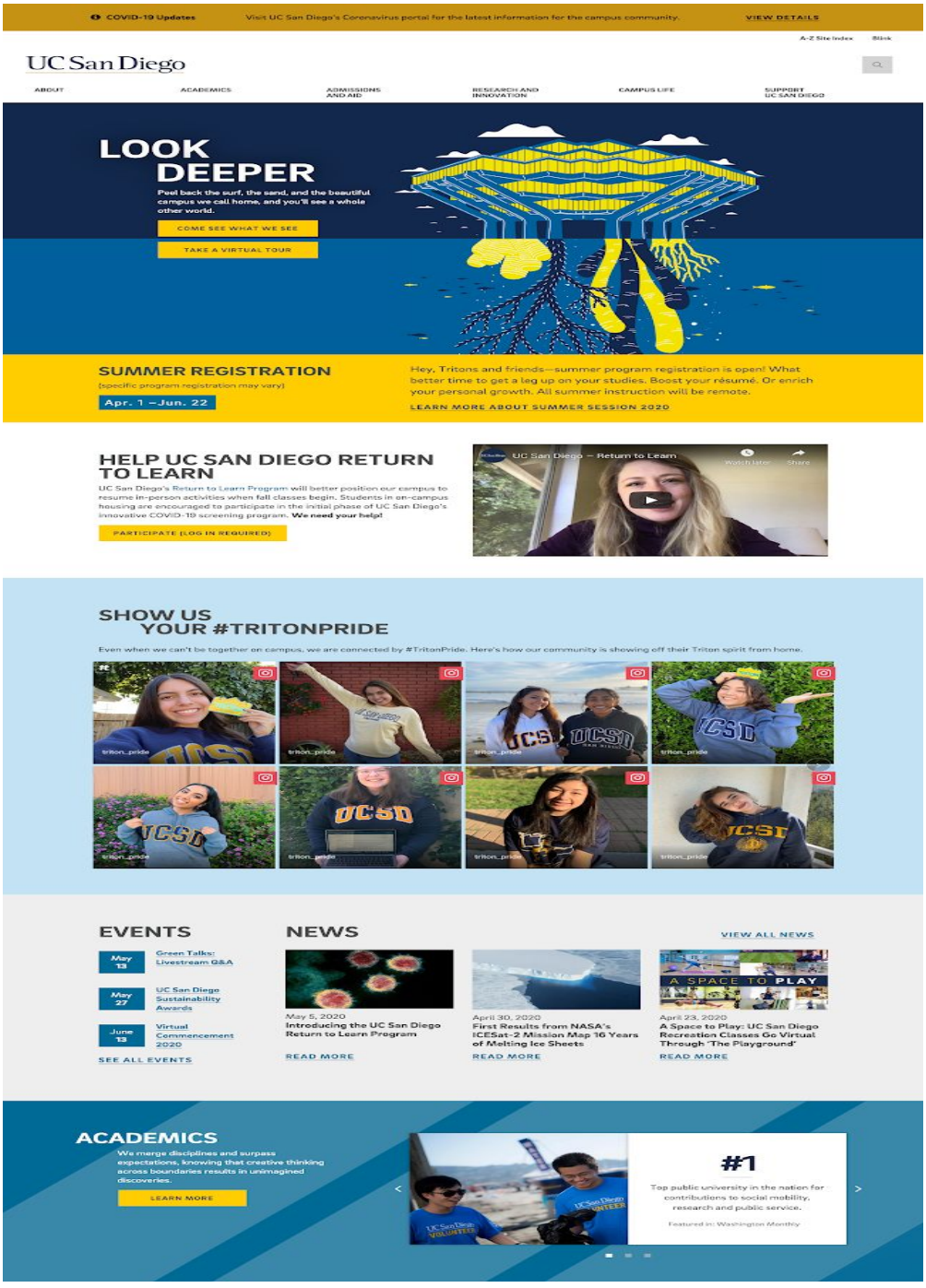
UCSD CSE 190 A00 WEB APP PERFORMANCE AND MONITORING SPRING 2020

