-item basis. Buttons often allow this, for instance, so your buttons, including their borders, can look like anything you want. Tables, trees, and lists sometimes permit you to define how their items are drawn You also can place static images on UI layouts, giving you the ability to put images of any dimension just about anywhere.

Accessibility

The biggest concern is accessibility. Operating systems such as Windows let users change desktop color/font themes, and that's not just for fun—visually impaired users use desktop themes with high-contrast color schemes and giant fonts just so they can see what they're doing. Make sure your design works with those high-contrast themes. It's the right thing to do.[1]

Another danger is fatiguing your users. If you design an application meant to be used at full size or for a long time, tone down the saturated colors, huge text, high contrast, and eye-catching textures—make the design quiet, not loud. More important, if your application is meant to be used in high-stress situations, such as a control panel for heavy machinery, strip out anything superfluous that might distract users from the task. Here, cognitive concerns are far more important than aesthetic .

Ranges of Visual Styles

Visual design styles change fairly quickly. Most often, new releases of operating systems drive a change in the visual UI styles that are adopted after their release. In this way, Apple, Microsoft, and now Google are setting the trends for the visual styles in the applications that will be used on their platforms. We take a deep-dive into a few of the more widely used styles across these platforms and across their touchpoints: web, desktop software, and mobile applications.

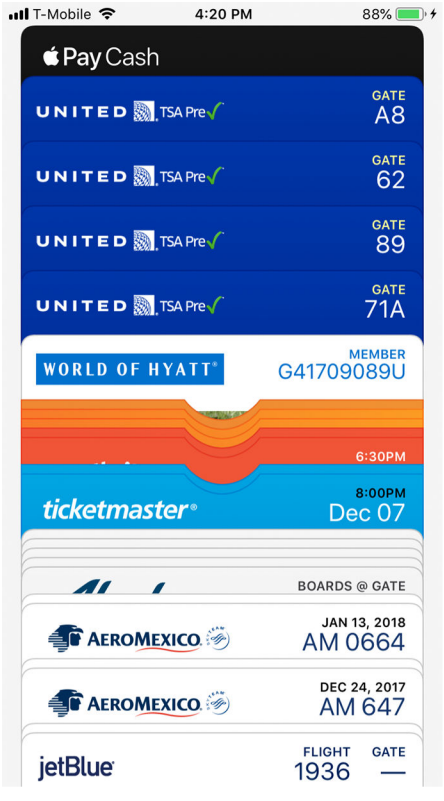
Skeuomorphic

A skeuomorphic design refers to a style of UI that mimics the characteristics of objects found in real life. Skeuomorphism is often used during a period of transition when a new type of interaction is used and you want the user to understand how something works by using an idea or concept with which they are already familiar.

When the iPad was first released, skeuomorphic visual designs were on almost every application. This was used as a visual affordance to inform the user how to interact with a touch interface.

In the Apple Wallet example shown in Figure 5-22 (left), the list of items stored in the list of data has been given a visual design that mimics the look of cards or tickets. This translation of real-life objects to the digital help the user to find what they are looking for and also helps them manage and arrange the content.

1 And, depending on who buys your software, it might also be the legal thing to do. The US government, for example, requires that all software used by federal agents be accessible to people with disabilities. For more information, see *http://www.section508.gov*. The Americans with Disabilities Act also has design standards.



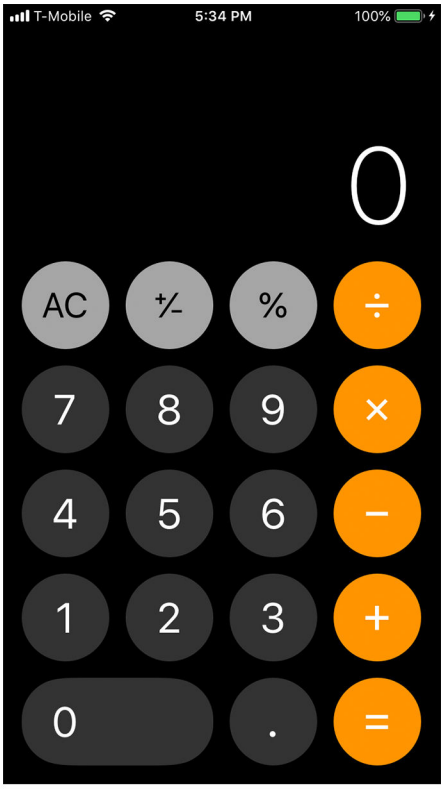


Figure 5-22. *Apple iOS Wallet and Apple iOS calculator*

Apple uses the skeuomorph successfully again in its calculator app Figure 5-22 (right). The rounded numbers are an optimal "touch target" size (we will discuss this more in the context of mobile design in Chapter 6) and the functionality mirrors what you would expect from a physical calculator. In this way, the designers of iOS have transcended the iPhone into a device that morphs and changes to the intended functionality of the application.

You can also use a skeuomorph within another design style to increase usability. In the Square Invoice set-up process (Figure 5-23), the designer chose to use the visual language of the real-life physical check so that a user would know where to locate the routing and checking account numbers needed to set up their account.

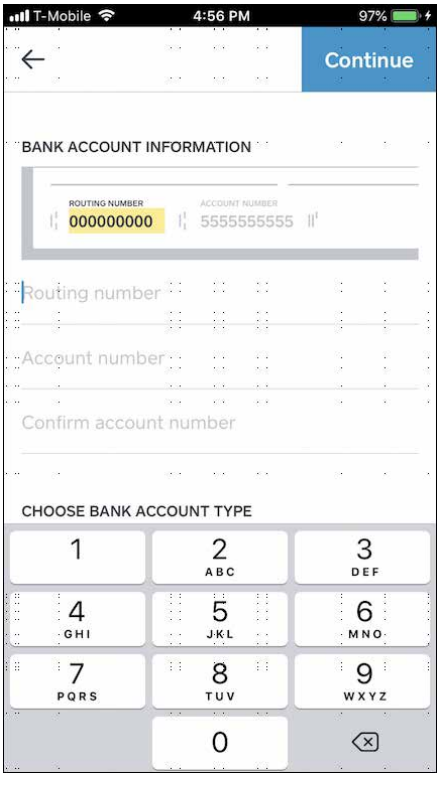


Figure 5-23. *Square Invoice bank account entry and Eventbrite interface*

Illustrated

Interface design doesn't need to be cold and sterile. If it makes sense for the brand, using illustration is a good way to set a fun and approachable tone to an application or website. Using an illustrated style also sets you free from what is possible in the real world and allows the designer to express complex concepts limited to only what's possible in the imagination.

Eventbrite (Figure 5-24) is an event-listing and ticket-purchase service that uses an illustrated style all around its mobile application interface providing a warm and inviting visual look and feel.

Figure 5-24. *Eventbrite interface*

Florence (Figure 5-25) is a site for self-employed nurses to find shifts. This could have been a boring, staid site but the illustrations and visual design set the tone for a fun and friendly brand.

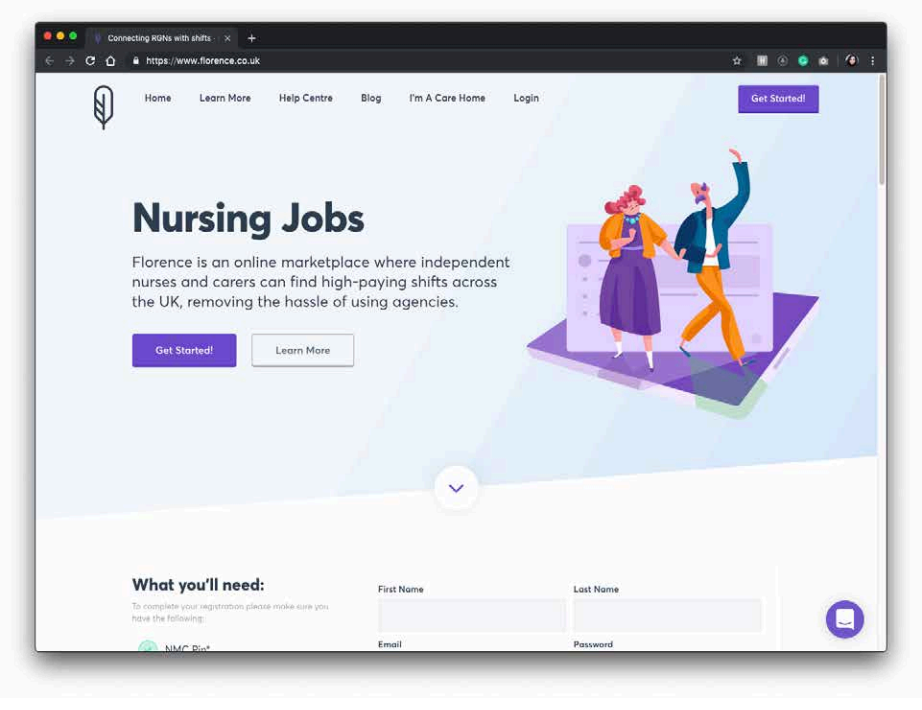


Figure 5-25. *Florence.co.uk*

Happy Cow (Figure 5-26) is a mobile application that helps vegetarians and vegans find restaurants and food anywhere in the world. The site uses the Happy Cow mascot and a lighthearted icon language throughout its UI.

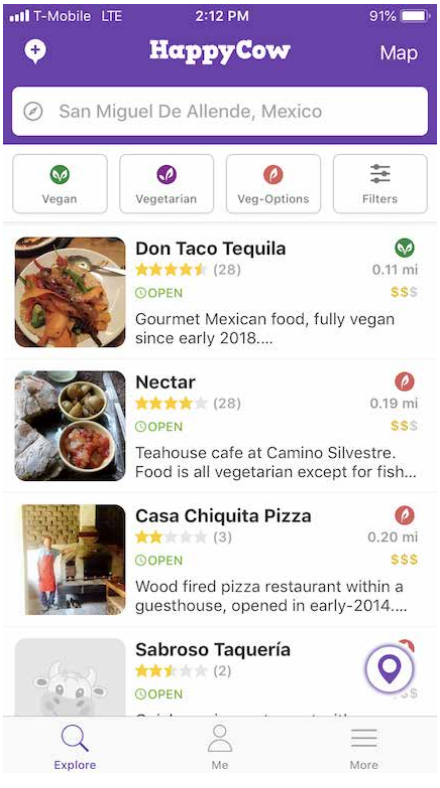


Figure 5-26. *Happy Cow*

When your product is your application or website, which is the case with the examples shown in this section, using custom illustrations is an effective way to establish a memorable brand.

Flat Design

Characterized by solid background colors, simple understandable icons and sans serif typography, flat design is one of the most widely used visual design styles you will see on the web and in mobile apps. Think of the style of signage used in airports and in transportation and you can see why this flat and minimal style has such universal appeal. It is culturally agnostic, easy to localize, and can scale to different viewports (screen sizes) easily.

Flat design is considered a truly digital style because the visual language of the UI (with the exception of the icons) is no longer pretending to be analogous to something in real life. Instead, the UI is meant to blend into the background and allow the user to focus on the content.

The examples shown in Figure 5-27 demonstrate flat design in a mobile application. What are the commonalities you see? Do you see solid colors, a single typeface used at different sizes, use of two-dimensional icons? When you pay attention to this visual style you will notice that it is used everywhere.

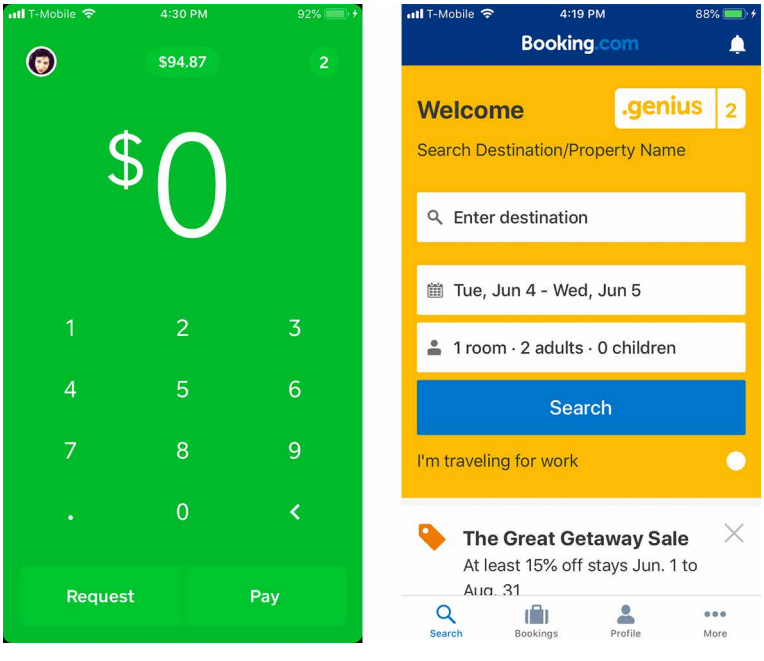


Figure 5-27. *Cash and Booking.com apps*

AirVisual (Figure 5-28) is a mobile application that shows the air quality in cities throughout the world. It employs a combination of flat design and custom illustrations for a clear and usable visual presentation.

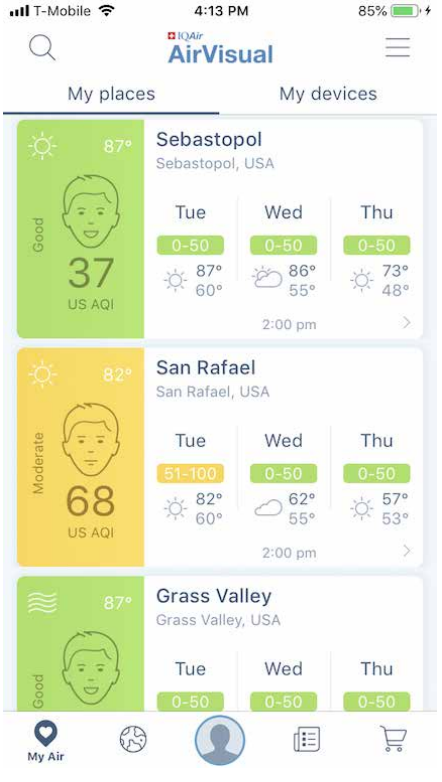


Figure 5-28. *AirVisual app*

For more details on flat design

- *https://www.microsoft.com/design/fluent*
- *https://material.io/design*

Minimalistic

Minimal designs reduce the screen elements to the barest minimum. You will often see minimal designs used in task-based apps when the things that a user can do are straightforward or the interface is primarily used for viewing data rather than inputting or manipulating data.

Clear Todos (Figure 5-29) is an extreme example of showing only shades of color and text and very limited Visual UI cues as to how the interface works.

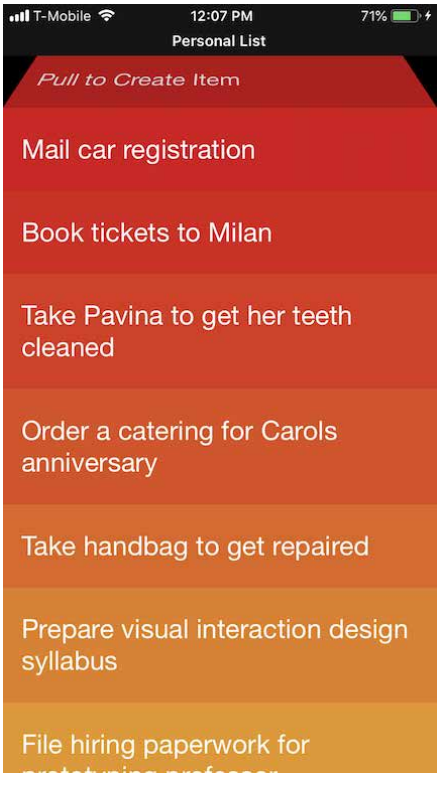


Figure 5-29. *Clear Todos app*

The Calm app (Figure 5-30, left) shows only what is absolutely necessary for the user to see on any given screen. In the example here, one button is used to control the functionality and a subtle but helpful animation instructs the user what to do.

Apple Health (Figure 5-30, right) uses bold colorful information graphics as the core of its visual UI.

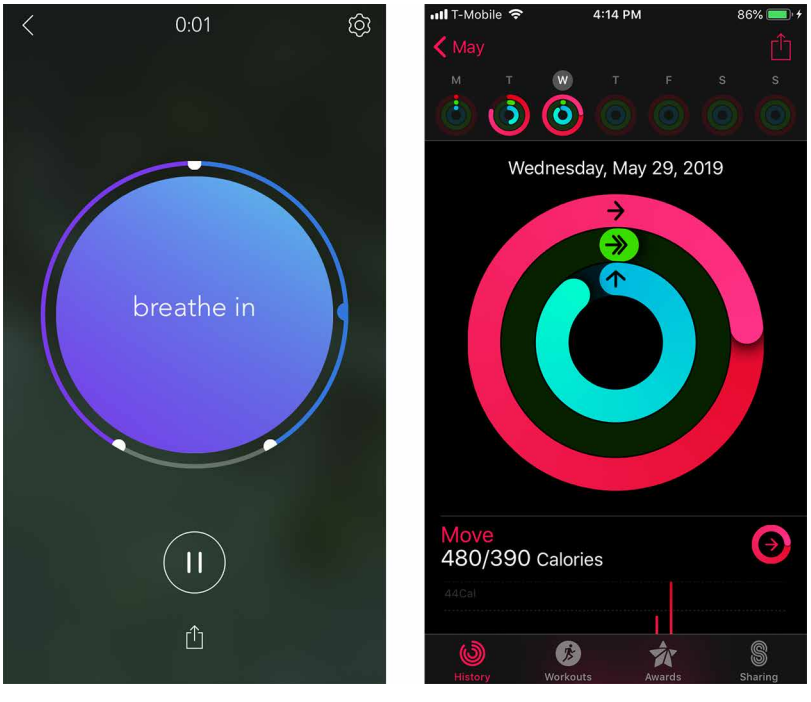


Figure 5-30. *Calm and Apple Health apps*

Glitché (Figure 5-31) is an application to create "glitch" style photos and videos. Its minimal UI allows the user to focus on the task and doesn't take valuable screen estate away from the primary task of photo and video editing.

Brian Eno's Bloom music application (Figure 5-31) uses only audio cues to inform the user how to interact with it. The experience relies on subtle, muted color changes and ambient sound that encourages exploration and creation.

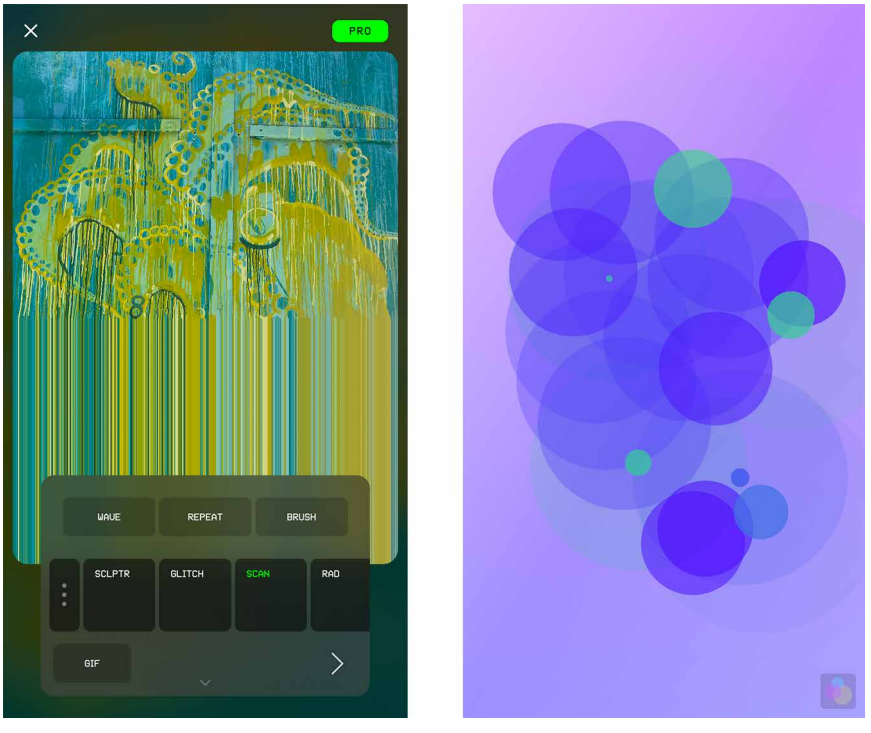


Figure 5-31. *Glitché and Bloom*

Adaptive/Parametric

On the extreme end of minimalism, adaptive or parametric refers to designs where the form is not static or defined but rather generated algorithmically, in relationship to the objects (static or dynamic) that come in proximity. You will see this type of interface paradigm often used in video or photo applications, and more and more you will see it in other types of interfaces. A written definition does not do this type of design justice. Imagine an interface that is mostly invisible until it comes into contact with something that enables an interaction to happen—then the UI reveals itself and wraps around the object with which a user can interact. When creating visual designs for this kind of interface, creating a high-contrast and fluid UI visual style is key.

Apple Measure (Figure 5-32) is a measuring tool that comes with iOS. To measure something, the user points their phone or tablet at the object and then the measuring tools and functions assemble themselves onto the interface.

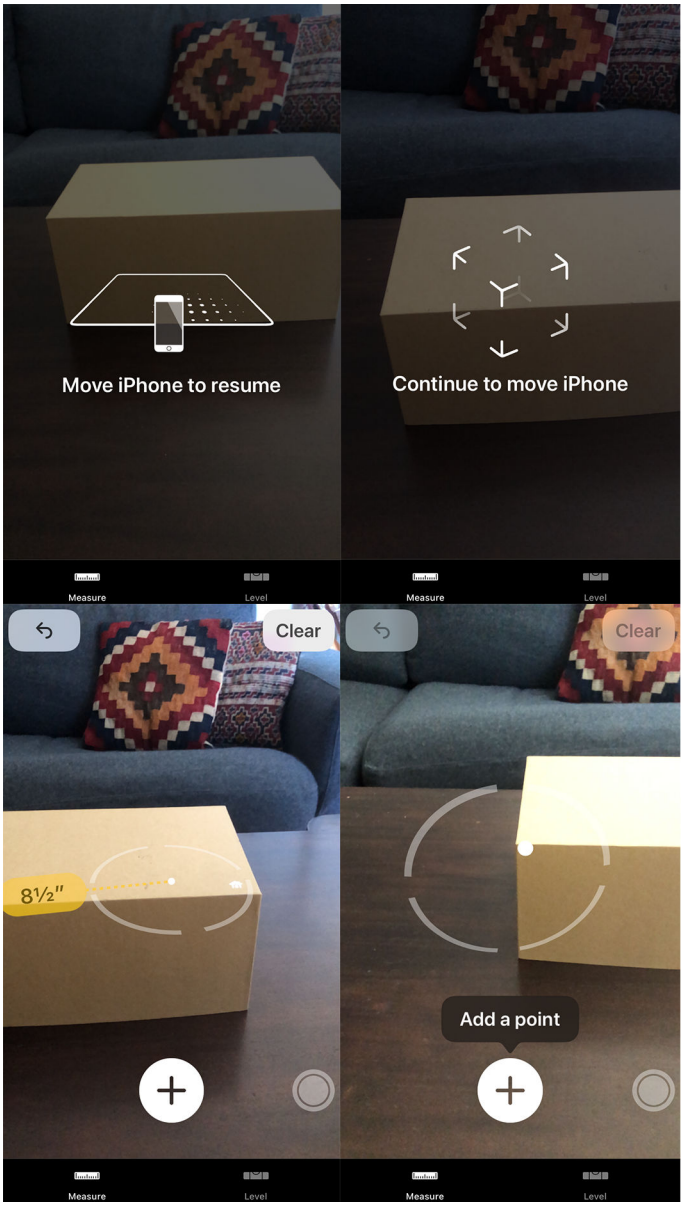


Figure 5-32. *Apple iOS Measure*

Simple (Figure 5-33) is a mobile banking application. To deposit a check, a user is prompted to take a photo of the check, and the UI will emerge to indicate when the photo is in the correct place and when a photo can be captured.

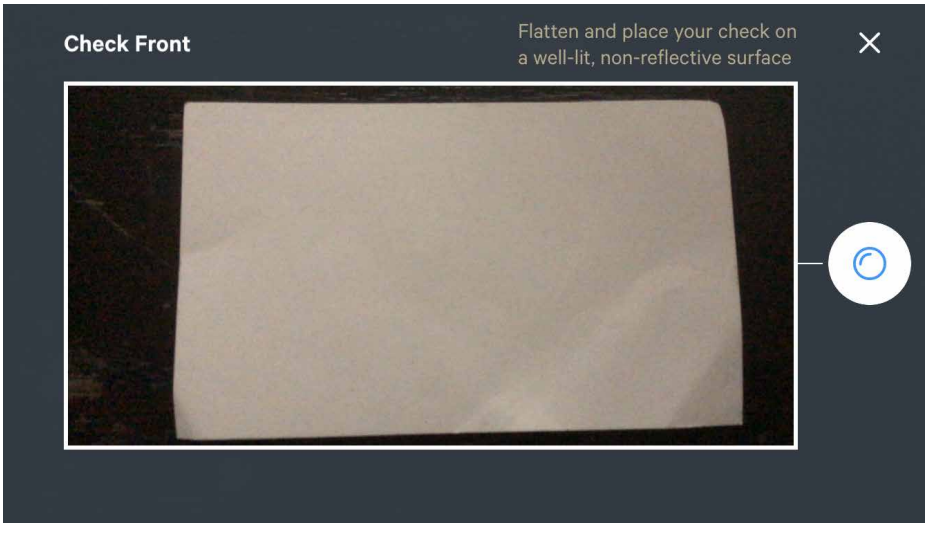


Figure 5-33. *Simple check capture*

Conclusion

Great visual design is a skill that takes time to perfect. Graphic design is a discipline all to itself, and it would take far more than a chapter to get into the nuances of the grid, color theory, typography, and gestalt to truly understand how to create a pixel-perfect visual design. Fortunately, with new website creation and editing tools, much of the finesse that it takes to get a perfectly kerned and detailed designed look are available to everyone with a credit card and content to share.

Web creation platforms like Squarespace, Wix, and WordPress have premade templates that have been created by visual designers. These tools take the guesswork out of designing layouts and worrying over spacing and margins and legibility.

We have just scratched the surface of what there is to know about visual design in the context of the UI. The big takeaways are that aesthetics matter, and getting those details right makes a huge difference in how your product or service will be perceived by your audience.

Chapter 6
Mobile Interfaces

Look around on any city street, and you will see people with their heads hunched down looking at their mobile devices. The world is full of iPhones, Android phones, and other smartphones and tablet computers. There are entire countries where people reach the internet primarily through their phones. By 2025, five billion people are projected to be mobile internet users worldwide.[1] It is likely the users of your product will primarily interact with your product via a mobile device. Designing for mobile is not just good design practice or a practical business concern, it's common sense.

It stands to reason that mobile devices have become an indispensable part of daily life. A cell phone is not simply a phone or means to connect to the internet; it has become a primary gateway to communication, commerce, entertainment, transportation, and navigating one's way. Smartphones and tablets in particular also have the benefit of direct manipulation—being able to touch the object you want to select or edit—which makes mobile interfaces easy and intuitive to learn.

Simply making a compact version of your website is no longer the way design is done, and a *mobile-first* (designing for the mobile experience prior to designing for a fuller featured web experience) or *responsive-design* (designing for a web experience that can gracefully scale to fit different screen sizes) approach is adopted by companies who want to ensure their digital products can scale for the future.

The world of mobile experience is vast and spans from mobile web to native mobile applications (apps). Some products will want to supply all of their functionality via the mobile site, but all of it would be tailored to the small screen and other mobile constraints. Many people view the internet exclusively through their mobile device,

1 *The Mobile Economy Report*, GSM Association, 2018

and they'll want all of your site's features. You might choose to do two separate and parallel designs, one for mobile and one for the desktop.

If you create tools and applications for large screens, instead of websites, this chapter might not apply to you at all. You and your organization might want to evaluate whether your tools (or some subset thereof) could be re-created as apps on mobile devices and still be useful. Know your users—understand their needs, tasks, and contexts of use.

Each have their pros and cons and depending on the richness of the experience you need to deliver you will likely be designing for a desktop (web), mobile-web, *and* mobile-native application. Creating mobile apps is a nontrivial investment, but it might be worth it.

Some of these users will see your sites through browsers that are small, slow, quirky, and difficult to interact with. They will use your sites in environmental conditions—and mindsets—that are entirely different from what they would experience if they were sitting quietly at a comfortable desk, in front of a large screen.

In this chapter, we won't go into the technical details of platform detection and how to present the correct design for the user's situation (e.g., different CSS stylesheets)—but the knowledge is out there and fairly easy to find. A relatively small investment of knowledge, design work, and time can go a long way toward improving the mobile experience of the sites you design.

The Challenges and Opportunities of Mobile Design

When you design for a mobile platform, you face challenges that you don't encounter when your user can be presumed to be viewing a large screen and using a keyboard:

Tiny Screen Sizes

Mobile devices just don't offer much space to present information or choices. Sadly, you don't have the luxury of sidebars, long header menus, big images that don't do anything, or long lists of links. You need to strip your design down to its essence—take away all the extra stuff you can. Leave the most important functions on the front screen and either discard the rest or bury them deeper in the site.

Variable Screen Widths

It's difficult to make a design that works well on screens that are 360 pixels wide and 640 pixels wide. Scrolling down a mobile screen isn't terribly onerous (which is why width gets special mention, not height), but the design needs to use the available screen width intelligently. Some sites end up creating different versions—with

different logo graphics, different navigation options, and so on—for the smallest keypad devices, and another for the iPhone-size class of touch devices.

Touch Screens

Most mobile web and app access comes from devices with touch screens. Keypad devices obviously should be served, too, because they constitute the majority of existing mobile devices, but you might want to bias the design toward the touch-screen experience. Links on keypad devices can be navigated with keys fairly easily, as long as you follow good design principles (restricted content, linearized layout, etc.).

It's difficult to touch small targets accurately with fingers. Make your links and buttons large enough to easily touch; at a minimum, make important targets at least 48 × 48 dp (9 mm) for Android devices[2] and 44pt × 44pt for Apple iOS[3] on each side, and put space between them. This reduces the available space for other content, of course.

Difficulty of Typing Text

No one likes typing text on a touch screen or keypad. You should design interaction paths through your site or tool in such a way that typing is unnecessary or very limited. Use "Auto Completion" (functionality that predicts the next letter a user is going to type to save the user keystrokes) in text fields whenever possible, for instance, and prefill form fields whenever you can do so reliably. Remember that numbers are much easier than text in some contexts, however.

Challenging Physical Environments

People use their phones and other devices in all kinds of places: outside in the bright sun, in dark theaters, in conference rooms, cars, buses, trains, planes, stores, bathrooms, and in bed. Think about the ambient light differences, to begin with—tasteful gray text on a gray background might not work so well in direct sun. Think also about ambient noise differences: assume that some users won't hear the device at all, and that others might find sudden noises jarring and inappropriate.

Finally, think about motion. Tiny text is difficult to read when the device (or the user) is moving around. And a tiny hit target on a touch-screen device will be challenging to use under the best of circumstances, but it can be nearly impossible on a rocking and jolting bus. Again, design for "fat fingers," and design so that mistakes are easily corrected.

2 Material.IO Accessibility Guidelines

3 Apple Developer Human Interface Guidelines

Location Awareness

Mobiles are on the go with their users. This also means that these devices are able to locate exactly where they are being used. This makes it possible to design for scenarios in which information specific to the location can be served up and paired with the local data. Systems can even infer what situations the user might be in to better anticipate their needs.

Social Influences and Limited Attention

Most of the time, mobile users won't spend lots of time and attention on your site or app. They'll be looking at your design while doing other things—walking, riding in a vehicle, talking with other people, sitting in a meeting, or—but most preferably not—driving. Occasionally a mobile user will focus their full attention on the device, such as when playing a game, but they won't do it as often as someone sitting at a keyboard will. Therefore, design for distracted users: make the task sequences easy, quick, and reentrant. And make everything self-explanatory.

Another assumption you can make is that lots of mobile users will be engaging in conversations or other social situations. They might pass around the device to show people something on the screen. They might have people looking over their shoulder. They might need to suddenly turn off the sound if it's not socially acceptable to have a noisy device—or they may turn it up to let others hear something. Does your design behave well in these situations? Can it support graceful social interaction?

How to Approach a Mobile Design

If you're simply trying to take a site's usual content and cram it into a 360 × 640 window, stop. Take a big step back and look at the whole picture.

1. What Do Users in a Mobile Context Actually Need?

A person who is out and about with a mobile device might want to use your site (or app) only in particular ways; they won't have the same range of needs that a user of the full site will have. Design for use contexts such as these:

- "I need to know this fact right now, quickly."
- "I have a few minutes to spare, so entertain me."
- "Connect me socially."
- "If there's something I need to know right now, tell me."
- "What's relevant to the place I'm in right now?"

2. Strip the Site or App Down to Its Essence

Don't be afraid to take away all that other stuff—the extra content, eye-catching features, sidebars, pull quotes, ads, images, site maps, social links, and so on. Focus tightly on the few tasks that mobile users will need from your site, use minimal branding, and chuck the rest.

In fact, make sure that even on the home screen (for a website) or the first working screen of an app, relevant content appears high on the screen. That means getting rid of the "layer cake effect" of logos, ads, tabs, and headers that stack up on the screen. See Figure 6-1 for an example of a poor mobile design; the only piece of content that a user really cares about is the score at the bottom of the screen. (If the user were to rotate the phone sideways, the score wouldn't even be visible "above the fold.")

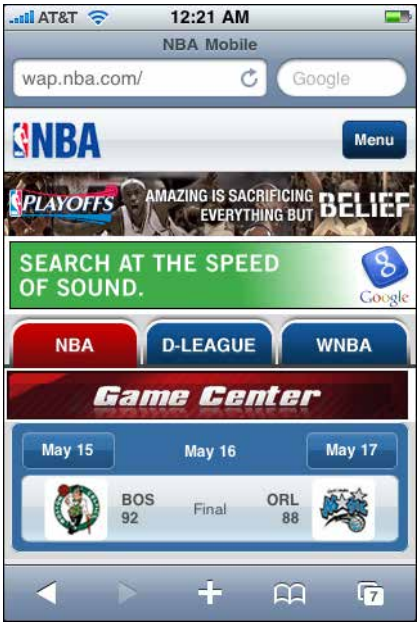


Figure 6-1. *An example of poor mobile design in which the only information the user cares about is at the bottom*

Having reduced the site to its minimal form, you should then make sure that a user who really needs the full nonmobile site can get it. Put a link to the full site in an obvious place. Remember that many of the world's population can access the web only through their phones, so you can't count on them just going to the full site on their large screen—they might not have one, let alone that device being connected to the internet.

Alternatively, you might create the two "separate and parallel" designs mentioned earlier, in which all of the site's functions and information are presented in the mobile site (meaning the user never needs to go to the full nonmobile site). You might still need to strip down the home screen or main screen. Instead of having a flat and broad navigational hierarchy in which the home screen has a zillion links directly to other screens, you might need to reorganize the site so that the hierarchy is somewhat narrower and deeper. This lets you put fewer options on the home screen, which means less clutter on a small screen. (Of course, you'll need to balance that against the time it takes for a user to jump from screen to screen.)

3. If You Can, Use the Device's Hardware

Mobile devices offer wonderful features that you don't get on the desktop. Location, camera, voice integration, gestural input, haptic feedback such as bumps and vibrations, and other features can be available to you. Some devices multitask so that your app can be running in the background while the user is doing other things; can you use that?

4. Linearize Your Content

This goes back to the width problem. Many devices simply don't give you enough pixels in the width dimension to do any interesting side-by-side layouts. Instead of forcing the issue, just accept that one way or another, your content will end up being laid out vertically. Order the mobile site's content so that it "reads well" when laid out this way. See the *Vertical Stack* pattern later in this chapter.

5. Optimize the Most Common Interaction Sequences

After you've decided which tasks your typical mobile users will want to perform, and you've narrowed down the site to only the most relevant content, try to make those tasks as easy as possible by following these heuristics:

- *Eliminate typing*, or reduce it to as few characters as possible.
- Use as *few screen loads* as possible, and don't inflate screens with unnecessary bytes. Download times can be very slow; most parts of the world are still outside the reach of high-bandwidth wireless internet facilities.
- *Reduce scrolling* and sideways dragging, except where it eliminates screen loads and typing. In other words, prefer one long vertical screen to many small screens if you need to present a lot of content.
- Reduce the *number of taps* it takes for a user to reach the desired information or accomplish a task. Tapping large hit targets—or using hardware buttons—is better than typing by a long shot, but try to reduce them anyway.

Some Worthy Examples

Here are some mobile versions of home screens that manage to meet most of the design constraints listed in the preceding section while retaining the branding and personality of each site. In some of these examples, I show the mobile web (internet) version and the mobile app version for comparison.

Lugg (Figure 6-2) is an on-demand moving service. Its mobile web and mobile app both follow good mobile design practices. Lugg focuses on the main task point of its service and have big touch targets for text inputs and buttons and a clear call to action. All the most important information is easily seen at a glance.

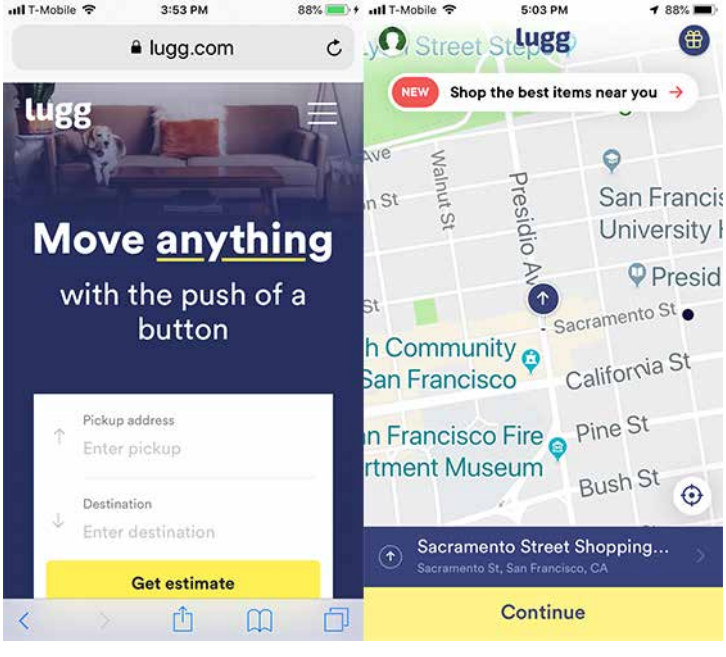


Figure 6-2. *Lugg's Mobile Web and Native iOS App*

Booking.com is a travel booking site that books hotels, flights, and rental cars, so it stands to reason that both its mobile web and native-mobile app home screens are 100% focused on getting the visitor to engage in a destination search as it s first and default experience (Figure 6-3). Booking.com also takes advantage of the geolocation data available on smartphones and incorporates this into the experience.

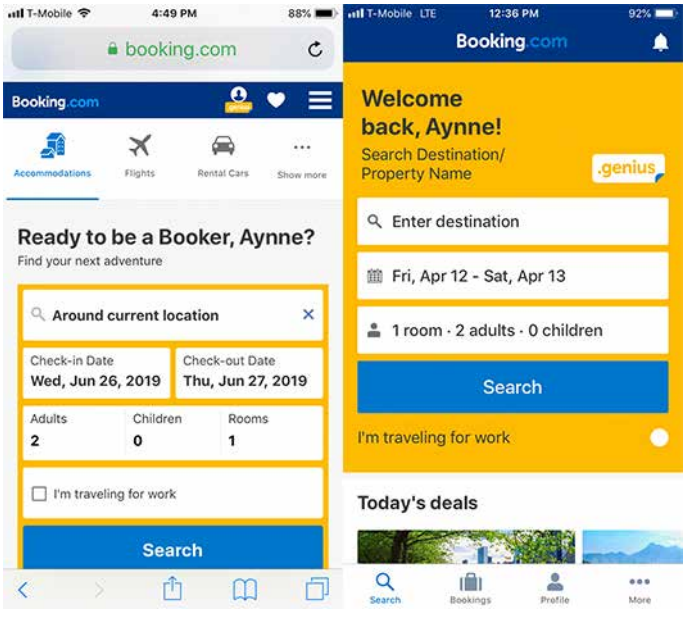


Figure 6-3. *Booking.com mobile website and native-iOS app*

The *New York Times* crossword puzzle (Figure 6-4, left) gets high marks for being optimized for the limitations and opportunities afforded by its mobile form factor. When the user taps the first space to enter a letter, the entire row highlights and the clue appears in the blue highlighted area above the keypad. This is an excellent visual affordance that saves keystrokes and simplifies what otherwise would have been a very complex interaction.

NPR's One App (Figure 6-4, right) shows an excellent example of streamlining functionality with its pared-down mobile user interface. It uses location-based data to show the closest station and displays single, large "play" button.

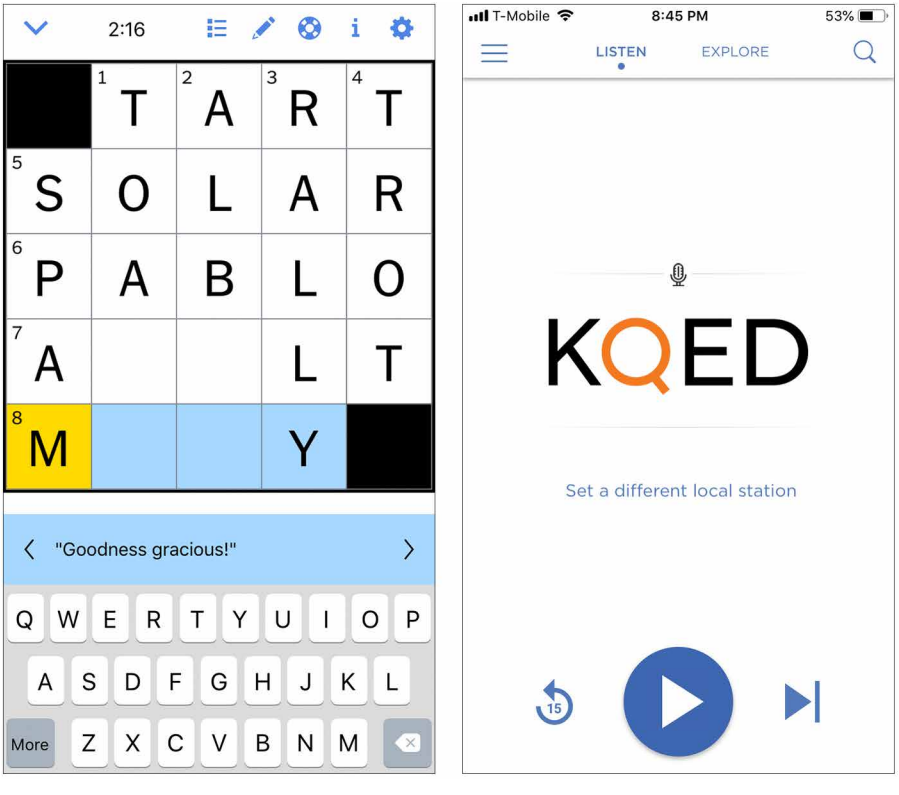


Figure 6-4. *The* New York Times *crossword puzzle on a native-iOS mobile app and the NPR One app*

Gratuity (Figure 6-5, left) is a tip calculator that gracefully places all the features on one screen and has buttons with ample touch targets.

Music Memos (Figure 6-5, right) allows a user to create a quick recording. The app is pared down to highlight the main functionality and offers a graceful animation to indicate the recording is taking place.

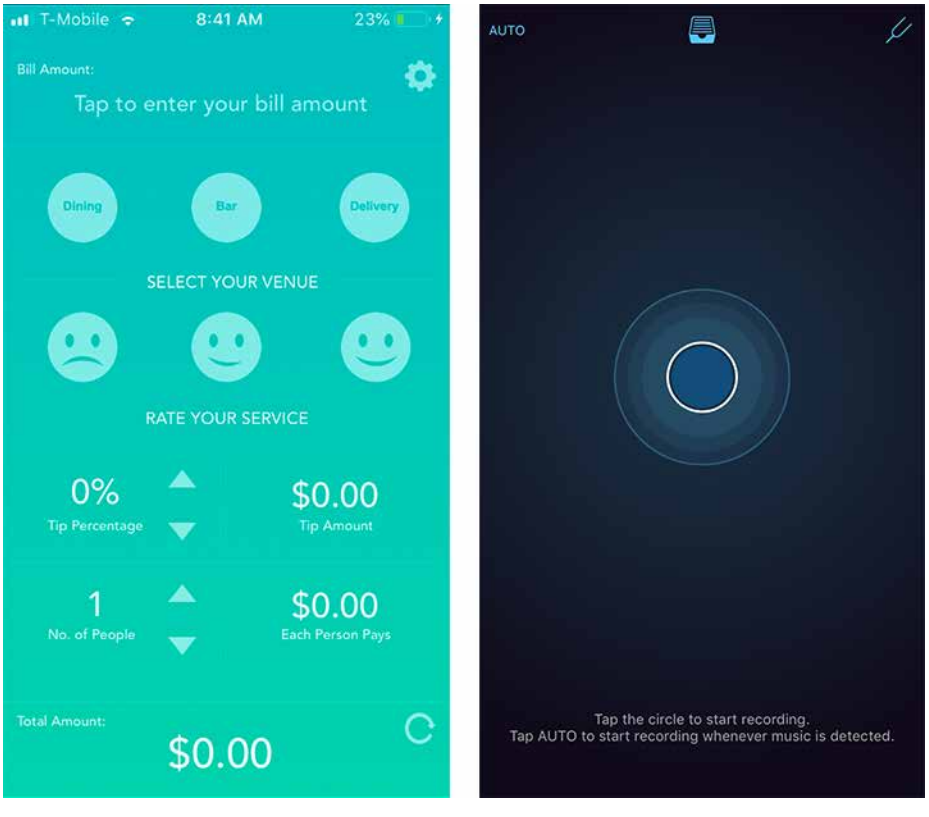


Figure 6-5. *Gratuity and Music Memos*

The Patterns

In the introduction, we talked about the need to structure content in a vertical column for maximum flexibility. The *Vertical Stack* pattern goes into more detail.

A mobile application needs a way to show its top-level navigational structure. A persistent toolbar across the top or bottom of each app screen is one standard way to organize a mobile interface; tabs and full-screen menus are two other common ways. Less obvious, yet worth mentioning, are the *Filmstrip* and *Touch Tools* patterns.

Mobile web screens often use the *Bottom Navigation* pattern for their global menus, preferring to use valuable top-of-screen space for more immediately relevant content.

Lists are everywhere in the mobile world—lists of apps, pictures, messages, contacts, actions, settings, everything! Both web screens and applications should present well-designed lists that look good and are usable. Ordinary text lists are often adequate, and *Carousel* and *Thumbnail Grid* work beautifully in mobile designs. (See Chapter 7 for those patterns and more discussion of list design.) Sometimes, an *Infinite List* suits the needs of mobile designs. Following are the other patterns we look at in this chapter:

- *Collections and Cards*
- *Infinite List*
- *Generous Borders*
- *Loading or Progress Indicators*
- *Richly Connected Apps*

Vertical Stack

What

Order the mobile screen's content in a vertical column, as shown in Figure 6-6, with little or no use of side-by-side elements. Text elements line-wrap, and the screen scrolls down past the bottom of most device screens.

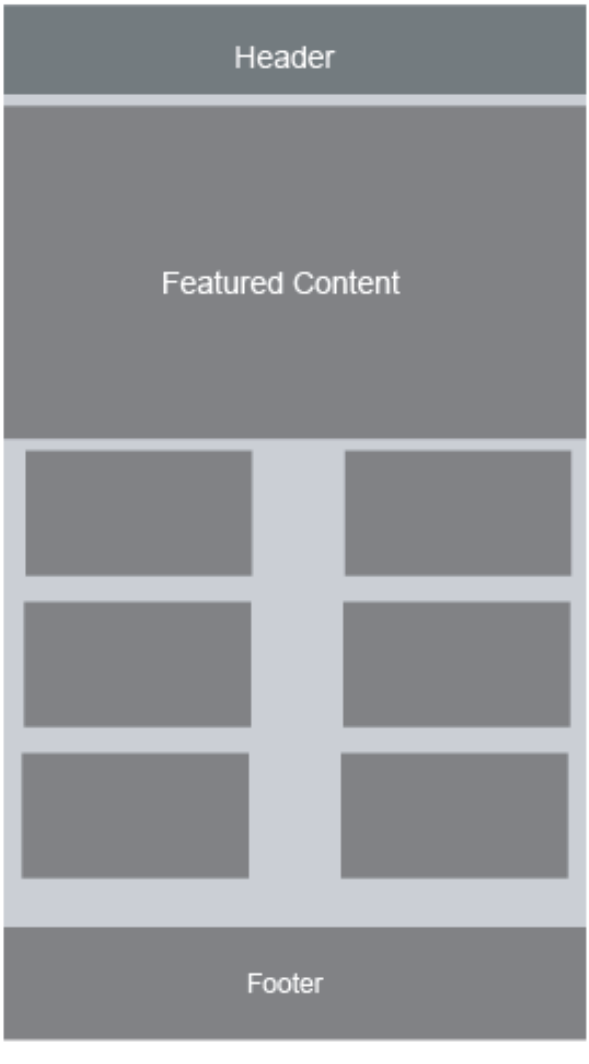


Figure 6-6. *Vertical Stack*

Use when

Most mobile web screens that must work on devices of different sizes should use this pattern, especially if they contain text-based content and forms. (Immersive content such as a full-screen video or games won't generally use this because it doesn't usually scroll like a text-based screen does.)

When going from one screen to another is expensive—as is the case with web screens, which take time to download—this pattern is applicable. On the other hand, an app that resides on the device can go from screen to screen almost instantly because the content doesn't need to be downloaded. For these, it makes more sense to structure the content into single screen so that the user never needs to scroll vertically—they can just tap or swipe. But vertical scrolling of a long screen is preferable to interminable waits for downloads.

Why

Devices come in different widths. You can't always anticipate what the actual width in pixels will be, unless you detect the screen width at runtime or build apps for particular devices. (You can create optimized designs for single devices or standard device-specific widths, but not everyone has the resources to do so.)

A fixed-width design that's too big for the physical device can scroll sideways or be zoomed, but these designs are never as usable as those that let the user simply scroll down.

Font sizes can also change, unbeknownst to you. A *Vertical Stack* with line-wrapped text elements will adjust gracefully when this happens.

How

Lay out the screen's content in a scrolling vertical column. Put the most important items on top and lesser important items farther down so that most users can see the essential material.

Useful content—from the user's perspective, that is—should show up in the first 100 pixels (or less) of the *Vertical Stack*. This top part of the screen is precious real estate. Don't waste it with too-tall logos, ads, or endless toolbars all stacked up into a "layer cake" that pushes all the useful content off the bottom of the screen. That annoys users to no end.

Put form labels above their controls, not next to them, to save horizontal space. You will need all the space you can get to show text fields and choice controls with adequate width.

Put buttons side by side only if you're really sure their total width will never be wider than the visible screen. If the buttons contain long text that might be subject to localization or font enlargements, forget it.

Thumbnail images can fit beside text fairly easily, and it's common to do this in lists of articles, contacts, books, and so on—see the *Collections and Cards* pattern. Make sure the design degrades well when the screen width is reduced to 128 pixels (or whatever the realistic minimum happens to be when you create your design).

Examples

The sites for ESPN, the *Washington Post*, and REI (Figure 6-7) demonstrate three styles of using a *Vertical Stack*. ESPN places only the most immediately relevant content on the home screen, preferring to put the rest behind menu items on a strip at the top of the screen. The *Washington Post* puts it all out there; the stack shown in the figure is just a small fragment of the entire screen! REI simply shows a menu of all the available places and ways to shop, with a teaser, on its home screen.

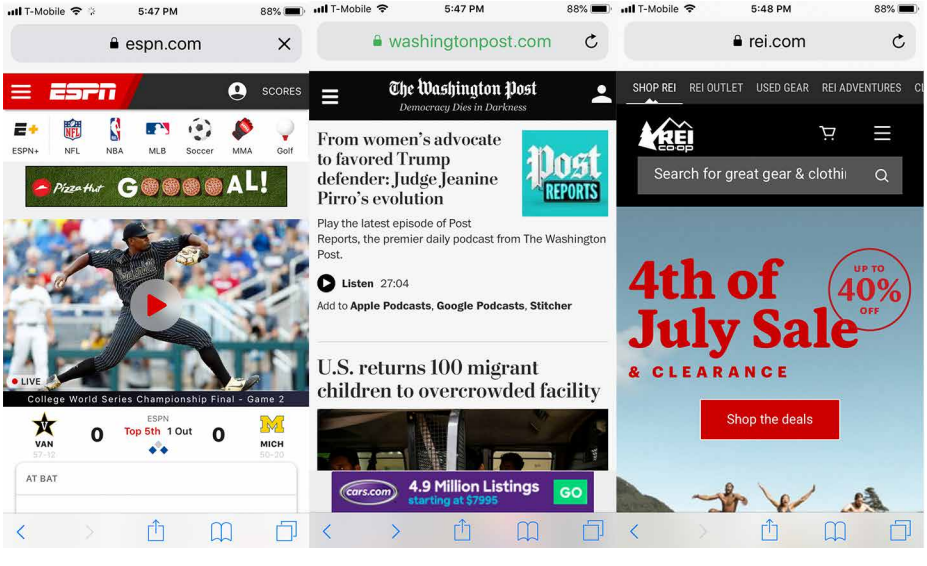


Figure 6-7. *Vertical Stack on the mobile sites for ESPN, the* Washington Post*, and REI*

Salon.com (Figure 6-8) features a vertical stack layout for its mobile web and mobile applications. This allows for flexibility in the amount of content it is able to show and makes it easy for a user to scroll the new articles with their thumb, making it possible to browse the content using only one hand.

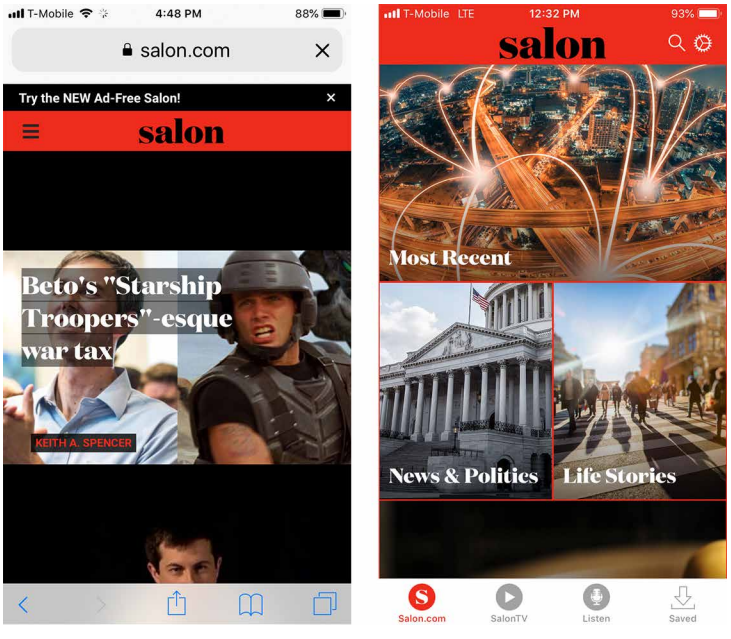


Figure 6-8. *Salon.com*

Filmstrip

What

Allow users to navigate by swiping back and forth to view content one screen at a time (Figure 6-9).

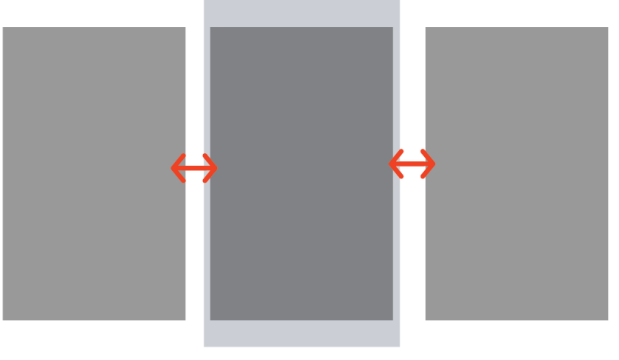


Figure 6-9. *Filmstrip*

Use when

You have screens of content that are conceptually parallel, such as the weather in different cities or the scores in different sports. Users won't mind swiping through these screens, going through several before reaching the one they're looking for, because they are all potentially interesting.

This pattern can sometimes be a viable alternative to other navigation schemes for mobile apps, such as toolbars, tabs, or full-screen menus.

Why

Each item to be displayed can occupy the entire screen; no space needs to be used for tabs or other navigation.

Because the user can't jump straight to a desired screen—they must swipe through others to get there—this pattern encourages browsing and serendipity.

Swiping seems to be a very satisfying gesture for some users.

A disadvantage of this pattern is that it doesn't scale very well; you can't use too many top-level screens, or users might become irritated at being forced to swipe too many times to get to a desired screen. Another disadvantage is lack of transparency. A new user, just seeing your app for the first time, cannot easily see that swiping is how they get from one screen to another.

How

Essentially, a *Filmstrip* is like a carousel for a mobile application's main screens. One difference is that a carousel usually shows metadata—information about the item or screen—and context, such as fragments of the previous and next screens. Mobile apps that use *Filmstrip* as a top-level organizing device don't generally do that.

If you want to give the user a clue that multiple top-level screens exist, and that they can swipe between them, use a dot indicator like the Weather app uses at the bottom of the screen.

Examples

The BBC News iPhone app (Figure 6-10) structures its main screens as a *Filmstrip*. The user can swipe or tap back and forth between the Top Stories, My News, Popular, Video, and Live.

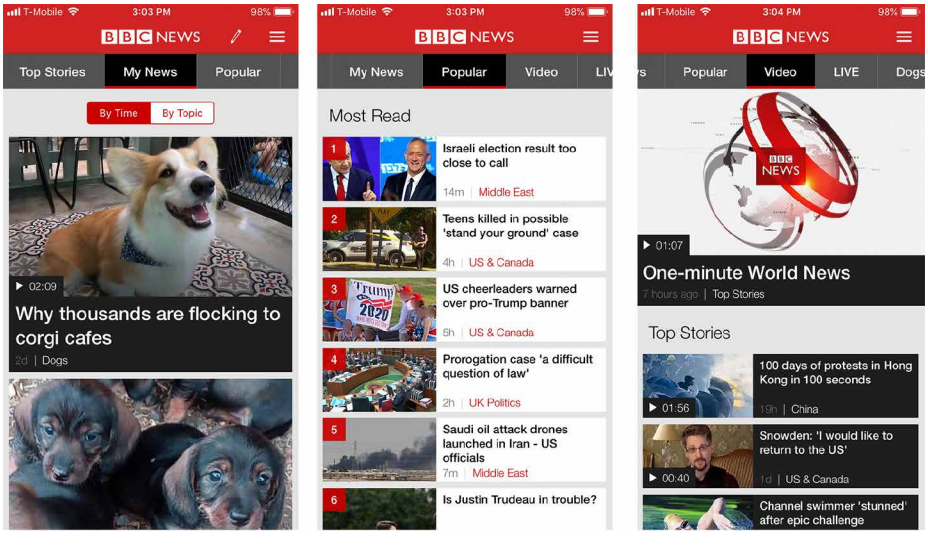


Figure 6-10. *BBC News*

The iPhone's built-in Weather app (Figure 6-11) uses a *Filmstrip* to show the weather in the various geographic locations that the user chooses.

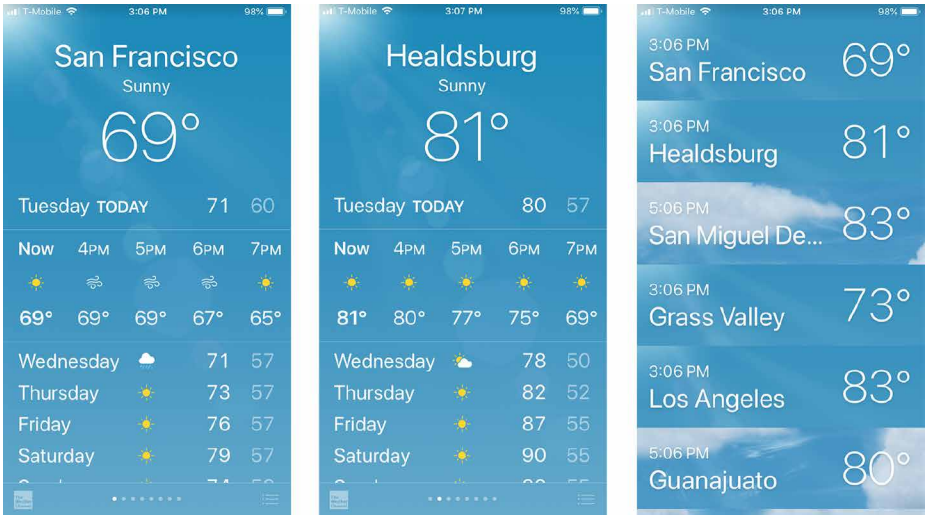


Figure 6-11. *iPhone Weather app*

Touch Tools

What

Show certain tools in response to a touch or key press. Functions appear in small, dynamic overlays atop the content.

Netflix (Figure 6-12) is a digital product that allows viewers to watch video content. The intent is that the user will be mostly focused on the video content but might want to pause, turn captioning on and off, rewind or fast forward the video from time to time. The user can invoke the functionality by tapping the screen. The options go away again after about five seconds of nonuse.

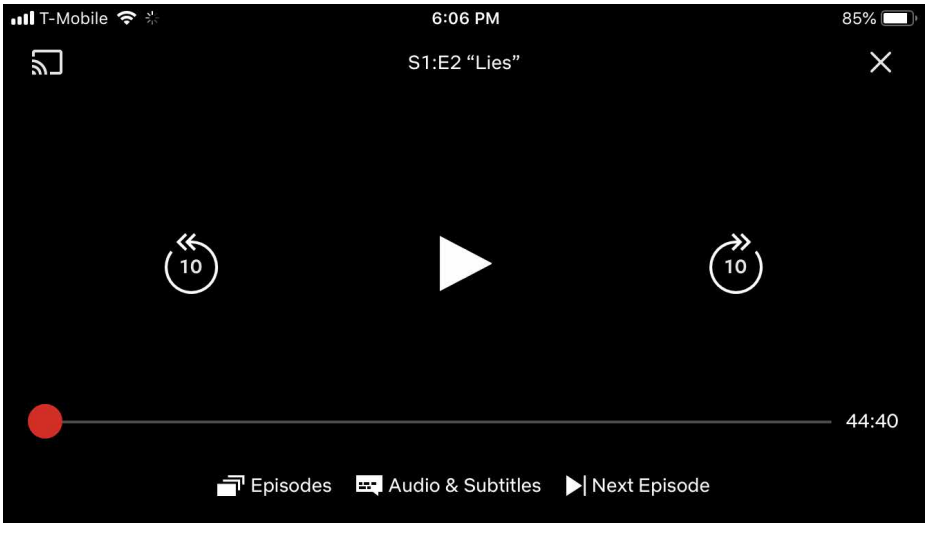


Figure 6-12. *Touch Tools on the Netflix mobile application*

Use when

You are designing an immersive or full-screen experience, such as videos, photos, games, maps, or books. To manage that experience, the user will sometimes need controls—navigation tools, media player tools, information about the content, and so forth. The tools require significant space but are needed only sometimes.

Why

The content is allowed to dominate the experience most of the time. The user isn't distracted by controls taking space and attention away from the content. Remember

that in a mobile context, space and attention are even more precious resources than usual.

The user controls the experience by choosing when to show the tools.

How

Show the unadorned content using the full screen. When the user touches the device's screen or presses a particular key or softkey, show the tools.

Many apps only show *Touch Tools* when the user touches a certain region of the screen. This way, the user doesn't accidentally bring up the tools just by ordinary handling of the device.

Show the tools in a small, translucent area that appears to float above the content. The translucency makes the tools look ephemeral (which they are).

Remove the tools after a few seconds of nonuse, or immediately if the user taps the screen outside the bounds of the tools. It can be annoying to wait for the tools to go away by themselves.

Examples

The video player on the iPhone shows *Touch Tools* when the user taps the indicated area of the screen (see Figure 6-13). They disappear after about five seconds of nonuse.

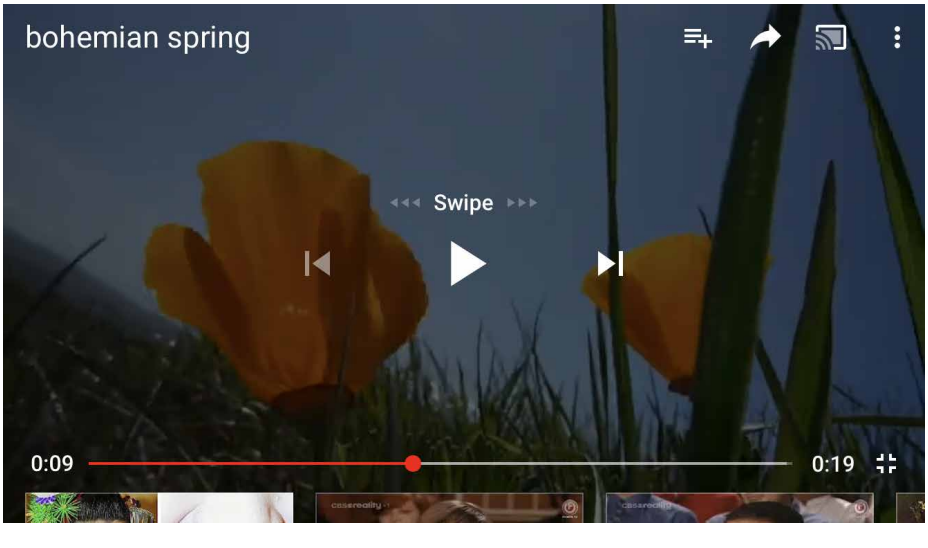


Figure 6-13. *YouTube for iPhone Touch Tools*

The Apple Notes application (Figure 6-14, left) allows the user to focus on reading the notes, but upon touching the screen, editing tools appear.

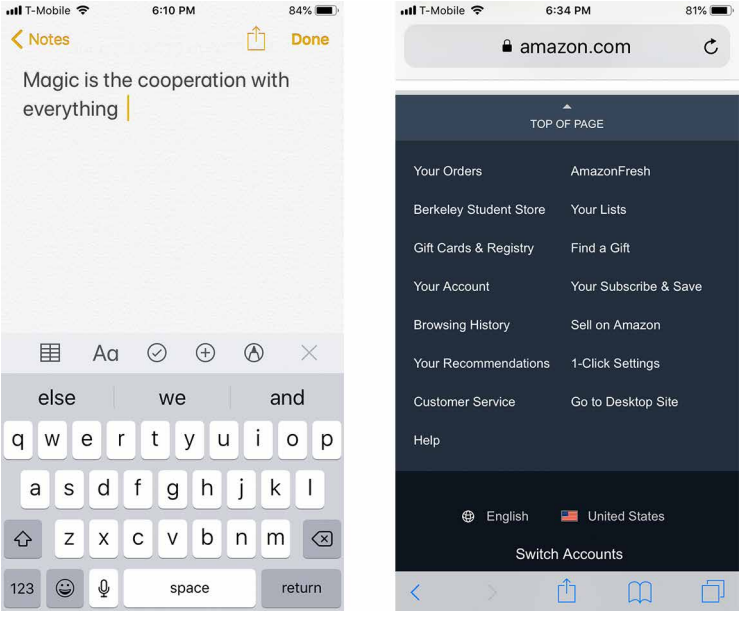


Figure 6-14. *Apple Notes and Amazon's Bottom Navigation*

Bottom Navigation

What

Place the global navigation at the bottom of the screen. Amazon uses a simple "Bottom Navigation" system on the global footer of its mobile website (Figure 6-14, right).

Use when

A mobile website needs to show some global navigation links, but these links represent low-priority paths through the interface for many users.

Your highest priority on the site's front screen is to show fresh, interesting content.

Why

The top of a mobile home screen is precious real estate. You should generally put only the two or three most important navigation links there—if any at all—and devote the rest of the front screen to content that will interest most users.

A user looking for navigational links can easily scroll to the bottom of a screen, even when those links are far below "the fold."

How

Create a set of vertically arranged menu items on the bottom of the screen. Make them easy to tap with a finger on touch screens—stretch them across the full width of the mobile screen, if necessary, and make the text large and readable.

In a mobile application, you probably aren't trying to fit an entire site map into the footer—you have room for only a few well-chosen links. But the idea is similar: instead of taking up too much top-of-screen space for navigation, you can push it to the bottom of the screen, where real estate is less valuable.

Examples

NPR puts a footer across the bottom of each of its screens (Figure 6-15, left). It includes standard navigational links and a text-only version.

REI uses a more web-like simple footer links and a large button with its 800 number (Figure 6-15, right) that is a good size for mobile use.

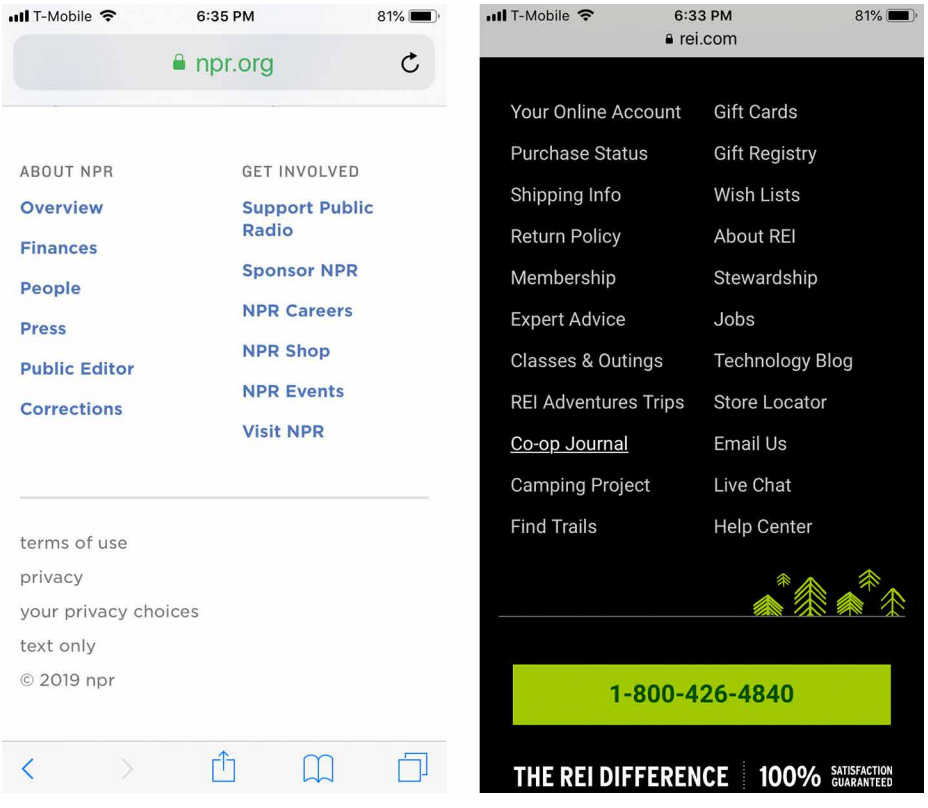


Figure 6-15. *NPR's Bottom Navigation and REI's footer*

Collections and Cards

What

Collections are series of thumbnail photos that serve as a list of items from which a user can choose, as illustrated in Figure 6-16. Cards are similar but they contain content and functionality. You will see these used frequently on ecommerce sites, sites that have video content, and news sites.

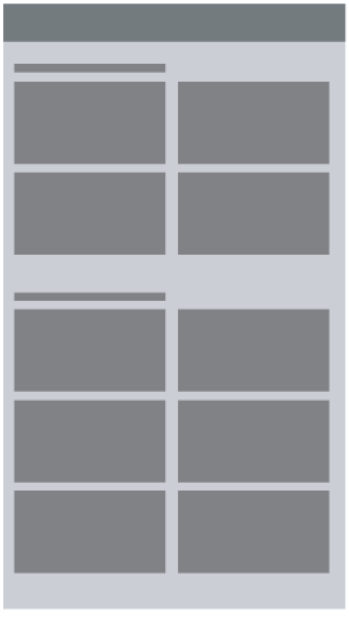


Figure 6-16. *The Collections and Cards pattern*

Use when

You need to show lists of articles, blog entries, videos, applications, or other complex content. Many or all of these have associated images. You want to invite the user to click or tap these items and view them.

Why

Thumbnail images improve text-only lists because they look appealing, help identify items, and establish a generous height for the list items.

Reading conditions on mobile devices are rarely ideal. By adding colorful images, you can improve the visual differentiation among items, which helps people scan and parse the list quickly.

Many news and blog websites have converged on this design pattern as a way to show links to their articles. They look more appealing, and more "finished" than similar sites that list only article titles or text fragments.

How

Place a thumbnail image next to the text of the item. Most sites and apps put the thumbnail on the left.

In addition to picture thumbnails, you can include other visual markers, such as five-star ratings or icons representing people's social presence.

