How-To Build HTML5 Applications - Using HTML5 to Create Mobile Experiences

Last month, I introduced you to CSS3 media queries (msdn.microsoft.com/magazine/hh882445), a new module that allows you to adapt page styles based on conditional rules. Though media queries have broad applicability across the device spectrum, they're often mentioned in the context of building mobile sites and applications. To wit, the introduction to media queries in the previous article was framed around the creation of tablet and mobile experiences.

Considering the difficulties building mobile sites and apps presents, it's no wonder that media queries took off. Compared with undesirable alternatives such as browser sniffing (sometimes called device detection) and having to create mobile experiences on a per-platform basis, media queries seem like a true gift. They're brilliant modules, for sure, and the reason I wrote about them last month is because you should be using them today.

Responsive Web Design, Revisited

But there's more to the story: CSS media queries are great, but they're only a piece of what you really need to create great mobile Web experiences. Last month, I mentioned the term responsive Web design, a term coined by Ethan Marcotte in his seminal article of the same name (bit.ly/9AMjxh). Marcotte focuses on media queries, but he also points out two other necessary practices: fluid grids and flexible images. Media queries are the engines that drive responsive, adaptive sites, but they're only effective when site design is also responsive and adaptive. This month, I'll introduce you to some ideas around these other two pillars of responsive Web design. I'll start with an overview of some up-and-coming CSS layout modules, and then discuss some techniques for making non-textual elements such as images and embedded video adaptive as well. Along the way, I'll note some frameworks and libraries that help you adopt these techniques. I'll also mention some popular frameworks for creating mobile Web applications, and wrap up with a brief discussion on using HTML5 to build "native" applications. By the time you've finished this article, you should have a solid foundation for implementing responsive Web design in your own applications.

Fluid Grids and Layouts

Using a grid for typographic design is a practice that has been around in some form or another for centuries, predating even the invention of moveable type. That two-dimensional structure made up of intersecting vertical and horizontal axes allows a designer to align and organize elements in a visually pleasing way on a layout, as illustrated in **Figure 1**. Over the last few years, Web designers have begun to apply many of the same principles to their digital work, and a number of popular frameworks, such as the 960 Grid System (960.gs) and the Semantic Grid System (semantic.gs), now make grid layouts accessible to all.

