User interface design for the mobile web
Best practices for designing applications for multiple device platforms

James L Lentz

Web application technology reduces the cost of creating multiplatform applications. Developers can create applications that run on mobile platforms that differ in development technology, user interface style, input mechanisms, display form factor, size, and resolution. To design applications that are easy to use, and that integrate well across diverse platforms and devices, you need to consider several factors beyond conventional web applications and native mobile applications. This article explores the usability challenges of the mobile web, and provides several best practices for designing mobile web applications.

Introduction

Emerging technologies often go through a period of rapid change as innovators seek to exploit new possibilities. Alternative solutions to problems compete for mind- and market-share. Mobile user interface (UI) technology is in the midst of this evolutionary phase. Phones and tablets that use Apple's iOS (iPhone, iPod Touch, and iPad), Google's Android architecture, Blackberry's operating system, HP's webOS, and Windows® Phone 7 mobile operating system all offer diverse UI design approaches.

UI diversity is intentional. Platforms must differentiate themselves to claim a share of the market. On the Android platform, carriers and device vendors also must distinguish their products, creating even more diversity. Though the resulting diversity of products is necessary for competitive improvement, it creates challenges for developers and designers who are creating applications and websites for these devices. To create applications that run natively on multiple device types, development teams need:

- Skills in diverse development technologies
- Knowledge of capabilities of a vast, continually changing array of devices
- Knowledge of differing UI style conventions and standards
- Multiple programming and cross porting efforts
- Expensive multidevice and multiplatform test efforts
Mobile web technology provides a more cost-effective way to develop applications that are usable on a variety of device platforms. With newly developed JavaScript libraries such as Dojo Mobile, jQuery Mobile, and Sencha Touch, mobile UI developers can “write once, run anywhere.” Developers don't need to learn different frameworks for different platforms, or redevelop or port applications to each supported platform. Users will also benefit from the zero-install nature of web applications. They'll always be using the latest version of the application, and won't have to install updates from the online application store. Application deployers benefit by not having to worry about supporting users running on different versions of the same application.

However, designing quality mobile web applications has its own set of complications. First, mobile user interfaces are essentially a new paradigm for user interaction due to:

- Smaller form factors
- Touch interfaces
- Acceleration sensing
- Orientation awareness
- Pervasive animation
- Simulations of physical behavior

Second, because a web UI should run on any device regardless of the its size, form factor, or feature set, designers must consider more variables when compared to designing a traditional desktop web application.

This article explores considerations and best practices for designing mobile web UIs. Though there's very little detail on implementation, the article occasionally references HTML5 and the Dojo Toolkit mobile UI components. By necessity, there is discussion of issues relevant for native mobile application design, but the main focus is on cross-platform design. Because of relative market share, emphasis is on the iOS, Android, and Blackberry user expectations.

**Display size**

Compactness can enable a device to be used just about anywhere, but it can also work against many aspects of usability. A small screen limits the information that can be legibly displayed. While designing, text and images can quickly consume the limited screen space, causing a trade-off between content and user interactions.

Smartphones are small, and tablets are in the range of netbook to laptop size. Many vendors offer both types of devices in a variety of display sizes, as shown in Figure 1. Mobile web application designs must scale to handle the wide range of displays without appearing cramped at the low end or stretched at the high end.
Input

Many popular mobile devices use touch and gestural input. While touch input can be intuitive, it is relatively imprecise. Touch targets, such as buttons, must be fairly large and widely spaced in comparison with the mouse and pointer style input in conventional installed or web applications.

On phones, constrained screen sizes, coupled with large interaction targets, result in fewer controls per panel. Fingers and hands also obscure much more of the screen on a UI than a mouse pointer icon.

Because mobile web applications are inherently cross-platform, the input characteristics of different types of devices also must be considered. Some mobile devices have a physical keyboard, some have only a virtual keyboard, and others have both. Some Blackberry devices use a touchpad for pointing, selection, and dragging. Both Blackberry and Android devices have dedicated, but differently ordered, physical buttons for various navigation actions.

