Page Speed Audit

[redacted]

Table of Contents

Purpose of This Document

The purpose of this document is to...

- Provide results of our performance analysis of your website.
- Provide recommendations for site speed improvements.

In this document, you will see each recommendation accompanied by a difficulty and priority rating.

Please review what those ratings mean to us:

Difficulty

Low - Does not require extensive expertise to execute

Medium - Requires some expertise to execute

High - Requires a high level of expertise to execute

Priority

Low - A non-critical issue, should be addressed within the next three months

Medium - A non-critical issue, should be addressed within the next month

High - A critical issue, should be addressed as soon as possible

Browser Caching

Browser caching, also known as expires headers, are a web server's way of telling a visiting browser "This file won't be changing for a while." Used correctly, they reduce the number of HTTP requests required per page view, and that's a huge performance gain.

Homepage

Images

Images on the homepage have short expiration header timeframes, approximately 1-3 days. Visitors who return after 1-3 days are re-downloading these files, slowing their page loading.

Examples:
[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

JavaScript

There are also javascript files that have low expiration headers. These are coming from the [redacted] with only 60 minute expiration definitions.

[redacted]

[redacted]

[redacted]

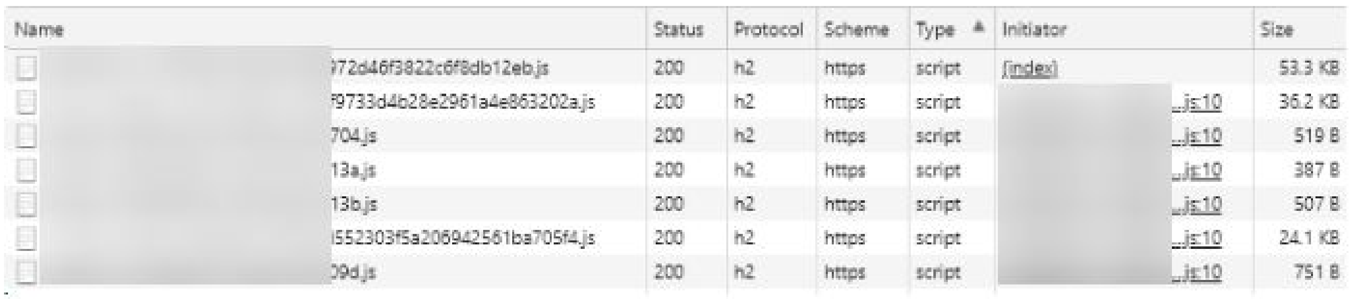
[redacted]

[redacted]

[redacted]

[redacted]

These scripts add up to 116 kilobytes and will be re-downloaded every hour.



Name	Status	Protocol	Scheme	Type	Initiator	Size
972d46f3822c6f8db12eb.js	200	h2	https	script	(index)	53.3 KB
f9733d4b28e2961a4e863202a.js	200	h2	https	script	...js:10	36.2 KB
704.js	200	h2	https	script	...js:10	519 B
13a.js	200	h2	https	script	...js:10	387 B
13b.js	200	h2	https	script	...js:10	507 B
i552303f5a206942561ba705f4.js	200	h2	https	script	...js:10	24.1 KB
09d.js	200	h2	https	script	...js:10	751 B

Recommendation One

Set expiration headers for these static assets (images, javascript) to a far off date like 6 months or 1 year.

Difficulty: Medium **Priority:** High

Eliminate render-blocking resources

To create a visible web page, browsers have to assemble the HTML, CSS, javascript and everything else. Most javascript interrupts the critical rendering path: If a browser finds a javascript, it has to stop everything else until the javascript is loaded and executed.

We found these render-blocking resources on the homepage:

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/1.12.3/jquery.min.js"></script>

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

On an individual page, we found these render-blocking resources:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/1.12.3/jquery.min.js"></script>
```

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

Defer or Load Asynchronously

Deferring or loading asynchronously is the solution for handling render-blocking resources. Note that not all resources can be deferred or loaded asynchronously -- it depends on their importance in your application.

Defer

The simplest way to defer resources is to put them at the bottom of your HTML source code.

For javascript, the *defer* attribute can also be utilized. For example:

```
<script defer
src="//[redacted]
[redacted].js"></script>
```

Async

For javascript, the *async* attribute can be utilized. Keep in mind that scripts loaded asynchronously fire as they are loaded, ignoring any procedural order. So, if you require certain assets to be loaded in order, utilize a defer loading method.

