# Design Strategies

Design Principles

Aesthetic Integrity

Aesthetic integrity doesn’t measure the beauty of an app’s artwork or characterize its style; rather, it represents how well an app’s appearance and behavior integrates with its function to send a coherent message.



People care about whether an app delivers the functionality it promises, but they’re also affected by the app’s appearance and behavior in strong—sometimes subliminal—ways. For example, an app that helps people perform a serious task can put the focus on the task by keeping decorative elements subtle and unobtrusive and by using standard controls and predictable behaviors. This app sends a clear, unified message about its purpose and its identity that helps people trust it. But if the app sends mixed signals by presenting the task in a UI that’s intrusive, frivolous, or arbitrary, people might question the app’s reliability or trustworthiness.

On the other hand, in an app that encourages an immersive task—such as a game—users expect a captivating appearance that promises fun and excitement and encourages discovery. People don’t expect to accomplish a serious or productive task in a game, but they expect the game’s appearance and behavior to integrate with its purpose.

Consistency

Consistency lets people transfer their knowledge and skills from one part of an app’s UI to another and from one app to another app. A consistent app isn’t a slavish copy of other apps and it isn’t stylistically stagnant; rather, it pays attention to the standards and paradigms people are comfortable with and it provides an internally consistent experience.

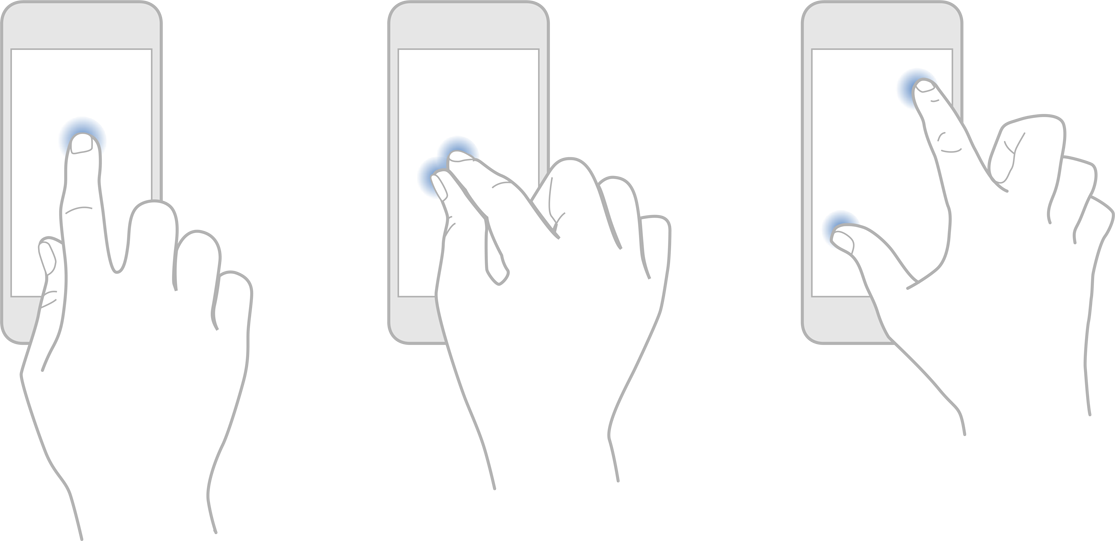


To determine whether an iOS app follows the principle of consistency, think about these questions:

* Is the app consistent with iOS standards? Does it use system-provided controls, views, and icons correctly? Does it incorporate device features in ways that users expect?
* Is the app consistent within itself? Does text use uniform terminology and style? Do the same icons always mean the same thing? Can people predict what will happen when they perform the same action in different places? Do custom UI elements look and behave the same throughout the app?
* Within reason, is the app consistent with its earlier versions? Have the terms and meanings remained the same? Are the fundamental concepts and primary functionality essentially unchanged?

Direct Manipulation

When people directly manipulate onscreen objects instead of using separate controls to manipulate them, they're more engaged with their task and it’s easier for them to understand the results of their actions.