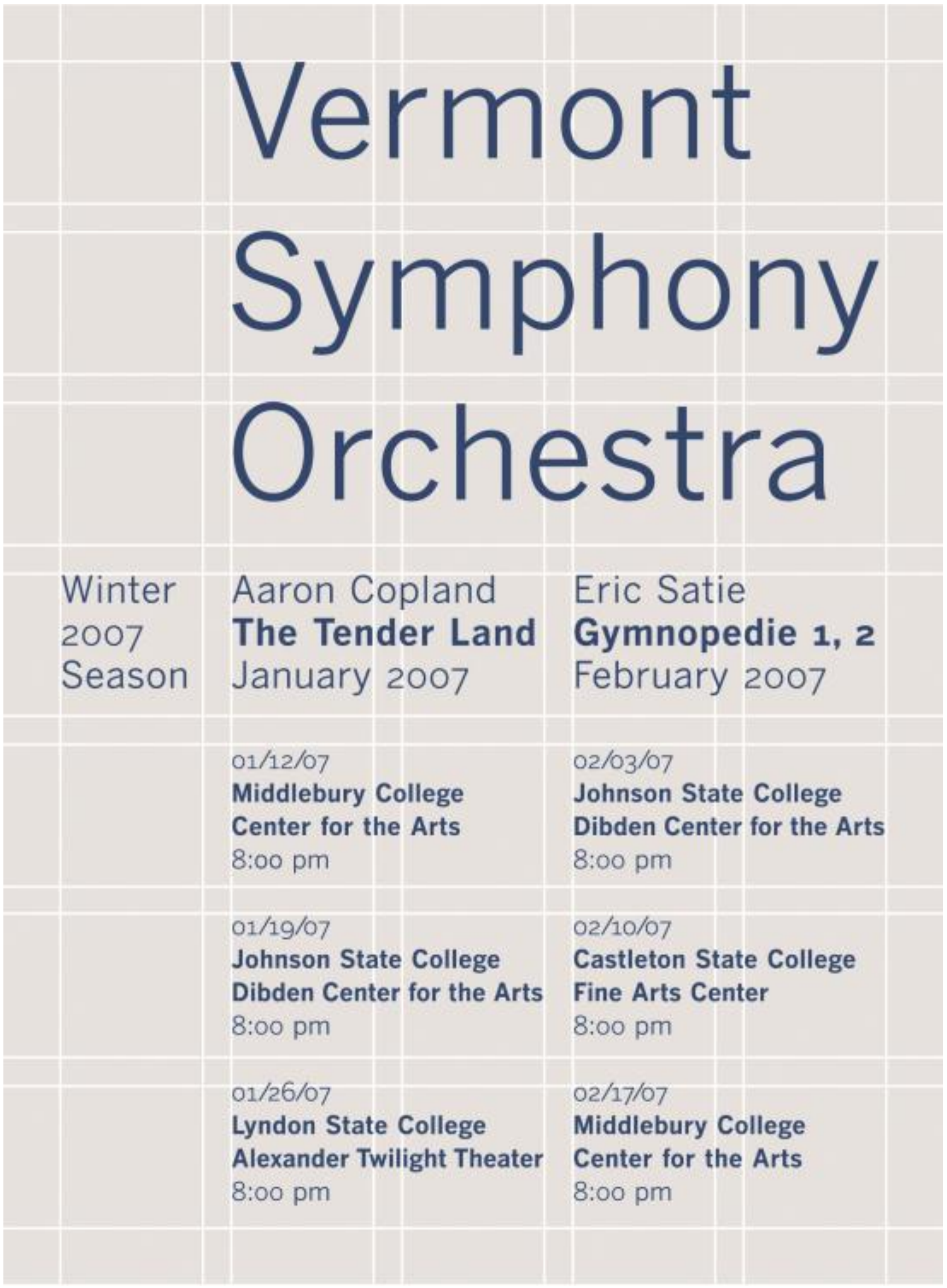

Figure 1 A Typographic Grid

However, the direct application of a typographic grid to the Web has a key flaw: print layouts are fixed, Web layouts are not. What's more, many grid implementations aren't terribly semantic, meaning that they require the addition of markup to define the grid, mixing presentation with content in your HTML pages.

Which brings us to the "fluid" in Marcotte's "fluid grid." To be truly responsive, grids (or layouts) should adapt to changing experiences. As I discussed last month, media queries help you define the rules for repositioning elements, but in order to be effective, those elements must first be defined in a fluid or flexible container. The tools I mentioned earlier (as well as many others) do address this issue either natively (the Semantic Grid) or through the use of a complementary library (Adapt.js for the 960 Grid—adapt.960.gs), but there are also new and emerging CSS modules that assist in the creation of fluid layouts.

Note that I'm taking some liberties with Marcotte's term fluid grids by restating it as fluid layouts. I'm doing so because some of the new CSS modules, though not based on a grid, can still help you create fluid, adaptable containers.

Let's look first at the CSS3 Flexible Box Layout Module (or Flexbox), which can be found at bit.ly/yguOHU. Flexbox is designed to help you create layout containers for elements that will automatically position children in a horizontal or vertical flow, and provide automated spacing (Goodbye "ul li { float: right; }"!). This module is supported—with vendor prefixes—in the Internet Explorer 10 Platform Preview, Firefox, Chrome, Safari, iOS Safari and Android (check out caniuse.com/flexbox for more information).

We'll start by applying Flexbox to the Photo Gallery site I introduced last month. Given the CSS in **Figure 2**, you can see the result, styled up a bit for illustration purposes, in **Figure 3**. Please note that the CSS in **Figure 2** is using only the "-ms-" vendor prefix. In the sample code available online (msdn.com/magazine/msdnmag0512), I include the other vendor prefixes (-webkit, -moz, -o), and you should do this for your sites as well.

