GYMNASIUM

RESPONSIVE WEB DESIGN FUNDAMENTALS

Lesson 6 Handout

Performance

ABOUT THIS HANDOUT

This handout includes the following:

- A list of the core concepts covered in this lesson
- The assignment(s) for this lesson
- A list of readings and resources for this lesson including books, articles, and websites mentioned in the videos by the instructor, plus bonus readings and resources hand-picked by the instructor
- A transcript of the lecture videos for this lesson

CORE CONCEPTS

1. The performance of a website may be just as important, if not more so, than any layout or design decisions. Research has shown that you have about five seconds to get content on the screen before losing 75% of your audience that are visiting on mobile devices.
2. Their is an important difference between “actual” and “percieved” performance to be aware of. Actual performance is the amount of time it takes for all the assets on a page to be completely loaded. Percieved performance is how long it takes for the user to act on your content or to see that something is happening on screen (even if the page is not completely loaded).
3. Understanding how your page is loading and how the various assets such as images, JavaScriot and Web Fonts are involved can be partially accomplished by using Browser Developer Tools. In this course we used Chrome developer tools and specifically the Network tab in order to help gain insights on the various components of the page.
4. Once you have an identified components of your site that may be “roadblocks” to your content, such as references to third-party JavaScript libraries or unnecessarily large images, then you can begin prioritizing them and coming up with solutions.
5. Online tools such as webpagetest.org and whatdoesmysitecost.com are important resources that can help you identify and prioritize performance issues on your sites.
6. Images are a common culprit when it comes down to slow performance. In addition to the responsive techniques covered in Lesson 05, you will also want to reduce all your images as much as possible without sacrificing image quality significantly. Tools such as reSmushit and imageOptim are very useful for this.
7. Another important technique for improving performance is a process called *minifying*, which involves reducing the file size of your CSS and JavaScript through techniques such as reducing whitespace, stripping comments and other optimizations. One handy online tool for this is Minify.

ASSIGNMENTS

- Quiz

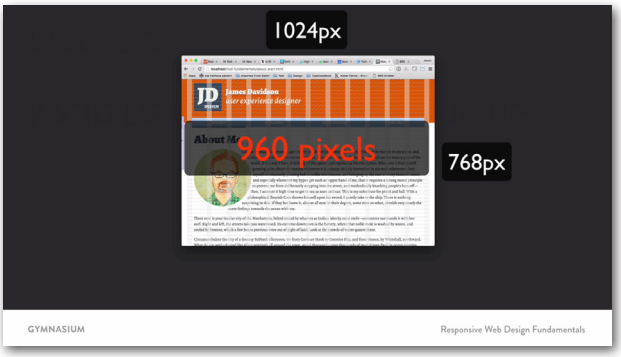
INTRODUCTION

(Note: This is an edited transcript of the Responsive Web Design Fundamentals lecture videos. Some students work better with written material than by watching videos alone, so we're offering this to you as an optional, helpful resource. Some elements of the instruction, like live coding, can't be recreated in a document like this one.)

Welcome to lesson six—Performance. In this lesson, we're going to take a look at a few things to tidy up everything you've been working on and get it serving as fast as we possibly can. We're going to start off by looking at why Mobile First, and what sorts of benefits that will give you in the long run, tools for testing and finessing your site, looking critically at what you're loading, how you're loading it, and some further research that you can use to dive into the intricacies of performance and see what's coming down the big pipe.

CHAPTER ONE: WHY MOBILE FIRST

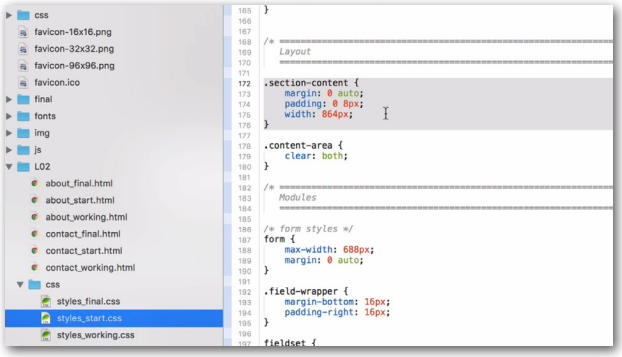
In this chapter, we're going to talk a little bit about attention span, different kinds of performance, user trends and behavior patterns, and why this is going to result in a benefit for all your users—not just ones on slow mobile connections. Research has shown that you have about five seconds to get content on the screen before losing 75% of your audience that are visiting on mobile devices. And given that mobile devices are really taking over what people are looking at and when, it's really important that we get this right, because even more recent research says that you've got about three seconds before losing over 50% of your audience.