Don't be afraid to use bright or saturated colors. You probably wouldn't design so much visual stimulation in a desktop context, but in a mobile context, it works. Even if the colors seem garish, don't worry—small screens can handle strong colors better than large screens can.

Examples

Many news sites use this pattern to show their articles, and videos and other media fit this pattern naturally. These help a user scan down a list and pick out items. *Jacobin*, NPR and *The Atlantic* offer examples of this pattern effectively for their feature articles, in Figure 6-17.

This pattern is extremely versatile for most contexts. In the example in Figure 6-18, you will see a variety of apps that show a variation of the thumbnail and text or card list.

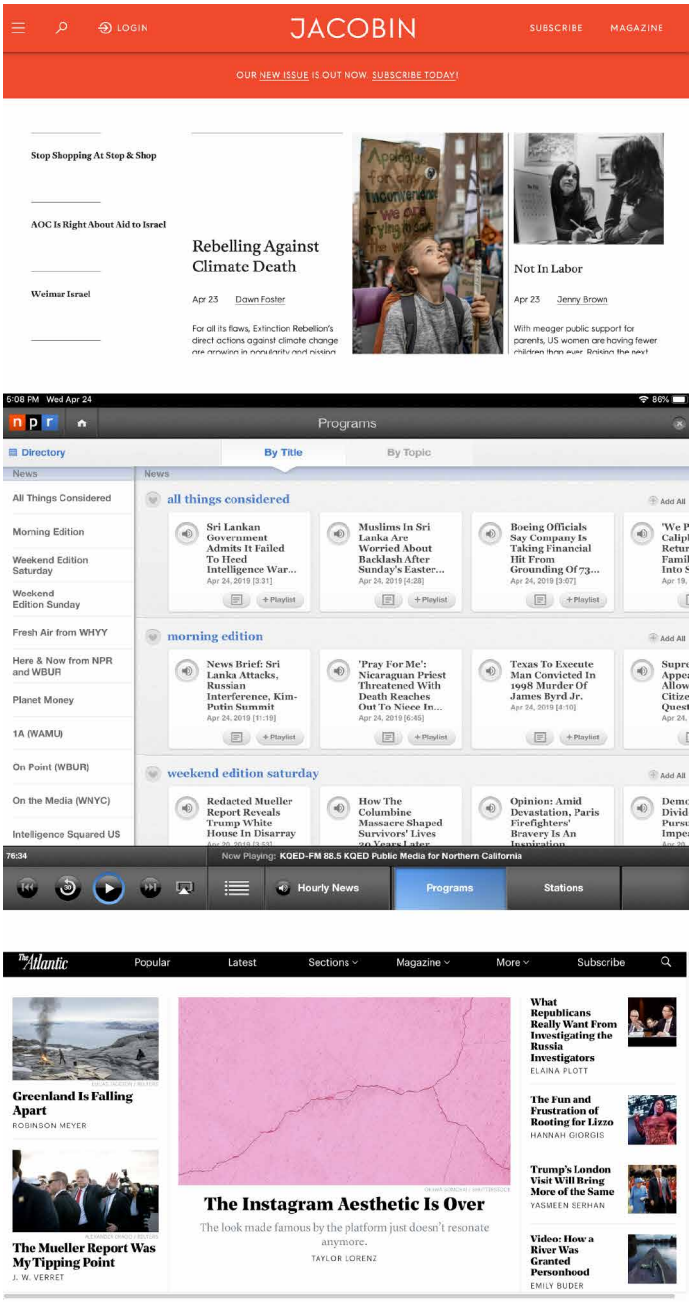


Figure 6-17. Jacobin, *NPR, and* The Atlantic *iPad apps*

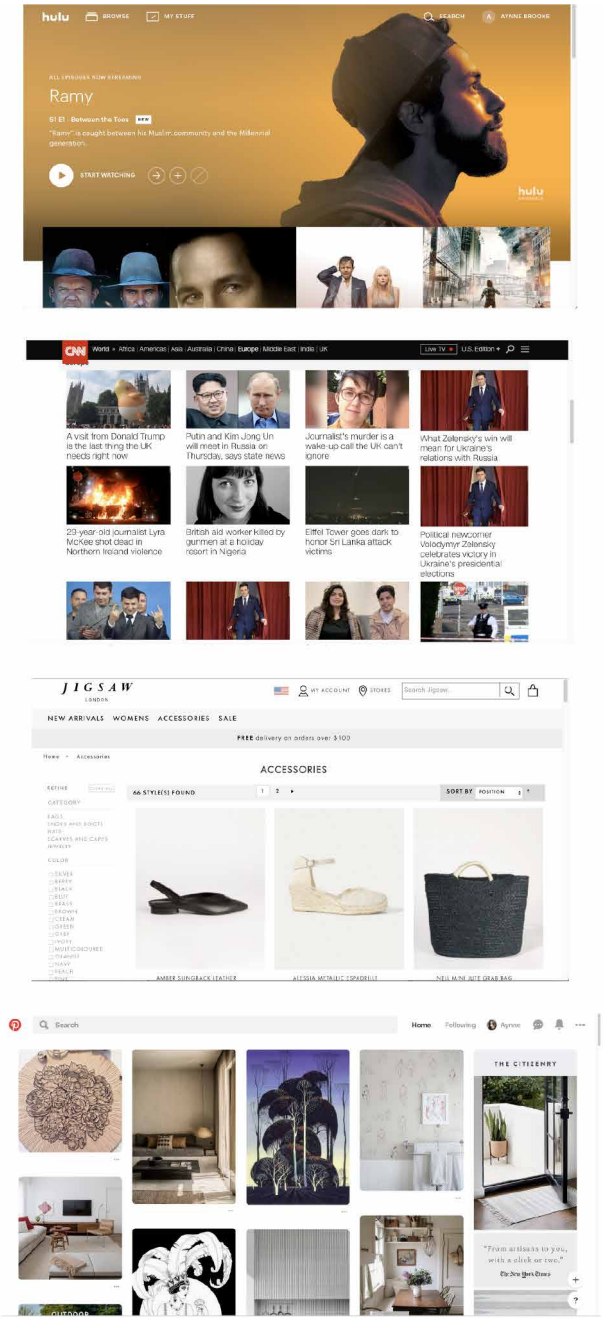


Figure 6-18. *Hulu, CNN, Jigsaw and Pinterest iPad apps*

Infinite List

What

An *Infinite List* shows more and more content as the user scrolls to the bottom of a long list, as demonstrated in Figure 6-19.

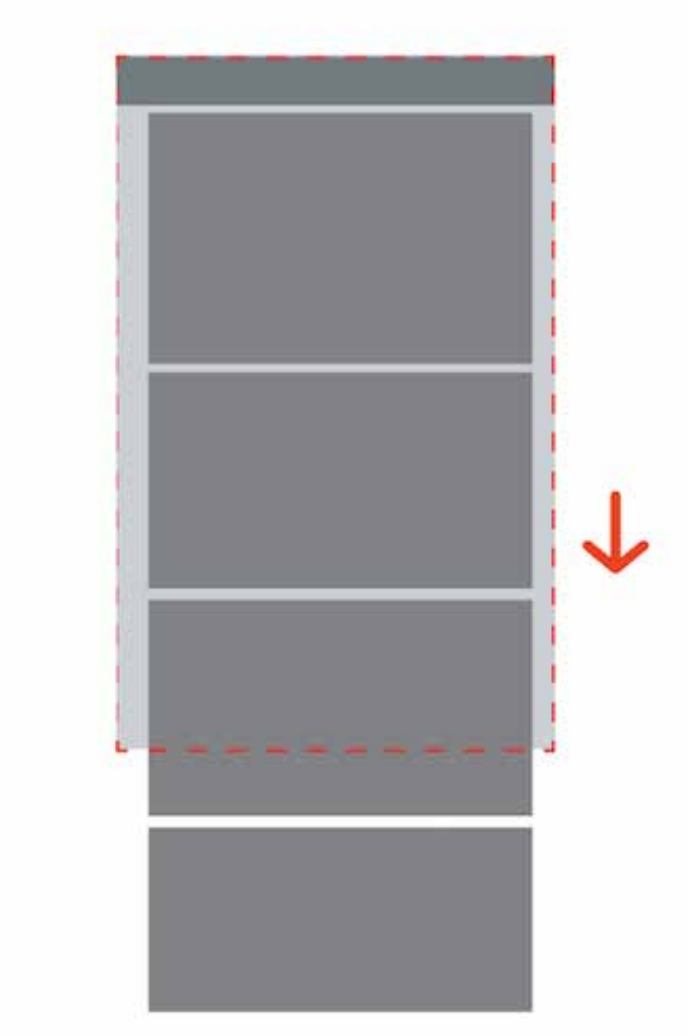


Figure 6-19. *Infinite scrolling*

Use when

You need to show long lists of email messages, search results, an archive of articles or blog posts, or anything else that is effectively "bottomless."

Users are likely to find desired items near the top, but they sometimes need to search further.

Why

The initial loading of a screenful or two of items is fast, and the user isn't forced to wait for a very long initial screen load before they see anything useful.

Each subsequent loading of a new chunk of items is also fast, and it's under user control—the user decides when (and whether) they need to load more items.

Because the new items are just appended to the current screen, the user never needs to context-shift by going to a new screen to see new items, as they would with paginated search results.

How

When the screen or list is initially sent to the mobile device, truncate the list at a reasonable length. That length will vary greatly with item size, download time, and the user's goal—is the user reading everything, or just scanning a large number of items to find the one they want (as with search results)?

At the bottom of the scrolled screen, put a button that lets the user load and show more items. Let the user know how many more will be loaded.

Alternatively, you could use no button at all. After the user has loaded and can view the first chunk of items, silently start loading the next chunk. Append them to the visible list when they're ready, and the user has scrolled down to the end of the original list. (This is your clue that the user might want to see more. If the user doesn't scroll down, don't bother getting more items.)

In software engineering, this well-known approach to managing lists of undefined length is often called *lazy loading*.

Examples

Several iPhone applications use *Infinite List*, including Mail, as well as third-party apps such as Facebook (Figure 6-20, left). The Facebook mobile web, like the full-size Facebook screen, loads up the first several screens of updates and then lets the user load more.

Apple's email application (Figure 6-20, right) offers an endless scroll to accommodate varying lengths of email a user might have in their inbox.

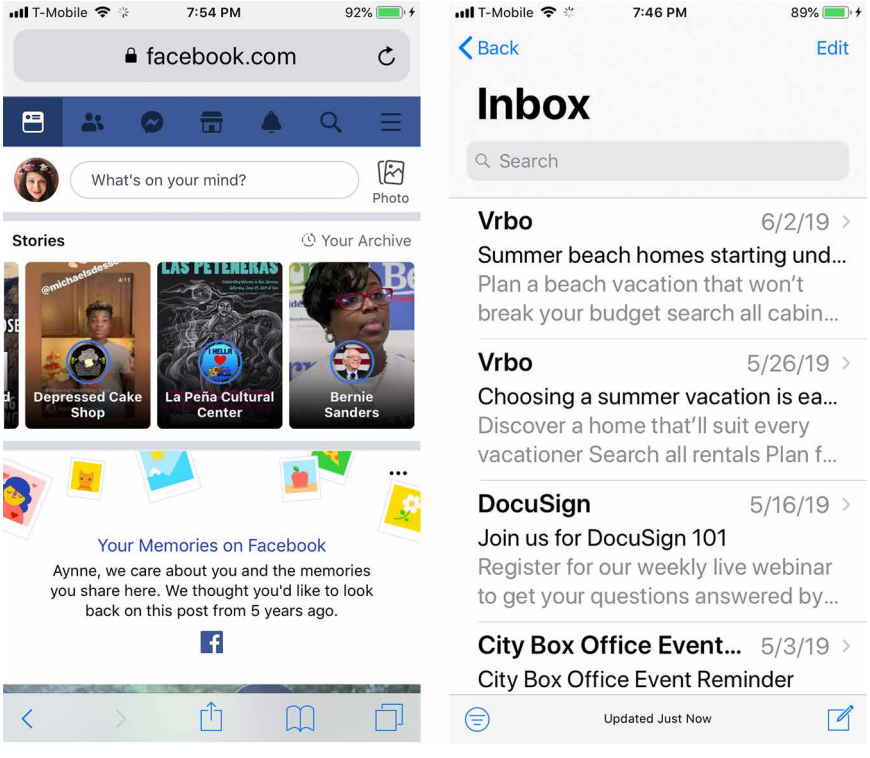


Figure 6-20. *Facebook and Apple iOS email*

Generous Borders

What

Leave lots of space around tappable UI elements. On devices with touch screens, put large margins and whitespace around buttons, links, and any other tappable control.

Figure 6-21. *Generous Borders*

Use when

You need to use buttons with text labels, or a list of items, or ordinary text-based links—in short, any touch target that isn't already large on the screen.

Why

Touch targets must be large enough for fingers to tap successfully. In particular, they need to be tall enough, which is challenging for buttons and links that consist only of text.

How

Surround each touch target with enough inner margin, border, and surrounding whitespace to make a sufficiently large hit target for fingertips.

One trick is to make the whitespace immediately surrounding a target tappable. The button will look the same size, thus fitting into your visual design as expected, but you gain a few pixels of sensitivity in each direction around the button. Exactly how big to make these targets is a very good question. Ideally, you want a size that ends up large enough on the physical device to be manipulated by most people—many of whom will have large fingers. Some others will not have great control over their fingertips. Yet others will be using their mobile devices in challenging conditions: bad light, moving vehicles, little attention to spare.

So ultimately, how big should you make your targets? It depends on what device and resolution you are designing for, but here are a couple of dimensions for reference:

- 48 × 48 dp (9 mm) for Android devices
- 44 pt × 44 pt for Apple iOS

Examples

Autodesk's Sketchbook application (Figure 6-22, left) for the iPhone reliably puts plenty of margin space around its touch targets. The entire application has a relaxed, uncramped feeling.

The Zoom mobile app (Figure 6-22, right) has large buttons that make it easy for a finger to hit the touch target.

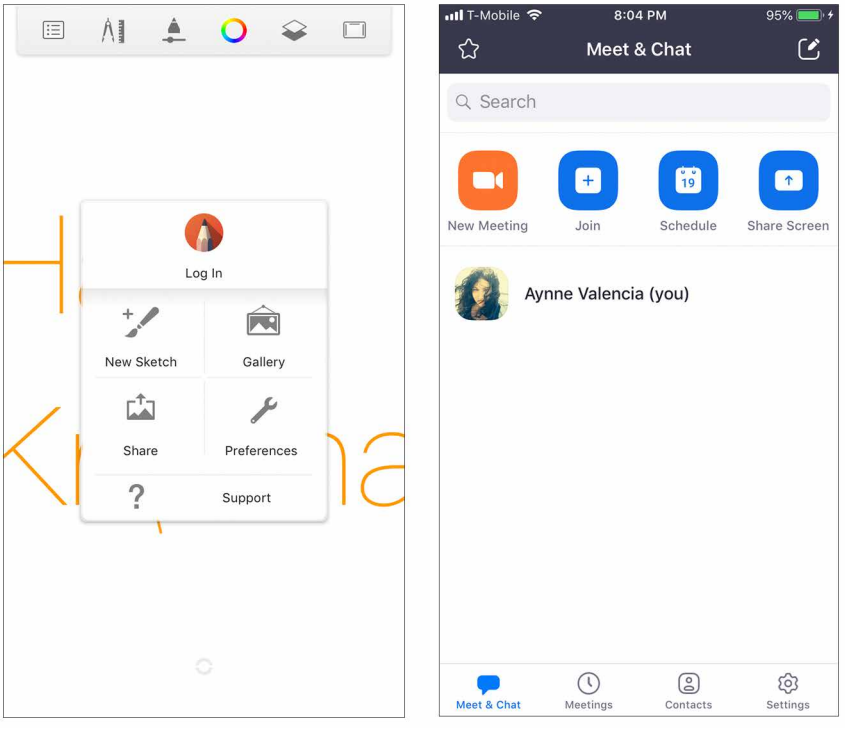


Figure 6-22. *Autodesk's Sketchbook and Zoom app*

The Instacart app (Figure 6-23) is similar, though its visual styling is quite different. The buttons for key actions—"add item," navigation elements—are distinctive.

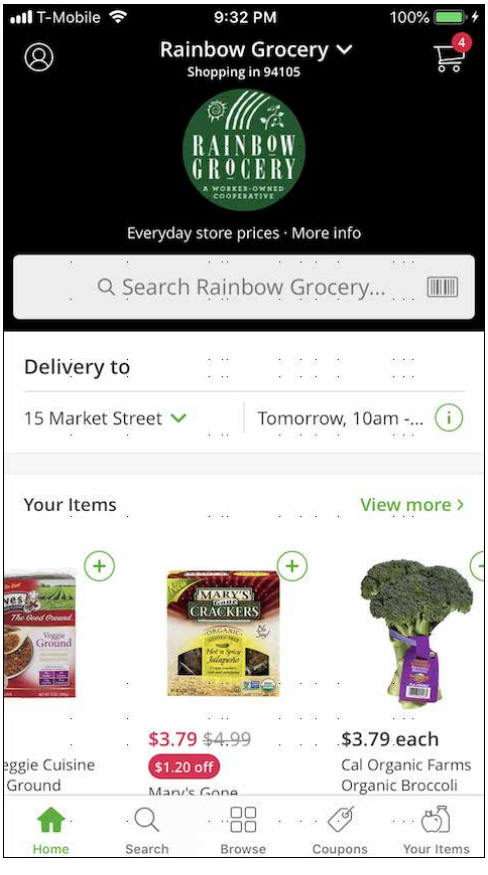


Figure 6-23. *Instacart*

Loading or Progress Indicators

What

Use *microinteraction* (a single task-based event with a device) animations used to inform the user that something is going to happen but it just hasn't appeared on the screen yet. You can use these to indicate an unspecified or estimated amount of time it will take for the screen to load or for a task to be completed and show on the screen. Done well, they can make a slow download bearable and even be a way to create a brand-building moment.

Use when

The user has to wait for content to load, especially in a screen that changes dynamically in response to user interaction.

Why

Loading new content can be slow and erratic over mobile connections. You should always show as much of a partially loaded screen as you can so that the user can actually see something useful.

In general, progress indicators make loading times appear faster to a user. They are reassured that something is actually happening in response to a gesture, especially when that indicator appears where the gesture occurred.

How

Show as much of the screen as can be loaded quickly, but if part of it takes a long time, such as a graphic or video, show a lightweight animated progress indicator where the graphic will appear. (The mobile platform might supply a default indicator.)

When the user initiates an action that causes part of the screen to be reloaded—or loads a whole new screen—show a progress indicator *in situ* on the screen.

Examples

Trulia (Figure 6-24, left) is a real estate application that uses an *infinite loop* on the top of the screen to indicate the screen is loading, and image placeholders to let the user know something is about to happen.

SoundHound (Figure 6-24, right) is an app that allows a user to "listen" to music to search for the name and artist of a song. After a user taps the button to listen to a song, a beautiful and simple animation starts to indicate that the system is listening and searching for a match.

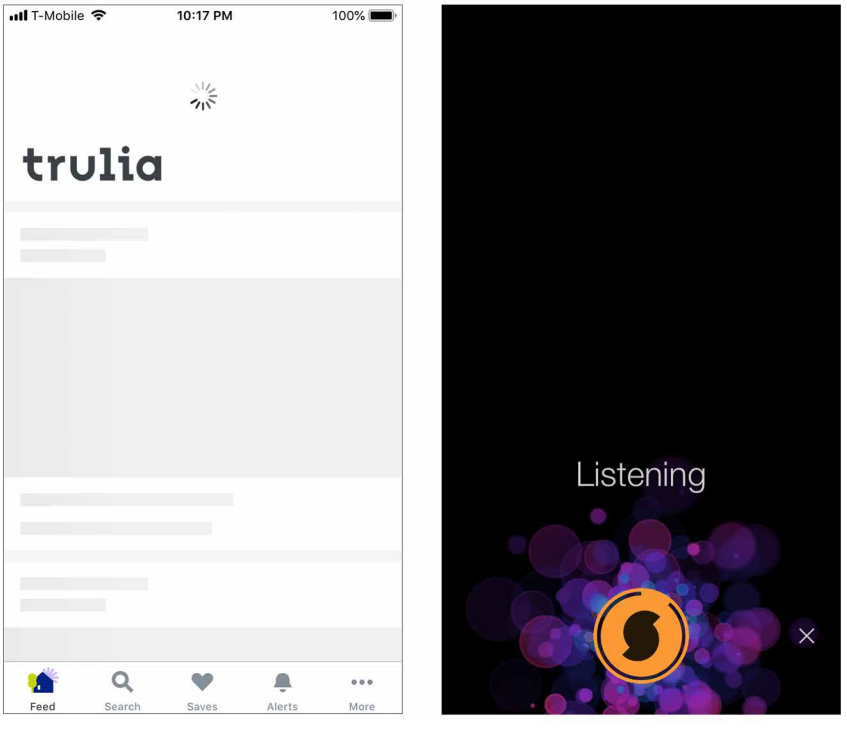


Figure 6-24. *The Trulia loading screen and SoundHound's animated progress indicator*

When an iPhone installs a new app, a miniature circular progress indicator appears over the app's icon to show how far it's gotten with the download (see Figure 6-25). It's cute, and its meaning is unmistakable.

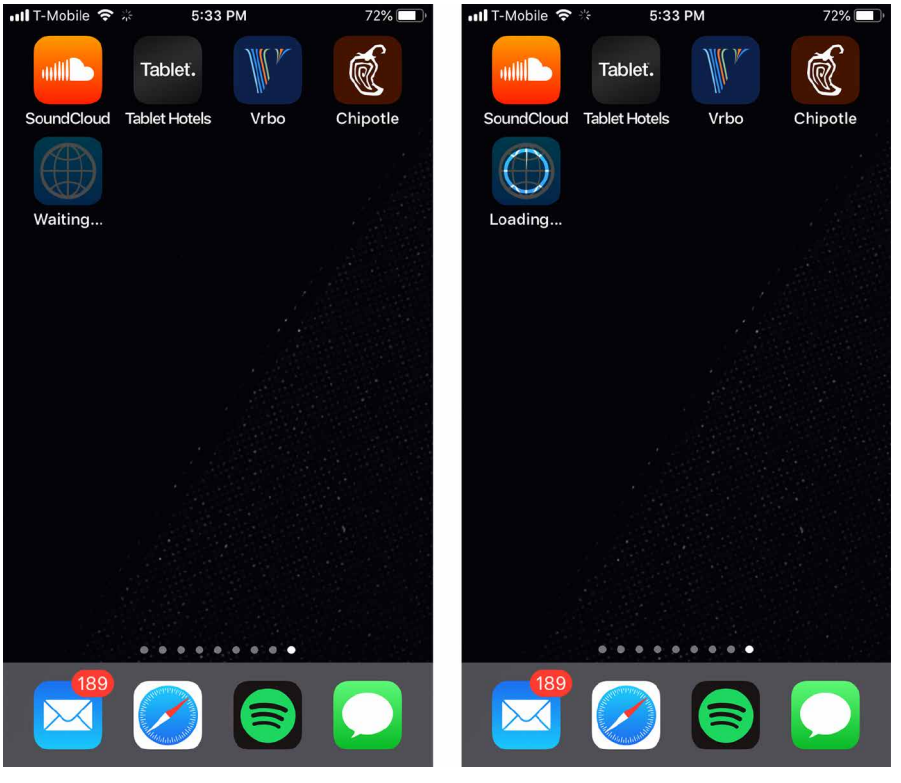


Figure 6-25. *iPhone's app installation progress indicator*

Richly Connected Apps

What

Use the features that come for "free" with your mobile device. Some examples of these are direct links to other apps, such as the camera, the phone dialer, map, or browser; and prefilling credit card passwords and address with data from the user's current context.

Use when

The mobile app shows data that is "connectable" in obvious ways, such as phone numbers and hyperlinks.

More subtly, your app can offer ways to capture images (via the device camera), sound, or video. It might even be aware of social networking conventions, such as Facebook or Twitter usernames. In all cases, your app might direct the user to another app to perform these device-based functions.

Why

A user can see only one mobile app at a time, even when multiple apps are being used at once, and it's annoying to switch between them by hand.

Mobile devices often have enough context and available functionality to offer intelligent paths between apps.

As of this writing, mobile devices have no good way to arbitrarily shuffle small amounts of information from one application to another. On the desktop, you can easily type, or use copy and paste, or even use the filesystem. You don't have those options on a mobile platform. So, you need to support moving that data automatically.

How

In your app, keep track of data that might be closely associated with other apps or services. When the user taps or selects that data, or uses special affordances that you provide, open another app and handle the data there.

Here are some examples. Consider all the ways that data in your app can connect directly to other mobile functions:

- Phone numbers connect to the dialer
- Addresses connect to the map, or to the contacts app
- Dates connect to the calendar

- Email addresses connect to the email app
- Hyperlinks connect to the browser
- Music and videos connect to media players

In addition, you might be able to do such things as take a picture, or use a map, entirely within the context of your application.

You can do some of this on a desktop, but the walled-garden nature of many mobile devices makes it easier to launch the "right" app for certain kinds of data. You don't need to decide which email reader to use, or which address or contact management system, and so on. Plus, many mobile devices supply a phone dialer, a camera, and geographic location services.

Examples

Citizen (Figure 6-26) is a live information reporting application that uses geolocation data to keep the user informed about crime and other activity that is going on around them. When a user wants to contribute information, the app uses the user's geolocation data and invokes the phone's camera or video recording functionality so that the user can post it directly from the app.

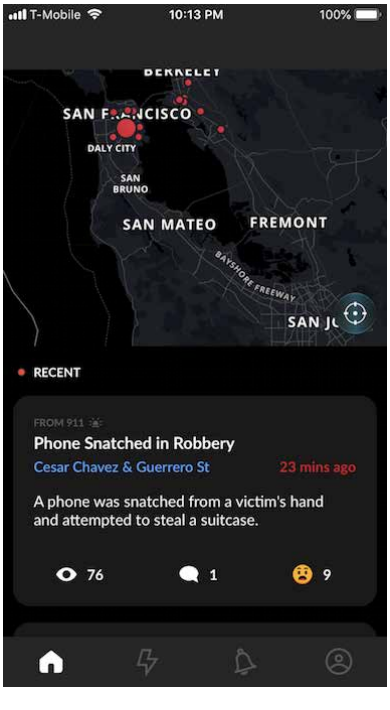


Figure 6-26. *Citizen*

Simple (Figure 6-27) is a banking application. To deposit a check, the camera functionality is invoked and the checks information is read by the application.

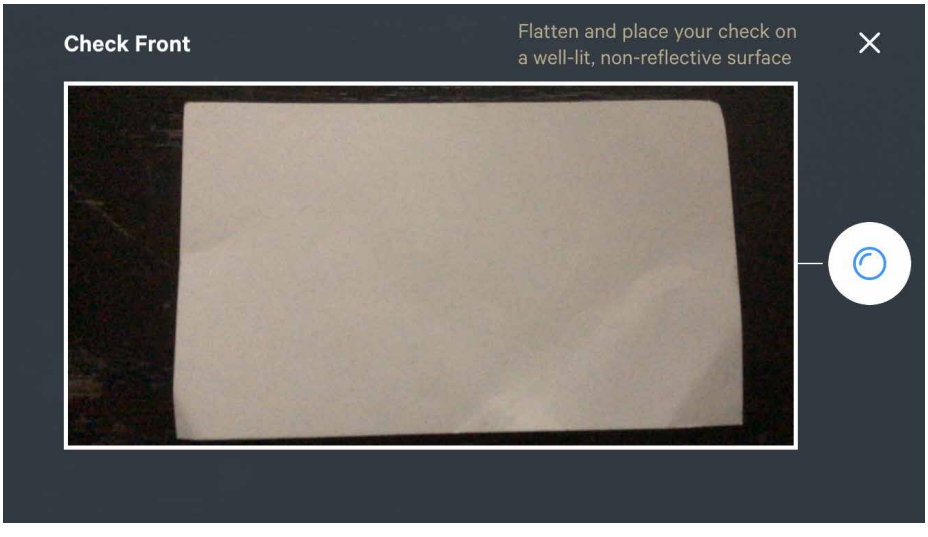


Figure 6-27. *Simple's integration with the camera*

Google's Calendar application (Figure 6-28) integrates with the phone, maps, contact lists, and email.

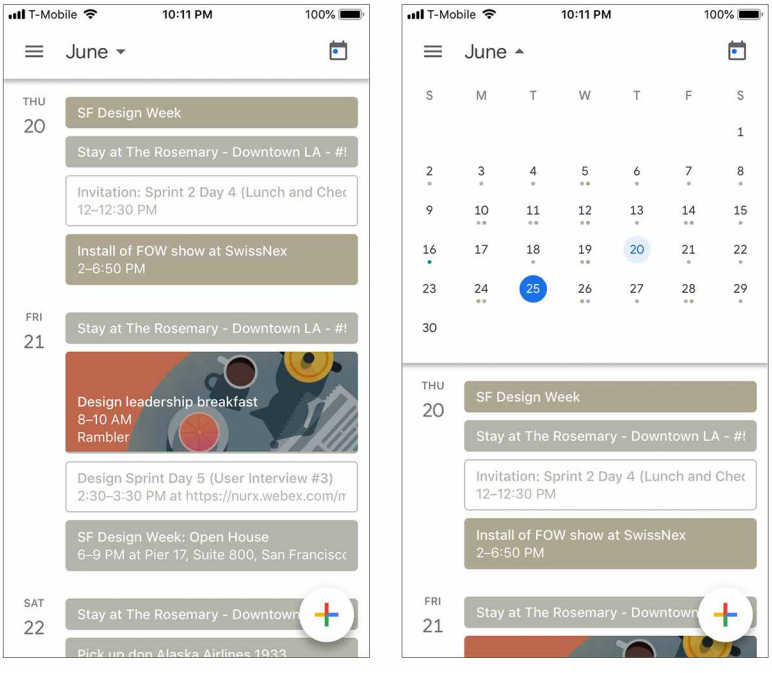


Figure 6-28. *Google's Calendar app*

Make It Mobile

It isn't difficult to see that smartphone and tablet interfaces are not afterthoughts but critical to the success of digital products. The mobile web or application you create might be the primary way a user experiences your brand, so give special attention to the small details, microinteractions, usability, and the mobile context of use.

Chapter 7

Lists of Things

This chapter covers only one topic: how to display lists of items in an interactive setting. Just lists.

Why do lists merit their own chapter, you may ask?

Lists are everywhere in screen design. Consider the many types of items that we show in lists: articles, screens, photos, videos, maps, books, games, movies, TV shows, songs, products, email messages, blog entries, status updates, forum posts, comments, search results, people, events, files, documents, apps, links, URLs, tools, modes, and actions. And that list of lists goes on and on.

Practically every interface or website ever designed includes lists. This chapter will help you think about them logically and clearly, understand different design aspects, and make good trade-offs when designing interfaces that use lists.

Use Cases for Lists

Before jumping into a design, it's useful to analyze the use cases for a list. What will people need to do with the list? Consider which of these scenarios apply:

Getting an overview
: What impression will someone get from the list as a whole? In some cases, a user should be able to skim down the list and understand what it's about. Often that requires more than just words; it might require images or carefulvisual organization to convey that impression.

Browsing item by item
: Will the user peruse items randomly or in order? Do they need to click or tap items to open them? If so, it should be easy to go back to the list and find another item, or move directly to the next one.

Searching for a specific item
: Is the user looking for something in particular? They should be able to find it quickly, with a minimum of clicks, scrolling, and back-and-forth.

Sorting and filtering
: If someone is looking for an item or group of items with a specific characteristic (e.g., "anything with a date between X and Y") or is looking for general insight into a set of data, sorting and filtering functions might help.

Rearranging, adding, deleting, or recategorizing items
: Does the user have the need to rearrange the items? Does the user own the list and the items within it? Most apps and sites that show personal collections permit direct manipulation of those lists so that the user can drag items around into a desired order or grouping scheme. They should also be able to select multiple items at a time for moving, editing, or deleting; a design should either use the platform standards for multiple selection (e.g., Shift-select or tapping in an edit mode), or supply checkboxes beside each item to permit the user to select an arbitrary subset.

Back to Information Architecture

We discussed information architecture in Chapter 2—organizing information, independent of its visual representation. Let's return to it for a minute. If you have a list of things to show on a screen, what are the salient nonvisual characteristics of that list?

- Length
  - How long is the list? Can it fit in the space you've designed for it?
  - Could the list sometimes be "bottomless"? For example, web search results often constitute such a long list that the user will never reach the end; likewise for items taken from a very large and deep archive.
- Order
  - Does the list have a natural order, such as alphabetical or by time? (See Chapter 2 for a more in-depth discussion of ways to organize data and content.)
  - Would it make sense for a user to change the sorting order of the list? If so, what would the user sort on?
  - If you choose to put a list into an order, would it actually make more sense as a grouping scheme, or vice versa? As an example, think about a blog archive: the articles are naturally ordered by time, and most blogs categorize them by month and year, rather than offering a flat ordered list. Someone looking for a particular article might remember that "it was before article X but after article Y," but not remember exactly which month it was published. A monthly

grouping thus makes it hard to find that article; a time-ordered flat list of titles might work better.

- Grouping
  - Do the items come in categories? Is it a natural categorization that users will immediately understand? If not, how can you explain it, either verbally or visually?
  - Do these categories come in larger categories? More broadly, do the items fit into a multilevel hierarchy, such as files in a filesystem?
  - Are there several potential categorizations? Would they fit different use cases or user personas? And can users create their own categories for their own purposes?
- Item types
  - What are the items like? Are they simple, or are they rich and complex? Are they just stand-ins for larger things, such as headlines for articles or thumbnails for video clips?
  - Are the items in a list very different from each other (e.g., some are simple and some are complex)? Or are they homogeneous?
  - Does each item have an image or picture associated with it?
  - Does each item have a strict field-like structure? Would it help the user to know that structure, or possibly even sort the list based on different fields? (Email messages typically have a strict and sortable structure—timestamp, from field, subject, etc.—and this structure is shown in lists of messages.)
- Interaction
  - Should you show the whole item at once in the list, or can you just show a representation of the item (such as its name or the first few sentences) and hide the rest?
  - What is the user supposed to do with those items? Should they be looked at? Should they be selected for inspection, or for performing tasks on them? Or are they links or buttons to be clicked on?
  - Does it make sense for the user to select multiple items at a time?
- Dynamic behavior
  - How long does it take to load the entire list? Can it be more or less immediate, or will there be a noticeable delay as the list is put together somewhere and finally shown to the user?
  - Will the list change on the fly? Should you show the updates as they happen? Does this mean inserting new items at the top of the list automatically?

The answers to these questions might suggest a variety of design solutions to you. Of course, a solution should also take into account the type of content (blogs should look different from, say, contact lists), the surrounding screen layout, and implementation constraints.

What Are You Trying to Show?

The interaction questions listed in the preceding section set the tone for almost all other decisions. For instance, a fully interactive list—multiple selection, drag-and-drop, editing items, and so on—tends to dominate the interface. You might be building a photo management app, an email client, or some other full-fledged application that people use to manage and enjoy the content that they own.

In this and other types of interfaces, a common requirement is to show only item names or thumbnails in a list—just a representation of each item—and then display the whole item when the user selects one from the list. There are at least three ways to do this.

"When the user selects an item from a list, where should I show the details of that item?"

Two-Panel Selector or Split View shows the item details right next to the list. It supports the overview and browsing use cases quite well because everything's visible at once; the surrounding screen stays the same, so there's no awkward context switch or screen reload.

One-Window Drilldown replaces the list's space with the item details. This is often used for small spaces that cannot accommodate a *Two-Panel Selector*, such as mobile screens or small module panels. It does lead the user to "pogo-stick" between the list screen and the item screen, though, so browsing and searching are not so easy.

List Inlay shows the item details embedded in the list itself. The list expands to show the detail when the user clicks or taps. This pattern supports the overview and browsing use cases, too—though an overview is more difficult if lots of items are open.

Now let's shift our attention to the items that appear on a list. How much detail should you show with each item, assuming that the user will click or tap through to see the whole thing? Again, you have three main use cases to serve: get a quick overview, browse the list, and find items of interest. For really focused tasks, such as finding a person's phone number in a long contact list, all that's needed is the item name. But for a broader, more browsing-oriented experience—news articles on a web screen, for instance—more information makes an item more interesting (up to a point, anyway). And if you have visuals associated with each item you should show the thumbnails.

"How can I show a list of items that have heavy visuals?"

Cards consolidate images, text and functionality into one user interface (UI) element. See the *Grid of Equals* (Chapter 4) for the basis of this pattern.

Thumbnail Grid is a common pattern for pictorial objects. A 2D grid of small pictures is visually powerful; it dominates the screen and draws attention. Text data is often shown with the thumbnails, but it tends to be small and less important than the pictures. Again, see the *Grid of Equals* pattern for a generalization.

Carousel is an alternative to Thumbnail Grid that can use less space on the screen. It is strictly linear, not 2D, and the user must actively scroll through it to see more than a few objects. Depending on its design, a Carousel implementation might actually give you more space to show the selected or center object than a Thumbnail Grid.

Very long lists can be difficult to design. Certainly, there are technical challenges around loading times and screen length, but interaction design might be even harder —how does a user browse and move through such a list? How can they find something specific, especially if a text search doesn't behave as desired? The following techniques and patterns apply to all the previously listed ways to show a list and its items (except maybe a *Carousel*, which has tighter constraints):

"How can I manage a very long list?"

Pagination lets you load the list in sections, putting the onus on the user to load those sections as needed. This is, of course, quite common in websites—it's easy to design and implement. Pagination is most useful when the user is likely to find the desired item(s) in the first screen given that many people won't bother going to subsequent screens anyway. You could also resort to Pagination when loading the entire list will result in a ridiculously long screen or take a ridiculously long time. A good Pagination control shows the user how many screens of items there are as well as letting a user jump among those screens.

Common in mobile designs, the *Infinite List* (Chapter 6) is a single-screen alternative to Pagination. The first section of a long list gets loaded, and at the bottom the user finds a button that loads and appends the next section. The user stays on one screen. This pattern is useful when you don't actually know how long the list will be, or when it's bottomless.

A variant on Infinite List has the list automatically loading itself as the user scrolls down. See the *Continuous Scrolling* pattern.

When a very long alphabetized list or range of dates is kept in a scrolled box, consider using an *Alpha/Numeric Scroller*. This device shows the alphabet arrayed along the scroll bar itself; the user can then jump directly to a desired letter.

Direct searching via a *Find* field may be critical for helping your users to find specific items. Also, filtering a list—screening out entire classes of items that don't meet certain criteria—can help shorten a list to a manageable size.

So far, this section has talked mostly about flat lists: those that have no categories, containment, or hierarchy. However a list might be rendered, you might still want to break it up into categories for clarity.

"How can I show a list that's organized into categories or hierarchies?"

Titled Sections work well for a single level of containment. Just separate the list into sections with titles, and perhaps allow the user to sort the list within a single section so as not to disrupt the categorization. If you have only a few sections, try an *Accordion*—this lets the user close list sections that they don't need.

For two or more levels of hierarchy, basic *trees* are the standby solution. These are normally presented with indented hierarchy levels, and with icons such as pluses and minuses (commonly found on Windows) or rotating triangles. The levels can be closed and opened by the users or automatically by the interface as needed. Many UI toolkits offer tree implementations.

Next, we will take a deeper dive into how the solutions for showing lists actually appear on the web and in mobile UIs.

The Patterns

Now that you have identified what the purpose is for the list you are showing, let's take a look at when and how to use them.

- *Two-Panel Selector or Split View*
- *One-Window Drilldown*
- *List Inlay*
- *Cards*
- *Thumbnail Grid*
- *Carousel*
- *Pagination*
- *Jump to Item*
- *Alpha/Numeric Scroller*
- *New-Item Row*

Two-Panel Selector or Split View

What

Also known as a *Split View*, this consists of two side-by-side panels on the interface. In the first one, show a list of items that the user can select at will; in the second one, show the content of the selected item, as demonstrated in Figure 7-1, which shows the Spotify website.

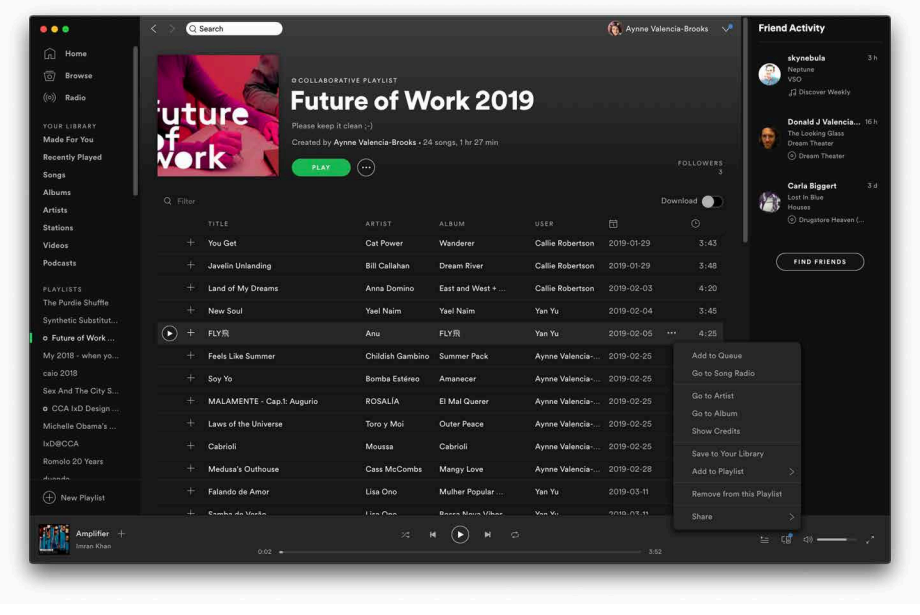


Figure 7-1. *Spotify*

Use when

You have a list of items to show. Each item has interesting content associated with it, such as the text of an email message, a long article, a full-sized image, contained items (if the list is a set of categories or folders), or details about a file's size or date.

You want the user to see the overall structure of the list and keep that list in view all the time, but you also want them to be able to browse through the items easily and quickly. People won't need to see the details or content of more than one item at a time.

Physically, the display you're working with is large enough to show two separate panels at once. Very small smartphone displays cannot cope with this pattern, but many larger mobile devices like tablets can.

Why

This is a learned convention, but it's an extremely common and powerful one. People quickly learn that they're supposed to select an item in one panel to see its contents in the other. They might learn it from their email clients or from websites; whatever the case, they apply the concept to other applications that look similar.

When both panels are visible side by side, users can quickly shift their attention back and forth, looking at the overall structure of the list ("How many more unread email messages do I have?"), and now at an object's details ("What does this email say?"). This tight integration has several advantages over other physical structures, such as two separate windows or *One-Window Drilldown*:

- It reduces physical effort. The user's eyes don't need to travel a long distance between the panels, and they can change the selection with a single mouse click or key press rather than first navigating between windows or screens (which can take an extra mouse click).
- It reduces visual cognitive load. When a window pops to the top, or when a screen's contents are completely changed (as happens with *One-Window Drilldown*), the user suddenly must pay more attention to what their now looking at; when the window stays mostly stable, as in a *Two-Panel Selector*, the user can focus on the smaller area that did change. There is no major "context switch" on the screen.
- It reduces the user's memory burden. Think about the email example again: when the user is looking at just the text of an email message, there's nothing on-screen to remind them of where that message is in the context of their inbox. If they want to know, they must remember, or navigate back to the list. But if the list is already on-screen, the user merely has to look, not remember. The list thus serves as a "You are here" signpost.
- It's faster than loading a new screen for each item, as can happen with *One-Window Drilldown*.

How

Place the selectable list on the top or left panel, and the details panel below it or to its right, as shown in Figure 7-2. This takes advantage of the visual flow that most users who read left-to-right languages will expect (so try reversing it for right-to-left language readers).

When the user selects an item, immediately show its contents or details in the second panel. Selection should be done with a single click. But while you're at it, give the user a way to change their selection from the keyboard, particularly with the arrow keys—this helps reduce both the physical effort and the time required for browsing, and contributes to keyboard-only usability.

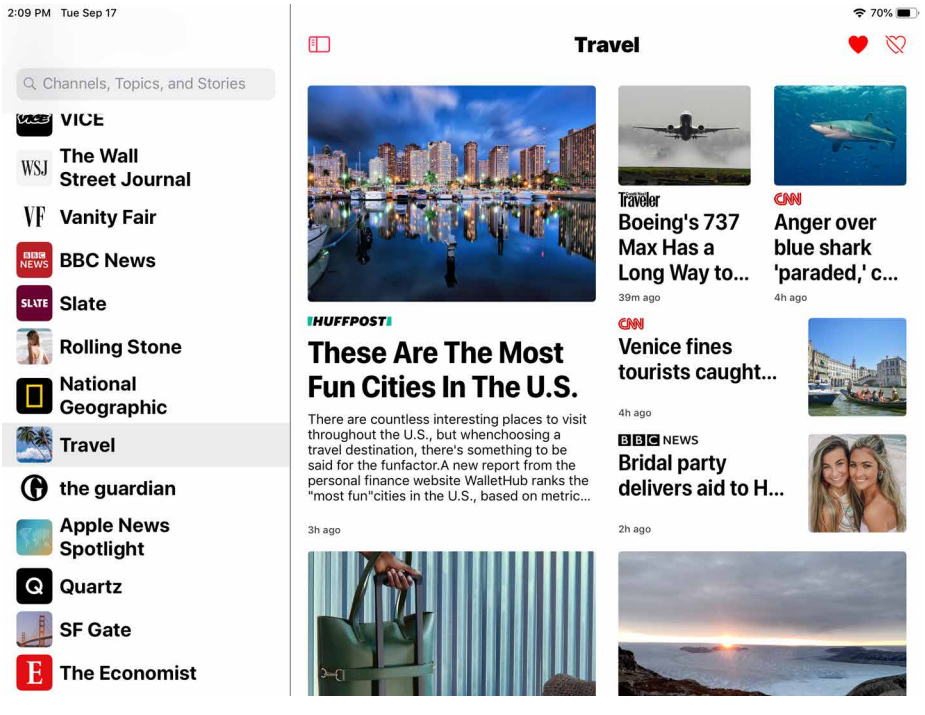


Figure 7-2. *Apple iOS News*

Make the selected item visually obvious. Most toolkits have a particular way of showing selection (e.g., reversing the foreground and background of the selected list item). If that doesn't look good, or if you're not using a toolkit with this feature, try to make the selected item a different color and brightness than the unselected ones—that helps it stand out.

What should the selectable list look like? It depends—on the inherent structure of the content, or perhaps on the task to be performed. For instance, most filesystem

viewers show the directory hierarchy, since that's how filesystems are structured. Animation and video editing software use interactive timelines. A graphical user interface (GUI) builder may simply use the layout canvas itself; selected objects on it then show their properties in a property editor next to the canvas.

A *Two-Panel Selector* has identical semantics to tabs: one area for the selectors, and one area next to it for the content of the selected thing. Likewise, a *List Inlay* is like an *Accordion*, and *One-Window Drilldown* is like a *Menu Screen*.

Examples

Many email clients (Figure 7-3) use this pattern to show a list of email messages next to the currently selected message. Such listings benefit from being nearly as wide as the entire window, so it makes sense to put the listing on top of the second panel, not to its left.

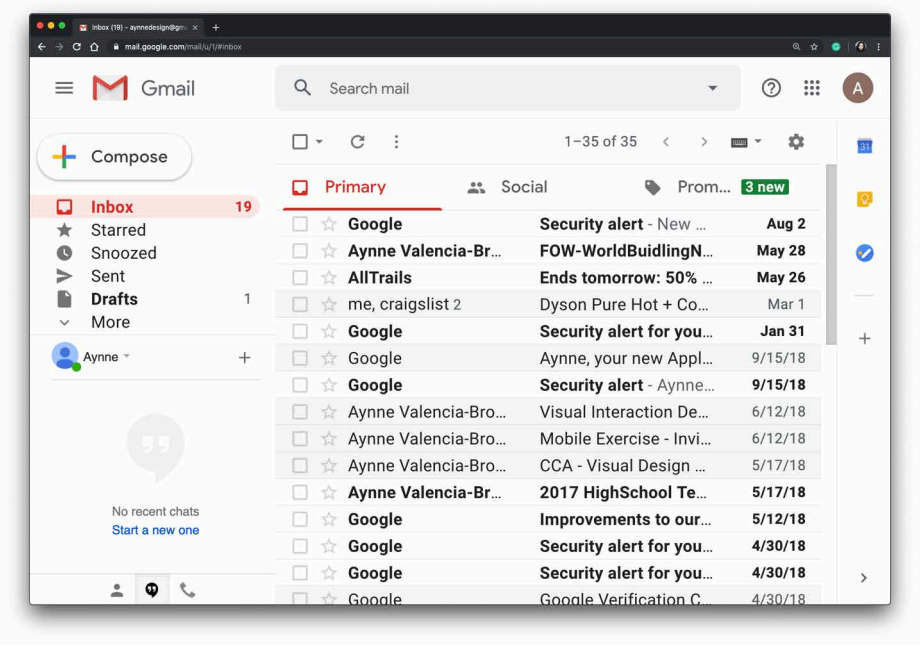


Figure 7-3. *Google Mail*

Apple Photos (Figure 7-4) lists the various image folders and categories in its *Two-Panel Selector*. The result is a second list, of images. When the user selects an image, the whole window is replaced; see *One-Window Drilldown*.

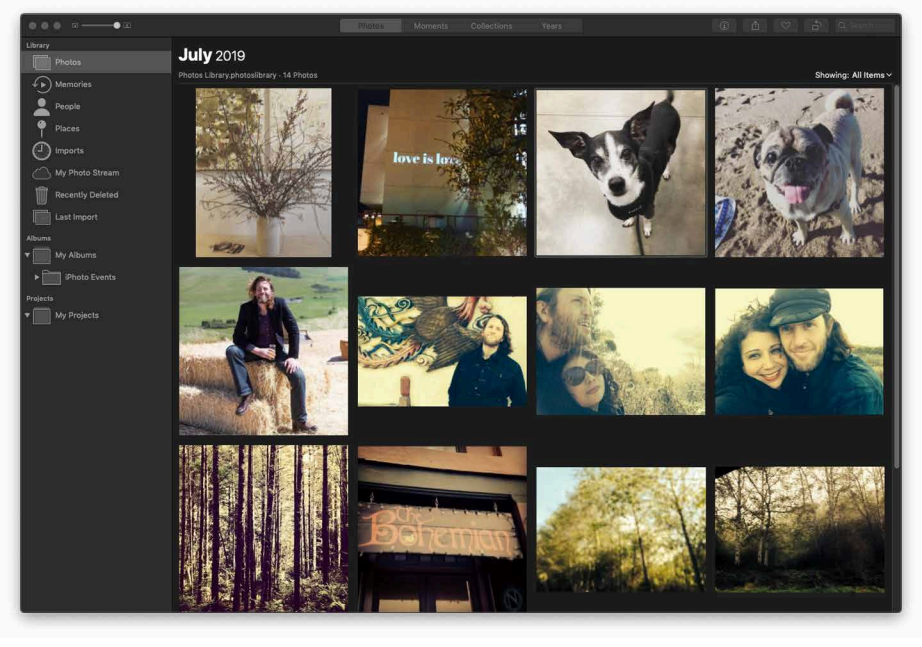


Figure 7-4. *Apple Photos*

One-Window Drilldown

What

Show a list of items in a single screen or window. When the user selects an item from the list, show the details or contents of that item in the screen, replacing the list, as in the example in Figure 7-5.

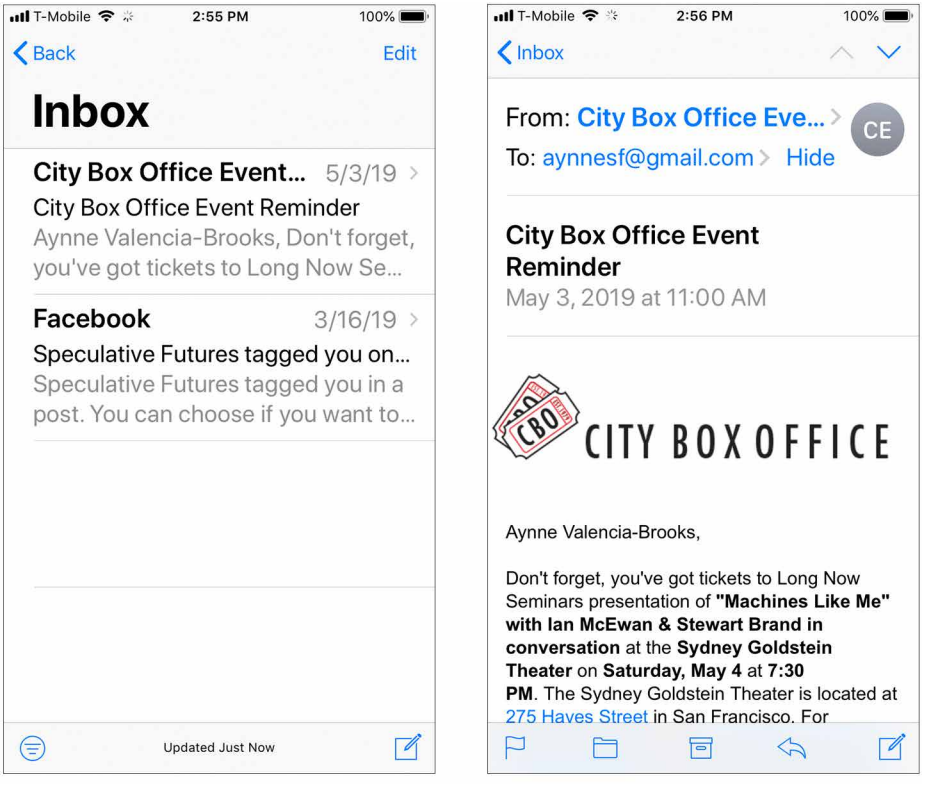


Figure 7-5. *Mac Mail on iPhone*

Use when

You are designing for mobile applications or websites. You have a list of items to show. Each item has interesting content associated with it, such as the text of an email message, a long article, a full-size image, or details about a file's size or date.

Alternatively, the list items and contents might just be large. You might need the entire screen or window to show the list, and again to show the contents of an item. Online forums tend to work this way, requiring the whole width of the screen to list conversation topics and a separate scrolled screen to show the conversations themselves.

Why

In a very constrained space, this might be the only reasonable option for presenting a list and item details. It gives each view the entire available space to "spread out" on the screen.

The shallow hierarchy of the one window drilldown pattern helps your user to not get too deep in the UI and makes it easy to return to the list where they started.

How

Create the list using whatever layout or format you find best—simple text names, cards, rows, trees, or outlines all work fine with *Thumbnail Grid*, as do other formats. Vertically scroll it if necessary, to fit it into the available space.

When the user clicks, taps, or otherwise selects one of the list items, replace the list display with a display of the item details or contents. On it, place a Back or Cancel button that brings the user back to the list screen (unless the platform supplies hardware buttons for such).

The item screen can offer additional navigational possibilities, such as drilling down further into the item details, stepping down into an item contained within that item (as in a hierarchy), or going "sideways" to the previous or next item in the list (as discussed in the next paragraph). In each case, replace the previous screen with the new one, and make sure the user can easily step back to the previous screen.

One disadvantage of this pattern is that to go from item to item, the user must "pogo-stick" between the list screen and the item screen. It takes a lot of clicks or taps to see more than a few items, and the user certainly can't flick between them quickly (as with *Two-Panel Selector*) or compare them easily (as with *List Inlay*). You can mitigate this problem by using Back and Next links to connect the user directly to the previous and next items in the list.

Examples

Examples abound in mobile design. Contrast the mobile version of a mail client shown in Figure 7-5 with its desktop counterpart. For instance, the *One-Window Drilldown* approach requires more text to be shown in the list, so the user has enough context to identify messages and triage them.

In the example in Figure 7-6 a reader can scroll through all the comments on a post in Reddit, and by clicking the back arrow in the header, easily return back to the Topic to view other threads.

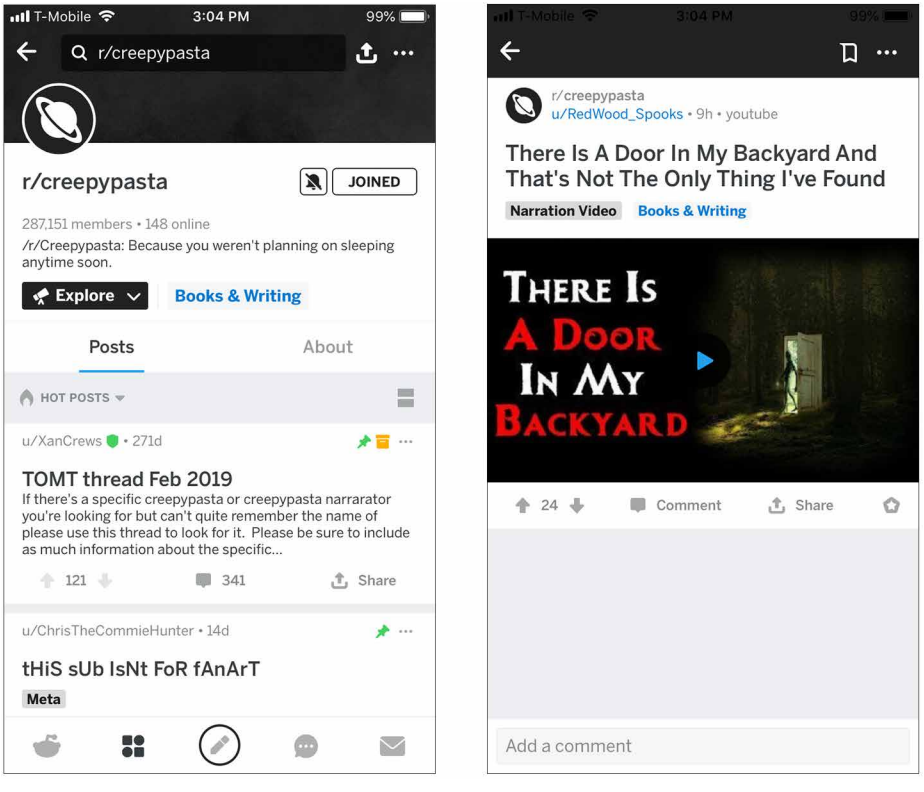


Figure 7-6. *Reddit*

List Inlay

What

Show a list of items as rows in a column. When the user selects an item, open that item's details in place, within the list itself (Figure 7-7). Allow items to be opened and closed independently of each other.

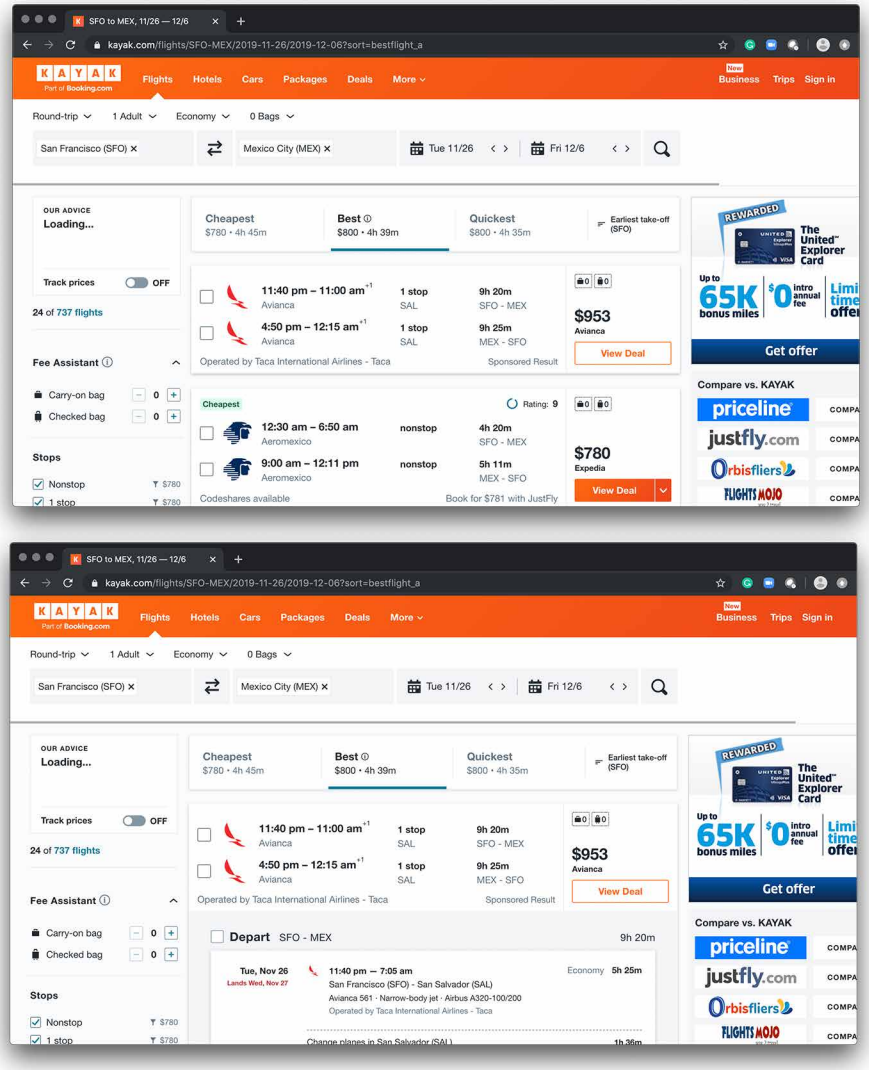


Figure 7-7. *Kayak's expanding list items*

Use when

You have a list of items to show. Each item has interesting content associated with it, such as the text of an email message, a long article, a full-size image, or details about a file's size or date. The item details don't take up a large amount of space, but they're not so small that you can fit them all in the list itself.

You want the user to see the overall structure of the list and keep that list in view all the time, but you also want them to browse through the items easily and quickly. Users might want to see two or more item contents at a time, for comparison.

The list of items has a vertically oriented, columnar structure, rather than a grid.

Why

A *List Inlay* shows an item's details within the context of the list itself. The user can see the surrounding items, which might help in understanding and using the item contents.

Also, a user can see the details of multiple items at once. This is not possible in *Two-Panel Selector*, *One-Window Drilldown*, rollover windows, or most other ways of displaying item details. If your use cases call for frequent comparison of two or more items, this might be the best option.

Because a *List Inlay* is neatly contained within a vertical column, it can be combined well with a *Two-Panel Selector* to present a three-level containment hierarchy. Consider an email client or RSS reader, for instance—the messages or articles might be viewed in a *List Inlay*, whereas the item containers (mailboxes, groupings, filters, etc.) are shown next to it in a *Two-Panel Selector* structure.

How

Show list items in a column. When the user clicks one, open the item in place to show the details of that item. A similar gesture should close the item again.

When an item is opened, enlarge the item's space downward, pushing the subsequent items down the screen. Other items do the same when opened. A scrolled area should be used to contain this ever-changing vertical structure because it could become very tall indeed!

To close the details panel, use a control that clearly indicates its purpose (e.g., "Close" or "X"). Some implementations of *List Inlay* only put that control at the end of the details panel, but users might need it at the top of the panel if it is long and they don't want to move all the way down to the bottom. Put a closing control very near the original "open" control (or replace one with the other). This at least ensures that the

user's pointer won't move very far if they want to open an item, glance at it, close it, and move on.

Use an *Animated Transition* as the item opens and closes to keep the user oriented and to focus attention on the newly opened item. If your application permits the user to edit items, you could use a *List Inlay* to open an editor instead of item details (or in addition to them).

A list that uses *List Inlay* works the same way as an *Accordion*: everything lies in a single column, with panels opening and closing *in situ* within it. Likewise, a *Two-Panel Selector* works like a set of tabs, and *One-Window Drilldown* is like a *Menu Screen*.

Examples

Apple iOS Voice Memos (Figure 7-8) uses a *List Inlay* to hide functionality and only shows the playback and recording controls when a user taps on the recording in the list.

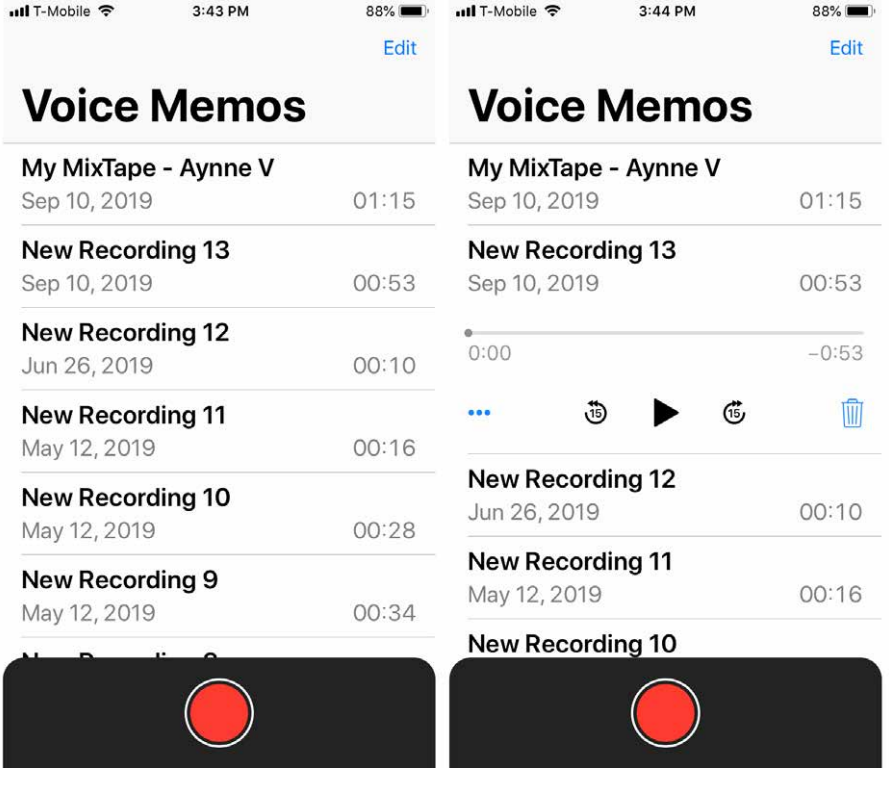


Figure 7-8. *Apple iOS Voice Memos*

The Transit app (Figure 7-9) demonstrates a unique hybrid use of a *List Inlay* behavior combined with a modal window. When a user sees routes nearest them, they can tap the route and the list expands not only to see the times the next bus or train arrives, but it also reveals the specific stops they can expect along the route.

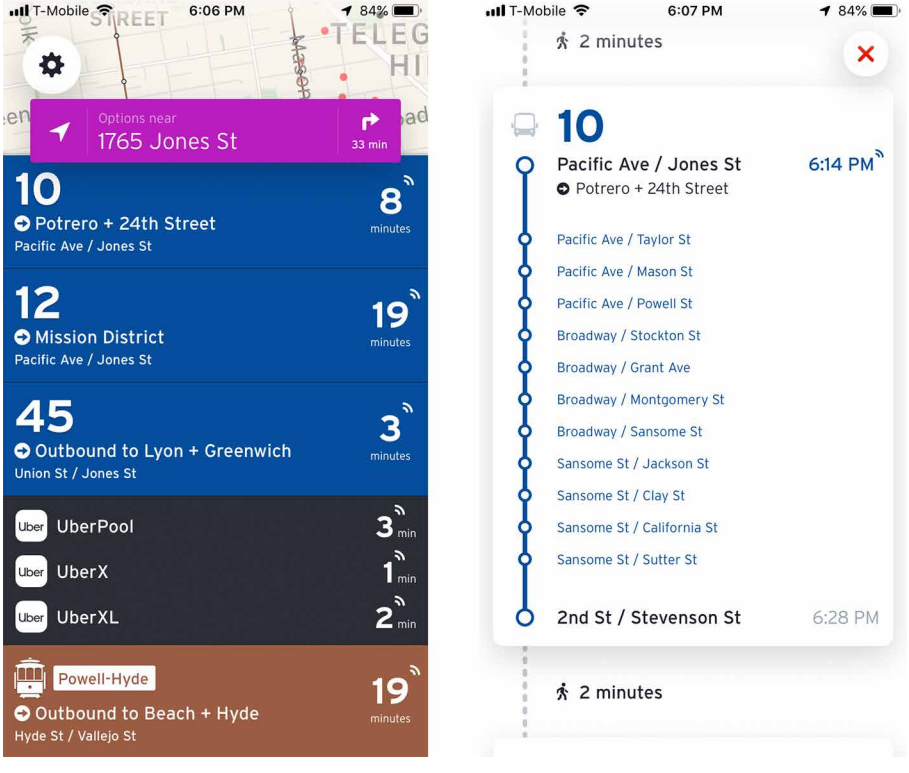


Figure 7-9. *Transit*

Further reading

Bill Scott and Theresa Neil identified this technique in their book, *Designing Web Interfaces* (O'Reilly, 2009). *List Inlay* is one of a set of inlay techniques that includes Dialog Inlays and Detail Inlays.

Cards

What

Cards are one of the most popular and flexible UI components in use. *Cards* are self-contained components that have images, text, and sometimes actions. They were given their name because they resemble playing cards. You can find examples of these nifty components in newer websites and mobile applications that were designed to be responsive (resizable) and use popular UI component libraries. Pinterest's use of cards is shown in Figure 7-10.

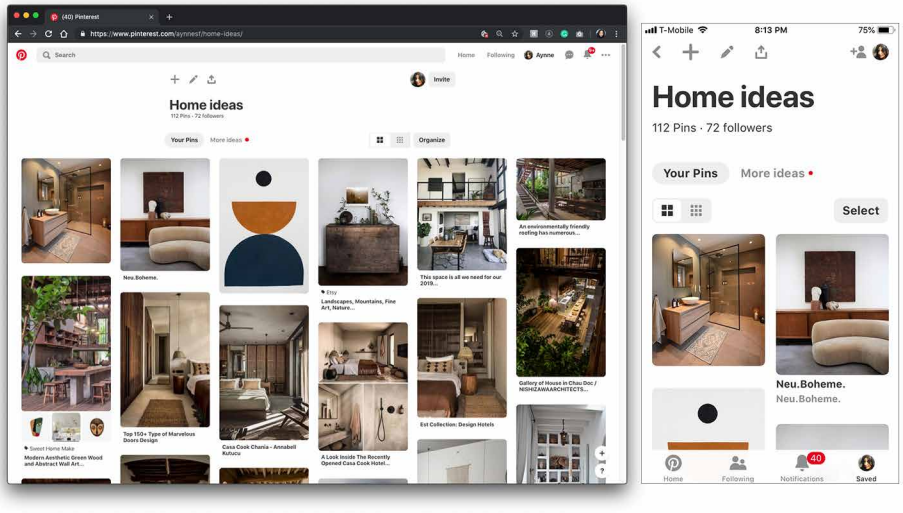


Figure 7-10. *Pinterest website and mobile app*

Use when

You are showing a list of heterogeneous items and they will all have the same behaviors associated with them. For example, all of the items will contain an image, some text, a way to "favorite" or share the item, and a link to a detail screen.

You need to show a collection of items, but each item might differ in size and/or aspect ratio.

Why

Cards are used frequently in mobile and web design, so a user will likely have seen this UI convention before. Cards allow a great deal of flexibility in the layout or size of the viewport (screen type) your user will be viewing them on.

How

Think about the things you want a user to do with the list of items you are showing. What are their commonalities? Do they all have a picture, a title, and a short description? Perhaps they will show a rating? What do you want the user to do? Do you want them to link to a detail screen? Add the item to a cart? Share the item to social media? Will all of the items do the same thing? If so, a card will be likely be the best way to show your list.

A good practice when designing is to take the item with the most or longest amount of content and mock it up. Do the same for the one with the least or smallest amount of content. Tweak your layouts until your design looks good and the information is readable and clear at both sizes.

Consider which actions will be icons and which will be text links. Use real examples of the photos you will be using to determine the best orientation (portrait or landscape) that works for the type of content you will be showing.

Examples

Airbnb (Figure 7-11) uses cards all over its website and mobile application to display homes and experiences listings. Using this style for its listings ensures a consistent look and feel and also offers a visually pleasing way to view the information without giving more visual weight to one listing over another.

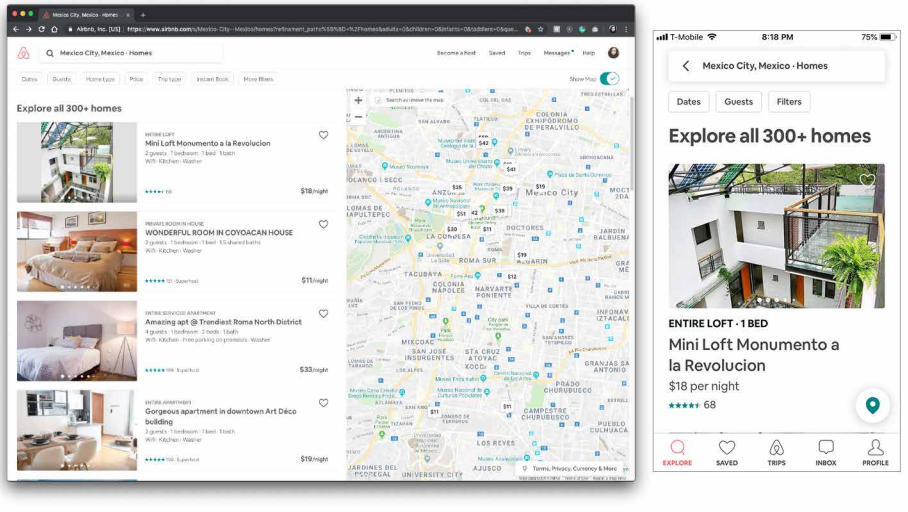


Figure 7-11. *Airbnb website and mobile App*

Uber Eats (Figure 7-12) uses cards without a border to show a photo, the restaurant name, and user ratings.