Operating system diversity

Three of the most important UI design differences driven by operating systems are: navigation design, control implementation, and visual style.

Navigation

To move the user through views, iOS uses gestures and widgets. The home button on the bezel is also used to close applications and navigate out of folders. Android uses gestures, widgets, and hardware buttons (home, back, menu, search), and Blackberry uses gestures, widgets, and hardware buttons (menu and escape). To make matters more complex, input methods can vary by device model and by service providers. The problem is acute for Android devices because virtual keyboard layouts and the left to right order of bezel buttons varies by service provider and device manufacturer.

Control implementation

iOS relies heavily on software UI control features, such as virtual buttons. Users interact with widgets by touch, with the single exception of exiting an application or folder. In contrast, both Android and Blackberry devices provide physical buttons for navigating back to the previous view and for opening option menus. On iOS devices these actions are invoked with buttons.

Often, iOS applications allocate the back and context menu actions to tab row buttons. By convention, on iPhone and iPod Touch applications, tab buttons are placed at the bottom of the
view so that they're easily reached by the user's thumbs. For different reasons, Android style conventions place tab buttons near the top of the screen; they cause inadvertent physical button presses if located at the bottom of the screen.

**Visual style**

Each platform defines its own visual style through color themes, icon styles, metaphors, and widget rendering. The use of a platform visual theme is more than an aesthetic choice. Platform themes establish user expectations that user interactions in the application will follow platform conventions.

**Performance**

Although JavaScript performance is improving, mobile devices still performance-challenged. They use less powerful processors and must contend with lower network bandwidth compared to laptops and desktop systems.

**Multiple device usage**

It's probably fair to say that most smartphone users use a single phone. On the other hand, many people use the same application on multiple types of devices. A user might access the same application on an iPod Touch, a Blackberry phone, an Android tablet, and a laptop running Microsoft® Windows. As far as the user is concerned, device types are essentially different viewers into the same content space.

Multiplatform, multidevice design is complicated by differences between device types. Smartphones are good for brief interactions to accomplish focused goals, anywhere at any time. Personal computers are good for extended interactions, dealing with complex information, and switching back and forth between tasks in relatively fixed locations. Tablet interactions fall somewhere between smartphones and laptops.

Designing for multiple devices requires careful consideration and inevitable compromise among these competing requirements:

- Make good use of each device's capabilities
- Intelligently handle each device's limitations
- Provide a similar user experience on all devices

To provide a good user experience, a web application on a smartphone will often need to support different functions relative to its desktop equivalent. When displaying on a phone, you might need to remove some capabilities that make sense on the desktop, or add capabilities that make sense in a mobile context. It can be difficult to predict which functions won't be greatly missed on the mobile rendering of an otherwise complex web application.

There is no magic here. You must take care to understand and validate use cases for functions in device context in which an application is likely to be used, and to match capabilities to devices. Table 1 outlines some differences in user experiences with mobile and desktop devices.
Table 1. Mobile and desktop experience differences