PUBLIC SITE PERFORMANCE REMEDIATION

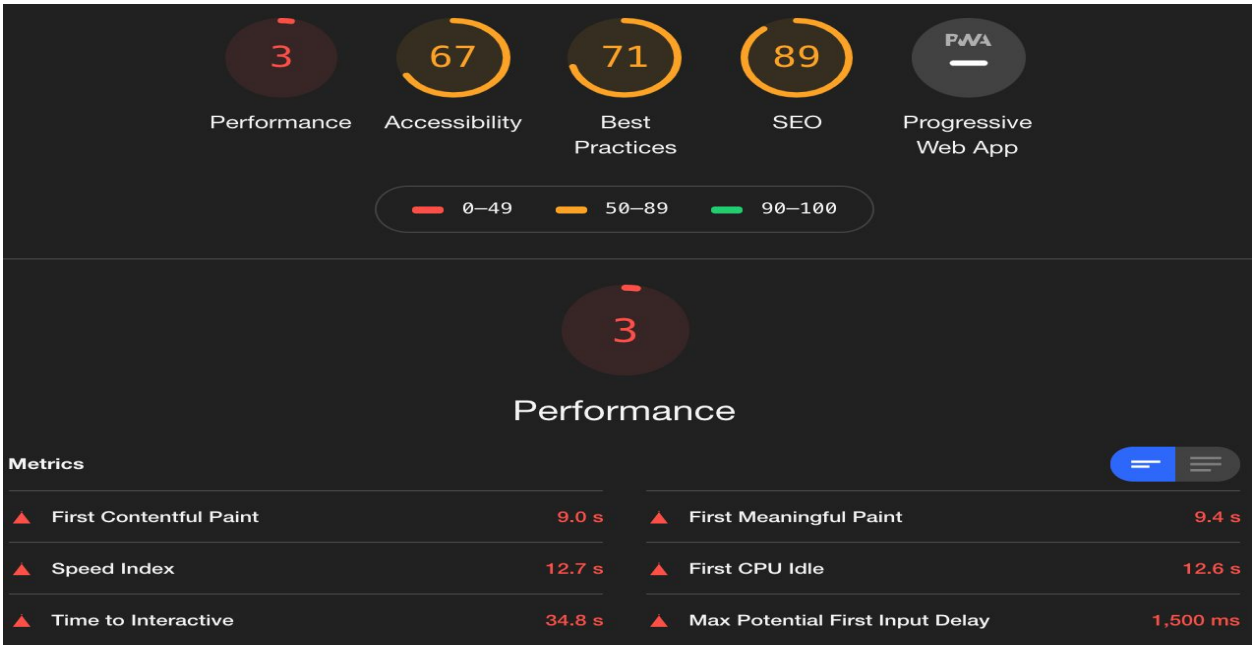
By Gabriela Shirley

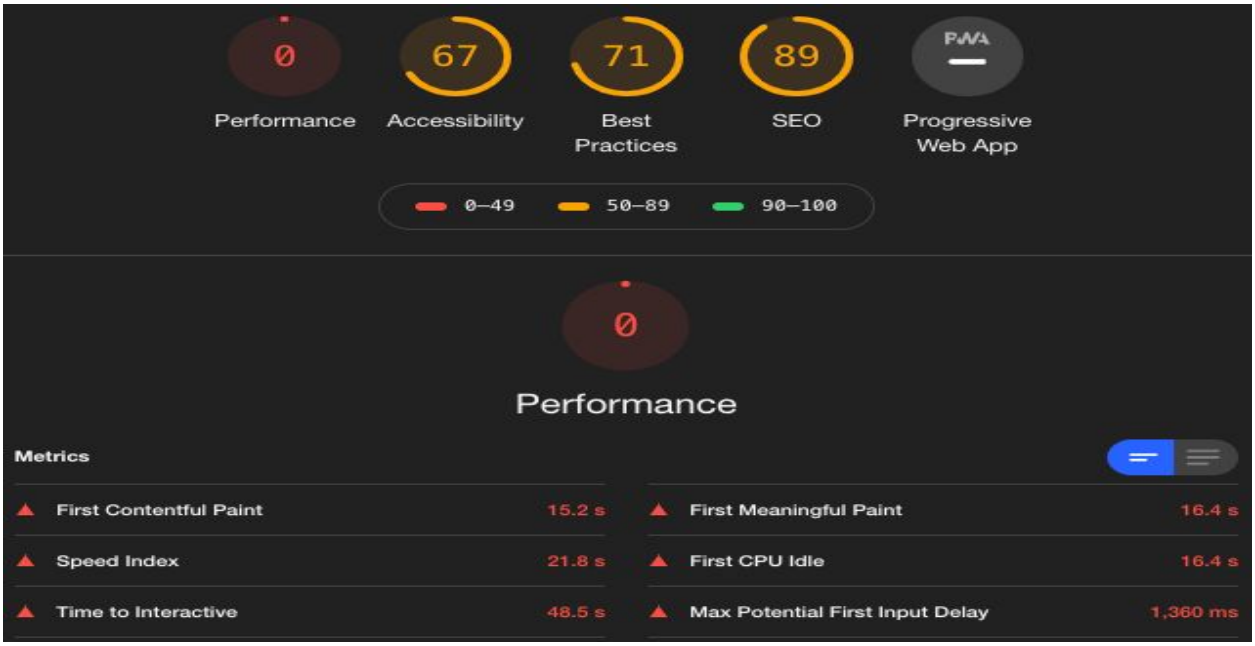


Mirror UCSD Homepage to my Github Pages

Moving only the assets of the homepage to my gabrielashirley.github.io repository, it takes up 5.9MB space to download instead of 72.1MB when I tried to mirror the entire UCSD site. The link to the original site that I mirrored is here: https://gabrielashirley.github.io/original/.

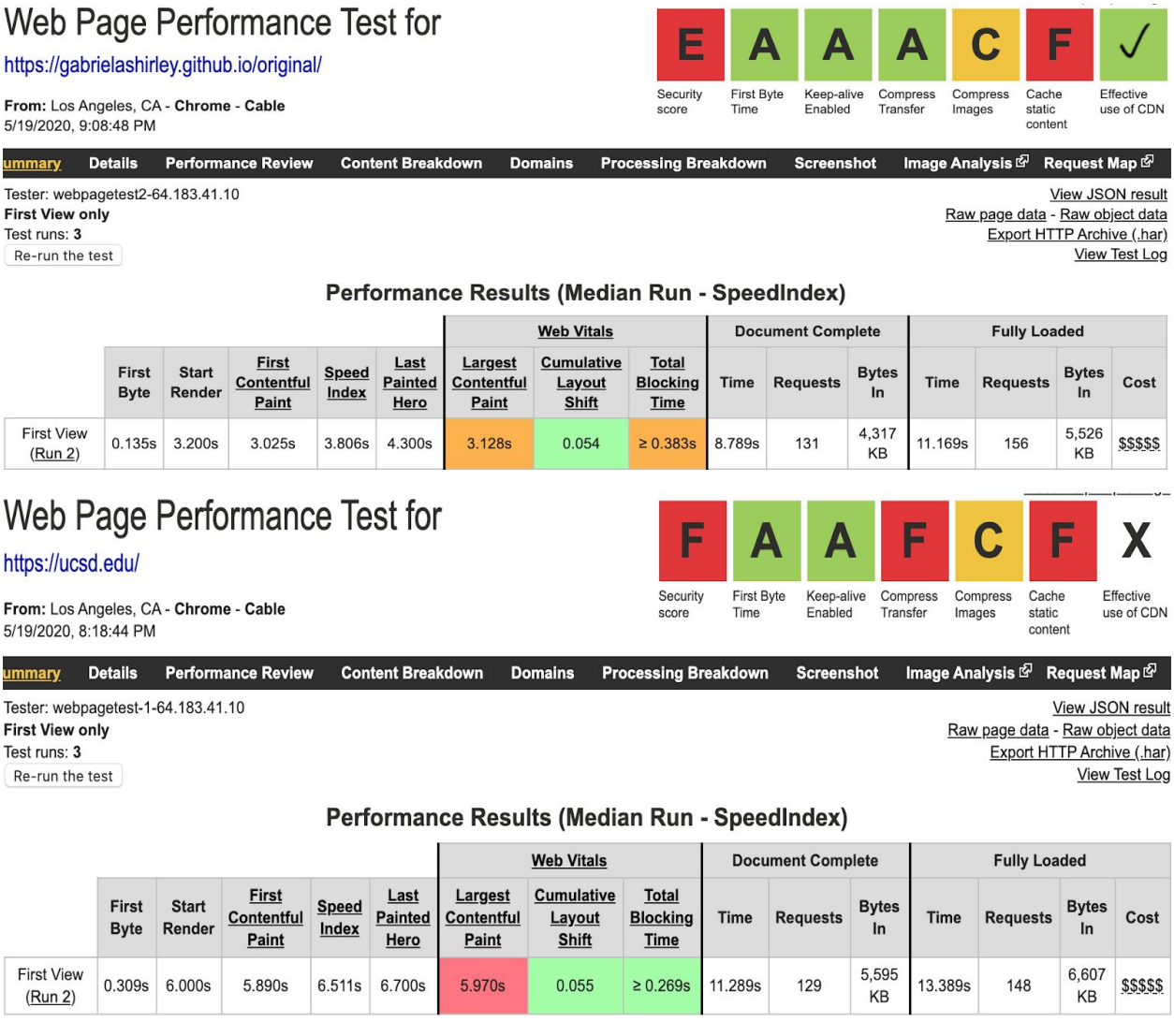
Taking a baseline reading from Lighthouse:





