UC Davis 2.0
An experiment in web performance optimization

Site Mirroring and Hosting Setup

A local copy of the UC Davis website was pulled using the HTTrack (http://www.httrack.com/page/1/en/index.html) CLI tool. This copy was then hosted via Firebase static hosting (https://firebase.google.com/).

Limitations

- **Font** - UC Davis' website relies on the Proxima Nova font from Adobe Typekit. This is tied to the school domain. Because I was unable to find a free, hosted version of the Proxima Nova font family, I opted to use an alternative font (Montserrat) which I downloaded and hosted on Firebase along with the website. I replaced references to the hosted Proxima Nova font accordingly within the source code.

- **Image Assets** - UC Davis hosts many of its media assets on its domain, along with its website. To simulate this with Firebase hosting, I pulled as many media assets as were available/necessary to setup a viable copy and hosted these on Firebase as well.

After establishing and hosting the baseline (davis-baseline), I duplicated it and hosted it as a separate project on Firebase. I would optimize the second copy (davis-optimized) and repeatedly compare web performance.

Link to hosted baseline site: https://davis-baseline.firebaseapp.com/

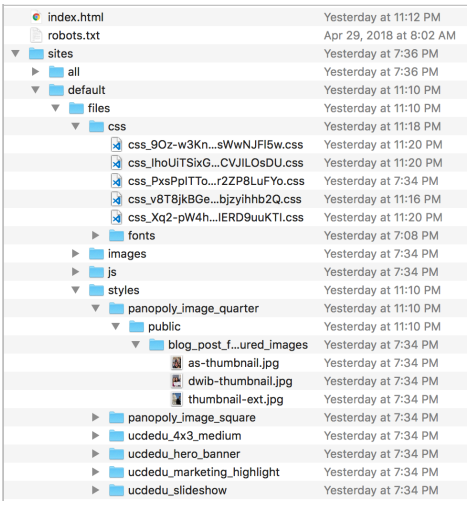
Link to hosted optimized site: https://davis-optimized.firebaseapp.com/

General Optimizations

URI Path Reduction

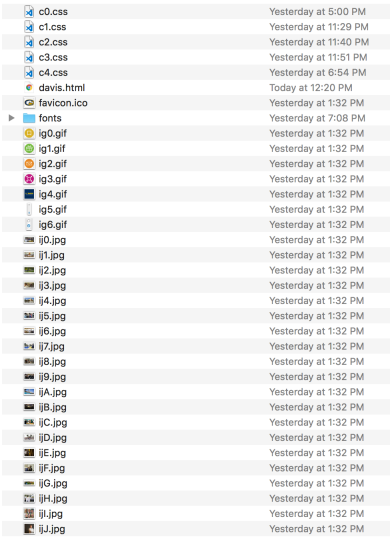
I flattened the site directory structure, and renamed files using minimal filenames.

Directory structure pre-optimization



Keeping filenames minimal

Directory structure post-optimization



and laying out all files in a single directory kept path names and asset references extremely short, somewhat shrinking HTML and CSS file sizes. This resulted in a couple KB of savings.