Figure 2 CSS for Using the Flexbox Module

XMLCopy
```
ul.thumbnails {
  width: 100%;
  background: #ababab;
  display: -ms-box;
  -ms-box-orient: horizontal;
  -ms-box-pack: center;
  -ms-box-align: center;
}
ul.thumbnails li {
  -ms-box-flex: 1;
}
```

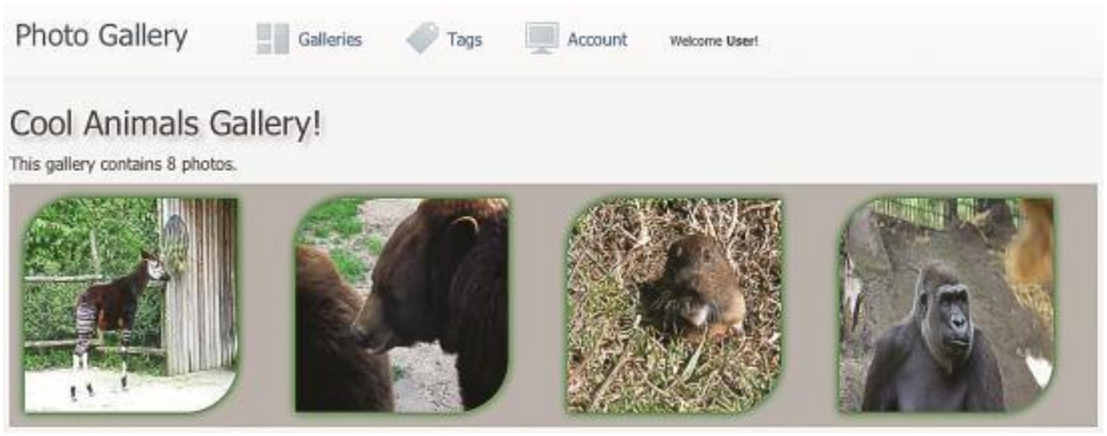

Figure 3 Photo Gallery Images with Flexbox Applied

This is nice, but of course it looks like what we already had. To illustrate the flex in Flexbox, resize your browser window, or open the page in a mobile emulator or on a device. There are no media queries defined in this example and yet the layout is acting a bit more fluid. Combine the two modules and you'll be able to create fluid containers that align, and space and shift elements in a responsive manner. For instance, you can create a media query rule for screens smaller than 480 pixels, change box-orient to vertical and voilà—you have the beginning of a mobile layout.

The CSS Grid Layout (or just CSS Grid), which can be found at bit.ly/ylx7Gq, is a newer specification, submitted to the World Wide Web Consortium (W3C) CSS Working Group in April 2011 and currently implemented only in the Internet Explorer 10 Consumer Preview. The idea is to provide robust grid support natively within the browser. For developers and designers, the result is a rich typographic grid without the need for table layouts or the presence of semantically neutral markup.

The CSS Grid enables you to define a page layout with predefined rows and columns, as well as rules that specify how content flows into and across those elements. The first step is to define a grid container, specifying "grid" as the display property for selected elements:

XMLCopy
```
body {
  display: -ms-grid; // A vendor prefix is required.
}
```

I'm selecting the body element here, which means my grid will fill the entire document. This isn't required, and I could easily craft a grid from a smaller section of the page or even define multiple grids on a single page. Once I've defined the grid, I need to define the size of its rows and columns:

XMLCopy
```
body {
  display: -ms-grid;
  -ms-grid-rows: 50px 30% 20% auto;
  -ms-grid-columns: 75px 25px 2fr 1fr;
}
```

Here, I'm specifying a grid of four columns and four rows, making the size absolute in a few cases (50px, 75px, for example), relative to the size of the window for a few (30%, 20%) and automatic based on the width of its content (auto). I'm also using the new fraction value unit, fr, which is defined in the CSS Grid specification as "… a fraction of the available space." Fraction values are calculated after fixed sizes are assigned and then divided proportionately among all rows and columns defined with these values. In the context of my example, this means that once 100 pixels are set aside for columns one and two, column three will be given two-thirds of the remaining space and column four will be given one-third.

With the grid defined, it's easy to position child elements within the grid by assigning row and column values, like so:

XMLCopy
```
#main {
  -ms-grid-row: 2;
  -ms-grid-column: 2;
  -ms-grid-row-span: 2;
  -ms-grid-row-align: center;
}
```

Here, I'm placing my "main" sectioning element at the second row and column of the grid. I'm allowing that element to span two rows and am centering the content inside the container. The Microsoft Internet Explorer Test Drive Demo site uses its implementation of CSS Grid Layout to create an exact implementation of the popular Grid System site, thegridsystem.org, and you can check it out for yourself at bit.ly/gEkZkE.