It does not show any significant difference compared to when I tested the real site (https://ucsd.edu/) on the second picture. However, my first guess on the 3 points better in performance, with every smaller (means better) number under the metrics when I served the homepage on Github Pages has something to do with how good is Github Pages' backend on serving static sites. As discussed in this site, https://www.savjee.be/2017/10/Static-website-hosting-who-is-fastest/, Github Pages wins as the fastest and that everyone performs fairly consistently compared to other hosting services such as Netlify, Amazon S3, CloudFront, Firebase Hosting, and Google Cloud.

Taking a baseline reading from WebPageTest:



Web Page Performance Test for
https://gabrielashirley.github.io/original/

From: Los Angeles, CA - **Chrome** - **Cable**
5/19/2020, 9:08:48 PM

E	A	A	A	C	F	✓
Security score	First Byte Time	Keep-alive Enabled	Compress Transfer	Compress Images	Cache static content	Effective use of CDN

ummary | Details | Performance Review | Content Breakdown | Domains | Processing Breakdown | Screenshot | Image Analysis | Request Map

Tester: webpagetest2-64.183.41.10
First View only
Test runs: **3**
Re-run the test

View JSON result
Raw page data - Raw object data
Export HTTP Archive (.har)
View Test Log

Performance Results (Median Run - SpeedIndex)

						Web Vitals			Document Complete			Fully Loaded			
	First Byte	Start Render	First Contentful Paint	Speed Index	Last Painted Hero	Largest Contentful Paint	Cumulative Layout Shift	Total Blocking Time	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 2)	0.135s	3.200s	3.025s	3.806s	4.300s	3.128s	0.054	≥ 0.383s	8.789s	131	4,317 KB	11.169s	156	5,526 KB	$$$$$

Web Page Performance Test for
https://ucsd.edu/

From: Los Angeles, CA - **Chrome** - **Cable**
5/19/2020, 8:18:44 PM

F	A	A	F	C	F	X
Security score	First Byte Time	Keep-alive Enabled	Compress Transfer	Compress Images	Cache static content	Effective use of CDN

ummary | Details | Performance Review | Content Breakdown | Domains | Processing Breakdown | Screenshot | Image Analysis | Request Map

Tester: webpagetest-1-64.183.41.10
First View only
Test runs: **3**
Re-run the test

View JSON result
Raw page data - Raw object data
Export HTTP Archive (.har)
View Test Log

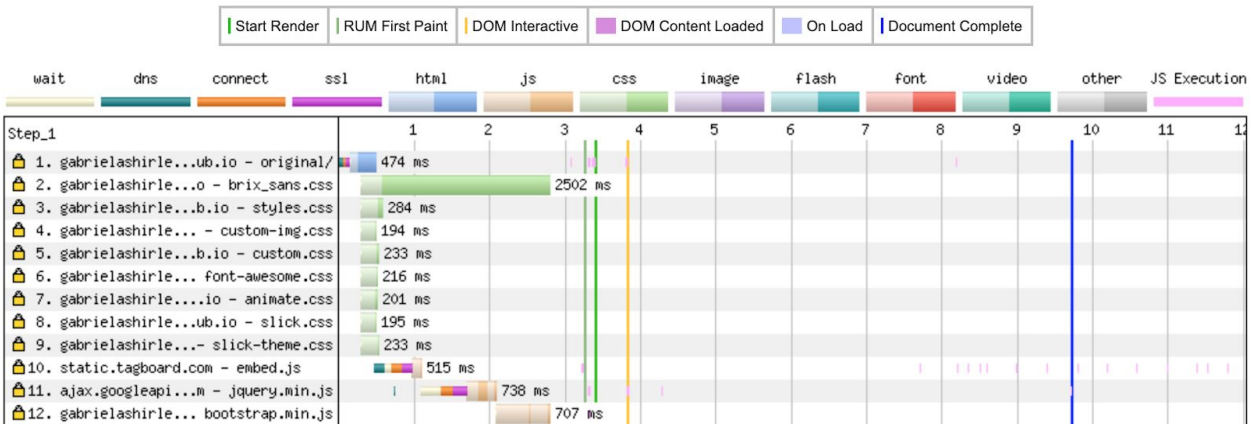
Performance Results (Median Run - SpeedIndex)

						Web Vitals			Document Complete			Fully Loaded			
	First Byte	Start Render	First Contentful Paint	Speed Index	Last Painted Hero	Largest Contentful Paint	Cumulative Layout Shift	Total Blocking Time	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 2)	0.309s	6.000s	5.890s	6.511s	6.700s	5.970s	0.055	≥ 0.269s	11.289s	129	5,595 KB	13.389s	148	6,607 KB	$$$$$