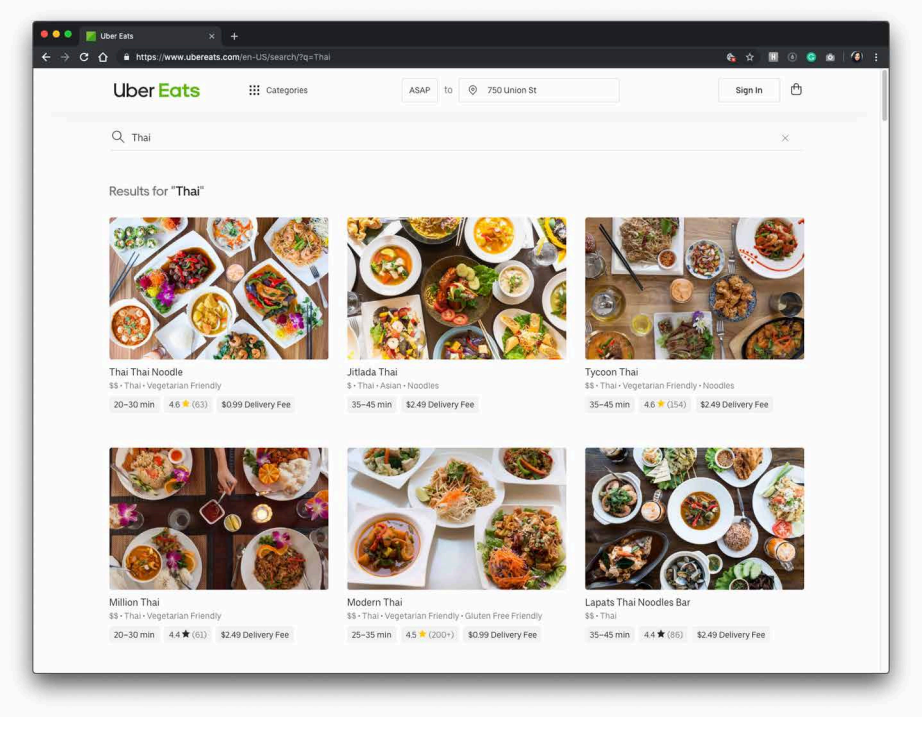


Figure 7-12. *Uber Eats search results*

Further reading

For more information on cards, check out the following:

- *https://material.io/design/components/cards.html*
- *https://getbootstrap.com/docs/4.0/components/card*

Thumbnail Grid

What

Arrange a list of visual items into a "small multiples" grid of photos, as illustrated in Figure 7-13.

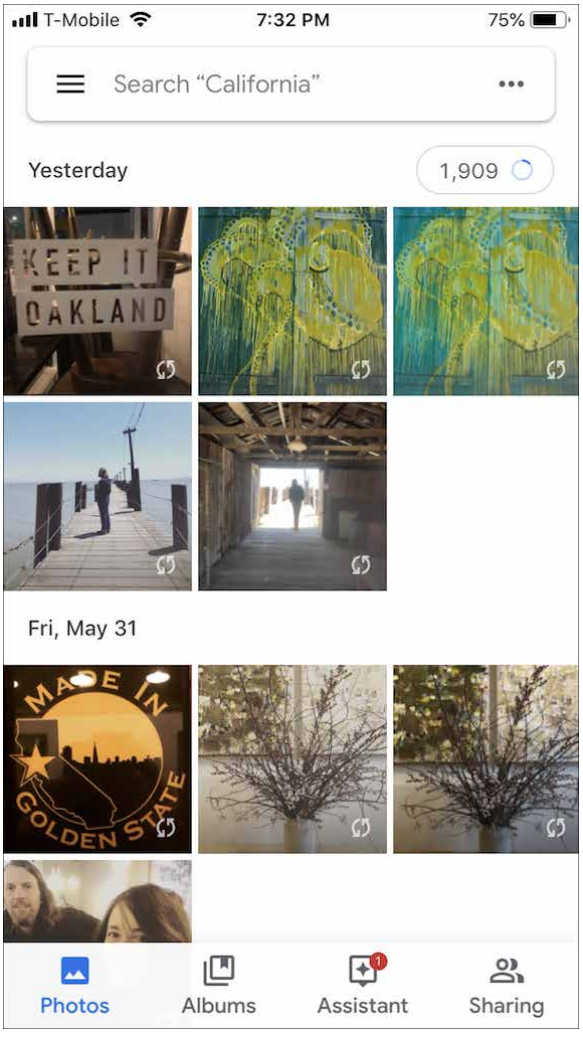


Figure 7-13. *Google Photos app*

Use when

The list items have small visual representations that uniquely identify them: images, logos, screen captures, reduced photos, and so forth. These tend to be similar in size and style. The list can be long, and it can be divided into *Titled Sections*.

You want to show a little bit of metadata (information about the item) with each one, such as its name and date, but you don't need to show a lot of that—the picture should take up most of the space devoted to the item.

Users will want an overview of the entire list, and they might need to scan it quickly to find a particular item of interest. Users might also need to select one or more items at a time for moving, deleting, or viewing.

Why

A *Thumbnail Grid* is a dense, attractive presentation of large numbers of items. A specialized instance of *Grid of Equals*, this pattern creates a visual hierarchy that shows the list items as peers, and a strong grid tends to draw the eye to that part of the screen.

It might be easier to show the list items in text form, but sometimes pictures can be recognized and differentiated more easily than text.

Thumbnails that are roughly square make easy targets for fingertips (on touch screens) and for indirect pointing devices, as well. This pattern works well on mobile and tablet screens.

How

Scale the thumbnails so that they're the same size, to keep the grid tidy. Place the text metadata close to the thumbnail, but in small print in order to maintain the thumbnail visual prominence.

Some *Thumbnail Grids* look much nicer when the thumbnails all have similar width and height. If you're working with graphics that come in different sizes or aspect ratios (the ratio of width to height), or if they're large, some image processing will need to be done to construct thumbnails. Try to find a size and aspect ratio that works reasonably well with all of them, even if some images will be cropped to fit it. (Reducing image size is easy; cropping appropriately is not. Be careful to preserve the image's integrity by choosing the most relevant piece of the image to show when possible.)

An exception is if you're dealing with images whose size and proportion are useful information to the viewer. For instance, a set of personal photos will contain some that are in a landscape format and some in a portrait (vertical) format. There's no

need to crop these to match an idealized thumbnail—the user will want to see which photos are which!

On the other hand, a thumbnail gallery of products (such as shoes or shirts) should all have the same height and width, with the products presented consistently within those photos.

Examples

The macOS Finder (Figure 7-14) displays a variety of thumbnail types for a file directory listing. When a file is an image, a shrunken version of that image is shown; for directories, a simple folder; for files without an available visual, just the file type (e.g., "DOC") over a generic icon. The thumbnail grid is not at all uniform, so it doesn't look as clean as the others in this pattern, but the size and style variations communicate useful information to the user.

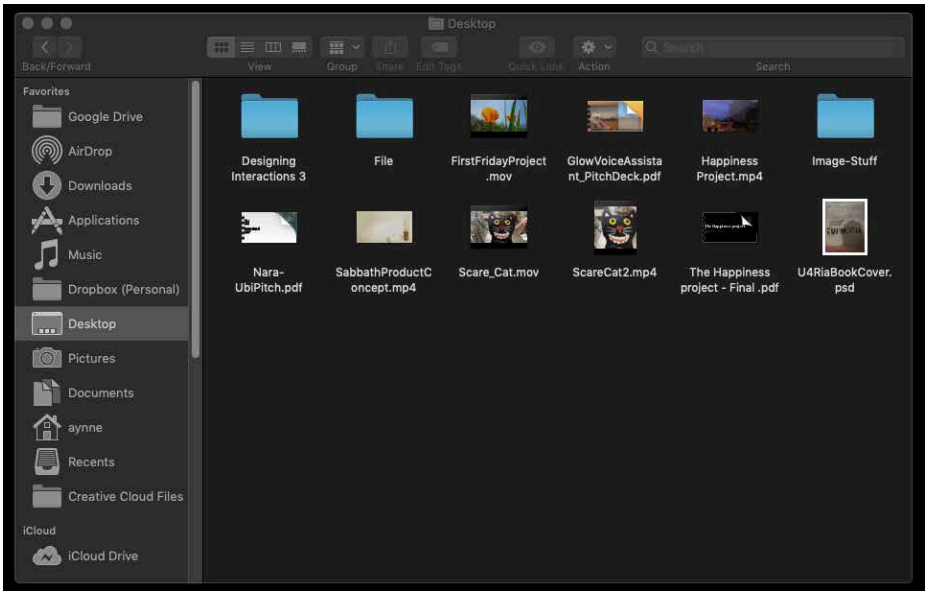


Figure 7-14. *The macOS Finder*

The video application, HBO Now, shown in Figure 7-15, features a photo and title of the movie or TV show in a thumbnail grid.

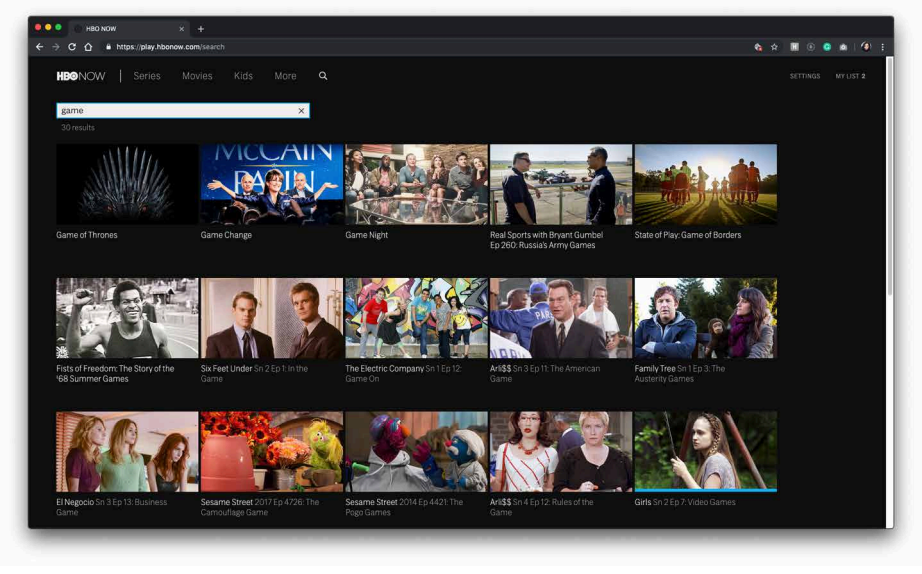


Figure 7-15. *HBO Now search results*

Mobile devices need *Thumbnail Grid* in many contexts: to show applications, features, and images themselves. Note the relative sizes of the thumbnails in Figure 7-16; the Google Images and iPhone home screen examples are just big enough to be touched easily by human fingertips.

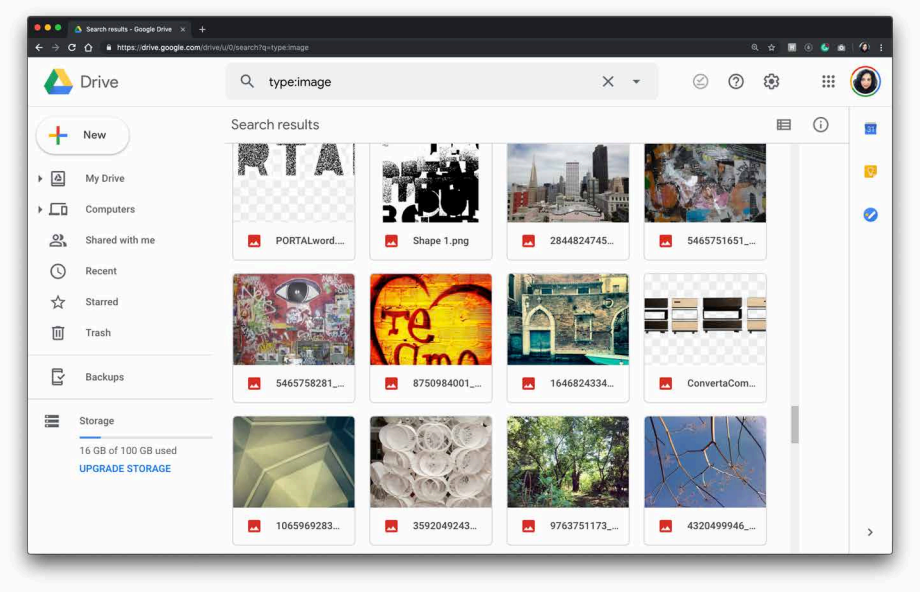


Figure 7-16. *Google Drive*

Further reading

For more information on thumbnail lists, check out the following:

- *https://developer.apple.com/design/human-interface-guidelines/ios/views/collections*
- *https://material.io/design/components/image-lists.html#types*

Carousel

What

Arrange a list of visually interesting items arranged into a horizontal strip or arc, as demonstrated in Figure 7-17. This lets the user scroll or swipe the image thumbnails back and forth to view them.

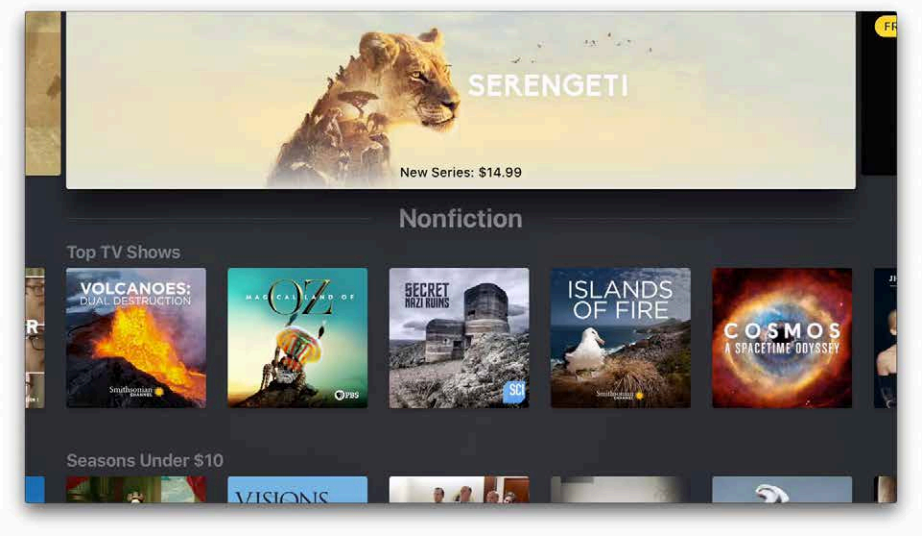


Figure 7-17. *Apple TV carousel*

Use when

The list items have visual representations that uniquely identify them: images, logos, screen captures, reduced photos, and so forth. These tend to be similar in size and style. The list is *flat* (i.e., not divided into categories or containers).

You want to show a little bit of metadata (information about the item) with each one, such as its name and date, but you don't need to show a lot of that—the picture should take up most of the space devoted to the item.

Each item is potentially of interest. Users will browse the items casually; they won't normally search for a specific item, or need to get an overall look at the entire list at once. If someone does look for a specific item, they won't mind moving past many items before finding the one they're looking for. You might be able to order the items with the most interesting ones first, or in chronological order.

You don't have enough vertical space for a *Thumbnail Grid*, and you may not have a lot of horizontal space either, but you need to make this list look interesting and attractive.

Why

A *Carousel* offers an engaging interface for browsing visual items, encouraging the user to inspect the items that are in view and to see what's next. A user can't easily jump to a certain point deep in the list, so they must scroll through everything—this pattern thus encourages browsing and serendipity.

Carousels are compact vertically, so they can be a better solution than a *Thumbnail Grid* for a small space. Horizontally, they can be either compact or spread out.

If a particular implementation focuses attention on a central item or selection, such as by enlarging it, this pattern delivers "focus plus context"—users get a detailed view of one item while also seeing the ones immediately around it.

How

First, create thumbnails for each item shown in the *Carousel*. See the *Thumbnail Grid* pattern for issues related to thumbnail size and proportion (keeping in mind that *Carousel* imposes even stricter restraints—thumbnails of different sizes or aspect ratio tend to look more awkward in *Carousel* than in *Thumbnail Grid*). Place the text metadata close to the thumbnail, but in small print in order to maintain the thumbnail's visual prominence.

In a horizontal scrolling widget, arrange the thumbnails horizontally, either randomly or in an order that makes obvious sense to the user (such as by date). Show a small number of them—fewer than 10—and hide the rest on either side. Put large arrows on the left and right for paging through the *Carousel*; each click on an arrow should move more than one item. Animate this scrolling for extra visual interest.

If users will want to move quickly through a long list, as though they are looking for something in particular, put a scroll bar below the *Carousel*in addition to the arrows. You might find that users do this a lot; if so, consider restructuring the list as a more conventional vertical list, and add a "find" capability.

You can choose to enlarge the central item in the *Carousel* to draw attention to it. This gives the *Carousel* single-selection semantics—the enlarged item is clearly the selected one, and you can then do dynamic things based on that selection, such as showing text details about it, or offering video controls if the items are video thumbnails.

Some *Carousels* are straight; some are curved or circular. These usually use the trick of a 3D perspective, in which items shrink and are partially obscured as they drift farther away from the center.

In the mobile design space, the *Filmstrip* pattern is a variant on a *Carousel*. Only one item at a time is shown on the small screen, and the user swipes or scrolls back and forth to see other items.

Examples

Many websites use a basic, linear *Carousel* for browsing products. Amazon shows book covers this way (Figure 7-18); note the different amounts of text metadata and the implications for design. How much information should be provided with each book? How tightly packed should the book covers be?

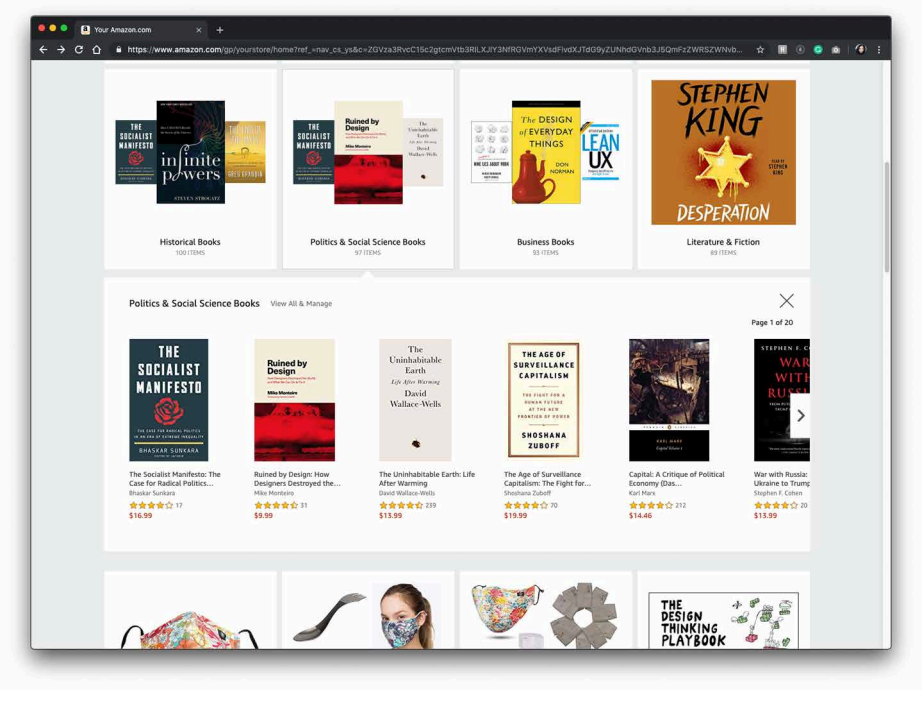


Figure 7-18. *Amazon Books*

Both Amazon and Airbnb (Figure 7-19) provide arrows with their *Carousels*. This helps the user understand that there is more content beyond the bounds of the window.

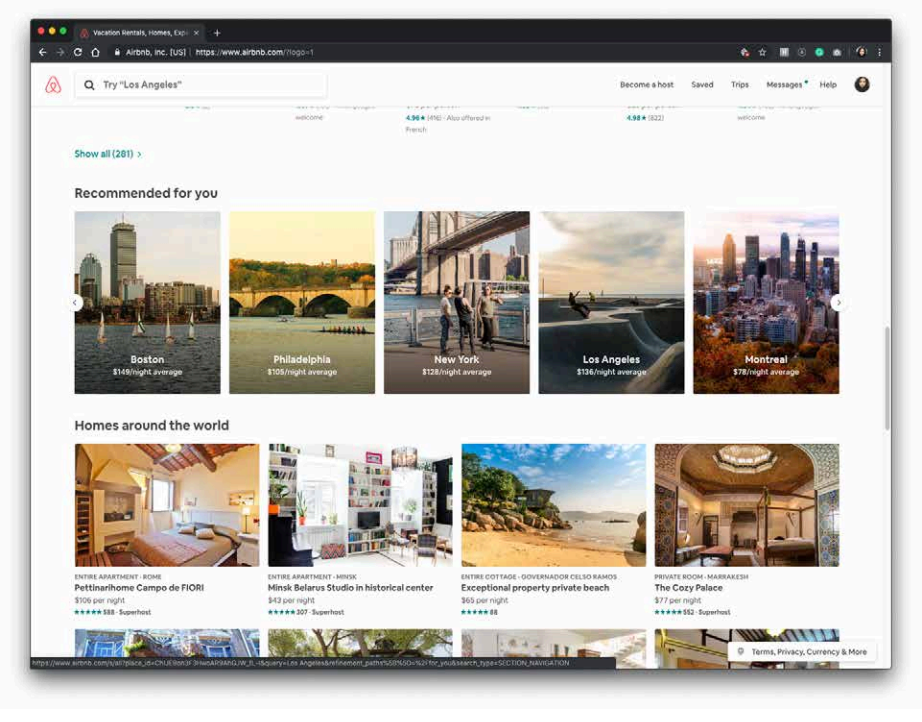


Figure 7-19. *Airbnb*

Pagination

What

Break up a very long list into pages, and load them one page at a time. Provide controls to navigate the list, as shown in Figure 7-20.

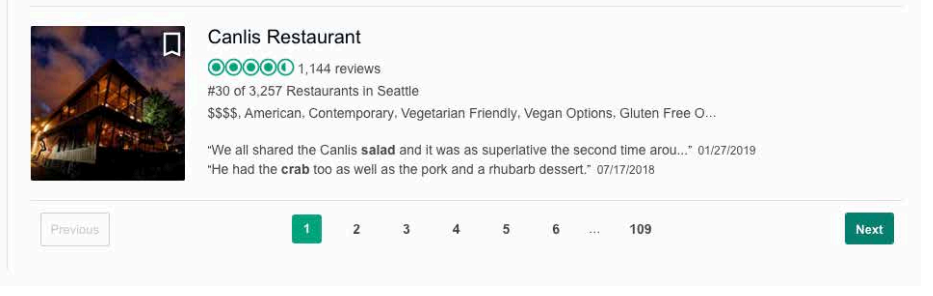


Figure 7-20. *TripAdvisor pagination control*

Use when

You're showing a list that might be very, very long. Most users will either look for a particular item or browse the top of the list for relevant items (e.g., with search results); in any case, they won't really want to see the entire list.

The technology you're using doesn't support loading the entire list into a single screen or scrolled are, for any of the following reasons:

- Loading the entire list would take too much time, and you don't want to make the user wait. This might be the case over a slow internet connection or with a slow backend server.
- Rendering the list would take too much time.
- The list is effectively "bottomless," and implementing an *Infinite List* or a continuously scrolling list (which both handle bottomless lists) isn't feasible for some reason.

Why

"Pagination" breaks a list into chunks that a user can easily take in without being overwhelmed. Furthermore, it puts the choice to see more into the user's hands—do you want to load more items from the list, or is this screen of items enough for you?

This pattern also has the advantage of being very common on the web, especially (though not exclusively) for search results. It's easy to implement and can come prebuilt in some systems.

How

First, you'll need to decide how many items will be in each screen. Base this on the amount of space each item takes up, the screen sizes users are likely to have (don't forget to consider mobile platforms), the time it takes to load or show the items, and the likelihood that the user will find one or more desired items in the first screen.

This is fairly important: the first screen should be enough! The odds are good that most users won't go beyond that first screen of items, so if they can't find what they're looking for on the first screen, they might become discouraged. (If you're dealing with a search facility, make sure that it returns high-quality results at the top of that first screen.)

For screens on which users might linger, such as lists of products or videos, consider letting the user set the number of items per screen. Some people are irritated by having to screen back and forth to see all the items of interest.

Next, you'll need to decide how to present the pagination controls. They're usually found at the bottom of the screen, but some designs also have them at the top—if a user really does need to go to a subsequent screen of items, there's no need to make them scroll all the way down the screen.

Consider these elements in the pagination control:

- Previous and Next links, with arrows or triangles for emphasis. Disable the Previous link when the user is on the first screen and the Next link when the user is on the last screen (if there is a known last screen).
- A link to the first screen. This should always be visible; remember that the first screen is supposed to contain the most relevant items.
- A sequence of numbered links to screens. Don't link the screen the user is on, of course; instead, show it in a contrasting color and type size to give the user a "You are here" navigational clue.
- Ellipses to cut out parts of the sequence if there are too many screens to reasonably show in the control—more than 20, for instance. Again, keep the first screen, and the last screen if the list isn't "bottomless." Keep the screens immediately before and after the user's current screen. Elide the rest.
- Optionally, the total number of screens (if known). You could do this in several ways, such as showing text like "screen 2 out of 45," or simply showing the last screen as a numbered link at the end of the pagination control.

Examples

Etsy does an excellent job of including all the elements and cues from the preceding list. Figure 7-21 shows the first screen for a very large number of items.

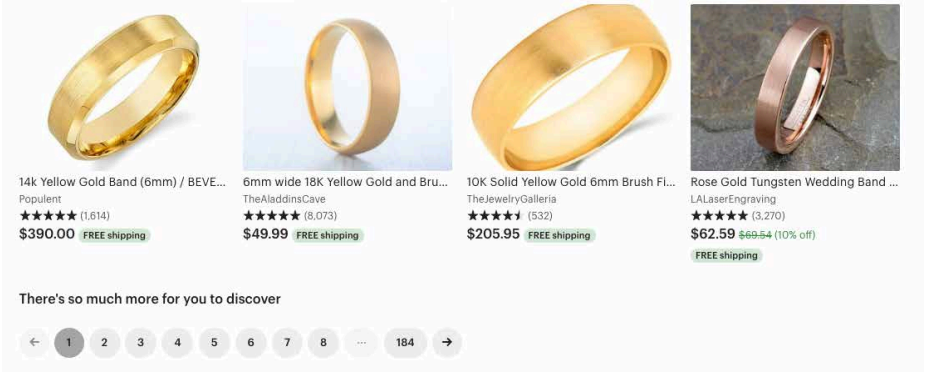


Figure 7-21. *Etsy search results pagination control*

Figure 7-22 shows a gallery of examples from all over the web. Notice which ones are easier to parse visually—Which link is which? Where should I click next?—and which ones give you sufficient information about your location and the total number of screens. Also note the size of the click targets. How accurate does the user need to be with their mouse or fingertip?

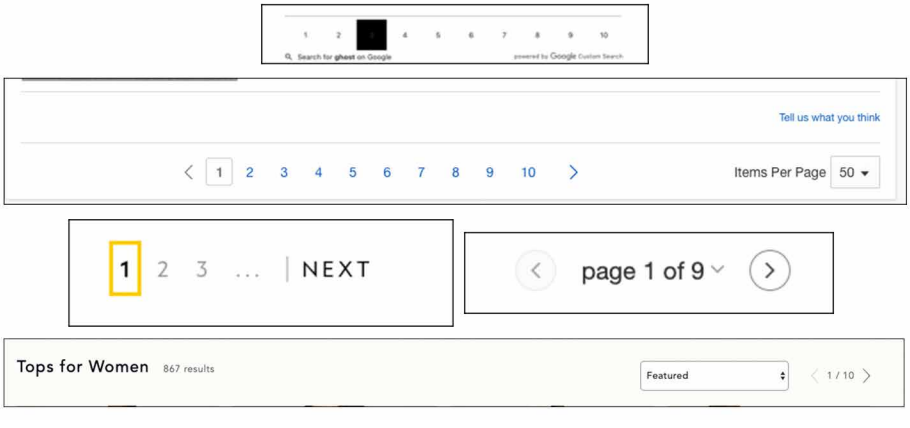


Figure 7-22. *Counterclockwise from top: The Atlantic, eBay, National Geographic, Target, Anthropologie*

Jump to Item

What

When the user types the name of an item into a list, jump straight to that item, as depicted in Figure 7-23.

Figure 7-23. *Font Book app*

Use when

The interface uses a scrolling list, table, drop-down box, combo box, or tree to present a long list of items. These items are sorted, either alphabetically or numerically. The user wants to select one particular item quickly and accurately, and preferably with the keyboard.

This pattern is often used in file finders, long lists of names, and drop-down boxes for state or country selection. You can also use it for numbers—such as years or dollar amounts—or even calendar time, such as months or days of the week.

Why

People aren't good at scanning down long lists of words or numbers for a particular item. But computers are. Let them do what they're good at!

Another nice thing about this technique is that it lets the user keep their hands on the keyboard. As the user moves through a form or dialog box, they might find it nice to select from a list simply by typing the first few characters of the item they want—the system then picks the item for the user, and they can continue on to the next thing. No scrolling or clicking is necessary; the user's hand never needs to move from the keyboard to the mouse.

How

When the user types the first letter or number of the item they're looking for, jump to the first item that matches what the user typed: automatically scroll the list so that the item is visible and then select it.

As the user types more characters in rapid succession, keep changing the selection to the first exact match for the user-typed string to that point. If there is no match, stay put at the nearest match, and don't scroll back to the top of the list. You might want to beep at the user to inform them that there's no match—some applications do, some don't.

Examples

A variant of *Jump to Item* is used by Spotify's search functionality. A user can start typing and the search results will begin to show different results as each letter allows the system to better predict what songs or artists the user is looking for.

In the example in Figure 7-24 in the upper left, I have typed "Brian" and the results show Spotify's algorithms best guess at what I am searching for. I don't even have to type the second word before the search results display the artist I was searching for. In some ways, this incremental search is more convenient—and certainly faster.

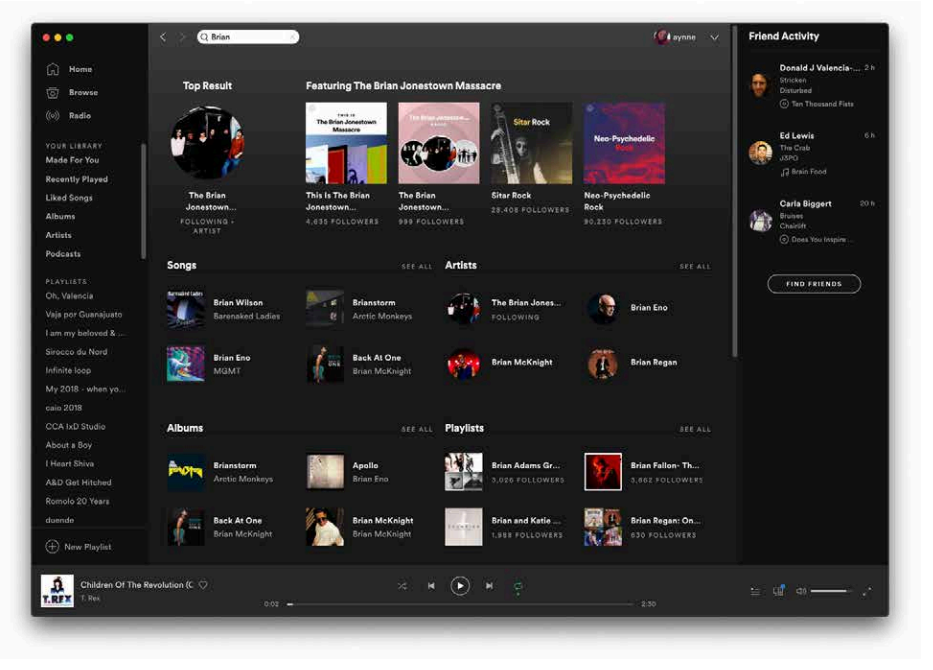


Figure 7-24. *Spotify's variant of the Jump to Item pattern*

Alpha/Numeric Scroller

What

Display the letters of the alphabet, numbers, or a timeline arrayed along the scroll bar of a list, as shown in Figure 7-25 (left).

Use when

Users will be searching for very specific items in a long list, which is usually rendered as a scrolled list, table, or tree. You want to make item finding as easy and quick to achieve as possible.

Why

Alpha/Numeric Scrollers are not common, but their use is self-explanatory. They provide an interactive map to the list content, in much the same way as an *Annotated Scroll Bar*. They're closely related to *Jump to Item*—both enable immediate jumping to a point in an ordered list.

This pattern probably arose from physical books (such as dictionaries) and notebooks (such as address books) that use tabs to mark points in the alphabet.

How

Place a long list into a scrolled area. Along the scroll bar, show the letters of the alphabet or date; when the user clicks a letter, scroll the list to that point.

Examples

The iPhone offers what is probably the best-known example of this pattern. Figure 7-25 (right) shows its built-in Contacts app.

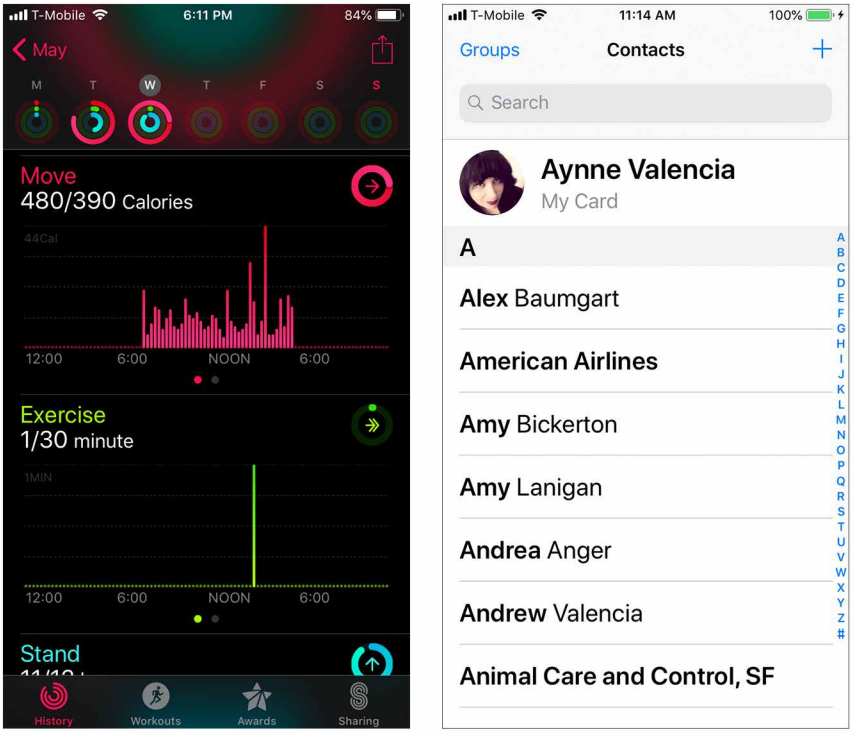


Figure 7-25. *Apple iOS Health app Date Scroller control and the iPhone Contacts list*

New-Item Row

What

Use the last or first row in the list or table to create a new item in place, as in the fourth list item in Figure 7-26.

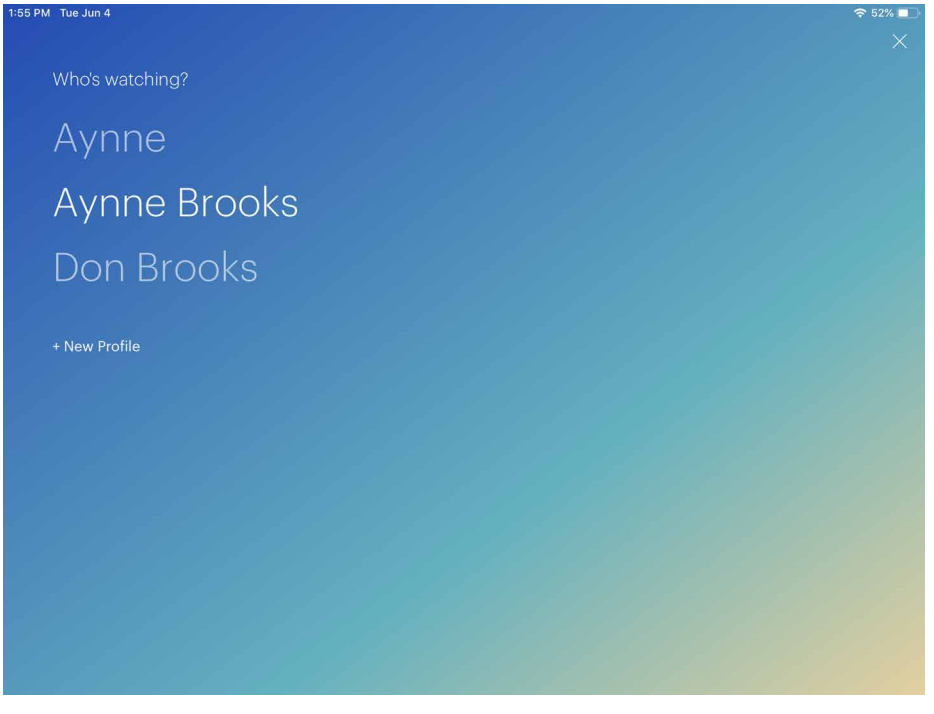


Figure 7-26. *Hulu, Profile Switcher*

Use when

The interface contains a table, list, tree view, or any other vertical presentation of a set of items (one item per row). At some point, the user needs to add new items to it. But you don't have a lot of room to spare on the UI for extra buttons or options, and you want to make item creation very efficient and easy for the user.

Also use when you want to be explicit about what type of information or functionality the user is adding.

Why

By letting the user type directly into the end (or the beginning) of the table, you put the act of creation into the same place where the item will ultimately "live." It's conceptually more coherent than putting it in some other part of the UI. Also, it makes the interface more elegant than having an entirely different UI for item creation—it uses less screen real estate, it reduces the amount of navigation that needs to be done (thus eliminating a "jump" to another window), and it's less work for your users.

How

Give the user an easy and obvious way to initiate a new object from the first empty table row. A single mouse click in that row might start editing, for instance, or the row might contain a "New Whatever" button.

At that point, the UI should create the new item and put it in that row. Each column in the table (if it's a multicolumn table) should then be editable, thus letting the user set up the values of that item. The cells could have text fields in them, or drop-down lists, or whatever else is necessary to set the values quickly and precisely. As with any form-like user input, *Good Defaults* (Chapter 10) help save the user work by prefilling those values; the user doesn't have to edit every column.

There are still some loose ends to clean up, though. What happens if the user abandons the new item before finishing? You can establish a valid item right from the beginning—if the user abandons the edits at any time, the item exists until the user goes back and deletes it. Again, *Good Defaults* help by prefilling valid values if there are multiple fields.

Depending on how it's implemented, this pattern can resemble *Input Prompt* (also Chapter 10). In both cases, a dummy value is set up for the user to edit into a real value, and that dummy value is worded as a "prompt" that shows the user what to do.

Examples

You can find the new item row pattern most frequently in applications where the user organizes or created their own content.

Slack (Figure 7-27) uses this same pattern in its "Add Workspaces" function.

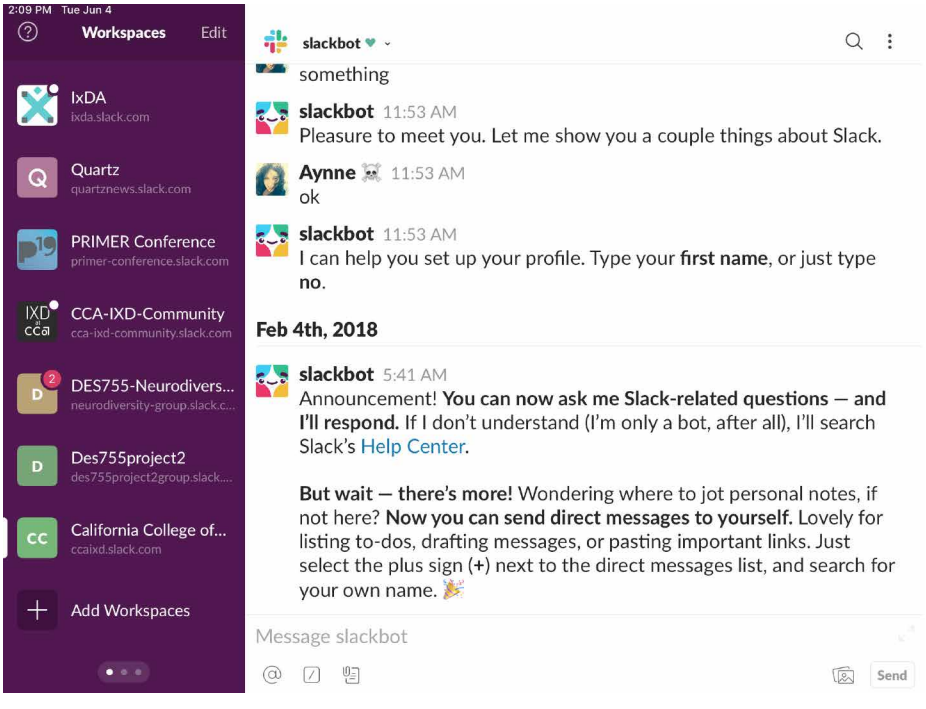


Figure 7-27. *Slack → Add Workspaces*

Lists Abound

In this chapter, we have covered the most common ways to show lists and when to use them. As is evident, a lot of the content you see on the web and in mobile applications are actually lists. When designing for lists, try out different ways in which you might display your content, and remember to always keep in mind what the task your user is trying to do and optimize your design for it.

Chapter 8
Doing Things: Actions and Commands

This chapter is devoted to the "verbs" in the interface: how people can take an action or use a command. In other words, we're going to look at the ways people get work done in software. This chapter explores the following:

- Different methods for initiating action or activating commands
- How to make it clear that an item can be acted on with affordances
- Patterns and components that promote controlling and editing

This is in contrast to our discussions of "nouns" in interface design so far. First, we discussed structure and flow and visual layout. We reviewed interface objects such as windows, text, links, and static elements in pages. In subsequent chapters, we will look at complex components such as data visualizations and forms.

We think of the verbs—designing actions and commands—as the methods people can use to perform tasks in your application. Specifically, what we mean by that is how the person using your software can carry out these tasks:

- Start, pause, cancel, or complete an action
- Enter a setting, configuration, or value
- Manipulate an object or component in the interface
- Apply a change or transformation
- Remove or delete something
- Add or create something

Many of the patterns described in this chapter come from hardware interfaces that were developed and standardized long before software interfaces became ubiquitous.

Other patterns mimic real-world behaviors and methods directly. It's true that there is a lot of history here, and there are many best practices to follow. The standard platform style guides, such as those for Android and iOS, Windows and Macintosh, and JavaScript frameworks for responsive web and mobile user interfaces (UIs) will generally get you pretty close to a workable UI.

Most users now depend upon behaviors they have learned from other applications to negotiate menus and find buttons, so it benefits your users and you to follow those conventions, even if they seem to lack originality. Most of the time, people want to get work done using the interaction methods they already know.

A good UI strategy is to realize that the "lack of originality" in today's software environments just means there are now near-universal UI standards for many common patterns and processes that your audience has already learned. They're ready to use these skills immediately. So, a savvy UI designer, product manager, engineer, or product team will regard today's established software standards, UI toolkits, component libraries, and off-the-shelf frameworks as a useful foundation. Many of the most common application features and functions no longer need to be coded from scratch. This frees up time and energy to design the unique features that truly excite your users and set your endeavor apart.

Examples of common functionality that we can now take for granted include such actions as cut, copy, and paste. Even though this is an abstract process, it is based on real-world actions. The "cut" or removed object or text is held temporarily in the "clipboard"–out of sight, not visible, temporarily in computer memory. Moderately experienced desktop computer users have learned how it's "supposed to work." The same is true for pop-up menus (context menus), which some users seem to look for everywhere, and other users never think to look for at all.

Another example is drag-and-drop. Drag-and-drop is more directly modeled on real-world behaviors: picking up objects and putting them down. But it absolutely has to work the way users intuitively expect it to—putting an object onto a "target drop zone" or onto a folder—or the illusion of direct manipulation is broken.

That being said, you can do many things to make your interface less dull and more usable. Your goals should be to make the appropriate actions available, label them well, make them easy to find, and support sequences of actions. There are a few creative ways to do it.

First, let's list the common ways actions are available to the user:

Tap, Swipe, and Pinch

In mobile operating systems and applications, finger gestures are the primary method for performing actions. There is a wide variety of actions that we can perform in a touch screen operating system (OS). A deep dive into mobile interaction design is beyond the scope of this book. But the major actions to be aware of are tap, swipe, and pinch. Tap means to touch an icon, button, or object in the mobile OS. Doing this will either launch an application, click a button, select an object (like an image), or some other action. This is all determined by context.

Swiping is a common method for doing several other actions. Within an application, swiping on a screen is a way of navigating: scroll a page up or down, move to the next image in a series, move through a carousel or list of screens, or bring up another screen in the app, such as a settings screen or information panel. In a list UI, swiping on a line item is a way of revealing actions that can be applied to the item, such as archive or delete. Pinching usually controls the view or zoom. Pinching—sliding two fingertips inwards, toward each other on a touch screen—causes the view to zoom out on a photo or web page. Reverse pinching on an image or web page—sliding two fingertips apart—causes the view to zoom in, or magnify the page.

Rotate and Shake

Mobile devices are small enough that the entire device can be manipulated to perform commands—something that's impossible with larger devices. The accelerometers and other sensors in the mobile device enable this. For example, it's now almost universally understood that when viewing a video or image on a mobile device, rotating it ninety degrees from vertical to horizontal will change the viewport orientation from portrait to landscape. Most often this is done to maximize the video for better viewing. Shaking the device is also a common way to perform actions. Depending on the application, shaking can skip a song or undo an action.

Buttons

Buttons are placed directly onto the interface, without requiring the user to perform any action to see them, and are usually grouped semantically. (See the *Button Groups* pattern.) They're big, readable, obvious, and extremely easy to use for even the most inexperienced computer users. But they take up a lot of space on the interface, unlike menu bars and pop-up menus. On landing pages, such as corporate home pages and product startup pages, calls to action are usually represented as single, large, eye-catching buttons—this is entirely appropriate for their purpose, which is to attract attention and say, "Click me!"

Menu Bars

Menu bars are standard on most desktop applications. They generally show an application's complete set of actions, organized in a mostly predictable way (such as File, Edit, or View). Some actions operate on the entire application, and some operate only on individually selected items. Menu bars often duplicate functionality found in context menus and toolbars because they are accessible—screen readers can read them, users can reach them via keyboard accelerators, and so on. (Accessibility alone makes menu bars indispensable in many products.) Menu bars appear in some web applications, especially productivity software, drawing programs, and other products that emulate desktop apps.

Pop-Up Menus

Also known as *context menus*, pop-up menus are raised with a right-mouse click or some similar gesture on panels or items. They usually list context-specific, common actions, not all the actions that are possible on the interface. Keep them short.

Drop-Down Menus

Users raise these menus by clicking on a drop-down control such as a combo box. However, drop-down controls are intended for selecting choices on a form, not for performing actions. Avoid using them for actions.

Toolbars

The canonical toolbar is a long, thin row of icon buttons. Often, they have other kinds of buttons or controls on them, too, such as text fields or *Drop-down Choosers* (Chapter 10). Iconic toolbars work best when the portrayed actions have obvious visual renderings; when the actions really need to be described with words, try other controls, such as combo boxes or buttons with text labels. Cryptic icons are a classic source of confusion and lack of usability.

Links

Buttons don't need borders. Thanks to the web, everyone understands that colored text (especially blue text) usually indicates a clickable link. In a UI area where actions are expected but where you don't need to draw attention or clutter the page, you can use simple clickable "link" text for actions instead of buttons. The link can be underlined by default, or you can have the underline appear only on hover. When the user rolls the mouse over the text, change the cursor and the link rendering (background color or border, for example) to reinforce the impression of clickability.

Action Panels

These are menus that the user doesn't need to open; they're always visible on the main interface. They are a fine substitute for toolbars when actions are better described verbally than visually. See the *Action Panel* pattern.

Hover Tools

If you want to show two or more actions for each item on an interface but you don't want to clutter the page with lots of repeated buttons, you can make those buttons invisible until the mouse hovers over the item. (This is great for mouse-driven interface, but it doesn't work well for touch screens.) See the *Hover Tools* pattern for more.

Then there are invisible action , which don't have any labels at all to announce what they do. Users need to know (or guess) that they're there, unless you put written instructions on the UI. Therefore, they don't help with discovery at all, because users can't read over them to find out what actions are possible. With buttons, links, and menus, the UI actions are available for inspection, so users learn from those. In usability tests, I've seen many users look at a new product and methodically walk down the menu bar, item by item, just to find out what it can do.

That being said, you almost always need to use one or more of the following invisible actions. People often expect to be able to double-click on items, for example. However, the keyboard (or the equivalent) is sometimes the only means of access for visually impaired users and people who can't use a mouse. In addition, the expert users of some operating systems and applications prefer to work by typing commands into a shell and/or by using its keyboard actions.

Single-Clicking Versus Double-Clicking Items

Users in object-oriented operating systems—Windows and macOS—have learned that single-clicking an object such as an image or a document file means they are selecting it in order to perform an action on it. First, select the object. Then, apply an action or command, and it will be performed on the selected object. For example, selecting a file on your computer desktop allows you to perform an action on it, such as "move to trash." Inside an application, single-clicking on an element will allow you to move it, scale it, or apply some action or command to it.

Users tend to view double-clicking as either "open this item," "launch this application," or "edit this item," depending on context. Double-clicking on an image often means opening it in the creator or default application for viewing and editing it. Double-clicking an application's icon directly in most operating systems launches that application. Double-clicking a piece of text might edit it in place.

Keyboard Actions

There are two types of keyboard actions to consider including in your UI designs. Both could be considered types of "accelerators." That is, they are capabilities or features that seem more hidden or complicated, but actually enable more experienced users to complete tasks more quickly. Ideally, the goal for this group is less mouse and arm movement.

Keyboard commands are also critical to enabling access to the interface by people with different levels of physical ability and who might need assistive technology. The goal for this group is to not be required to use the mouse and graphical user interface (GUI) components to enter commands. That's why both of these techniques help the user control the UI without their hands leaving the keyboard. The two types of keyboard actions are *shortcuts* and *tab order*.

Shortcuts

Keyboard shortcuts, such as the well-known Ctrl-S to save, should be designed into most desktop applications to support accessibility by differently abled persons, and efficient, rapid use by experienced users. The major UI platforms, including Windows, Mac, and some Linux environments, each have style guides that describe the standard shortcuts—and they're all very similar. Additionally, menus and controls often have underlined access keys, which let users reach those controls without mouse-clicking or tabbing. (Press the Alt key, and then press the key corresponding to the underlined letter, to invoke these actions.)

Tab Order

In desktop applications, both native OS and web, we have the same accessibility and efficiency goals for tab ordering. Tab ordering means being able to use the tab (or other designated) key to move the keyboard "focus" or selection from one screen UI component to the next. The user can cycle continuously through all the selectable options on a screen. A selected UI component can receive keyboard commands, until the user moves the focus to the next screen component. When a form field or a submit button is selected this way, they can be modified or "clicked" without having to use the mouse. This is useful for people who need to use voice browsers or who find full keyboard and mouse controls beyond their physical capability.

Drag-and-Drop

Dragging and dropping items on an interface usually means either "move this here" or "do this to that." In other words, someone might drag a file onto an application icon to say, "Open this file in that application." Or they might drag that file from one

place in a file finder to another place, thus moving or copying the item. Drag-and-drop is context-dependent, but it almost always results in one of these two actions.

Typed Commands

Command-line interfaces (CLIs) hark back to a much earlier computer era when the GUI had not yet been invented. Computer screens showed only text. Computer operating systems could be controlled by typing commands directly into a line or position on the screen for text input. CLIs generally allow free-form access to all the actions in the software system, whether it's an operating system or an application. We consider these kinds of actions "invisible" because most CLIs don't easily divulge the available commands. They're not very discoverable, though they're quite powerful once you learn what's available—much can be done with a single well-constructed command. As such, CLIs are best for users committed to learning the software very well. Both macOS and Windows allow access to a Terminal mode for interacting with the computer in this way. Unix and DOS operating systems worked this way. Today, written SQL queries are a widely used form of typed commands.

Affordance

When a UI object looks like it might let you do something, such as tap it, click it, or drag it, we say it "affords" performing that action. It is, or has, an affordance. For example, traditional raised-edge buttons afford tapping or clicking; a scroll bar affords dragging; a date picker looks spinnable or rollable and so affords spinning; a text field affords typing; a blue underlined word affords clicking or tapping.

Contemporary mobile UIs and desktop GUIs offer functionality via exactly this type of direct perception, invitation to action, and manipulation. You have good grounds for designing your UI based on the rule of thumb that every interesting visual feature does something.

Building on this, affordances for actions could include the following:

- Icons, objects, or shapes that are different from the rest of the interface
- Text that is styled differently from regular reading copy
- Something that reacts when the mouse pointer rolls over it
- Something that reacts when tapped or swiped
- A highlighted or high-contrast visual design treatment
- Some object that looks manipulable: drop shadows, ridges or texture, highlights
- An object or component that just looks different, or is separated with whitespace, from everything else on the screen

Direct Manipulation of Objects

Today, when the majority of interactions are on a mobile device, the design approach is to assume direct manipulation of the screen components. Tap a button to submit it, swipe a list item to delete it or open a contextual menu, drag an object to move it, pinch a map to zoom out, tap an image to access contextual image controls. This is in contrast to the older, desktop-menus approach in which the user must select an object and then go to another part of the interface to activate a command to apply to the selected object. The main point here is that mobile interfaces mostly go without complicated, indirect action menus and multiple selections. They favor acting on objects one by one, using a few simple gestures to act on the addressable object directly, or invoke contextual menus when needed.

Figure 8-1 illustrates the affordances in the Adobe Premier Rush interface. It is a mobile-only video editing app for social media. It's a good example of a UI challenge. In software interfaces, the user doesn't get many sensory clues about what can be tweaked or handled: visuals give most of the clues, and affordances like highlighting or movement do the rest.

The video player controls at top are a near-universal control—certainly not an inscrutable UI. In the lower panel, things are more challenging. The bright blue vertical line overlies the tracks that hold the video and audio, so that indicates it's a tool or reference point for the media. In fact, it's the playhead, the video scrubber that marks the playback point for the media file above. The clips themselves are displayed as jumbo icons: squares or rectangles with an image from the underlying video. The assumption is that the creator remembers the original recorded events or at least recognizes that these are indeed the clips they want to work with. So the user knows what they represent. These clip objects automatically display a selected highlight outline (so you'll know they're selectable) when they touch the blue playback line or are tapped to select.

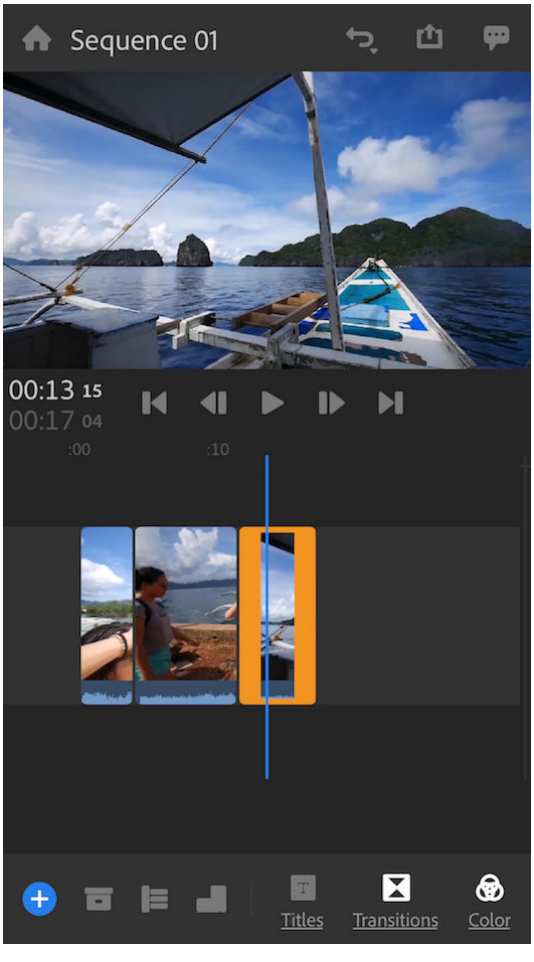


Figure 8-1. *Adobe Premier Rush*

The Patterns

The first patterns in this chapter talk about three of the many ways to present actions. When you find yourself reflexively putting actions on an application's menu bar or pop-up menu, stop for a moment and consider using one of these instead:

- *Button Groups*
- *Hover or Pop-Up Tools*
- *Action Panel*

Prominent "Done" Button or Assumed Next Step improves the single most important button on many web pages and dialog boxes. *Smart Menu Items* is a technique for improving some of the actions you put on menus; this is a very general pattern, useful for many kinds of menus (or buttons or links).

We'd like it if all the user-initiated actions in an application could be completed instantly, but that's not reality. *Preview* shows the user what's going to happen before a time-consuming action is committed. *Spinners and Loading Indicators* is a well-known technique for letting the user know where they are in a multi step process, whereas *Cancelability* refers to a UI's ability to stop an operation when the user asks it to.

The last three patterns all deal with sequences of actions:

- *Multilevel Undo*
- *Command History*
- *Macros*

These three interlocking patterns are most useful in complex applications, especially those whose users are committed to learning the software well and using it extensively. (That's why the examples come from complex software such as Linux, Photoshop, and MS Word.) Be aware that these patterns are not easy to implement. They require the application to model a user's actions as discrete, describable, and sometimes reversible operations, and such a model is very difficult to retrofit into an existing software architecture.

For further discussion of designing actions and commands, we recommend *Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma, Erich, et al.; Addison-Wesley, 1998.)

Button Groups

What

A group of related actions as a small cluster of buttons, aligned and with similar graphic treatments. Multiple groups are possible if there are more than three or four actions.

Use when

There are many actions to show on the interface. You want to make sure they are all visible all the time, but you need to visually organize them so that they're not chaotic or difficult to sort out. Some of these actions are similar to one another—they have

similar or complementary effects, for instance, or they operate with similar semantics—and they can thus be assembled into groups of two to five.

You can use *Button Groups* for app-wide operations (such as Open or Preferences), item-specific actions (Save, Edit, Delete), or any other scope. Actions with different scope ought not to be grouped together, however.

Why

Grouping buttons helps make an interface self-describing. Well-defined clusters of buttons are easy to pick out of a complex layout, and because they're so visible, they instantly communicate the availability of those actions. They announce, "These are the actions that are available to you in this context."

Gestalt principles (Chapter 4) apply here. Proximity hints at relatedness; if the buttons are all together, they probably do related things. So does visual similarity; if you make all the buttons the same dimensions, for instance, they look like they belong together. Conversely, button groups that are separated in space—or that are different in shape—imply unrelated groups of actions.

Proper sizing and alignment help the *Button Groups* form a larger composite visual shape (this is the principle of closure).

How

Make a group out of related buttons so that they form a natural or logical set. An additional option is to label them with short but unambiguous verbs or verb phrases. Use vocabulary that makes sense to the users. Do not mix buttons that affect different things or have different scope; separate them into different groups.

All buttons in the group should have the same graphic treatment: borders, color, height and/or width, icon style, dynamic effects, and so on. You can line them up in a single column, or arrange them in a single row if they aren't too wide.

(However, treat them differently if one action is a "primary" action, such as a Submit button on a web form. A primary action is an action that you want most users to take or that most users will expect to take. Give that button a stronger graphic treatment to make it stand out among the others.)

If all the buttons in a group act on the same object or objects, put the *Button Groups* to the left or right of those objects. You could put them below the objects instead, but users often have a "blind spot" at the bottom of complex UI elements such as multicolumn lists and trees—the user might not see the buttons at all. To make them more visible, keep the rest of the interface clean and uncluttered. If you have a specific design that works better with the buttons at the bottom, test its usability and find out.

If there are enough buttons and if they have icons, you could also put them on a toolbar or ribbon-like strip at the top of the page.

By using *Button Groups*, you're trying to avoid a crowded mess of buttons and links, or perhaps a long and plodding list of actions with no apparent differentiation at all. With this pattern, you create a visual hierarchy of actions: the user can see at a glance what's related and what's important.

Examples

Standard tools for graphic editors are often grouped by function. Figure 8-2 shows some common tools in groupings in Google Docs (separated by vertical lines, or "pipes") that actually aid recognition. There are no fewer than 27 buttons on this interface. There's a lot to understand and keep track of. But thanks to careful visual and semantic organization, the interface is never overwhelming.

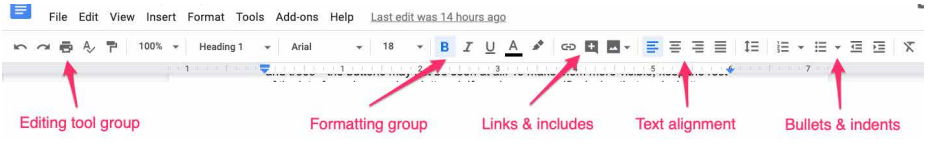


Figure 8-2. *Button Groups in Google Docs*

The second example (Figure 8-3) shows the header of a finder window from macOS. True to its design tradition, buttons are clearly more button-like. The Navigation group is two buttons placed together in a group. The View controls button group is a segmented button. The whole set has rounded edges only on the left and right, not for the contiguous buttons in the middle.

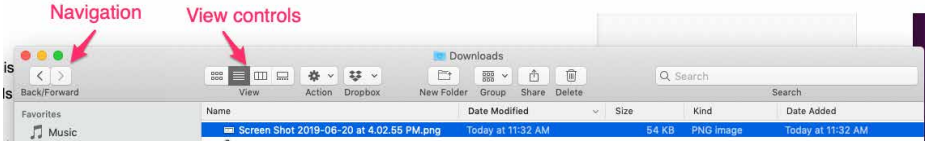


Figure 8-3. *Apple macOS Finder*

Hover or Pop-Up Tools

What

Place buttons and other actions next to the items they act upon, but hide them until the user hovers the pointer over them. In a mobile UI, have the tools appear next to an object when the user taps it.

Use when

There are many actions to show on the interface. You want a clean, uncluttered look most of the time, but you need to put those actions somewhere, preferably on or next to the items they act upon. You've already allocated the space to show those actions, but they just make things too crowded and busy if they're all visible all the time.

Hover Tools are commonly used in list interfaces, in which many small items—photos, messages, search results, and so on—are displayed in a column or list. The user can perform a number of actions on each one.

You don't intend the interface to be used with fingertips, as with a touchpad device—you're certain that almost all users will interact with your UI via a mouse.

Why

Hover Tools reveal themselves exactly when and where they're needed. They stay out of sight otherwise, allowing the UI to remain clean and uncluttered. They appear when the user requests them, and by appearing in response to the user's gesture, they draw attention to themselves.

Pop-up (right-click) menus, pull-down menus, and menu bars also meet these criteria, but they are not discoverable enough for some kinds of interfaces—they're best used on traditional desktop applications, not web-based interfaces. (And sometimes they're not the best choice on traditional applications, either.) *Hover Tools* are more easily discoverable because the gesture that produces them—a rollover—is so simple and natural.

Unfortunately, on touch screens, we have lost the ability to have a mouse and so there isn't a hover state. On a touchpad, the only way a user can see the "Hover Tools" is if they actually touch the hover area. In these situations, if there are tools or actions that can apply to the object, display them in a pop-up panel or list that is grouped with the tapped object, or on top of it.

How

Design each item or hover area with enough space to show all the available actions. Hide the ones that clutter the interface too much, and show them only when the user hovers the mouse pointer over the area in question.

Respond quickly to the hover, and don't use an *Animated Transition*—simply show the tools immediately, and hide them immediately when the user moves the pointer away. Likewise, never enlarge the hover area or otherwise rearrange the page when the user hovers the pointer over it. The idea is to make the hover action as lightweight and quick as possible so that the user can easily reach the necessary tools.

If the hover area is an item in a list, you might want to highlight the item by changing its background color or drawing a border around it. The act of showing tools will draw the user's eyes to that area, but highlighting the item will do so even more.

Consider *Hover Tools* as an alternative to a drop-down menu, a pop-up menu, an *Action Panel*, a *List Inlay* with buttons in it, or a set of buttons repeated in each item.

Examples

Slack uses hover tools extensively (Figure 8-4). They appear for each posting in the main feed or in a thread. The alternatives would have been to show all the tools all the time—far too busy and crowded—or to move the tools to the top toolbar, where they would only operate on posts selected in the feed. That's rather complicated. In contrast, the "Hover Tools" are right there and self-explanatory (or at least quickly learned).

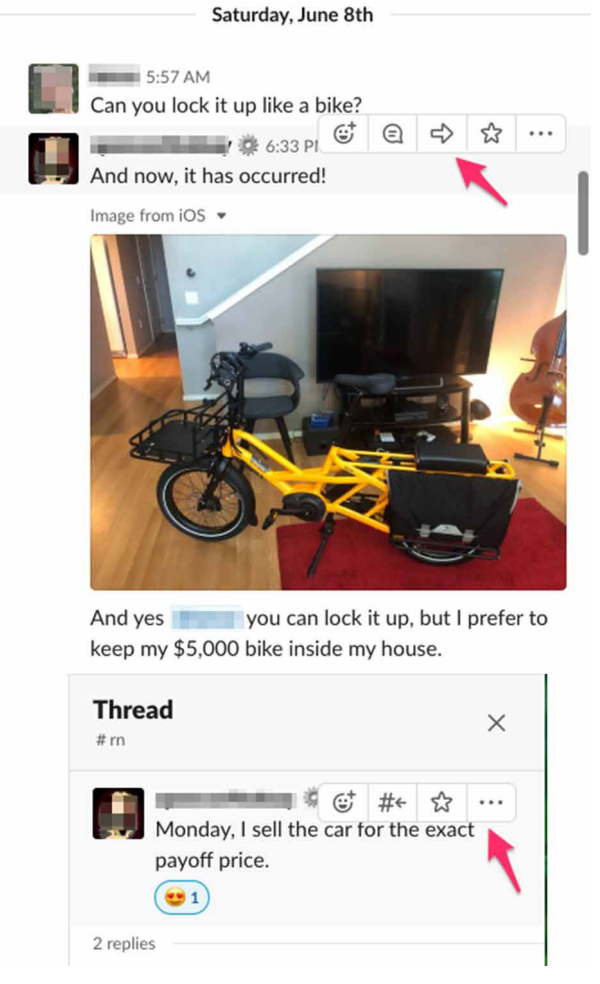


Figure 8-4. *Slack; examples of hover tools for posts and threads*

Other implementations of *Hover Tools* use a show/hide overlay to display buttons or controls such as sliders. This is similar to the *Drop-down Chooser* pattern in Chapter 10, the only difference being your intent to use it for actions and not settings.

In YouTube (Figure 8-5), the YouTube player uses a hover to show the volume slider and other controls. The video player controls only display when the mouse scrolls over the player area itself. Otherwise, they are hidden so that there is less clutter to distract from the video itself.

Figure 8-5. *YouTube web player*

Action Panel

What

A panel or other grouping of commands that is more than just a static menu. The actions can promote the most common actions or the most relevant commands, depending on where the user is or what they are doing in the software. Action panels are a way to feature commands in a more eye-catching way.

Use when

You have a list of items, and a set of actions that can be performed on each one—too many to show all the actions for each item, and too many for *Hover Tools*. You could put them into a menu, but you might not have a menu bar at all, or you'd rather make the actions more discoverable than they would be on menu bars. Same for pop-up menus; they're just not visible enough. Your users might not even realize the pop-up menus exist.

Or maybe your set of possible actions is too complex for a menu. Menus are best at showing a flat set of actions (because pull-right menus, or cascading menus, are difficult for some users to manipulate) in a very simple, linear, one-line-per-item

presentation. If your actions need to be grouped, and especially if those groups don't fit the standard top-level menu names—such as File, Edit, View, Tools, and so on—you might want a different presentation altogether.

This pattern can take up a lot of screen space, so it's not usually a good choice for small devices.

Why

There are three main reasons to use *Action Panel* instead of menus or per-item buttons: visibility, available space, and freedom of presentation.

By placing the actions out on the main UI and not hiding them inside a traditional menu, you make those actions fully visible to the user. Really, *Action Panel* are menus in the generic sense; they just aren't found in menu bars, drop downs, or pop ups. Users don't need to do anything to see what's on an *Action Panel*—it's right there in front of them—so your interface is more discoverable. This is particularly nice for users who aren't already familiar with the traditional document model and its menu bars.

There are many, many ways to structure objects on an interface: lists, grids or tables, hierarchies, and just about any custom structure you can devise. But *Button Groups* and traditional menus give you only a list (and not a very long one at that). An *Action Panel* is free-form—it gives you as much freedom to visually organize verbs as you have for nouns.

How

Putting the *Action Panel* on the UI. Set aside a block of space on the interface for the *Action Panel*. Place it below or to the side of the target of the action. The target is usually a list, table, or tree of selectable items, but it might also be a document in *Center Stage* (Chapter 4). Remember that proximity is important. If you place the *Action Panel* too far away from whatever it acts on, users might not grasp the relationship between them.

The panel could be a simple rectangle on the page. It could be one of several tiled panels on the page, perhaps a *Movable Panels* (see Chapter 4), a "drawer" in macOS, or even a separate window. If it's closable, make it very easy to reopen, especially if those actions are present only on the *Action Panel* and aren't duplicated on a menu!

The odds are good that you'll need to show different actions at different times. So, the contents of the *Action Panel* might depend on the state of the application (e.g., are there any open documents yet?), on the items selected in some list somewhere, or other factors. Let the *Action Panel* be dynamic. The changes will attract the user's attention, which is good.

Structuring the actions. Next, you need to decide how to structure the actions you need to present. Here are some ways you could do it:

- Simple lists
- Multicolumn lists
- Categorized lists with headings and groupings
- Tables or grids
- Trees
- Any combination of these in one panel

If you categorize the actions, consider using a task-centered approach. Group them according to what people intend to do. However, try to present them linearly. Imagine reading the actions aloud to someone who can't see the screen—can you proceed through them in a logical fashion, with obvious start and end points? That, of course, is how a blind user would "hear" the interface.

Labeling the actions. For each action's label, you could use text, icons, or both, depending on what conveys the nature of the actions best. In fact, if you use mostly icons, you end up with…a traditional toolbar! (Or a palette, if your UI is a visual builder-style application.)

Text labels on an *Action Panel* can be longer than those on a menu or a button. You can use multiline labels, for instance—better to explain fully. Just remember that longer, more descriptive labels are better for first-time or infrequent users who need to learn (or be reminded) what these actions do. The extra space spent on long labels might not be appreciated in high-performance interfaces used mostly by experienced users. If there are too many words, even first-time users' eyes will glaze over.

Examples

The example in Figure 8-6 is from Dropbox. This is the desktop web UI for that service. on the right of the screen, there is an action panel. The purpose is to have always-visible, one-click access to the most common actions. The most likely or most frequently used command is given special treatment. Altogether, the grouping and separation of these commands show that this is an action panel.

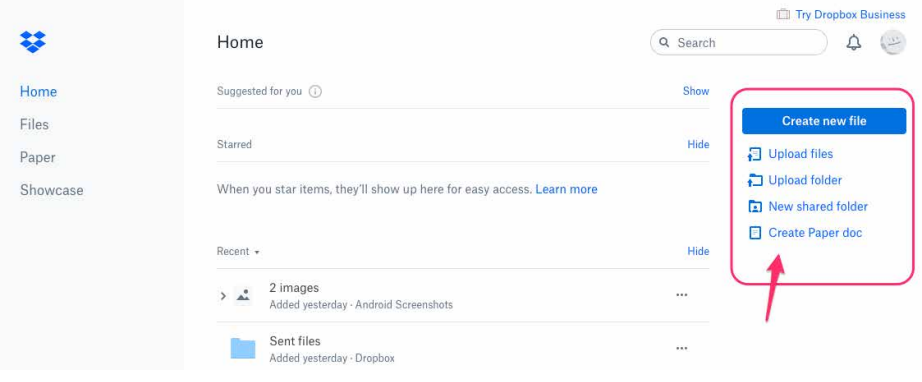


Figure 8-6. *An Action Panel in Dropbox*

The following screenshots from Windows 10 show two examples of a show/hide *Action Panel*. These are not always visible due to their screen-filling size. The user must click to open them. But after they're open, they stay open and show a large number of options and actions the user could take.