If you've ever tried something similar with markup and CSS2.1, you've no doubt seen the flexibility that the CSS Grid can provide. What's more, when combined with media queries, the CSS Grid can be used to create adaptive layouts that are easy to tweak with fewer rule changes as users adjust device size and orientation.

The final layout specification I'll present is the CSS Multi-Column Layout Module, which you'll find at bit.ly/yYEdts. The CSS Multi-Column is at the "Candidate Recommendation" state (bit.ly/x5IbJv), and enjoys wide support across all of the browsers, including planned support for Internet Explorer 10. As it sounds, Multi-Column allows you to lay out columns on a page without manual positioning or floats. All you need is to apply the "column-width" property (with prefixes, where required) on a container, like so:

XMLCopy
```
article {
  width: 960px;
  column-width: 240px;
}
```

With this rule, article elements will be split into columns of 240 pixels, creating as many columns as the container allows (in this case, it takes four 240-pixel columns to fill the 960-pixel container). I could also use the column-count property to define a fixed number of columns, in which case the column widths would be evenly distributed within the width of my container.

As with the Flexbox and Grid modules, combining the module with media queries lets you quickly define simple, adaptable rules to deliver an ideal user experience to mobile users.

The three modules I've described have a lot in common, and each has features you can use to create the kind of fluid layouts that are a requirement for truly responsive Web sites. I encourage you to research and play with each so that you can choose the appropriate module when solving a given layout challenge.

You'll also want to check out the emerging frameworks that leverage these specifications. Using one of these can be a nice jump-start to building fluid layouts for your own sites. Two notable frameworks are Skeleton (getSkeleton.com) and Bootstrap (getbootstrap.com), a full site starter kit from Twitter. I recently rebuilt one of my own sites with the help of Skeleton (check it out at html5tx.com).

Responsive Media

In the realm of responsive Web design—mobile in particular—media is the sticky wicket. Let's start with images. One of the easiest ways to style images to make them more responsive is to add the following to your stylesheets:

XMLCopy
```
img {
    max-width: 100%;
  }
```

This rule will always scale your images (up or down) to fit within that image's parent container. Thus, if your container elements are responsive (perhaps using one of the techniques previously described), your images are as well.

The challenge with this method is that the images on your site need to be large enough to scale to any size that they might conceivably need to be. In the Photo Gallery site I've been using, the raw images are quite large, and thus can handle being resized.

The use of huge, resizable images, however, leads to a huge problem for mobile: overhead, and thus to potentially poor mobile experiences. Even if you're resizing a large image to fit a 320 x 240 window, the full image is transferred to the device. This means you could potentially be sending a 2MB photo when a 10KB photo is all the device requires. In a mobile world, bandwidth still matters, so the device-width strategy must be supplanted with other approaches.

Unfortunately, this is still a challenge, and the W3C doesn't yet formally support a particular approach. A number of methods exist for dealing with responsive images, but all fall into one of two categories: they either deal with the problem on the server, or attempt to do so on the client. Many server approaches, like the one described at bit.ly/rQG0Kw, rely on the use of 1 x 1 pixel placeholder images, client cookies and server-rewrite rules in order to deliver the right image to the right device. Client approaches, such as the one described at bit.ly/tIdNYh, often utilize JavaScript, <noscript> fallbacks, conditional comments and some interesting tricks in CSS. Both approaches feel like hacks (because they are), but they represent the best we've been able to come up with given the constraints of the <img> tag. In the short term, you'd be wise to take a look at both approaches and decide which works for your applications.

Over the long term, however, there might be hope. In an article in Smashing Magazine, "HTML5 Semantics" (bit.ly/rRZ5Bl), Bruce Lawson of Opera argues for the addition of a <picture> element that would behave similarly to the <audio> and <video> tags, meaning that developers would have access to multiple <source> child elements inside of the parent <picture>. When combined with inline media queries, a new <picture> element could provide a nice way to finally deliver a robust solution for responsive images:

XMLCopy
```xml
<picture alt="cat gallery">
  <source src="nyan-high.png" media="min-width:800px" />
  <source src="nyan-med.png" media="min-width:480px" />
  <source src="nyan-low.png" />
  <!-- fallback for unsupporting browsers -->
  <img src="nyan-med.png" alt="cat gallery" />
</picture>
```

Though this solution has proven to be popular, and a W3C Community Group has been formed around it (bit.ly/AEjoNt), there's no formal working group that I'm aware of. Perhaps we'll see this element make it in time for HTML6.

