BYU-Idaho Responsive Framework Readme

Jakob Anderson
Updated December 17th, 2012

This readme file will outline the techniques and pieces used within the BYU-Idaho Responsive Framework, as well as instructions on how to implement and adapt it to your needs.

Table of Contents

Priority 1 criteria

The Priority 1 criteria of this responsive framework:

- Highly-usable and consistent user experience, university branding & identity in all major devices and browsers
- Fast-loading for all major devices and browsers
- Very light server load and asset request total
- Flexible to be used in all web apps that have strong multi-screen/device use cases
- Can toggle responsive on and off from a single point

Toggling “responsive” mode

To toggle the “responsive” mode on and off from one point in the code is very simple. The responsive CSS code inherits only from a “.responsive” class, which should be placed on the <body> element of the html if responsive behavior is desired, and removed if it is not wanted.

Responsive on:
```
<body class="responsive">
```

Responsive off:
```
<body class="">
```

Writing your app/site’s custom responsive code

Specific instructions for the Responsive Framework

There is a prepared bootstrap responsive CSS framework in the “css/style.css” file within the BYU-Idaho Responsive Framework. It has been prepared with the same media query transition points as the responsive framework, so the code you put in here will behave consistently with the global framework, as long as you stay within the rules set by it.

Put all default static CSS code in the top of the “style.css” document where it indicates. These rules will apply to all browsers first, including those browsers and devices that do not recognize the CSS3 media queries.

Put all general responsive code in the area right after it, inheriting underneath the “.responsive” class, so that it will only be applied to pages that have “opted-in” by using the “responsive” class on the <body> element.

Underneath that, place resolution-specific responsive view code inside the corresponding media queries. NOTE: any CSS rule you specify in one resolution view, you must also specify and override in all of the other media queries, otherwise your rule will activate in one, but not deactivate for the others. To keep this responsive code as DRY as possible, it is very helpful to write all general responsive and static rules above as indicated, and then selectively override as small a number of things as possible inside the media queries.

If the code inside the media queries gets too large, it will incur user-perceived performance costs for every rule overridden and every reflow to the page it causes. Keeping the layout simple will save you much grief and complication in the end.

Learn by doing

Learn how to do Responsive Web Dev/Design by doing. You will quickly find out what works and doesn’t in responsive web by testing given techniques out on multiple screen sizes as you build. Test accurately for your given users and test often. Find out what the most commonly used devices are and build for standards first, then small exceptions for the rest.

Use resources

Here are a few resources that helped me with Responsive Web:

CSS-Tricks

http://css-tricks.com/video-screencasts/102-braindump-on-responsive-web-design/

Smashing-Magazine Mobile Development
http://uxdesign.smashingmagazine.com/2012/01/17/designing-well-tempered-web/
http://mobile.smashingmagazine.com/2011/07/22/responsive-web-design-techniques-tools-and-design-strategies/
http://www.slideshare.net/vitalyfriedman/responsive-web-design-clever-tips-and-techniques#btnNext
http://mobile.smashingmagazine.com/

A List Apart
Ethan Marcotte: "Responsive Web Design" (Ethan actually coined this phrase)
http://www.alistapart.com/articles/responsive-web-design/
"Mobile First" Book
http://www.abookapart.com/products/mobile-first

Browser/device compatibility techniques

Browser/Device Compatibility Baseline

Our current browser/device compatibility baseline includes all browsers and devices that show a serious percentage of our user-base, as found by evidence in the university's web analytics. We quantify this as being > 1% monthly unique page loads. These device and browser statistics for campus site usage are changing very fast, so stay in touch with Web Services for the newest web analytics data. The browsers and devices that fit this profile as of Dec 2012 are the following:

***Full Support**

- **Chrome 4+**
- **Firefox 4+**
- **Internet Explorer 9+**

- **Safari 5+**

****Basic Support**

- **Internet Explorer 7+**
- **Firefox 3.6+**

Device Support

- **iOS 5/6 Safari Browser (tablet and phone sizes)**
- **Android 2.3-4.1 Browser (tablet and mobile sizes)**