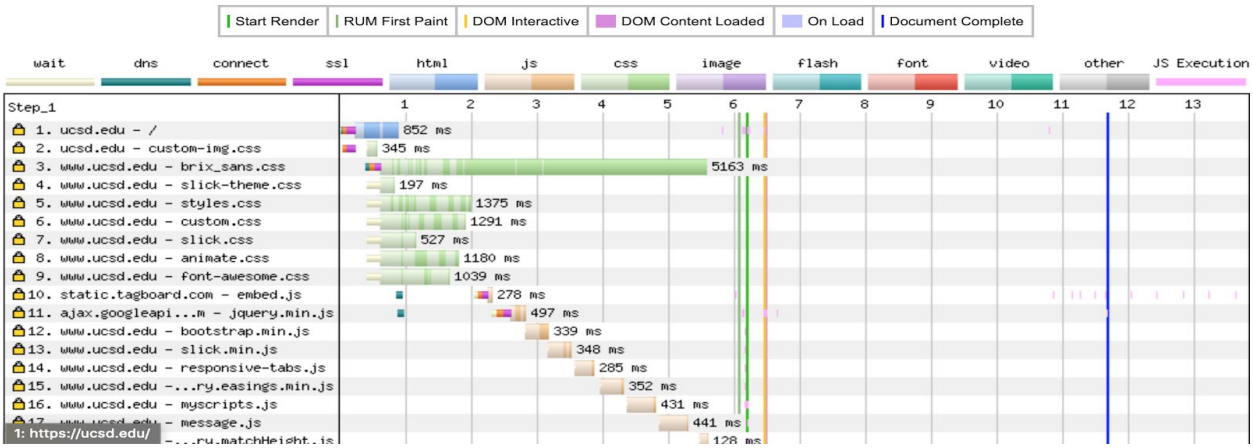
Comparing the test results of my mirror and the real site, I observed that the total opposite score of Compress Transfer and Effective use of Content Delivery Network (CDN) (Mirror: A and ✓ vs Real: F and X) which in other words means the way the we fetch compressible static assets does make a difference. Since Github automatically compresses required files into GZip if the browser supports it, whereas UCSD Server seems to not turn the feature on (or maybe it does not support it), it helps saving time on the resource load times as well as slow server response times. One extra thing that might help improve the Time to First Byte (TTFB) of the real site is the use of CDN like Cloudflare, where it will manage the server routing to the nearest ones from the user, thus reducing latency. Currently only the third party assets use CDN like Google (including YouTube), Facebook, Cloudflare, etc.

Start render time is suggested to be around 1-2s. However, I noticed that both my mirror and the real site have some kind of render blocking due to several CSS files which results in having start render in 3.2s and 6.0s respectively.

Waterfall View



Waterfall View

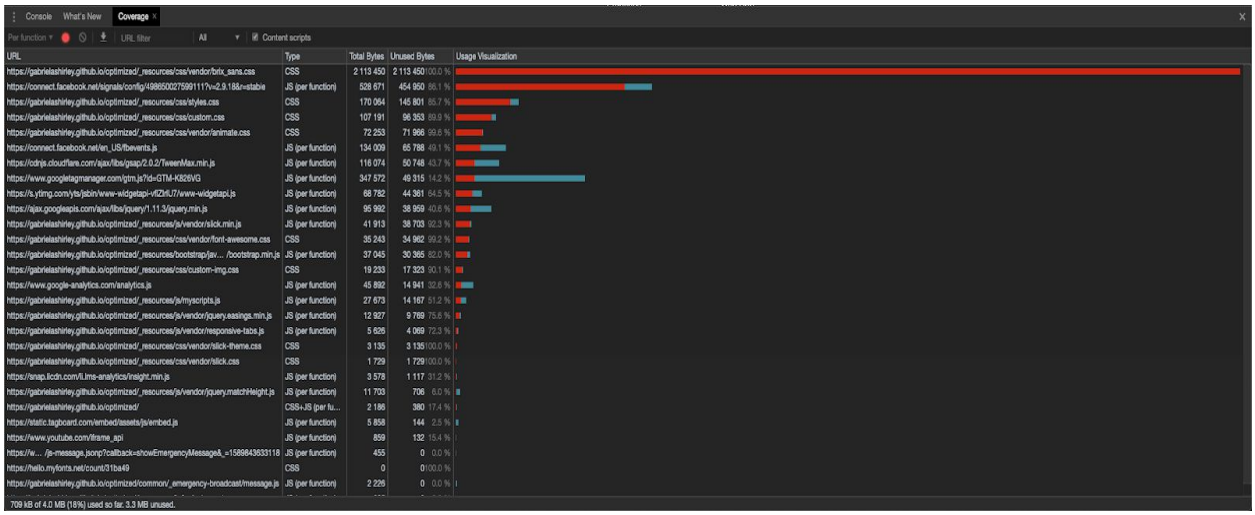


Optimization

So to begin with my optimization attempts, I would take my mirror test results as my baseline from now on.

1) Eliminate render-blocking (apply *defer* and *rel=preload*) and add <noscript> fallback

Based on my observation, these resources can be deferred for later load and execution or removed thus increasing start render time. Taken from class notion notes, scripts with the defer attribute execute after HTML parsing is completely finished, but before the DOMContentLoaded event. defer guarantees scripts will be executed in the order they appear in the HTML and will not block the parser.



a) CSS resources:

- `_resources/css/vendor/brix_sans.css`

  This is a font declaration. It can be deferred later in favor of similar fallback fonts and reducing 2.1 MB load. Although it seems that this file is the biggest render-blocking of this page and has 100% unused bytes from the Coverage report, we cannot guarantee that this file is totally useless for the entire site unless we check coverage on every single page, while executing JavaScript, and under any possible combinations of state and media queries. And apparently, I found that some of the font-faces are used on some *button* elements on the homepage. Therefore, I chose to defer.

- `_resources/css/custom-img.css`

  This file contains background-images that are used throughout the bottom sections of

the page. I chose to defer since it will not affect the first impression considering the limitation of viewports.

- `_resources/css/custom.css`
  This file contains a lot of styles that are not used in the homepage so it is recommended to either remove or defer this resource. I chose to defer because it might be used in other pages, so the user can cache this resource for later-use.
- `_resources/css/vendor/font-awesome.css`
  This is a font awesome integration usually for icons, and the only usage I found is for the left and right arrow on the carousel. I chose to defer this since the icon usage in the home page is minimal.
- `_resources/css/vendor/slick.css` & `_resources/css/vendor/slick-theme.css`
  This comes from a slider vendor "Slick". I chose to defer this since it is only used in the "Academics" section in the middle of the page.

b) JavaScript resources:

- `https://cdnjs.cloudflare.com/ajax/libs/gsap/2.0.2/TweenMax.min.js` & `_resources/js/look-deeper.js`

  This is used for graphic animation in the “Look Deeper” section, although it is barely noticable in my opinion. I chose to defer, because deferring this would only make the animation load later after the initial render has occurred.