<table>
<thead>
<tr>
<th>Mobile</th>
<th>Desktop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief focused interactions.</td>
<td>Continuous interaction on a single task.</td>
</tr>
<tr>
<td>Respond to a tweet or SMS message.</td>
<td>Compose a memo.</td>
</tr>
<tr>
<td>More intrusive interruptions on device.</td>
<td>Less intrusive interruptions.</td>
</tr>
<tr>
<td>Receive call while reviewing mail on smartphone.</td>
<td>Receive call on phone while reviewing mail on laptop.</td>
</tr>
<tr>
<td>Frequently changing environmental context.</td>
<td>Infrequently changing context.</td>
</tr>
<tr>
<td>Use phone during air travel (upon landing, walking to gate, and so on).</td>
<td>Using a spreadsheet application at desk.</td>
</tr>
<tr>
<td>Tendency toward transactional interactions.</td>
<td>Supports non-transactional interactions.</td>
</tr>
<tr>
<td>Checking weather forecast.</td>
<td>Composing a report.</td>
</tr>
<tr>
<td>Viewing dominates interacting.</td>
<td>Balanced viewing and interacting.</td>
</tr>
<tr>
<td>Flipping through photos.</td>
<td>Editing vacation photo albums.</td>
</tr>
<tr>
<td>Page loads are more disruptive.</td>
<td>Page loads are less disruptive.</td>
</tr>
<tr>
<td>Mobile web browsing experience.</td>
<td>Desktop web browsing experience.</td>
</tr>
<tr>
<td>Reading an e-book.</td>
<td>Using a word processing application.</td>
</tr>
<tr>
<td>Relatively poor response time.</td>
<td>Good response time.</td>
</tr>
<tr>
<td>Map updates.</td>
<td>Highly immersive gaming.</td>
</tr>
<tr>
<td>More social.</td>
<td>Less social</td>
</tr>
<tr>
<td>Relative importance of phone, texting, geo-social, sharing applications.</td>
<td>Relative importance of office productivity applications.</td>
</tr>
</tbody>
</table>

Above all, when designing for multiple device usage scenarios, it's important that user data be shared among all devices on which a user accesses an application. Think of device-type instances of an application as providing tailored "views" into the same content.

**Mobile web UI best practices**

Thus far I've discussed the usability challenges of the mobile web, and have covered several considerations for multiplatform applications. The remainder of this article explores some best practices when designing mobile web UIs.

**Use a toolkit**

From a strictly development-oriented perspective, libraries, toolkits, and frameworks are work-savers. Using a standard UI component library can save an enormous amount of development and test time that can be spent on low-level design, coding, handling browser differences, testing, debugging, and maintenance. Quality UI component libraries also improve ease of use by ensuring that commonly used controls look and behave as expected.

The Dojo Toolkit version 1.5 introduced basic mobile UI features. Version 1.6 and the version 1.7 Beta cover the UI control features and interactions typically found on native applications. The WebSphere® Application Server Feature Pack for Web 2.0 and Mobile includes the Dojo Toolkit version 1.7, along with several useful application services and Diagrammer visualization features.

**Design responsively**

The primary design challenge for mobile web applications is: How do you create positive user experiences for an application that appears on screens measuring from a few inches square up
through tablets, laptops, and devices using large displays? The responsive web is the design philosophy and set of techniques that try to address this issue.

The goal of response web design is to make each website or web application appear as if it were designed specifically for each device and browser on which it's displayed. Perhaps the responsive web should be called the "more responsive web," because the web has always dealt with issues of displaying content in un-maximized browsers and on monitors of varying size and resolution. The smaller sizes and diversity of mobile devices has simply raised the ante.

Responsive web implementation relies on the use of CSS, media queries, and JavaScript to adapt the presentation of content to devices. Media Queries, a sub-specification of CSS3, lets you associate a different style sheet with different media or display characteristics. For example, a style sheet can be selected based on the device's screen height, width, aspect ratio, and resolution.

A detailed explanation of how to implement responsive designs is beyond the scope of this article. Several techniques are described in Responsive Web Design: What It Is and How To Use It and A List Apart post on responsive design. The rest of this section provides an overview of some common design approaches.

**Responsive layouts**

Complex layouts often work well on large displays, but they become unusable on smartphones. Conversely, extremely simple content layouts can appear uninteresting, or unreasonably tedious to read and navigate, when ample real estate is available. Many hand-held devices sense changes in orientation. Well-designed responsive content automatically adjusts its layout to fit the size and orientation of the device.

One approach is to reformat multicolumn layouts to a single column layout when the device screen size and resolution become too small to support multiple columns. Figure 2 and Figure 3 show an example.

iPad applications sometimes exploit the familiar left pane navigation – right pane content pattern, as shown below. This works well in large format screens but does not scale down to smaller, phone-sized devices.
In this case, a mobile web application can be designed so the left navigation list appears by itself in a view and the content appears in a second view (Figure 3). You can use the Dojo toolkit mobile widgets FixedSplitter and FixedSplitterPane to implement this approach.