Time's ticking, attention spans are getting shorter, and expectations are getting higher. So I want to talk about actual versus perceived performance. The two important things to consider here are how long are people waiting to get the whole page and how long are people waiting to be able to act on your content. Sometimes, it's just getting something on the screen that makes the most difference, so people can see that something's happening.

So in thinking about the difference between actual and perceived performance, let's take a look at their responsive web design site put together by Ethan Marcotte and Karen McGrane. So I'm just going to hit Load here. And I'm going to go somewhere else, so you can see. I'm going to go over to the Workshop page. And I'll click around a little bit. And this is in real time, just over average Wi-Fi. And you can see things are showing up pretty darn quickly.

Now, what I want to look at and show you is if we were to inspect this and throttle our bandwidth connection—so using the Network tab on the Developer Tools in Chrome. We've done this before. I'm going to slow this down to a regular 3G connection. Let's make this Windows small, and let's start clicking around again. So their response time is obviously a little bit slower, because we've slowed down the actual data connection.



So clicking around on some of the links that we have in here, you can see that the page loads, even on a slow 3G connection, are coming up really fast—even when you've got a lot of content on the page like the podcast site. So one of the things that they're doing here is obviously being minimal about the content they are loading. If you scroll down, there's not a lot of image content. But they're being very creative about how they're using it.

But he's doing a lot of things to unblock the actual rendering of content, so that things can get on the screen quickly. And then styling can come in after. So that's something that's really important to consider—that acting on that content is the most important thing. The design actually coming in more completely is a bonus.

So coming back to this notion of actual versus perceived performance, when we think about the amount of traffic that's coming to your website and where it's coming from, one of the things here in the US that we have to consider is that we've reached past that tipping point, where over 50% of the traffic on the web is actually coming from a mobile device. And when you think about that communication starting out with email, you need to remember that about 75% of all email is opened first on one of those mobile devices, upon which they'll click things. I promise you they will.

And they're are going to come to your website. And if they don't get that content on the screen, they're going to just go play Angry Birds or something else. Coming back around here full circle, we have to remember that people are going to be using a variety of devices. They have a very short attention span, and it's only getting shorter. It's far more likely than it ever has been before that link, that content, that first contact, is going to be through a mobile device.

And that is the point at which you have a mirror few seconds to capture their attention, and get them engaged in your web site before they just go away. Now, if you think back to the lesson on loading the web fonts, we know that we've actually already done some work on perceived performance. We've set things up with a web front loader so that we know that the web fonts are going to be loading, but the content will display on screen.

That was the whole idea behind adding font face observer. And the last lesson, we also went through responsive images. Were actually going to do a test comparison on that pre-optimized page from the beginning of lesson five and compare that to how it looks when we're done optimizing everything in this lesson. And we'll see just how big a change that can be.

CHAPTER 2: TOOLS FOR TESTING AND FINESSING

```
}

.main-nav {
  background-color: #B7D0E8;
  background: -moz-linear-gradient(top, #B7D0E8 0%, #8BB1DD 100%);
  background: -webkit-linear-gradient(top, #B7D0E8 0%,#8BB1DD 100%);
  background: linear-gradient(to bottom, #B7D0E8 0%,#8BB1DD 100%);
  -webkit-box-shadow: 0 1px 3px 0 rgba(0,0,0,0.5);
  -moz-box-shadow: 0 1px 3px 0 rgba(0,0,0,0.5);
  box-shadow: 0 1px 3px 0 rgba(0,0,0,0.5);
  display: block;
  margin: -20px auto 0 auto;
  float: none;
  padding: 0;
  position: relative;
  top: 0;
  max-width: 864px;
  width 90%;
  z-index: 15;
}

.main-nav[aria-expanded="true"] {
  display: block;
  top: 0;
  position: relative;
}
```

So let's move on to chapter 2. Let's look at tools that we have for testing and finessing our site. In this chapter, we're going to take a look at browser tools that we have at our disposal—some of which we've used before, some of which we'll learn a few new things. We've got external testing tools that we're going to look at, and we're going to talk about performance budgets.