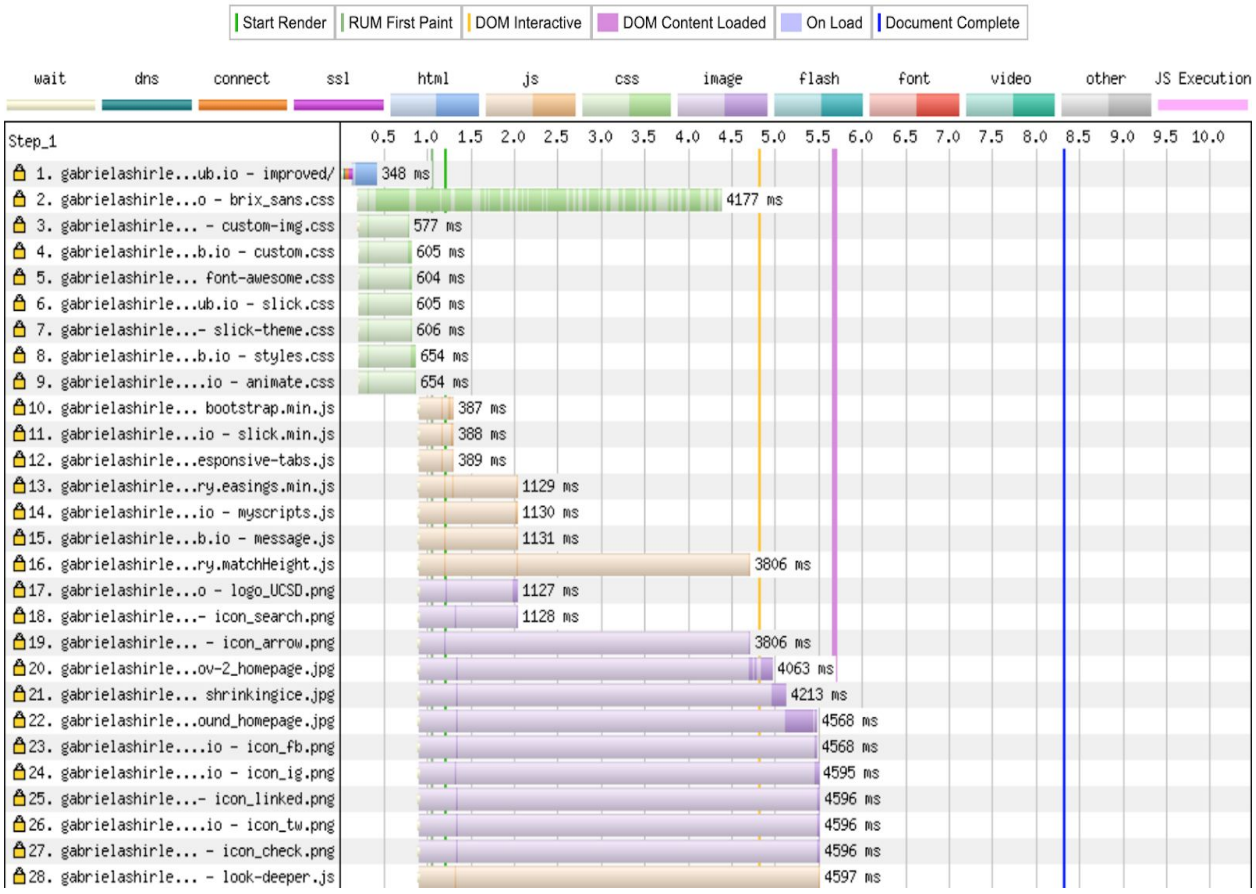
After taken the step I re-ran both Lighthouse and WebPageTest, the result is way better, the start render time improved from 3.2s to 1.2s:

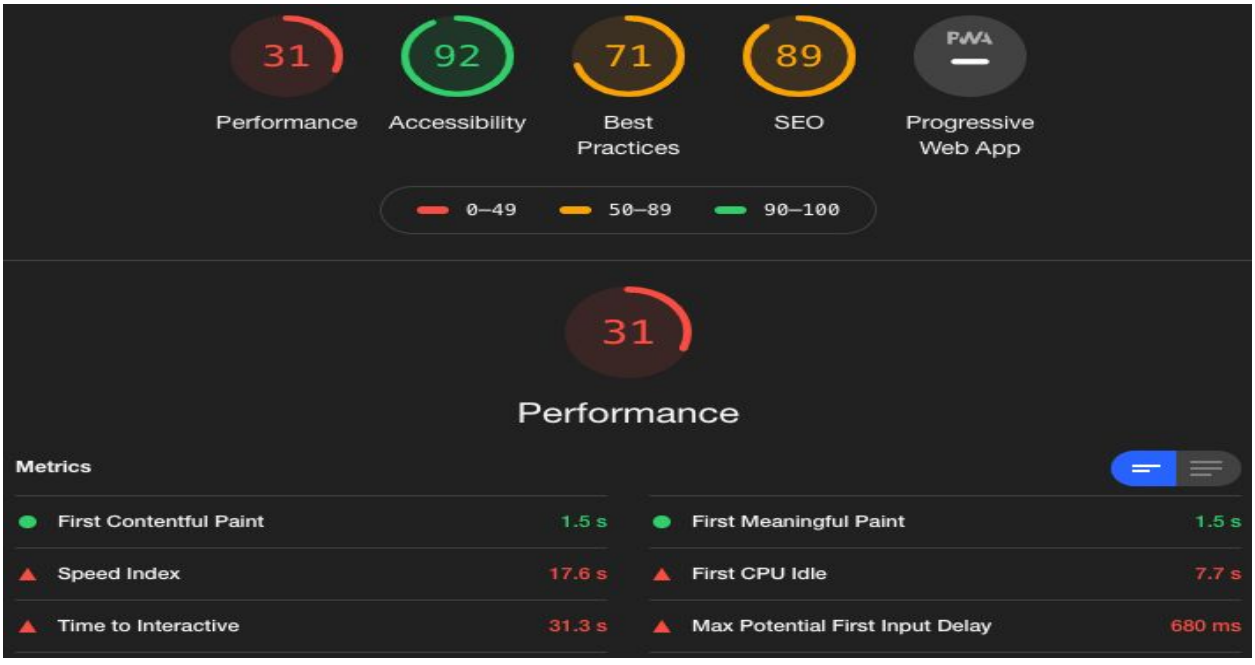
Performance Results (Median Run - SpeedIndex)

						Web Vitals		Document Complete			Fully Loaded			
	First Byte	Start Render	First Contentful Paint	Speed Index	Last Painted Hero	Largest Contentful Paint	Cumulative Layout Shift	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 1)	0.171s	1.200s	1.044s	2.898s	8.000s	1.044s	0.047	8.317s	104	4,303 KB	10.451s	139	5,071 KB	$$$$$

						Web Vitals			Document Complete			Fully Loaded			
	First Byte	Start Render	First Contentful Paint	Speed Index	Last Painted Hero	Largest Contentful Paint	Cumulative Layout Shift	Total Blocking Time	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 3)	0.144s	0.200s	0.214s	2.646s	7.500s	0.947s	1.056	0.200s	8.249s	121	3,974 KB	10.350s	138	5,007 KB	$$$$$