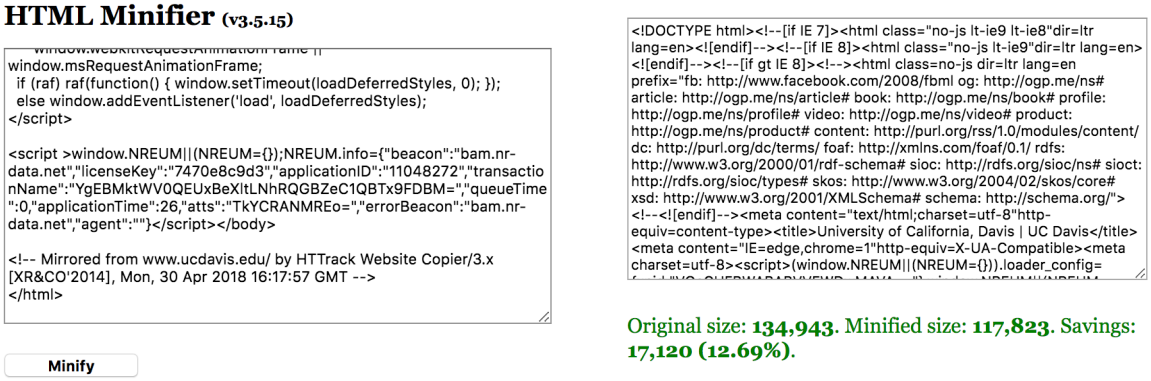
Browser Caching

I edited my firebase.json to allow some level of browser caching. As I was not all too familiar with the various configuration options, the setup was not optimal but allowed for caching most resources for approximately 7 days.

HTML Optimizations

Minification

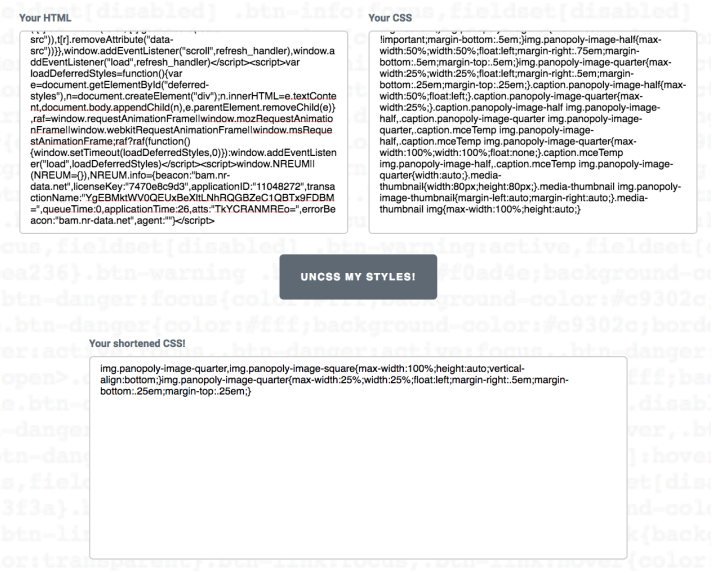
I used an html minification tool (https://kangax.github.io/html-minifier/) to minify the main HTML page. This resulted in ~12% savings.



CSS Optimizations

Minification and Unused Rule Removal

I used a CSS minificaiton tool (https://uncss-online.com/) to minify CSS files as well as remove unused rules. One resulting CSS file was so small, I decided to inline its content in a <style> tag within index.html. This lowered both the number of CSS requests from 5 to 4 as well as the total bytes of CSS requested/transferred.



Defer Loading of Non-Critical Style Sheets

Using a small snippet of vanilla JS, and a

```
<noscript id="deferred-styles">
  <link type="text/css" rel="stylesheet" href="c0.css" media="all" />
  <link type="text/css" rel="stylesheet" href="c1.css" media="all" />
</noscript>
```

```
<script>
  var loadDeferredStyles = function() {
    var addStylesNode = document.getElementById("deferred-styles");
    var replacement = document.createElement("div");
    replacement.innerHTML = addStylesNode.textContent;
    document.body.appendChild(replacement)
    addStylesNode.parentElement.removeChild(addStylesNode);
  };
  var raf = window.requestAnimationFrame || window.mozRequestAnimationFrame ||
      window.webkitRequestAnimationFrame || window.msRequestAnimationFrame;
  if (raf) raf(function() { window.setTimeout(loadDeferredStyles, 0); });
  else window.addEventListener('load', loadDeferredStyles);
</script>
```

<noscript> tag nested in the HTML's <body>, I was able to hold off on loading non-critical CSS files until after the page has been loaded. This prevents such CSS resources from becoming render blocking.