Tab rows pose another concern for responsive design. As mentioned, iOS devices place tab rows at the bottom of the view and Android devices place them at the top of the view. The Safari mobile web browser follows the iOS convention and positions browser navigation controls in a tab row at the bottom of the view, as shown in part a of Figure 4. A problem can arise with web applications that follow the tabs on bottom convention because the browser and application tab rows will end up stacked. The solution is simple. Placing the meta tag `<meta name="apple-mobile-web-app-
in the HTML header of a mobile web application will suppress browser artifacts when the user adds an application link to the home screen, as shown in part b of Figure 4.

**Figure 4. Tab row placement in iOS and Android**

To avoid a similar problem, and to follow proper Android application guidelines, web applications should use media queries to detect and place the tab row below the view heading on Android devices (Figure 4, b and d).

**Responsive widgets**

Because devices differ in the physical controls that are present, software buttons that would otherwise be redundant with physical buttons on Android and Blackberry devices should be hidden when a web application is viewed on these devices.

The size of widgets is another consideration. Because displays vary in resolution (pixels per inch), mobile UI design guidelines are often expressed in physical size units. Minimum recommendations for touch target size vary between 7 mm and 10 mm square (these sizes are in the ranges of 36 – 52 pixels square for the iPad 2, and 90 – 128 pixels square for the iPhone Retina display). The minimum space between targets should be around 1-2 mm.

**Responsive content**

Sometimes, reorganizing content on the page isn't sufficient to handle the range of display sizes. You shouldn't expect to shoehorn every application into a 2 by 3 inch square and still retain usability. Usage patterns often differ between smaller and larger devices. Small devices are most suitable for brief and focused interactions; large devices are suited for longer, more intensive interactions. For example, a weather application on your mobile might contain geolocation-sensitive status on current and upcoming conditions. A desktop web version of the same application might provide more content on weather history for this and other locations, news articles or videos on weather events, and so forth.

Text content can be scaled down by selectively displaying only the most important material, or by linking and moving subordinate sections onto separate pages.

Images can also be responsively scaled for large and small devices. A range of approaches is possible. The simplest approach is to allow the image to rescale itself, but there are performance implications. When large images are sized down, important details can become difficult for users to see. An alternative approach is to provide thumbnail images that zoom in on touch when viewed.
in a mobile device but are displayed full-size on a desktop browser. In other cases, it might make sense to create multiple images, cropped to show only the most important features, and selectively display the image appropriate to the target device resolution.

**Look native**

Each major device platform has its own visual signature defined by color palette, icon styles, and typography. An application designed for one device can look hopelessly out of place on another provider's device. Properly adapting applications to the displaying device, in visual style and interaction, can greatly help in meeting user expectations for a consistent experience on their device.

Figure 5 shows the platform themes supported by the `dojox.mobile.deviceTheme` (Dojo version 1.7 Beta) module. You can use this module to automatically detect the device type and set the appropriate theme for a target device.

**Figure 5. Dojox Mobile themes for Android, Blackberry, and iOS**

![Dojox Mobile themes for Android, Blackberry, and iOS](image)

**Interact natively**

Responsive design involves more than just handling display differences. The design still must consider responsively hiding or showing virtual buttons, depending whether the target device handles the capability with a physical button (for example, back, home, and option). The device browser will handle other interaction differences between devices.

**Recycle designs for a better experience**

Exploiting UI design patterns facilitates ease of use for two reasons.

- By definition, a pattern is a proven solution to a common design problem. Some patterns are proven by formal usability testing, and others through the "natural selection" of designs. Over time, unusable designs will be abandoned and better designs will be copied.
- Because patterns are common design practices, they're familiar to users. Pattern-based UIs will usually require less learning by users.