So I've loaded up the starting version of the About page from chapter 6. And looking through, it looks pretty much just like it did before. It has the responsive images built in, but it does not have any other kinds of performance optimizations besides the web fonts that we've already set up.

And what I want to do is right click—or Control-click, depending on your setup—and Inspect, and that opens up our browser tools here in Chrome. I'm going to click on the Network tab, and then Command-R to reload. And you'll see that we get a little waterfall view of all of our content that's loading, and you can see a list of all the files.

Now, I'm pointing this out because this gives us the opportunity to see what's loading when and do a little bit of testing as we optimize things. We're able to learn an awful lot by simply looking at—we can click on these buttons and see which files are loading—JavaScript, CSS, images, fonts. All of these things allow us to get a good sense of exactly what's going on during this load process and see which things are taking longer and how we might be able to optimize just a little bit more.

So browser tools are really our first line of defense as we start to think about performance and what's going to take time as we load. And this Network View giving us this little waterfall is incredibly helpful. Now, there's another great article from Paul Irish, who's a developer evangelist over at Google.

He's done a lot of writing and speaking about developer tools and really worked hard to make sure that we've got some great things at our disposal. So I'm going to give you the link in a sidebar so you can go check that out and learn a little bit more about it.

Now, another thing that we have at our disposal are some external tools. So I'm going to open up another tab here. I'm going to go to WebPageTest.org. Now, this site has been put up as a way for us to do a little bit of our own experimentation to see exactly how a site's doing and get a grade—see which things are working well and which things could be improved.

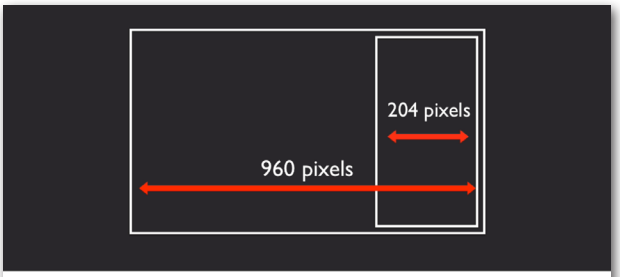
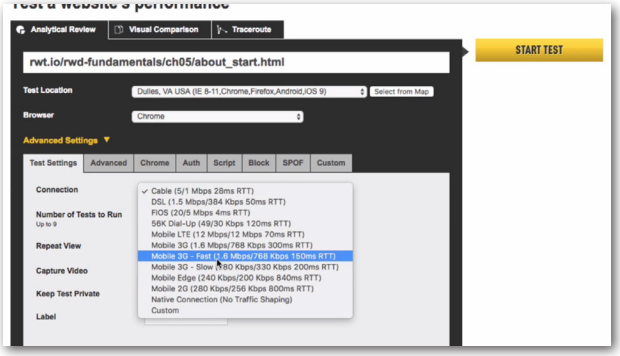


Illustration of formula concept

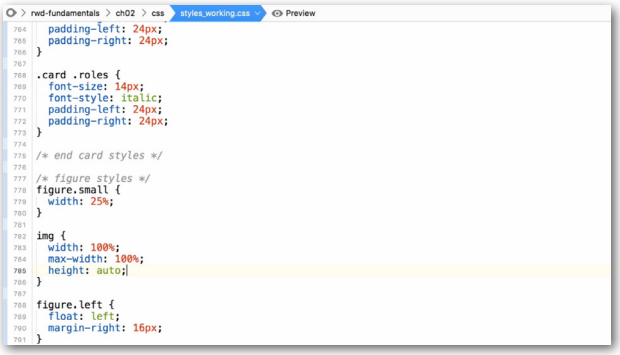
GYMNASIUM Responsive Web Design Fundamentals

So I've copied the link to a page that I have hosted that's going off my site to take a look at the sample code from chapter 5—at the starting point before we did our responsive images. And if you take a look at what's available here, you can choose different test locations. You can choose different browsers.