```
<script async
src="[redacted]
[redacted].js"></script>
```

Recommendation Two

Eliminate render-blocking resources by implementing deferred or asynchronous loading techniques.

Difficulty: High **Priority:** High

Remove/Defer Unused CSS and JavaScript

Code coverage analysis shows a large amount of unused CSS and JavaScript. Removing or deferring as much of this as possible will aid in page load times and JavaScript execution time.

Findings

Homepage

[redacted].css: **95.1% or 1.68mb unused**

[redacted].js: **90.1% or 1.47mb unused**

[redacted].js: **73.5% or 190kb unused**

[redacted].css: **92.8% or 125kb unused**

[redacted].js: **98.4% or 101kb unused**

[redacted].js: **98.9% or 54kb unused**

[redacted].js: **39.8% or 47kb unused**

[redacted].js: **50.5% or 34kb unused**

[redacted].css: **94.5% or 27kb unused**

[redacted].js: **58.0% or 20kb unused**

https://www.[redacted].com/ (inline CSS+JS): **54.1% or 18kb unused** [redacted].js: **75.2% or 17kb unused**

Total savings: ~ 3.8 mb

Location Hub

[redacted].css: **96.1% or 1.77mb unused**

[redacted].js: **93.2% or 1.52mb unused**

[redacted].js: **77.5% or 200kb unused**

[redacted].css: **99.1% or 124kb unused**

[redacted].js: **98.4% or 101kb unused**

[redacted].js: **98.9% or 54kb unused**

[redacted].js: **38.8% or 46kb unused**

[redacted]js: **50.5% or 34kb unused**

[redacted].css: **94.7% or 27kb unused**

[redacted].html (inline CSS+JS): **60.4% or 21kb unused**

[redacted].js: **58.0% or 20kb unused**

[redacted].js: **75.2% or 17kb unused**

Total savings: ~3.9mb

Location page

[redacted].css: **91% or 1.61mb unused** [redacted].js: **88.9% or 1.45mb unused**

[redacted].js: **77% or 199kb unused**

[redacted].css: **96.4% or 120kb unused**

[redacted].js: **63.5% or 65kb unused**

[redacted].js: **85.5% or 64kb unused**

[redacted].js: **98.8% or 59kb unused**

[redacted].js: **98.9% or 54kb unused**

[redacted].js: **38.8% or 46kb unused**

[redacted].js: **98.3% or 40kb unused**

[redacted].html (inline CSS+JS): **56.1% or 37kb unused**

[redacted].js: **46.6% or 32kb unused**

[redacted].j
s: **97.4% or 26kb unused**

[redacted].css: **88.7% or 26kb unused**

[redacted].js: **93.9% or 23kb unused**

[redacted].js: **55.4% or 19kb unused**

Total savings: ~3.87mb

Recommendation Three

Remove or defer as much unused CSS and JavaScript from each page/template as possible.

Difficulty: Medium **Priority:** High

Lazy-loading Imagery

Usually, a browser loads every asset on the page, all at once. So, if you visit a page with lots of below-the-fold images, you download every image upon visiting that page.

Lazy-loading below-the-fold images after critical resources have finished loading will lower the time it takes until a user can interact with the page. It also more efficiently uses the pipe and improves the browser experience. Here's how it works:

1. You visit a web page
2. The page loads visible, above-the-fold images first
3. The page loads the remaining images as you scroll down

Recommendation Four

Implement a lazy-loading imagery solution.

Difficulty: High **Priority:** Medium

Next-gen Images

Next-gen image formats like webP, JPEG 2000, and JPEG XR, are typically more compressed and thus smaller in size and faster to load. This is not a replacement solution, but rather an “in addition to” solution for your current images. This is because not all next-gen formats are supported by major browsers.

WebP

The WebP image format is supported by Google Chrome and Opera, which make up a very large percentage of your visitor base. It also has solid lossy and lossless compression and thus we recommend implementing WebP imagery.

Before putting too many resources behind implementing, be sure your hosting environment is capable and configured to serve webP.

Recommendation Five

Implement the next-gen image format, WebP, on your images.

Difficulty: Medium **Priority:** Medium

JavaScript Loading & Execution