There are a few mobile UI pattern sites on the web; pttrns and Mary Sheibley's Mobile-Patterns are the most comprehensive. Both sites provide examples of common design idioms. While
these pattern libraries don't provide design guidance, they can serve as a design inspiration while helping you design UIs that leverage user experience with other applications.

**Let users choose**

Aside from degree of mobility, two primary differences between mobile and traditional computer devices are pointing precision and ease of typing. In both cases, traditional computer interfaces are superior to mobile device interfaces. Touch input interfaces provide less tactile feedback and are imprecise. Regardless of whether a miniature physical keyboard or a virtual keyboard is employed, typing speed and accuracy on mobile devices are poor when compared to performance on full size computer keyboards. When a device employs a virtual keyboard, typing is a disruptive operation—the virtual keyboard pops up and obscures much of the view.

Recommendations:

- When possible, let users select options rather than requiring them to input text.
- Give the user adequately-sized targets for touch interactions (for example, 10 mm square).
- When a set of possible options is large and keyboard input can't be avoided, provide auto-completion or progressive filtering of selection lists to minimize typing.
- Use the rich input type capabilities of HTML5 to provide the optimal virtual keyboard for the input (text, numeric, phone, date, time, or URL) to avoid the need for users to manually swap keyboard screens to access infrequently used characters.

**Stay close to home**

Tiny displays and large human fingers mean that navigation elements and content must compete for room on mobile device screens. Consequently, content and navigation must often be presented in separate views. Deep navigation becomes tedious and taxes user memory.

Like most aspects of mobile device design, navigation should be simple. When possible, navigation should be kept to two or three levels. When navigation exceeds two levels, make sure there's a convenient way to return to the application "Home" view. On iOS devices, it will likely mean providing both a virtual home and back buttons in some views. When accessed by Android devices, these buttons should not be displayed because physical back and home buttons are provided.

**Show users how**

One way that mobile device interfaces have addressed small screen sizes is by supporting gestural interaction. For example, rather than dragging a scroll bar control feature, the user can scroll vertically or horizontally by swiping with a finger anywhere on the page. The scrollbar control can be reduced to a minimally sized or temporarily visible position indicator because visual control graphics, such as up/down buttons and a scroll bar "thumb" feature, are no longer needed. Multiple gestures can be available in the same view, for different operations, without any visible control features occupying space (for example, pinch open/close for zooming, swipe right/left for panning, swipe up/down for scrolling, and tap to close).
While gestures save screen real estate, they do so at a cost. Merely enabling gestural control leaves the user no way of knowing what actions are available. The remedy is to provide hints through visual cues, called affordances, that suggest to the user how to interact with the view. Visual affordances do consume some real estate, but they usually use much less of it than controls that serve as touch targets.

A very simple example of a visual affordance is the ">" symbol in list items that indicates a view change follows a tap. Many other affordances are based on realistic renderings of physical objects. For example, Apple’s iBook uses a physical book metaphor to indicate page flipping gestures. Spinner widgets provide visual affordances by their graphic appearance. In this case, the visual metaphor of a tumbler or roller control indicates up and down dragging to change values.

**Be graphic**

Mobile UIs are usually graphical. An application with appealing graphical design will be preferred over applications that are textual in nature. This may be simply due to aesthetic qualities, but well designed, highly graphical UIs usually offer many practical benefits as well. Good graphic design communicates simplicity and engages the user. As described in the previous section, rich visual control affordances can efficiently "tell" the user how to interact with the device. Statistical visualizations can usually provide more information that is more easily grasped than verbal or tabular descriptions of data.

As shown in Figure 6, version 1.7 of the Dojo toolkit provides many rich graphical and visualization widgets that have been optimized for use in mobile web applications.

**Figure 6. Dojo graphical and visualization widgets**

![](image)

**Avoid mobile web UI design pitfalls**

Avoid the following common pitfalls when designing your mobile web UI.