Optional, but close to 1% and growing fast

- **Mobile Firefox**
- **Mobile Chrome**
- **Desktop Opera 11+ (very small user percentage)**

***Accepts valid HTML5 spec code, may use progressive enhancements with proper fallback-handling techniques**
****Accepts legacy-spec code, use "graceful degradation" techniques and "polyfills" to ensure consistent behavior with HTML5 and CSS3 techniques**

Browser-conditional classing, as per the HTML5 Boilerplate:

Looks like this:

```
<!--[if IE 9]>         <html class="no-js lt-ie10"> <![endif]-->
```

This method, when used at the top of an html document allows you to only check the browser version once, and to assign classes to the html element based on the condition, so any browser exceptions may be handled by javascript and css, rather than making another request for the browser's user agent string.

See Paul Irish's explanation:
http://paulirish.com/2008/conditional-stylesheets-vs-css-hacks-answer-neither/

Graceful degradation of HTML elements via Modernizr:

Modernizr contains detectors for modern web features, as well as polyfills that give old browsers many of the new features. The most important of these polyfills is the HTML5 shim, which tells old browsers how to handle basic HTML5 elements (such as <nav>) in the DOM.

See:

http://modernizr.com/
https://github.com/h5bp/html5-boilerplate/blob/v4.0.2/doc/html.md#modernizr

Feature-detection classing for Progressive Enhancement and fallbacks via Modernizr:

Modernizr performs feature detection on each browser the page is loaded in, then adds classes for those features to the <html> element. You may then build your css and javascript accordingly to be predictive to these detections. The “no-js”, “rgba”, “borderradius”, and “cssgradients” detection classes can be used quite often to manage progressive enhancements and smarter fallbacks in your css and javascript code. The “no-js” class allows you to have fallbacks prepared in your code for the instances where javascript is not enabled, or not loaded by default. You can either give your page another method by which to load/display, or you can give the users actionable instructions or feedback.

See:
http://modernizr.com/
https://github.com/h5bp/html5-boilerplate/blob/v4.0.2/doc/html.md#the-no-js-class
https://github.com/h5bp/html5-boilerplate/blob/v4.0.2/doc/html.md#modernizr

HTTP-EQUIV

Looks like this:

```
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
```

This should cause any IE browser version to force the newest rendering engine it has. If loaded in IE10 with compatibility mode of IE7 document type, it should force the browser to use the IE10 document rendering mode. On another note, the “http-equiv” meta element has been found to obstruct IE8’s read-ahead parser, so if your IE7 compatibility mode traffic is < 1%, you should remove this meta element to see a modest speed improvement in IE8.

See: https://github.com/h5bp/html5-boilerplate/blob/v4.0.2/doc/html.md#x-ua-compatible

Chrome Frame

"The main content area of the framework includes a prompt to install Chrome Frame (which no longer requires admin rights for users of IE6). If you intended to support IE6, then you should remove the snippet of code." – HTML5 Boilerplate documentation

See: https://github.com/h5bp/html5-boilerplate/blob/v4.0.2/doc/html.md#google-chrome-frame

Mobile Device Features

Mobile Icons

Devices need touch icons to save page shortcuts to device desktops. These icons need to be much higher resolution than favicons, so an additional set will need to be used. The responsive framework includes a prepared set of standard BYU-Idaho-branded icons in the required higher resolutions and filenames. The Shortcut icons should be put in the root directory of the web app to be implicitly linked to the html page. Otherwise, they need to be linked to another location that houses them, using the following file names and the following code adapted to your icon path, including attributes for the icon sizes:

```
<link rel="apple-touch-icon-precomposed" sizes="144x144"
href="img/touch/apple-touch-icon-144x144-precomposed.png">

<link rel="apple-touch-icon-precomposed" sizes="114x114"
href="img/touch/apple-touch-icon-114x114-precomposed.png">

<link rel="apple-touch-icon-precomposed" sizes="72x72"
href="img/touch/apple-touch-icon-72x72-precomposed.png">

<link rel="apple-touch-icon-precomposed" href="img/touch/apple-touch-
icon-57x57-precomposed.png">

<link rel="shortcut icon" href="img/touch/apple-touch-icon.png">
```