And then if you go under Advanced Settings, you can also choose speed. So if we take this, we can choose different connection speeds—similar to what we were doing in the Network tab in our Chrome developer tools. So I'm going to go ahead and run a test, and we can take a look a little bit.

And usually it goes pretty quickly. Sometimes it might take a minute or two, depending on how many other people are running tests at the same time. And what this is doing is it's running through a whole series of tests, loading that page. You can actually have it perform the test a number of times.



But in any event, what we get here are some overall grades up top, which you have to take with a grain of salt and see which things you can optimize and which things you can't. But overall, it doesn't look too badly. And we're going to start to look at some of these bits of information.

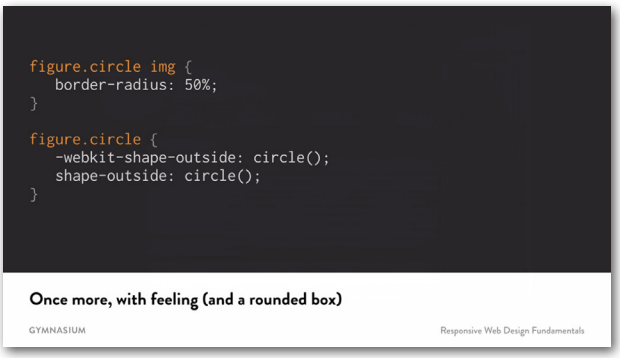
Generally speaking, we want to look at—given our short attention span issues—what is the start render time? So at what point does the user typically start to see something on the screen—so the first view of the page? It looks like it's about 1.3 seconds.

You get a general speed index, and a lower number is better. This is already pretty low. Given that it's a small, static website, that's not that surprising. It should be pretty fast.

But you get an idea of—how many requests are there, and how many bytes are being loaded? So this is loading about 1 and 1/2 megabytes on this page. And all told, it's taking about 3.73 seconds to load.

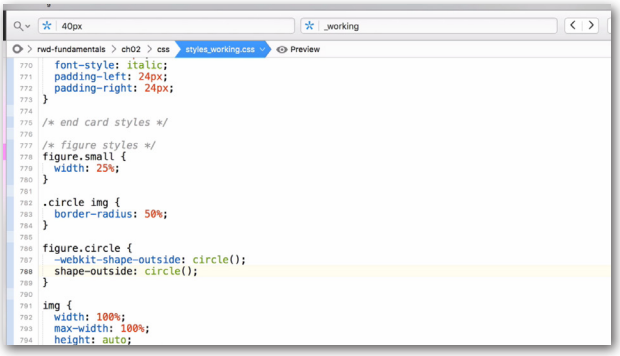
Now, you also see a little dollar sign cost here. If we click on that, this is really interesting. And it's something that I think is really important for us to keep in mind. What does it actually cost to view this site?

Now, I live in the Boston area. And so this area is really well covered in 4G most of the time. I've got a good iPhone. I've got Wi-Fi at home and in the office. Generally speaking, my connection life is good.



But that's something that's not really the case for a lot of users. Even on the train coming up to Boston, there are several places where I'm down to one bar. So even if you are in a well-covered area, you still have to think about—what is the speed and latency like, and what does the data cost?

So if we take a look here, you can see that it's actually an expensive page to look at. In Canada, it costs—on an average data plan, pay-as-you-go in Canada—it costs $0.18 to look at that About page. So if we factor that across a few page loads, somebody coming to our website to look at our portfolio here is going to spend about $1 in data costs. That's asking a lot of people coming to visit your website.