**Doing nothing**

Mobile web browsers will display many web applications without modification. Anyone who has tried to use an unmodified web application on a smartphone knows this can be merely
annoying to highly frustrating. Complex web applications are simply not suited for small displays, imprecise touch interactions, and low network bandwidth.

**Ignoring performance**

Poor performance is rarely, if ever, tolerated by users, especially if alternatives are available. The usual advice for keeping native mobile applications simple is especially true for mobile web applications. Large images, page loads, and other server fetches impact performance and provide opportunities for users to notice response lags.

**Relying on help**

As described earlier, mobile applications tend to be used intermittently in brief interactions with crisp goals. This type of engagement means that users are less tolerant of having to learn how to interact with the device.

Application use should be so intuitive that no help is needed. This is true of all UIs, but interacting with help content is seen by users as an additional task that stands between them and their goals. When a mobile web application is so intuitive that help UI and content isn't needed, there is less code and content to download to the device.

**Misapplying creativity**

Take care when introducing novel interaction techniques, idiosyncratic icons, and visual treatments in mobile application design. Users will understand how to use applications that look and behave similarly to other applications that they use. Unnecessarily inventive interactions and styling increase the likelihood that users will need some form of assistance.

**Ignoring browser interactions**

By definition, mobile web applications run in mobile web browsers. As with the desktop web, the browser presents an additional layer of user interface around the web application. And, again similar to the desktop web, navigation issues arise when the user has access to navigation controls inside the application as well as in the browser. When possible, run your application in kiosk mode and provide all necessary navigation inside of the application.

**Summary**

Many issues associated with developing native applications for multiple device platforms can be avoided by developing and deploying mobile web technology. However, design issues that are unique to both mobile and mobile web applications should be addressed to ensure ease of use.

When designing applications for the mobile web, you have to pay greater attention to both small device form factors and the wide range of device sizes when tablets are considered. Because of UI differences imposed by mobile device platform vendors and carriers, designs must adapt to unique characteristics of each platform. Standardized controls, and native device themes available with the Dojo 1.7 Toolkit, can ease both the design and development of mobile web applications.
Related topics

- Download the WebSphere Application Server Feature Pack for Web 2.0 and Mobile V1.1.0.
- Mary Sheibely, the lead designer at FourSquare, provides a collection of Mobile UI Patterns.
- Learn more about Android UI Design Patterns.
- Pttrns is another resource for mobile UI patterns.
- Read 5 tips for converting iOS UI to Android to explore iOS and Android style differences.
- Learn more about the iOS Human Interface Guidelines from Apple Inc.
- The BlackBerry Smartphones UI Guidelines can help you make application design decisions for BlackBerry devices that are supported in the BlackBerry Java Development Environment 6.0.
- Check out the Android Design Guidelines - Version 1 by Adam Beckley, Mutual Mobile, Feb 2011. (Android does not yet provide an official human interface guideline.)
- Read the Media Queries specification from W3C.
- "Mobile Usability" explores how, in user testing, website use on mobile devices got very low scores, especially when users accessed "full" sites that weren't designed for mobile.
- Read "iPhone Apps need low starting hurdles."
- Learn all about the Dojo Mobile features.
- Use the Dojo toolkit mobile widgets FixedSplitter and FixedSplitterPane.
- The WebSphere Application Server Feature Pack for Web 2.0 and Mobile V. 1.1 includes Dojo 1.7 and many more features for developing rich internet and mobile applications.
- The blog Touch Target Sizes summarizes minimum recommended touch target sizes.
- "Responsive Web Design: What It Is and How To Use It" discusses how design and development should respond to the user's behavior and environment based on screen size, platform, and orientation.
- "Responsive Web Design" provides a discussion on how to design in a forward-thinking mode.
- Get Dojo
- Try out IBM software for free. Download a trial version, log into an online trial, work with a product in a sandbox environment, or access it through the cloud. Choose from over 100 IBM product trials.