This analysis finding goes in hand with our recommendation for removing unused JavaScript. The more JavaScript included on the page, the more time browsers spend parsing, compiling, and executing it.

Homepage: **2.1 seconds**
Location Hub: **2.2 seconds**

Single Location: **8.7 seconds**

The single location page is the most problematic in its execution time, while the homepage and location hub are respectable.

Consider analyzing and reducing unused scripts as a first measure. Additionally, with the advent of HTTP/2, which your site is utilizing (that's good!), breaking large JavaScript files into smaller snippets can help improve execution.

Additionally, work with your analytics team to take a hard look at all 3rd party scripts that are being included on the site, either directly in the source code or via Google Tag Manager or the like. These scripts are often major culprits in poor page loading times.

Recommendation Six

Analyze and remove as much unused JavaScript as possible. This includes 3rd party scripts! Also, break out large JavaScript files into multiple smaller files for faster parallel execution.

Difficulty: Medium **Priority:** Medium

DOM Size

The pages on [redacted].com have, on average, a really large Document Object Model (or DOM) size -- especially single location pages.

Homepage: **4,389** nodes
Location Hub: **5,623** nodes
Single Location: **23,931** nodes

Having a really large DOM size has been reported to cause longer CSS analysis and rendering, increase browser memory usage, and affect reflow of layouts. According to Google's PageSpeed Insights, browser engineers recommend that pages contain fewer than approximately 1,500 nodes.

Recommendation Seven

Analyze the DOM and simplify nodes where possible to reduce overall DOM size.

Difficulty: Medium   **Priority:** Low

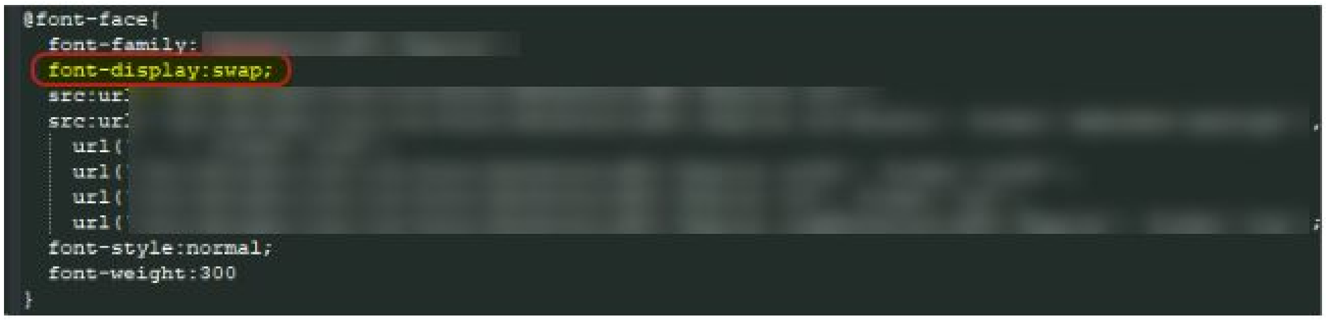
Font Loading

While the web fonts being used on the site look really nice, the downloading and rendering of them can create a poor user experience. All users who do not already have your fonts downloaded will have to do so as a first step when arriving on the site. This is all normal and expected. Most major browsers have timeout logic with loading web fonts. Firefox and Chrome have 3 second timeouts where Internet Explorer has 0. If the web font hasn't loaded within that timeout period, the fallback font is used, and once the main web font has loaded, the browser re-renders the text.

A newer *font-display* attribute has been created for *@font-face* that gives some control to the developer for performance and user experience enhancements. The possible values for this attribute are: auto, block, swap, fallback, optional.

For more information on each of these attributes, please visit the specifications page: https://www.w3.org/TR/css-fonts-4/#font-display-desc

We recommend utilizing the *swap* or *auto* values when defining your web fonts. Here is what it would look like for one of your fonts found on all your templates:



```
@font-face{
  font-family:[redacted]
  font-display:swap;
  src:ur[redacted]
  src:ur[redacted]
    url([redacted]
    url([redacted]
    url([redacted]
    url([redacted]
  font-style:normal;
  font-weight:300
}
```

Recommendation Eight

Implement the font-display descriptor on @font-face definitions.

Difficulty: Low **Priority:** Low

Priorities & Next Steps

The final page of this document contains a table with all of the recommendations made in this document, sorted by priority level.

Please let us know if you have any questions. Thank you!

Table of Recommendations

Recommendation	Priority	Difficulty
Recommendation One	High	Medium
Recommendation Two	High	High
Recommendation Three	High	Medium
Recommendation Four	Medium	High
Recommendation Five	Medium	Medium
Recommendation Six	Medium	Medium
Recommendation Nine	Medium	Low
Recommendation Fifteen	Medium	Low
Recommendation Seven	Low	Medium
Recommendation Eight	Low	Low