Microsoft Windows 10 Start Menu (Figure 8-7)—the legendary pop-up menu—has been expanded beyond the classic list of app icons to launch. Now the panel is much larger in order to show groups of tiles (actually large, square buttons, some with dynamic status, such as the latest weather). The tiles are grouped according to the users' anticipated most likely tasks. Selecting one will launch an application or open a directory.

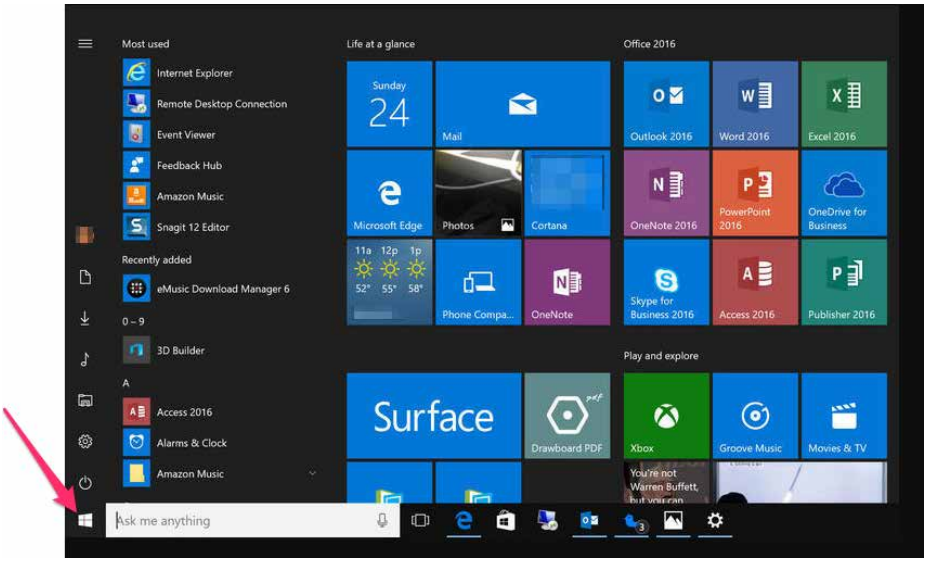


Figure 8-7. *Microsoft Windows 10 Start Menu*

The Microsoft Windows 10*Action Panel* (Figure 8-8) is accessed via the "speech bubble" icon in the lower right. Most of this panel is a scrolling list of notifications. Many of these notifications are calls to action: the user needs to change a setting, activate a process or fix a problem. The notifications are clickable, so the user can take action directly from this list. At bottom is a set of buttons to access notification and system settings. Other design resources call this pattern a *Task Pane*.

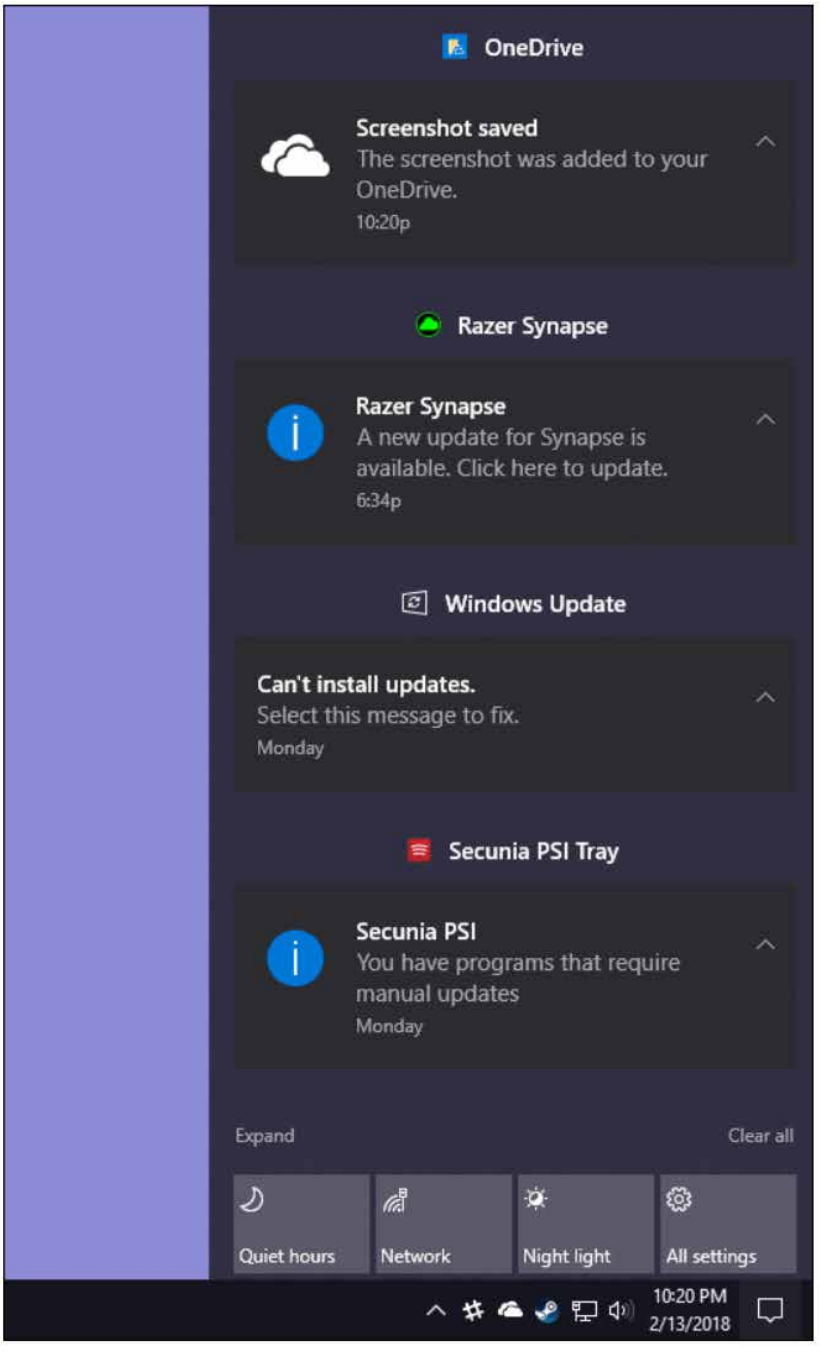


Figure 8-8. *Microsoft Windows 10 Action Panel*

Prominent "Done" Button or Assumed Next Step

What

A button or other obviously selectable screen component that is the preferred next step or obvious conclusion to a screen or process.

Use when

Whenever you need to put a button such as Done, Submit, OK, or Continue on your interface , you should use this pattern. More generally, use a visually prominent button for the final step of any transaction—such as an online purchase—or to commit a group of settings.

Why

A well-understood, obvious last step gives your users a sense of closure. There's no doubt that the transaction will be done when that button is clicked; don't leave them hanging, wondering whether their work took effect.

Making that last step obvious is what this pattern is really about. Doing it well draws on the layout concepts in Chapter 4: visual hierarchy, visual flow, grouping, and alignment.

How

Create a button that actually looks like a button, not a link; either use platform standards for pushbuttons, or use a large or medium-sized button graphic with bold colors and well-defined borders. This will help the button stand out on the page and not be lost among other things.

Place the button that finishes a transaction at the end of the eye's travel through the visual layout of the screen. Give it a label that is easy to understand. Make the whole button very obvious to see.

When labeling the button, prefer text labels to icons. They're easier to understand for actions such as this, especially given that most users will look for a button labeled "Done" or "Submit." The text in that label can be a verb or a short verb phrase that describes what will happen in the user's terms—for example, "Send," "Buy," or "Change Record" are more specific than "Done" and can sometimes communicate more effectively.

Place the button where the user is most likely to find it. Trace the task flow down through the page or form or dialog box , and put the button just beyond the last step. Usually that will be on the bottom and/or right of the page. Your page layouts might

have a standard place for them (see the *Visual Framework* pattern in Chapter 4), or the platform standard might prescribe it; if so, use the standard place.

In any case, make sure the button is near the last text field or control. If it's too far away, the user might not find it immediately upon finishing their work, and they might go looking for other affordances trying to find out "what to do next." On the web, users can end up abandoning the page (and possibly a purchase) without realizing it.

Examples

The Google Play store on an Android OS mobile device (Figure 8-9) displays information about a specific game. The preferred action, Install, is obvious from the size, color, position and white space around the Install button.

This is a good implementation in a mobile context. You can see the action button without even reading the labels, due to visual design alone:

- The green color stands out. It's a saturated color and it contrasts with the white background. (A white or light gray button with a black border would blend into the form.)
- The graphic used for the button looks like a button. It's a rectangle with subtle rounded corners. It is large, too.
- The button is positioned under and to the right of the content, in this case, a mobile game. Both the task flow (the user scans from top to bottom) and the visual flow bring the user's eye to rest at that button.
- The button is set off by whitespace.

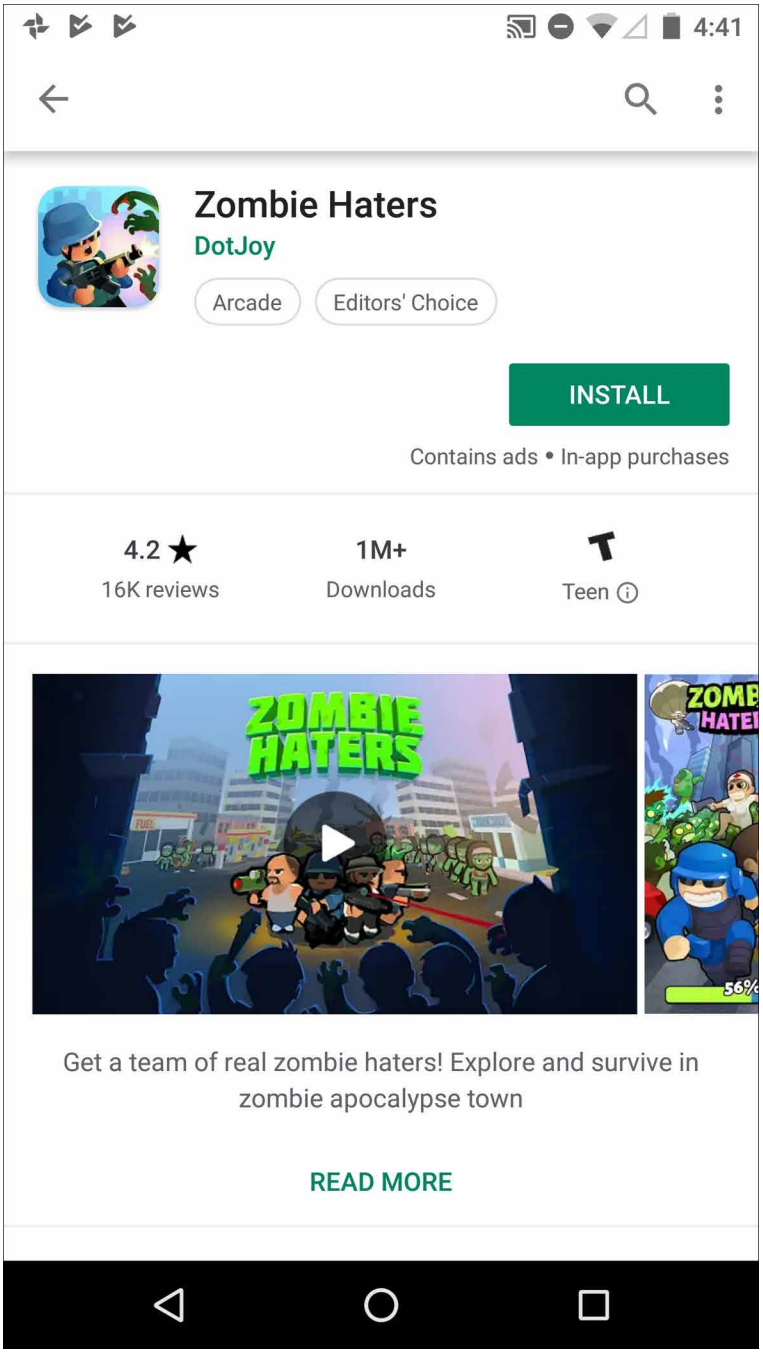


Figure 8-9. *Google Play store, Android OS mobile device*

JetBlue.com (Figure 8-10), *Kayak.com* (Figure 8-11), and *Southwest.com* (Figure 8-12) use strong buttons on their home page flight-search interfaces. These follow all the guidelines for *Prominent "Done" Buttons*, and again, you can see them immediately. In each, Search is the most prominent action step on the screen. It is called out as a button element and formatted with prominent size and contrasting color. (Southwest actually has two calls to action: Search and Book Now, for the current flight promotion).

Among these examples, JetBlue is using the most effective design. The button is easy to see due to color contrast, centered location, generous surrounding negative space, and its label.

Kayak uses a strong color pop for its search button, too, but it is less effective because its button uses an icon only and is on the far-right edge of the screen, where it is not immediately seen. The magnifying glass is a standard shorthand for search, but the user still must translate the shape into the word or concept.

Southwestuses the same button design for two buttons, so the user's attention is split. It's no longer a single call to action. In the lower Book panel, the button is effectively used: contrasting color to the white and blue, and a label and anchor location that message the next step for the job the panel has to do (help users find a flight). At the top, in the promotion area, the red and yellow are a lower-contrast combination, and the button seems a little lost. There are other, much larger components to this area of the screen that draw the eye, and the button is not so clearly aligned or placed so that the eye is driven to it.

Airbnb (Figure 8-13) offers a clear done/next step button on its home screen. The user fills out the booking search form. A single large Search button, designed to draw the eye, is the action that Airbnb wants to promote.

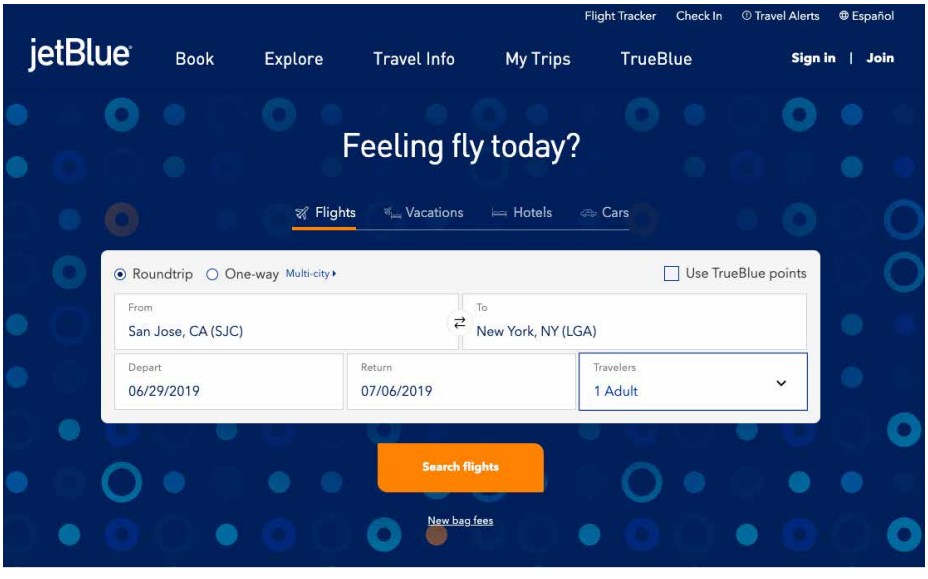


Figure 8-10. *JetBlue.com*

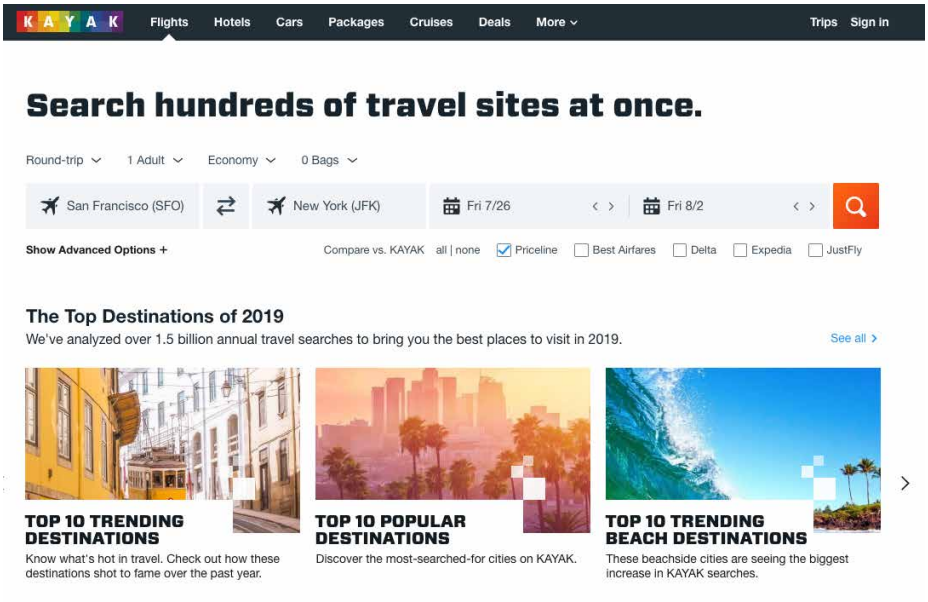


Figure 8-11. *Kayak.com*

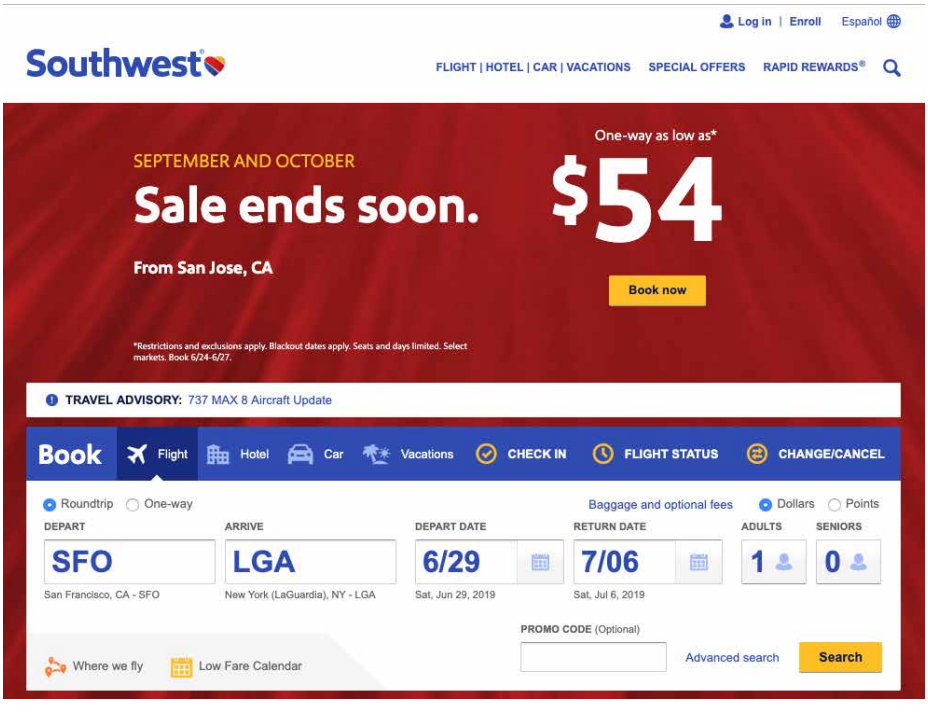


Figure 8-12. *Southwest.com*

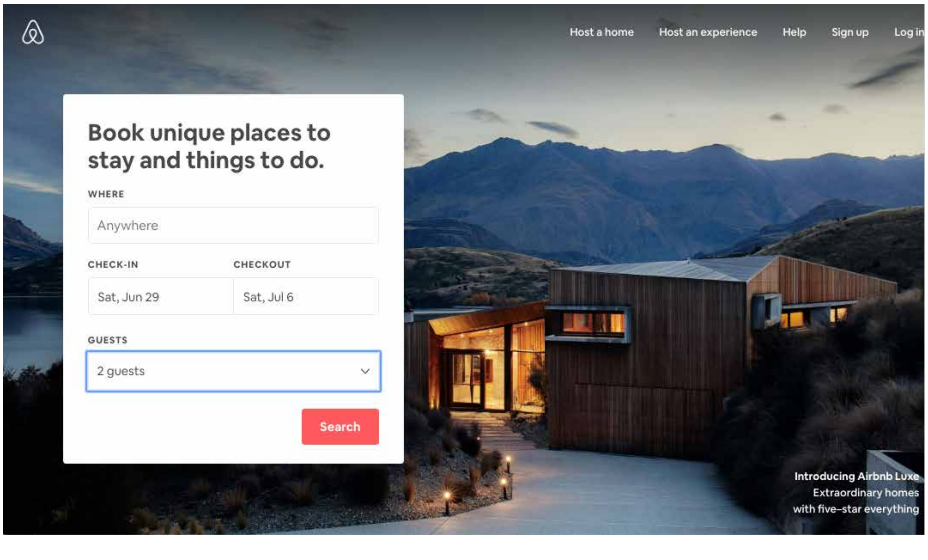


Figure 8-13. *Airbnb.com*

Smart Menu Items

What

Menu labels that dynamically show precisely what they will do when invoked. This is a mechanism for making menus more efficient and responsive by offering different choices depending on what the user is doing.

Use when

Your UI has menu items that operate on specific documents or items, such as Close, or that behave slightly differently in different contexts, such as Undo.

Why

Menu items that say exactly what they're going to do make the UI self-explanatory. The user doesn't need to stop and figure out what object will be affected. They're also less likely to accidentally do something that they didn't intend to do, such as deleting "Chapter 8" instead of "Footnote 3." It thus encourages safe exploration.

How

Every time the user changes the selected object (or current document, last undoable operation, etc.), change the menu items that operate on it to include the specifics of the action. Obviously, if there is no selected object at all, you should disable the menu item, thus reinforcing the connection between the item and its object.

Incidentally, this pattern could also work for button labels, or links, or anything else that is a "verb" in the context of the UI.

What if there are multiple selected objects? There's not a whole lot of guidance out there—in existing software, this pattern mostly applies to documents and undo operations—but you could write in a plural, as in "Delete Selected Objects."

Examples

Figure 8-14 shows a menu from the Adobe Lightroom menu bar. The first selection in the Edit drop-down menu is dynamic. The last filter the user applied in this case was the "Increase Clarity" filter. The menu remembers that, so it changes its first item to reflect that. The Undo action that it triggers is based on the immediately previous action performed by the user. The label changes depending on the actions the user takes.

The accelerator keystrokes are handy for repeated application of the same filter.

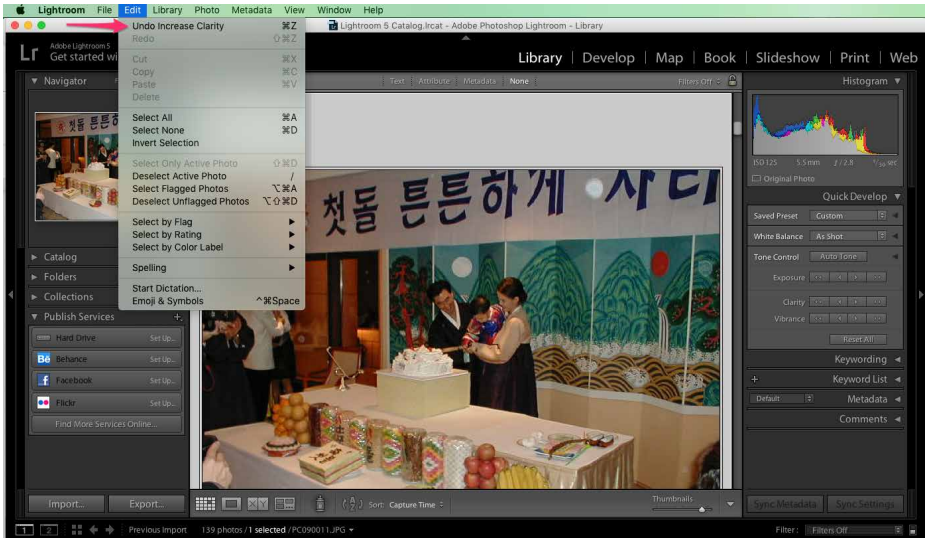


Figure 8-14. *Adobe Lightroom*

The previous example is from application menu bars , but you also can use this pattern effectively in contextual tools, such as the drop-down menu in Gmail (Figure 8-15). Several commands in the drop-down menu change depending on the currently selected email message. The menu item "Add [person from email] to Contacts list" is much clearer and more self-explanatory than a generic alternative, such as "Add sender to Contacts list."

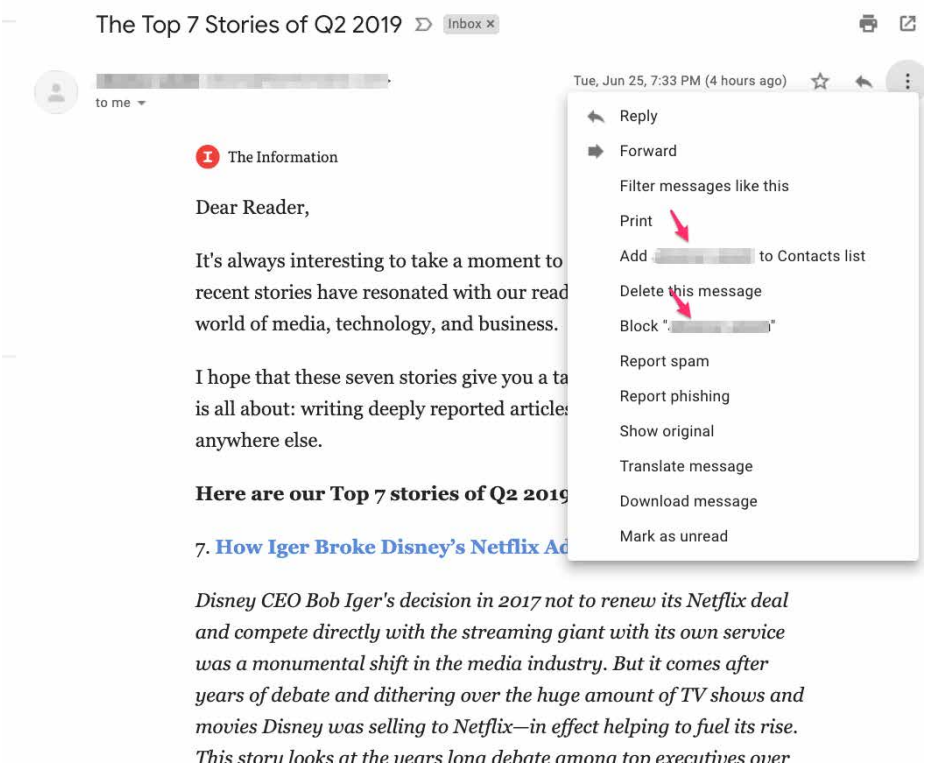


Figure 8-15. *Gmail*

Preview

What

Render a lightweight sample of the effects of a command, or the likely outcomes of an action. They are proactively rendered before the user has even submitted the action. This is a pattern in which the user is presented with realistic modeling of their possible action outcomes so that they can choose the one they like or want. This pattern is the opposite of the more simplistic and traditional interaction design method whereby the user must activate a command first and then wait to see what the results are.

Use when

The user is just about to perform a "heavyweight" action such as opening a large file, printing a 10-page document, submitting a form that took time to fill out, or committing a purchase over the web. Users want some assurance that they're doing it

correctly. Doing it incorrectly would be time-consuming, difficult to reverse, or otherwise costly.

Alternatively, the user might be about to perform some visual change with a difficult-to-predict result, such as applying a filter to a photo. It's better to know in advance whether the effect will be desirable.

Why

Previews help prevent errors. A user might have made a typo or misunderstood something that led to the action in question (such as purchasing the wrong item online). By showing the user a summary or visual description of what's about to happen, you give them a chance to back out or correct any mistakes.

Previews can also help an application become more self-describing. If someone's never used a certain action before, or doesn't know what it will do under certain circumstances, a preview explains it better than documentation—the user learns about the action exactly when and where they need to.

How

Just before the user commits an action, display whatever information gives them the clearest picture of what's about to happen. If it's a print preview, show what the page will look like on the chosen paper size; if it's an image operation, show a close-up of what the image will look like; if it's a transaction, show a summary of everything the system knows about that transaction. Show what's important—no more, no less.

Give the user a way to commit the action straight from the preview page. There's no need to make the user close the preview or navigate elsewhere.

Likewise, give the user a way to back out. If they can salvage the transaction by correcting information previously entered, provide a chance to do that too, with "Change" buttons next to changeable information. In some wizards and other linear processes, this might just be a matter of navigating a few steps backward.

Examples

Apple Photos (Figure 8-16, left) gives users a wide variety of photo filters. Each filter offers a "what you see is what you get" prospective render. When editing a selected photo, each filter choice at the bottom of the screen displays what that image would look like, using that filter. Users do not need to guess what a filter might do, or make a filter selection first in order to see what it does. They can simply review the rendered thumbnails and select the one they like, based on the realistic preview of the image. From a usability perspective, it is much easier and quicker for people to recognize the choice they like and select it, as opposed to having to remember what the command is

and guessing at the outcome. (Photoshop and other image-processing applications use similar previews.)

The Bitmoji app example (Figure 8-16, right) shows a closely related use case. Bitmoji creates customized illustration-style avatars and amusing cartoons that users can share via messaging and social media. The first step in using Bitmoji is for the user to build their own cartoon likeness by adding their choices for hair, eyes, expression lines, skin tone, and other features. Here, the user is trying to find the closest match to their real-world appearance from a limited set of pre rendered options. For skin tone, Bitmoji renders faces based on the user's selections, and then offers a different preview for each of the available skin tones. Creating a more realistic, personalized avatar is easier and faster when the user can scroll through a large selection of skin tone previews.

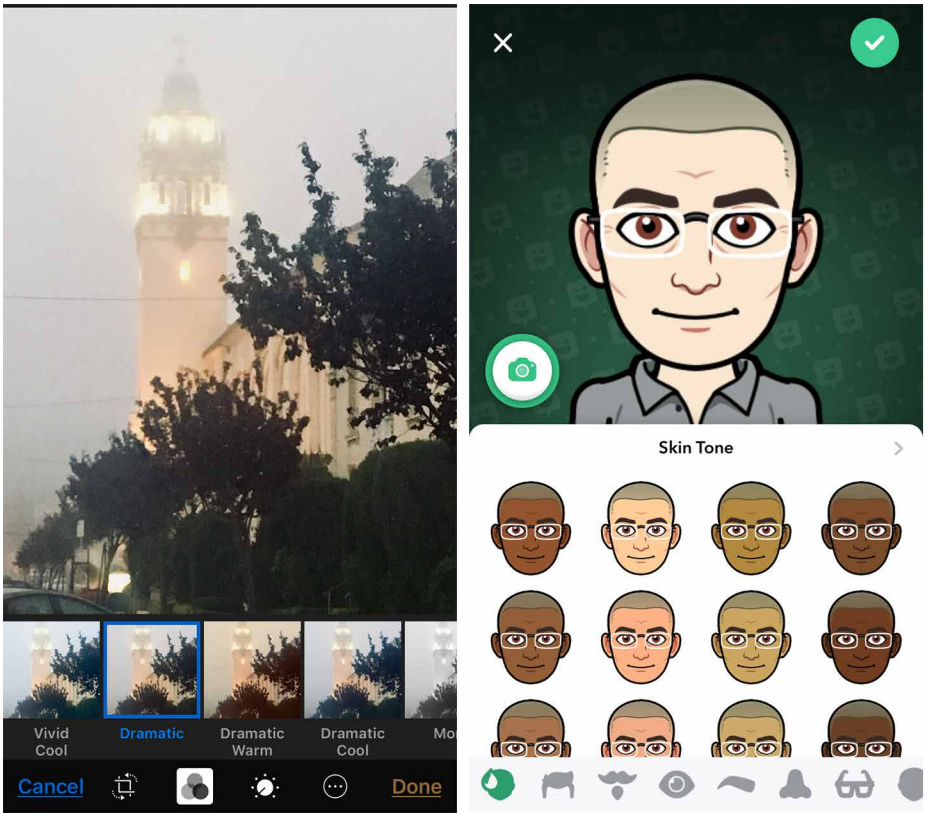


Figure 8-16. *Apple Photos app and Bitmoji app*

Online product builders and customizers often use *Previews* to show what the user has created so far as part of the sales process. The customizable Prius car in Figure 8-17 is a good example. As the user specifies the exact features they would like for their Prius, the preview of the vehicle updates to show the user's selections. Multiple previews for exterior and interior help potential buyers get a better idea of what their choices might look like. The user is able to move back and forth between the major steps in the process, and also to experiment with variations at each step to see what it would actually look like. The goal is to get a quote for a car that's based on what the customer exactly wants. *Preview* tools like this are highly engaging.

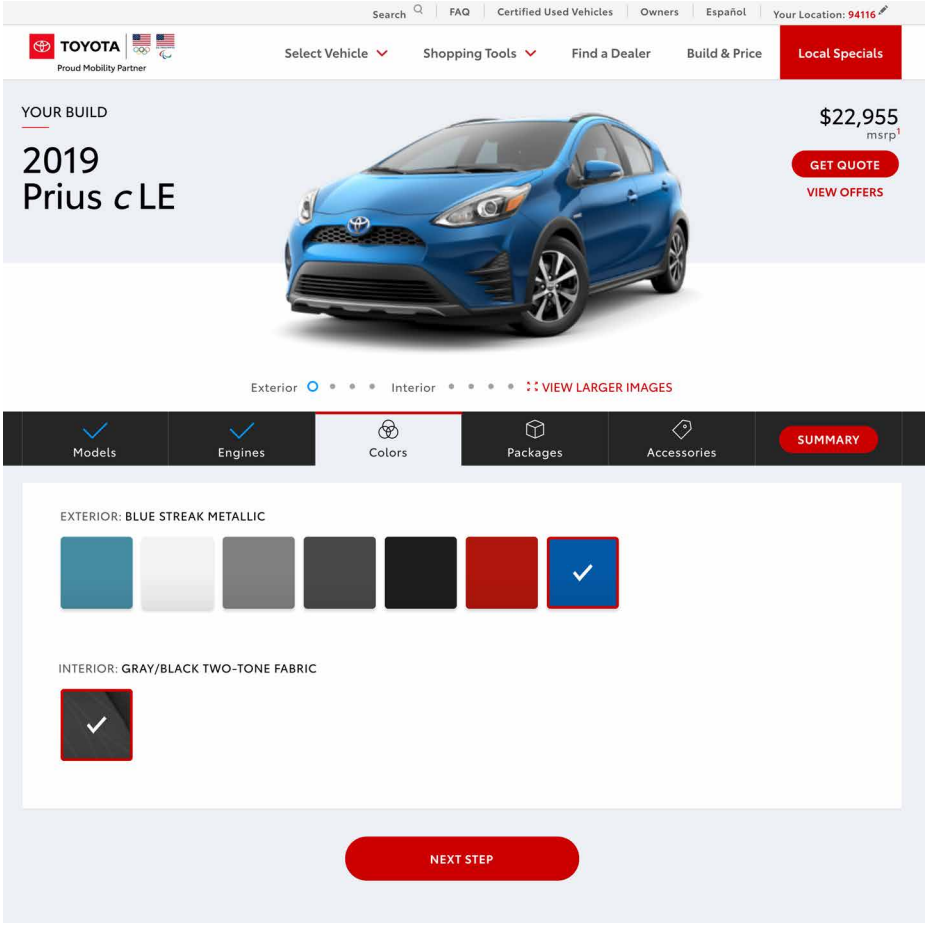


Figure 8-17. *Toyota.com*

The customizable makeup regimen from Sephora.com (Figure 8-18) is a more personal example. Here, the customer is trying to find the right beauty products out of a huge number of possible brands and selections. They can experiment with how things would look. This makeup preview app lets the shopper add an image of their own face and then virtually try out a huge range of cosmetics and techniques. They can preview what different products and colors would do for skin, eyes, lashes, eyebrows, and lips. The outcome of experimenting and previewing is a list of the products that will give them the look she wants. The products that are applied to the face preview automatically appear in the "What You're Wearing" purchase list.

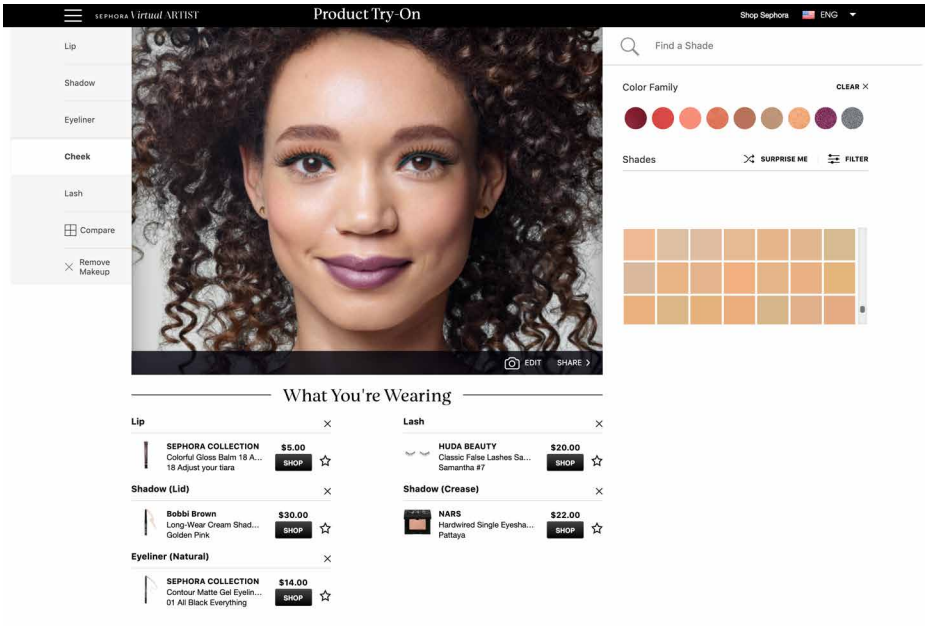


Figure 8-18. *Sephora.com*

Spinners and Loading Indicators

What

An animation or other indicator that displays after a user submits an action and before there is a response. If there is a delay in this response, *Spinners and Loading Indicators* let a waiting user know that their interaction session is "live" and that a response from the system is in progress. This prevents users from breaking their task focus.

Spinners. A spinner is an animation that shows the system is processing something. It is normally stateless; that is, it does not communicate a changing state such as percentage complete (although this is not a rule).

Loading indicators. A loading indicator is usually a meter or thermometer-style animation that shows key data about a task that takes a long time, for example, uploading large files or images, or loading a mobile app on the consumer's mobile device. Loading indicators show a constantly updating "empty/full" meter plus helpful data such as percent complete, bytes of data processed versus unprocessed, and how much time remains to completion.

Use when

A time-consuming operation interrupts the UI, or runs in the background, for longer than two seconds or so.

A summary of advice from prominent usability and digital design experts Don Norman and Jakob Nielsen, reviewing research on this topic, can offer a good set of guidelines:[1]

- Less than one-tenth of a second, users feel they are interacting with a "live" UI because the response from the software feels instantaneous. There is no delay in going from one UI action to the next. This is the expected response time for usable software.
- Between one-tenth of a second and one second, the user is aware of the delay but they will wait, staying on task, with the expectation of continuing immediately.
- Longer than a one-second delay in response, the user is likely to think the UI is not working, that something might be wrong, or they might abandon the task. In this situation, spinners or loading indicators are mandatory if you want users to

1 Nielsen, Jakob. "Response Times: The 3 Important Limits." *Nielsen Norman Group*, Nielsen Norman Group, 1 Jan. 1993, *https://oreil.ly/6IunB*. This article, updated in 2014, cites additional sources of research into software response time and its effect on users.

know that your software is indeed working. Alternately, you might want to let them know that they have time to move on to another activity while the process completes.

Why

Users become impatient when the UI just sits there. Even if you change the mouse pointer to a clock or hourglass (which you should in any case, if the rest of the UI is locked out), you don't want to make a user wait for an unspecified length of time.

Experiments show that if users see an indication that something is going on, they're much more patient, even if they must wait longer than they would without a *Loading Indicator*. Maybe it's because they know that "the system is thinking," and it isn't just hung or waiting for them to do something.

How

Show an animated indicator of how much progress has been made. Either verbally or graphically (or both), tell the user:

- What's currently going on
- What proportion of the operation is complete
- How much time remains
- How to stop it

As far as time estimates are concerned, it's OK to be wrong sometimes, as long as your estimates converge on something accurate quickly. But sometimes the UI can't determine how far along it is. In that case, show a stateless spinner.

Most GUI toolboxes provide a widget or dialog box that implements this pattern. Beware of potentially tricky threading issues, however—the *Loading Indicator* must be updated consistently while the operation itself proceeds uninhibited. If you can, keep the rest of the UI alive, too. Don't lock up the UI while the *Loading Indicator* is visible.

If it's possible to cancel the operation whose progress is being monitored, offer a cancel button or similar affordance on or near the *Loading Indicator*; that's where a user is likely to look for it. See the *Cancelability* pattern (next) for more information.

Spinners are usually used when there is a very slight wait. Their function is to let the user know "we're working on it, hang on a second."

Apple's Touch ID service on its iPhones (Figure 8-19) allows app developers (in this case, CVS) to securely log in their customers without typing usernames and passwords. We see the iOS spinner momentarily while the iPhone/iOSand CVS are processing the log in.

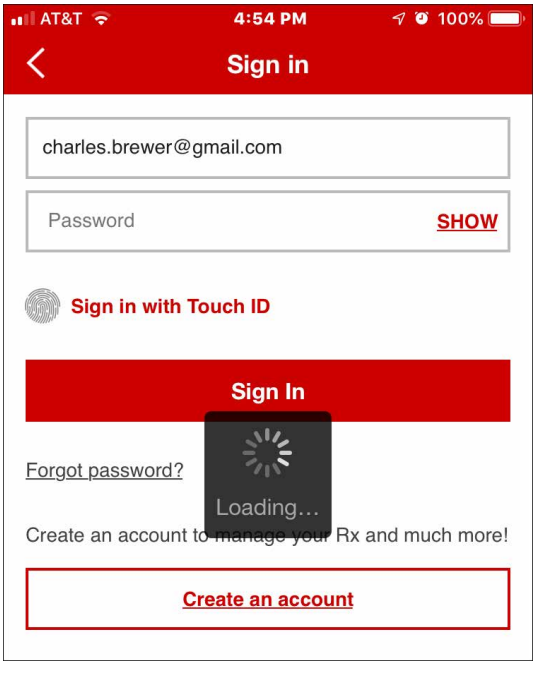


Figure 8-19. *Apple iOS, CVS mobile app; An iOS spinner example*

Spinners are also a standard part of most UI toolkits and frameworks. Figure 8-20 shows the specification and example in the Twitter Bootstrap UI framework. One of the standard components in the Bootstrap library is a customizable spinner, which is ready to be used anywhere in a Bootstrap web application.

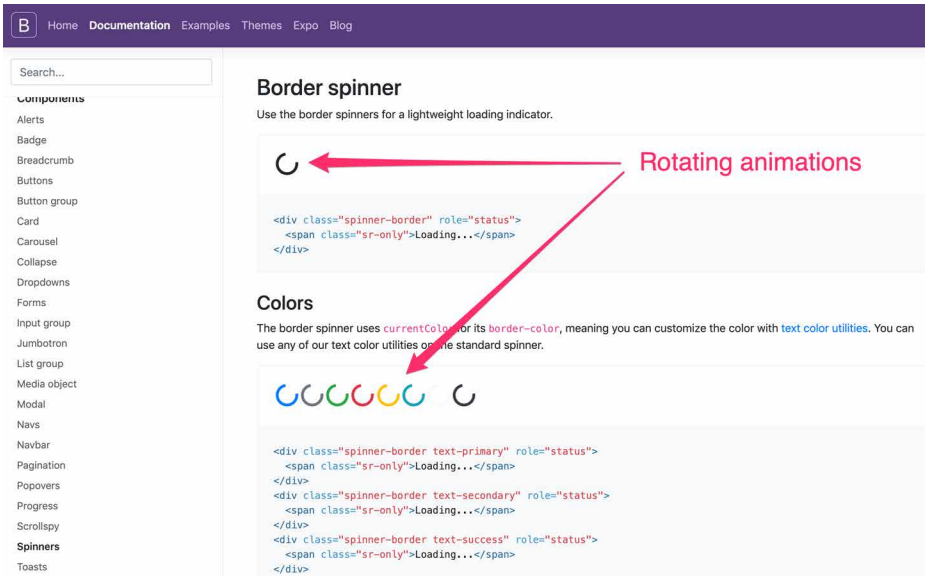


Figure 8-20. *Twitter Bootstrap component library (getbootstrap.com) border spinner*

The Blueprint UI toolkit (Figure 8-21) offers a button component that supports the inline display of a spinner as part of the button. When the user clicks one of these buttons, they would see the label or icon on the button change to the spinner momentarily.

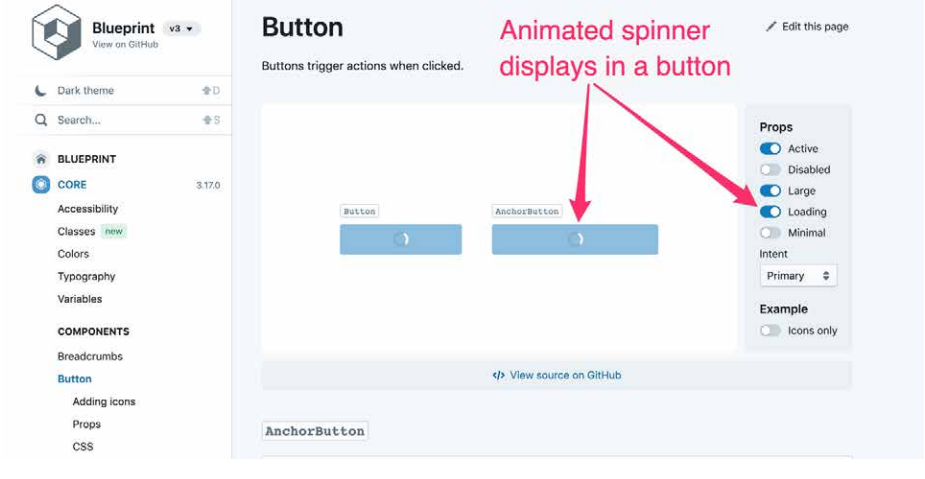


Figure 8-21. *Blueprint UI toolkit (blueprintjs.com), button loading state*

Loading indicators offer more robust status and information for processes that take a longer time. They are for situations in which there is enough time to generate and display this information. The user will then know how long it's going to take and can wait, cancel, or do something else and come back later.

Figure 8-22 shows the Google Play store again, this time with a game download in progress to a user's Android device. The "Install" button has disappeared. It was replaced with the loading indicator. This is an informative loading indicator. The loading indicator is the green and gray horizontal line. The animated green bar represents how much of the game has been downloaded compared to the total file size, represented by the gray bar. The same information is given as updating numerical totals. There is also a percentage complete figure.

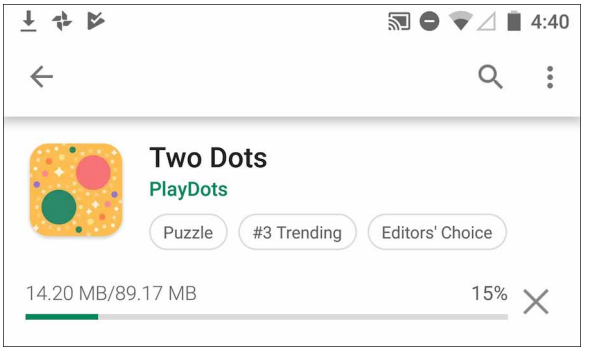


Figure 8-22. *Google Play store, Android OS*

Adobe uses loading indicators in its Creative Cloud application for macOS desktop. This is a small and compact version that is still usable. Figure 8-23 shows an update to Photoshop CC in progress. There is a small thermometer-style loading indicator, which here is 3% complete.

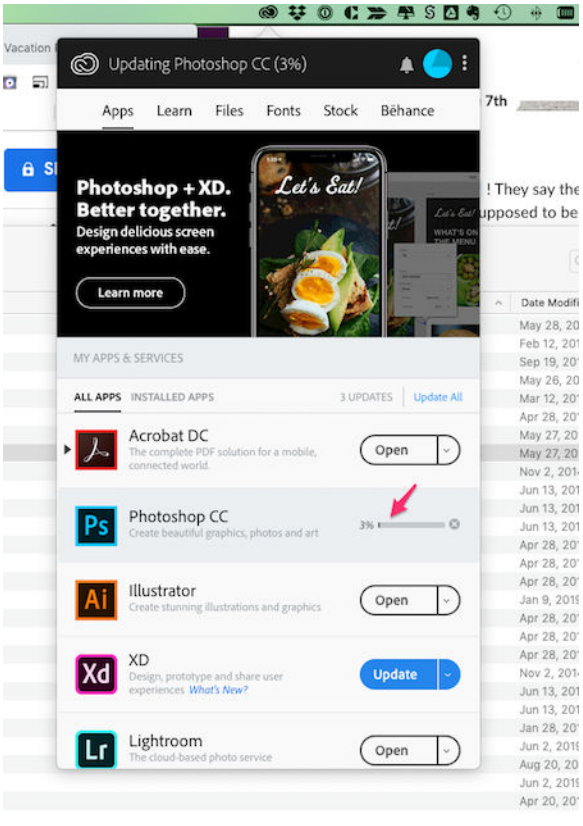


Figure 8-23. *Adobe Creative Cloud desktop manager, macOS*

Cancelability

What

Away to instantly cancel a time-consuming operation, with no side effects.

Use when

A time-consuming operation interrupts the UI or runs in the background for longer than two seconds or so—such as when you print a file, query a database, or load a large file. Alternatively, the user is engaged in an activity that literally or apparently shuts out most other interactions with the system, such as when working with a modal dialog box.

Why

The ability for software users to cancel a task or process in the UI at any time is an important usability standard. It relates to "user control and freedom," one of the top 10 usability heuristics, or guidelines, that software expert Jakob Nielsen of Nielsen Norman Group found from a review of industry research and findings.[2]

Users change their minds. After a time-consuming operation starts, a user might want to stop it, especially if a *Loading Indicator* informs the user that it will take a while. Or the user might have started it by accident in the first place. *Cancelability* certainly helps with error prevention and recovery—a user can cancel out of something they know will fail, such as loading a page from an unreachable website.

In any case, a user will feel better about exploring the interface and trying things out if they know that anything is cancelable. It encourages *Safe Exploration* (Chapter 1), which in turn makes the interface easier and more fun to learn.

How

First, find out if there's a way to speed up the time-consuming operation so that it appears to be instantaneous. It doesn't even need to be genuinely fast; if a user perceives it as immediate, that's good enough. On the web or a networked application, this can mean preloading data or code—sending it to the client before it's asked for—or sending data in increments, showing it to the user as it comes in. Remember, people can read only so fast. You might as well use the loading time to let the user read the first page of data, then another page, and so on.

2 Nielsen, Jakob. "10 Usability Heuristics for User Interface Design: Article by Jakob Nielsen." *Nielsen Norman Group*, 24 Apr. 1994, *https://oreil.ly/Sdw4P*.

But if you really do need *Cancelability*, here's how to do it. Put a Cancel button directly on the interface, next to the *Loading Indicator* (which you *are* using, right?) or wherever the results of the operation will appear. Label it with the word Stop or Cancel, and/or put an internationally recognizable stop icon on it: a red octagon, or a red circle with a horizontal bar, or an "X."

When the user clicks or presses the Cancel button, cancel the operation immediately. If you wait too long—for more than a second or two—the user might doubt whether the cancel actually worked (or you might just dissuade them from using it because they might as well wait for the operation to finish). Inform the user that the cancel worked—halt the *Loading Indicator*, and show a status message on the interface, for instance.

Multiple parallel operations present a challenge. How does the user cancel a particular one and not others? The Cancel button's label or tool tip can state exactly what is being canceled when it's clicked (see the *Smart Menu Items* pattern for a similar concept). If the actions are presented as a list or a set of panels, you might consider providing a separate Cancel button for each action to avoid ambiguity.

Examples

The game install screen in the Google Play store (Figure 8-24) shows a minimalist cancel icon. The green and gray bar here is actually a spinner. The green bar animates left-right for a few seconds while the download connection is established. Note the large "X" cancel icon to the right of this. The consumer can cancel the download and install process at any time.

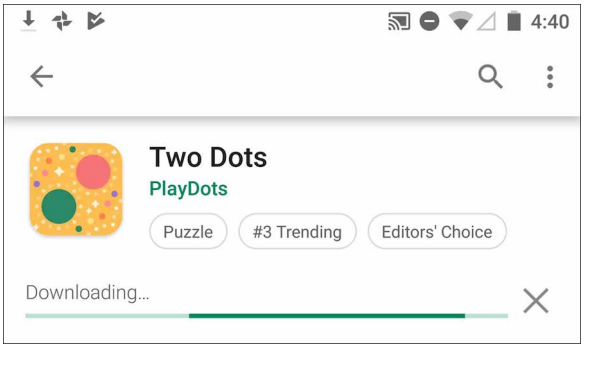


Figure 8-24. *Google Play store, Android OS*

Adobe displays a different style of cancel "X" button in its Creative Cloud desktop app for macOS (Figure 8-25). In this desktop drop-down panel, in the Photoshop CC line item, next to the loading indicator, there is an "X" icon. The customer can select this to cancel the update/install process anytime.

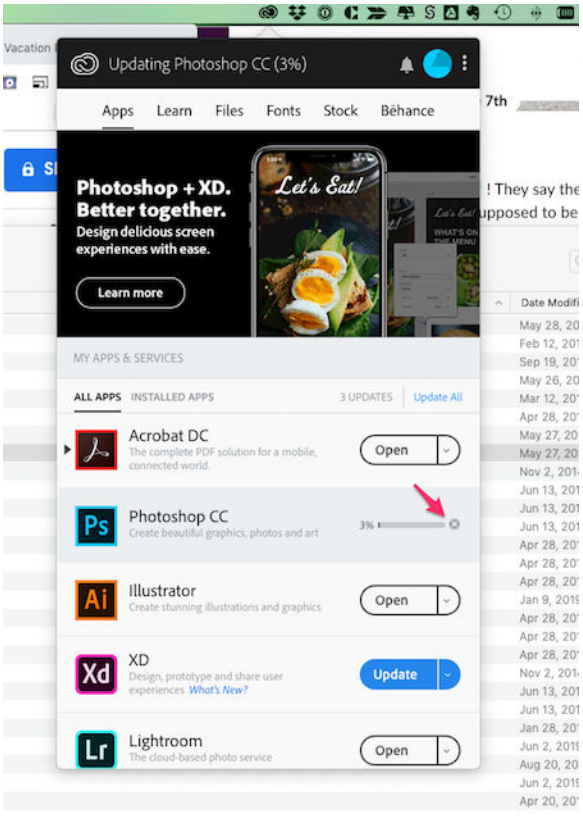


Figure 8-25. *Adobe Creative Cloud desktop manager, macOS*

Multilevel Undo

What

The ability to reverse a series of actions performed by the user. *Multilevel Undo* is a combination of simple undo combined with a history of user actions, with their sequence captured as well as the actions. *Multilevel Undo* is a way of reversing any recent history of commands or actions step by step, in the opposite order in which they were performed. That is, the first undo is for the most recently completed action. The second undo reverses the second most-recent action, and so on. There are usually limits to the length of the history file to support *Multilevel Undo*.

Use when

You're building a highly interactive UI that is more complex than simple navigation or form fill-in. This includes mail readers, database software, authoring tools, graphics software, and programming environments. Design this function when you want to give your users the ability to back out of or recover from a series of actions, not just a single action.

Why

The ability to reverse a long sequence of operations step by step lets users feel that the interface is safe to explore. While they learn the interface, they can experiment with it, confident that they aren't making irrevocable changes—even if they accidentally do something "bad." This is true for users of all levels of skill, not just beginners.[3]

After the user knows the interface well, they can move through it with the confidence that mistakes aren't permanent. If the user's finger slips and they hit the wrong menu item, no complicated and stressful recovery is necessary; they don't need to revert to saved files, shut down and start afresh, or go ask a system administrator to restore a backup file. This spares users wasted time and occasional mental anguish.

Multilevel Undo also lets expert users explore work paths quickly and easily. For instance, a Photoshop user might perform a series of filtering operations on an image, study the result to see whether they like it, and then undo back to the starting point. Then they might try out another series of filters, maybe save it, and undo again. The user could do this without *Multilevel Undo*, but it would take a lot more time (for closing and reloading the image). When a user works creatively, speed and ease of use are important for maintaining the experience of flow. See Chapter 1 for

3 Alan Cooper and Robert Reimann devote an entire chapter to the undo concept in their book *About Face 2.0: The Essentials of Interaction Design* (Wiley, 2003).

more information, especially the *Safe Exploration* and *Incremental Construction* patterns.

How

Reversible operations. The software your UI is built on first needs a strong model of what an action is—what it's called, what object it was associated with, how to record it, and how to reverse it. Then, you can build an interface on it.

Decide which operations need to be reversible. Any action that might change a file or database—anything that could be permanent—should be reversible, whereas transient or view-related states, such as selecting between tabs, often are not. Specifically, these kinds of changes are expected to be reversible in most applications:

- Text entry for documents or spreadsheets
- Database transactions
- Modifications to images or painting canvases
- Layout changes—position, size, stacking order, or grouping—in graphics applications
- File operations, such as deleting or renaming files
- Creation, deletion, or rearrangement of objects such as email messages or spreadsheet columns
- Any cut, copy, or paste operation

The following kinds of changes are generally untracked in the action history and are not reversible. Navigation actions are a good example of this kind of ribbon-like, nonreversible action.

- Text or object selection
- Navigation between windows or pages
- Mouse cursor and text cursor locations
- Scroll bar position
- Window or panel positions and sizes
- Changes made in an uncommitted or modal dialog box

Some operations are on the borderline. Form fill-in, for instance, is sometimes reversible and sometimes not. However, if tabbing out of a changed field automatically commits that change, it's probably a good idea to explore making it reversible.

> **Note**
>
> Certain kinds of operations are impossible to undo, but usually the nature of the application makes that obvious to users with any experience at all. Impossible undos include the purchase step of an ecommerce transaction, posting a message to a forum or chat room, or sending an email—as much as we'd sometimes like that to be undoable!

In any case, make sure the reversible operations make sense to the user. Be sure to define and name them in terms of how the user thinks about the operations, not how the computer thinks about them. You should be able to undo a block of typed text, for instance, in chunks of words, not letter by letter.

Design an undo or action history stack. Each operation goes on the top of the action history stack as it is performed. Each undo reverses the operation at the top (the most recent action) first, then the next one below it (the next most recent), then the next, and so on. Redo works its way back up the stack step by step.

The stack should be at least 10 to 12 items long to be the most useful, and longer if you can manage it. Long-term observation or usability testing can tell you what your usable limit is. (Constantine and Lockwood assert that having more than a dozen items is usually unnecessary because "users are seldom able to make effective use of more levels."[4] Expert users of high-powered software might tell you differently.)

Presentation. Finally, decide how to present the undo stack to the user. Most desktop applications put Undo/Redo items on the Edit menu. Also, Undo is usually hooked up to Ctrl-Z or its equivalent. The best-behaved applications use *Smart Menu Items* to let the user know exactly which operation is next up on the undo stack.

Examples

Microsoft Word (Figure 8-26) shows a more typical presentation of *Multilevel Undo*. In this case, the user typed some text and then inserted a table. The first undo removes the table. When that's done, the following undo—the next action in the undo stack—represents the typed text, and invoking Undo again will remove that text. Meanwhile, the user has the opportunity to "undo the undo" by using the Redo menu item. If we're at the top of the stack (as in the first screenshot), there is no Redo, and that menu item is overloaded with a Repeat action. This is a complicated abstract

4 Constantine, Larry L., and Lucy A.D. Lockwood. "Instructive Interaction: Making Innovative Interfaces Self-Teaching." *User Experience*, vol. 1, no. 3, 2002. *Winter*, *https://oreil.ly/QMNpz*.

concept to try to bring to life in an interface. If you attempt something this, add a scenario or two in your help system to explain more fully how it works.

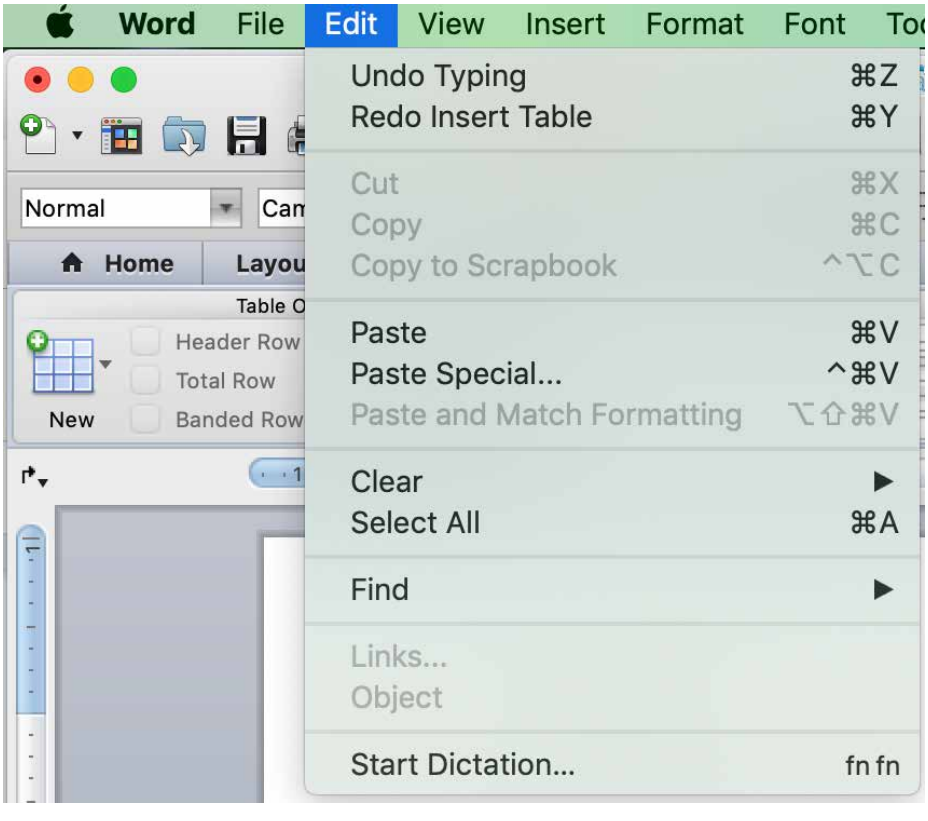


Figure 8-26. *Microsoft Word History of Actions*

Although it is not visible, Word keeps an undo stack in memory. This allows the user to select Undo multiple times to return to a previous state of the file. When the user reverses their most recent action with the Undo command, the first item in the Edit smart menu changes to the next action in the undo stack.

Most users will never develop a clear mental picture of the algorithms being used here; most people don't know what a "stack" is, let alone how it is used in conjunction with Repeat and Redo. That's why the *Smart Menu Items* are absolutely critical to usability here. They explain exactly what's going to happen, which reduces the cognitive burden on the user. What is important is that they see that they can back up and move forward again through the sequence of their recent actions.

Command History

What

Undoable actions. As the user performs actions, keep a visible record of those actions —what was done to what, and when. This is a list or record of the steps that the user took. This list is visible, and can be manipulated by the user, applying or removing or changing the sequence of these actions. Usually this is in conjunction with a file, photo, or other digital object that is being changed by these commands.

Browser history. As the user browses the internet browsers keep a visible record of the sites, apps and URLs they visit. This is more like a log file. This data can be searched for keywords in the URL string, or browsed by date. This is useful for finding a site that the user visited before, but can't remember the exact URL.

Use when

Users perform long and complex sequences of actions, either with a GUI or a command line. Most users are fairly experienced, or if not, they at least want an efficient interface that's supportive of long-term and recurring work. Graphical editors and programming environments are usually good candidates.

Why

Sometimes, a user needs to remember or review what they did in the course of working with the software. For instance, they might want to do any of these things:

- Repeat an action or command done earlier, which they don't remember well
- Recall the order in which some actions were done
- Repeat a sequence of operations, originally done to one object, on a different object
- Keep a log of their actions, for legal or security reasons
- Convert an interactive series of commands into a script or macro (see the *Macros* pattern in this chapter)

How

Keep a running list of the actions taken by the user. If the interface is driven from a command line, you have it easy—just record everything typed there. If you can, keep track of the history across sessions so that the user can see what was done even a week ago or longer.

If it's a GUI, or a combination of graphic and command-line interfaces, things become a little more complicated. Find a way to express each action in one consistent, concise way, usually with words (though there's no reason why it can't be done visually). Make sure you define these with the right granularity—if one action is done *en masse* to 17 objects, record it as one action, not 17.

What commands should be recorded, and what shouldn't? See the *Multilevel Undo* pattern for a thorough discussion of what commands should "count." If a command is undoable, it should be recorded in the history, too.

Finally, display the history to the user. That display should be optional in most software, because it will almost certainly play a supporting role in the user's work, not a starring role. Lists of commands—oldest to newest—tend to work well. If you'd like, you could timestamp the history display somehow.

Examples

Google's Chrome browser (Figure 8-27), like all browsers, keeps a history of the websites and web applications that the user visits. The user can view, search, and browse it, which allows the user to go back to a URL they went to before, or share an item from the list. Google Chrome History screen, although not strictly a history of actions, is a history file from the user's browsing history. This allows for predictive text options when typing in a URL if the URL or file is already in the history. The user can also search the history file or manually select a URL to go back to a previously visited URL.

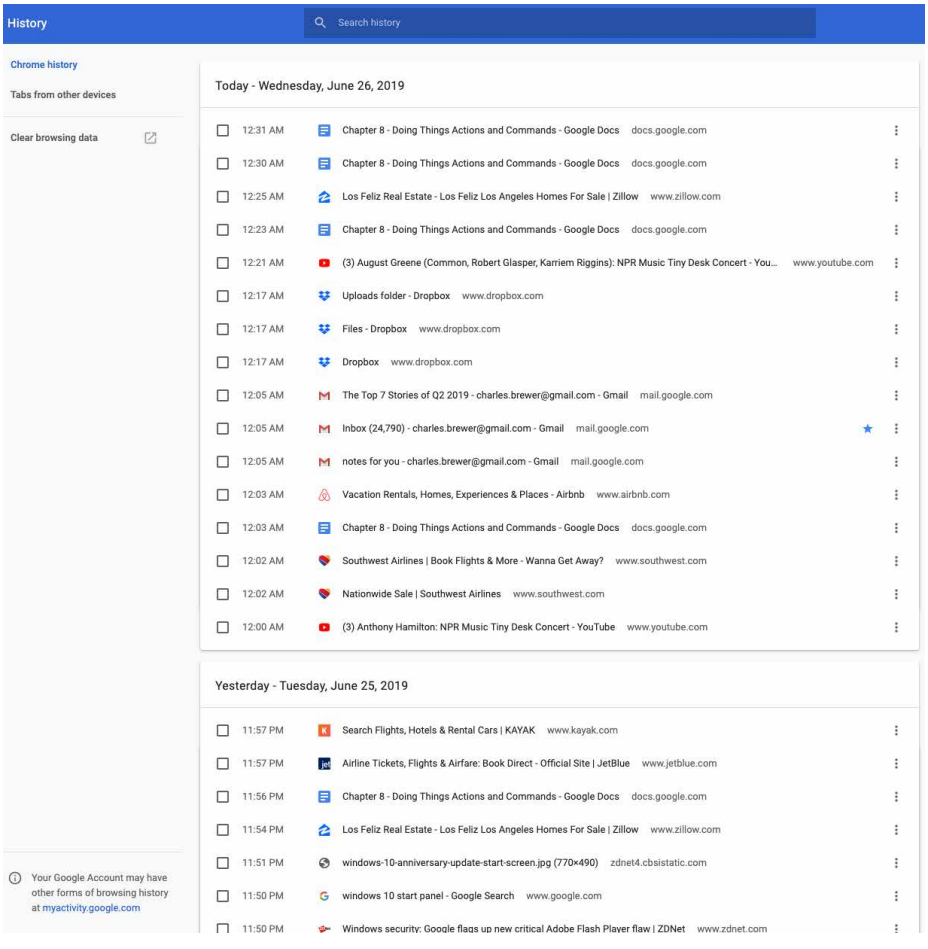


Figure 8-27. *Google Chrome History screen*

Adobe Photoshop CC's undo stack (Figure 8-28) is effectively a command history. You can use it to undo the actions you performed, but you don't have to; you can also just look at it and scroll through it, reviewing what you did. It uses icons to identify different classes of actions, which is unusual, but nice to use. This bona fide history of actions is a long-established feature of Photoshop. Each tool, action, filter, or other command is recorded in a chronological list. This is visible in the History palette (in the lower left of the figure). More than a simple history for undo, the History feature allows the user to selectively turn actions on or off or rearrange their sequence, which affects the current image.

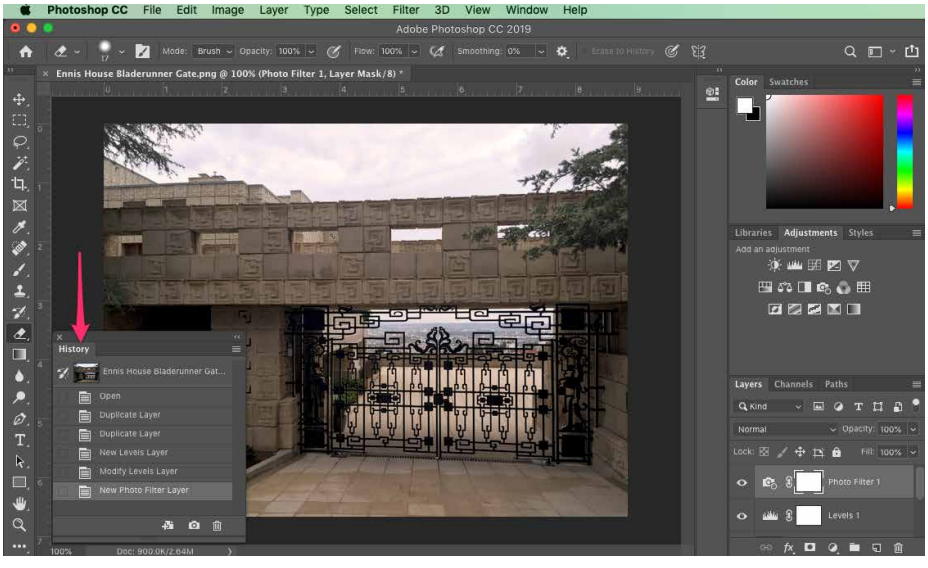


Figure 8-28. *Adobe Photoshop CC*

Macros

What

Macros are single actions composed of other, smaller actions. Users can create them by recording or putting together sequences of actions. These can be saved for reuse, by themselves or in a sequence of other commands. This can enable huge time savings and workflow efficiencies.

Use when

Users might want to repeat long sequences of actions or commands. They might want to loop over lists of files, images, database records, or other objects, for instance, doing the same things to each object. You might already have implemented *Multilevel Undo* or *Command History*.

Why

No one wants to perform the same set of repetitive interactive tasks over, and over, and over again! This is exactly what computers are supposed to be good at. Chapter 1 described a user-behavior pattern called *Streamlined Repetition*; macros are precisely the kind of mechanism that can support that well.

Macros obviously help users work faster. But by reducing the number of commands or gestures needed to get something done, they also reduce the possibility of finger slips, oversights, and similar mistakes.

You might also recall the concept of "flow," also discussed in Chapter 1. When a long sequence of actions can be compressed down into a single command or keyboard shortcut, the experience of flow is enhanced—the user can accomplish more with less effort and time, and they can keep their larger goals in sight without becoming bogged down in the details.

How

Provide a way for the user to "record" a sequence of actions and easily "play them back" at any time. The playback should be as easy as giving a single command, pressing a single button, or dragging and dropping an object.

Defining the macro. The user should be able to give the macro a name of their choice. Let them review the action sequence somehow so that they can check their work or revisit a forgotten sequence to see what it did (as in the *Command History* pattern). Make it possible for one macro to refer to another so that the user can build on each other.

Users will certainly want to save macros from one day to the next, so make sure that those macros can be saved to files or a database. Present them in a searchable, sortable, and even categorizable list, depending on the needs of your users.

Running the macro. The macro itself could be played back literally, to keep things simple; or, if it acts upon an object that can change from one invocation to another, you could allow the sequence to be parameterized (e.g., use a placeholder or variable instead of a literal object). Macros should also be able to act on many things at once.

