IN THIS CHAPTER

- Learning about mobile-first web development
- Understanding the main principles of coding a mobile-first site
- Getting started with jQuery Mobile
- Delivering images responsively to mobile users
- Storing user data in the web browser instead of on the server

Chapter 1
Exploring Mobile-First Web Development

Don't be afraid to start small. Some of the biggest successes in mobile today came from small experiments and teams of passionate web designers and developers. You don't need to know everything about mobile — just take what you do know and go.

— LUKE WROBLEWSKI

If you've been hanging around the web for a while, you probably remember the days when you'd surf to a site using a small screen such as a smartphone or similar portable device, and instead of seeing the regular version of the site, you'd see the "mobile" version. In rare cases, this alternate version would be optimized for mobile viewing and navigation, but more likely it was just a poor facsimile of the regular site with a few font changes and all the interesting and useful features removed.

Seen from the web developer's viewpoint, the poor quality of those mobile sites isn't all that surprising. After all, who wants to build and maintain two versions of the same site? Fortunately, the days of requiring an entirely different site to

support mobile users are long gone. Yes, using responsive web design enables you to create a single site that looks and works great on everything from a wall-mounted display to a handheld device. But in modern web development, there's a strong case to be made that all web pages should be built from the ground up as though they were going to be displayed only on mobile devices. In this chapter, you explore the principles and techniques behind this mobile-first approach to web development.

What Is Mobile-First Web Development?

As I discuss in Book 7, Chapter 1, when you develop a web page to look good and work well on a desktop-sized screen, there are a number of responsive tricks you can employ to make that same code look good and work well on a mobile device screen:

- You can use percentages for horizontal measurements.
- You can use relative units such as `em` and `rem` for vertical measurement and font sizes.
- You can use media queries to remove elements when the screen width falls below a specified threshold.


REMEMBER

That third technique — the one where you remove stuff that doesn't fit on a smaller screen — is known in the web coding trade as *regressive enhancement (RE)*. RE has ruled the web development world for many years, but lately there's been a backlash against it. Here's why:

- RE relegates mobile screens to second-class web citizens.
- RE leads to undisciplined development because coders and designers inevitably stuff a desktop-sized screen with content, widgets, and all the web bells and whistles.


REMEMBER

What's the solution? You've probably guessed it by now: *progressive enhancement*, which means starting with content that fits on a base screen width and then adding components as the screen gets bigger. When that original content represents what's essential about your page, and when that base screen width is optimized for mobile devices — especially today's most popular smartphones — then you've got yourself a *mobile-first* approach to web development.

Learning the Principles of Mobile-First Development

Let me be honest right off the top: Mobile-first web development is daunting because if you're used to having the giant canvas of a desktop screen to play with, starting instead with a screen that's a mere 320- or 400-pixels across can feel a tad claustrophobic. However, I can assure you that it only seems that way because of the natural tendency to wonder how you're possibly going to shoehorn your massive page into such a tiny space. Mobile-first thinking takes the opposite approach by ignoring (at least at the beginning) large screens and, instead, focusing on what works best for mobile screens which, after all, represent the majority of your page visitors. Thinking the mobile-first way isn't hard: It just means keeping a few key design principle in mind.

Mobile first means content first

One of the biggest advantages of taking a mobile-first approach to web development is that it forces you to prioritize. That is, a mobile-first design means that you include in the initial layout only those page elements that are essential to the user's experience of the page. This essential-stuff-only idea is partly a response to having a smaller screen size in which to display that stuff, but it's also a necessity for many mobile users who are surfing with sluggish Internet connections and limited data plans. It's your job — no, scratch that, it's your duty as a conscientious web developer — to make sure that those users aren't served anything superfluous, frivolous, or in any other way nonessential.

That's all well and good, I hear you thinking, but define "superfluous" and "frivolous." Good point. The problem, of course, is that one web developer's trivial appetizer is another's essential meat and potatoes. Only you can decide between what's inconsequential and what's vital, depending on your page goals and your potential audience.

So the first step towards a mobile-first design is to decide what's most important in the following content categories:

REMEMBER

- **Text:** Decide what words are essential to get your page's message across. Usability expert Steve Krug tells web designers to "Get rid of half the words on each page, then get rid of half of what's left." For a mobile-first page, you might need to halve the words once again. Be ruthless. Does the user really need to see that message from the CEO or your "About Us" text? Probably not.