See:
https://github.com/h5bp/html5-boilerplate/blob/v4.0.2/doc/html.md#favicons-and-touch-icons
https://github.com/h5bp/mobile-boilerplate/blob/master/index.html

Device scale & zoom

In mobile devices with different pixel scaling ratios, it is important to normalize how these device browsers scale and zoom. When loading a page in an iOS safari browser, it often zooms in when switching to landscape view. The following meta tag in the head should prevent that glitch.

```
<meta name="viewport" content="width=device-width,initial-scale=1">
```

See: https://github.com/h5bp/html5-boilerplate/blob/v4.0.2/doc/html.md#mobile-viewport

Performance Optimizations

Reducing Server round-trips

Every effort possible was made to reduce the number of server requests and round trips made by this framework. Each server round-trip is expensive in page load time, and especially server load and network traffic. This framework was optimized to reduce 2nd-load weight to 232 bytes, rather than 130-260Kb per load after the first caching load. It accomplishes that using several of the techniques described below.

App Cache

App Cache is a great way to offload persistent assets from the server to the local browser.

On a normal page, the browser will check the “modified” and “Expires” data on each asset used in the page.

With App Cache, the browser locally downloads the manifest and its assets on the first load. On the next load, the browser checks if the App Cache manifest file has changed, then downloads the new assets if the manifest has changed. If the manifest has not changed since last load, the browser uses the locally saved assets implicitly, rather than ever requesting them from the server. This results in substantially reduced load times, trip requests and far lower server and network load.

In this framework, the only things that should be requested each load from the server should be the html, if the custom css doesn't change. The only tricky bit with App Cache is that when assets are edited and changed, you need to change their filenames, as well as change their filename as listed in the App Cache manifest file, in order for the manifest to be aware of the change. If you don't, browsers will continue to use the old local version, instead of updating their local cache with the new remote version.

This is usually done using filename version naming schemes, such as banana-1.0.1.appcache.jpg. I just inserted app cache in the example name so that I will notice and remember it is being handled by the app cache and should be changing the filename on each change accordingly. While this naming scheme change is a bit tedious, only implement app cache after you have a solid and tested version of the code locked down, and app cache will make sure it is loaded extremely fast and server-light.

See:
http://www.html5rocks.com/en/tutorials/appcache/beginner/

Concatenation & Minification

Concatenation & Minification can help load times, user-perceived performance, and server load in big ways.

Concatenating all global framework CSS or JS into one file prevents additional server request trips, and makes all CSS or JS cached into one resource as well.

Minification removes whitespace in CSS and JS, and in JS, it can also reduce function and variable names to single or double-character names in deployed code. These make the resources much smaller to transfer, while keeping the App Cache below its maximum local cache 5MB limit (as of 2012, may change).

See:

- CodeKit [http://incident57.com/codekit/]
- Uglify.js [https://github.com/mishoo/UglifyJS]
- google closure compiler [https://developers.google.com/closure/compiler/%5D]

Data URI-encoding and embedding of interface images

Data URI-encoding is a great way to reduce server round-trip counts. It consists of embedding the image data directly into the CSS or HTML files, rather than referencing them as external and separate files. There are a couple of drawbacks though, so it is important to know where it is most appropriately used. It works best when images are rarely changed, not very large, and few in number. Once an image is embedded into the CSS or HTML, it will be cached and referenced as part of that document, rather than separately, so the images will cost 0 server round trips if done right.

On the responsive framework, we already fit this criteria, with one exception, which I fixed. The footer sprite was just barely too large to work with IE8's data URI max file size limit, so I remade the sprite, eliminating all unnecessary spacing and then was able to encode it.

I created a re-usable CSS class for each image data URI, so that it would only be embedded once, and referenced multiple times within a file where called, rather than embedded multiple times in each call. This eliminated all image server requests, while only adding a small overhead in file size to the documents they were embedded in. Since HTML data URI's have no known fallback for unsupported browsers, I turned all <img> elements into background-images of other supported elements to use the data URI technique.