How the names of the macros (or the controls that launch them) are presented depends heavily upon the nature of the application, but consider putting them with built-in actions rather than making them second-class citizens.

The ability to record these sequences—plus the facility for macros to build on one other—create the potential for the user to invent an entirely new linguistic or visual grammar, a grammar that is finely tuned to their own environment and work habits. This is a very powerful capability. In reality, it's programming; but if your users don't think of themselves as programmers, don't call it that or you'll scare them off. ("I don't know how to program anything; I must not be able to do this.")

Examples

Adobe Photoshop CC (Figure 8-29) has extensive macro capabilities. The primary one is the ability to create or record complex, multi step image edit and transformation commands. These are called Actions in Photoshop. Actions greatly speed up image workflows through automation and reuse. In this example, on the left, an existing action called Sepia Toning is selected to show the multiple, nested steps that all happen in sequence within this action. On the right, the Actions menu shows the options for recording, editing, and fine-tuning complicated, multi step actions.

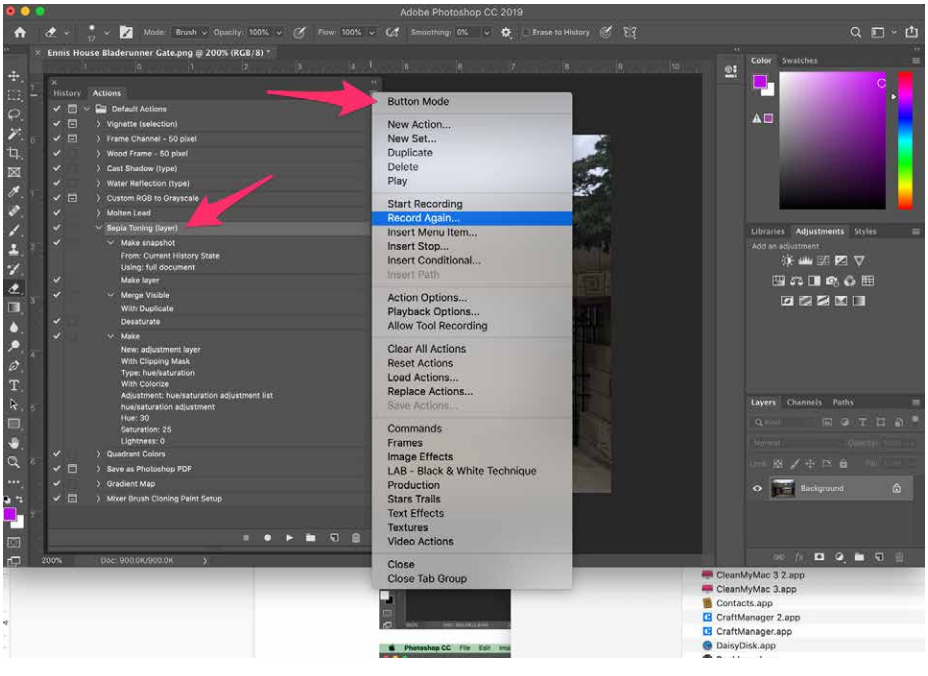


Figure 8-29. *Recording a macro in Adobe Photoshop CC*

Adobe Photoshop CC (Figure 8-30) shows the Batch automation menu and dialog box in Photoshop. This is another macro builder. These are user-created workflow scripts that direct Photoshop to open images from one location, apply saved actions, and save the images with specified naming in a different location. This creates even more time savings because the user does not need to open each image by hand to apply the action macro. In this way the user can quickly and automatically process very large numbers of files, saving huge amounts of manual effort.

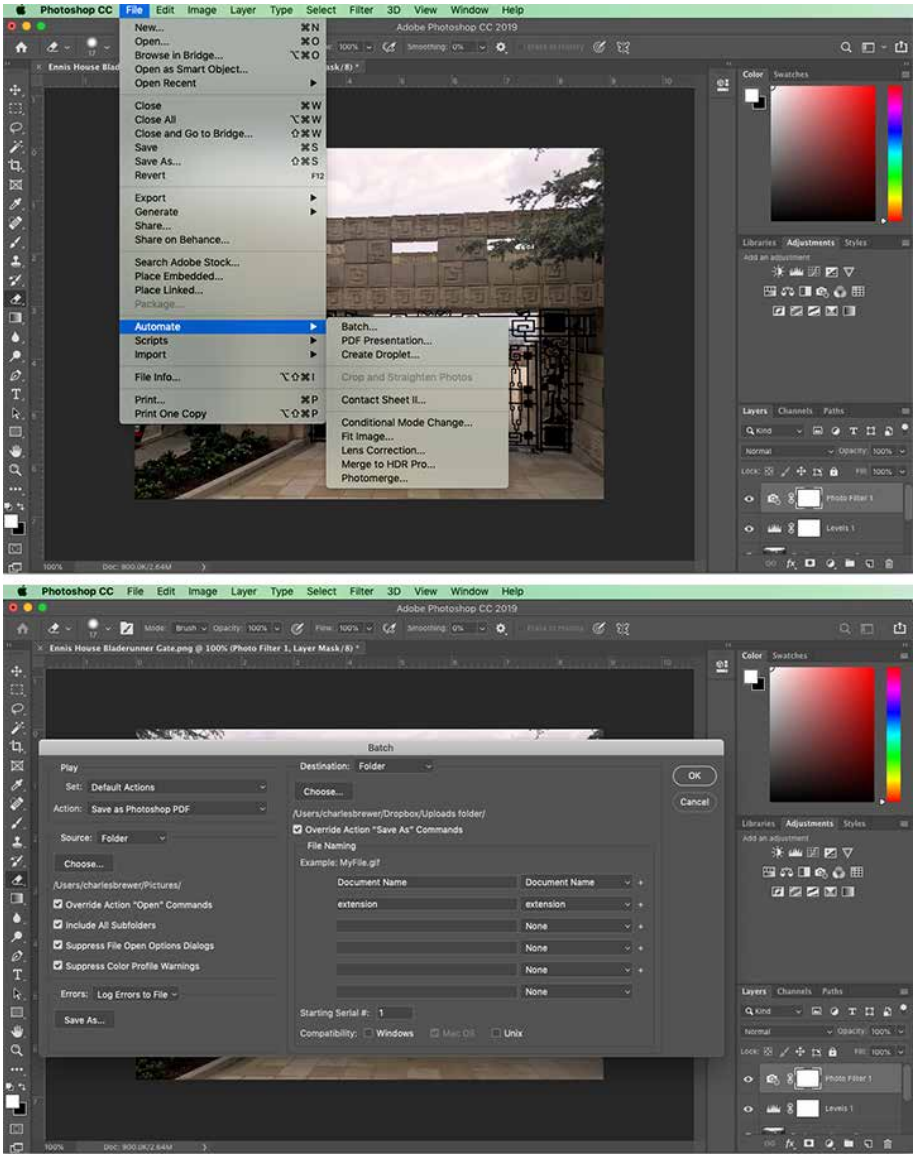


Figure 8-30. *Batch automation: configuring a series of actions to perform on multiple files automatically*

It is now possible to integrate and script different web apps, services, and platforms as if they were one application. IFTTT (If This, Then That) (Figure 8-31) is a web application for doing this. Third-party software companies that have API access and have integrated with the IFTTT platform become available for use. A given customer can provide their third-party logins to IFTTT and begin connecting their disparate web apps with macros, called "recipes" in IFTTT, to do work.

Here are some examples of what IFTTT recipes can do:

- Synchronize the profile photo across all your social media accounts
- Automatically back up image files from social media to a cloud storage account
- Turn home automation devices on/off based on your mobile
- Save social media posts to a cloud spreadsheet
- Save fitness data from devices to a cloud spreadsheet

IFTTT recipes are built by providing your login credentials to your online accounts and then building simple macros using the IFTTT web app (Figure 8-31). The large phrase in the screen "if [Twitter] then [+] that" is a macro in the process of being created. The first part is ready. This account has integrated with their Twitter account(s). Selecting the Twitter icon opens a different screen for configuring the Twitter-driven trigger for the macro. The next step is to configure the "that" step. For example, log each tweet to a Google spreadsheet. These are the actions IFTTT or other integrated internet service should execute. These macros allow for custom automated business processes that integrate unconnected web apps and services.

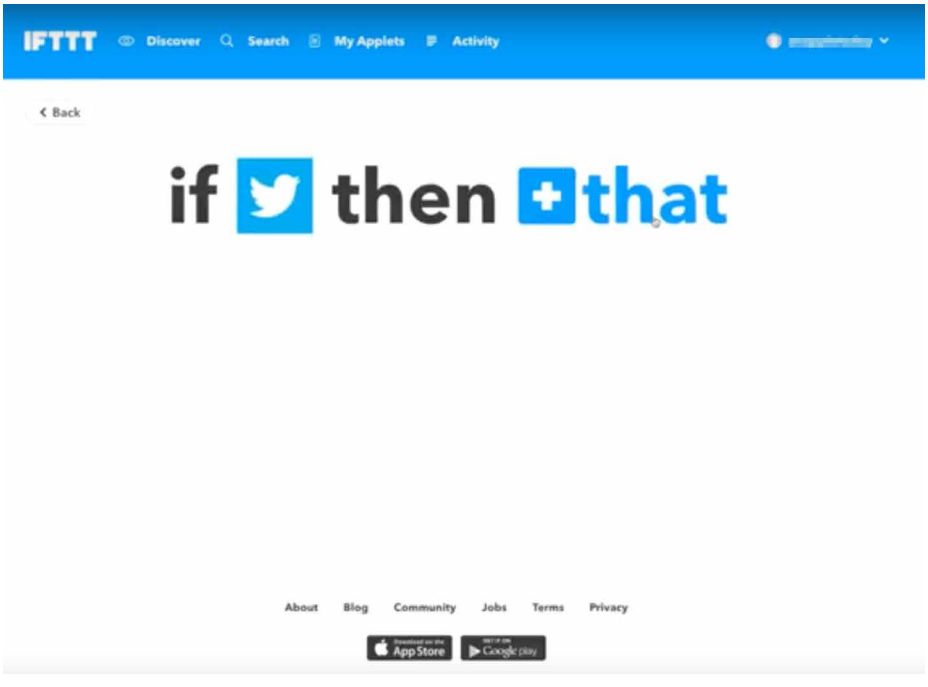


Figure 8-31. *IFTTT (If This, Then That) applet creator*

Microsoft Excel (Figure 8-32) allows macros to be recorded, named, stored along with the document, and even assigned to a keyboard shortcut. In this example, the user can record a macro and later edit it in a programming environment (a lightweight Visual Basic editor included in Excel). The user records macros to process data and manipulate spreadsheets within Excel. These can be edited and saved for reuse.

If you are developing a truly scriptable application, the lesson from Excel is to think about how such macros could be abused. You might want to put constraints around what can be done via macros.

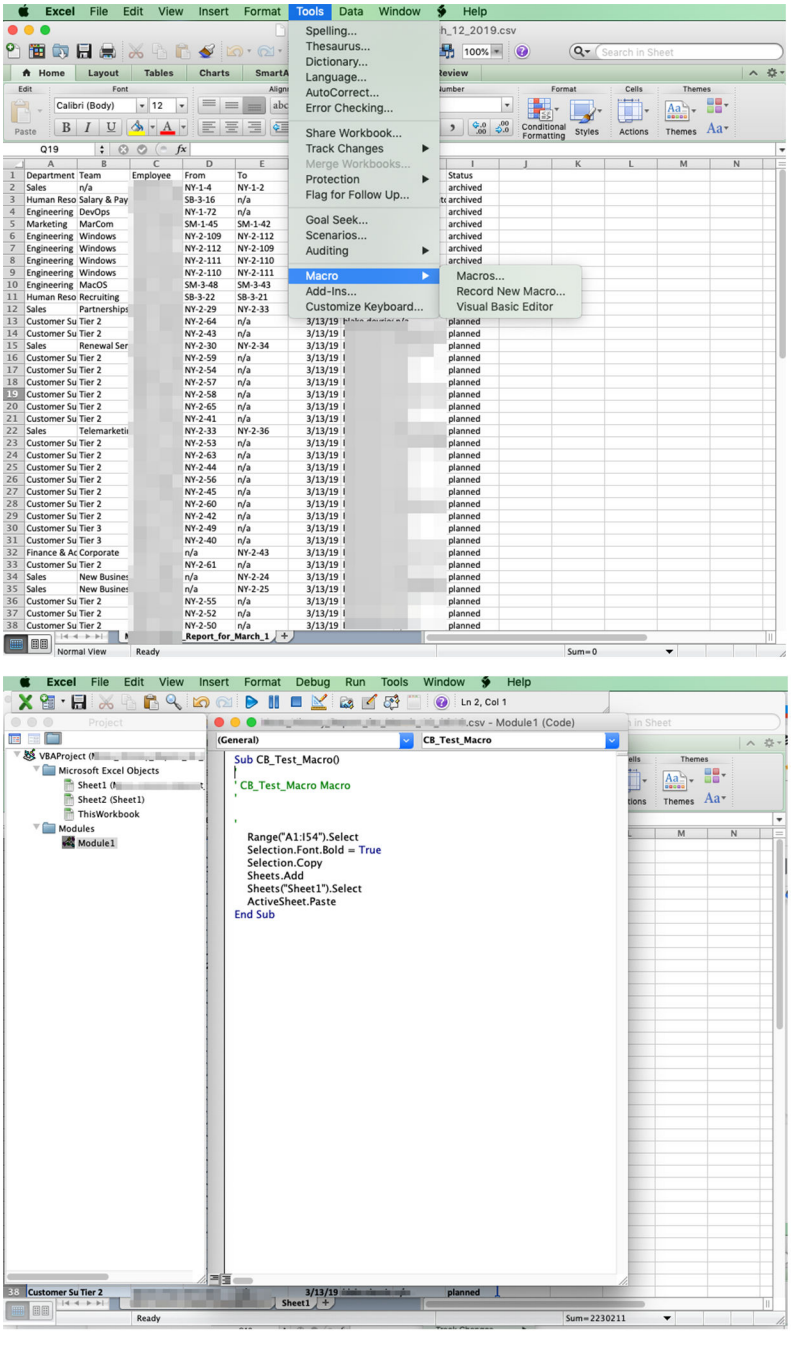


Figure 8-32. *Microsoft Excel*

Conclusion

In this chapter, we examined the different modes and means of taking action or initiating commands in software. As an interaction designer, you have a number of patterns and best practices to help users see and understand what they can do and what's going on. The important point is that you want to make the most important actions visible. To accomplish this, you can use the graphic design methods discussed in this chapter. The benefits of making actions clear is that you can guide new and existing users to the preferred next step. Patterns like *Preview* and *Multilevel Undo* help users to avoid error. They will learn the software more quickly, too.

Don't underestimate the positive feelings that come from giving people the ability to play around safely with your interface (that is, they stay in control of what they're doing because the understand how to preview, initiate, cancel and reverse actions.) For more advanced users or in complicated interactions, designing the ability to record actions as if on a timeline, with the ability to go forward and back in time, is a powerful level of control over actions. Finally, you might want to consider designing macro-like abilities into your software for users who need the speed and efficiency of executing multiple actions automatically and programmatically.

Chapter 9

Showing Complex Data

Done well, information graphics—including maps, tables, and graphs—communicate knowledge visually rather than verbally in an elegant and magical way. They let people use their eyes and minds to draw their own conclusions; they show, rather than tell. Visualizing data is an art and a science. Data visualization is a specialization within the discipline of design and requires subject matter expertise in addition to a strong visual design sensibility.

Creating artful visualizations is a discipline unto itself. This chapter discusses the basics of information graphics and shows some of the most common ways to display data in a mobile application or website.

The Basics of Information Graphics

Information graphics simply means data presented visually, with the goal of imparting knowledge to the user. We include tables and tree views in that description because they are inherently visual, even though they're constructed primarily from text instead of lines and polygons. Other familiar static information graphics include maps, flowcharts, bar plots, and diagrams of real-world objects.

But we're dealing with computers, not paper. You can make almost any good static design better with interactivity. Interactive tools let the user hide and show information as they need it, and they put the user in control, allowing them to choose how to view and explore that information.

Even the mere act of manipulating and rearranging the data in an interactive graphic has value—the user becomes a participant in the discovery process, not just a passive observer. This can be invaluable. The user might not end up producing the world's best-designed plot or table, but the process of manipulating that plot or table puts the user face to face with aspects of the data that they might never have noticed on paper.

Ultimately, the user's goal in using information graphics is to learn something. But the designer needs to understand *what* the user needs to learn. The user might be looking for something very specific, such as a particular street on a map, in which case they need to be able to find it—for instance, by directly searching or by filtering out extraneous information. The user needs to get a "big picture" only to the extent necessary to reach that specific data point. The ability to search, filter, and zero in on details is critical.

On the other hand, the user might be trying to learn something less concrete. They might look at a map to grasp the layout of a city rather than to find a specific address. Or, they might be a scientist visualizing a biochemical process, trying to understand how it works. Now, overviews are important; users need to see how the parts interconnect with the whole. They might want to zoom in, zoom back out again, look at the details occasionally, and compare one view of the data to another.

Good interactive information graphics offer users answers to these questions:

- How is this data organized?
- What's related to what?
- How can I explore this data?
- Can I rearrange this data to see it differently?
- How can I see only the data that I need?
- What are the specific data values?

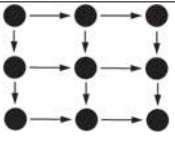
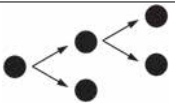
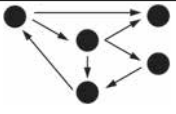
In these sections, keep in mind that the term *information graphics* is a very big umbrella. It covers plots, graphs, maps, tables, trees, timelines, and diagrams of all sort; the data can be huge and multilayered, or small and focused. Many of these techniques apply surprisingly well to graphic types that you wouldn't expect.

Before describing the patterns themselves, let's set the stage by talking about some of the questions posed in the previous list.

Organizational Models: How Is This Data Organized?

The first thing a user sees in any information visualization is the shape you've chosen for the data. Ideally, the data itself has an inherent structure that suggests this shape to you. Table 9-1 shows a variety of organizational models. Which of these best fits your data?

Table 9-1. Organizational models

Model	Diagram	Common graphics
Linear		List, single-variable plot
Tabular		Spreadsheet, multicolumn list, *sortable table*, *Multi-Y Graph*, other multivariable plots
Hierarchical		Tree, list, *tree table*
Network of interconnections		Directed graph, flowchart
Geographic (or spatial)		Map, schematic, scatter plot
Textual		Word cloud, directed graph
Other		Plots of various sorts, such as parallel coordinate plots, *treemap*, etc.

Try these out against the data you're trying to show. If two or more might fit, consider which ones play up which aspects of your data. If your data could be both geographic and tabular, for instance, showing it as only a table might obscure its geographic nature—a viewer might miss interesting features or relationships in the data if it's not shown as a map, too.

Preattentive Variables: What's Related to What?

The organizational model you choose tells the user about the shape of the data. Part of this message operates at a subconscious level; people recognize trees, tables, and maps, and they immediately make some assumptions about the underlying data before they even begin to think consciously about it. But it's not just the shape that does this. The look of the individual data elements also works at a subconscious level in the user's mind: things that look alike must be associated with one another.

If you've read Chapter 4, that should sound familiar—you already know about the Gestalt principles. (If you jumped ahead in the book, this might be a good time to go back and read the introduction to Chapter 4.) Most of those principles, especially *similarity* and *continuity*, will come into play here, too. Let's look a little more at how they work.

Certain visual features operate *preattentively*: they convey information before the viewer pays conscious attention. Take a look at Figure 9-1 and find the blue objects.

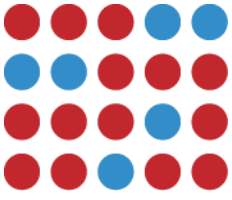


Figure 9-1. *Find the blue objects*

I'm guessing that you can do that pretty quickly. Now look at Figure 9-2 and do the same.

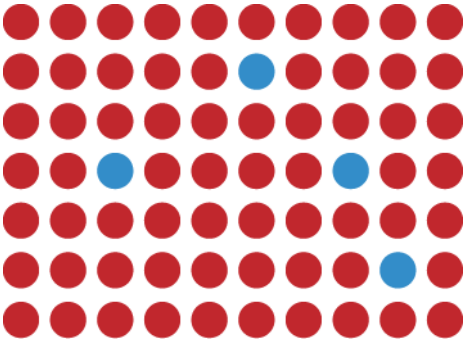


Figure 9-2. *Find the blue objects again*

You did that pretty quickly too, right? In fact, it doesn't matter how many red objects there are; the amount of time it takes you to find the blue ones is constant! You might think it should be linear with the total number of objects—order-*N* time, in algorithmic terms—but it's not. Color operates at a primitive cognitive level. Your visual system does the difficult work for you, and it seems to work in a "massively parallel" fashion.

On the other hand, visually monotonous text forces you to read the values and think about them. Figure 9-3 shows exactly the same problem with numbers instead of colors. How fast can you find the numbers that are greater than one?

0.103	0.176	0.387	0.300	0.379	0.276	0.179	0.321	0.192	0.250
0.333	0.384	0.564	0.587	0.857	1.064	0.698	0.621	0.232	0.316
0.421	0.309	0.654	0.729	0.228	0.529	0.832	0.935	0.452	0.426
0.266	0.750	1.056	0.936	0.911	0.820	0.723	1.201	0.935	0.819
0.225	0.326	0.643	0.337	0.721	0.837	0.682	0.987	0.984	0.849
0.187	0.586	0.529	0.340	0.829	0.835	0.873	0.945	1.103	0.710
0.153	0.485	0.560	0.428	0.628	0.335	0.956	0.879	0.699	0.424

Figure 9-3. *Find the values greater than one*

When dealing with text such as this, your "search time" really is linear with the number of items. But what if we still use text, but make the target numbers physically larger than the others, as in Figure 9-4?

0.103	0.176	0.387	0.300	0.379	0.276	0.179	0.321	0.192	0.250
0.333	0.384	0.564	0.587	0.857	1.064	0.698	0.621	0.232	0.316
0.421	0.309	0.654	0.729	0.228	0.529	0.832	0.935	0.452	0.426
0.266	0.750	1.056	0.936	0.911	0.820	0.723	1.201	0.935	0.819
0.225	0.326	0.643	0.337	0.721	0.837	0.682	0.987	0.984	0.849
0.187	0.586	0.529	0.340	0.829	0.835	0.873	0.945	1.103	0.710
0.153	0.485	0.560	0.428	0.628	0.335	0.956	0.879	0.699	0.424

Figure 9-4. *Find the values greater than one again*

Now we're back to constant time again. Size is, in fact, another preattentive variable. The fact that the larger numbers protrude into their right margins of their respective columns also helps you find them—alignment is yet another preattentive variable.

Figure 9-5 shows many known preattentive variables.

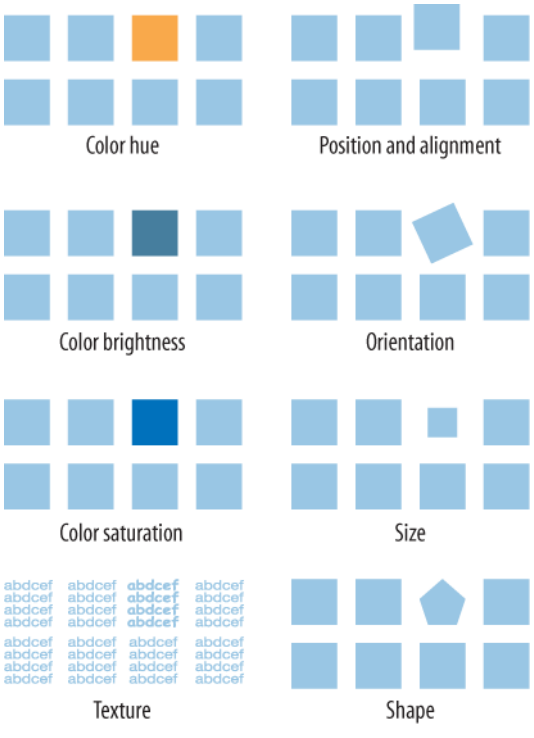


Figure 9-5. *Eight preattentive variables*

This concept has profound implications for text-based information graphics, like the table of numbers shown earlier in Figure 9-3. If you want some data points to stand out from the others, you need to make them look different by varying their color, size, or some other preattentive variable. More generally, you can use these variables to differentiate classes or dimensions of data on any kind of information graphics. This is sometimes called *encoding*.

When you need to plot a multidimensional data set, you can use several different visual variables to encode all those dimensions in a single static display. Consider the scatter plot shown in Figure 9-6. Position is used along the x- and y-axes; color hue encodes a third variable. The shape of the scatter markers could encode yet a fourth variable, but in this case, shape is redundant with color hue. The redundant encoding helps a user visually separate the three data groups.

All of this is related to a general graphic design concept called *layering*. When you look at well-designed graphics of any sort, you perceive different classes of information on the screen. Preattentive factors such as color cause some of them to "pop" out

of the screen, and similarity causes you to see those as connected to each other, as though each were on a transparent layer over the base graphics. It's an extremely effective way of segmenting data—each layer is simpler than the whole graphic, and the viewer can study each in turn, but the relationships among the whole are preserved and emphasized.

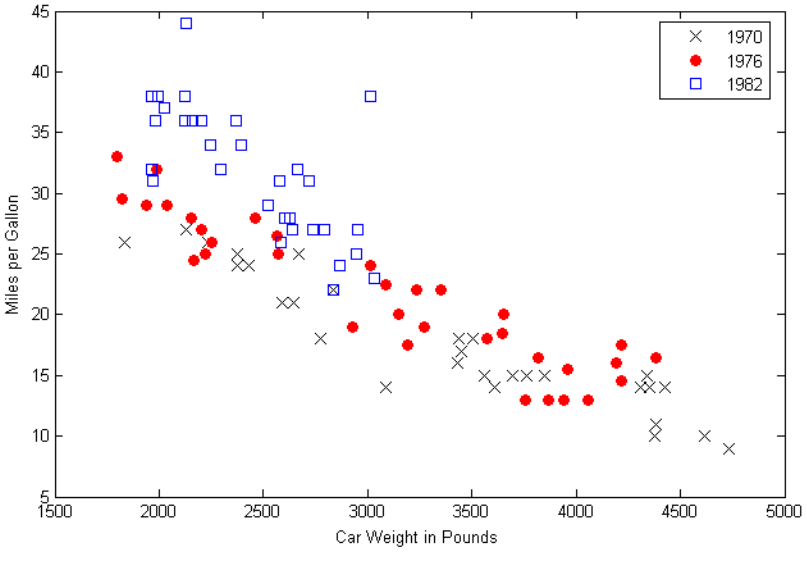


Figure 9-6. *Encoding three variables in a scatter plot*

Navigation and Browsing: How Can I Explore This Data?

A user's first investigation of an interactive data graphic might be browsing—just looking around to see what's there. The user might also navigate through it to find some specific thing they're seeking. Filtering and searching can serve that purpose, too, but navigation through the "virtual space" of a data set is often better. The user can see points of interest in context with the rest of the data.

There's a famous mantra in the information visualization field: "Focus plus context." A good visualization should permit a user to focus on a point of interest, while simultaneously showing enough stuff around that point of interest to give the user a sense of where it is in the big picture. Here are some common techniques for navigation and browsing:

Scroll and pan

If the entire data display won't fit on the screen at once, you could put it in a scrolled window, giving the user easy and familiar access to the off-screen portions. Scroll bars are familiar to almost everyone and are easy to use. However,

some displays are too big, or their size is indeterminate (thus making scroll bars inaccurate), or they have data beyond the visible window that needs to be retrieved or recalculated (thus making scroll bars too slow to respond). Instead of using scroll bars in those cases, try setting up buttons that the user must click to retrieve the next screenful of data. Other applications do panning instead, in which the information graphic is "grabbed" with the cursor and dragged until the point of interest is found, like in Google Maps.

These are appropriate for different situations, but the basic idea is the same: to interactively move the visible part of the graphic. Sometimes, an overview next to a detail view can help the user stay oriented. A small view of the whole graphic can be shown with an indicator rectangle showing the visible "viewport"; the user might pan by dragging that rectangle, in addition to using scroll bars or however else it's done.

Zoom

Whereas scrolling changes the location being viewed, zooming changes the *scale* of the section being viewed, When you present a data-dense map or graph, consider offering the user the ability to zoom in on points of interest. It means you don't need to pack every single data detail into the full view—if you have lots of labels, or very tiny features (especially on maps), that might be impossible anyway. As the user zooms in, those features can emerge when there is enough space.

Most zooms are triggered with a pinch, mouse click, or button press, and the entire viewing area changes scale at once. But that's not the only way to zoom. Some applications create nonlinear distortions of the information graphic as the user moves the mouse pointer over the graphic: whatever is under the pointer is zoomed, but the stuff far away from the pointer stays the same scale.

Open and close points of interest

Tree views typically let users open and close parent items at will, so they can inspect the contents of those items. Some hierarchically structured diagrams and graphs also give users the chance to open and close parts of the diagram "in place," without having to open a new window or go to a new screen. With these mechanisms, the user can explore containment or parent–child relationships easily, without leaving that window.

Drill down into points of interest

Some information graphics just present a "top level" of information. A user might click or double-click on a map to see information about the city they just clicked, or they might tap key points in a diagram to see sub diagrams. This "drilling down" might reuse the same window, use a separate panel on the same window, or bring up a new window. This technique is similar to opening and closing points of interest, except that the viewing occurs separately from the graphic and is not integrated into it.

If you also provide a search facility for an interactive information graphic, consider linking the search results to whichever of the aforementioned techniques is in use. In other words, when a user searches for the city of Sydney on a map, show the map zooming and/or panning to that point. The search user thus gets some of the benefits of context and spatial memory.

Sorting and Rearranging: Can I Rearrange This Data to See It Differently?

Sometimes, just rearranging an information graphic can reveal unexpected relationships. Look at Figure 9-7, taken from the National Cancer Institute's online mortality charts. It shows the number of deaths from lung cancer in the state of Texas. The major metropolitan regions in Texas are arranged alphabetically—not an unreasonable default order if you're going to look up specific cities, but as presented, the data doesn't lead you to ask interesting questions. It's not clear why Abilene, Alice, Amarillo, and Austin all seem to have similar numbers, for instance; it might just be chance.

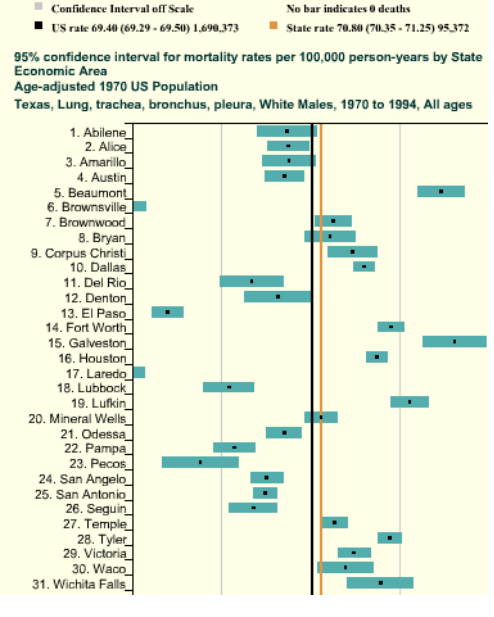


Figure 9-7. *Cancer data by city, sorted alphabetically*

But this chart lets you reorder the data into numerically descending order, as in Figure 9-8. Suddenly the graph becomes much more interesting. Galveston is ranked first—why is that, when its neighbor, Houston, is further down the scale? What's special about Galveston? (OK, you need to know something about Texas geography to

ask these questions, but you get my point.) Likewise, why the difference between neighbors Dallas and Fort Worth? And apparently the Mexico-bordering southern cities of El Paso, Brownsville, and Laredo have less lung cancer than the rest of Texas; why might that be? You can't answer these questions with this data set, but at least you can ask them.

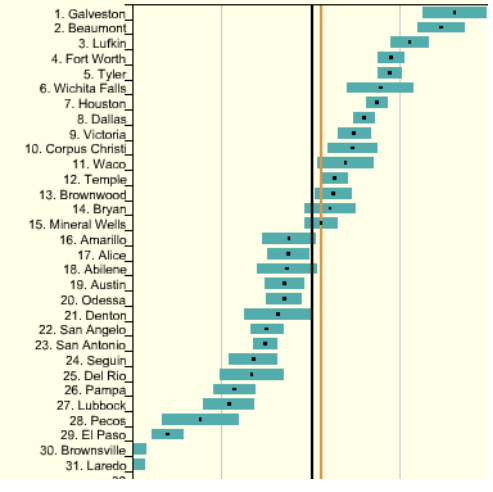


Figure 9-8. *The same Cancer data chart, sorted numerically*

People who can interact with data graphics this way have more opportunities to learn from the graphic. Sorting and rearranging puts different data points next to each other, thus letting users make different kinds of comparisons—it's far easier to compare neighbors than widely scattered points. And users tend to zero in on the extreme ends of scales, as I did in the preceding example.

How else can you apply this concept? A *sortable table* offers one obvious way: when you have a many-columned table, users might want to sort the rows according to their choice of column. This is pretty common. (Many table implementations also permit rearrangement of the columns themselves, by dragging.) Trees might allow reordering of their child nodes. Diagrams and connected graphs might allow spatial repositioning of their elements while retaining their connectivity.

Drawing on the information architecture approaches discussed in Chapter 2, consider these methods of sorting and rearranging:

- Alphabetically
- Numerically
- By date or time
- By physical location

- By category or tag
- By popularity—heavily used versus lightly used
- User-designed arrangement
- Completely random (you never know what you might see)

For a subtle example, take a look at Figure 9-9. Bar charts that show multiple data values on each bar (known as *stacked* bar charts) might also be amenable to rearranging—the bar segments nearest the baseline are the easiest to evaluate and compare, so you might want to let users determine which variable is next to the baseline.

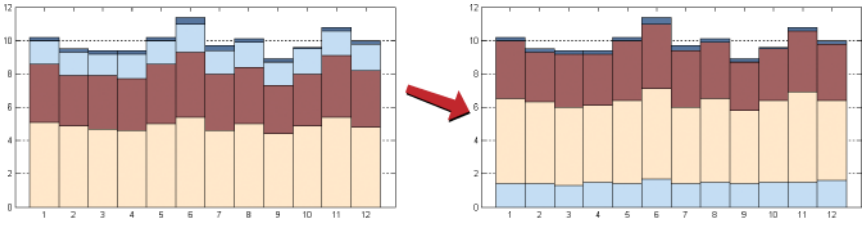


Figure 9-9. *Rearrangement of a stacked bar chart*

The light blue variable in this example might be the same height from bar to bar. Does it vary, and how? Which light blue bars are the tallest? You really can't tell until you move that data series to the baseline—that transformation lines up the bases of all those blue rectangles. Now a visual comparison is easy: light blue bars 6 and 12 are the tallest, and the variation seems loosely correlated to the overall bar heights.

Searching and Filtering: How Can I See Only the Data That I Need?

Sometimes you don't want to see an entire data set at once. You might start with the whole thing and narrow it down to what you need—filtering—or you might build up a subset of the data via searching or querying. Most users won't even distinguish between filtering and querying (though there's a big difference from, say, a database's point of view). The user's intent is the same: to zero in on whatever part of the data is of interest, and get rid of the rest.

The simplest filtering and querying techniques offer users a choice of which aspects of the data to view. Checkboxes and other one-click controls turn parts of the interactive graphic on and off. A table might show some columns and not others, per the user's choice; a map might show only the points of interest (e.g., restaurants) selected by the user. The *Dynamic Queries* pattern, which can offer very rich interaction, is a logical extension of simple filter controls such as these.

Sometimes, simply highlighting a subset of the data, rather than hiding or removing the rest, is sufficient. That way a user can see that subset in context with the rest of the

data. Interactively, you can do this with simple controls, as described earlier. The *Data Brushing* pattern describes a variation of data highlighting; it highlights the same data in several data graphics at once.

Look at Figure 9-10. This interactive ski trail map can show four categories of trails, coded by symbol, plus other features such as ski lifts and base lodges. When everything is "turned on" at once, it's so crowded that it's difficult to read anything. But users can click the trail symbols, as shown, to turn the data "layers" on and off. The screenshot on the left shows no highlighted trails; the one on the right switches on the trails rated black diamond with a single click.

Figure 9-10. *Interactive ski trail map*

Searching mechanisms vary heavily from one type of graphic to another. A table or tree should permit textual searches, of course; a map should offer searches on addresses and other physical locations; numeric charts and plots might let users search for specific data values or ranges of values. What are your users interested in searching on?

When the search is done and the results obtained, you might set up the interface to show the results in context, on the graphic—you could scroll the table or map so that the searched-for item is in the middle of the viewport, for instance. Seeing the results in context with the rest of the data helps the user understand the results better.

Here are the characteristics of the best filtering and querying interfaces:

Highly interactive
: They respond as quickly as possible to the user's searching and filtering. (Don't react to individual keystrokes if it significantly slows down the user's typing, however.)

Iterative
: They let a user refine the search, query, or filter until they get the desired results. They might also combine these operations: a user might do a search, get a screenful of results, and then filter those results down to what they want.

Contextual
: They show results in context with surrounding data, to make it easier for a user to understand where they are in a data space. This is also true for other kinds of searches, as it happens; the best text search facilities show the search terms embedded in sentences, for instance.

Complex
: They go beyond simply switching entire data sets on and off, and allow the user to specify nuanced combinations of conditions for showing data. For instance, can this information graphic show me all the items for which conditions X, Y, and Z are true, but A and B are false, within the time range M–N? Such complexity lets users test hypotheses about the data and explore the data set in creative ways.

The Actual Data: What Are the Specific Data Values?

Several common techniques help a viewer get specific values out of an information graphic. Know your audience—if they're interested in getting only a qualitative sense of the data, there's no need for you to spend large amounts of time or pixels labeling every little thing. But some actual numbers or text is usually necessary.

Because these techniques all involve text, don't forget the graphic design principles that will make the text look good: readable fonts, appropriate font size (not too big, not too small), proper visual separation between unrelated text items, alignment of related items, no heavy-bordered boxes, and no unnecessary obscuring of data. Here are some other aspects for you to consider:

Labels
: Many information graphics put labels directly on the graphic, such as town names on a map. Labels can also identify the values of symbols on a scatter plot, bars on a bar graph, and other things that might normally force the user to depend on axes or legends. Labels are easier to use. They communicate data values precisely and unambiguously (when placed correctly), and they're located in or beside the data point of interest—no going back and forth between the data point and a legend. The downside is that they clutter up a graphic when overused, so be careful.

Legends
: When you use color, texture, line style, symbols, or size on an information graphic to represent values (or categories or value ranges), the legend shows the

user what represents what. You should place the legend on the same screen as the graphic itself so the user's eyes don't need to travel far between the data and the legend.

Axes, rulers, scales, and timelines
: Whenever position represents data, as it does on plots and maps (but not on most diagrams), these tell the user what values those positions represent. They are reference lines or curves on which reference values are marked. The user has to draw an imaginary line from the point of interest to the axis, and maybe interpolate to find the right number. This is more of a burden on the user than direct labeling. But labeling clutters things when the data is dense, and many users don't need to derive precise values from graphics; they just want a more general sense of the values involved. For those situations, axes are appropriate.

Datatips
: The *Datatips* pattern (described in this chapter) are tool tips that show data values when the user hovers over a point of interest. They offer the physical proximity advantages of labels without the clutter. They work only in interactive graphics, though.

Data spotlight
: Like *Datatips*, a data spotlight highlights data when the user hovers over a point of interest. But instead of showing the specific point's value, it displays a "slice" of the data in context with the rest of the information graphic, often by dimming the rest of the data. See the *Data Spotlight* pattern.

Data brushing
: A technique called *data brushing* lets users select a subset of the data in the information graphic and see how that data fits into other contexts. You use this with two or more information graphics; for instance, selecting some outliers in a scatter plot causes those same data points to be highlighted in a table showing the same data. For more information, see the *Data Brushing* pattern in this chapter.

The Patterns

Because this book is about interactive interfaces, most of these patterns describe ways to interact with the data: moving through it; sorting, selecting, inserting, or changing items; and probing for specific values or sets of values. A few of them deal only with static graphics: information designers have known about *Multi-Y Graph* and *Small Multiples* for a while now, but they translate well to the world of digital interfaces.

You can apply the following patterns to most varieties of interactive graphic, regardless of the data's underlying structure (some are more difficult to learn and use than others, so don't throw them at every data graphic you create):

- *Datatips*
- *Data Spotlight*
- *Dynamic Queries*
- *Data Brushing*

The remaining patterns are ways to construct complex data graphics for multidimensional data—data that has many attributes or variables. They encourage users to ask different kinds of questions about the data, and to make different types of comparisons among data elements.

- *Multi-Y Graph*
- *Small Multiples*

Datatips

What

Data values appear when your finger or mouse rolls over a point of interest on an interactive data table or when you tap or click the icon.

The Pew Research chart (Figure 9-11) example shows *Datatips* in action.

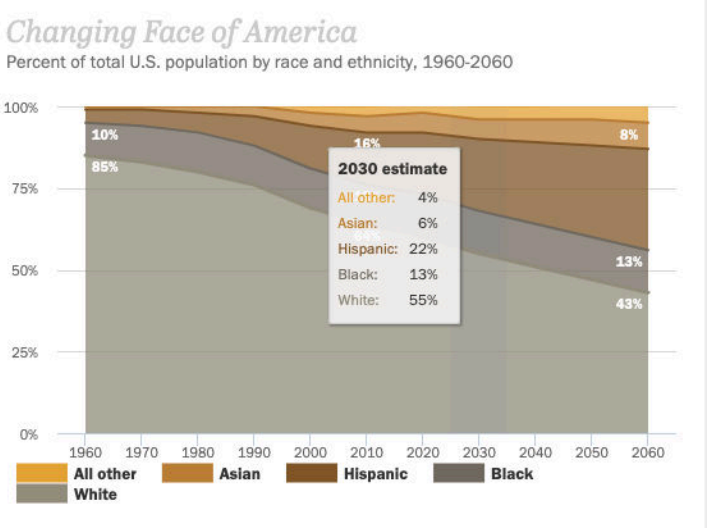


Figure 9-11. *Pew Research, Changing Face of America*

Use when

You're showing an overview of a data set, in almost any form. More data is "hidden behind" specific points on that graphic, such as the names of streets on a map or the values of bars in a bar chart. The user is able to "point at" places of interest with a mouse cursor or a touch screen.

Why

Looking at specific data values is a common task in data-rich graphics. Users will want the overview, but they might also look for particular facts that aren't present in the overview. *Datatips* let you present small, targeted chunks of context-dependent data, and they put that data directly where the user's attention is focused: the mouse pointer or fingertip. If the overview is reasonably well organized, users will find it easy to look up what they want, and you won't need to put it all on the graphic. *Datatips* can substitute for labels.

Also, some people might just be curious. What else is here? What can I find out? *Datatips* offer an easy, rewarding form of interactivity. They're quick (no screen loading), they're lightweight, and they offer intriguing little glimpses into an otherwise invisible data set.

How

Use a tool tip–like window to show the data associated with that point. It doesn't need to be technically a "tool tip"—all that matters is that it appears where the pointer is, it's layered atop the graphic, and it's temporary. Users will get the idea pretty quickly.

Inside that window, format the data appropriately. Denser is usually better, because a tool-tip window is expected to be small; don't let the window get so large that it obscures too much of the graphic while it's visible. And place it well. If there's a way to programmatically position it so that it covers as little content as possible, try that.

You might even want to format the *Datatips* differently depending on the situation. An interactive map might let the user toggle between seeing place names and seeing latitude/longitude coordinates, for example. If you have a few data sets plotted as separate lines on one graph, the *Datatips* might be labeled differently for each line, or have different kinds of data in them.

Many *Datatips* offer links that the user can click. This lets the user "drill down" into parts of the data that might not be visible at all on the main information graphic. The *Datatips* is beautifully self-describing—it offers not only information, but also a link and instructions for drilling down.

An alternative way of dynamically showing hidden data is to reserve some panel on or next to the graphic as a static data window. As the user rolls over various points on

the graphic, data associated with those points appears in the data window. It's the same idea, but using a reserved space rather than temporary *Datatips*. The user must shift their attention from the pointer to that panel, but you never have a problem with the rest of the graphic being hidden. Furthermore, if that data window can retain its data, the user can view it while interacting with something else.

In contemporary interactive infographics, *Datatips* often work in conjunction with a *Data Spotlight* mechanism . The spotlight shows a slice through the data—for example, a line or set of scattered points— whereas the *Datatips* shows the specific data point that's under the mouse pointer.

Examples

The CrimeMapping example (Figure 9-12) shows an icon to indicate what kind of crime has occurred and plots this point on a map. A user can zoom in and out of the map and filter the nature of the crime by selecting "What" from the panel on the left.

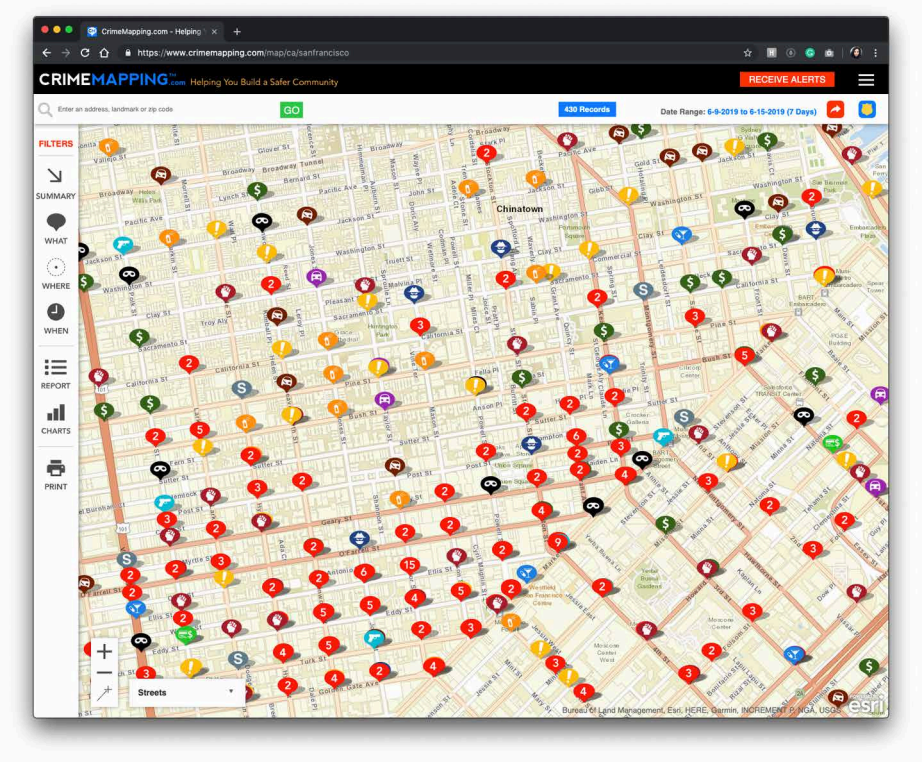


Figure 9-12. *CrimeMapping*

CrimeMapping uses both *Datatips* and a *Data Spotlight*. All incidents of theft are highlighted on the map (via the spotlight), but the *Datatips* pattern describes the particular incident at which the user is pointing. Note also the link to the raw data about this crime.

So many geographic information graphics are built upon Google Maps that it deserves a particular mention. The map can toggle between a simplified map or satellite view, and can have information like traffic, routes, and place markers overlaid on the map (Figure 9-13).

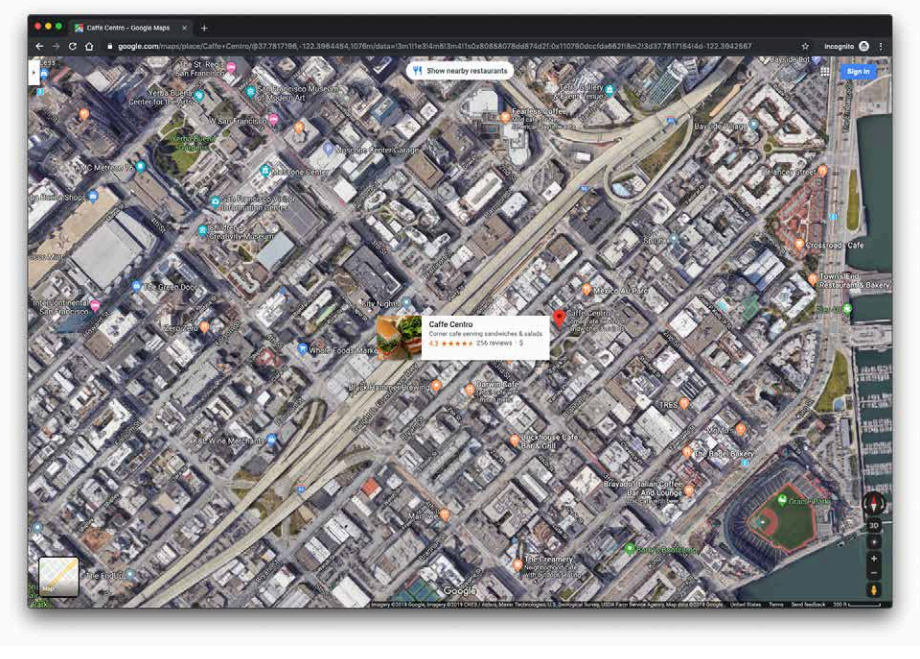


Figure 9-13. *Google Maps*

The Transit Mobile App (Figure 9-14) shows a route overlaid on a map with simplified route icons and *Datatips*. The colors, symbols and data change depending on what mode of transport is selected.

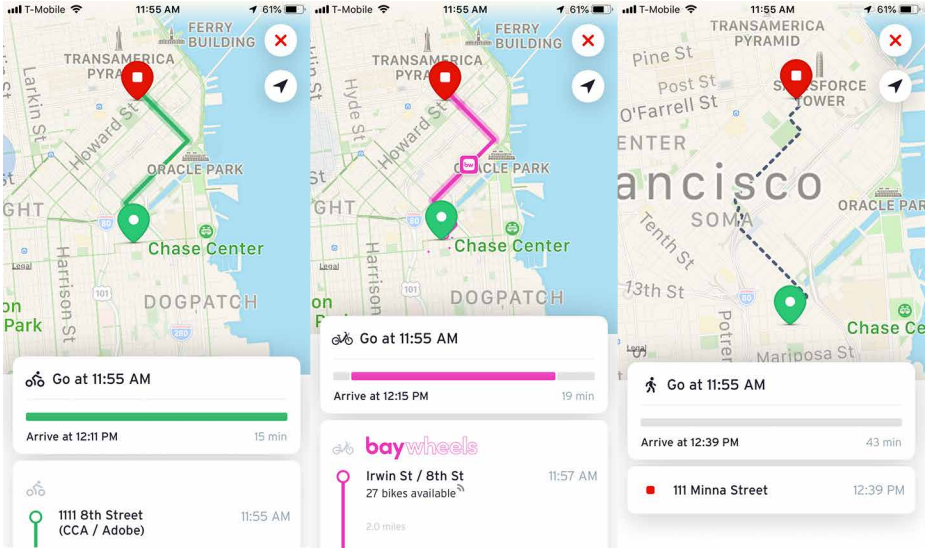


Figure 9-14. *Transit Mobile App*

Data Spotlight

What

As the user hovers over an area of interest, highlight that slice of data (graph line, map layer, etc.) and dim the rest.

The very beautiful Atlas of Emotions (Figure 9-15) shows a range of emotions and their intensity. When the user cursors over the data, it reveals associated emotions in that family.

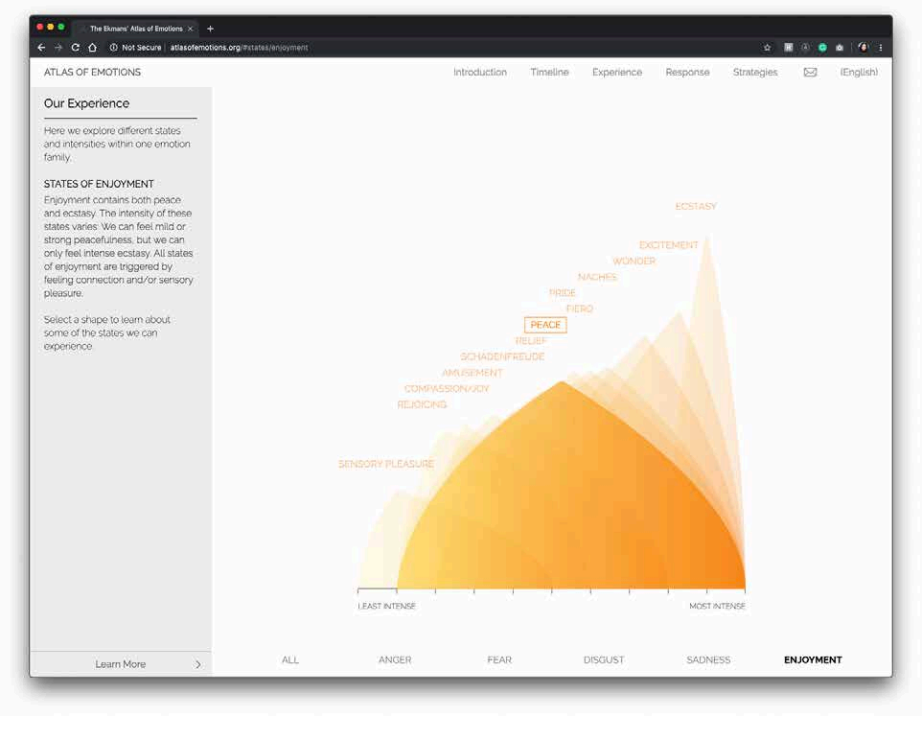


Figure 9-15. *Atlas of Emotions*

Use when

The graphic contains so much information that it tends to obscure its own structure. It might be difficult for a viewer to pick out relationships and trace connections among the data because of its sheer richness.

The data itself is structurally complex—it might have several independent variables and complicated "slices" of dependent data such as lines, areas, scattered sets of

points, or systems of connections. (If the rolled-over data is merely a point or a simple shape, *Datatips* is a better solution than *Data Spotlight*. They're often used in conjunction with each other, though.)

Why

A *Data Spotlight* untangles the threads of data from one another. It's one way that you can offer "focus plus context" on a complex infographic: a user eliminates some of the visual clutter on the graphic by quieting most of it, focusing only on the data slice of interest. However, the rest of the data is still there to provide context.

It also permits dynamic exploration by letting a user flick quickly from one data slice to another. The user can see both large differences and very small differences that way —as long as the *Data Spotlight* transitions quickly and smoothly (without flicker) from one data slice to another as the mouse moves, even very tiny differences will be visible.

Finally, *Data Spotlight* can be fun and engaging to use.

How

First, design the information graphic so that it doesn't initially depend on a *Data Spotlight*. Try to keep the data slices visible and coherent so that a user can follow what's going on without interacting with the graphic (someone might print it, after all).

To create a spotlight effect, make the spotlighted data either a light color or a saturated color while the other data fades to a darker or grayer color. Make the reaction very quick on rollover to give the user a sense of immediacy and smoothness.

Besides triggering a spotlight when the mouse rolls over data elements, you might also put "hot spots" onto legends and other references to the data.

Consider a "spotlight mode." In this, the *Data Spotlight* waits for a longer initial mouse hover before turning itself on. After it's in that mode, the user's mouse motions cause immediate changes to the spotlight. This means the spotlight effect won't be triggered accidentally, when a user just happens to roll the mouse over the graphic.

An alternative to the mouse rollover gesture is a simple mouse click or finger tap. This lacks the immediacy of rollovers, but it works on touch screens and it isn't as subject to accidental triggering. However, you might want to reserve the mouse click for a different action, such as drilling down into the data.

Use *Datatips* to describe specific data points, describe the data slice being highlighted, and offer instructions where necessary.

Examples

Here is an example of the data spotlight in action.

Winds and Words (Figure 9-16) is an interactive *Game of Thrones* data visualization. When a user clicks on a character, the other characters recede into the background and the selected character is shown along with their relationship to other characters.

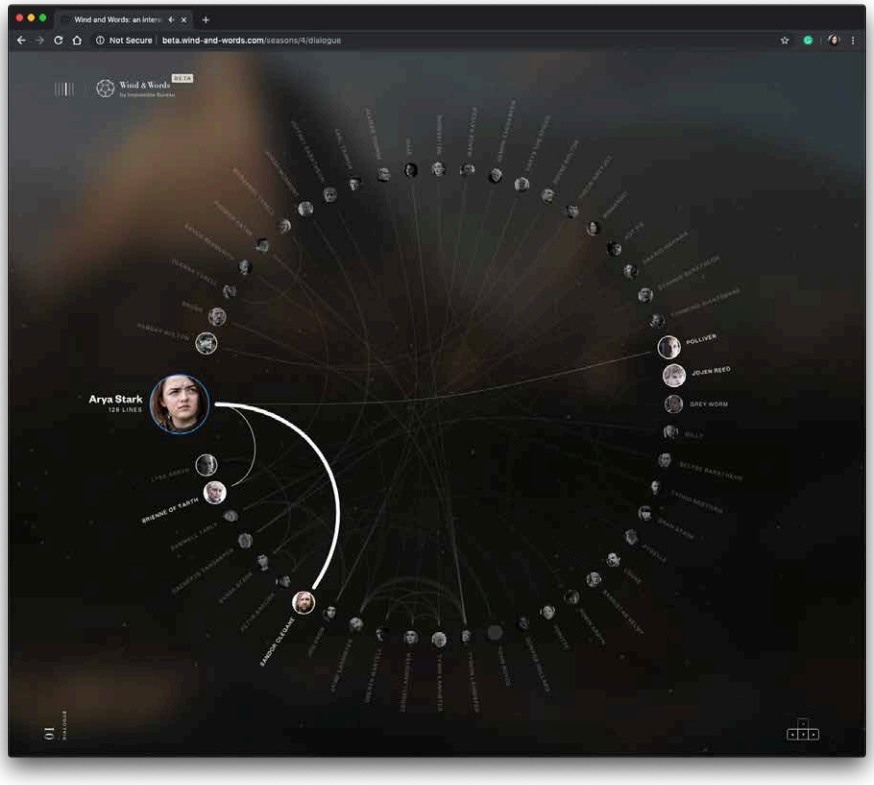


Figure 9-16. *Winds and Words*

Dynamic Queries

What

Employ easy-to-use standard controls such as sliders and checkboxes to define which slices or layers of the data set are shown, and immediately and interactively filter the data set. As the user adjusts those controls, show the results immediately on the data display.

The Google Public DataExplorer (Figure 9-17) allows the user to select a variety of variables and see the results immediately, the user can also move the slider on a timeline to see the results of the data over time.

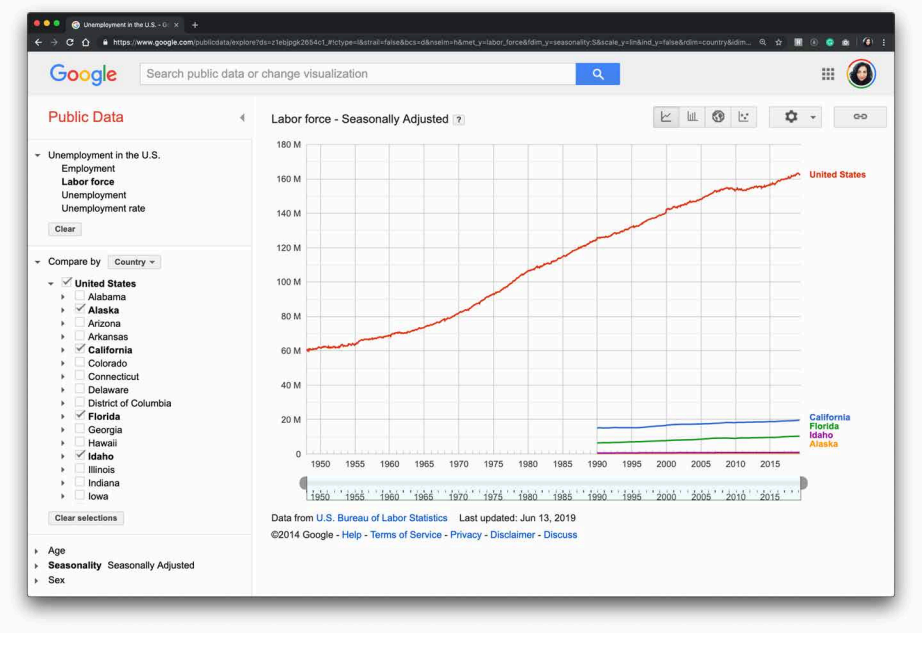


Figure 9-17. *Google Public Data Explorer*

Use when

You're showing the user a large, multivariate data set, of any shape, with any presentation. Users need to filter out some of the data in order to accomplish any of several objectives—to eliminate irrelevant parts of the data set, to see which data points meet certain criteria, to understand the relationships among the various data attributes, or simply to explore the data set and see what's there.

The data set itself has a fixed and predictable set of attributes (or parameters, variables, dimensions, whatever term you prefer) that are of interest to users. They are usually numeric and range bounded; they might also be sortable strings, dates, categories, or enumerations (sets of numbers representing non-numeric values). Or they might be visible areas of data on the information display itself that can be interactively selected.

Dynamic Queries can also apply to search results. Faceted searches might use a dynamic query interface to let users explore a rich database of items, such as products, images, or text.

Why

First, *Dynamic Queries* are easy to learn. No complicated query language is necessary at the user's end; well-understood controls are used to express common-sense Boolean expressions such as "price > $70 AND price < $100." They lack the full expressive power of a query language—only simple queries are possible without making the user interface too complicated—but in most cases, that's enough. It's a judgment call that you need to make.

Second, the immediate feedback encourages open-ended exploration of the data set. As the user moves a slider thumb, for instance, they see the visible data contract or expand. As the user adds or removes different subsets of the data, they see where the subsets go and how they change the display. The user can concoct long and complex query expressions incrementally, by tweaking this control, then that, then another. Thus, a continuous and interactive "question and answer session" is carried on between the user and the data. The immediate feedback shortens the iteration loop so that exploration is fun and a state of flow is possible.

Third—and this is a little subtler—the presence of labeled dynamic-query controls clarifies what the queryable attributes are in the first place. If one of the data attributes is a number that ranges from 0 to 100, for instance, the user can learn that just by seeing a slider that has 0 at one end and 100 at the other end.

How

The best way to design a dynamic query depends on your data display, the kinds of queries you think should be made, and your toolkit's capabilities. As mentioned, most programs map data attributes to ordinary controls that live next to the data display. This allows querying on many variables at once, not just those encoded by spatial features on the display. Plus, most people know how to use sliders and buttons.

Other programs afford interactive selection directly on the information display. Usually the user draws a box around a region of interest and the data in that region is removed (or retained while the rest of the data is removed). This is manipulation at its most direct, but it has the disadvantage of being tied to the spatial rendering of the data. If you can't draw a box around it—or otherwise select points of interest—you can't query on it! See the *Data Brushing* pattern for the pros and cons of a very similar technique.

Back to controls, then: picking controls for dynamic queries is similar to the act of picking controls for any kind of form—the choices arise from the data type, the kind of query to be made, and the available controls. Here are some common choices:

- Sliders to specify a single number within a range.
- Double sliders or slider pairs to specify a subset of a range: "Show data points that are greater than this number, but less than this other number."
- Radio buttons or drop-down (combo) boxes to pick one value out of several possible values. You might also use these to pick entire variables or data sets. In either case, "All" is frequently used as an additional metavalue.
- Checkboxes or toggles to pick an arbitrary subset of values, variables, or data layers.
- Text fields to type in single values. Remember that text fields leave more room for errors and typos than do sliders and buttons, but are better for precise values.

Example

The Apple Health interface (see Figure 9-18) allows the user to tap on days and view the individual day's activity.

Figure 9-18. *Apple Health*

Data Brushing

What

Allow the user to select data items in one view and show the same data selected simultaneously in another view. The example in Figure 9-19 shows a timeline that allows the user to easily shift the view over time and changes the data points on the map as well as the data on the side panel on the right.

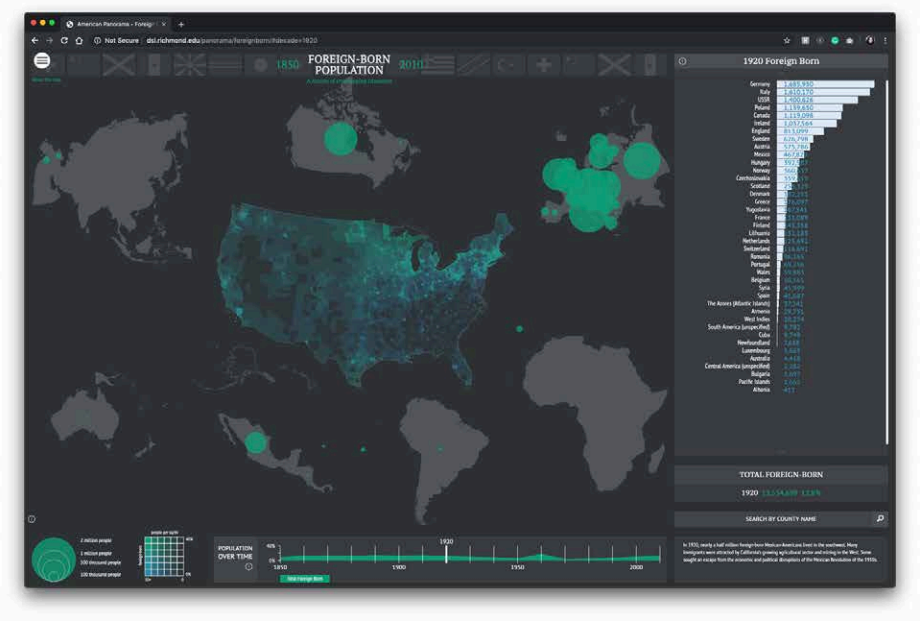


Figure 9-19. *American Panorama, Foreign-Born Population*

Use when

You can show two or more information graphicsat a time. You might have two line plots and a scatter plot, or a scatter plot and a table, or a diagram and a tree, or a map and a timeline, whatever—as long as each graphic is showing the same data set.

Why

Data Brushing offers a very rich form of interactive data exploration. First, the user can select data points using an information graphic itself as a "selector." Sometimes, it's easier to find points of interest visually than by less-direct means, such as *Dynamic Queries*—outliers on a plot can be seen and manipulated immediately, for instance, whereas figuring out how to define them numerically might take a few seconds (or longer). "Do I want all points where X > 200 and Y > 5.6? I can't tell; just let me draw a box around that group of points."

Second, by seeing the selected or "brushed" data points simultaneously in the other graphic(s), the user can observe those points in at least one other graphical context. That can be invaluable. To use the outlier example again, the user might want to know where those outliers are in a different data space, indexed by different variables —and by learning that, the user might gain immediate insight into the phenomenon that produced the data.

A larger principle here is *coordinated* or *linked views*. Multiple views on the same data can be linked or synchronized so that certain manipulations—zooming, panning, selection, and so forth—done to one view are simultaneously shown in the others. Coordination reinforces the idea that the views are simply different perspectives on the same data. Again, the user focuses on the same data in different contexts, which can lead to insight.

How

First, how will users select or "brush" the data? It's the same problem you'd have with any selectable collection of objects: users might want one object or several, contiguous or separate, selected all at once or incrementally. Consider these ideas:

- Tapping keywords
- Single selection by clicking with the mouse
- Selecting a range by turning them on and off

As you can see, it's important that the other views react immediately to *Data Brushing*. Make sure the system can handle a fast turnaround.

If the brushed data points appear with the same visual characteristics in all the data views, including the graphic in which the brushing occurs, the user can more easily find them and recognize them as being brushed. They also form a single perceptual layer (see the "Preattentive Variables: What's Related to What?" on page 435). Color hue is the preattentive variable most frequently used for brushing, probably because you can see a bright color so easily even when your attention is focused elsewhere.

Examples

Maps lend themselves well to *Data Brushing*, because data shown in a geographic context can often be organized and rendered in other ways, as well. A super delightful variation of the data brushing is in the AllTrails app (Figure 9-20, left). When the user moves their finger over the trail map on the bottom of the screen, the trail map pointer (the blue dot) moves along the trail to indicate the elevation and grade at that point in the trail.

Trulia's Search map view (Figure 9-20, right) shows available listings that fit within the search parameters by plotting them on a map. Users can filter the results of the specific boundaries outlined on the map by tapping or clicking Local Info and seeing data specific to the area within the boundaries.

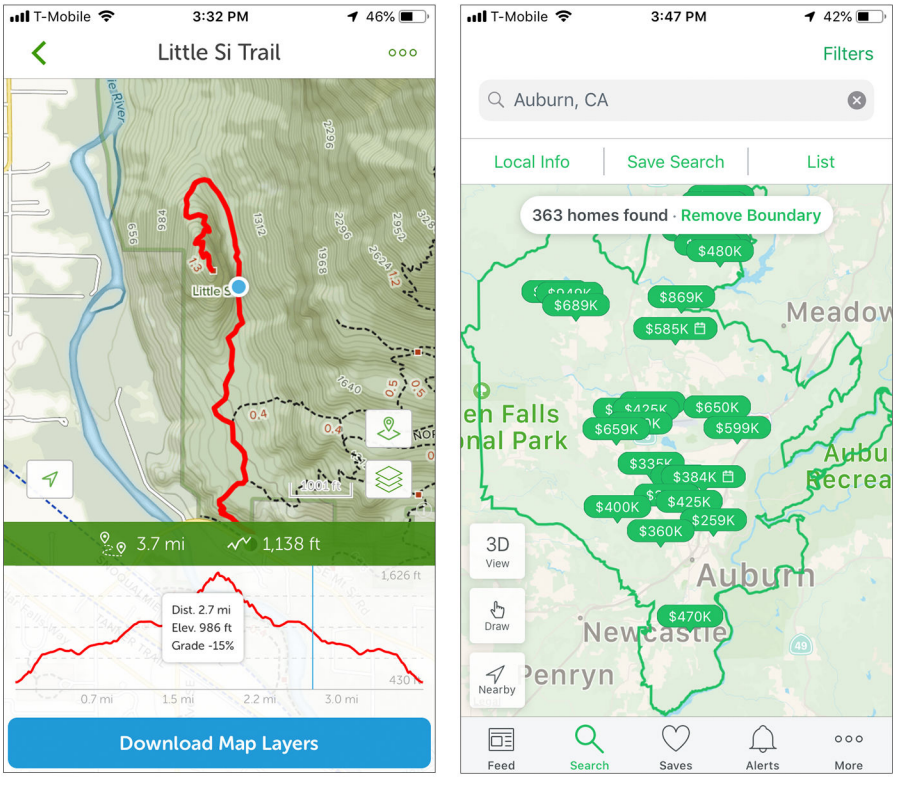


Figure 9-20. *AllTrails app and Trulia Search results*

Multi-Y Graph

What

Stack multiple graph lines in one panel to share the same x-axis (Figure 9-21).

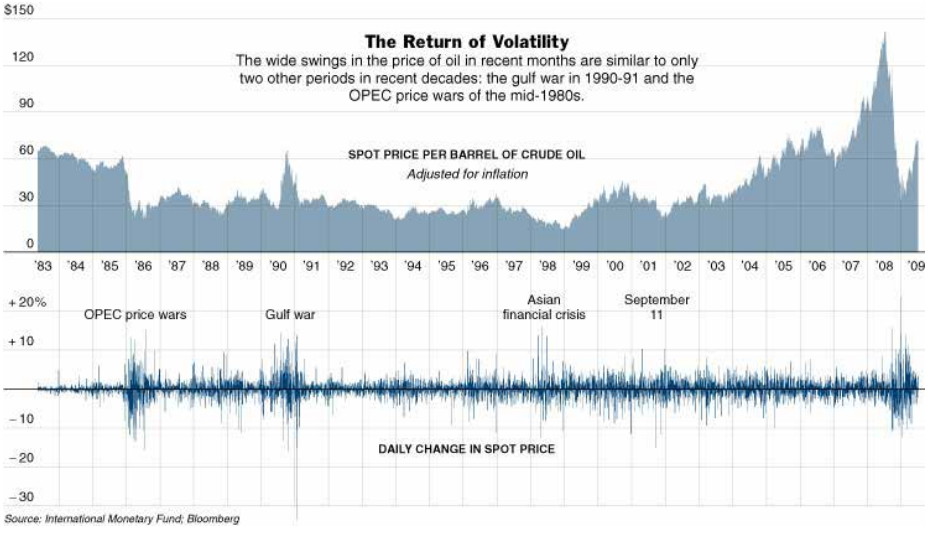


Figure 9-21. New York Times *graphic*

Use when

You present two or more graphs, usually simple line plots, bar charts, or area charts (or any combination thereof). The data in those graphs all share the same x-axis—often a timeline, but otherwise they describe different things, perhaps with different units or scale on the y-axis. You want to encourage the viewer to find "vertical" relationships among the data sets being shown—correlations, similarities, unexpected differences, and so on.

Why

Aligning the graphs along the x-axis first informs the viewer that these data sets are related, and then it lets them make side-by-side comparisons of the data.