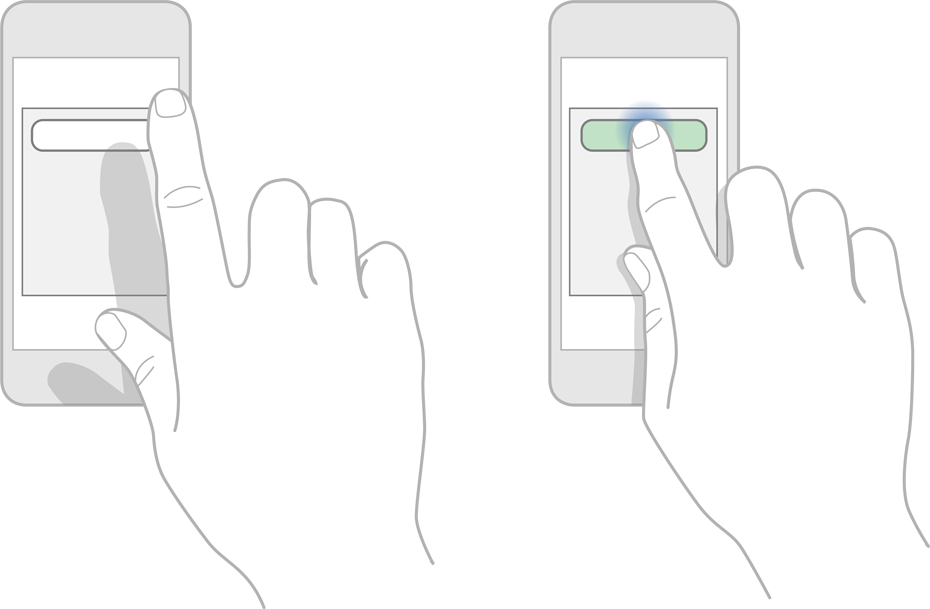
Using the Multi-Touch interface, people can pinch to directly expand or contract an image or content area. And in a game, players move and interact directly with onscreen objects—for example, a game might display a combination lock that users can spin to open.

In an iOS app, people experience direct manipulation when they:

* Rotate or otherwise move the device to affect onscreen objects
* Use gestures to manipulate onscreen objects
* Can see that their actions have immediate, visible results

Feedback

Feedback acknowledges people’s actions, shows them the results, and updates them on the progress of their task.



The built-in iOS apps provide perceptible feedback in response to every user action. List items and controls highlight briefly when people tap them and—during operations that last more than a few seconds—a control shows elapsing progress.

Subtle animation can give people meaningful feedback that helps clarify the results of their actions. For example, lists can animate the addition of a new row to help people track the change visually.

Sound can also give people useful feedback, but it shouldn’t be the only feedback mechanism because people can’t always hear their devices.

Metaphors

When virtual objects and actions in an app are metaphors for familiar experiences—whether these experiences are rooted in the real world or the digital world—users quickly grasp how to use the app.

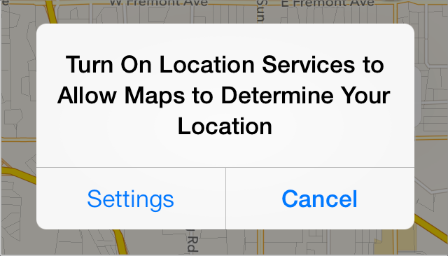
It’s best when an app uses a metaphor to suggest a usage or experience without letting the metaphor enforce the limitations of the object or action on which it’s based.

iOS apps have great scope for metaphors because people physically interact with the screen. Metaphors in iOS include:

* Moving layered views to expose content beneath them
* Dragging, flicking, or swiping objects in a game
* Tapping switches, sliding sliders, and spinning pickers
* Flicking through pages of a book or magazine

User Control

People—not apps—should initiate and control actions. An app can suggest a course of action or warn about dangerous consequences, but it’s usually a mistake for the app to take decision-making away from the user. The best apps find the correct balance between giving people the capabilities they need while helping them avoid unwanted outcomes.