Similar challenges exist for adding responsive video to sites and applications, though more robust solutions exist in HTML5 for video than for images. For starters, the media-query-enhanced <source> element—as illustrated with the previously mentioned fictional <picture> element—is valid for <video>, as illustrated here:

XMLCopy
```
<video>
  <source src="nyan-mashup-high.webm" media="min-width:800px" />
  <source src="nyan-mashup-med.webm" media="min-width:480px" />
  <source src="nyan-mashup-low.webm" />
  <!-- Insert Silverlight or Flash Fallback here -->
</video>
```

If you're serving the video from your own servers or are using a service that provides multiple versions to embed, I highly recommend using this syntax to ensure that your users get a device-friendly video.

While this solution will help save your users' bandwidth, you still need to think about sizing those embedded video elements, just as you do with images. With media queries, it's easy to adapt your video elements based on different screen sizes, but if you're looking for a solution that's a bit more automated, take a look at FitVids.js (fitvidsjs.com), a jQuery plug-in that will automatically make your video elements fluid and responsive. Keep in mind, however, that as a jQuery plug-in, this solution won't work for users with JavaScript disabled.

Building Mobile Web Apps with HTML5 Frameworks

Now that we've covered the other two pillars of responsive Web design, fluid layouts and flexible images, let's talk a bit about cases where you're not just building mobile-friendly Web sites or apps—you want to specifically target the mobile experience.

In the world of development, traditional desktop (or client) and Web paradigms have given way to a third type of application, often called native applications because they're resident on a given device (a Windows Phone-based smartphone or iPad, for example), are developed using device-specific frameworks (iOS and Android) and are installed via an App Store or Marketplace.

As rich and robust as these frameworks are, sometimes Web developers wish to provide a similar "native feel" to their mobile Web applications. Such applications still reside on your servers and can be delivered outside of a device App Store or Marketplace.

Though you can certainly build these types of applications by hand, it's common to use frameworks to do so. One popular choice for mobile Web applications is jQuery Mobile (jquerymobile.com), a mobile development framework that provides an HTML5-based UI system for targeting nearly every mobile platform. **Figure 4** shows an example of the mobile application for OpenTable.com, which was built using jQuery UI.

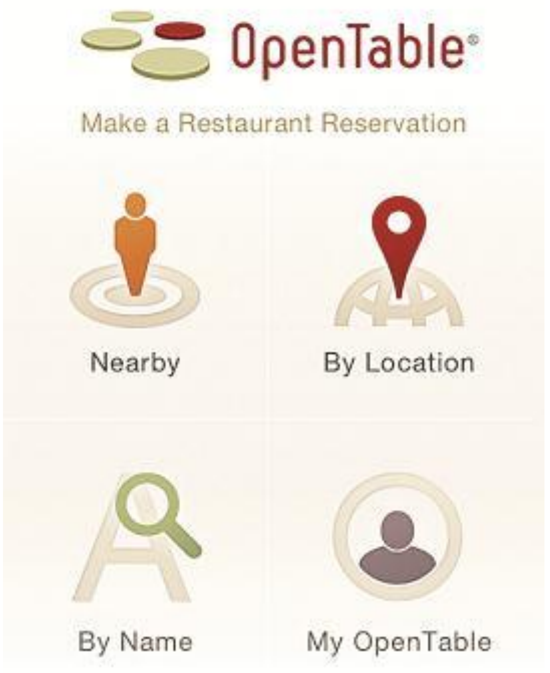

Figure 4 A Sample Mobile Screen Built with jQuery Mobile

Another popular option for building mobile applications with a native look and feel is Kendo UI Mobile (kendoui.com), an HTML5, JavaScript and CSS framework from Telerik Inc. Kendo UI allows you to create mobile applications that look fully native on iOS- and Android-based devices—and to do so with a single codebase. **Figure 5** and **Figure 6** show this feature in action, which you can check out for yourself at bit.ly/wBgFBj.