How

Stack one graph on top of the other. Use one x-axis for both, but separate the y-axes into different vertical spaces. If the y-axes need to overlap somewhat, they can, but try to keep the graphs from visually interfering with each other.

Sometimes, you don't need y-axes at all; maybe it's not important to let the user find exact values (or maybe the graph itself contains exact values, such as labeled bar charts). In that case, simply move the graph curves up and down until they don't interfere with each other.

Label each graph so that its identity is unambiguous. Use vertical grid lines if possible; they let viewers follow an X value from one data set to another, for easier comparison. They also make it possible to discover an exact value for a data point of interest (or one close to it) without making the user take out a straightedge and pencil.

Examples

Google Trends allows a user to compare the use frequency of different search terms. The example in Figure 9-22 shows two sports-related terms that are comparable in volume, so they're easy to compare in one simple chart. But Google Trends goes beyond that. Relative search volume is illustrated on the top chart, whereas the bottom chart shows news reference volume. The metrics and their scales are different, so Trends uses two separate y-axes.

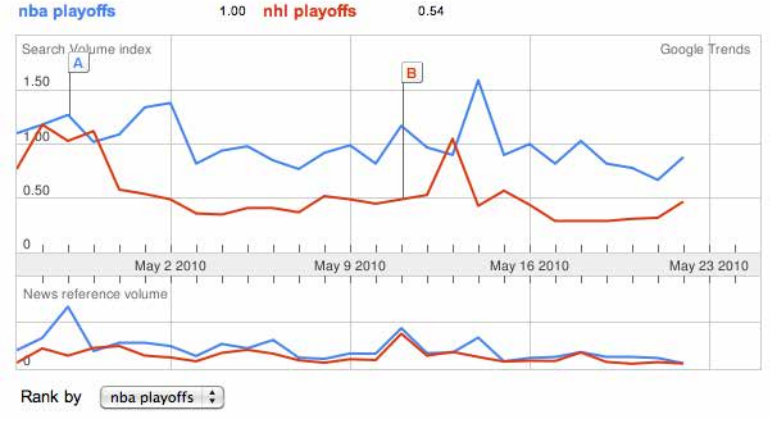


Figure 9-22. *Google Trends*

The example in Figure 9-23 shows an interactive *Multi-Y Graph* constructed in MATLAB. You can manipulate the three data traces' y-axes, color-coded on the left, with the mouse—you can drag the traces up and down the graph, "stretch" them vertically by sliding the colored axis end caps, and even change the displayed axis range by editing the y-axis limits in place. Here's why that's interesting: you might notice that the traces look similar, as though they were correlated somehow—all three drop in value just after the vertical line labeled 1180, for instance. But just how similar are they? Move them and see.

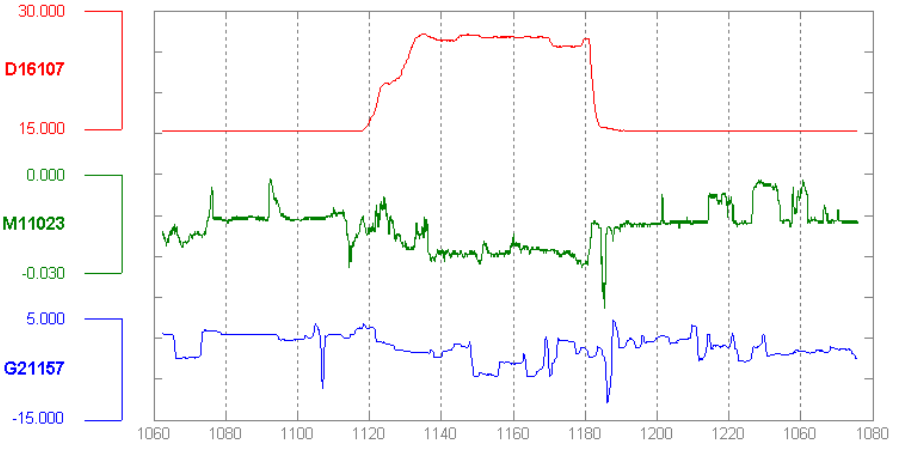


Figure 9-23. *MATLAB plot*

Your eyes are very, very good at discerning relationships among data graphics. By stacking and superimposing the differently scaled plot traces shown in Figure 9-24, a user might gain valuable insight into whatever phenomenon produced this data.

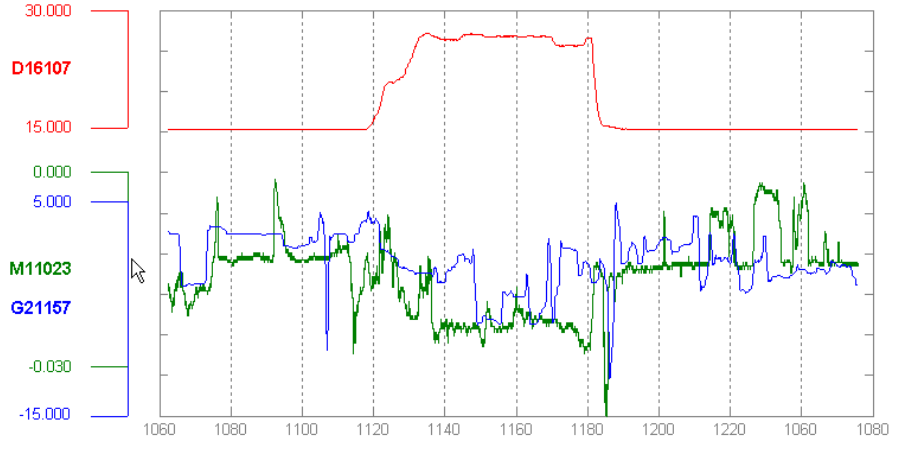


Figure 9-24. *MATLAB plot, again*

The information graphics in a multi-Y display don't need to be traditional graphs. The weather chart shown in Figure 9-25 uses a series of pictograms to illustrate expected weather conditions; these are aligned with the same time-based x-axis that the graph uses. (This chart hints at the next pattern, *Small Multiples*.)

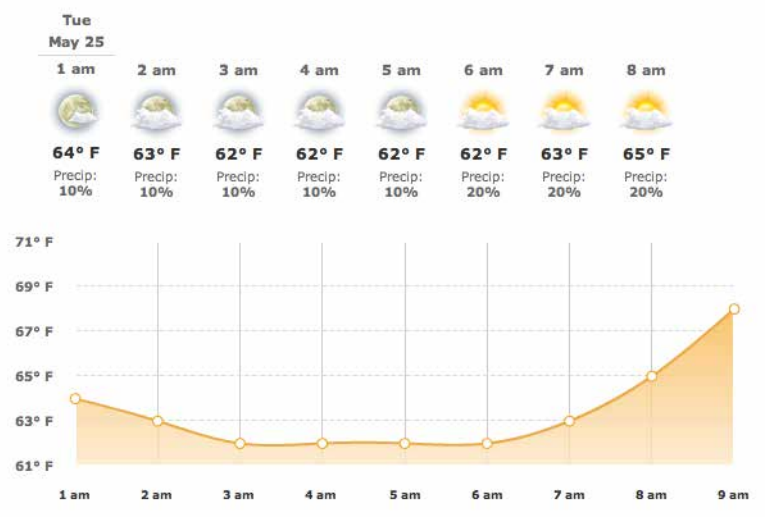


Figure 9-25. *Weather chart from The Weather Channel*

Small Multiples

What

Small multiples display small pictures of the data using two or three data dimensions. The images are tiled on the screen according to one or two additional data dimensions, either in a single comic-strip sequence or in a 2D matrix.

The climate heat map (Figure 9-26) shows data over time in small thumbnails to show dense information in a manner that is easy to understand.

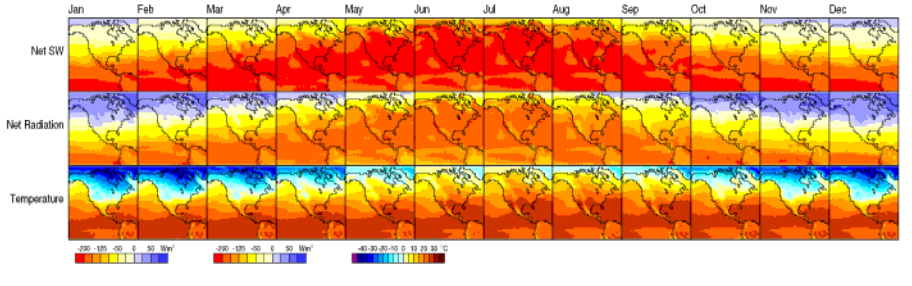


Figure 9-26. *Climate heat map, from a University of Oregon publication*

Use when

You need to display a large data set with more than two dimensions or independent variables. It's easy to show a single "slice" of the data as a picture—as a plot, table, map, or image, for instance—but you find it difficult to show more dimensions than that. Users might be forced to look at one plot at a time, flipping back and forth among them to see differences.

When using *Small Multiples*, you need to have a fairly large display area available. Mobile devices rarely do this well, unless each individual picture is very tiny. Use this pattern when most users will be seeing these on a large screen or on printed paper.

That being said, *sparklines* are a particular type of *Small Multiples* that can be very effective at tiny scales, such as in running text or in a column of table cells. They are essentially miniature graphs, stripped of all labels and axes, created to show the shape or envelope of a simple data set.

Why

Small Multiples are data-rich—they show a lot of information at one time, but in a comprehensible way. Every individual picture tells a story. But when you put them all together and demonstrate how each picture *varies* from one to the next, an even bigger story is told.

As Edward Tufte put it in his classic book, *Envisioning Information* (Graphics Press), "Small multiple designs, multivariate and data bountiful, answer directly by visually enforcing comparisons of changes, of the differences among objects, of the scope of alternatives." (Tufte named and popularized *Small Multiples* in his famous books about visualization.)

Think about it this way. If you can encode some dimensions in each individual picture, but you need to encode an extra dimension that just won't fit in the pictures, how could you do it?

Sequential presentation
: Express that dimension varying across time. You can play them like a movie, use Back/Next buttons to screen one at a time, and so on.

3D presentation
: Place the pictures along a third spatial axis, the z-axis.

Small multiples
: Reuse the x- and y-axes at a larger scale.

Side-by-side placement of pictures lets a user glance from one to the other freely and rapidly. The user doesn't need to remember what was shown in a previous screen, as would be required by a sequential presentation (although a movie can be *very*

effective at showing tiny differences between frames). The user also doesn't need to decode or rotate a complicated 3D plot, as would be required if you place 2D pictures along a third axis. Sequential and 3D presentations sometimes work very well, but not always, and they often don't work in a noninteractive setting at all.

How

Choose whether to represent one extra data dimension or two. With only one, you can lay out the images vertically, horizontally, or even line-wrapped, like a comic strip —from the starting point, the user can read through to the end. With two data dimensions, you should use a 2D table or matrix—express one data dimension as columns, and the other as rows.

Whether you use one dimension or two, label the *Small Multiples* with clear captions —individually if necessary, or otherwise along the sides of the display. Make sure the users understand which data dimension is varying across the multiples, and whether you're encoding one or two data dimensions.

Each image should be similar to the others: the same size and/or shape, the same axis scaling (if you're using plots), and the same kind of content. When you use *Small Multiples*, you're trying to bring out the meaningful differences between the things being shown. Try to eliminate the visual differences that don't mean anything.

Of course, you shouldn't use too many *Small Multiples* on one screen. If one of the data dimensions has a range of 1 to 100, you probably don't want 100 rows or columns of small multiples, so what do you do? You could *bin* those 100 values into, say, five bins containing 20 values each. Or you could use a technique called *shingling*, which is similar to binning but allows substantial overlap between the bins. (That means some data points will appear more than once, but that can be a good thing for users trying to discern patterns in the data; just make sure it's labeled well so that they know what's going on.)

Some small-multiple plots with two extra encoded dimensions are called *trellis plots* or *trellis graphs*. William Cleveland, a noted authority on statistical graphing, uses this term, and so do the software packages S-PLUS and R.

Examples

The North American climate graph, at the top of the pattern in Figure 9-26, shows many encoded variables. Underlying each small-multiple picture is a 2D geographic map, of course, and overlaid on that is a color-coded "graph" of some climate metric, such as temperature. With any one picture, you can see interesting shapes in the color data; they might prompt a viewer to ask questions about why blobs of color appear over certain parts of the continent.

The *Small Multiples* display as a whole encodes two additional variable : each column is a month of the year, and each row represents a climate metric. Your eyes have probably followed the changes across the rows, noting changes through the year, and comparisons up and down the columns are easy, too.

The example shown in Figure 9-27 uses the grid to encode two independent variables —ethnicity/religion and income—into the state-by-state geographic data. The dependent variable, encoded by color, is the estimated level of public support for school vouchers (orange representing support, green opposition). The resultant graphic is very rich and nuanced, telling countless stories about Americans' attitudes toward the topic.

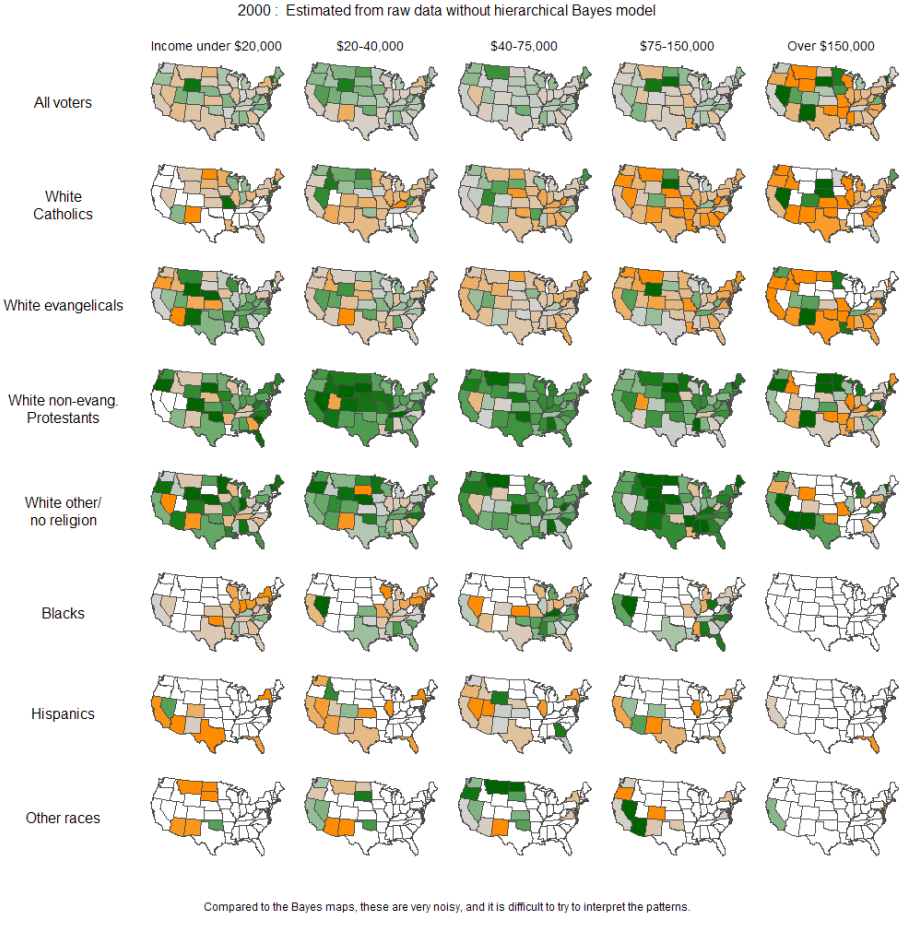


Figure 9-27. *Geographic and demographic Small Multiples chart*

The Power of Data Visualization

The example in Figure 9-28 is an excellent demonstration of how using data visualization done well can be aesthetically pleasing but informative. The Show Your Stripes information graphics shows temperature change data from 1850 to 2019 using only simple bars and color. Its designer compiled dense information and simplified it to inform the viewer. One visual image is doing the work of several charts and graphs in this compelling image.

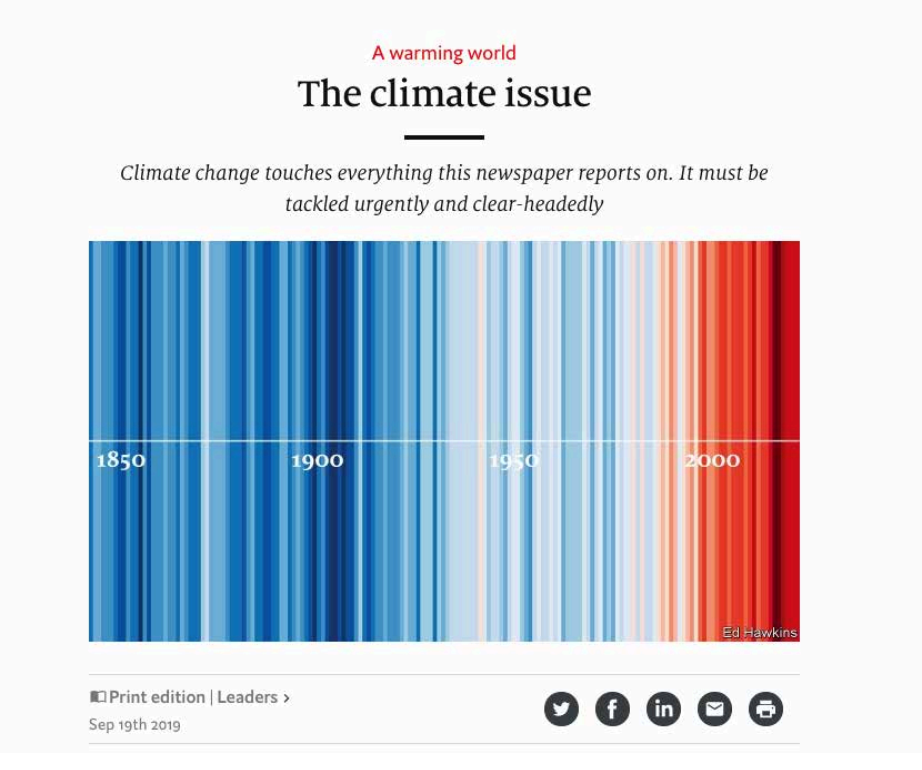


Figure 9-28. *Show Your Stripes via* The Economist*, Sept 2019*

Yes, a picture can tell a thousand words, but when you add interactivity to these pictures you increase understanding. The examples in this chapter show that graphical methods such as maps and diagrams can communicate dense amounts of information in graceful, delightful, and beautiful ways.

Chapter 10

Getting Input from Users: Forms and Controls

Sooner or later, the software you design will probably need to collect information from people. It might even happen in the first few minutes of interaction. What's your login name? What words do you want to search for? Where should we ship your order?

In this chapter, we look at a number of topics related to getting input from users:

- Guidelines for designing useful, usable forms
- Different forms for different purposes
- Developing effective autocompletion
- Designing complicated controls

Form interactions seem to be easy to design at first. This is for several reasons. Everyone is familiar with the standard form elements such as text fields, checkboxes, and combo boxes. We have more than two decades of interactive form design examples to choose from. These input controls are also a big part of the user interface (UI) frameworks we will review in Chapter 11. All of these interface toolkits have form elements and controls ready to go, out of the box.

However, you might stuggle with a design that is awkward, difficult for users to understand, or difficult to complete. Here's another sample question that can shed light on the kinds of problems designers need to think about in designing forms and controls: for what location do you want a weather report? The user might wonder, do I specify a location by neighborhood, city, region, state, country, postal code, or what? Are abbreviations OK? What if I misspell it? What if I ask for a city it doesn't know

about? Isn't there a map to choose from? And why can't it remember the location I gave it yesterday, anyhow?

This chapter discusses ways to smooth out these problems. The patterns, techniques, and controls described here apply mostly to form design—a form being simply a series of question/answer pairs. However, they will also be useful in other contexts, such as for single controls on web pages or on application toolbars. Input design and form design are core skills for interaction designers, as you can use them in every genre and on every platform.

First, let's review some design guidelines for creating effective, usable forms and controls.

The Basics of Form Design

Here are a few principles to remember when doing input and form design:

Respect the user's time and attention
: Approach form design with an awareness of how much time and effort it will cost a person to fill out a form. This cost in their eyes can be higher than the designer thinks. Make the form as short and simple as possible using the techniques that we describe in this chapter.

Make sure the user understands the purpose of the form
: A form asks the user to do work in exchange for something. The form's headlines, context, and wording should confirm why it's asking for this information, how the information will be used, and what the user will get out of it.

Minimize the number of form inputs
: Consider each question or element carefully. It's unkind to ask the user to do unnecessary work. If you ask for a US Zip Code, for instance, can you deduce the city and state? For credit cards, it's not necessary to ask for the card type (Visa, Mastercard, etc.) because the first two integers will identity the type of credit card. Consider not asking for first and last name if an email address can function as the username.

Minimize visual clutter
: Generally, a form is not the place to distract the user with other material. Keep it simple, clean, and focused.

Group and title the form elements into sections where possible
: If you design a long or complicated form, break it up into descriptive *Titled Sections* (see "Titled Sections" on page 238). Group and label the form elements. Use titles and subtitles to further organize and explain the form.

Consider dynamic, show/hide sections for long, complicated forms
: A long or complicated form can be intimidating if everything is displayed all at once. There is also the increased chance that the user will skip a step among all the fields. Instead, consider breaking up the form into sections with only the first section displayed by default. The other sections can be displayed one by one, in sequence. Optional parts of a form can be always hidden by default.

Use alignment for clear vertical flow
: Use layout and alignment so there is a strong vertical flow to the form whether in one column or more. Align the left edges of the inputs, and use the same vertical separation as much as possible. The eye should be able to move from label to input with minimum travel.

Indicate what are required and what are optional fields
: Indicating what fields must be filled out in a form is both a courtesy and also a usability and error-prevention strategy. Should you mark the required fields, or the optional ones? You will need to decide on a standard approach for all your forms in your app or website.

Labels, instructions, examples, and help
: Use descriptive form labels, input examples, and help text with individual form fields. Labels are still a best practice for ensuring accessibility by differently abled people. Avoid lots of placeholder text in fields because it can confuse users into thinking they filled them out already. Use vocabulary that is appropriate for the audience and task domain. Don't be afraid to put instructions into your form if necessary (and you always have the option of putting instructions in a user-triggered pop-up or modal window).

Use the width of the input fields to preview the length of the input
: Your choice of controls will affect the user's expectation of what is asked for. A radio button suggests a one-out-of-many choice, whereas a one-line text field suggests a single word or phrase. A large, multiline text entry field implies a longer answer, such as a paragraph.

Accept variations in input formatting
: Accept multiple formats for dates, addresses, phone numbers, credit card numbers, and so on, per the *Forgiving Format* pattern. If you ask for input that needs to be formatted in a specific way, offer the user examples about how to format it.

Error prevention and validation as quickly as possible
: The goal here is to help the user get it right the first time. Give instructions and examples to make it clear what information is needed. Deploy contextual help in the form. Show an error message as soon as it becomes clear that the user made a mistake. Consider giving feedback on a field-by-field basis. Help the person catch any individual validation errors before submitting the form as a whole. On the

form, give actionable validation messaging: Indicate which input field is problematic, why, and how the user might fix it. See the patterns *Password Strength Meter* and *Error Messages*.

Autocompletion goes a step further by telling the user what input is valid, or by reminding the user what they entered some previous time, or saving time by offering the most common entries.

Consider top-aligned labels for mobile and web-responsive designs
: Top-aligned labels are those that appear above the user input fields, instead of to the left. Using this alignment works best for responsive-design screens because the components can stack vertically without a change to layout. There is less chance the label and input will become misaligned.

Consider internationalization
: Account for the need to offer the form to users in countries and cultures outside your own. There are the immediate concerns that it be possible for the language of the form to be changed without breaking the layout (consider different lengths of translated text strings and different writing directions). There is also the need to switch to different units, styles, and formatting of numbers, measurements, dates, times, currencies, and other standards. Beyond that, different data securityand privacy legislation might affect what information you can legally gather, transmit, and store.

Message success
: When a user has successfully submitted a form, make sure to let them know this and specify to them what is going to happen next.

Usability test it
: For some reason, when input forms are involved, it's particularly easy for designers and users to make radically different assumptions about terminology, possible answers, intrusiveness, and other context-of-use issues. Do some usability testing, even if you're reasonably sure your design is good.

Form Design Continues to Evolve

One major caveat to form design is that it continues to change and evolve. As interaction designers, we need to consider the pros and cons of new form capabilities.

Required versus optional

The practice of marking required fields with an asterisk (*) is still common. It's a good idea in this case to explain in the form what the asterisk means, although many sites are now omitting the legend. Because this places a slight burden on the user to

figure out, other approaches are now used. Usability experts Nielsen/Norman Group says marking all required fields is still the most usable approach.[1]

One option is to count how many required fields versus optional fields there are and then label only the smaller number—the exceptions to the majority. A second option is to display only required fields, omitting all optional fields. Explaining that all fields are required helps remove confusion, but many forms do without this notice, as well.

A third way is to not mark required fields, and mark optional fields with the word "optional" next to the label or input field. This is the standard approach as used in the United States Web Design System and the UK Government web design standards.

Floating labels

It's now possible to display labels for input fields inside the form elements themselves. The labels display at full size by default, similar to placeholder text in an input field. However, if a user selects the input field, the floating label becomes small-sized text and moves up to be aligned close to the inside top of the input field. It gets out of the way but remains visible as the user enters text. Although it gives a snappy animation to your forms, consider whether this will be usable for your audience.

Further Reading

There are a number of design books focused specifically on form design. Here are three to consider if you want a more extensive analysis:

- Enders, Jessica. *Designing UX: Forms: Create Forms That Don't Drive Your Users Crazy*. SitePoint, 2016.
- Jarrett, Caroline, and Gerry Gaffney. *Forms That Work: Designing Web Forms for Usability*. Elsevier/Morgan Kaufmann, 2010.
- Wroblewski, Luke. *Web Form Design: Filling in the Blanks*. Rosenfeld Media, 2008.

1 Budiu, Raluca. "Marking Required Fields in Forms." *Nielsen Norman Group*, 16 Jun. 2019, *https://oreil.ly/vPQSQ*.

The Patterns

Most of these patterns describe controls—specifically, how you can combine input controls with other controls and text in ways that make them easier to use. Some patterns define structural relationships between elements, such as *Drop-down Chooser* and *Fill-in-the-Blanks*.Others, such as *Good Defaults and Smart Prefills* and *Autocompletion*, discuss the values of controls and how those values change.

The patterns in the list that follows deal primarily with text fields. That shouldn't be surprising. Text fields are as common as dirt, but they don't make it easy for users to figure out what should go in them. They're easiest to use when presented in a context that makes their usage clear. The patterns give you many ways to create that context.

- *Forgiving Format*
- *Structured Format*
- *Fill-in-the-Blanks*
- *Input Hints*
- *Input Prompt*
- *Password Strength Meter*
- *Autocompletion*

The next two patterns deal with controls other than text fields. *Drop-down Chooser* describes a way to create a custom control, and *List Builder*, referenced in the control table shown earlier, describes a commonly reinvented combination of controls that lets users construct a list of items.

You should design the remaining patterns into the whole form. They apply equally well to text fields, drop- down menus, radio buttons, lists, and other stateful controls, but you should use them consistently within a form (or within a dialog box, or even an entire application).

- *Good Defaults and Smart Prefills*
- *Error Messages*

Patterns from other chapters apply to form design, as well. You can place labels above the form fields (at the cost of vertical space, but with plenty of horizontal room for long labels), or left-aligned along the left edge of the form. The choice can affect the speed of form completion.

Chapters 3 and 4 also give you some larger-scale design possibilities. A gatekeeper form—any form that stands between the user and their immediate goal, such as

sign-up or purchase forms—should be in *Center Stage*, with very few distractions on the page. Alternatively, you might make it a *Modal Panel*, layered over the page.

If you have a long form that covers different topics, you might consider breaking it up into *Titled Sections* or even separate pages. (Tabs tend to work poorly as grouping mechanisms for forms.) If you break up a form into a sequence of pages, use the *Wizard* and *Progress Indicator* patterns to show users where they are and where they're going.

Finally, forms should use a *Prominent "Done" Button* (Chapter 8) for the completion or submission action. If you have secondary actions, such as a form reset or a help link, make those less prominent.

Forgiving Format

What

Permit users to enter inputs in a variety of choices, formats, and syntax, and make the application interpret it intelligently. *Weather.com* (Figure 10-1) shows an example of this.

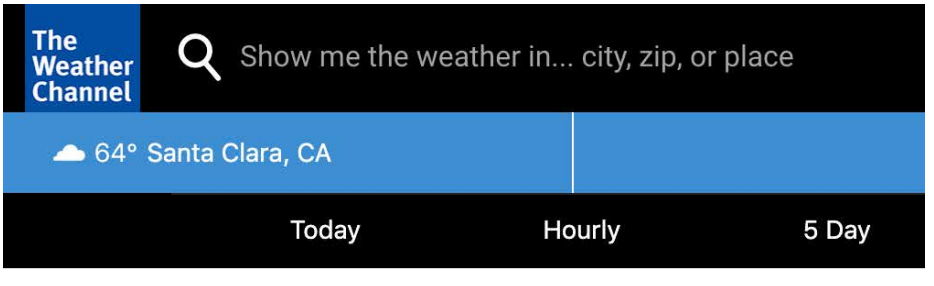


Figure 10-1. *Weather.com*

Use when

Your UI asks for data that users might type with an unpredictable mix of vocabulary or styling (whitespace, hyphens, abbreviations, or capitalizations). More generally, the UI can accept input of various kinds from the user—different meanings, formats, or syntax. But you want to keep the interface visually simple.

Why

The user just wants to get something done, not think about "correct" formats and complex UI . Computers are good at figuring out how to handle input of different types (up to a point, anyway). It's a perfect match: let the user type whatever they need, and if it's reasonable, make the software do the appropriate thing with it.

This can help simplify the UI tremendously, making it much easier to figure out. It can even remove the requirement for an *Input Hints* or *Input Prompt*, though they're often seen together, as in the example in Figure 10-1.

You might consider *Structured Format* as an alternative, but that pattern works best when the input format is entirely predictable (and usually numeric, like telephone numbers).

How

The catch: it turns a UI design problem into a programming problem. You need to think about what kinds of text a user is likely to type in. Maybe you ask for a date or time, and only the format varies—that's an easy case. Or maybe you ask for search terms, and the variation is what the software does with the data. That's more difficult. Can the software disambiguate one case from another? How?

Each application uses this pattern differently. Just make sure the software's response to various input formats matches what users expect it to do. Test, test, and test again with real users.

Examples

The *New York Times* uses *Forgiving Format* in several features that need information from users. Figure 10-2 shows examples from its real estate search and from its financial quotes feature.

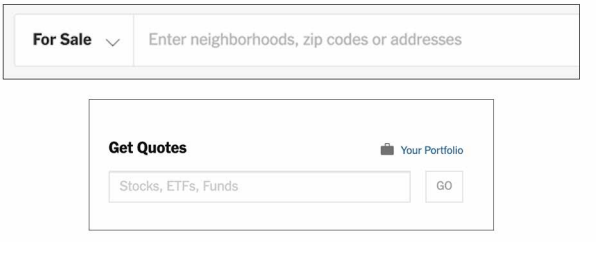


Figure 10-2. *Two search fields in the* New York Times *website hint at the variety of possible formats they will accept*

Google Finance (Figure 10-3) helps users find the correct stock symbol by mapping what they type to the most likely matching stock symbols. It's not necessary to know or enter the exact stock symbol.

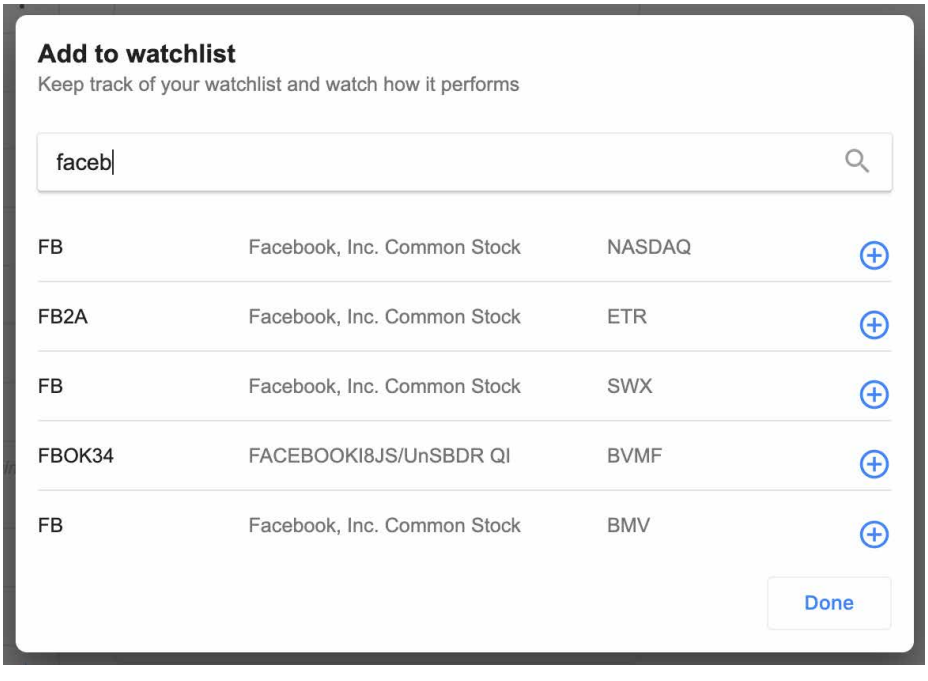


Figure 10-3. *Adding a stock to a personal watchlist in Google Finance*

Consider forms that request credit card numbers from the user. As long as 16 digits are typed, why should the form care whether the user separates them by spaces, or by hyphens, or by nothing at all? PayPal, for example, allows customers to enter their credit card number however they want. The credit card number field accepts spaces as separators, hyphens, or no spaces. Paypal standardizes the format immediately afterward. (Figure 10-4).

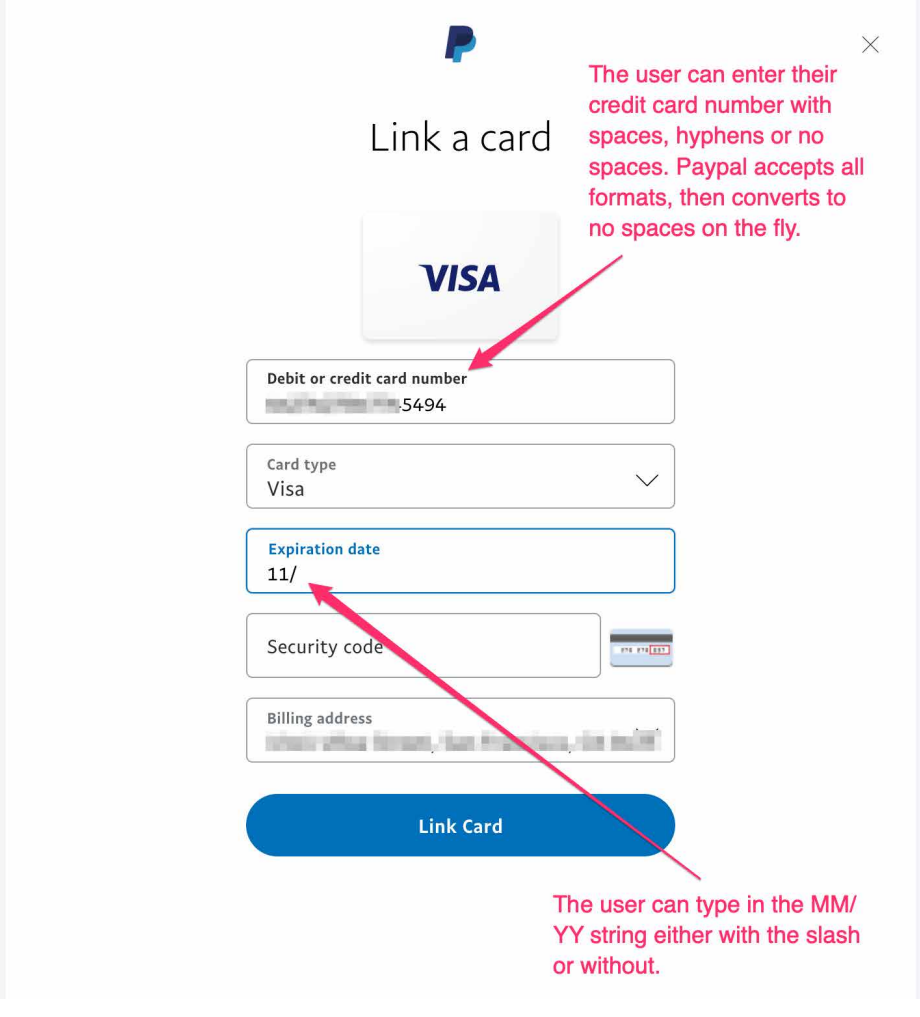


Figure 10-4. *PayPal*

Figure 10-5 comes from Google Calendar's tool for setting up a meeting. Look at the "from" and "to" fields in the screenshot—you don't need to give it a fully defined date, like what's in the text fields now. If today is July 13 and you want to set up a meeting for July 20, you can type any of the following terms:

- Saturday 7/20
- Saturday 20/7
- 20/7/2019
- 7/20/2019
- 20/7
- 7/20

The specified date then is "echoed back" to the user in the appropriate format for the user's language and location.

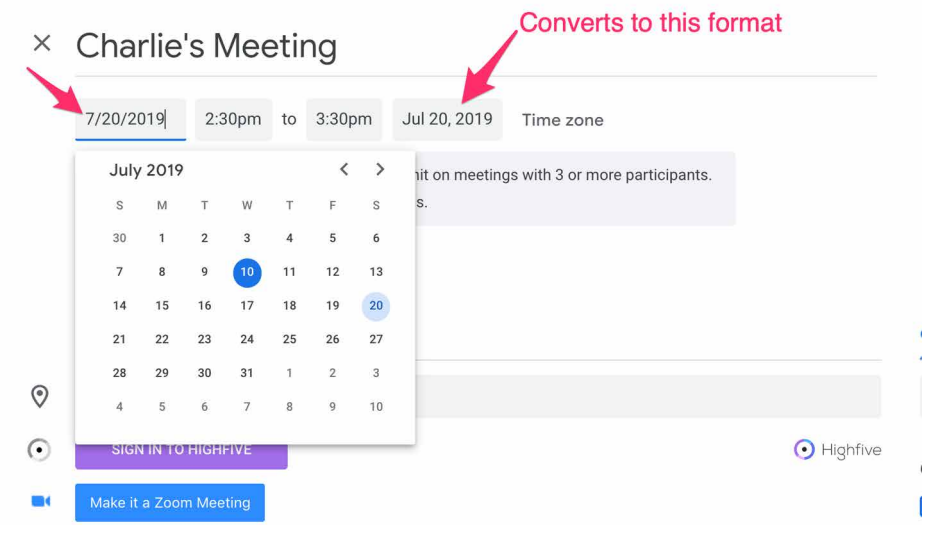


Figure 10-5. *Google Calendar*

Structured Format

What

Instead of using one text field, use a set of text fields that reflect the structure of the requested data.

Use when

Your interface requests a specific kind of text input from the user, formatted in a certain way. That format is familiar and well defined, and you don't expect any users to need to deviate from the format you expect. Examples include credit card information, local telephone numbers, and license strings or numbers.

It's generally a bad idea to use this pattern for any data in which the preferred format might vary from user to user. Consider especially what might happen if your interface is used in other countries. Names, addresses, postal codes, and telephone numbers all have different standard formats in different places. Consider using *Forgiving Format*in those cases.

Why

The structure of the text fields gives the user a clue about what kind of input is being requested. Expectations are clear. The user is spared from wondering whether they need to type in any spaces, slashes, or hyphens. The structure already takes care of that.

This pattern usually is implemented as a set of small text fields instead of one big one. That alone can reduce data entry errors. It's easier for someone to double-check several short strings (two to five characters or so) than one long one, especially when numbers are involved. Likewise, it's easier to transcribe or memorize a long number when it's broken up into chunk . That's how the human brain works.

Contrast this pattern to *Forgiving Format*, which takes the opposite tack: it allows you to type in data in any format, without providing structural evidence of what's being asked for. (You can use other clues, instead, like *Input Hints*.) *Structured Format* is better for very predictable formats, *Forgiving Format* for open-ended input.

How

Design a set of text fields that reflect the format being asked for. Keep the text fields short, as clues to the length of the input.

After the user has typed all the digits or characters in the first text field, confirm it by automatically moving the input focus to the next field. The user can still go back and

re-edit the first one, of course, but now they know how many digits are required there.

You can also use *Input Prompt* to give the user yet more clues about what's expected. In fact, structured format fields for dates often do use *Input Prompt*, such as "dd/mm/yyyy".

Examples

Airbnb uses structured format in the form for the customer to enter their security code to verify their identity (Figure 10-6). There are four placeholder dashes of one digit each. This corresponds to the code they are sent. This is structured format for the four-digit code, with the insertion point jumping automatically to the next integer field when the user types in a digit.

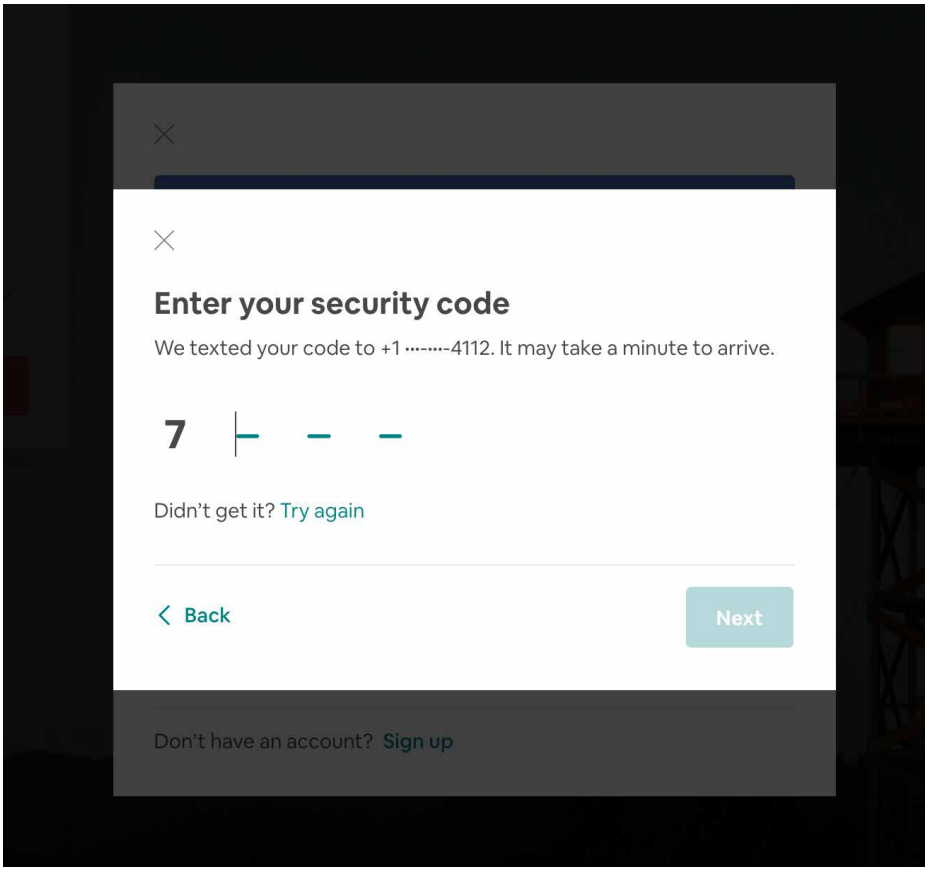


Figure 10-6. *Airbnb security code entry form*

At Official Payments, a service provider to the State of California, the online forms for making tax payments include structured data (Figure 10-7). The taxpayer's telephone number fields and the tax date range fields are separate and have added symbols to reinforce the proper data format. Telephone number and date fields are structured to promote successful data entry and avoid errors.

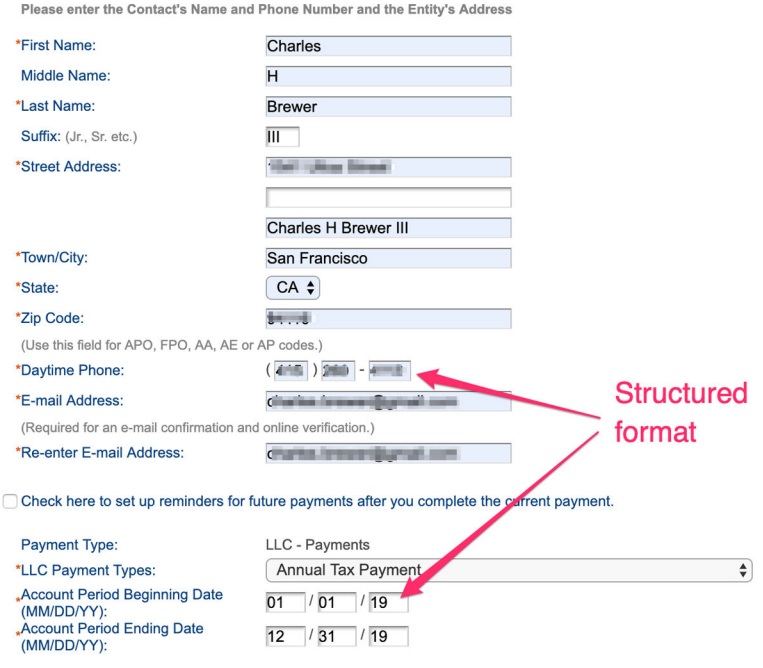


Figure 10-7. *Officialpayments.com*

Similarly, in Outlook for Microsoft Office 360, scheduling a meeting brings up a dialog window with date and time selectors (Figure 10-8). These are broken up into individual input fields so that the user must step through each one in sequence. It's not possible to select the entire date string or time string and type in new values, or try alternates such as DD/MM/YYYY. Each month, day, year, hour, minute and am/pm string must be entered separately, in the prescribed place and format (unless using the date picker drop-down list).

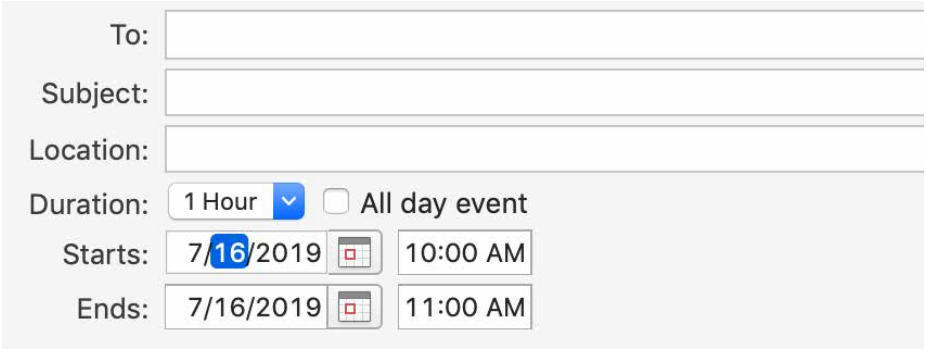


Figure 10-8. *Microsoft Outlook/Office 360*

Fill-in-the-Blanks

What

Arrange one or more fields in the form of a prose sentence or phrase, with the fields as "blanks" to be filled in by the user. San Francisco Public Library (Figure 10-9) uses this approach for its search.

Figure 10-9. *San Francisco Public Library*

Use when

You need to ask the user for input, usually one-line text, a number, or a choice from a drop-down list. You tried to write it out as a set of label/control pairs, but the labels' typical declarative style (such as "Name:" and "Address:") isn't clear enough for users to understand what's going on. You can, however, verbally describe the action to be taken once everything's filled out, in an active-voice sentence or phrase.

Why

Fill-in-the-Blanks helps to make the interface self-explanatory. It makes it easier to construct rules or conditions. After all, we all know how to finish a sentence. (A verb phrase or noun phrase will do the trick, too.) Seeing the input, or "blanks," in the context of a verbal description helps the user understand what's going on and what's required.

How

Write the sentence or phrase using all your word-crafting skills. Use controls in place of words.

If you're going to embed the controls in the middle of the phrase instead of at the end, this pattern works best with text fields, drop-down list , and combo boxes—in other words, controls with the same form factor (width and height) as words in the sentence. Also, make sure the baseline of the sentence text lines up with the text baselines in the controls, or it will look sloppy. Size the controls so that they are just long enough to contain the user's choices, and maintain proper word spacing between them and the surrounding words.

This is particularly useful for defining conditions, as one might do when searching for items or filtering them out of a display. The Excel and eBay examples in Figures 10-10, 10-11, and 10-12 illustrate the point. Robert Reimann and Alan Cooper describe this pattern as an ideal way to handle queries; their term for it is *natural language output*.[2]

There's a big "gotcha" in this pattern, however: it becomes very difficult to properly localize the interface (convert it to a different language) because comprehension now depends upon word order in a natural language. For some international products or websites, that's a nonstarter. You might need to rearrange the UI to make it work in a different language; at the very least, work with a competent translator to make sure the UI can be localized.

Examples

Microsoft Excel makes extensive use of *Fill-in-the-Blanks* for its conditional formatting rules. These allow users to set up automatic cell highlighting based on logical rules so that it's easy to see important status or results. Using the sentence-style format of *Fill-in-the-Blanks* makes it easy to specify the desired logic.

In Figures 10-10 and 10-11, we see two different ways Excel enables this. In the "Classic" example, a sequence of drop-down menus with phrases allows the user to set up a fairly complicated If-then instruction. The user selects a series of statements and conditions in a sentence-like sequence to specify the desired logic. In Figure 10-10, reading the fill-in-the-blank structure sounds very close to natural language instructions: "Format only cells that contain a cell value between 33 and 66. Format them with a yellow fill and bold yellow text."

2 See their book *About Face 2.0: The Essentials of Interaction Design* (Wiley), page 205.

In Figure 10-11, we're creating instructions for when to display green, yellow, and red icons. There's more to configure here, but Excel lays out a stack of three fill-in-the-blank statement builders. These are more compact, using logical symbols like ">=" instead of writing out "greater than or equal to." Still, it's easy to construct the display logic by stepping through the fill-in-the-blanks structure. It's a little more work, but for the green icon, reading left to right, we can get "Display the green icon when the value is greater than or equal to 67 percent." Excel takes the numerical value that the user enters here and automatically prefills the fill-in-the-blank logic for the second, yellow icon so that it doesn't conflict with the first.

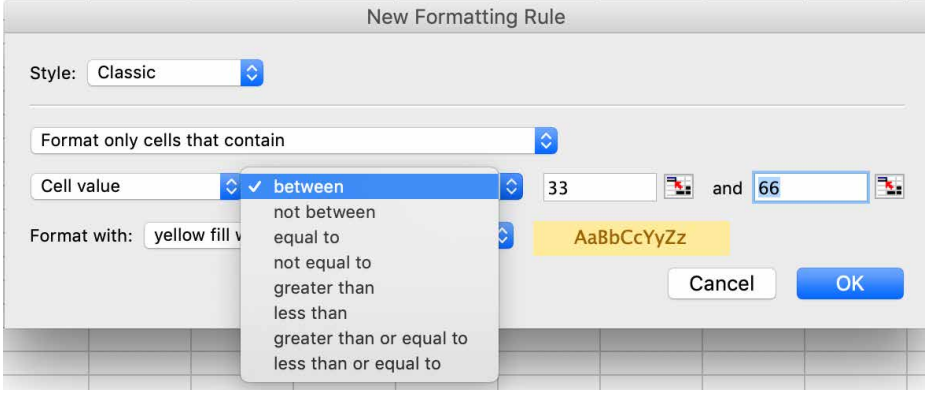


Figure 10-10. *Classic conditional fomatting in Microsoft Excel*

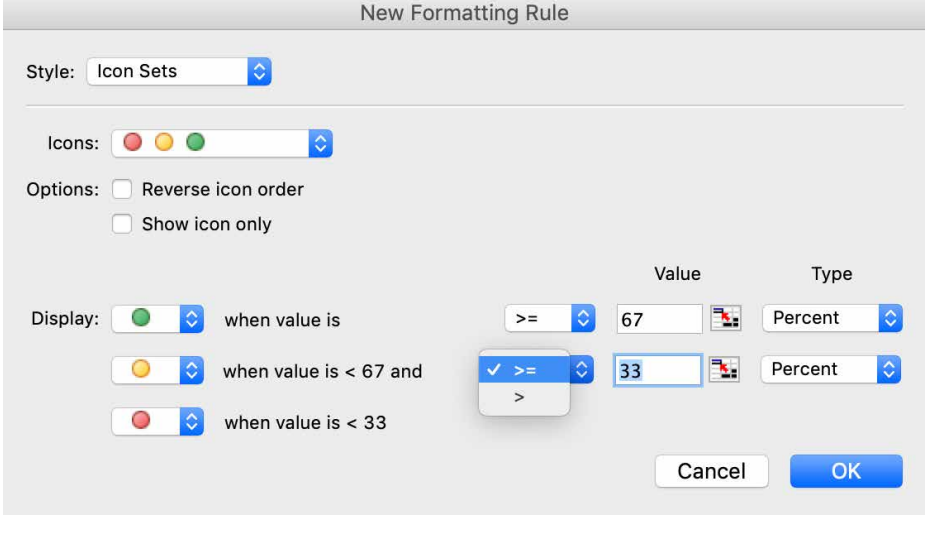


Figure 10-11. *Icon Set conditional formatting in Microsoft Excel*

When users search for items on eBay, they can use the Advanced Search form to specify various criteria. The form shown in Figure 10-12 has several examples of *Fill-in-the-Blanks*.

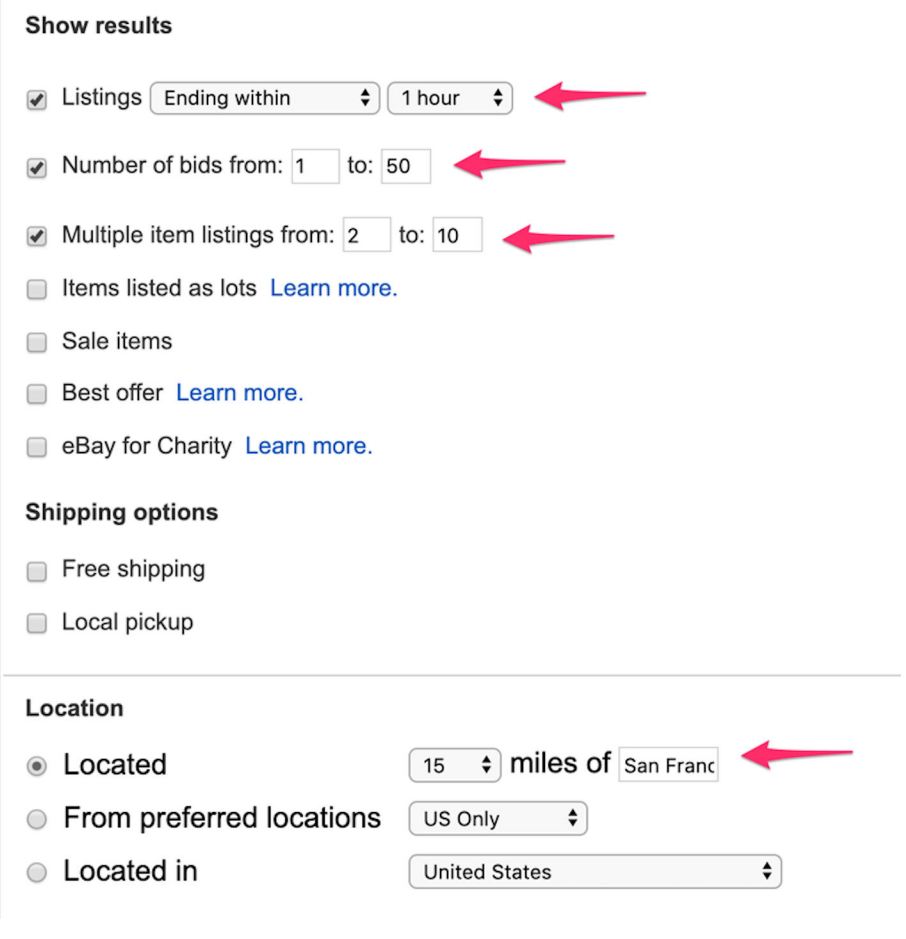


Figure 10-12. *eBay search filter form*

Input Hints

What

Beside or below an empty text field, place a phrase or example that explains what is required or gives additional detail about what is being requested.

Use when

The interface presents a text field, but the kind of input it requires isn't obvious to all users. You don't want to put more than a few words into the text field's label.

Why

A text field that explains what goes into it frees users from having to guess. The hint provides context that the label itself may not provide. If you visually separate the hint from the main label, users who know what to do can more or less ignore the hint, and stay focused on the label and contro .

How

Write a short example or explanatory sentence, and put it below or beside the text field. The hint can be visible all the time, or it can appear when the text field receives input focus.

Keep the text in the hint small and inconspicuous, though readable; consider using a font two points smaller than the label font. (A one-point difference will look more like a mistake than an intended font-size change.) Also, keep the hint short. Beyond a sentence or two, many users' eyes will glaze over, and they'll ignore the text altogether.

Examples

Figure 10-13 shows two short input hints in the 1-800-Flowers registration page. The advantage of *Input Hints* is that it leaves the control blank—the user is forced to consider the question and give an answer, and there is no chance the input field will be skipped over because it looks already filled in.

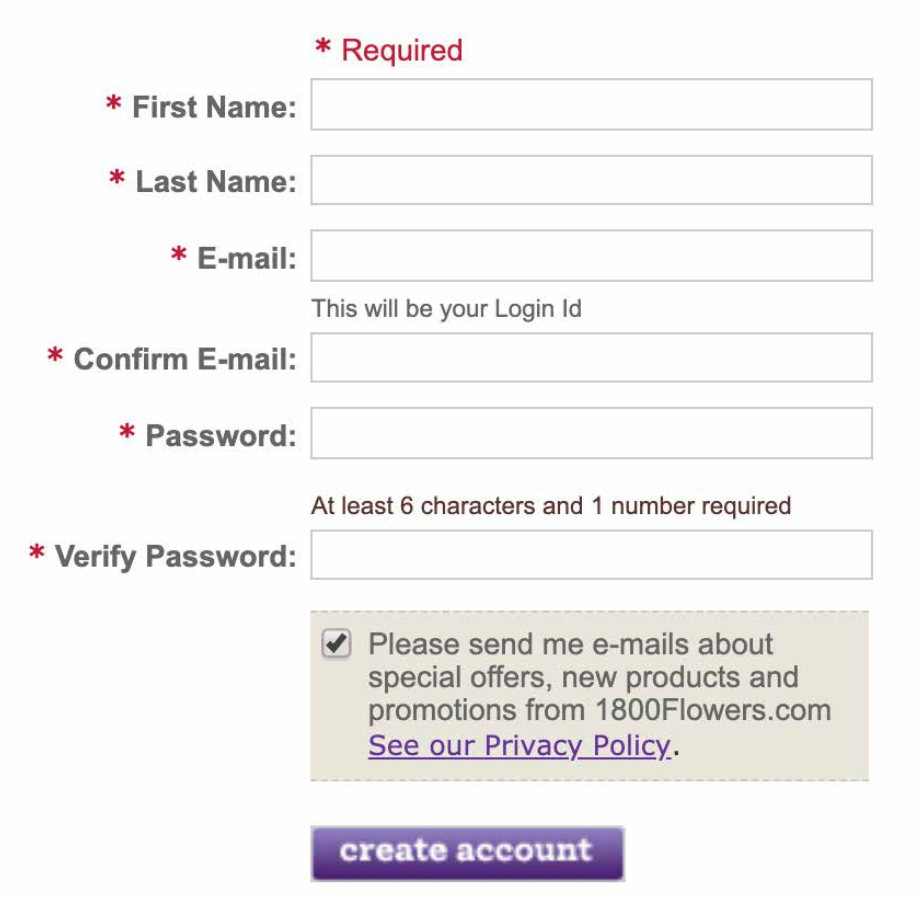


Figure 10-13. *The 1-800-Flowers registration screen*

The printing dialog boxes used by several Microsoft Office applications supply an *Input Hints* below a *Forgiving Format* text field—it takes page numbers, page ranges, or both (Figure 10-14). The hint explains how to use the Page Range print feature. The hint is very useful to anyone who's never had to use the Page Range option, but users who already understand it don't need to focus on the written text; they can just go straight for the input field.

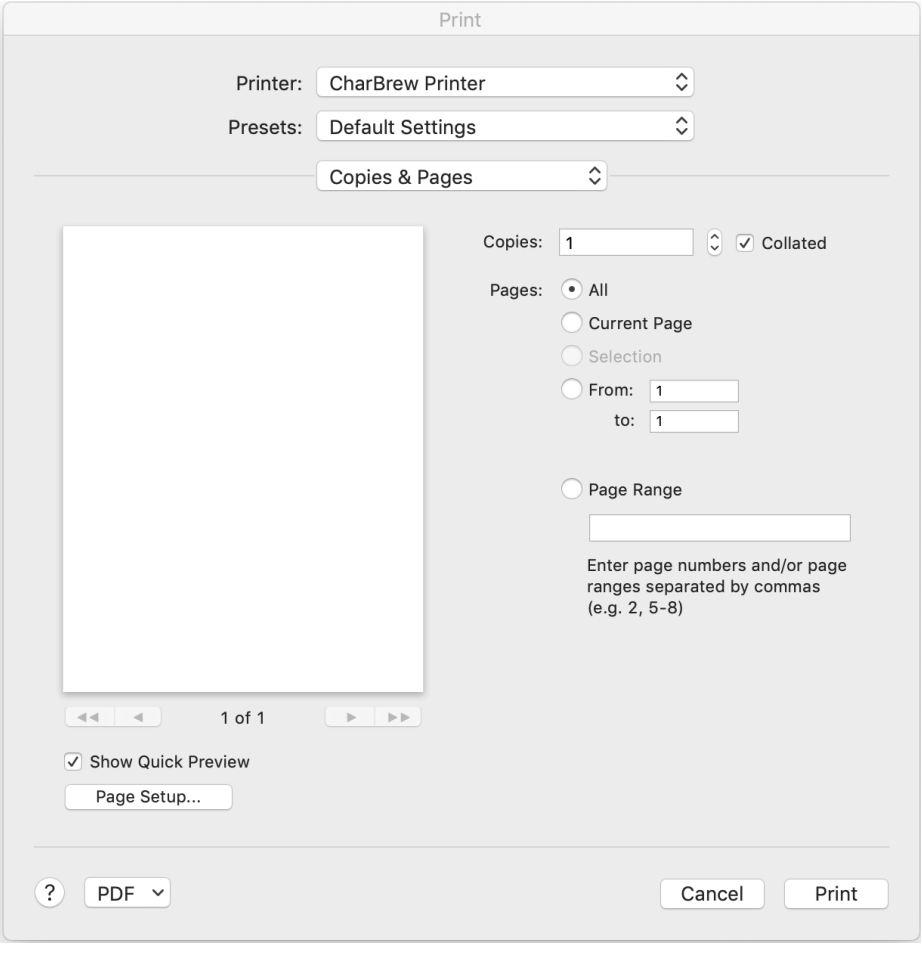


Figure 10-14. *Microsoft Word print dialog box*

Longer descriptions can be used in *Input Hints* when necessary. The examples from Gmail's registration page (Figure 10-15), are about as long as you'd want to put next to a text field. Indeed, Google offers additional information, but the user must select the link "Why we ask for this information"—which loads a web page in a new browser window. It's good customer service to explain what the company policy is, but most users will never follow a link when they're filling out a form, especially if they're trying to get through it quickly and don't have major privacy concerns. So, don't depend on linked pages to convey critical information.

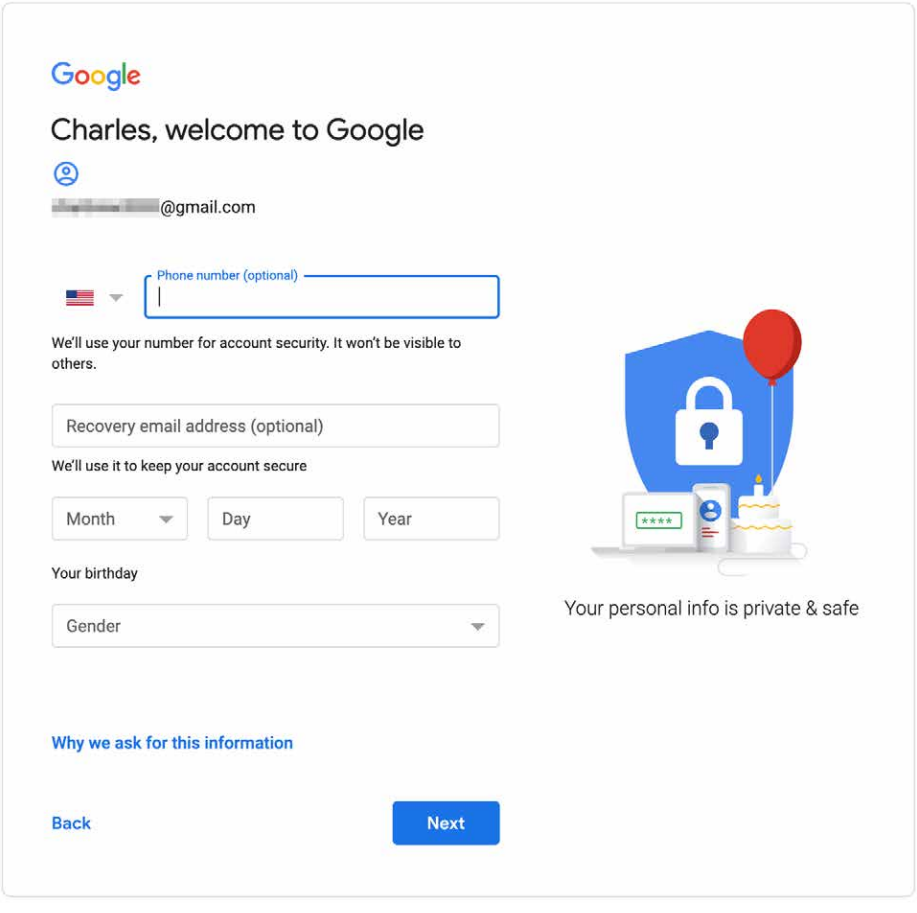


Figure 10-15. *Gmail registration page*

Apple places *Input Hints* on the right of the form, aligning the controls with their hints (Figure 10-16). This is a graceful way to structure a page full of *Input Hints*.

Figure 10-16. *Apple checkout screen*

Some forms show *Input Hints* only when the focus is on the text field itself, or only when a condition is met, as Yelp does (see Figure 10-17). This hint appears only after the user starts composing their password but has not included a required character. Otherwise, the hints are hidden. This is nice because the hidden hints don't clutter the interface or add visual noise; however, the user doesn't see them at all until they click (or tab into) the text field. If you use these, note that you must leave space for them in the interface, or have it expand.

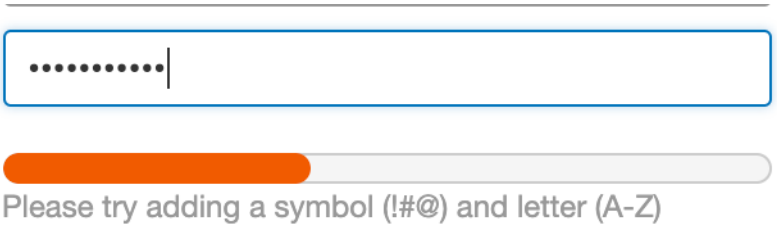


Figure 10-17. *Yelp password input hint*

Trunk Club (Figure 10-18) offers a fairly compact, vertical registration form. When the user selects the Password field, the form opens up to display the password hints. That is, the password hints appear only when the user tabs into the password field.

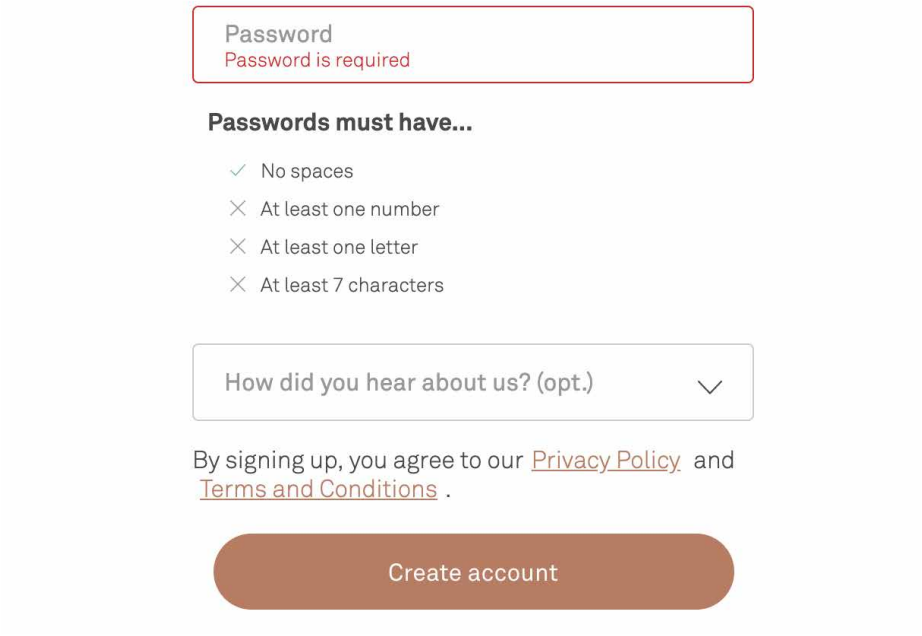


Figure 10-18. *Trunk Club*

Input Prompt

What

Prefill a text field with an example input or instructional text that helps the user with what to do or type. This is also called *placeholder text*.

Use when

The UI presents a text field, drop-down list, or combo box for input. Normally you would use a good default value, but you can't in this case—perhaps there is no reasonable default. The examples from the Blueprintjs UI Toolkit (Figure 10-19) show how prompts can help explain the function of the input field.

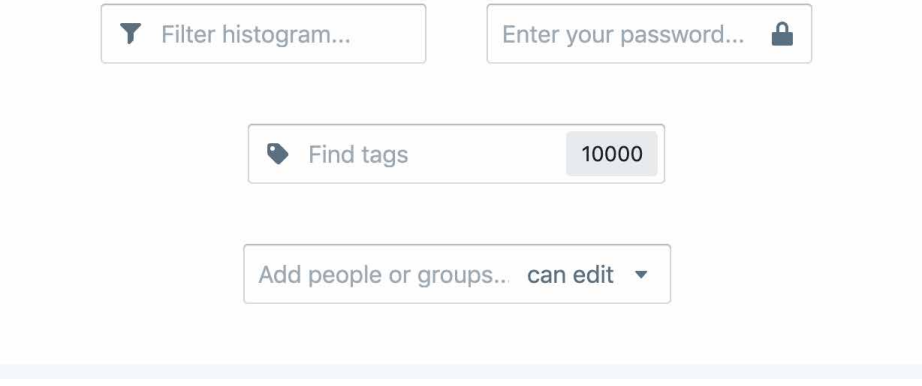


Figure 10-19. *Blueprintjs UI Toolkit; four example inputs with prompts*

Why

It helps make the UI self-explanatory. Like *Input Hints*, an *Input Prompt* is a handy way of supplying help information for controls whose purpose or format might not be immediately clear.

With *Input Hints*, someone quickly scanning the UI can easily ignore the hint (or miss it entirely). Sometimes this is your desired outcome. But an *Input Prompt* sits right where the user will type, so it can't be ignored. The advantage here is that the user doesn't have to guess whether they have to deal with this control or not—the control itself tells them to. (Remember that users don't fill out forms for fun—they'll do as little as needed to finish up and get out of there.) A question or an imperative "Fill me in!" is likely to be noticed.

Not the same as floating labels. Contemporary form design frequently uses "float labels" (see Brad Frost's article on Float Labels). This uses HTML label elements inside the form fields, very similar to input prompts, for the sake of elegance and simplicity. However, an input prompt disappears when the focus is on a form input that has prompt text. Floating labels move and change size but do not disappear on focus. Look again at Figure 10-15 (the Gmail registration screen); there are no labels outside the form inputs. The label "Phone number (optional)" and the next form input label "Recovery email address (optional)" are actually floating labels inside the input fields. In this case, input prompts are redundant. The phone number field has been selected, and instead of the text vanishing completely (formerly a huge drawback of input prompts), the label has moved up to the top edge of the input field. The user can still refer to it, and not need to remember it. On the other hand, only using floating labels can be problematic when you need to have both label text and input prompt text to explain different information.

How

Choose an appropriate prompt string:

- For a drop-down list, use Select, Choose, or Pick
- For a text field, use Type or Enter
- A short verb phrase

End it with a noun describing what the input is, such as "Choose a state," "Type your message here," or "Enter the patient's name." Put this phrase into the control where the value would normally be. (The prompt itself shouldn't be a selectable value in a drop down; if the user selects it, it's not clear what the software should do with it.)

Because the point of the exercise was to tell the users what they were required to do before proceeding, don't let the operation proceed until they've done it! As long as the prompt is still sitting untouched in the control, disable the button (or other device) that lets the user finish this part of the operation. That way, you won't need to throw an error message at the user.