Waterfall View





2) Removed unused CSS and resource minification

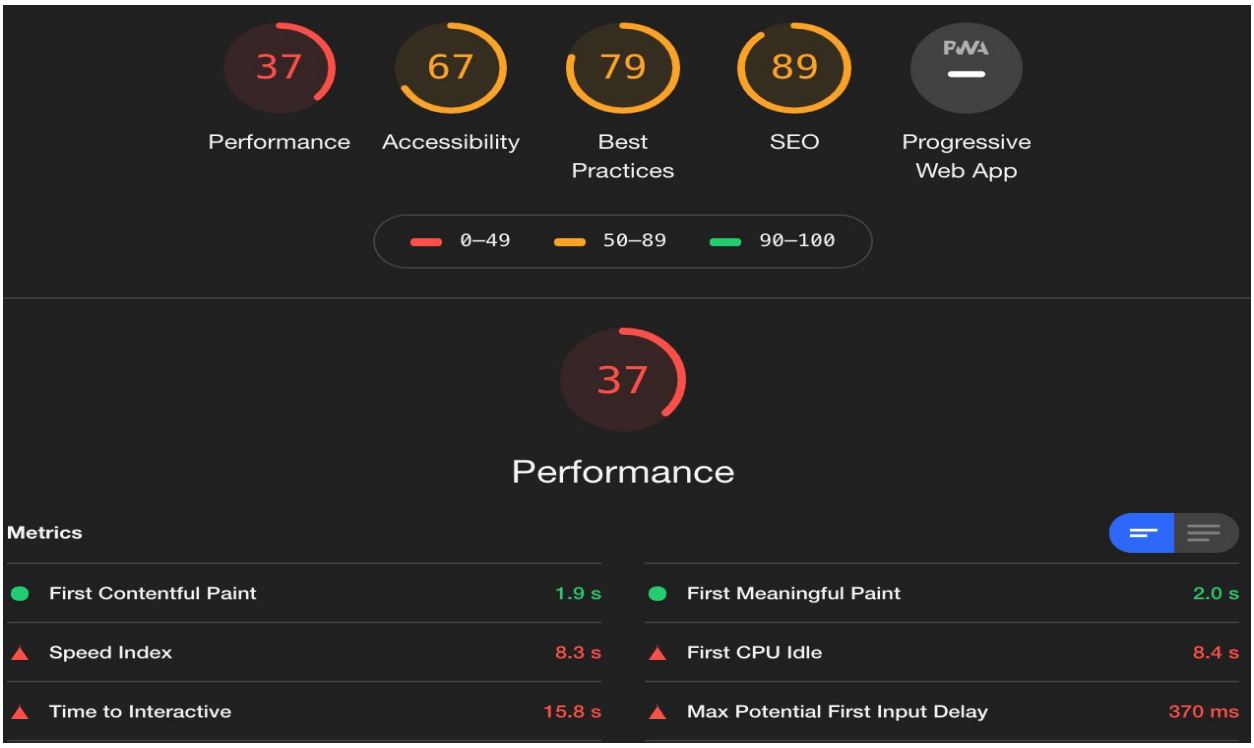
I noticed that most of the JS and CSS files used are not minified yet and full of unused CSS. First, I did some byte shaving by changing *if else* on to ternary function, removing comments and dead codes, changing *function ()* to *arrow =>* on *message.js* and *myscripts.js*.

I removed double declarations on background images on *custom-img.css*, using both *background* and *background-image* to refer to the same file and the same class selector. Moreover, there are some double even triple declarations on certain selectors as well such as *.attend-module* and *.attend-cta*.

Next, I found that @font-face in *font-awesome.css* has missing source files (it only has *woff2*) so I used only the *woff2* format and removed the other formats since *woff2* already covers 95.23% users of modern browsers with upgraded compression. The same thing goes for *glyphicons-halflings-regular.woff2* under *bootstrap/fonts*, so I only kept that one as the source file on *styles.css*. I also removed unused font awesome icons which cut around 2,000 lines of codes, since the homepage seems to only use left and right arrow icons.

Out of curiosity, I tried another hack on our biggest CSS problem *brix_sans.css*. I modified it to extract fonts to separate files under *_resources/fonts/brix_sans*. For controlling font performance, I applied *font-display: swap* as it has no block period and infinite swap period to ensure text remains visible during webfont load (https://developers.google.com/web/updates/2016/02/font-display). I also removed the 2 font-face declarations that are already commented out, and moved the `@import url("//hello.myfonts.net/count/31ba49");` to the bottom since this unnecessary import of an analytics URL seems to cause the block. When I see the Network tab and disable cache, this import takes a long time to load, perhaps the *myfonts* server is the problem.

Since I feel like I have done much on the manual deletion, I minified JS and CSS files using minifier tool online https://javascript-minifier.com/ and https://cssminifier.com/ for: *custom-img.css, custom.css, brix_sans.css, animate.css, font-awesome.css, slick-theme.css, slick.css, look-deeper.js, myscripts.js, jquery-matchHeight.js, responsive-tabs.js,* and *message.js.* This step shows a bit of improvement on the performance score:



Performance Results (Median Run - SpeedIndex)

						Web Vitals		Document Complete			Fully Loaded			
	First Byte	Start Render	First Contentful Paint	Speed Index	Last Painted Hero	Largest Contentful Paint	Cumulative Layout Shift	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 3)	0.131s	0.800s	0.646s	1.368s	2.200s	2.191s	0.01	4.902s	117	2,212 KB	7.028s	147	3,173 KB	$$$$$

Additionally, based on the networking data, the currently optimized site has 8.9 MB transferred and 14.0 MB resources whereas the originally mirrored site has 17.5 MB transferred and 23.8 MB resources. This shows great progress in all our minification attempts.

3) Image compressions and proper sizing

It seems that some of the images have been compressed well enough, there are no excessively large images > 1MB and most of the small web graphics are < 300KB. However, there are still some potentially compressed images and improper sizes. So I compressed the JPEG files and made it as progressive JPEG using http://optimizilla.com/. I changed one UCSD logo to download it into the local directory instead of fetching http://www.ucsd.edu/_resources/img/logo_UCSD_white.png.