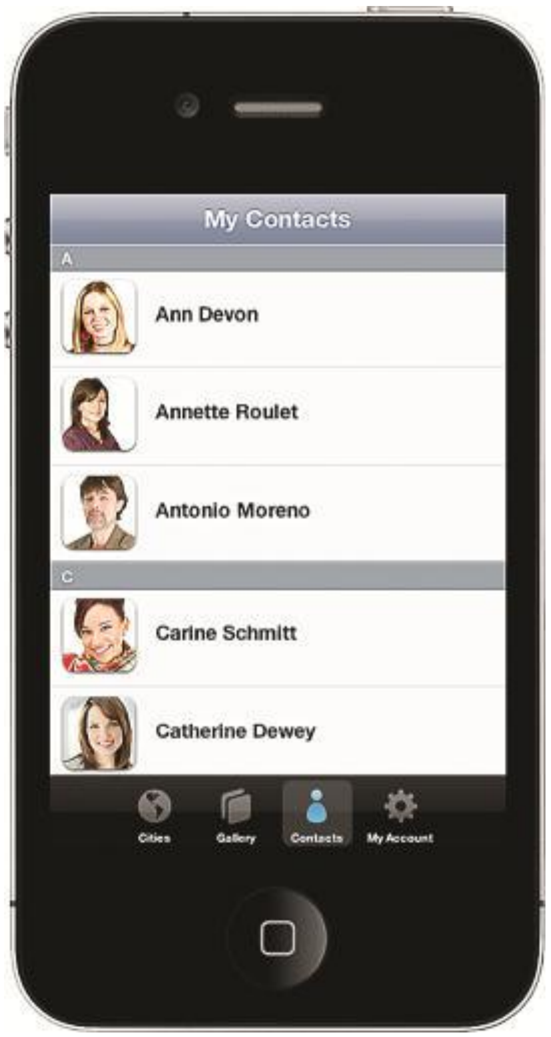

Figure 5 A Kendo UI Mobile Demo Application Viewed on an iOS-Based Device

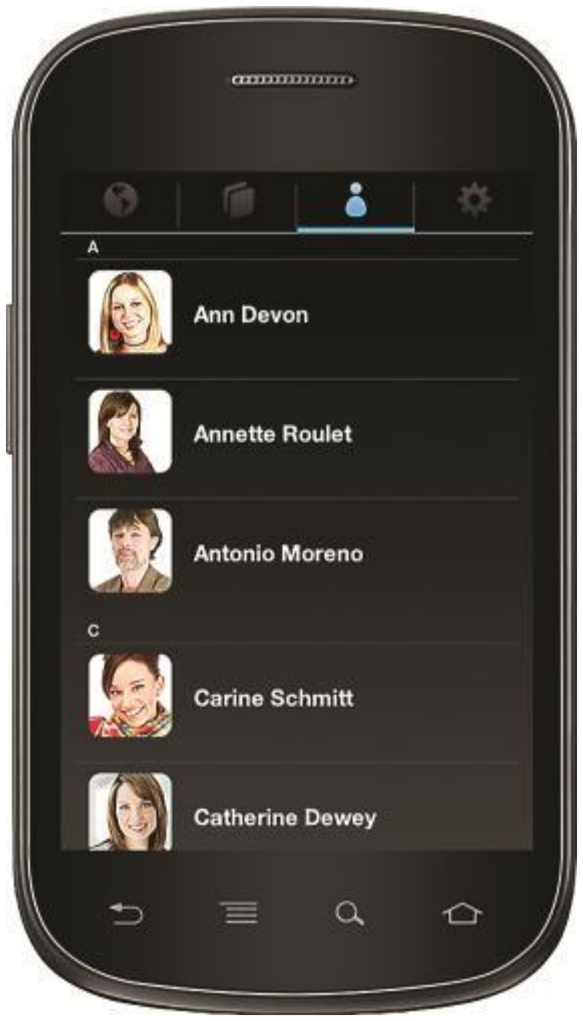

Figure 6 A Sample Kendo UI Mobile Application Viewed on an Android-Based Device

Building Native Applications with HTML5

Providing a native feel to Web applications is a great way to not only leverage your skills as a Web developer, but also to create applications that conform to a user's expectations in a mobile setting. Still, these applications can only go so far in leveraging native sensors and APIs on those devices. While some features—like geolocation—are provided to mobile browsers, many—like the accelerometer or video—are not. In order to access these features, native applications are still the way to go.