For text fields, put the prompt back into the field as soon as the user erases the typed response.

Use *Good Defaults and Smart Prefills* instead of an *Input Prompt* when you can make a very accurate guess about what value the user will put in. The user's email address might already have been typed somewhere else, for instance, and the originating country can often be detected by websites.

Examples

Lyft (Figure 10-20) deploys a form with an *Input Prompt*. The *Input Prompt* displays by default, and disappears when the user begins typing. If the user deletes their entry, the input prompt text returns.

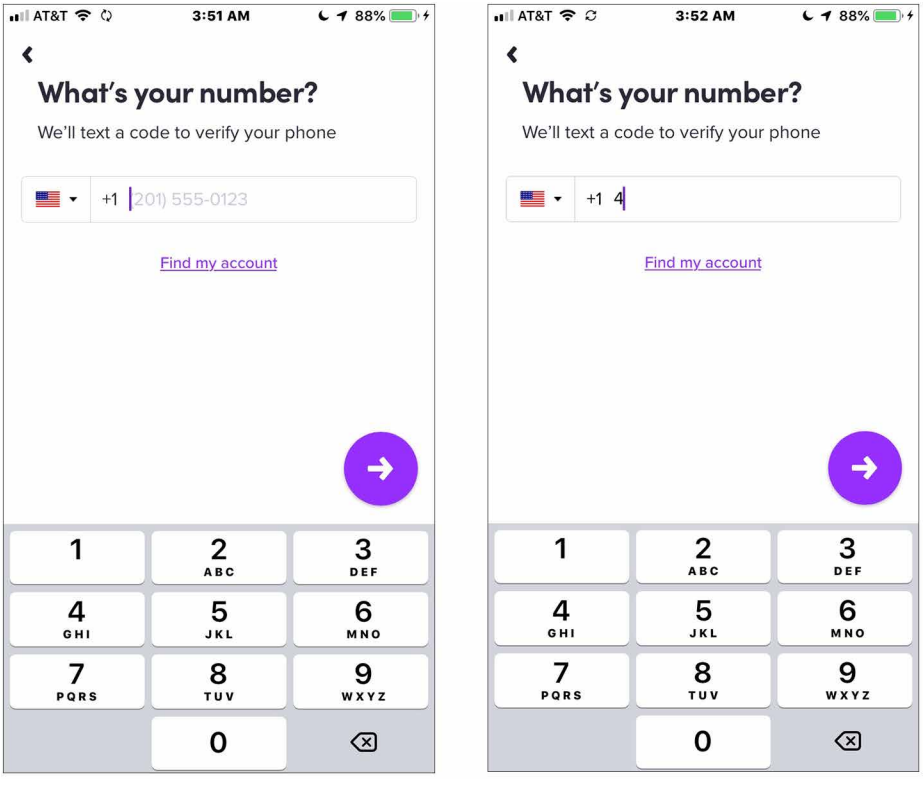


Figure 10-20. *Lyft mobile app input prompt*

Password Strength Meter

What

Give the user immediate feedback on the validity and strength of a new password while it is being typed.

Use when

The UI asks the user to choose a new password. This is quite common for site registrations. Your site or system cares about having strong passwords, and you want to actively help users choose good ones.

Why

Strong passwords protect both the individual user and the entire site, especially when the site handles sensitive information and/or social interactions. Weak passwords ought to be disallowed because they permit break-ins.

A *Password Strength Meter* gives immediate feedback to the user about their new password—is it strong enough or not? Do they need to make up a new one, and if so, with what characteristics (numbers, capital letters, etc.)? If your system is going to reject weak passwords, it's usually best to do it instantly, not after the user has submitted the registration form.

How

While the user types their new password or after keyboard focus leaves the text field, show an estimate of the password strength beside the text field. At minimum, display a text and/or graphic label indicating a weak, medium, or strong password, and special wording to describe a too-short or invalid password. Colors help: red for unacceptable, green or blue for good, and some other color (often yellow) in between. Yelp's password strength meter is a good example of this (Figure 10-21).

If you can, show additional text with specific advice on how to make a weak password better—a minimum length of eight characters (for instance), or the inclusion of numbers or capital letters. A user might get frustrated if they repeatedly fail to produce a valid password, so help them be successful.

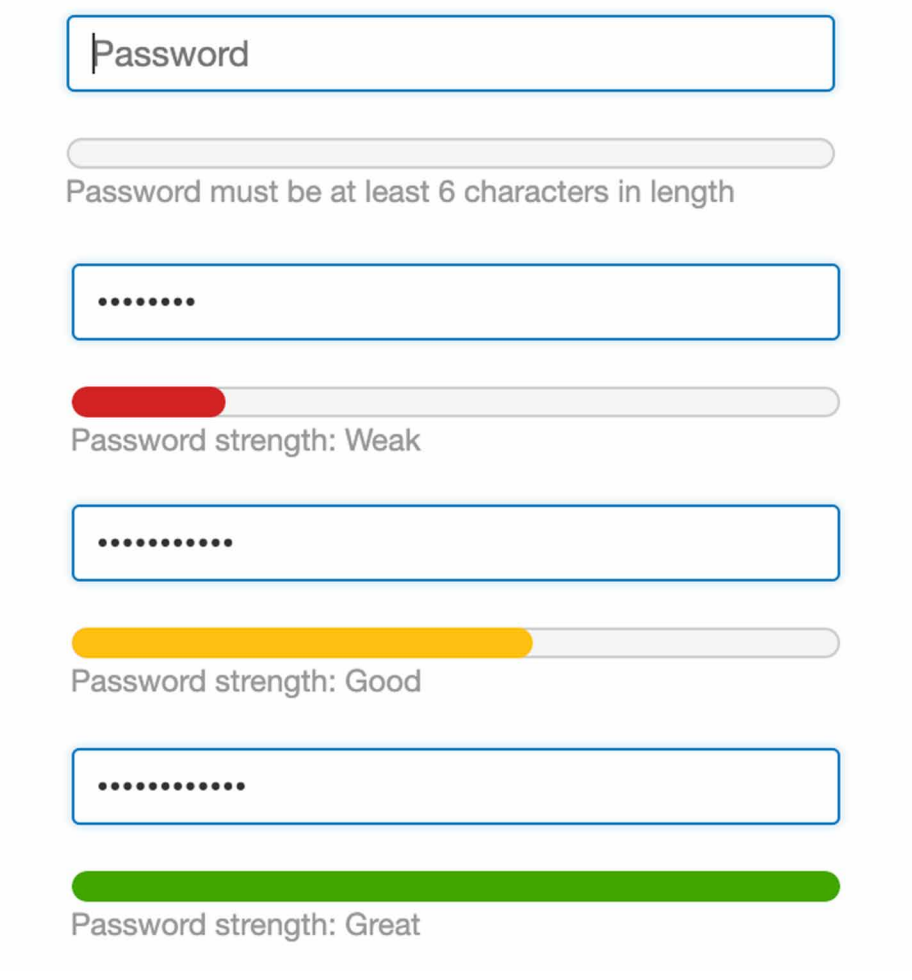


Figure 10-21. *Yelp password strength meter*

Also, the form containing the password field should use *Input Hints* or other text to explain this beforehand. A short reminder of good password heuristics can be useful to users who need reminders, and if your system will actually reject weak passwords, you should warn the user about it before they finish the form. Many systems require a minimum number of characters for a valid password, such as six or eight.

By default, don't show the password, but you might consider offering a toggle with which the user can view their password. Don't make suggestions of alternative passwords. General hints are all you can really give.

An explanation of password security is beyond the scope of a UI pattern. There are excellent online and print references for this topic, however, should you need to understand it more deeply.

Examples

GitHub's password strength meter (Figure 10-22) takes a contemporary approach that converts the password requirements or conditions (displayed as input hints) into a kind of meter. This could also be considered a kind of checkmark process. The user can read the requirements and follow them directly, of course. As they type, certain key words and phrases in the hint change from red (password requirement not met) to green (requirement is met). In this way, the user can adjust their password as they type to create a good one.

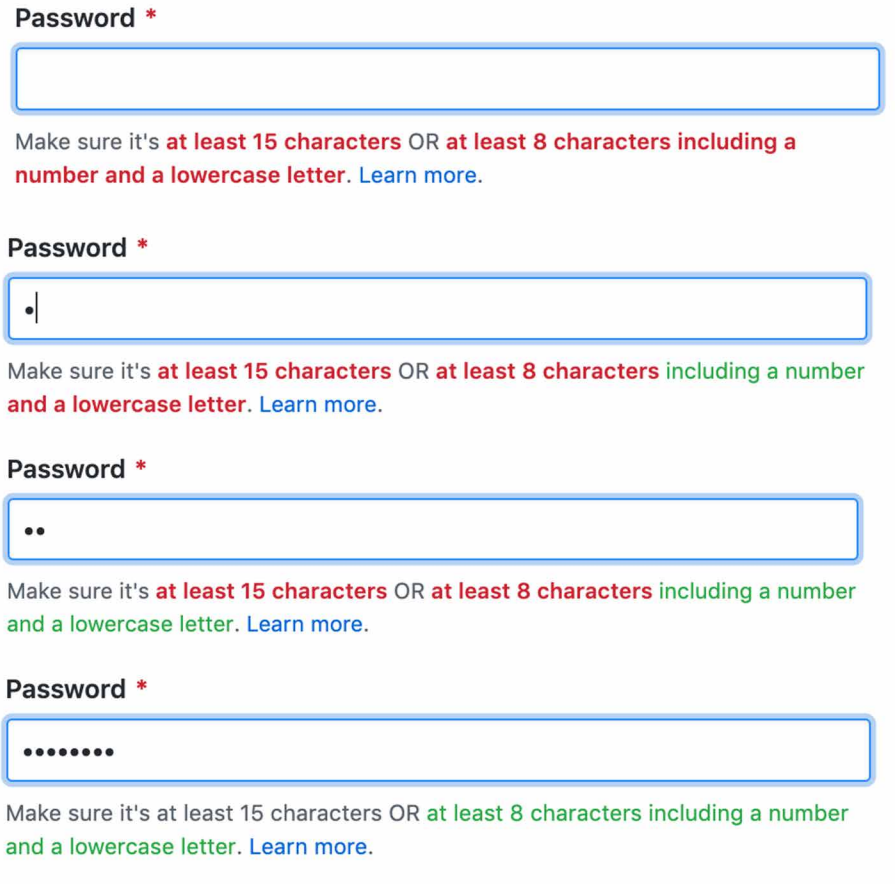


Figure 10-22. *GitHub password strength meter states*

Airbnb does a very similar thing, except that it makes the password meter into an explict checklist that updates as the user types (Figure 10-23)

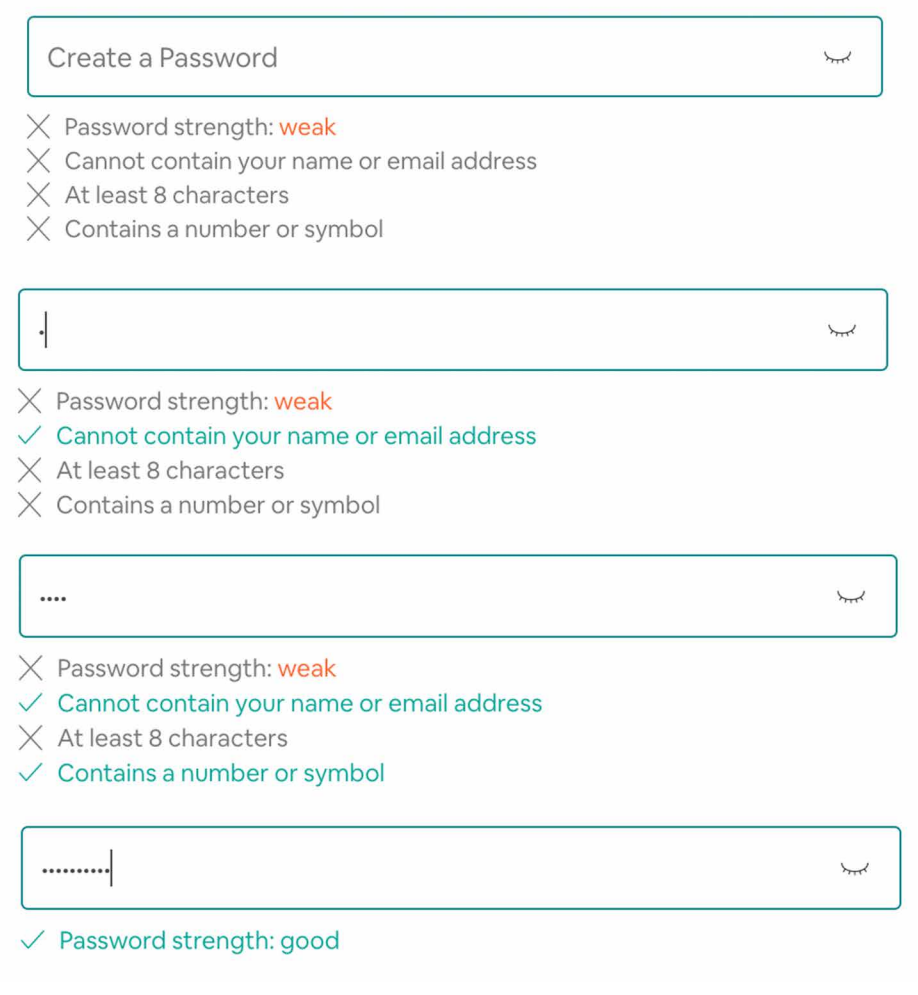


Figure 10-23. *Airbnb password strength meter states*

H&M uses a highly compact checklist style password strength meter (Figure 10-24). Again, it is presented as an input hint that is just a brief set of phrases. As the user enters a strong password, the requirements become checked.

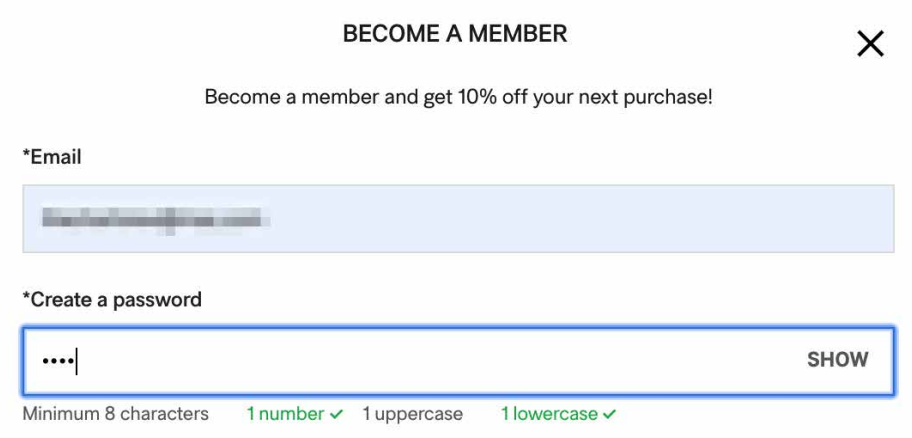


Figure 10-24. *H&M password strength meter states*

Retailer Menlo Club uses a minimalist approach, as well (Figure 10-25). It chose a classic thermometer-style password strength meter that uses just color and length. There is no input hint or instructions. As the user enters a password, the color bar "fills in" the meter, changing from red to yellow and finally to green.

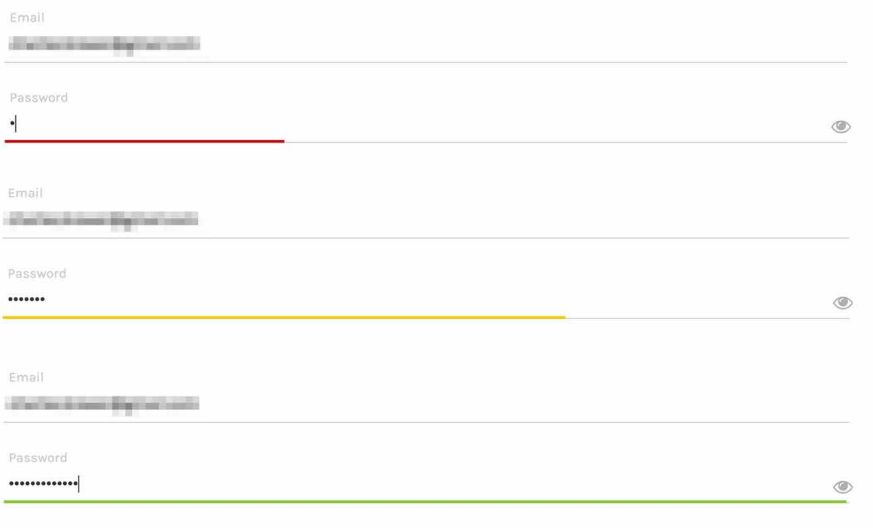


Figure 10-25. *Menlo Club password strength meter*

Glassdoor offers another minimalist approach (Figure 10-26). There is an input hint for the password, but the requirements are light. The password strength indicator has just two states: Red with warning icon, meaning not a compliant password, and a green checkmark, meaning the password is acceptable.

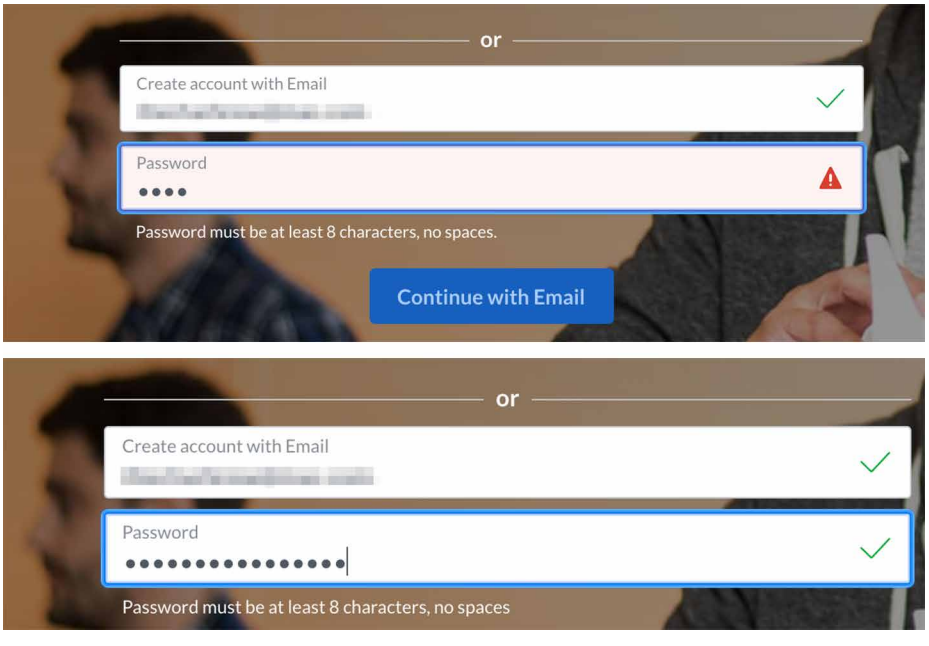


Figure 10-26. *Glassdoor password strength meter. This is reduced to just two password strength states: The rating is either "warning/error" or "OK."*

Autocompletion

Search becomes much more efficient and powerful if you add autocompletion. In real time, as the searcher enters their term(s) into the search input field, offer the most likely match based on the available string of characters. Offering up the most popular or most frequently searched terms is often part of making smart suggestions here. The benefit is that searchers save time because a match for their intent appears for selection without having to enter their entire search string. Amazon (Figure 10-27) offers an example of this capability in action.

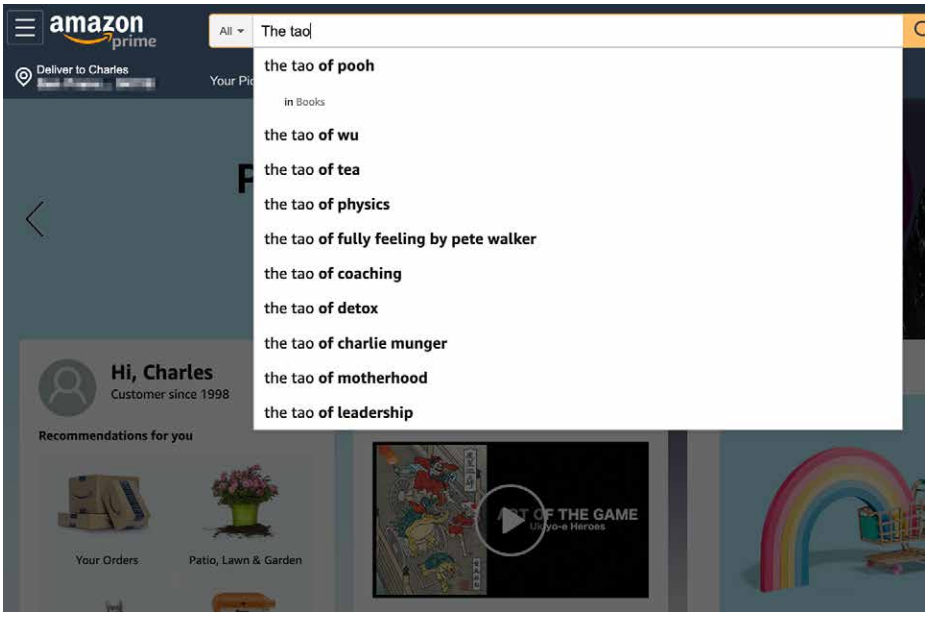


Figure 10-27. *Amazon autocomplete*

What

As the user types into a text field, anticipate the possible answers, show a selectable list of them, and automatically complete the entry when appropriate.

Use when

The user types something predictable, such as a URL, the user's own name or address, today's date, or a filename. You can make a reasonable guess as to what the user is attempting to type—perhaps there's a saved history of things this user has previously typed, for instance, or perhaps they are picking from a set of preexisting values, such as a list of filenames in a directory.

Search boxes, browser URL fields, email fields, common web forms (such as site registration or purchase), text editors, and command lines all seem to be much easier to use when supported by *Autocompletion*.

Predictive and social algorithms now drive autocomplete, as well. Popular, trending or most commonly entered search terms populate search engine autocompletes.

Why

Autocompletion saves time, energy, cognitive burden, and wrist strain for the user. It turns a laborious typing effort into a simple pick list (or less, if a single completion can be reliably supplied). You can thus save your users countless seconds of work, and contribute to the good health of thousands of wrists.

When the typed entries are long and difficult to type (or remember), like URLs or email addresses, *Autocompletion* is quite valuable. It reduces a user's memory burden by supplying "knowledge in the world" in the form of a drop-down list. An additional benefit can be error prevention: the longer or stranger the string that must be typed, the greater the odds of the user making a typographical error. Autocompleted entries have no such problems.

For mobile devices, it's even more valuable. Typing text on a tiny device is no fun; if a user needs to enter a long string of letters, appropriate *Autocompletion* can save them a great deal of time and frustration. Again, email addresses and URLs are excellent candidates to support mobile email and web usage.

Autocompletion is also common in text editors and command-line UI . As users type commands or phrases, the application or shell might offer suggestions for completion. Code editors and operating system shells are well suited for this, because the language used is limited and predictable (as opposed to a human language, such as English); it's therefore easier to guess what the user tries to type.

Finally, lists of possible autocompletions can serve as a map or guide to a large world of content. Search engines and site-wide search boxes do this well—when the user types the beginning of a phrase, an *Autocompletion* drop-down list shows likely completions that other people have typed (or that refer to available content). Thus, a searcher can get a view into the public mental landscape, the trends among the vast numbers of people online. Very often they are seeking the same things. This offers a curious or uncertain user a way to navigate based on the wisdom (or curiosity) of crowds.

How

With each additional character that the user types, the software quietly forms a list of the possible completions to that partially entered string. If the user enters one of a limited number of possible valid values, use that set of valid values. If the possible values are wide open, one of these might supply completions:

- Previous entries typed by this user, stored in a preferences or history mechanism
- Common phrases that many users have used in the past, supplied as a built-in "dictionary" for the application

- Possible matches drawn from the content being searched or perused, as for a site-wide search box
- Other artifacts appropriate to the context, such as company-wide contact lists for internal email
- Most popular or frequently submitted request strings

From here, you can approach the interaction design of *Autocompletion* in two ways. One is to show the user a list of possible completions on demand—for example, by pressing the Tab key—and let the user choose one explicitly by picking from that list. Many code editors do this. It's probably better used when the user would recognize what they want when they see it but might not remember how to type it without help. "Knowledge in the world is better than knowledge in the head."

The other way is to wait until there's only one reasonable completion and then put it in front of the user, unprompted. Word does this with a tool tip; many forms do it by filling in the remainder of the entry but with selection turned on, so another keystroke would wipe out the autocompleted part. Either way, the user gets a choice about whether to retain the *Autocompletion* or not—and the default is to not keep it.

Make sure that *Autocompletion* doesn't irritate users. If you guess wrong, the user won't like it—they then must erase the *Autocompletion* and retype what they meant in the first place, avoiding having *Autocompletion* pick the wrong completion yet again. These interaction details can help prevent irritation:

- Always give the user a choice to take the completion or not take it; default to "no."
- Don't interfere with ordinary typing. If the user intends to type a certain string and just keeps typing in spite of the attempts at autocompletion, make sure the result is what the user intended to type.
- If the user keeps rejecting a certain autocompletion in one place, don't keep offering it. Let it go at some point.
- Guess correctly.

Examples

Many email clients, of course, use *Autocompletion* to help users fill in To: and CC: fields. They generally draw on an address book, contacts list, or list of addresses with which you've exchanged email. The example from Apple Mail (Figure 10-28) shows a single completion suggested upon typing the letters f i d; the completed text is automatically highlighted, so a single keystroke can get rid of it. You can thus type straight "through" the completion if it's wrong.

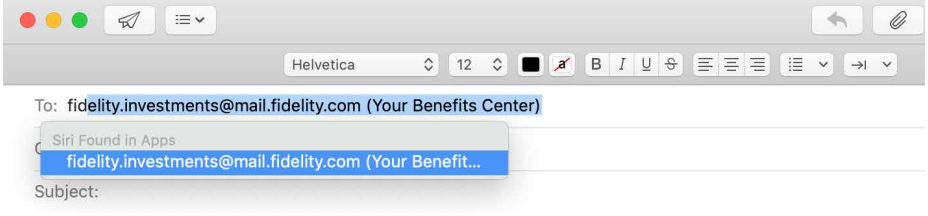


Figure 10-28. *Apple Mail autocomplete*

Google's Gmail offers autocomplete for email composition (Figure 10-29). Pressing the right arrow key completes the sentence with the suggestion.

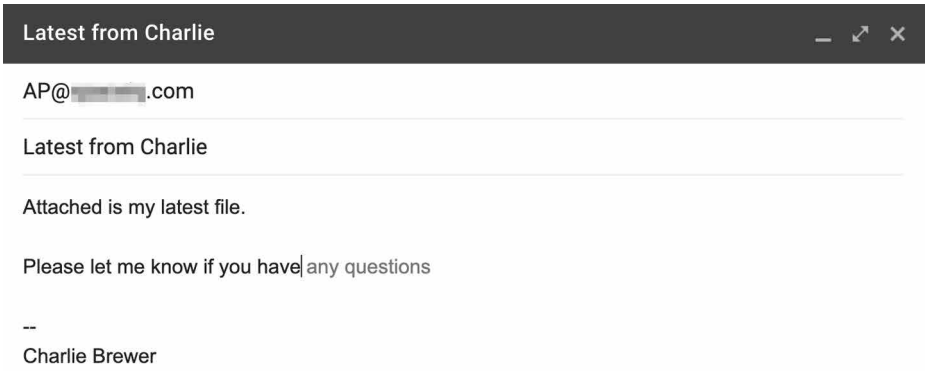


Figure 10-29. *Gmail autocompletion in the body of the email*

Drop-down lists of *Autocompletion* possibilities can take many forms. Figures 10-30 through 10-36 show examples of drop-down list formatting. Strategies for autocompletion can focus on specific data types and information only, or they can be more expansive, including searching varied data sources. *Autocompletion* can also be an opportunity for promotions and paid placement.

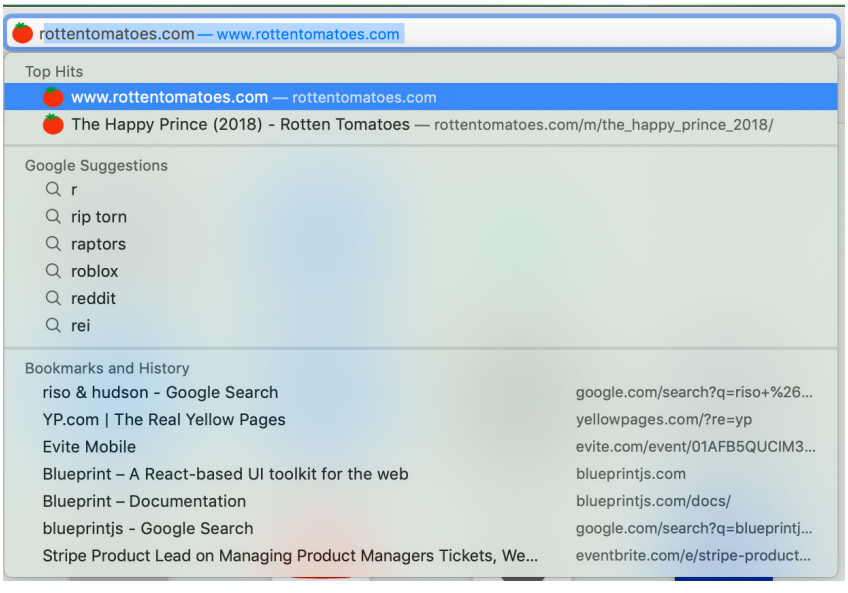


Figure 10-30. *Apple Safari browser autocomplete*

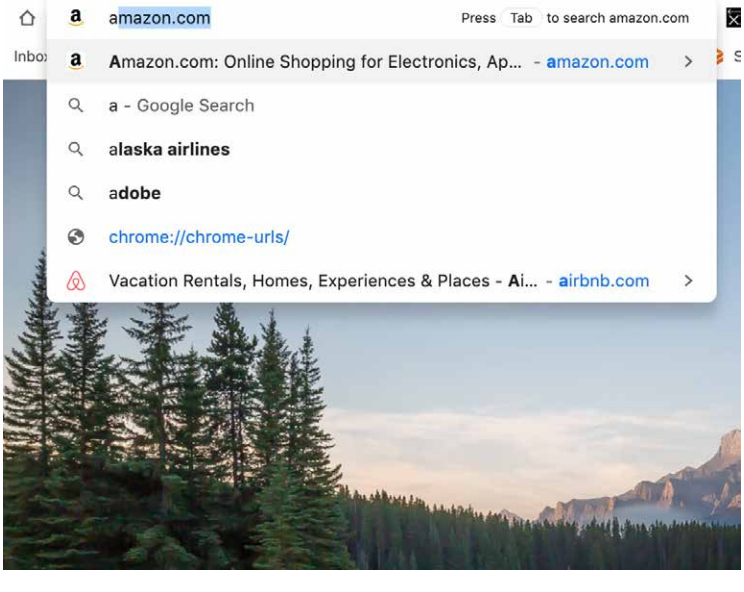


Figure 10-31. *Google Chrome browser autocomplete*

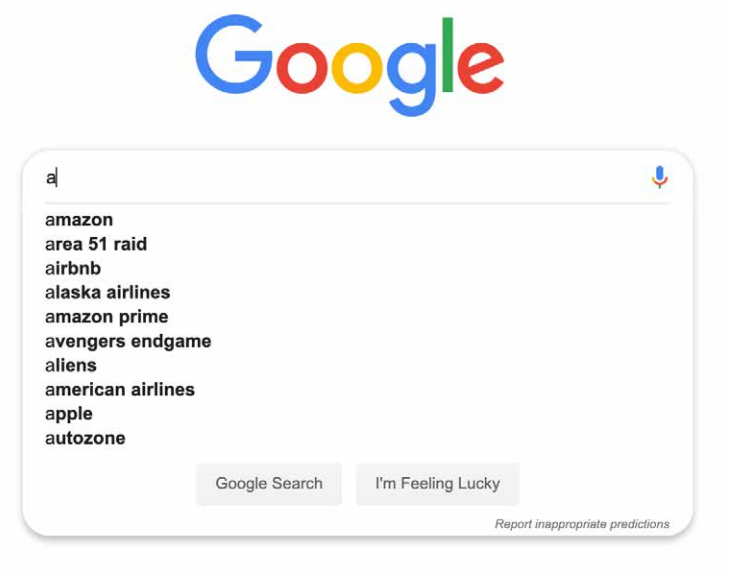


Figure 10-32. *Google.com Search autocomplete*

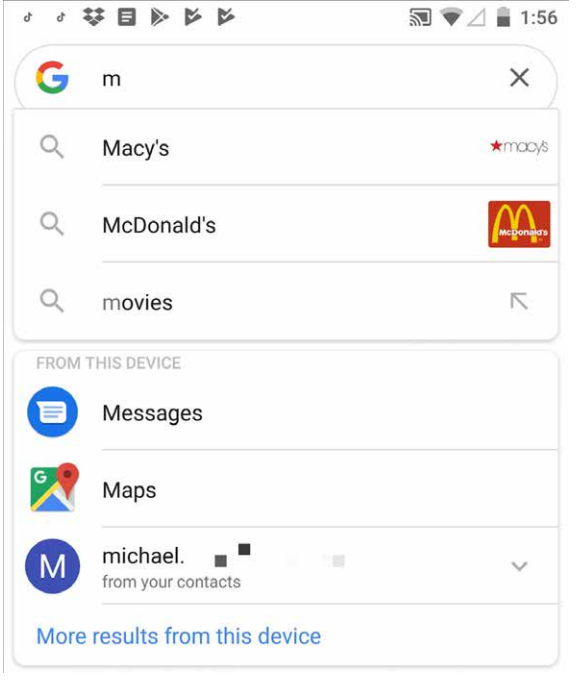


Figure 10-33. *Android Search autocomplete*

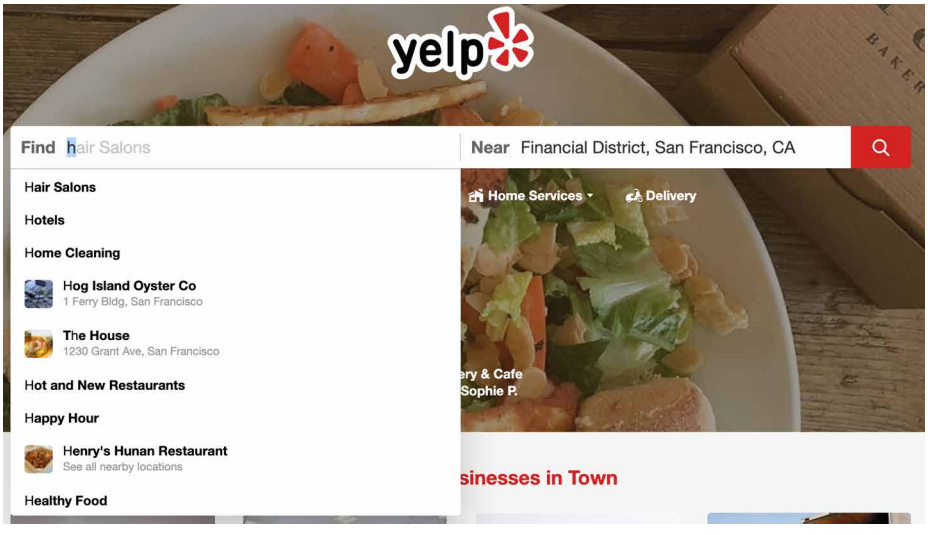


Figure 10-34. *Yelp.com Search autocomplete*

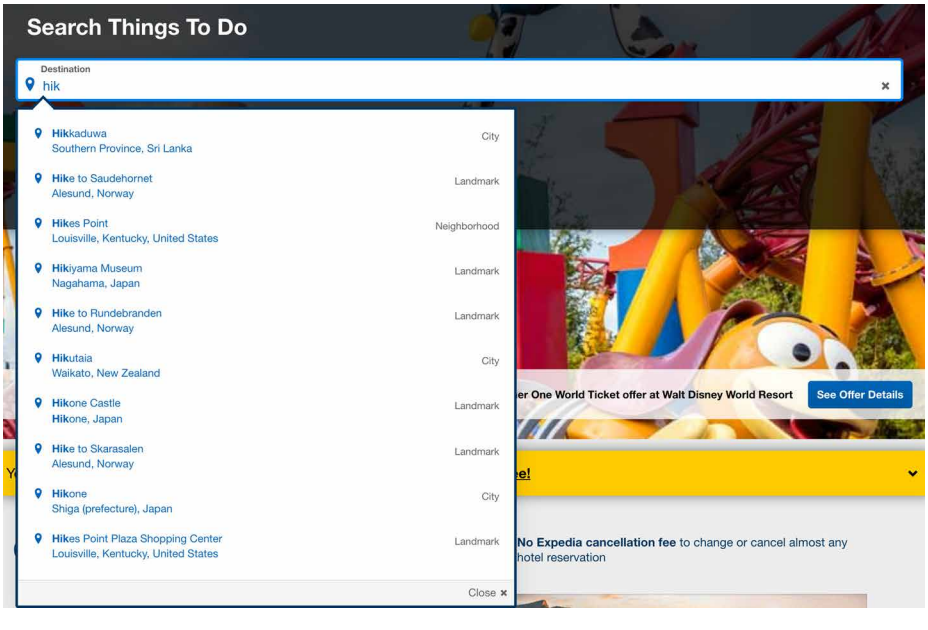


Figure 10-35. *Expedia.com "Things to Do" Search autocomplete*

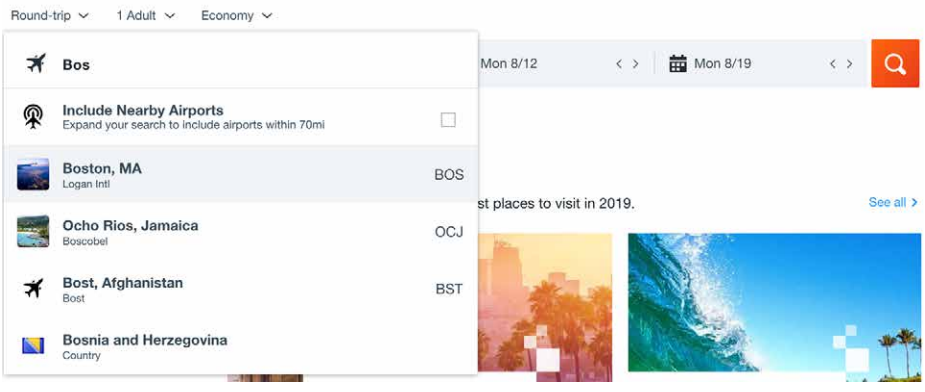


Figure 10-36. *Kayak.com Search autocomplete*

Drop-down Chooser

What

Expand the concept of a menu by using a drop-down list or pop-up panel that contains a more complex or hierarchical selection UI.

Use when

The user needs to supply input that is a choice from a set (such as in the color example in Figure 10-37), a date or time, a number, or anything other than free text typed at a keyboard. You want to provide a UI that supports that choice—a nice visual rendering of the choices, for instance, or interactive tools—but you don't want to use space on the main page for that; a tiny space showing the current value is all you want.

Why

Most users are very familiar with the drop-down list control (called a combo box when used with a free-typing text field). Many applications successfully extend this concept to drop-down lists that aren't simple lists, such as trees, 2D grids, and arbitrary layouts. Users seem to understand them with no problem, as long as the controls have down-arrow buttons to indicate that they open when clicked.

Drop-down Choosers encapsulate complex UIs in a small space, so they are a fine solution for many situations. Toolbars, forms, dialog boxes, and web pages of all sorts use them now. The page the user sees remains simple and elegant, and the chooser UI shows itself only when the user requests it—an appropriate way to hide complexity until it is needed.

How

For the *Drop-down Chooser* control's "closed" state, show the current value of the control in either a button or a text field. To its right, put a down arrow. This can be in its own button or not, as you see fit; experiment and see what looks good and makes sense to your users. A click on the arrow (or the whole control) brings up the chooser panel, and a second click closes it again.

Design a chooser panel for the choice the user needs to make. Make it relatively small and compact; its visual organization should be a familiar format, such as a list, a table, an outline-type tree, or a specialized format like a calendar or calculator (see the examples in the next section). See Chapter 7 for a discussion of list presentation.

Scrolling the panel is fine if the user understands that it's a choice from a large set, such as a file from a filesystem, but keep in mind that scrolling one of these pop-up panels is not easy for people without perfect dexterity!

Links or buttons on the panel can in turn bring up secondary UIs—for example, color-chooser dialog boxes, file-finder dialog boxes, or help pages—that help the user choose a value. These devices usually are modal dialog boxes. In fact, if you intend to use one of these modal dialog boxes as the primary way the user picks a value (say, by launching it from a button), you could use a *Drop-down Chooser* instead of going straight to the modal dialog box. The pop-up panel could contain the most common or recently chosen items. By making frequently chosen items so easy to pick, you reduce the total time (or number of clicks) it takes for an average user to pick values.

Examples

This first example shows several drop-down selectors in Microsoft Word (Figure 10-37).

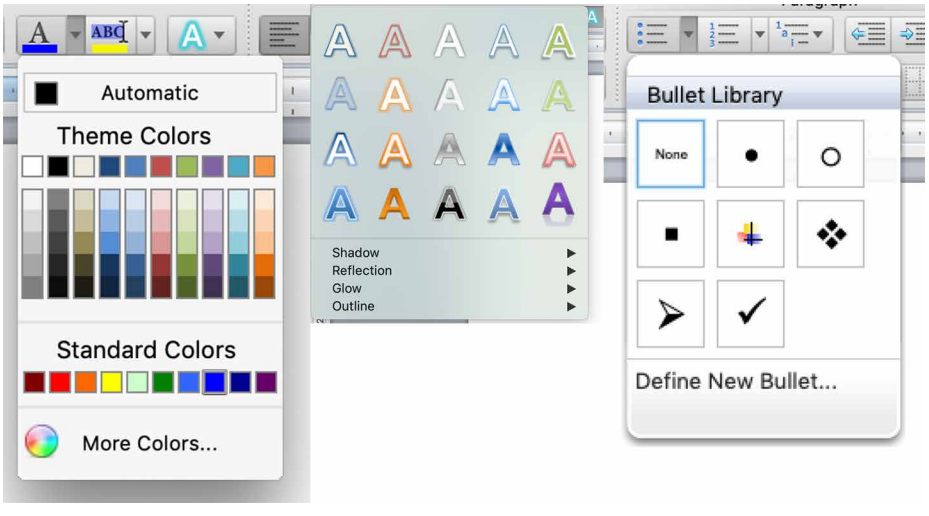


Figure 10-37. *Microsoft Word drop-down choosers*

Photoshop's compact, interaction-rich toolbars use *Drop-down Chooser* heavily. Figure 10-38 shows three examples: Brush, Marquee Tool, and Opacity. The Brush chooser is a selectable list with a twist—it has extra controls such as a sliders, brush direction dial, icons of presets, and an expandable folder directory of brushes for yet more choices. Selecting the lower-right corner of the Marquee Tool opens the options menu, showing the special versions of that, too. The Opacity chooser is a simple slider, and the text field above it echoes its value.

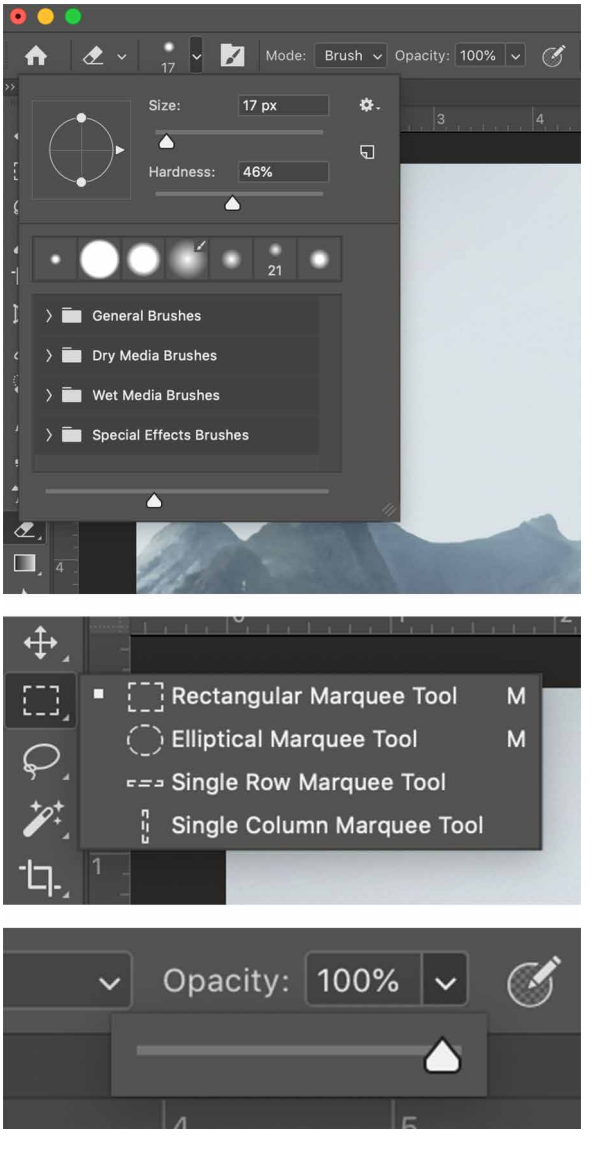


Figure 10-38. *Photoshop drop-down choosers*

Design tool Sketch offers a multifunction drop-down chooser for its color picker (Figure 10-39). It offers robust color space configuration with hue and shade controls, numerical color values, and a grid of saved colors. Several menus offer access to even more functionality.

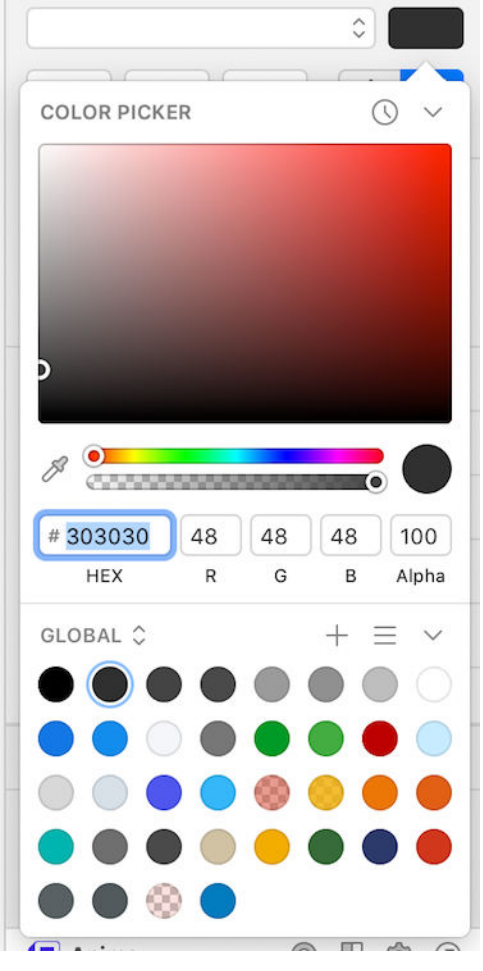


Figure 10-39. *Sketch drop-down chooser*

The *Thumbnail Grid* pattern (Chapter 7) is often used in *Drop-down Chooser* in place of a text-based menu. The examples fromPowerPoint (Figure 10-40) and Keynote (Figure 10-41) demonstrate several styles of *Thumbnail Grid*.

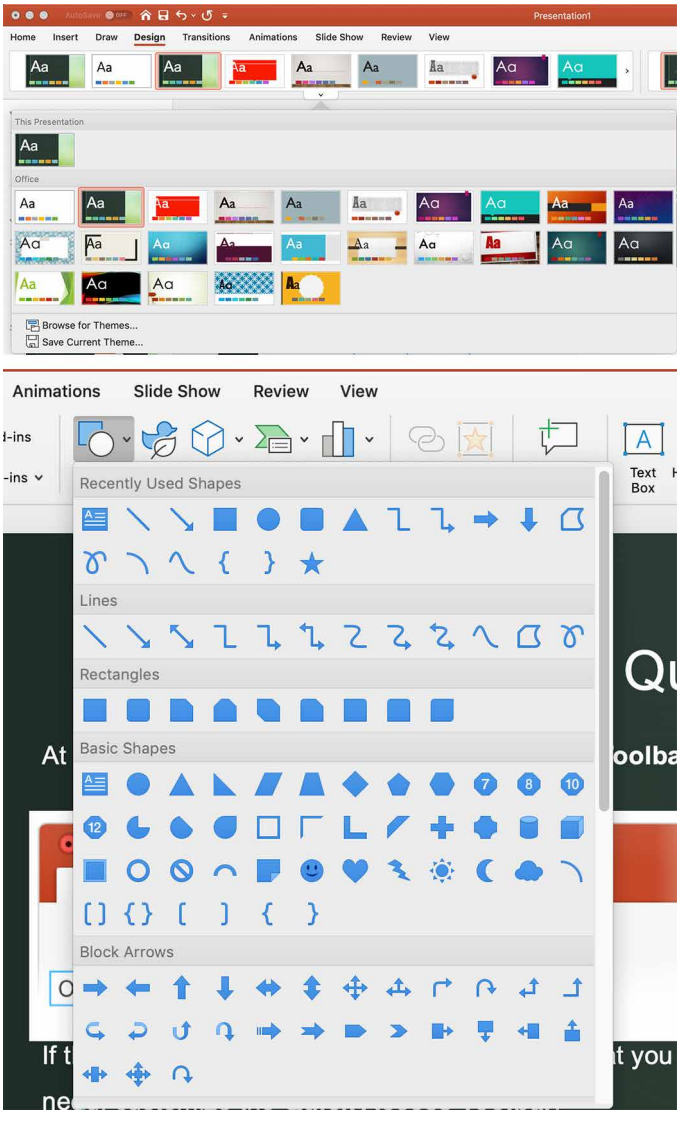


Figure 10-40. *Microsoft PowerPoint drop-down choosers with thumbnail grid*

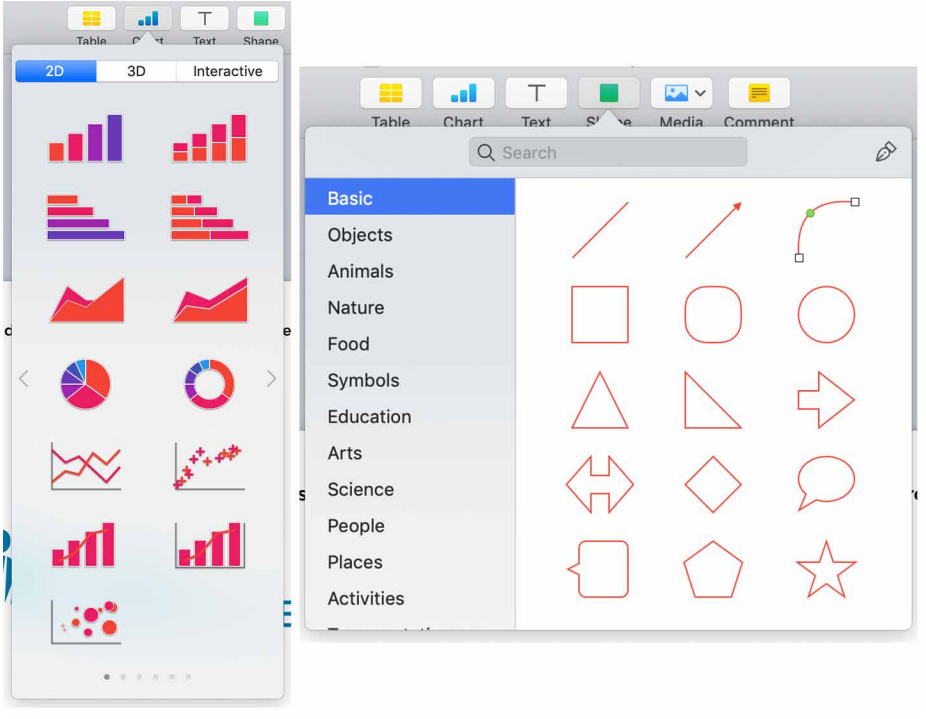


Figure 10-41. *Apple Keynote drop-down choosers with thumbnail grid*

List Builder

What

A pattern for creating a complicated selection set from a larger source set of objects. A list builder has the "source" and the "destination" lists visible in the same widget. This lets the user move items between them, via buttons or drag-and-drop. This pattern is also called a two column multiselector.

Use when

You're asking the user to create a list of items by choosing them from another list. The source list may be long—too long to easily show as a set of checkboxes, for instance.

Why

The key to this pattern is to show both lists on the same page. The user can see what's what—they don't need to jump to and from a modal chooser dialog box, for instance.

A simpler alternative to *List Builder* might be a single list of checkbox items. Both solve the "select a subset" problem. But if you have a very large source list (such as an entire filesystem), a list of checkboxes doesn't scale—the user can't easily see what's been checked off, and thus might not get a clear picture of what they selected. The user must continually scroll up and down to see it all.

How

Put the source list and the destination list next to each other, either left to right or top to bottom. Between the two lists, put Add and Remove buttons (unless your users find drag-and-drop to be obvious, not requiring explanation). You could label the buttons with words, arrows, or both. Clicking list elements can also cause them to jump to the opposite list.

This pattern provides room for additional functionality, too. Both the source list and the destination list may be searchable. The source list could be a multilevel directory with an open/close file folder structure. The destination list could be ordered, with tools to move or drag and drop the items into the desired order from top to bottom.

Depending on what kind of items you deal with, you could either move the items literally from the source to the destination—so the source list "loses" the item—or maintain a source list that doesn't change. A listing of files in a filesystem shouldn't change; users would find it bizarre if it did because they see such a list as a model of the underlying filesystem, and the files aren't actually deleted. That's a judgment call.

Give the lists multiple-selection semantics instead of single-selection, so users can move large numbers of items from list to list.

Examples

Most modern implementations of this pattern depend upon drag-and-drop or simple clicking to move items between areas; it's important that the user receive visual confirmation that selected items are moved to the destination list. The web application multiselector, *multiselect.js*, shows a fast and simple UI for creating lists (Figure 10-42). This open source developer project has developed a typical two-column multiselect widget for use in web applications.

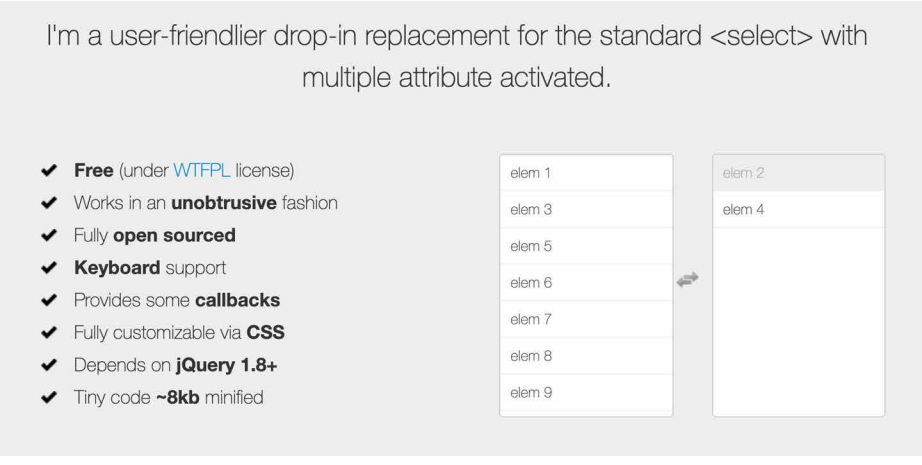


Figure 10-42. *Loudev.com multiselect.js*

Drupal offers a similar component, a multiselect list builder widget, as part of its enterprise content management system (Figure 10-43).

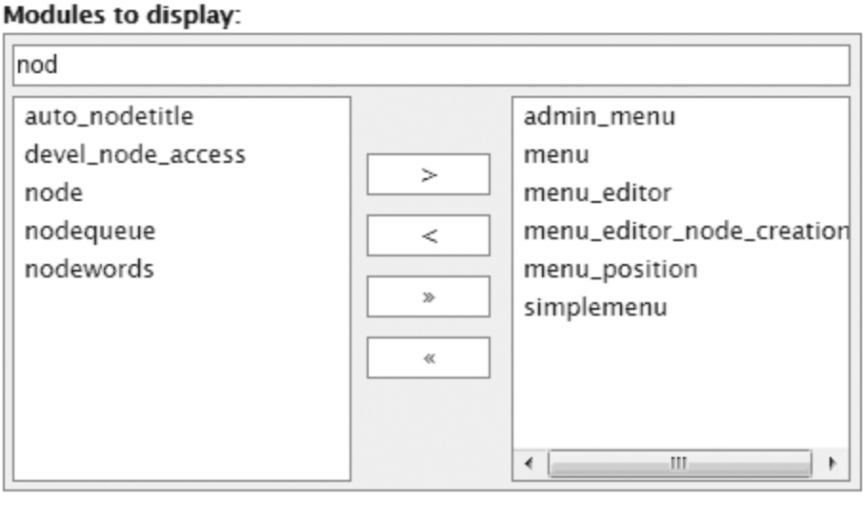


Figure 10-43. *Drupal content management system*

Graphical list building works, too. Lightroom (Figure 10-44) demonstrates a more contemporary approach to a *List Builder*: you can drag items from a potentially long list of source images into a "batch" group in order to perform operations on all batched images at once. Large text informs the user what to do at critical moments in the interaction, such as starting a new batch or removing an image from the batch.

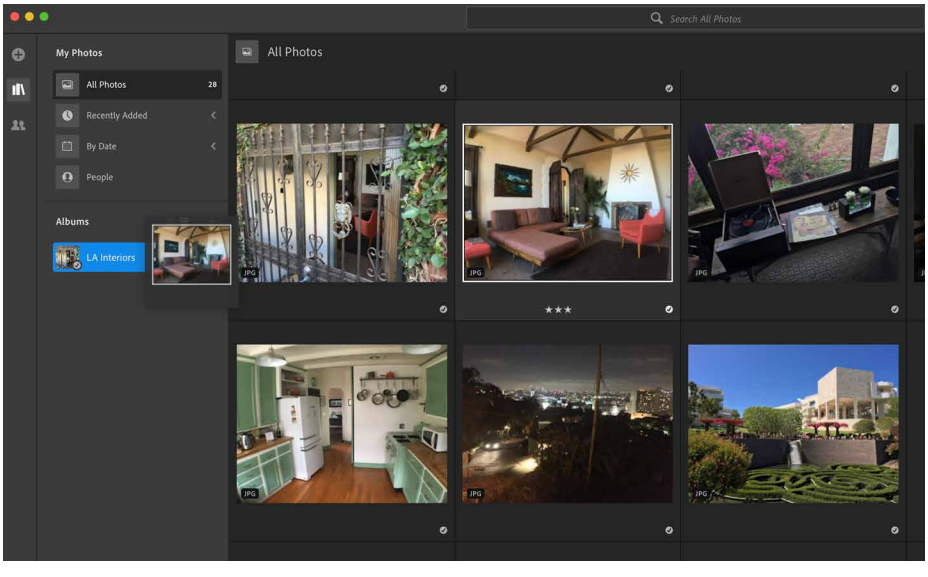


Figure 10-44. *Adobe Lightroom*

Good Defaults and Smart Prefills

What

Use default values for form elements that are intended to save the user time and effort in completing the form. Good defaults draw from a number of sources of data for prefills: previously entered data from the session, information from the user's account, current location, current data and time, and other values that the designer can identify as having a high probability of making it easier and quicker for the user to complete a form.

Use when

Your UI asks the user any questions requiring form-like input (such as text fields or radio buttons), and you want to reduce the amount of work that users need to do. Perhaps most users will answer in a certain way, or the user has already provided enough contextual information for the UI to make an accurate guess. For technical or

semirelevant questions, maybe the user can't be expected to know or care about the answer, and "whatever the system decides" is OK.

But supplying defaults is not always wise when answers might be sensitive or politically charged, such as passwords, gender. or citizenship. Making assumptions like that, or prefilling fields with data you should be careful with can make users uncomfortable or angry. (And for the love of all that is good in the world, don't leave "Please send me advertising email" checkboxes selected by default!)

Why

By providing reasonable default answers to questions, you save the users work. It's really that simple. You spare the user the effort of thinking about, or typing, the answer. Filling in forms is never fun, but if having default answers provided halves the time it takes the user to work through the form, they'll be grateful.

Even if the default isn't what the user wants, at least you offered an example of what kind of answer is asked for. That alone can save the user a few seconds of thought—or, worse, an error message.

Sometimes, you might run into an unintended consequence of *Good Defaults*. If a user can skip over a field, that question might not "register" mentally with the user. They might forget that it was asked; they might not understand the implications of the question, or of the default value. The act of typing an answer, selecting a value, or clicking a button forces the user to address the issue consciously, and that can be important if you want the user to learn the application effectively.

How

Prefill the text fields, combo boxes, and other controls with a reasonable default value. You could do this when you show the page to the user for the first time, or you could use the information the user supplies early in the application to dynamically set later default values. (For instance, if someone supplies a US Zip Code, you can infer the state, country, and municipality from just that number.)

Don't choose a default value just because you think you shouldn't leave any blank controls. Do so only when you're reasonably sure that most users, most of the time, won't change it—otherwise, you will create extra work for everybody. Know your users!

Occasional-use interfaces such as software installers deserve a special note. You should ask users for some technical information, such as the location of the install, in case they want to customize it. But 90% of users probably won't. And they won't care where you install it, either—it's just not important to them. So, it's perfectly reasonable to supply a default location.

Examples

Kayak (Figure 10-45) supplies default values when a user begins a search for flights. Kayak kindly suggests a week-long vacation: it prefills the flight search with duration of one week, starting a month from now. The other defaults are quite reasonable: a round-trip economy flight with one traveler is common, and the "From" city can be derived from either the user's geographic location or the user's previous searches. Kayak goes one step further by prefilling the departure date (a month in the future) and the return (one week later). The effect of having all these defaults is that the user spends less time thinking about those parts of the form, and they get a quicker path to their immediate goal—the search results.

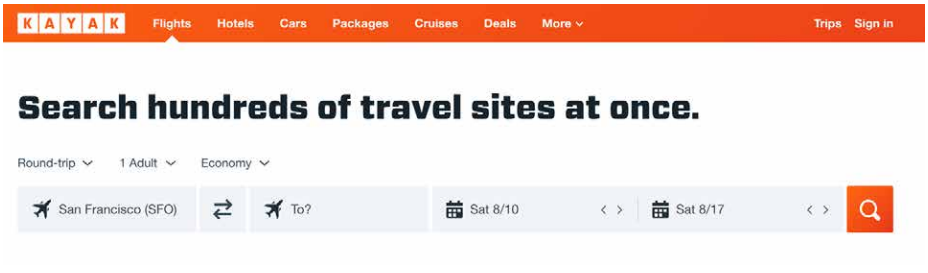


Figure 10-45. *Kayak*

The Fandango mobile app (Figure 10-46) uses the current location and date as its default movie search parameters. When the person opens this app, it uses the current date and location to generate a list of movies for today near where you are. It goes one step further: the default for this screen has a small "toast" or pop-up banner at the bottom (on top of the search results), offering a look at what's playing today at the very nearest movie theater.

Figure 10-46. *Fandango (iOS)*

When creating a new image file in Photoshop, it starts by default with the OS clipboard (Figure 10-47). The assumption here is that the user has just created a screenshot and is in the process of editing that image. So Photoshop takes the width and height of the image in the clipboard and uses that to prefill the dimensions of the Create New file, a smart way to save time. No worry that the image and the canvas will have a size mismatch.

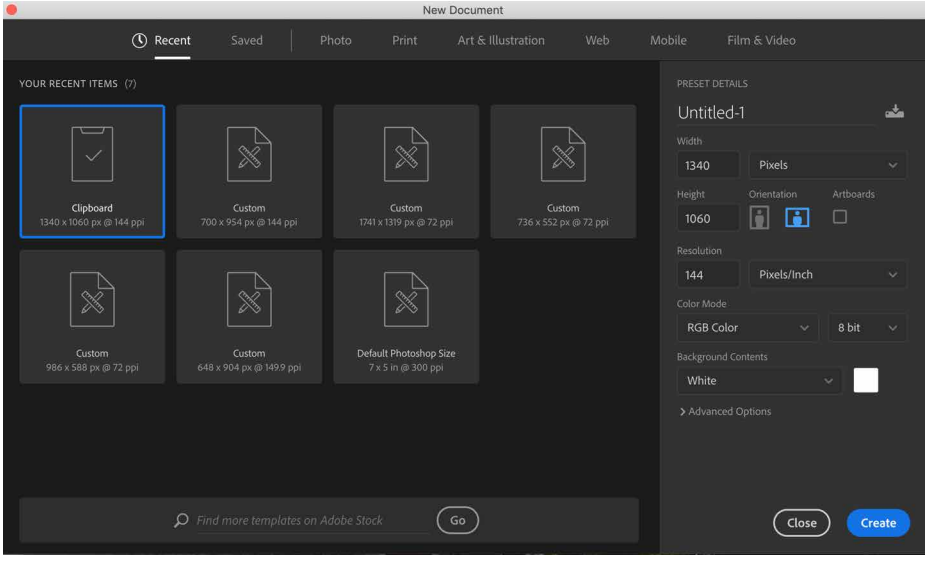


Figure 10-47. *A Create New dialog in Photoshop CC*

When an image canvas is resized in Photoshop, the dialog box shown in Figure 10-48 appears. The original image was 1340 × 1060, as shown. These dimensions become the default Width and Height, which is very convenient for several use cases. The application prefills the width and height of the current image as the starting point for a different canvas size. If the user wants to put a thin frame around the image, they can start with the existing dimensions and increase them by just two pixels each; if they want to make the image canvas wider but not taller, they need only change the Width field; or just click OK now and nothing changes.

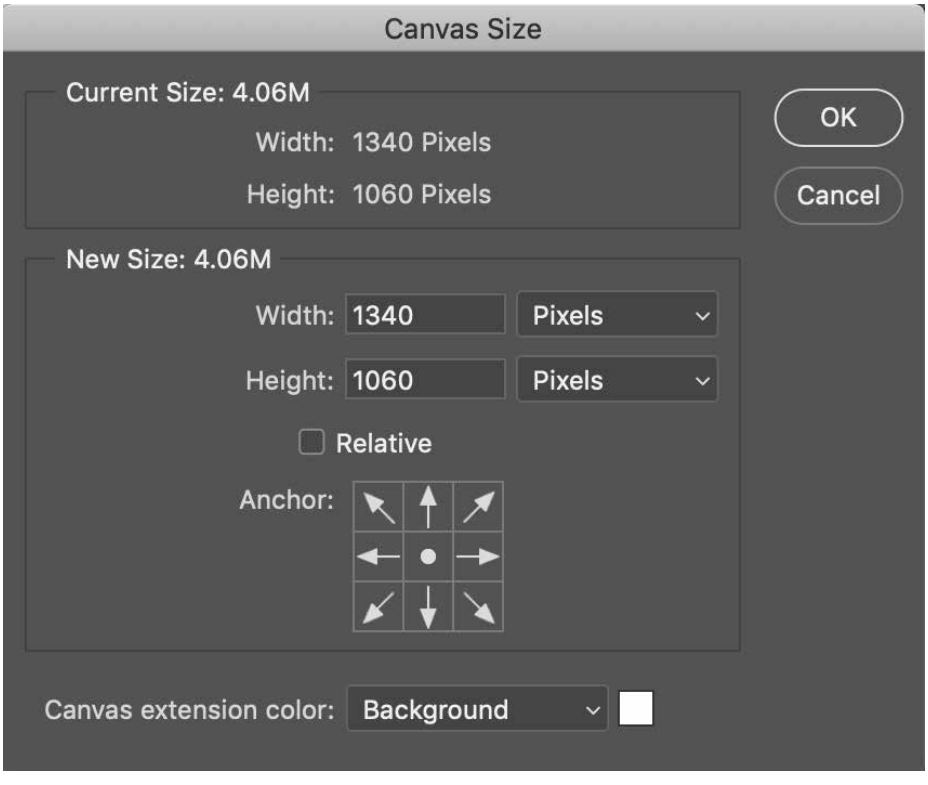


Figure 10-48. *Canvas Size dialog box in Photoshop CC*

Error Messages

What

When there is a form input error, such as a skipped required field, place an eye-catching explanatory error message directly on the form itself. The message might describe how the user can fix the error. If possible, put indicators next to the originating controls.

Use when

Users might enter form information that somehow isn't acceptable. They might skip required fields, enter numbers that cannot be parsed, or type invalid email addresses, for instance. You want to encourage them to try again. You want to point out typos before they become a problem, and help puzzled users understand what is asked for.

Why

The goal of displaying error messages is to help the user fix their issues and complete their task as quickly and painlessly as possible.

Two very traditional methods are seldom encountered and not recommended. One is to report error messages to users via modal dialog boxes. Those messages could be very helpful, pointing out what the problem was and how you could fix it. The problem is that you had to click away the modal dialog box to fix the error. And with the dialog box gone, you couldn't read the error message anymore. A second traditional approach is to show the form error messages on an error screen after you clicked the Submit button. Again, you can read the message, but you must click the Back button to fix the problem; after you do that, the error messages are gone. Then you need to remember what the error said and then scan the form to find the field with the error, and then fix the error. This is so much effort that it is not likely to be completed successfully or requires a great deal of back and forth.

Most web forms now place the error message on the form itself. By keeping both messages and controls together on the same page, you allow the user to read the message and make the form corrections easily, with no jumping around or error-prone memorization. Also, web forms put error messages physically next to the controls where the errors were made. Now the user can see at a glance where the problems were—no need to hunt down the offending field based just on its name—and the instructions for fixing it are right there, easily visible.

How

First, design the form to prevent certain kinds of errors. Use drop-down lists instead of open text fields if the choices are limited and not easy to type. For text fields, offer *Input Hints*, *Input Prompt*, *Forgiving Format*, *Autocompletion*, and *Good Defaults and Smart Prefills* to support text entry. Limit the total number of form fields as much as possible. Have a clear system for marking what fields are optional and what are required.

When errors do happen, and your form is long or complicated, you should show some kind of error message on top of the form. The top is the first thing people see. (It's also good for visually impaired users—the top of the form is read to them first, so they know immediately that the form has an error.) Put an attention-getting graphic there, and use text that's stronger than the body text: make it red and bold, for instance. In contemporary short forms, this step is often omitted.

The universal standard is to mark the form fields that caused the errors. You should display element-specific messages next to each affected control. This will require extra space beside, above, or below the fields—but at the least, use color and/or a small graphic to mark the field. Users commonly associate red with errors in this

context. Use it freely, but since so many people are colorblind with respect to red, use other cues, too: language, bold text (not huge), and graphics.

If you're designing for the web or some other client/server system, try to do as much validation as you can on the client side. This means checking for and displaying validation feedback to the user as they fill out the form, not after they have submitted it. The contemporary standard is to validate an input as soon as the focus moves to the next input field. Some forms have validation code that begins checking input as soon as the user selects the input field. Errors and feedback can appear once a user triggers it with their data entry. Either way is much quicker: The user has a chance to fix everything before submitting the form. (In some clumsy designs, an error message appears as soon as the user begins to type a valid entry, and it doesn't disappear until the user finishes a valid string or entry. This is inappropriate and annoying, so don't do this.)

A tutorial on error-message writing is beyond the scope of this pattern, but here are some quick guidelines:

- Make them short, but detailed enough to explain both which field it is and what went wrong: "You haven't given us your address" versus "Not enough information."
- Use ordinary language, not computerese: "Is that a letter in your zip code?" versus "Numeric validation error."
- Be polite: "Sorry, but something went wrong! Please click 'Go' again" versus "JavaScript Error 693" or "This form contains no data."

Examples

The registration forms for Mailchimp (Figure 10-49), Mint (Intuit) (Figure 10-50) and H&M (Figure 10-51) show a traditional approach. The user fills out the form and then selects submit. The Get Started! button for Mailchimp becomes active after successfully entering an email address and password, but omits checking for the required Username field. To discover this, the user has to submit and then try again. The specific problem inputs are highlighted for correction, with error messages. Mailchimp displays a generic error alert at the top of the form in addition to marking the error inputs. However, the error messaging is clear.