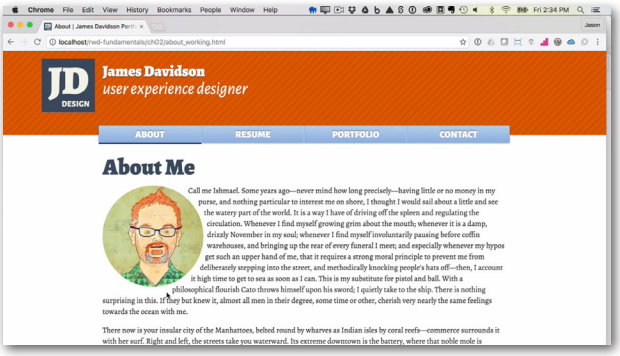
Thankfully, if we look at the typical costs for data in the US, it's about half that. In other countries—interestingly enough, some of the lowest data costs are some countries in Africa and Egypt. So if you look in India, for example, average pre-paid data cost to look at this is a penny.

So it's interesting. It's not always a one-to-one comparison. Data costs and network speed are not always on a direct line together. But there are factors that you really want to consider, especially if you need to think about where your audience is geographically, especially as it relates to a much more global economy that we have.

So these are some really important tools. We've got Web Page Test. What does my site cost? If you want to get some other measures of performance, there's another website called GTmetrix. I'm going to load that one up, as well.

And this is another one where you can just paste in the URL, hit Analyze, and it's going to go through—this one, without a paid account, you end up just having to take whatever test server region and browser they give to you. You don't have nearly as much say in that. But it does give you some really good information.

So it generates this report, and you can see PageSpeed score's not so great. The YSlow score—also not so great. It gives us a bunch of information to look at.



Now, there's a couple of things here—Enabling gzip compression—that's something that happens on the server. Since we're just setting this up for ourselves, we may not be able to get that same kind of information. But at least if we factor that out, there's other things that we can do to make this work well.

Namely, serving scaled images, because we don't have responsive images factored in here. And we can make sure that things are being served from a consistent URL, because we've got some things being loaded from one source and some being loaded from another. So those are a few things that we can take a look at here to optimize things.

Now, there are also some other things that we can do to reduce the amount of file size in our CSS and JavaScript, and also optimizing our images. So we're going to take a look at some other tools there. So there's one more thing I'm going to show you for optimizing images.

So I'm going to open up another tab here and go to reSmush.it. Now, Smush.it was a tool that was developed by Yahoo! a number of years ago and was kept online for a very long time and was eventually shut down. This has been the relaunch of it with the same codebase. The API works the same way.

And what this site does—what this service does—is it performs a number of optimizations on any images that you pipe through. This is using a bunch of open source tools all stitched together. And with this site, there are a number of integrations.

So if you're working with Drupal, WordPress, Magento—the point here is that there are a bunch of different ways that we can put tools in place to ensure that we have the most highly-optimized files that are being served. And that means little bit better compression, removing extraneous data, reorganizing the bytes in the file—all of those things help serve those images significantly faster.

And there are some tools that you can use on the desktop, as well. We're going to take a look at one of those called ImageOptim that's good on the Mac, and we'll look at some other tools that you can download for other platforms, as well.

So one more tab here—and we're going to look at Imageoptim.com. This is a free tool that you can download and use as a desktop app on your Mac. I'll show you how that works here.

```
<script src="path/to/jquery.min.js"></script>
<script src="path/to/jquery.fitvids.js"></script>
<script>
  $(document).ready(function(){
    // Target your .container, .wrapper, .post, etc.
    $("#thing-with-videos").fitVids();
  });
</script>
```

And then if we take a look at the Other Versions link, this gives you stuff for older versions of the Mac OS, Linux, command line tools, as well as Windows. Different versions, different applications—ImageOptim is only for the Mac, but this does give you some other really easy tools to use.

Now, the last thing I'm going to show you in this chapter is a performance budget. A performance budget is a way for you to quantify how well or poorly different websites are performing and why—using the same tools that we were just looking at. So this is an article from Dan Mall.