- **Images:** Decide what images are essential for the user, or whether any images are needed at all. The problem with images is that, although everyone likes a bit of eye candy, that sweetness comes at the cost of screen real estate and bandwidth. If you really do need to include an image or two in your mobile-first page, then at least serve up smaller images to your mobile visitors. To learn how to do that, see "Delivering images responsively," later in this chapter.
- **Navigation:** All users need to be able to navigate your site, but the recent trend is to create gigantic menus that include links to every section and page on the site. Decide which of those links are truly important for navigation and just include those in your mobile-first layout.
- **Widgets:** Modern web pages are festooned with widgets for social media, content scrollers, photo lightboxes, automatic video playback, and, of course, advertising. Mobile users want to see content first, so consider ditching the widgets altogether. If there's a widget you really want to include, and you're sure it won't put an excessive burden on either the page's load time or the user's bandwidth, push the widget to the bottom of the page.

Pick a testing width that makes sense for your site


REMEMBER

For most websites, testing a mobile-first layout should begin with the smallest devices, which these days means smartphones with screens that are 320 pixels wide. However, you don't necessarily have to begin your testing with a width as small as 320px. If you have access to your site analytics, they should tell you what devices your visitors use. If you find that all or most of your mobile users are on devices that are at least 400 pixels wide, then that's the initial width you should test for your mobile-first layout.

Get your content to scale with the device

For your mobile-first approach to be successful, it's paramount that you configure each page on your site to scale horizontally with the width of the device screen. You do that by adding the following `<meta>` tag to the `head` section of each page:

```
<meta name="viewport" content="width=device-width, initial-
  scale=1.0">
```

This instructs the web browser to do two things:

- Set the initial width of the page content to the width of the device screen.
- Set the initial zoom level of the page to 1.0, which means the page is neither zoomed in nor zoomed out.

Build your CSS the mobile-first way

When you're ready to start coding the CSS for your page, understand that the style definitions and rules that you write will be your page defaults — that is, these are the styles the browser will display on all devices, unless you've defined one or more media queries to override these defaults. (I talk more about mobile-first media queries shortly.) You shouldn't have to write any special rules as long as you follow a few basic tenets of responsive web design:

REMEMBER

- Use the relative units `%` or `vw` for horizontal measures such as width and padding.
- Use the relative units `rem` or `em` for vertical measures and font sizes.
- Make all your images responsive.
- Use flexbox for the page layout, and be sure to apply `flex-wrap: wrap` to any flexbox container.

It's also important to make sure that your mobile-first layout renders the content just as you want it to appear on the mobile screen. This means avoiding any tricks such as using the flexbox `order` property to mess around with the order of the page elements.

Finally, and perhaps most importantly, be sure to hide any unnecessary content by styling that content with `display: none`.

In the end, your mobile-first CSS should be the very model of simplicity and economy.

Pick a "non-mobile" breakpoint that makes sense for your content

Your mobile-first CSS code probably includes several elements that you've hidden with `display: none`. I assume you want to show those elements eventually (otherwise, you'd have deleted them altogether), so you need to decide when

you want them shown. Specifically, you need to decide what the minimum screen width is that will show your content successfully.

Notice I didn't say that you should decide when to show your hidden content based on the width of a target device. For example, many developers consider a screen to be "wide enough" when it's at least as wide as an iPad screen in portrait mode, which is 768 pixels. Fair enough, but will future iPads use this width? In fact, the current iPad Pro is 1,024 pixels wide in portrait mode.


TIP

Devices change constantly and it's a fool's game to try and keep up with them. Forget all of that. Instead, decide what minimum width is best for your page when the hidden content is made visible. How can you do that? Here's one easy way:

1. **Load your page into the Chrome web browser.**
2. **Display Chrome's developer tools.**

   Press either Ctrl+Shift+I (Windows) or ⌘+Shift+I (Mac).
3. **Use your mouse to adjust the size of the browser window:**
   - If the developer tools are below or undocked from the browser viewport, drag the right or left edge of the browser window.
   - If the developer tools are docked to the right or left of the browser viewport, drag the vertical bar that separates the developer tools from the viewport.
4. **Read the current viewport dimensions, which Chrome displays in the upper right corner of the viewport.**

   The dimensions appear as width x height, in pixels, as pointed out in Figure 1-1.
5. **Narrow the window to your mobile-first testing width (such as 320px).**
6. **Increase the width and, as you do, watch how your layout changes.**

   In particular, watch for the width where the content first looks the way you want it to appear in larger screens. Make a note of that width.