Users feel more in control of an app when behaviors and controls are familiar and predictable. And when actions are simple and straightforward, users can easily understand and remember them.

People expect to have ample opportunity to cancel an operation before it begins, and they expect to get a chance to confirm their intention to perform a potentially destructive action. Finally, people expect to be able to gracefully stop an operation that’s underway.

From Concept to Product

Define Your App

An *app definition statement* is a concise, concrete declaration of an app’s main purpose and its intended audience.

Create an app definition statement early in your development effort to help you turn an idea and a list of features into a coherent product that people want to own. Throughout development, use the definition statement to decide if potential features and behaviors make sense. Take the following steps to create a robust app definition statement.

1. List All the Features You Think Users Might Like

Go ahead and brainstorm here. At this point, you’re trying to capture all the tasks related to your main product idea. Don’t worry if your list is long; you’ll narrow it down later.

Imagine that your initial idea is to develop an app that helps people shop for groceries. As you think about this activity, you come up with a list of related tasks—that is, potential features—that users might be interested in. For example:

* Creating lists
* Getting recipes
* Comparing prices
* Locating stores
* Annotating recipes
* Getting and using coupons
* Viewing cooking demos
* Exploring different cuisines
* Finding ingredient substitutions

2. Determine Who Your Users Are

Now you need to figure out what distinguishes your app’s users from all other iOS users. In the context of your main idea, what’s most important to them? Using the grocery-shopping example, you might ask whether your users:

* Usually cook at home or prefer ready-made meals
* Are committed coupon-users or think that coupons aren’t worth the effort
* Enjoy hunting for speciality ingredients or seldom venture beyond the basics
* Follow recipes strictly or use recipes as inspiration
* Buy small amounts frequently or buy in bulk infrequently
* Want to keep several in-progress lists for different purposes or just want to remember a few things to buy on the way home
* Insist on specific brands or make do with the most convenient alternatives
* Tend to buy a similar set of items on each shopping trip or buy items listed in a recipe

After musing on these questions, imagine that you decide on three characteristics that best describe your target audience: Love to experiment with recipes, are often in a hurry, and are thrifty if it doesn’t take too much effort.

3. Filter the Feature List Through the Audience Definition

If, after deciding on some audience characteristics, you end up with just a few app features, you’re on the right track: Great iOS apps have a laser focus on the task they help users accomplish.

For example, consider the long list of possible features you came up with in Step 1. Even though these are all useful features, not all of them are likely to be appreciated by the audience you defined in Step 2.

When you examine your feature list in the context of your target audience, you conclude that your app should focus on three main features: Creating lists, getting and using coupons, and getting recipes.

Now you can craft your app definition statement, concretely summarizing what the app does and for whom. A good app definition statement for this grocery-shopping app might be:

“A shopping list creation tool for thrifty people who love to cook.“

4. Don’t Stop There

Use your app definition statement throughout the development process to determine the suitability of features, controls, and terminology. For example:

**As you consider adding a new feature**, ask yourself whether it’s essential to the main purpose of your app and to your target audience. If it isn’t, set it aside; it might form the basis of a different app. For example, you’ve decided that your users are interested in adventurous cooking, so emphasizing boxed cake mixes and ready-made meals would probably not be appreciated.

**As you consider the look and behavior of the UI**, ask yourself whether your users appreciate a simple, streamlined style or a more overtly thematic style. Be guided by what people might expect to accomplish with your app, such as the ability to accomplish a serious task, to get a quick answer, to delve into comprehensive content, or to be entertained. For example, although your grocery list app needs to be easy to understand and quick to use, your audience would probably appreciate a themed UI that displays plenty of beautiful pictures of ingredients and meals.

**As you consider the terminology to use**, strive to match your audience’s expertise with the subject. For example, even though your audience might not be made up of expert chefs, you’re fairly confident that they want to see the proper terms for ingredients and techniques.

Tailor Customization to the Task

The best iOS apps balance UI customization with clarity of purpose and ease of use. To achieve this balance in your app, be sure to consider customization early in the design process. Because concerns about branding, originality, and marketability often influence customization decisions, it can be challenging to stay focused on how customization impacts the user experience.