Preload Critical Style Sheets and Fonts

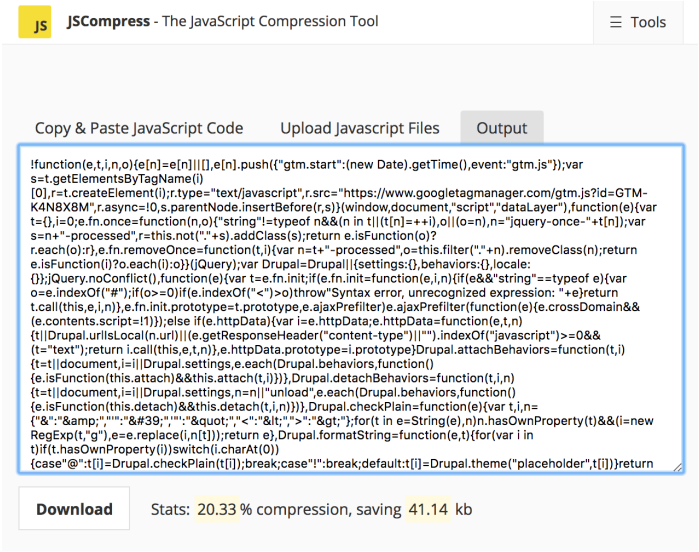
I used the 'rel="preload"' <link> attribute and value to specify key style sheets and fonts that need to be prioritized. For example, the Montserrat font is first referenced in a CSS file which, itself, is referenced in the primary HTML file. By specifying that it should be preloaded, instead of waiting for the browser to discover that the font is necessary only after parsing HTML and CSS, it becomes immediately known that it is a key resource and should be downloaded as soon as possible.

```
<link rel="preload" href="fonts/montserrat-v12-latin-regular.woff" as="font" type="font/woff" />
```

JS Optimizations

JS Bundling and Minification

I used an online JS Compression tool (https://jscompress.com/) to bundle, and minify javascript files. This lowered both the number of JavaScript requests as well as the total size of JavaScript assets.



Asynchronous JS <script> Tags

```
<script async src="j0.js"></script>
```

Many of the JS scripts in UC Davis' site are non-critical to the page's first render. I use the 'async' <script> attribute to specify that such scripts can be loaded asynchronously or the 'defer' <script> attribute to defer loading of such scripts until after the page has loaded.

Simpler Lazy Loading (see also Image Optimization - Lazy Loading)

I believe UC Davis was using some sort of jQuery library or plugin to implement lazy loading on images which were part of a slideshow on their webpage. I used a simple, inline, vanilla JS script to replace the jQuery plugin being used and implement lazy loading on all offscreen image assets (as opposed to only those that were part of the slideshow).

```
<script type="text/javascript">
  refresh_handler = function(e) {
    var elements = document.querySelectorAll("*[data-src]");
    for (var i = 0; i < elements.length; i++) {
      var boundingClientRect = elements[i].getBoundingClientRect();
      if (elements[i].hasAttribute("data-src") && boundingClientRect.top < window.innerHeight) {
        elements[i].setAttribute("src", elements[i].getAttribute("data-src"));
        elements[i].removeAttribute("data-src");
      }
    }
  };
  window.addEventListener('scroll', refresh_handler);
  window.addEventListener('load', refresh_handler);
</script>
```

Image Optimizations

Lower JPG Quality and Convert to Progressive JPG

I used an online jpg optimization tool (http://compressimage.toolur.com/) to convert jpg assets to progressive jpg format as well as downgrade the image quality. On average, I ended up being able to lower the quality of most jpg images to 50% without any perceptible changes to page appearance. This led to a 50% savings on each jpg image asset.

sample asset comparison: On the left, progressive, downgraded jpg asset (25KB). On the right, original, non-progressive asset (44KB).

Convert images with little to no gradients to GIF and optimize

Many of UC Davis' image assets were logos which were only comprised of a handful of colors and had little to no gradient. I used an online image resizing tool (http://resizeimage.net/) to convert such images to the gif format and reduced the number of colors used to minimize size.

sample asset comparison: On the left, asset converted to gif and color reduced from 256 to 3 (6KB). On the right, asset in original jpg format (24KB).