Modified images:

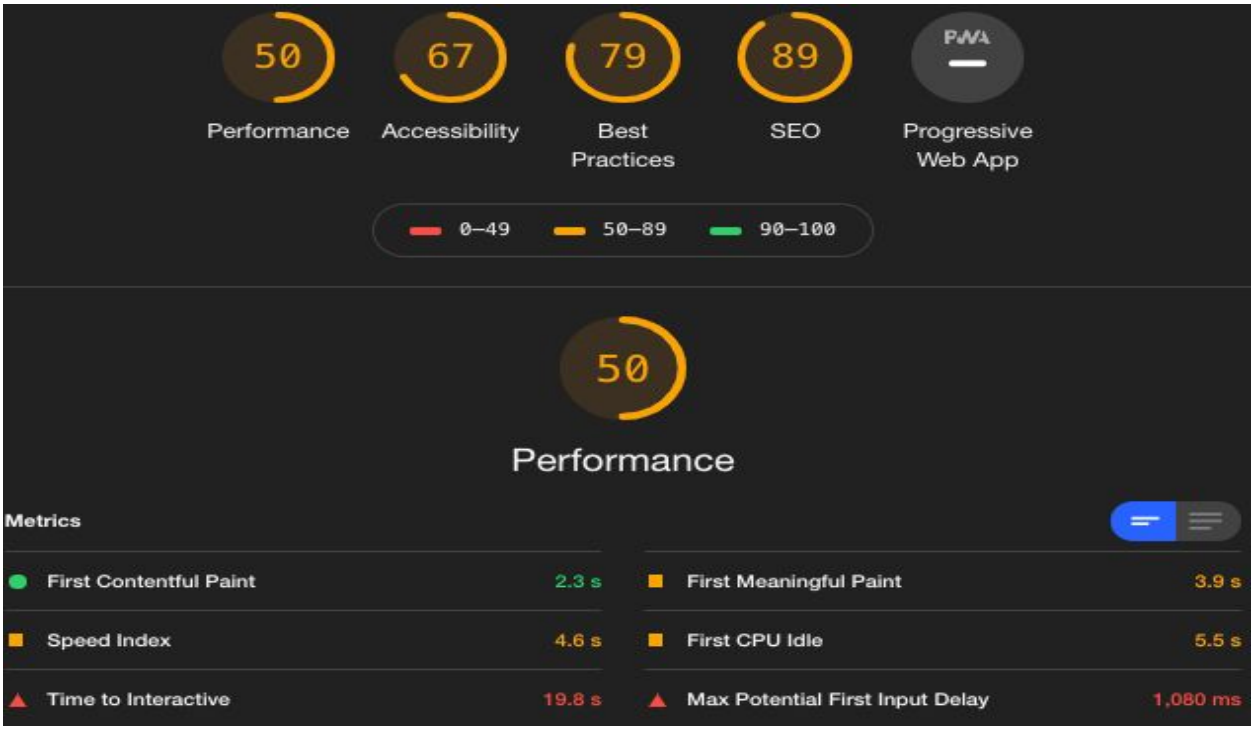
- Compressed:
    i. `_resources/img/bg_attend.jpg`
    Result: from 587.9 KB to 253.3 KB
    ii. `_resources/img/bg_map.jpg`
    Result: from 94.0 KB to 45.6 KB
    iii. `_images/homepage/bg_studentlife_alt2.jpg`
    Result: from 244.3 KB to 62.1 KB
    iv. `_images/The-Playground_homepage.jpg`
    Result: from 141 KB to 66 KB
    v. `_images/shrinkingice.jpg`
    Result: from 82 KB to 8.7 KB
    vi. `_images/homepage/u-care-ucsd-1400x439.jpg`
    Result: from 17 KB to 14.1 KB
    vii. `_images/home-financial-aid.jpg`
    Result: from 54 KB to 47.7 KB
- Resized:
    i. `_images/sars-cov-2_homepage.jpg`
    Status: resized to 450px wide as shown on https://ucsdnews.ucsd.edu/pressrelease/introducing-the-uc-san-diego-return-to-learn-program
    Result: from 104 KB to 38 KB
    ii. `_images/The-Playground_homepage.jpg`
    Status: resized to 265px wide since as shown on home page
    Result: from 66 KB to 25.4 KB
    iii. `_images/stats-number-one-public-service.jpg`
    Status: resized to 310px high as shown on home page through CSS rules
    Result: from 45 KB to 31 KB
    iv. `_images/stats-second-quality-education.jpg`
    Status: resized to 310px high as shown on home page through CSS rules
    Result: from 58 KB to 37 KB

v. `_images/stats-top-ten.jpg`

Status: resized to 310px high as shown on home page through CSS rules

Result: from 71 KB to 41 KB

The improvement seems pretty good, we 100% compressed all images and use progressive JPEGs.



Compress Images: 100/100 Learn More

679.5 KB total in images, target size = 679.5 KB - potential savings = **0.0 KB**

Use Progressive JPEGs: 100/100 Learn More

646.1 KB of a possible 646.1 KB (100%) were from progressive JPEG images

Performance Results (Median Run - SpeedIndex)

						Web Vitals			Document Complete			Fully Loaded			
	First Byte	Start Render	First Contentful Paint	Speed Index	Last Painted Hero	Largest Contentful Paint	Cumulative Layout Shift	Total Blocking Time	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 1)	0.131s	0.700s	0.600s	1.293s	1.700s	0.600s	0.047	≥ 0.419s	5.034s	121	2,170 KB	7.346s	146	3,281 KB	$$$$$

4) HTML Formatting (removed unused line breaks), Extracted SVG, and Utilized CDN for several vendor resources

My first impression looking at the index.html file is the messy, excessively long code with too many unused line breaks. So I formatted the document using VSCode built-in HTML Language Feature formatter, and removed all the unused line breaks. Then, I extracted the 2 SVG elements to *hero-big.svg* and *hero-small.svg* under *_resources*:

```
<object data="_resources/hero-big.svg" type="image/svg+xml" class="hide-for-small-only"></object>
<object data="_resources/hero-small.svg" type="image/svg+xml" class="show-for-small-only"></object>
```

Next, I utilized CDN usage for several vendor resources: slick.min.css, slick-theme.min.css, jquery.matchHeight-min.js, and slick.js.

```
<!-- <script src="_resources/js/vendor/jquery.matchHeight.js"></script> -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery.matchHeight/0.7.2/jquery.matchHeight-min.js" crossorigin="anonymous"></script>
```

```
<!-- <script src="_resources/js/vendor/slick.min.js" type="text/javascript"></script> -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.6.0/slick.js" crossorigin="anonymous"></script>
```

```
<!-- Slick Slider for custom carousel -->
<!-- <link href="_resources/css/vendor/slick.css" rel="stylesheet" type="text/css"> -->
<link rel="preload" href="https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.6.0/slick.min.css" as="style" onload="this.onload=null;this.rel='stylesheet'">
<!--  Slick Slider Theme -->
<!-- <link href="_resources/css/vendor/slick-theme.css" rel="stylesheet" type="text/css"> -->
<link rel="preload" href="https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.6.0/slick-theme.min.css" as="style" onload="this.onload=null;this.rel='stylesheet'">
```