Start by considering the tasks in your app: How often do users perform them and under what circumstances?

For example, imagine a calculator app that uses an elaborate, artistic style and imaginative layout to display familiar calculator elements. The meticulously rendered artwork and the imaginative layout don’t prevent people from understanding how to tap the buttons and read the results. But for people who simply need to get their jobs done, the novelty of the experience wears off quickly and the beautiful custom UI becomes a hindrance.



In contrast, consider GarageBand. GarageBand could have helped people make music without displaying beautiful, realistic instruments, but this would have made the app less intuitive and less enjoyable to use. In GarageBand, the custom UI not only shows people how to use the app, it also makes the main task—that is, making music—easier to accomplish.



As you think about how customization might enhance or detract from the task your app enables, keep these guidelines in mind.

**Always have a reason for customization.** Ideally, UI customization facilitates the task people want to perform and enhances their experience. As much as possible, you need to let your app’s task drive your customization decisions.

**As much as possible, avoid increasing the user’s cognitive burden.** Users are familiar with the appearance and behavior of the standard UI elements, so they don’t have to stop and think about how to use them. When faced with elements that don’t look or behave at all like standard ones, users lose the advantage of their prior experience. Unless your unique elements make performing the task easier, users might dislike being forced to learn new procedures that don’t transfer to any other apps.

**Be internally consistent.** The more custom your UI is, the more important it is for the appearance and behavior of your custom elements to be consistent within your app. If users take the time to learn how to use the unfamiliar controls you create, they expect to be able to rely on that knowledge throughout your app.

**Always defer to the content.** Because the standard elements are so familiar, they don’t compete with the content for people’s attention. As you customize your UI, take care to ensure that it doesn’t overshadow the content people care about. For example, if your app allows people to watch videos, you might choose to design custom playback controls. But whether you use custom or standard playback controls is less important than whether the controls fade out after the user begins watching the video and reappear with a tap.

**Think twice before you redesign a standard control.** If you plan on doing more than customizing a standard control, make sure your redesigned control provides as much information as the standard one. For example, if you create a switch control that doesn’t indicate the presence of the opposite value, people might not realize that it’s a two-state control.

**Be sure to thoroughly user-test custom UI elements.** During testing, closely observe users to see if they can predict what your elements do and if they can interact with them easily. If, for example, you create a control that has a hit target smaller than 44 x 44 points, people will have trouble activating it. Or if you create a view that responds differently to a tap than it does to a swipe, be sure the functionality the view provides is worth the extra care people have to take when interacting with it.

Prototype & Iterate

Before you invest significant engineering resources into the implementation of your design, it’s a really good idea to create prototypes for user testing. Even if you can get only a few colleagues to test the prototypes, you’ll benefit from their fresh perspectives on your app’s functionality and user experience.

In the very early stages of your design you can use paper prototypes or wireframes to lay out the main views and controls, and to map the flow among screens. You can get some useful feedback from testing wireframes, but their sparseness may mislead testers. This is because it’s difficult for people to imagine how the experience of an app will change when wireframes are filled in with real content.

You’ll get more valuable feedback if you can put together a fleshed-out prototype that runs on a device. When people can interact with your prototype on a device, they’re more likely to uncover places where the app doesn’t function as they expect, or where the user experience is too complex.

The easiest way to create a credible prototype is to use a storyboard-based Xcode template to build a basic app, and populate it with some appropriate placeholder content. (A *storyboard* file captures the entire UI of your app, including the transitions among different screens.) Then, install the prototype on a device so that your testers can have as realistic an experience as possible.