REMEMBER

The width where your full content looks good is the basis for a CSS media query breakpoint that you'll use to display the elements that were hidden in the mobile-first layout. For example, say that your mobile-first layout hides the `aside` element and that you found that your full content looks right at a width of 742px. You then can set up the following media query (using 750px for a round number):

```
@media (min-width: 750px) {
    aside {
        display: block;
    }
}
```

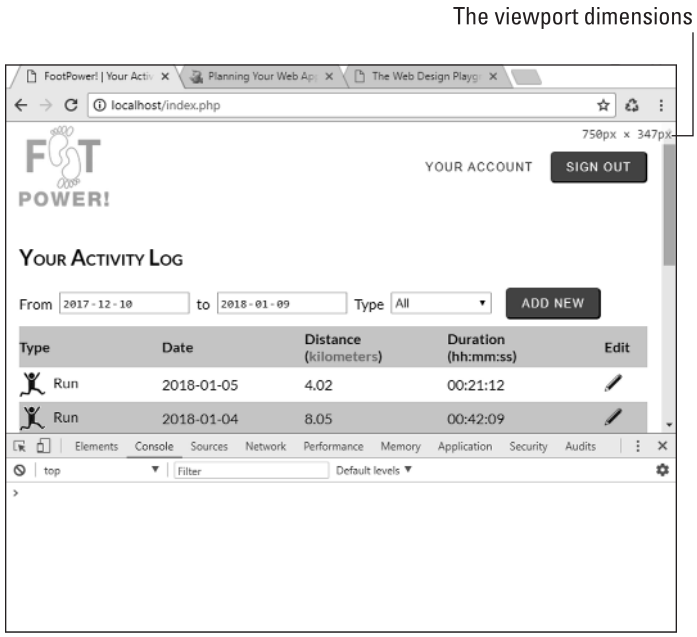

FIGURE 1-1: With Chrome's developer tools displayed, as you change the width of the browser window, Chrome displays the current viewport width and height.

This media query tells the browser that when the screen width is `750px` or more, display the `aside` element.

Going Mobile Faster with jQuery Mobile

I talk quite a bit about jQuery in Book 4, and it's safe to say that jQuery makes web development faster, easier, and even more pleasurable. I also introduce jQuery UI in Book 4, Chapter 3, and I show that it's an easy way to incorporate sophisticated components such as dialog boxes and tabs into your web projects. Now I'm going to talk briefly about yet another jQuery library: jQuery Mobile, which offers widgets optimized for mobile web apps.

What is jQuery Mobile?

Most folks nowadays have a mobile device of some description, which means that most of us are used to doing our digital duties using mobile interfaces. These interfaces include standard mobile elements such as fixed headers and footers, navigation bars, list views, tabs, switches that turn on and off, and hidden menus invoked by a "hamburger" icon.

Coding elements such as these is possible, but it would be a ton of work. Fortunately, you can skip all of that because the hardcore geeks at jQuery Mobile have done it all for you. jQuery Mobile is a set of mobile-optimized widgets that make it easy for you to design your mobile web app to have the look and feel of a native mobile app.

Best of all, the jQuery Mobile components work just like the jQuery UI widgets I talk about in Book 4, Chapter 3, so you already know how to use them. Now all you have to do is incorporate jQuery Mobile into your app.

Adding jQuery Mobile to your web app

jQuery Mobile consists of two files:


REMEMBER

- A JavaScript (`.js`) file that you add to your page by using a `<script>` tag with a reference to the external script file.
- A CSS (`.css`) file that you add to your page by using a `<link rel="stylesheet">` tag with a reference to the external CSS file.

How do you get these files? You have three ways to go about it:


REMEMBER

- **Download the files and use the default jQuery Mobile styles.** In this case, surf to `jquerymobile.com/download` and click the ZIP File link. The file you get will have a name like `jquery.mobile-1.x.y.zip`, where *x* and *y* denote the current version. Decompress the ZIP file and then copy the `jquery.mobile-1.x.y.min.js` and `jquery.mobile-1.x.y.min.css` files to your mobile web app's JavaScript and CSS folders. Then set up your `<link>` and `<script>` tags to reference the files:

```
<link rel="stylesheet" href="/css/jquery.mobile-1.x.y.min.
  css">
<script src="/js/jquery.mobile-1.x.y.min.js"></script>
```

  (Remember to replace *x* and *y* with the actual version numbers of your downloaded file.)