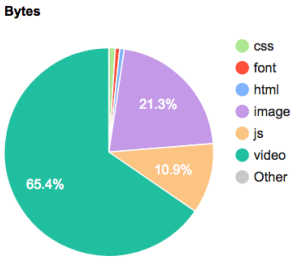
ImageOptim

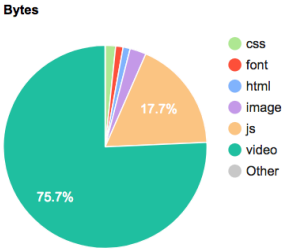
I ran any remaining image assets through ImageOptim to minimize size. On average, I saw 30% savings per file.

Lazy Loading (see also JS Optimization - Simpler Lazy Loading)

I assign the 'src' attribute of every <img> tag to a blank gif while storing the desired image in a 'data-src' attribute. The JS script mentioned in "JS Optimizations - Simpler Lazy Loading" then listens for scroll or load events and swaps the 'src' and 'data-src' attributes of any images whose bounds are within view. This allows the browser hold off on loading any offscreen images until necessary. UC Davis did integrate a lazy loading plugin but only used it on images that were part of a slideshow. I was able to implement lazy loading on all image elements.

Final Results





MIME Type	Bytes	Uncompressed
video	6,011,710	6,011,710
image	1,961,031	1,985,211
js	1,003,798	3,099,063
css	90,164	532,508
html	63,692	220,175
font	63,462	62,632
other	1,477	291
flash	0	0

MIME Type	Bytes	Uncompressed
video	4,085,536	4,085,536
js	957,838	2,982,757
image	139,437	158,864
css	89,488	527,442
font	63,386	62,632
html	60,824	202,481
other	1,475	291
flash	0	0

Requests Less Data

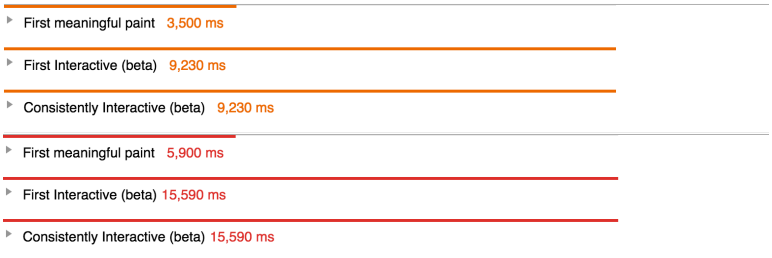
The two charts are taken from performance reviews by webpagetest.org. On the left is a breakdown of requests made by the baseline site on first view broken down by MIME type. The right is the same chart but for the optimized site. Notice that, because of the implementation of lazy loading on all images, the number of image bytes requested on the first view of the optimized site is drastically smaller than that of the baseline site (1,961,031 bytes vs. 139,437 bytes).

"Loads Faster"

To the right are more statistics taken from the same performance tests. The bottom chart holds the statistics for the baseline app while the top chart holds the statistics for the optimized app. Notice a 3 second speed up in document completion time and a 4 second speed up in time to fully loaded.

Document Complete			Fully Loaded		
Time	Requests	Bytes In	Time	Requests	Bytes In
5.753s	91	5,257 KB	11.171s	111	5,274 KB

Document Complete			Fully Loaded		
Time	Requests	Bytes In	Time	Requests	Bytes In
10.672s	144	8,969 KB	15.020s	152	8,983 KB



"Looks Faster"

The two sets of statistics to the right are taken from Google's Lighthouse tool. The bottom set is for the baseline site while the top is for the optimized site.

webpagetest optimized results:
https://www.webpagetest.org/result/180502_J5_a01df4efe293e266c2594047d7ff970c/

webpagetest baseline results:
https://www.webpagetest.org/result/180502_0P_60b8e77f2a51b579c7b36e67381a9843/