See:
http://css-tricks.com/data-uris/

Google CDN for jQuery

"The Google CDN version of jQuery JavaScript library is referenced towards the bottom of the page using a protocol-independent path. A local fallback of jQuery is included for rare instances when the CDN version might not be available, and to facilitate offline development."
- HTML5 boilerplate documentation

The users may already have the same version of the Google CDN jQuery library in there browser cache from some other site, which means it will already be cached for you before the first load.

See:
https://github.com/h5bp/html5-boilerplate/blob/v4.0.2/doc/html.md#google-cdn-for-jquery

Google Analytics Tracking Code (Async loader version)

"An optimized version of the latest Google Analytics tracking code is included. If you place this script near the bottom of the page, it will not count users who didn't load the page completely."
- HTML5 Boilerplate documentation

The asynchronous load script version here will not block or delay the document.ready or window.onload events from triggering, unlike normal <script src=""> imports and scripts.

See:
https://github.com/h5bp/html5-boilerplate/blob/v4.0.2/doc/html.md#google-analytics-tracking-code

Load scripts after body content

If you are going to use javascript on your web app, you will have faster user-perceived performance if you load it after all the body content. Because javascript imports and script code are "blocking" (must request, receive, and execute before page load can continue below it), it is important to have all of your body content loaded already.

This will help prevent a "FOUC" (flash of unformatted code), or serious load-blocking delays to your users.

See:
http://developer.yahoo.com/performance/rules.html#js_bottom
http://paulirish.com/2009/avoiding-the-fouc-v3/

Keep components under 25K
iPhone is specified to not automatically cache components bigger than 25K.

See:
http://developer.yahoo.com/performance/rules.html#under25
http://yuiblog.com/blog/2008/02/06/iphone-cacheability/

Server-side performance optimizations:

Add expires headers
Specify the “freshness lifetime” of resources, in order to better leverage browser caching implicitly. These include the Expires, Cache-Control: max-age, Last-Modified, and ETag header information.

See:
https://developers.google.com/speed/docs/best-practices/caching

Compress components with gzip
Servers can be configured to automatically gzip components before transferring them to the client’s browser. This can result in big reductions in transfer time, size and server-load for all assets.

See:
http://developer.yahoo.com/performance/rules.html#gzip

Use CDN for static assets (images, js, css, etc.)
Using a content delivery network like Akamai gives all users around the globe access to your assets from a nearby server location, rather than routing all

the way to your server. Not only does it relieve most high-weight traffic to your servers, but it does so in a globally-scalable way. They manage the servers to cache aggressively in order to make things as fast as possible for the users and their servers. Having static assets on multiple domains also aids in the parallel-downloading to the user's browsers, since browsers usually have a max number of parallel connections to each given domain.

Use cookie-free domain for static assets

Static assets like images, js, and css will never have cookies, so you should avoid the complication and excess round-trips of cookie detection and handling when dealing with them. If you set up a cookie-free asset management subdomain, like assets.byui.edu, it will increase parallel-ization and reduce cookie detection overhead on these assets and their header requests.

See:
http://developer.yahoo.com/performance/rules.html#cookie_free

Additional References

Analyze

https://developers.google.com/chrome-developer-tools/
https://developers.google.com/speed/pagespeed/
http://yslow.org/
http://getfirebug.com/
http://msdn.microsoft.com/en-us/library/ie/gg589507(v=vs.85).aspx

Performance

http://html5boilerplate.com/
http://html5boilerplate.com/html5boilerplate.com/dist/mobile/
https://developers.google.com/speed/
https://developers.google.com/speed/docs/best-practices/rules_intro
http://barkingiguana.com/2009/12/04/simulating-slow-or-laggy-network-connections-in-os-x/

Cross-Browser/Device
http://modernizr.com/

Responsive/Mobile Web
http://css-tricks.com/search-results/?q=responsive
http://css-tricks.com/search-results/?q=mobile+web