- **Use custom jQuery Mobile styles.** In this case, surf to `themeroller.jquerymobile.com` and use the ThemeRoller app to set your custom colors, fonts, and other styles. Click the Download Theme ZIP File button, type a theme name, and then click Download ZIP. Decompress the downloaded ZIP file, copy the CSS file from the Themes folder, and then add it to your mobile web app's CSS folder. Then set up your `<link>` tag to reference the file:

```
<link rel="stylesheet" href="/css/custom.min.css">
```

Replace *custom* with the custom theme name you provided. Note that this only gives you the CSS for jQuery Mobile. You still need to download the jQuery Mobile JavaScript file, as I describe previously.

- **Link to a remote version of the file.** Several content delivery networks (CDNs) store the jQuery Mobile files and let you link to them. Here are the tags to use for Google's CDN:

```
<link rel="stylesheet" href="https://ajax.googleapis.com/
  ajax/libs/jquerymobile/1.x.y/jquery.mobile.css">
<script src="https://ajax.googleapis.com/ajax/libs/
  jquerymobile/1.x.y/jquery.mobile.min.js"></script>
```

Again, in both cases, be sure to replace *x* and *y* with the actual version numbers of the latest version of jQuery Mobile.

I hold off talking about specific jQuery Mobile widgets until Book 8, Chapter 2, where I build a mobile web app using a number of jQuery Mobile components.

WARNING

You also need to add jQuery to your page, as I describe in Book 4, Chapter 1. However, as I write this, jQuery Mobile is only compatible with version 2 of jQuery, so be sure to link to that version, not version 3.

Working with Images in a Mobile App

When planning a mobile web app, you always need to consider the impact of images, both on your design and on your users.

Making images responsive

On the design side, you need to ensure that your images scale responsively, depending on the screen width or height. For example, if the user's screen is 1,024 pixels wide, an image that's 800 pixels wide will fit no problem, but that same image will overflow a 400-pixel-wide screen. As I mention in Book 7, Chapter 1, you create responsive images with the following CSS rule:

```
image {
    max-width: 100%;
    height: auto;
}
```

REMEMBER

Here, *image* is a selector that references the image or images you want to be responsive. Setting `max-width: 100%` enables the image width to scale smaller or larger as the screen (or the image's container) changes size, but also mandates that the image can't scale larger than its original width. Setting `height: auto` cajoles the browser into maintaining the image's original aspect ratio by calculating the height automatically based on the image's current width.

TIP

Occasionally, you'll want the image height to be responsive instead of its width. To do that, you use the following variation on the preceding rule:

```
image {
    max-height: 100%;
    width: auto;
}
```

Delivering images responsively

On the user side, delivering images that are far larger than the screen size can be a major problem. Sure, you can make the images responsive, but you're still sending a massive file down the tubes, which won't be appreciated by those mobile surfers using slow connections with limited data plans.

Instead, you need to deliver to the user a version of the image file that's appropriately sized for the device screen. For example, you might deliver the full-size image to desktop users, a medium-sized version to tablet folk, and a small-sized version to smartphone users. That sounds like a complex bit of business, but HTML5 lets you handle everything from the comfort of the `<img>` tag. The secret? The `sizes` and `srcset` attributes.

REMEMBER

The `sizes` attribute is a collection of *expression-width* pairs:

- The *expression* part specifies a screen feature, such as a minimum or maximum width, surrounded by parentheses.
- The *width* part specifies how wide you want the image displayed on screens that match the expression.

For example, to specify that on screens up to 600 pixels wide you want an image displayed with a width of `90vw`, you'd use the following expression-width pair:

```
(max-width: 600px) 90vw
```

A typical `sizes` attribute is a collection of expression-width pairs, separated by commas. Here's the general syntax to use:

```
sizes="(expression1) width1,
       (expression2) width2,
       etc.,
       widthN"
```

Notice that the last item doesn't specify an expression. This tells the web browser that the specified width applies to any screen that doesn't match any of the expressions.