Performance Results (Median Run - SpeedIndex)

						Web Vitals			Document Complete			Fully Loaded			
	First Byte	Start Render	First Contentful Paint	Speed Index	Last Painted Hero	Largest Contentful Paint	Cumulative Layout Shift	Total Blocking Time	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 2)	0.165s	0.600s	0.453s	1.111s	1.300s	0.589s	0.022	≥ 0.171s	5.000s	139	2,141 KB	7.118s	148	3,264 KB	$$$$$

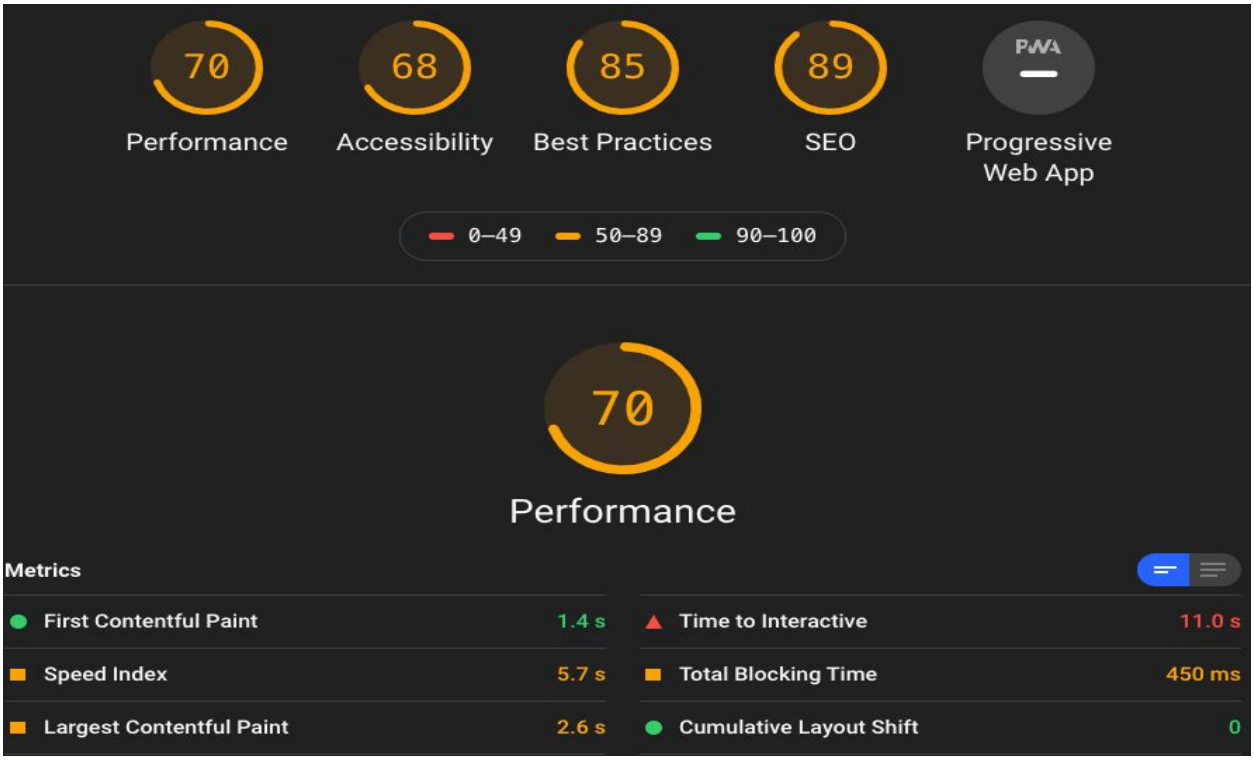
This step apparently does not show any significant improvement from before.

5) Extract Critical CSS

Since our current goal is to improve the web performance, I chose to sacrifice the code readability and purify the CSS to only one main file *purified.css*, resulting in less HTTP requests and staying DRY (Don't Repeat Yourself). I used https://github.com/purifycss/purifycss and it reduced all the CSS files by ~25.5%. This step significantly raises the performance score from 50 to 70.

```
gabrielashirley@Shirleys-MBP-3 css % purifycss custom-img.css custom.css styles.css vendor/animate.css vendor/slick-t
heme.css vendor/slick.css vendor/font-awesome.css vendor/brix_sans.css ../js/look-deeper.js ../js/myscripts.js ../js/
vendor/jquery.easings.min.js ../js/vendor/jquery.matchHeight.js ../js/vendor/slick.min.js ../js/vendor/responsive-tab
s.js ../../common/_emergency-broadcast/message.js ../../index.html --min --info --out purified.css

    ------------------------------------------------
    |
    |   PurifyCSS has reduced the file size by ~ 25.5%
    |
    ------------------------------------------------
```



Performance Results (Median Run - SpeedIndex)

						Web Vitals			Document Complete			Fully Loaded			
	First Byte	Start Render	First Contentful Paint	Speed Index	Last Painted Hero	Largest Contentful Paint	Cumulative Layout Shift	Total Blocking Time	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 2)	0.134s	0.500s	0.412s	1.041s	1.300s	0.523s	0.023	≥ 0.223s	4.880s	119	2,124 KB	7.250s	141	3,264 KB	$$$$$

Conclusion

Up until this point, I have tried to optimize the mirror of https://ucsd.edu/ from 3 to 70 https://gabrielashirley.github.io/improved/. Reading the baseline report from Lighthouse and WebPageTest, I quickly noticed how bad the render-blocking CSS affects the performance, especially the start render time metric. So I manipulated the JS and CSS load using *defer* and *rel=preload* which gave score 31. I then removed unused CSS manually and minified all JS and CSS files, I modified the huge *brix_sans.css* too. This step cut ~42% resources and ~20% transferred according to the Network tab. Next, I optimized images by compressing and resizing. This shows quite an increase to 50. Then, I removed unused line breaks on HTML, extracted 2 SVGs, and utilized CDN for some vendor resources. And last but not least, I extracted critical CSS into one file which reduced the CSS file size by ~25.5%. This step apparently solved another performance bug from this site, adding 20 more points to the performance score. So in the meantime, I got 70, but hopefully I could work on some more improvements in the future.