The great news, however, is that the popularity of Web programming has enabled HTML5, JavaScript and CSS to "go native," as it were. Popular frameworks such as PhoneGap (phonegap.com) and Titanium Appcelerator (appcelerator.com) enable you to use HTML5 and JavaScript to build native applications for iOS, Android and Windows Phone, with device API access to boot. What's more, mobile development frameworks like jQuery Mobile and Kendo UI Mobile work just as well in these environments as they do in the browser. **Figure 7** shows a native iOS application built using PhoneGap and Kendo UI. For more information, check out the blog post at blogs.telerik.com/kendoui/posts/12-02-23/building_your_first_kendo_ui_mobile_phonegap_application.

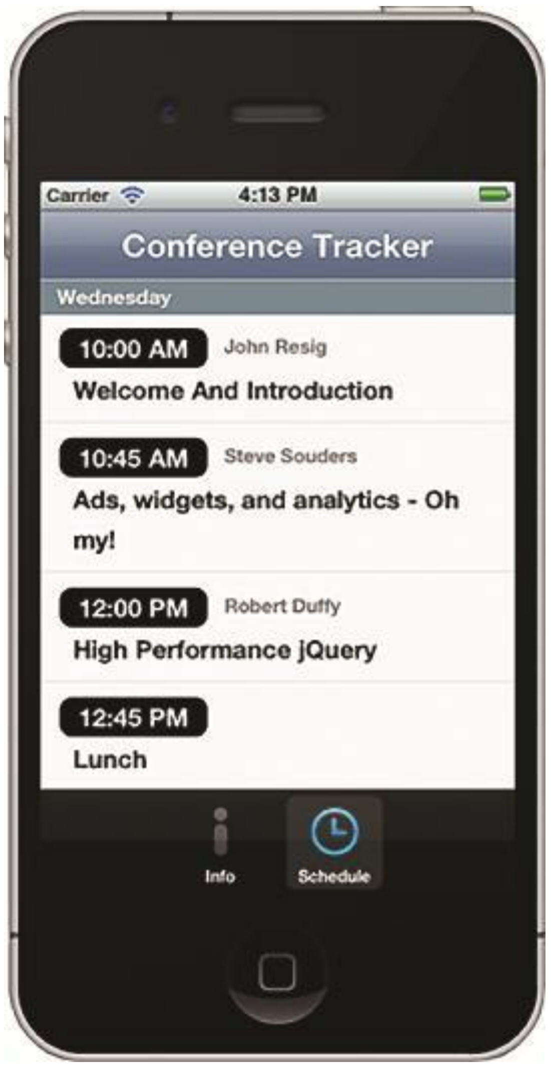

Figure 7 An iOS Application Built with HTML, JavaScript and CSS

Microsoft has taken native Web development to a new level by formally adding support for building Windows 8 applications using HTML5, JavaScript and CSS, all with no additional abstractions or frameworks required. You can check out the Consumer Preview of Windows 8, as well as the new developer tools for these platforms, at dev.windows.com.

When it comes to the mobile Web, you've got choices! If you're a Web programmer who wants to build native experiences, complete with augmented reality features, check out Windows 8 or a framework such as PhoneGap or Titanium Appcelerator. If you're just looking for a native feel in the browser, look at jQuery UI and Kendo UI Mobile. Finally, if your goal is to create a single Web site or application that responds to many devices and experiences, try the responsive strategies I discussed in this and last month's article. There's no question that mobile is one of the hottest platforms for development right now. Your best bet, no matter which of the strategies or platforms you choose, is to make mobile your top development priority.

Courtesy: https://docs.microsoft.com/en-us/archive/msdn-magazine/2012/may/building-html5-applications-using-html5-to-create-mobile-experiences

Modified: 2021.10.06.8.10.PM

Dököll Solutions, Inc