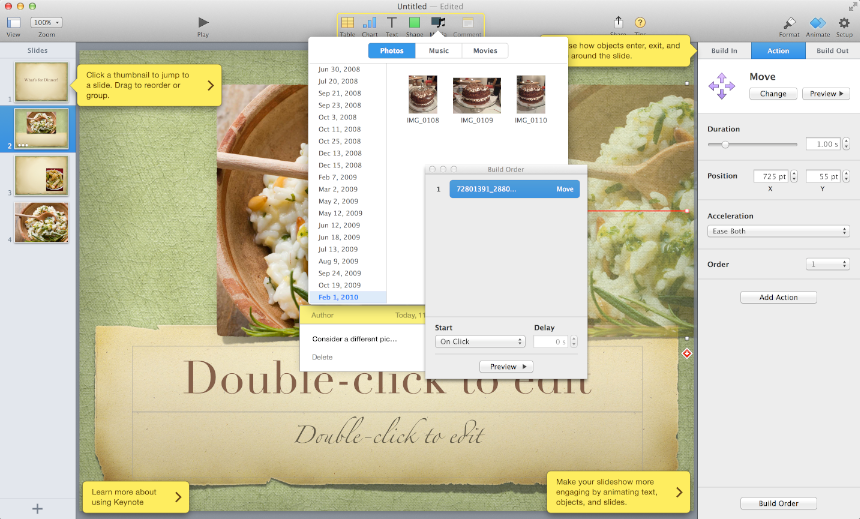
You don’t need to supply a large amount of content or enable every control in your prototype app, but you do need to provide enough context to suggest a realistic experience. Aim for a balance between the typical user experience and the more unusual edge cases. For example, if it’s likely that your app will handle long lists of items, you should avoid creating a prototype that displays only one or two list items. And for testing user interactions, as long as testers can tap an area of the screen to advance to the next logical view or to perform the main task, they’ll be able to provide constructive feedback.

When you base your prototype on an Xcode app template, you get lots of functionality for free and it’s relatively easy to make design adjustments in response to feedback. With a short turnaround time, you should be able to test several iterations of your prototype before you solidify your design and commit resources to its implementation. To get started learning about Xcode, see *Xcode Overview*.

Case Study: From Desktop to iOS

Keynote on iPad

Keynote on the desktop is a powerful, flexible app for creating world-class slide presentations. People love how Keynote combines ease of use with fine-grained control over myriad precise details, such as animations and text attributes.



Keynote on iPad captures the essence of Keynote on the desktop, and makes it feel at home on iPad by creating a user experience that:

* Focuses on the user’s content
* Reduces complexity without diluting capability
* Provides shortcuts that empower and delight
* Adapts familiar hallmarks of the desktop experience
* Provides feedback and communication via eloquent animation

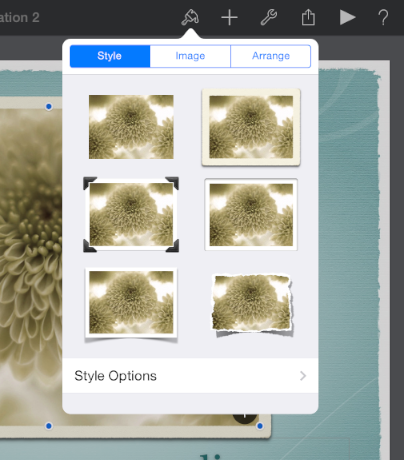
Keynote users instantly understand how to use the app on iPad because it delivers expected functionality using native iPad paradigms. New users easily learn how to use Keynote on iPad because they can directly manipulate their content in simple, natural ways.

The transformation of Keynote from the desktop to iPad is based on myriad modifications and redesigns that range from subtle to profound. These are some of the most visible adaptations:

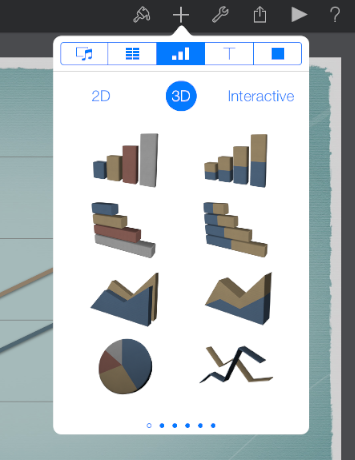
**A streamlined toolbar.** Only a handful of items are in the toolbar, but they give users consistent access to all the functions and tools they need to create their content.

mage: ../Art/keynote_toolbar_2x.png

**A simplified, prioritized inspector that responds to the user’s focus.** The Keynote on iPad inspector automatically contains the tools and attributes people need to modify the selected object. Often, people can make all the modifications they need in the first inspector view. If they need to modify less frequently changed attributes, they can drill down to other inspector views.