Here's an example:

```
sizes="(max-width: 600px) 90vw,
       (max-width: 1000px) 60vw,
       30vw"
```

REMEMBER

The `srcset` attribute is a comma-separated list of image file locations, each followed by the image width and letter `w`. Here's the general syntax:

```
srcset="location1 width1w,
        location2 width2w,
        etc.">
```

This gives the browser a choice of image sizes, and it picks the best one based on the current device screen dimensions and the preferred widths you specify in the `sizes` attribute. Here's a full example, and Figure 1-2 shows how the browser serves up different images for different screen sizes:

```
<img src="/images/img-small.jpg"
     sizes="(max-width: 600px) 90vw,
            (max-width: 1000px) 60vw,
            30vw"
     srcset="/images/img-small.png 450w,
             /images/img-medium.png 900w,
             /images/img-large.png 1350w">
```

WARNING

The `sizes` and `srcset` attributes don't always work the way you might expect. For example, if the browser finds that, say, the large version of the image is already stored in its cache, then it will usually decide that it's faster and easier on the bandwidth to just grab the image from the cache and scale it, instead of going back to the server to download a more appropriately sized file for the current screen.

FIGURE 1-2: With the `<img>` tag's `sizes` and `srcset` attributes on the job, the browser serves up different versions of the image for different screen sizes.

Storing User Data in the Browser

I spend big chunks of this book talking about using MySQL to store data on the server, using PHP to access that data, and using JavaScript/jQuery Ajax calls to transfer data between the browser and the server. It's a robust and time-tested technique, but no sane person would describe it as trivial.

What's a web developer to do, then, when she wants to save just a few small or temporary tidbits of data for the current user? For example, perhaps her mobile web app enables each user to set custom background and text colors. That's just two pieces of data, so setting up the MySQL–PHP–Ajax edifice to store that data would be like building the Taj Mahal to store a few towels.

Fortunately, our developer doesn't have to embark on a major construction job to save small amounts of data for each user. Instead, she can take advantage of a technology called *web storage* that enables her to store data for each user right in that person's web browser.

Understanding web storage

Web storage is possible via an HTML5 technology called the Web Storage API (application programming interface), which defines two properties of the `window` object:

- `localStorage`: A storage space created within the web browser for your domain (meaning that only your local code can access this storage). Data within this storage can't be larger than 5MB. This data resides permanently in the browser until you delete it.
- `sessionStorage`: The same as `localStorage`, except that the data persists only for the current browser session. That is, the data is erased when the user closes the browser window.

WARNING

It's also possible for users to delete web storage data by using their browser's command for removing website data. If your mobile web app really needs its user data to be permanent (or, at least, completely under your control), then you need to store it on the server.

Both `localStorage` and `sessionStorage` do double-duty as objects that implement several methods that your code can use to add, retrieve, and delete user data. Each data item is stored as a key-value pair.

Adding data to storage

REMEMBER

You add data to web storage using the `setItem()` method:

```
localStorage.setItem(key, value)
sessionStorage.setItem(key, value)
```

- *`key`:* A string that specifies the key for the web storage item.
- *`value`:* The value associated with the web storage key. The value can be a string, number, Boolean, or object. Note, however, that web storage can only store strings, so any value you specify will be converted to a string when it's stored.

Here's an example:

```
localStorage.setItem('bgcolor', '#ba55d3');
```

It's common to store a collection of related key-value pairs as a JSON string. For example, suppose you collect your data into a JavaScript object:

```
var userData = {
    bgcolor: "#ba55d3'",
    fgcolor: "#f8f8f8",
    subscriber: true,
    level: 3
}
```

Before you can add such an object to web storage, you have to *stringify* it — that is, turn it into a JSON string — using the `JSON.stringify()` method:

```
localStorage.setItem('user-settings', JSON.stringify(userData));
```

REMEMBER

When you store user data using web storage, that data is only available to the user in the same web browser running on the same device. For example, if you save data for a user running, say, Safari on an iPhone, when he returns to your site using, say, Chrome on a desktop computer, that data will not be available to him. Therefore, a good reason for going to the trouble to store user data on the server is that this makes the data available no matter what browser or device the user brings to your site.

Getting data from web storage

To retrieve an item from web storage, use the `getItem()` method:

```
localStorage.getItem(key)
sessionStorage.getItem(key)
```

» *key:* A string that specifies the key for the storage item

Here's an example:

```
var userBG = localStorage.getItem('bgcolor');
```

If you stored a JavaScript object as a JSON string, use `JSON.parse()` to restore the object:

```
var userData = JSON.parse(localStorage.getItem('user-
  settings'));
```

Removing data from web storage

If you no longer require data in web storage, use the `removeItem()` method to delete it:

```
localStorage.removeItem(key)
sessionStorage.removeItem(key)
```

» *key:* A string that specifies the key for the storage item

Here's an example:

```
localStorage.removeItem('bgcolor');
```

If you want to start fresh and delete everything from web storage, use the `clear()` method:

```
localStorage.clear();
sessionStorage.clear();
```