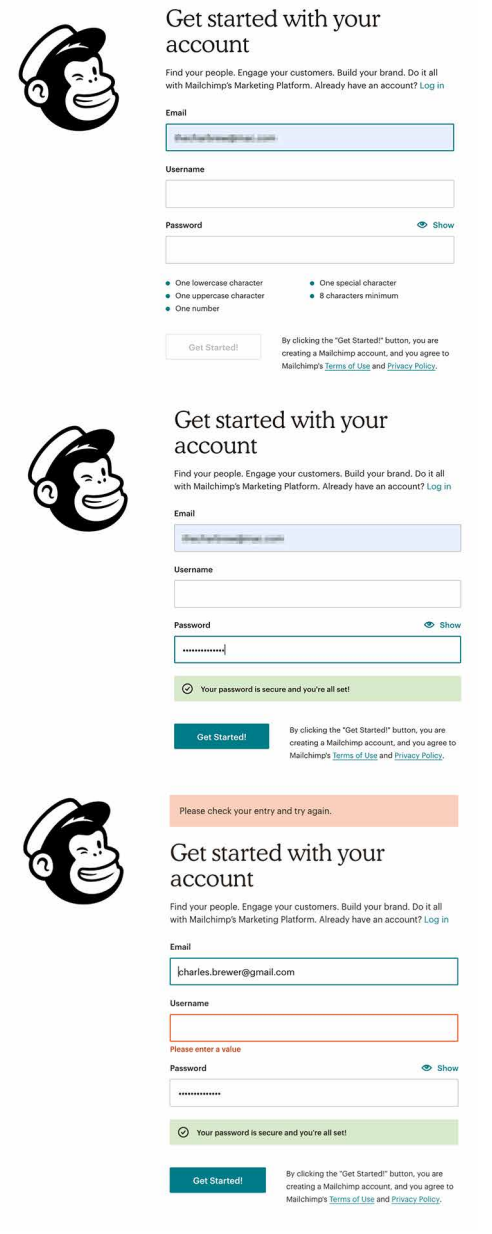


Figure 10-49. *Mailchimp registration page*

In the Mint example (Figure 10-50), the error message explains that the second password string does not match the first, and needs to be re-entered so it does match.

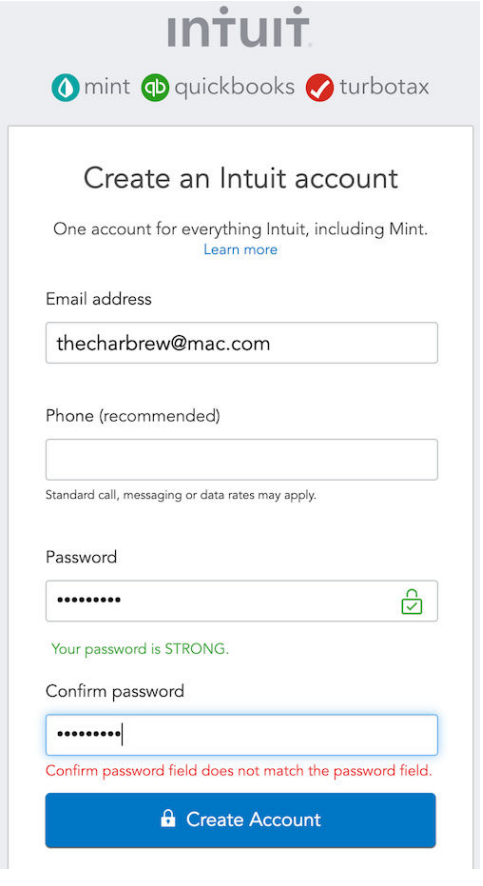


Figure 10-50. *Mint (Intuit) registration screen*

At H&M (Figure 10-51) the Become a Member button is always active. If the customer forgets a required field, they get an error message only after selecting Become a Member. The error message itself is good: It says exactly what needs to be done to fix the problem.

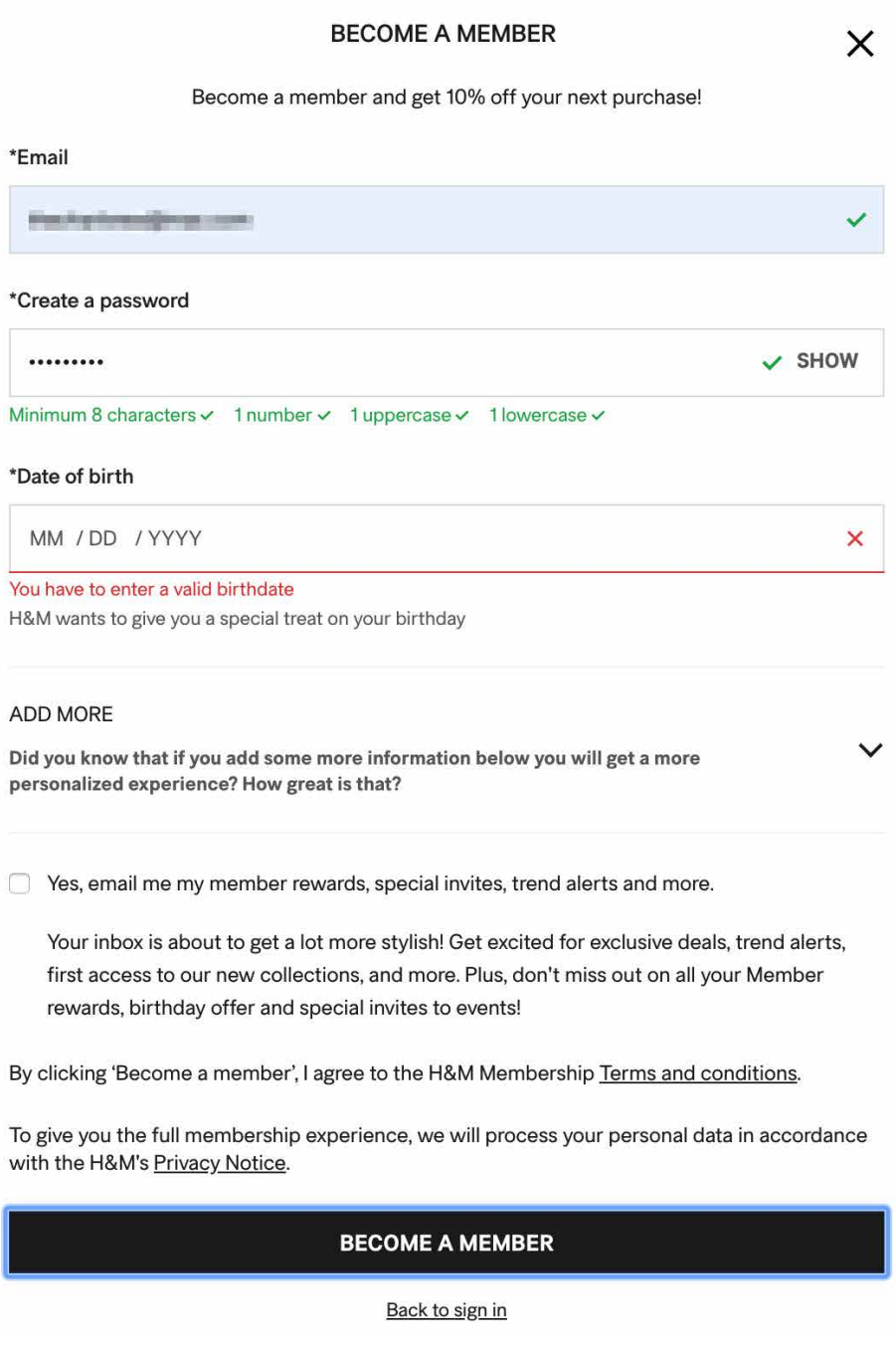


Figure 10-51. *H&M*

The standard solution today is to avoid creating a situation where the user skips a required step or has an invalid entry in the first place. The submit option simply doesn't become active until the required steps are filled in successfully.

Twitter breaks their sign-up process into multiple steps (Figure 10-52). In Step 1, the option to continue is not active by default. There is no option to go to Step 2 until this screen is successfully filled out. This drives the outcome of completing the form successfully with properly formatted data in order to get to the goal of the next step in registration. Bonus: The form is in a modal dialog box. There is no other navigation to distract, except to quit the whole process. Validation happens in real time, before the user selects the Next button. After the user fixes the required fields, the next step selector becomes active. Email address validation happens in real time. Actionable feedback appears without having to submit the form, saving the user a wasted action.

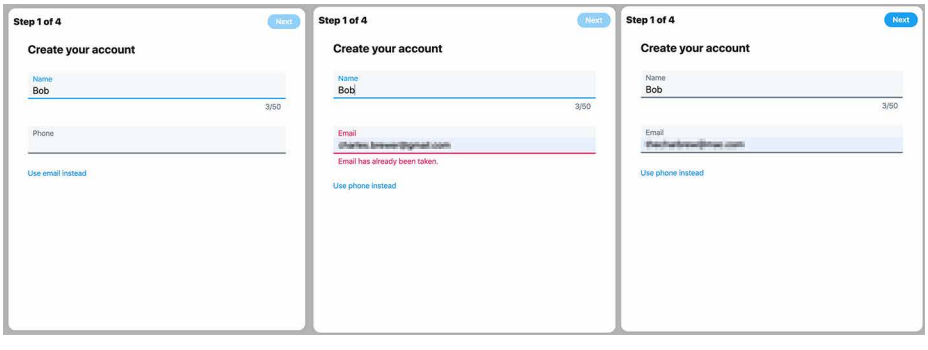


Figure 10-52. *Twitter registration*

When you add a not fully specified item to your cart at the Jos. A. Bank site, it uses a gentle message to remind you to fill in missing information, as shown in Figure 10-53. In this case, the Add to Bag button is always active. The shopper must select it to generate the validation error message.

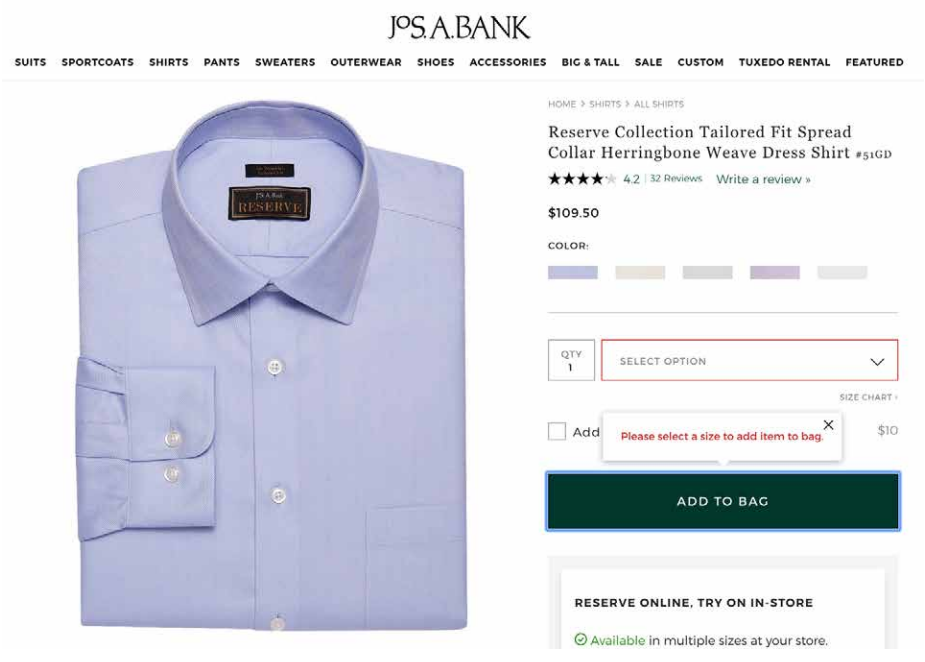


Figure 10-53. *Jos. A. Bank*

Conclusion

Form and control design is one area of interaction design in which the designer really can analyze all the possible use cases and think through exactly how they would like to handle the various interactions—standard, error, and edge cases. Form design can start out fairly simple but can become an exacting and challenging task. However, the design principles in this chapter give you a solid grounding in how to approach this activity and create useful, usable designs. The examples we reviewed can also be models for how you can manage important processes such as registration and passwords, validation and error messages, data formatting, and creating controls to handle complicated tools and settings.

Chapter 11
User Interface Systems and Atomic Design

The previous chapter gives many examples that point to a strong pattern in interface design: screen interfaces often are made up of a system of commonly used components. Recent developments in how designers and developers approach interface design build on this trend of design as a system. In this chapter, we'll look at component-based front end UI frameworks. Understanding and using this approach will help you deliver consistency, scalability, and better usability to your software. Thus, this chapter covers the following:

- An overview of component-based user interface (UI) systems
- A summary of the Atomic Design philosophy that takes a components-based approach
- A look at selected UI frameworks (component libraries) specifically for web and mobile web that you will likely be working with

Designing software applications today means creating a rich interactive experience for consumers and business customers, no matter what device they are using or where they are. The expectation is that the experience is always on, always communicating with the world around it, communicating with other people, and dynamically responding to its user from second to second.

Today, software is created from the ground up to connect to the internet in order to take advantage of powerful cloud-based information processing, storage, and communication. Mobile devices are the dominant consumer computing platform. Customers expect a given application to take full advantage of the capabilities of the local device, such as cameras, voice input, live location data, their own prior activities and preferences, and more. They also want to access the same software from multiple devices—mobile, tablet, desktop, watch, TV, and more—with a seamless transition and an experience that stays recognizably the same. They also do not expect to

compromise functionality and capability with smaller devices. Designers create UI systems to set UI standards in this environment.

The design and development approach that makes this possible is centered around components and widgets as well as connection to other systems in real time for communication, transaction, live data, and storage. The idea is to design and build a system of components that lets users accomplish their tasks without regard to the many different devices, screen sizes, operating systems, or web browsers.

UI Systems

UI systems, or UI design systems, are UI styles and standards systems that help a company's designers, developers, and partners maintain quality and consistency in the look and feel of their software products. They use a components-based approach. They focus on standardizing the functionality and look and feel as much as possible while still staying in line with different operating system (OS) standards. They do not specify implementation technologies, such as what programming language to use.

The main point for you as an interaction designer is that a components-based approach to interfaces and design is the standard approach now (at least for the nuts-and-bolts functionality like filling out forms, picking dates and times, and so on.) Let's look at this briefly.

Technology companies such as Microsoft, Apple, Google, and many others have UI systems that cover multiple operating systems and multiple devices and screens:

- Microsoft's Fluent Design System offers a standardized library of styles and code modules for Windows OS, web, iOS, and Android.
- Apple's User Interface Guidelines cover macOS and iOS apps, watchOS (for Apple Watch) and tvOS (for Apple TV).
- Google's Material Design System covers web, Android, iOS, and now native desktop OS applications through its Flutter UI framework.

An Example Component-Based UI System: Microsoft's Fluent

Let's look at one UI system and one component in it as an example. In 2017, Microsoft released Fluent, its UI system. Its goal is to help any product in Microsoft's software ecosystem look and feel like Microsoft, whether it is on the Windows desktop, on Android, on iOS or on the web.

A quick look at the website for the Fluent UI system shows its scope (Figure 11-1). Let's imagine that you are an interaction designer at Microsoft, and you are designing an app that needs a date and time picker. If you're designing for the web, you'll want to look at the Fluent web date picker (Figure 11-2). If you are a mobile designer, you will want to look at either the Fluent iOS date picker (Figure 11-3) or the Fluent Android date picker (Figure 11-4). Finally, if you are a Windows desktop designer, you'll want the Fluent Windows date picker (Figure 11-5).

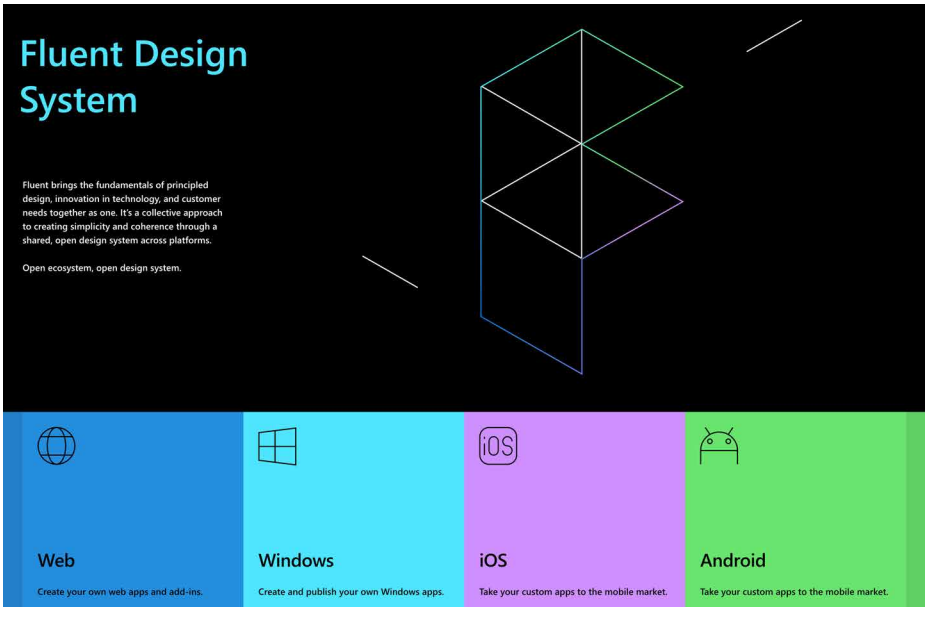


Figure 11-1. *The Microsoft Fluent Design System home page*

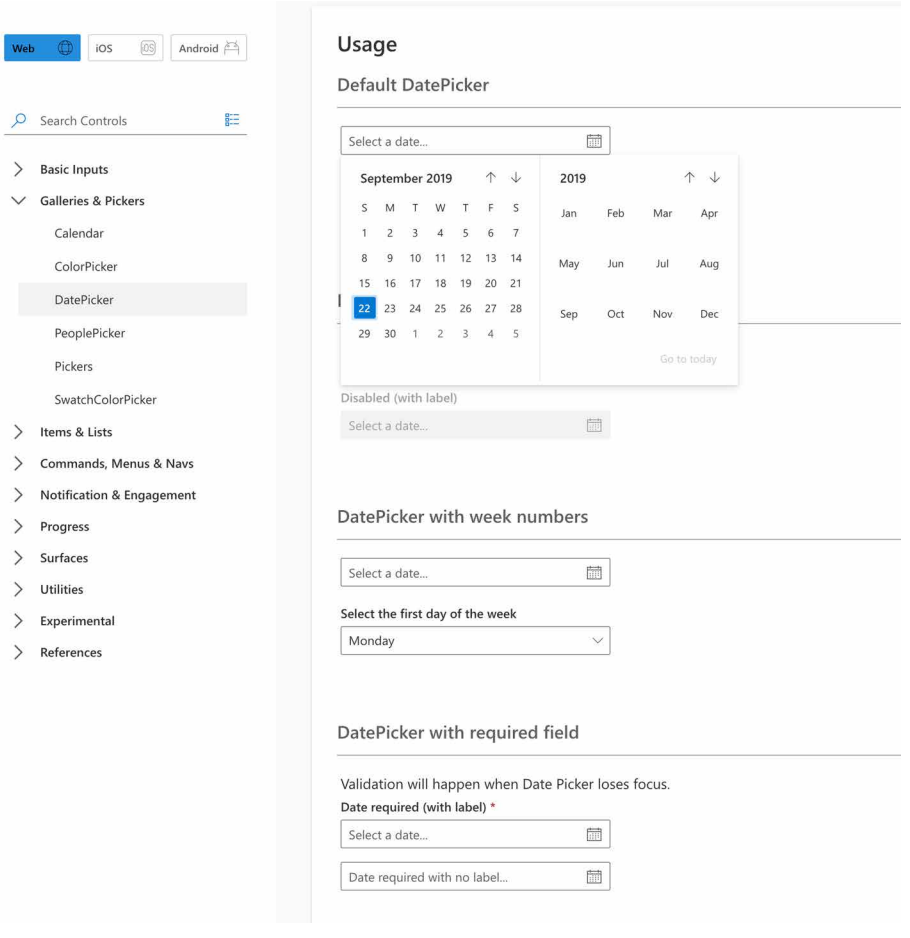


Figure 11-2. *The Microsoft Fluent web date picker*

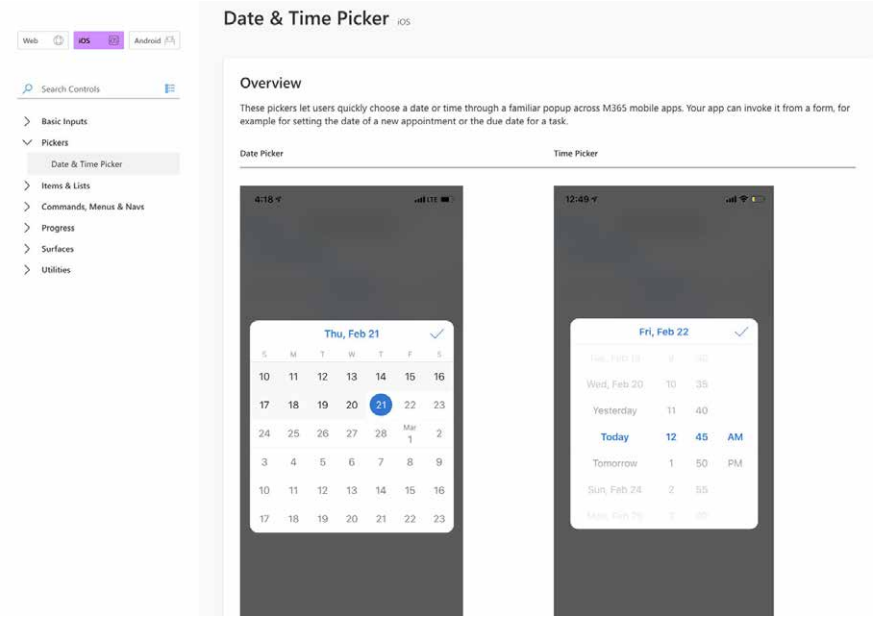


Figure 11-3. *The Microsoft Fluent iOS date picker*

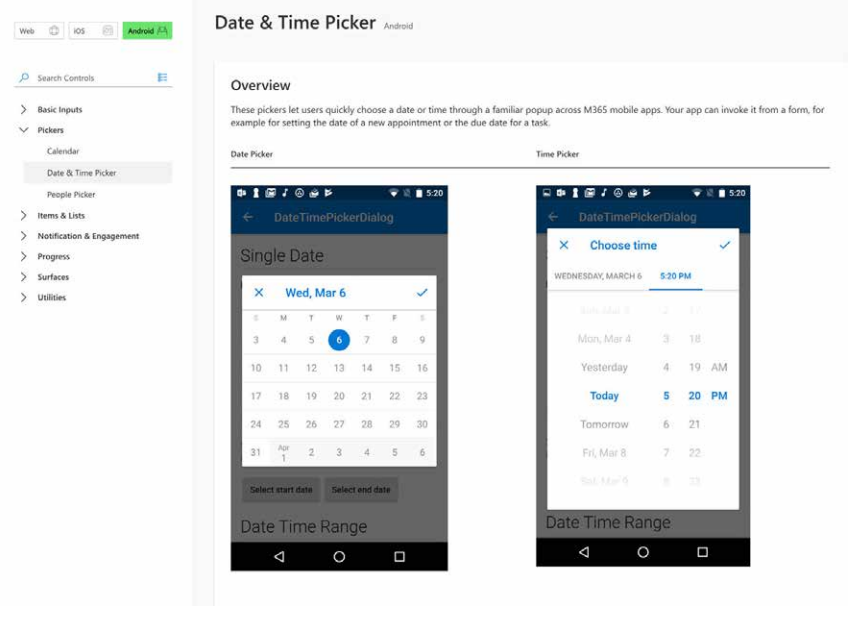


Figure 11-4. *The Microsoft Fluent Android date picker*

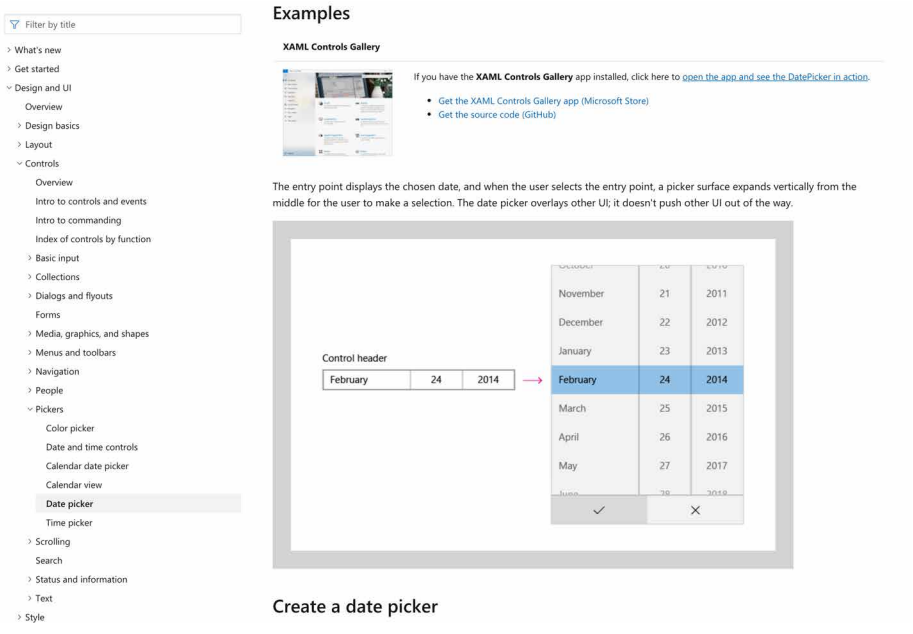


Figure 11-5. *The Microsoft Fluent Windows date picker*

That is a quick look at a components-based UI design system. How can interaction designers develop design systems that give them the same advantages of consistency, scalability, and reusability in their own work? One method has become popular: *Atomic Design*. We review this approach to design in the next section.

Atomic Design: A Way of Designing Systems

Design and design methodologies have been evolving directly alongside UI design systems, as described previously. UI design now includes the idea that we are designing a system of flexible, reusable components for assembling interfaces for almost any screen or device.

Many designers have begun thinking this way. *Atomic design* is an example of probably the most widely known name for this approach. It was originally developed by designer Brad Frost on his blog and now also his book, *Atomic Design* (Brad Frost, 2017). It is now widely adopted and adapted by design teams in solving design problems and creating their UI design systems. We present a brief summary of this approach here.

Overview

Atomic design consists of selected principles and a structure for categorizing UI components. We summarize them broadly in the following subsections.

Break it down

Atomic design is basically a bottom-up approach to building interfaces. A starting point is analyzing the software you are designing at the moment. Then, break down the pages and screens into their smallest pieces that still remain usable and functional on their own. Many of the aforementioned UI frameworks can also be a starting point because they are already broken down into components.

Style guide

The basic and universal guidelines for design are set up: color, typography, a layout grid (or a plan for contemporary, grid-less, resizing containers), icons, and more. See Chapter 4 for a discussion of layout grids, and Chapter 5 for color and typography. It's important to remember that this is still in service of delivering a unique and pleasing look and feel in the end. Following brand guidelines, or creating a branded look and feel, drives the choices here. Everything in your atomic design system will build from this and inherit these styles through CSS. Begin with styling the most basic, atomic units of your software. As you combine these elements, you'll need to add more style standards to control appearance, behavior, and layout, up to templates for entire screens of components.

Consistency

Avoiding "UX debt," the slow accumulation of diverging design styles and a different look and feel, is a major goal. The same component is reused everywhere, retaining its specified design. It's reused as an entire component. There's no need to create it from scratch, with the associated chance of changing how it looks. In this way, the software can grow and expand while maintaining consistency in its UI.

Modularity

The main idea is that you want a system of software building blocks that can be assembled into tools and screens to meet a specific need. Little pieces join together to make bigger pieces. The metaphor here is creating a LEGO toy set, except for user interfaces. You can use small pieces on their own, or you can put them together in combination to create much bigger, more robust components and tools.

Nesting

As mentioned a moment ago, the strategy is to start with the smallest possible functional elements and the most universal style standards, such as typography and color

palettes. You can use the smaller pieces to create larger ones. For example, a simple text input field can be combined with a label, an *Input Hint* or *Input Prompt* (see Chapter 10), and a drop-down selector within the input field to make a more usable, dynamic form component. There is also the idea of inheritance of styles. The styling of the smaller components is carried with them into the larger aggregation. Changing the style declaration at the base level will then change it everywhere in the system. For example, a color from the color palette could be adjusted for better contrast and accessibility, and this propagates everywhere that this color is used. You could adjust the corner radius for buttons, and anything using the button element picks up this shape change right away. For the design team, this can be accomplished by using a common library file or shared file that all the designers reference. When linked properly, design files can automatically update when the source file updates. In live products, the engineering team must integrate any design updates into the source code that renders the associated component. Once this is done, the product will reflect the changes.

Build it up

It almost goes without saying that the point of creating a set of LEGO-like building blocks is so that we can build entire screens, workflows, and applications. Designers can style existing web app components such as buttons and input fields and then build up from there. They can also start with the modules and components already available in UI frameworks, like those we discussed earlier. There are many to choose from, with different levels of customizability. Some require more component building; others already have a big library of UI components. Pick the one that best suits your needs and works with the JavaScript framework that your developers are using.

Not tied to any technology

Despite the tech-centric discussion in this chapter, Frost and many other designers take pains to say that designers should stay focused on design and not on the limits of any particular technology or framework. The goal is the same as it always was: understanding users; finding opportunities to make their lives easier; and designing delightful, usable software experiences for them. Atomic design, UI frameworks, and UI patterns are merely tools to achieve this, not any kind of restraint or blocker.

The Atomic Design Hierarchy

A UI design system that is built up with atomic design has a structure based on going from small and simple to large and complex.

Atoms

Atoms are defined as the smallest, most basic building block of the UI system. They are modules that can't be broken down any further without losing their ability to

function as a workable component. Examples would include a text input field, a label, a color, a typeface.

Molecules

Molecules are groupings of two or more atomic components to make a more complete functional element. Examples of molecule elements might be an image with title and caption to form a news or promotional molecule within a Grid of Equals (Chapter 4), or a form input with a label, *Input Hint*, *Input Prompt*, and a submit button.

Organisms

Organisms represent collections of molecules into fairly complex objects that handle multiple functions or a major function of your software. A good example of this would be a header module in your app or website. It might consist of a company logo, the entire global navigation component (such as *Fat Menus*, in Chapter 3), a search module, utility navigation or *Sign-In Tools*, user avatar, and a notifications counter.

Templates

Templates are the scaffolding for putting molecules and organisms together for specific purposes. They are the layout recipes for creating the types of screens that you will need based on the content you have. Often these templates represent a screen type that occurs over and over. Examples of this would include a form, a home page, a report with a chart, or a screen for tabular data.

Pages

Pages represent the end product: templates now filled out with real content. The underlying template and module system can be standardized, but each page is different based on the unique content it serves up. And you now have a consistent *Visual Framework* (Chapter 4) enforced throughout the site.

How can an interaction designer bring an atomic design–inspired system to life? Fortunately, there are many libraries of interface components that are ready to be styled, modified and put to use quickly. These are called UI frameworks. It's likely your development team has selected one already. In the next section, we take a look at some selected UI frameworks.

UI Frameworks

The UI frameworks we discuss in this section are tied to web and mobile web technologies: HTML, CS , and JavaScript (or similar programming language, such as CoffeeScript or TypeScript). That is because this third edition of *Designing Interfaces* focuses specifically on screen-based interfaces for web and mobile.

UI frameworks have a number of different names: frontend frameworks, CSS frameworks, UI kits, UI toolkits. But they all consist of the same thing: A system of software components for building a screen-based UI. It's worth looking at these more closely for two reasons:

- UI frameworks are great examples of component systems in action
- It's likely you will be designing software that uses one of these frameworks

Overview

Software developers today for web and mobile have developed JavaScript (or other similar language) frameworks for rendering their frontend code. These frameworks are simply libraries of prepared, configurable code modules which can be customized more easily than writing code from scratch. They are used to generate the appropriate HTML, CSS, and JavaScript for web and mobile web applications. This makes for rapid software development and more consistent code and UI. Some examples of the most popular JavaScript frameworks are Angular, React, Vue, Ember, and Node.js, and there are many others. These code frameworks handle user inputs, make API calls, and process data returned by these APIs.

Benefits

Code frameworks are the middleware engine that is churning away to make a lot of underlying complexity go away. This offers a number of benefits:

Speed
: Prepared code to handle the majority of expected use cases means that you can generate working software faster.

Consistency
: Code modules and common CSS render UI components the same way.

Masking different browser capabilities
: In the not so distant past, browsers had varying levels of support for HTML, CSS, and JavaScript standards. They also implemented browser-specific quirks. This was an attempt to differentiate and "win the browser wars." This created a huge problem getting early web and mobile software to operate and look the same on different browsers. Modern JavaScript frameworks now automatically take care of browser and OS differences.

Responsive design automatically included
: There's no doubting that we have seen an explosion of different device types and screen sizes. Responsive design is an answer to this challenge. Liquid layouts that automatically change based on the user's screen size helped designers create

designs that worked across devices. Modern code frameworks include this important capability.

The Rise of UI Frameworks

These code frameworks also provide the component libraries for the front end or presentation layer of your application. This is why they are called UI frameworks, UI kits, UI toolkits, CSS frameworks, or frontend frameworks. They contain modules and components with which interaction designers are familiar: headings, buttons, form inputs, images, and so on.

Designers can customize the appearance of these UI frameworks in many ways. Indeed, the main point of this chapter is that this is often the bulk of the design job on a software project today. This is the context for UI design today:

- Component-based UI frameworks are part of a systems approach to UI design.
- Design the system, not one-off screens.
- UI frameworks are a starting point, and can be adapted and styled.
- UI frameworks are a floor, not a ceiling. They save time and energy that you can use to solve other, more complex problems.

In the following sections, we look at some common UI toolkits and compare and contrast them by looking at just a few examples of components from each.

A Look at Selected UI Frameworks

In this section, we take a quick tour of a selection of the UI frameworks that are out there today. There are dozens, with more appearing every day. An exhaustive comparison and contrast is beyond our scope here. One key point is that they all use CSS so that color, typography, and other design styles are open to customization. Another key point is that these toolkits consist of modules and component . The goal is to show that UI frameworks will often be your starting point for your atomic design–inspired design process. Here are the selected UI frameworks that we look at:

- Bootstrap
- Foundation
- Semantic UI
- Materialize
- Blueprint
- UIkit

Bootstrap

Bootstrap is one of the most popular UI frameworks today. Originally developed by Twitter, it's available for anyone to use. Figure 11-6 shows the list of components that are included by default. Figure 11-7 shows examples of common button components.

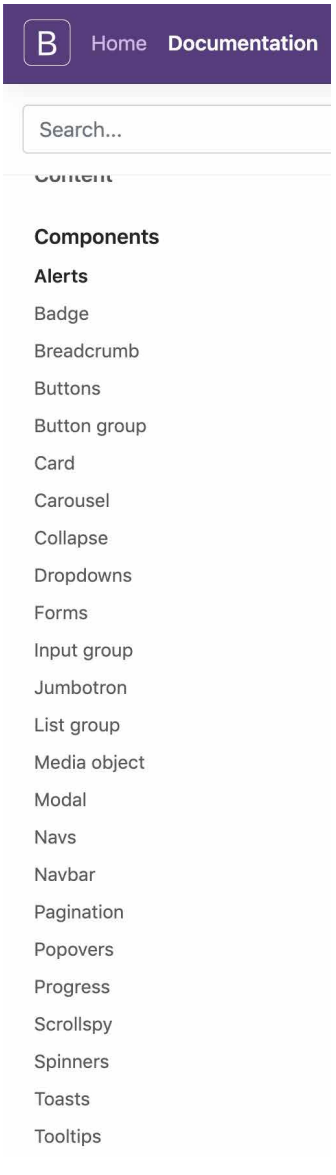


Figure 11-6. *Bootstrap components*

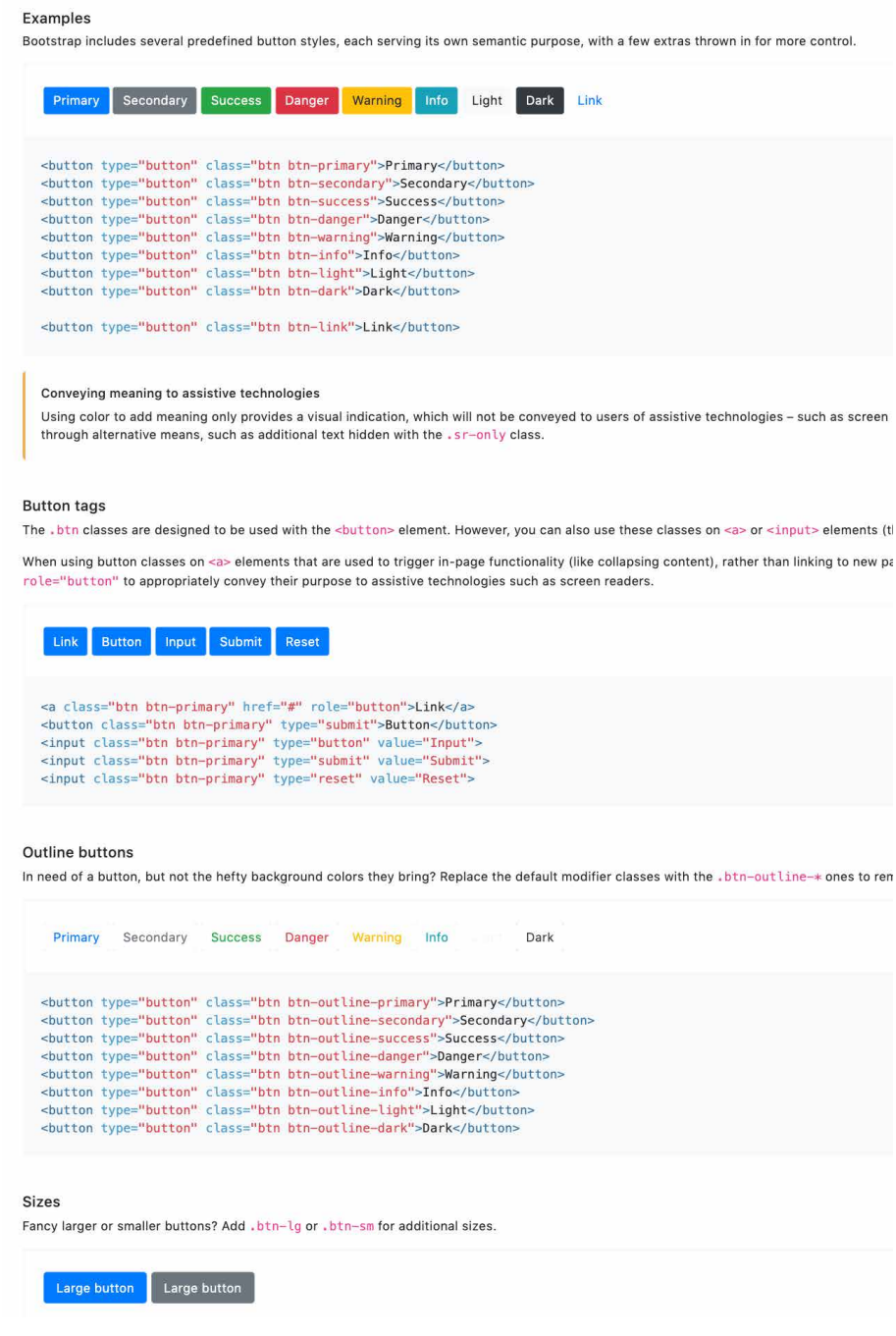


Figure 11-7. *Bootstrap button components*

Foundation

Foundation is another popular UI framework today. Originally developed by a company called Zurb, it is robust and has a large contributor community. Many large enterprises use Foundation. Figure 11-8 shows the list of components that are included by default. Figure 11-9 shows examples of button components.

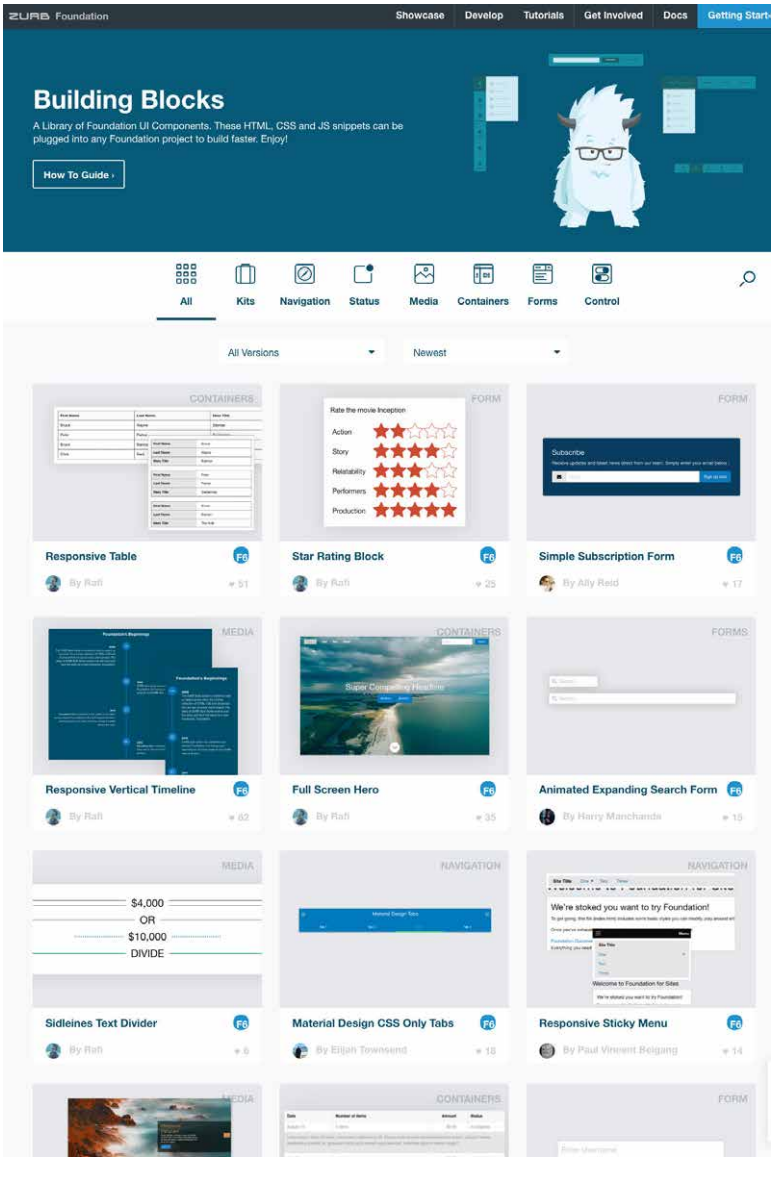


Figure 11-8. *Foundation components*

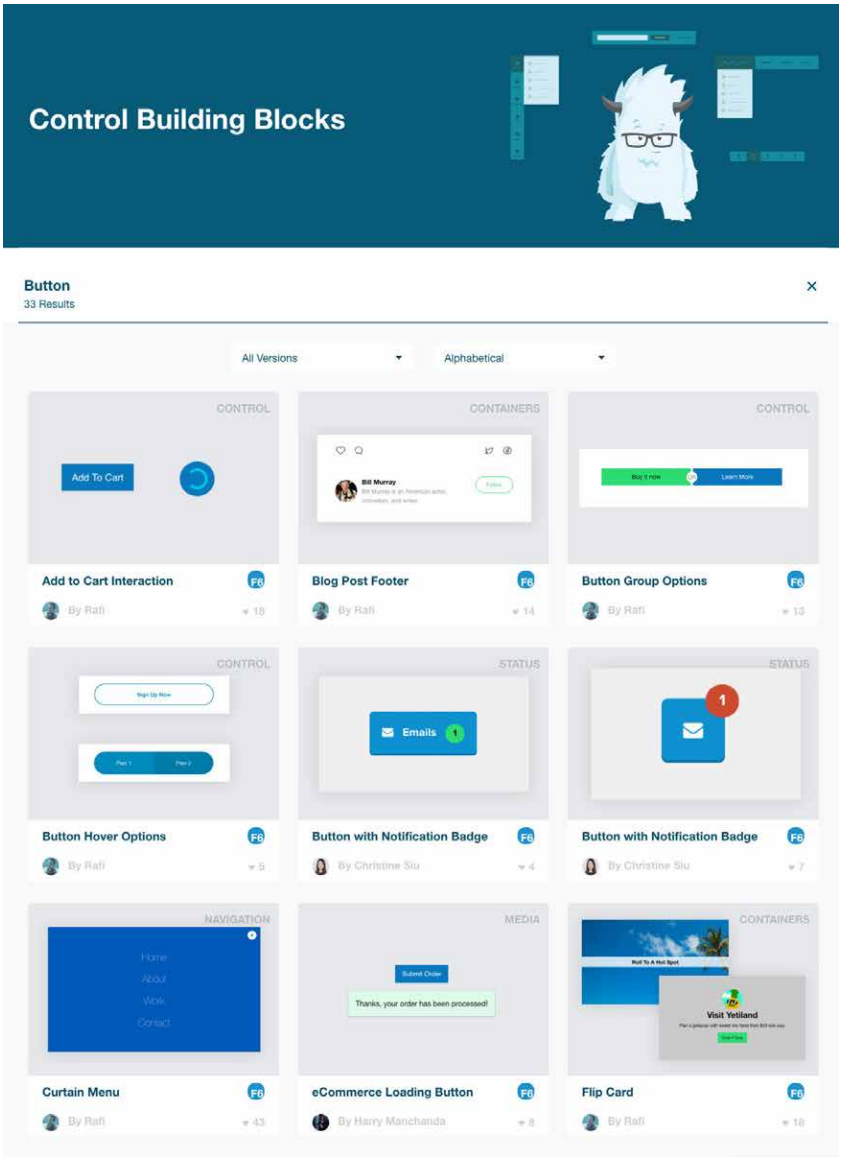


Figure 11-9. *Foundation button components*

Semantic UI

Semantic is a UI framework that aims to be friendly to the ordinary person. Its organization and naming use natural language and real-world concepts. Figure 11-10 shows the list of components that are included by default, and Figure 11-11 shows its button components.

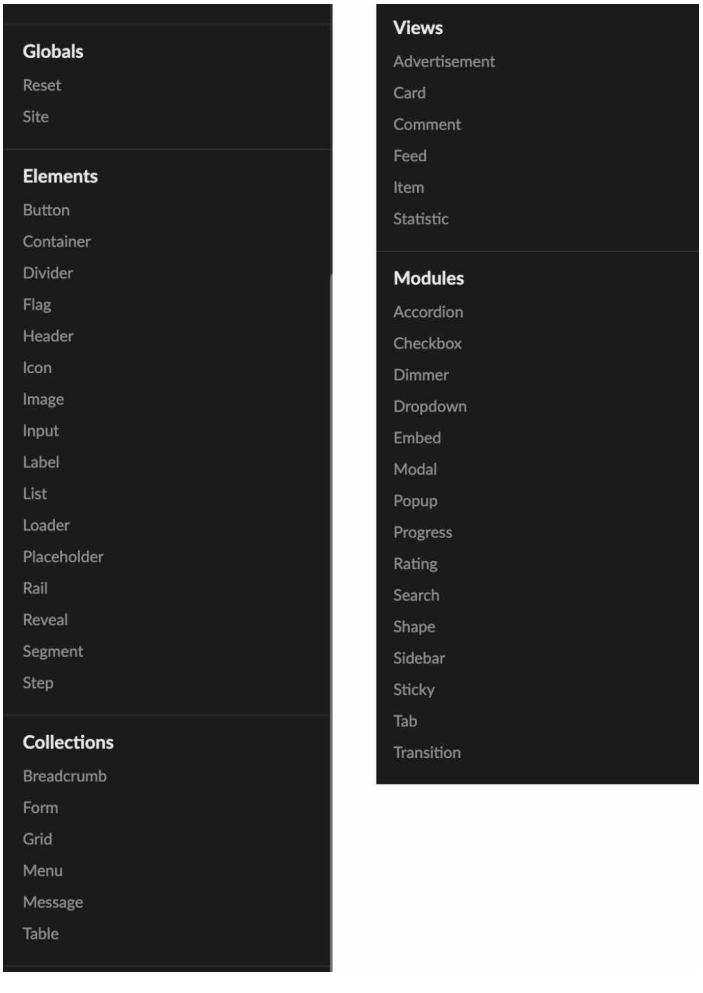


Figure 11-10. *Semantic UI components*

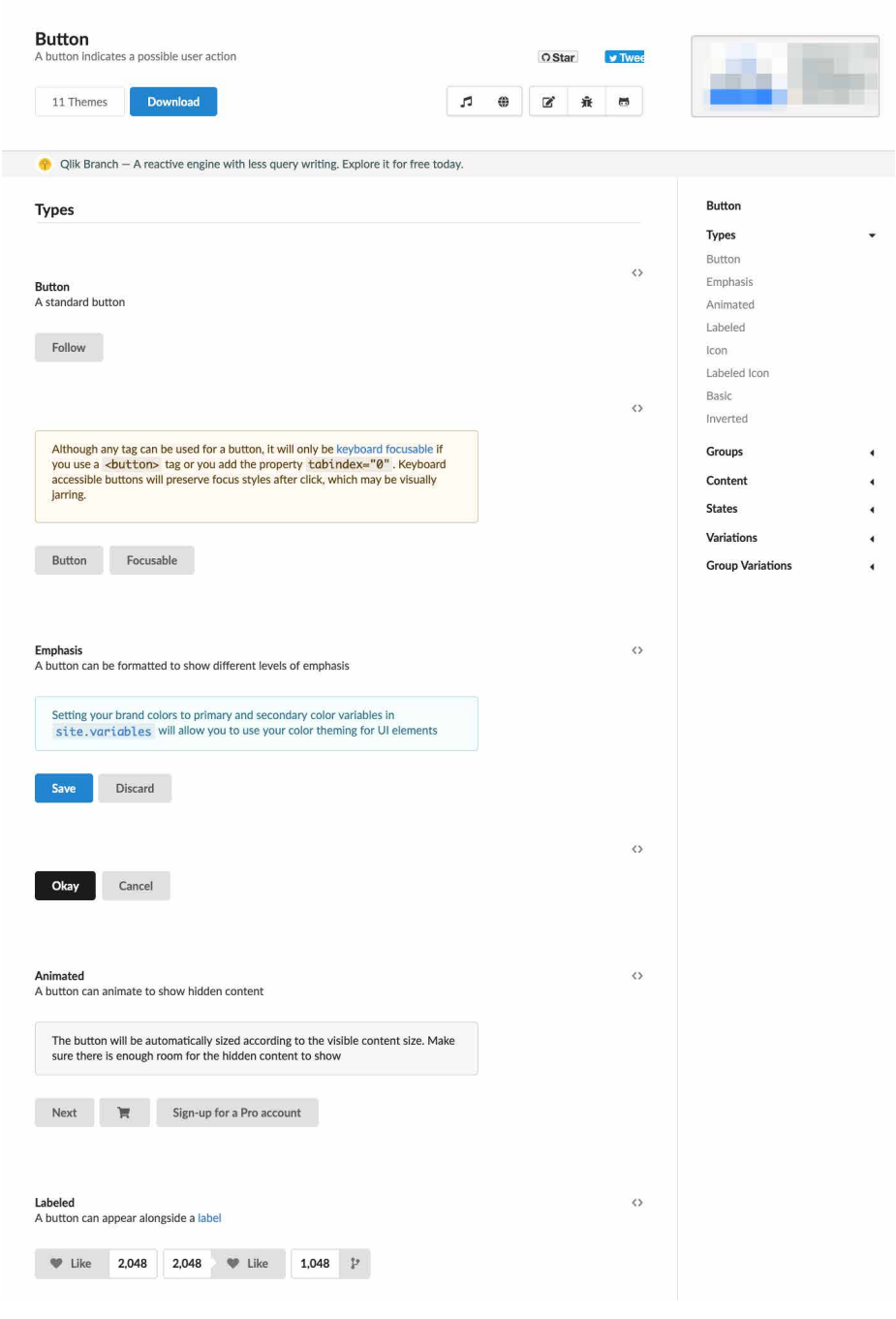


Figure 11-11. *Semantic UI button components*

Materialize

Materialize is one of many UI frameworks that is based on Material, the design system for Android. Originally developed by Google, Material is widely used for Android-native apps but also for web and mobile apps by third parties. Materialize helps them do this and stay in line with Material style. Figure 11-12 shows the list of components that are included by default. Figure 11-13 shows button component examples.

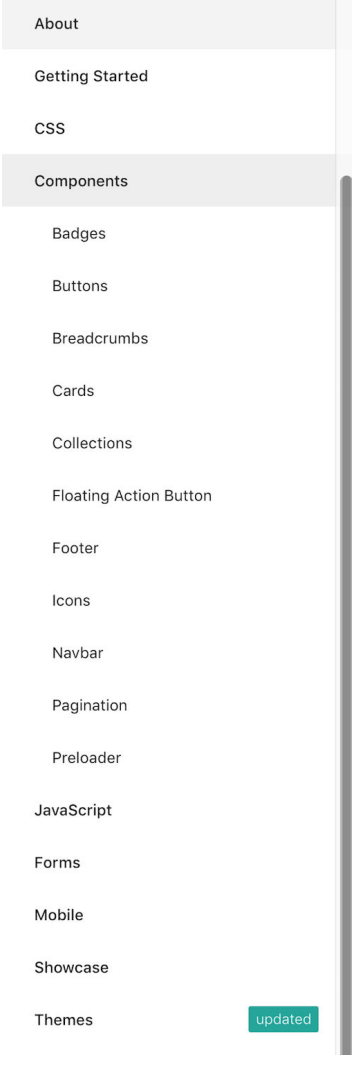


Figure 11-12. *Materialize components*

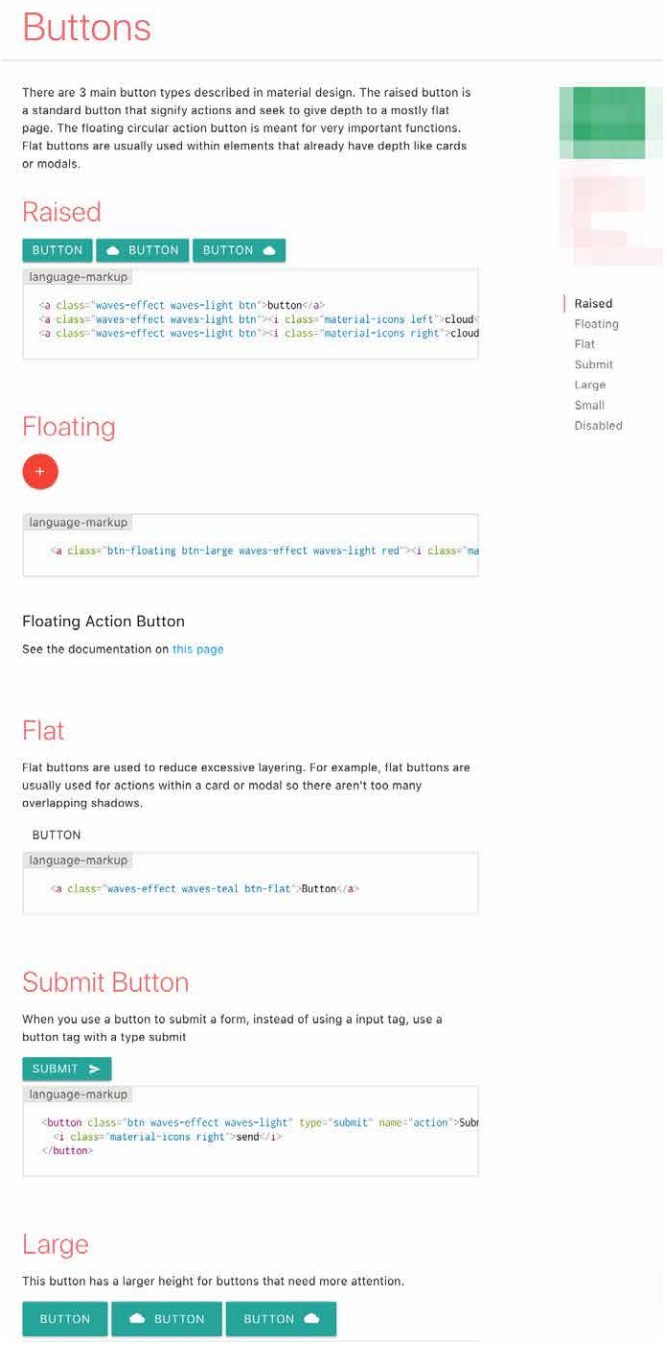


Figure 11-13. *Materialize button components*

Blueprint

Blueprint is a UI framework that is optimized for data-intensive web applications. Developed by Palantir Technologies, it's available to any software project. Figure 11-14 shows the list of components that are included by default; Figure 11-15 presents examples of buttons.

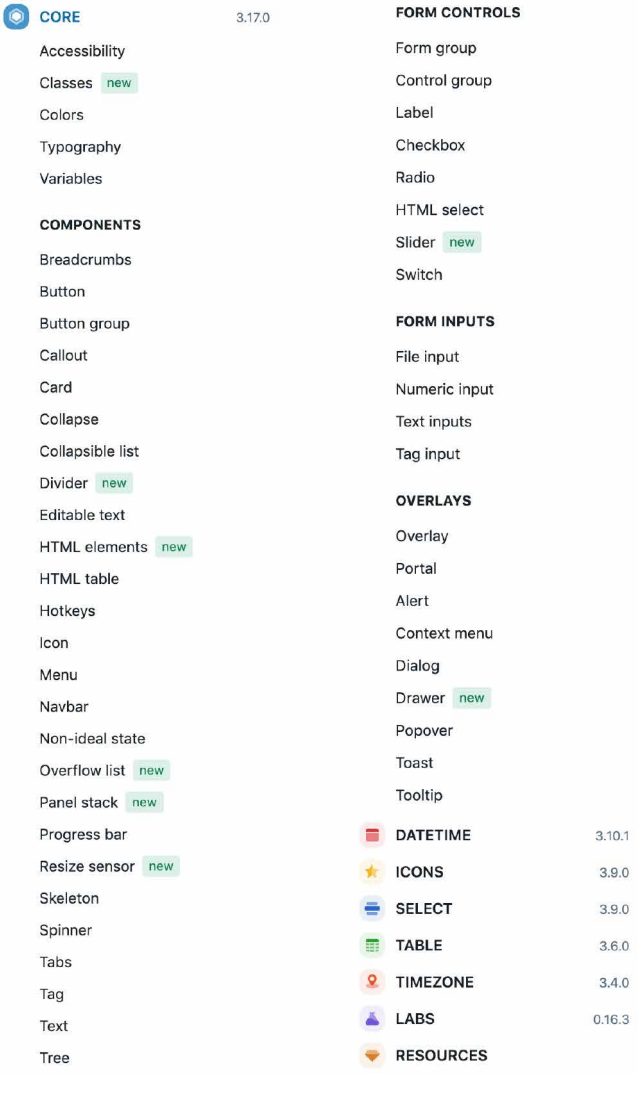


Figure 11-14. *Blueprint components*

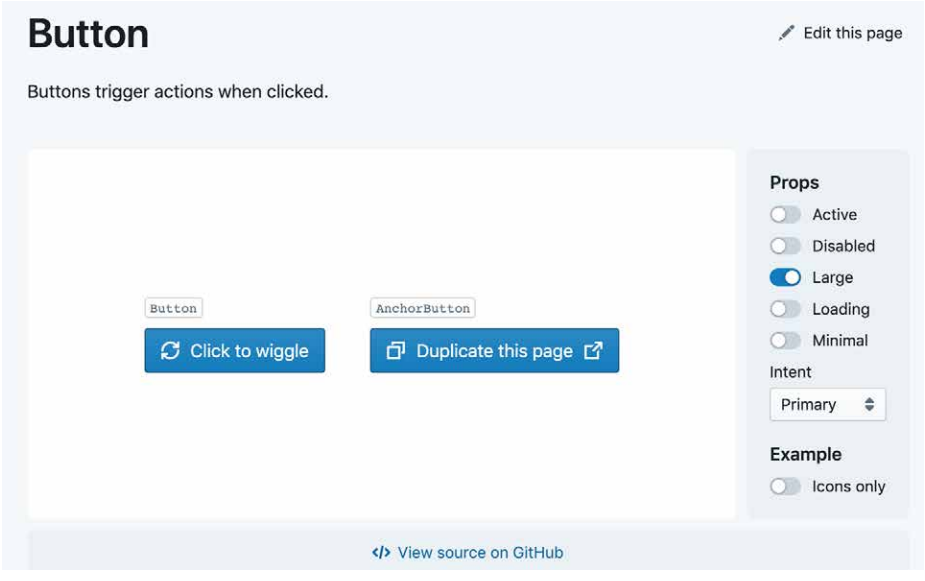


Figure 11-15. *Blueprint buttons component*

UIkit

UIkit is a minimalist UI framework intended for a quick start with a frontend component design. Figure 11-16 shows UIkit's list of components that are included by default, and Figure 11-17 shows button component examples.

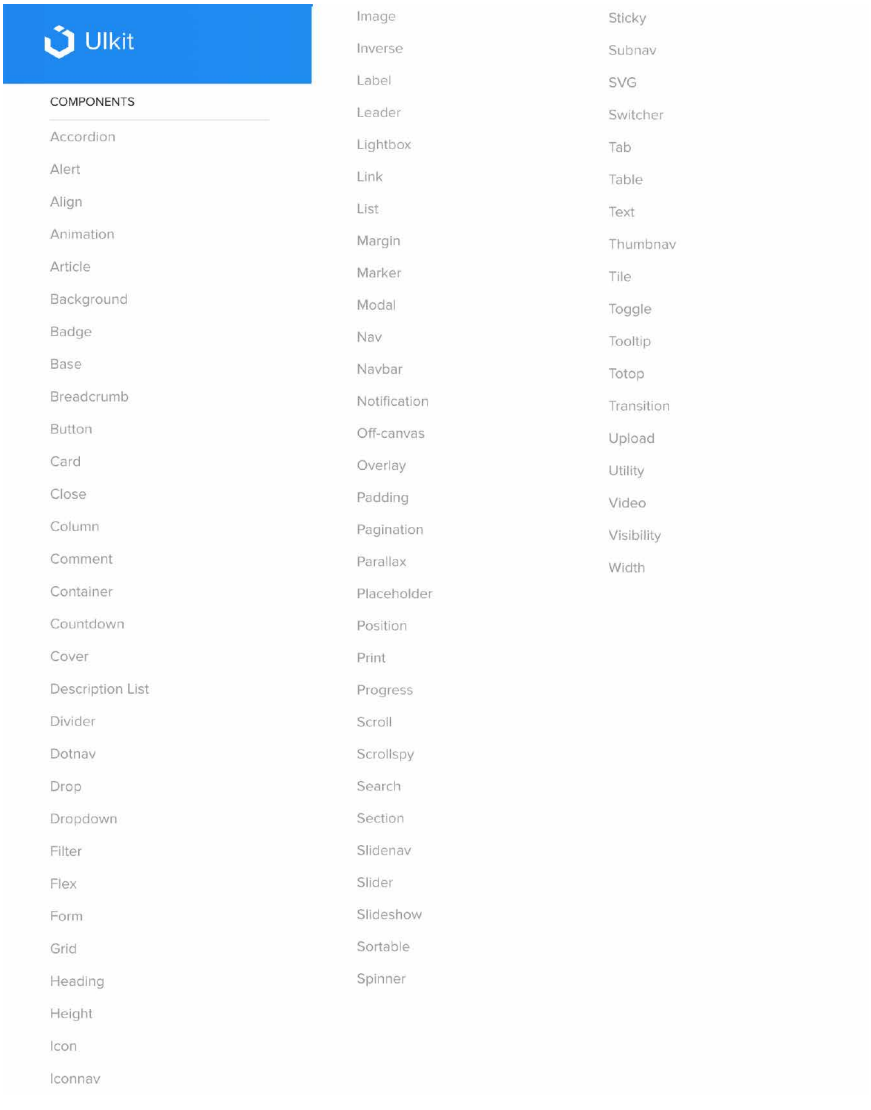


Figure 11-16. *UIKit components*

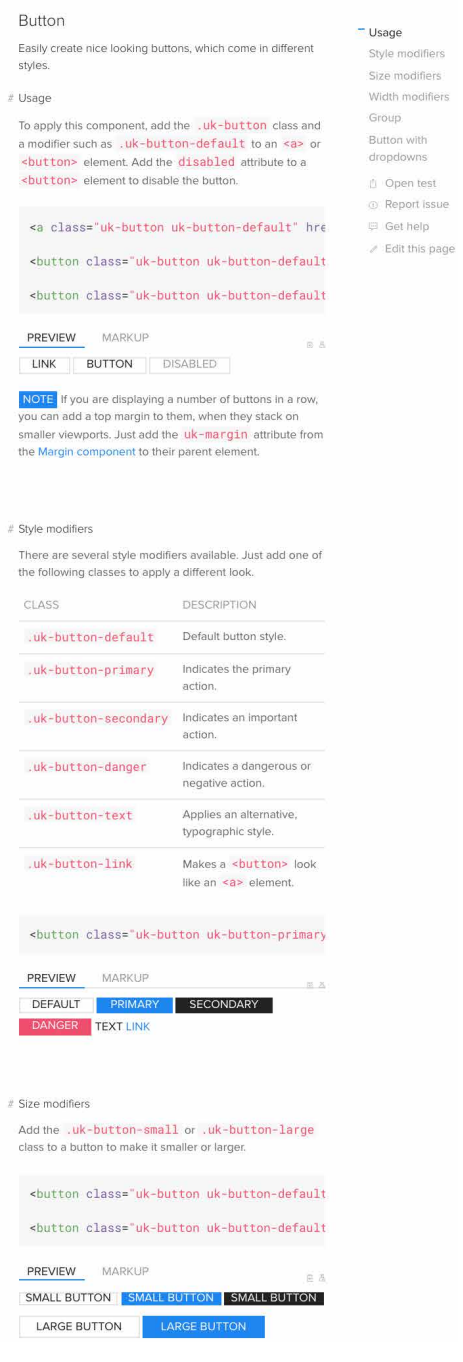


Figure 11-17. *UIkit buttons component*

Conclusion

A strong basis for designing scalable, consistent interfaces today is using a design systems approach, such as atomic design, paired with a modern UI framework like the those previewed in this chapter. UI frameworks are a standard part of the engineering foundation for software projects, so you can expect your engineering team to select one. These frameworks use a patterns-based approach to the UI. The benefits are quicker and easier design and deployment of the standardized, repeated parts of your interaction design. For your customers, there is the benefit of an easier learning curve and more confidence in using your designs. There are also the benefits of scaling the design team and your software without introducing random variations or different ways of doing the same thing. This allows you to free up time to solve the really difficult interaction design challenges.

Chapter 12
Beyond and Behind the Screen

In the preceding chapters, we explored the world of designing screen-based interfaces for the web, software, and mobile devices. These patterns and best practices cover the world of people-facing digital product design, characterized by screen-based experiences that people can click and tap. Behind the scenes, however, changes are happening in the complex systems that power these interfaces and experiences, and more and more these changes are manifesting themselves in the way users interact with the systems and the systems interact (or don't need to interact) with the user.

The majority of systems that are visible to the user involve the user contributing information or transactions to a system, and an interface that shows the results to the user. The patterns these systems use vary according to the intended use of that system.

Social media–oriented experiences such as YouTube, Facebook, or Twitter follow similar patterns, because at their core, the things they need to do are very similar. At a very basic level, the interactions involve a user posting content to a system that other people can view and comment on. The owner of the content can edit the content, delete it, or modify who can view it. Other users can like or share the content with others, or even tell the system if the content is offensive or if they choose not to see it. In this way, although the system mediates and shows content based on a complex set of algorithms, from the user's perspective, the interactions are fairly straightforward.

Likewise, news sites like the New York Times, The Atlantic, or your local online newspaper also follow a similar pattern from a systems perspective. Behind the scenes there is a content management system that reporters, editors, and photographers use to put content into the system and review it before it goes out to the public. Users can view articles, share them, and in some systems, post and edit comments.

Ecommerce sites can also be broken down into a simple pattern. Behind the scenes, there is a system in which an employee inputs a photo and description and all the options available for a particular item; there is a system that keeps track of how many of these items the company has available and where it can be located. A user can view items by categories or collections, and eventually navigate to a product detail screen where they can select a particular size and quantity of items, add them to a cart, and make the purchase.

These are the patterns we see all over the web and mobile apps. But as technology is capable of handling more complex operations, the complexity of input and output likewise increases. Behind the scenes, algorithms (sets of rules of calculation of data) power the information and content that is displayed to users. These algorithms become more sophisticated as they evolve through *machine learning*: a subset of artificial intelligence in which systems look for patterns to infer identification and classification of data.

Ubiquitous computing (also referred to as the Internet of Things or the industrial internet) refers to the capability of objects or spaces to be embedded with internet-connected hardware, such as sensors, that can read information about the environment and communicate this information back. These systems can be profoundly complex, and most of the time are completely invisible to people.

These complex systems "interface" with users differently from the screen-based interactions we have focused on in this book. As interactions become more automated and invisible, the user won't even need a keyboard--the interactions will become simpler, as a user confirms and approves of actions a system takes on the user's behalf. These types of interactions will be more and more pervasive in the future.

The Ingredients: Smart Systems

The world of technology is undergoing a massive change in its infrastructure. Systems are moving from the user actively inputting data to systems that "read" activity and location to establish meaning of these indirect inputs.

Connected Devices

Connected devices are those that are connected to the internet. Your smartphone, TV, car, thermostat, lightbulbs, and even the dog's food bowl can be connected and monitored via the internet.

Anticipatory Systems

Anticipatory systems are systems that quietly observe what the user is doing and serve up data, suggestions or even proactively place orders for the user. An example of this might be a connected refrigerator that knows what food is in the fridge and reorders the milk when the supply is low.

Assistive Systems

Assistive systems allow for the user to have their human capabilities augmented and enhanced through technology.

Natural User Interfaces

Natural user interface refers to "interfaces" that involve using motions, gestures, touch, or other tactile and sensory ways to provide input and output. The touch screen you use on your smart phone or tablet is an early example of a natural user interface. There are more examples, such as Amazon's Alexa, or Microsoft's Kinect system that reads gestures, and there will be many more of these types of interfaces that you can tap, squeeze, hold, wave at, or talk to.

Conclusion

The third edition of Designing Interfaces covers a vast amount of territory of design patterns and practices for designing for screen-based design, from desktop software to websites to mobile. As these systems become more complex, we hope the experience for the people who use these systems will become simpler and easier to understand. The role of a designer is to understand these patterns and apply them to their particular context. In this way, storytelling and narrative and imagining the scenarios and the "rules" that will apply to those scenarios will be one of the primary responsibilities of design.

Regardless of how interfaces find their way in front of a user, the patterns and principles of this book are evergreen. Understanding the foundational patterns of a sound UI architecture, creating a visual presentation with a clear information hierarchy that follows the Gestalt principles, supporting the user with appropriate help—all of these will bring clarity and understanding to an interface's users.

We leave you with this thought. As designers and people who work in technology create the experiences that touch people's lives in ways great and small, we are presented with the opportunity to move into a future with an ethical, human-centered mindset that makes life better for people and our planet. We have this amazing palette of technology and tools to build the future. The question we must now ask ourselves is this: what type of future will we create with it?

Index

A

B

C

D

E

F

G

H

I

J

K

N

0

P

Q

R

S

T

U

V

W

Y

Z

About the Authors

Jenifer Tidwell has been designing and building user interfaces for industry for more than a decade. She has been researching user interface patterns since 1997, and designing and building complex applications and web interfaces since 1991. She recently pivoted from digital design to landscape design, where she still balances usability, beauty, and good engineering every day.

Charlie Brewer is a user experience design leader with deep experience in B2B web applications and SaaS platforms. He works in organizations to build the design capability to turn insights into digital products. His background includes independent film, teaching, and digital brand-building for global clients. Charlie went on to design a groundbreaking programmatic TV ad marketplace, cofound a social gaming startup, and launch multiple digital products.

Aynne Valencia is Design Director for The City of San Francisco Digital Services and Associate Professor at California College of the Arts. Her experience includes building creative teams, launching major products and services, mentoring and educating designers and collaborating with major global brands. She has an extensive design background in both physical product design, digital product design, service design and software.

Colophon

The animal on the cover of *Designing Interfaces* is a mandarin duck (*Aix galericulata*), one of the most beautiful of the duck species. Originating in China, these colorful birds can be found in southeast Russia, northern China, Japan, southern England, and Siberia. The males have diverse and colorful plumage, characterized by an iridescent crown, chestnut-colored cheeks, and a white eye stripe that extends from their red bills to the back of their heads. Females are less flamboyant in appearance and tend to be gray, white, brown, and greenish-brown, with a white throat and foreneck.

These birds live in woodland areas near streams and lakes. Being omnivorous, they tend to have a seasonal diet, eating acorns and grains in autumn; insects, land snails, and aquatic plants in spring; and dew worms, grasshoppers, frogs, fish, and mollusks during the summer months.

The mating ritual of mandarin ducks begins with an elaborate and complex courtship dance that involves shaking movements, mimed drinking gestures, and preening. Males fight each other to win a female, but it is ultimately the female who chooses her mate. Mandarin ducklings instinctively follow their notoriously protective mothers, who will feign injury to distract predators such as otters, raccoons, dogs, mink, polecats, eagle owls, and grass snakes.

Mandarin ducks are not an endangered species, but they are considered to be threatened. Loggers continuously encroach upon their habitats, and hunters and poachers prize the males for their plumage. Their meat is considered unpalatable by humans, and so are generally not hunted for food.

Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery, based on a black and white engraving from *Wood's Illustrated Natural History*. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.