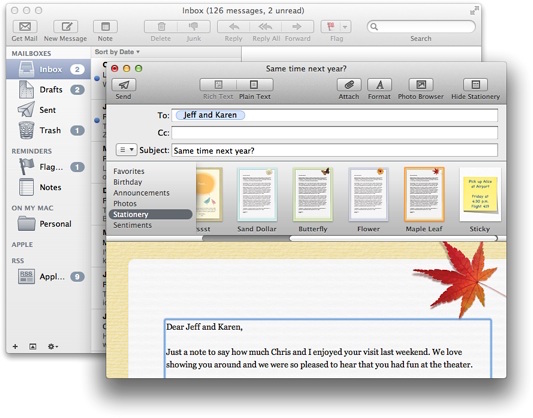
**Lots of prebuilt style collections.** People can easily change the look and feel of objects such as charts and tables by taking advantage of the prebuilt styles. In addition to color scheme, each collection includes prestyled attributes, such as table headings and axis-division marks, that are designed to coordinate with the overall theme.



**Direct manipulation of content, enriched with meaningful animation.** In Keynote on iPad, a user drags a slide to a new position, twists an object to rotate it, and taps an image to select it. The impression of direct manipulation is enhanced by the responsive animations Keynote on iPad performs. For example, a slide pulses gently as users move it and, when they place it in a new location, the surrounding slides ripple outward to make room for it.

Mail on iPhone

Mail is one of the most highly visible, well-used, and appreciated apps in OS X. It is also a very powerful program that allows users to create, receive, prioritize, and store email, track action items and events, and create notes and invitations. Mail on the desktop offers this powerful functionality in a couple of windows.



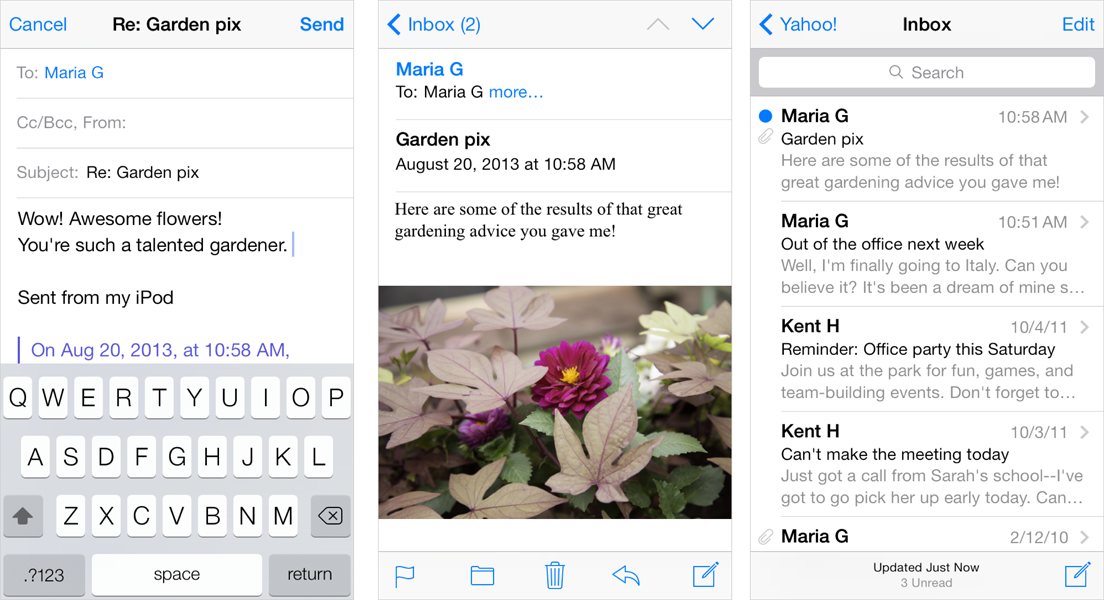
Mail on iPhone focuses on the core functionality of Mail on the desktop, helping people to receive, create, send, and organize their messages. Mail on iPhone delivers this condensed functionality in a UI tailored for the mobile experience that includes:

* A streamlined appearance that puts people’s content front and center
* Different views designed to facilitate different tasks
* An intuitive information structure that scales effortlessly
* Powerful editing and organizing tools that are available when they’re needed
* Subtle but expressive animation that communicates actions and provides feedback

It’s important to realize that Mail on iPhone isn’t a better app than Mail on the desktop; rather, it’s Mail, redesigned for mobile users. By concentrating on a subset of desktop features and presenting them in an attractively lean UI, Mail on iPhone gives people the core of the Mail experience while they’re mobile.

To adapt the Mail experience to the mobile context, Mail on iPhone innovates the UI in several key ways.

**Distinct, highly focused screens.** Each screen displays one aspect of the Mail experience: account list, mailbox list, message list, message view, and composition view. Within a screen, people scroll to see the entire contents.



**Easy, predictable navigation.** Making one tap per screen, people drill down from the general (the list of accounts) to the specific (a message). Each screen displays a title that shows people where they are, and a back button that makes it easy for them to retrace their steps.

**Simple, tappable controls, available when needed.** Because composing a message and checking for new email are primary actions people might want to take in any context, Mail on iPhone makes them accessible in multiple screens. When people are viewing a message, functions such as reply, move, and trash are available because they act upon a message.

**Different types of feedback for different tasks.** When people delete a message, it animates into the trash icon. When people send a message, they can see its progress; when the send finishes, they can hear a distinctive sound. By looking at the subtle text in the message list toolbar, people can see at a glance when their mailbox was last updated.

Web Content in iOS

Safari on iOS provides a preeminent mobile web-viewing experience on iOS devices. People appreciate the crisp text and sharp images and the ability to adjust their view by rotating the device or pinching and tapping the screen.

Standards-based websites display well on iOS devices. In particular, websites that detect the device and do not use plug-ins look great on both iPhone and iPad with little, if any, modification.

In addition, the most successful websites typically:

* Set the viewport appropriately for the device, if the page width needs to match the device width
* Avoid CSS fixed positioning, so that content does not move offscreen when users zoom or pan the page
* Enable a touch-based UI that does not rely on pointer-based interactions

Sometimes, other modifications can be appropriate. For example, web apps always set the viewport width appropriately and often hide the UI of Safari on iOS. To learn more about how to make these modifications, see [Configuring the Viewport](https://developer.apple.com/library/ios/documentation/AppleApplications/Reference/SafariWebContent/UsingtheViewport/UsingtheViewport.html#//apple_ref/doc/uid/TP40006509) and [Configuring Web Applications](https://developer.apple.com/library/ios/documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html#//apple_ref/doc/uid/TP40002051-CH3) in[*Safari Web Content Guide*](https://developer.apple.com/library/ios/documentation/AppleApplications/Reference/SafariWebContent/Introduction/Introduction.html#//apple_ref/doc/uid/TP40002051).

Websites can adapt the desktop web experience to Safari on iOS in other ways, too:

**Accommodate the keyboard in Safari on iOS.** When a keyboard and the form assistant are visible, Safari on iPhone displays your webpage in the area below the URL text field and above the keyboard and form assistant.

**Accommodate the pop-up menu control in Safari on iOS.** In Safari on the desktop, a pop-up menu that contains a large number of items displays as it does in an OS X app; that is, the menu opens to display all items, extending past the window boundaries, if necessary. In Safari on iOS, a pop-up menu is displayed using native elements, which provides a much better user experience. For example, on iPhone, the pop-up menu appears in a *picker*, a list of choices from which the user can pick. (To learn more about the picker control, see [Picker](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Controls.html#//apple_ref/doc/uid/TP40006556-CH15-SW23).)