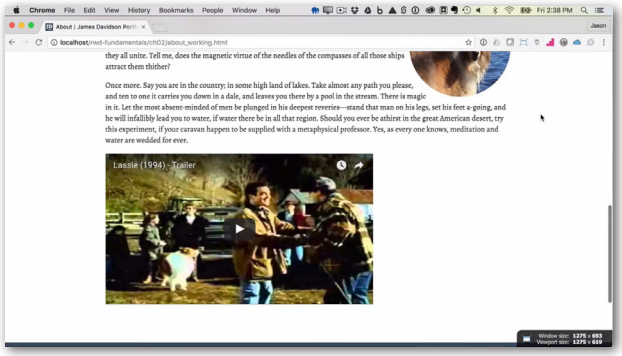
It's got some great information in here. We'll include a link to this in the notes. I really encourage you to read through this and follow some of these links to get a really good understanding.

But basically, thinking about those numbers and metrics I was just talking about when we were looking at it, it lays out a really solid case. Now, this is something that's interesting for you to do to make comparisons with your own project, but when you start to have conversations with a client, it's really important that you think through performance first, because ultimately, in order for a design to function—to convey that idea and to influence behavior—it has to show up.

So anything that we do to design a site effectively has to start with performance. So we analyze first. We figure out what our targets are.

We want to make sure that we're going to deliver a site that is noticeably faster than whatever came before, and ensure that all of those constraints are in place that give us the boundaries within which we need to plan for—how many k of fonts are going to be downloaded? How many images can we show on the screen? How much do we have to allow for for the different JavaScript libraries that we're loading?

All of those factors come into play in ensuring that we can deliver a really fast site. Now, if you scroll all the way down to the bottom, you'll see Dan's highlighting a bunch of the different aspects that people might want to track and how you might break these things up in measuring what's being loaded on your page. As we get all the way down to the bottom, he's got a nice, handy link to a Google Sheet template.



So you can see the sample that he's developed here. And I also have another one that I added in a couple of other columns. I was analyzing some different library websites. I wanted to keep track of speed index and the number of bytes loaded, as well so I could start to make some comparisons and recommendations for just how much we could allow on a given page for amount of images, amount of HTML and CSS, JavaScript and web fonts to guide that whole design process.

So now we're going to see exactly how we can pull all of these things together and start to optimize our own project. So we've learned how to test. We've learned some different health checks we can put the site through. And now we're going to start to figure out exactly how to apply this to our own site, quantify the things that we can change, and start to put a plan into action.

CHAPTER 3: WHAT YOU'RE LOADING

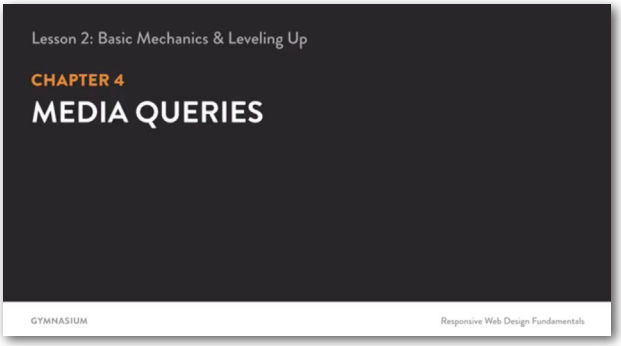
```
/* Flexslider */
$(window).load(function(){
  try {
  $('.flexslider').flexslider({
    animation: "slide",
    controlNav: "thumbnails",
    start: function(slider){
      $('body').removeClass('loading');
    }
  });
  } catch(e) {
    return;
  }
});

// Basic FitVids Test
$(document).ready(function(){
    // Target your .container, .wrapper, .post, etc.
    $(".video-container").fitVids();
});
```

In this chapter, we're going to catalog all the external assets that we're loading, and we're going to take a look at that loading timeline and see exactly what impact that's having on both the actual and perceived performance of our website. We're going to talk a little bit about page rendering and exactly how that works and what it is. So I'm back in the browser with the starting version of our page.

So I'm going to open the Web Inspector and click on Network. And hit Command-R so we can reload everything again. And take a look at this waterfall and see where it is things are coming in along this rendering path.

And you see on this list, starting from the top—it's loading the HTML. It's then loading in all these other little polyfills and images, web fonts, the icon fonts, JavaScript, the icon again, and all these little bits that go into showing the page and loading all the icons. So when we think about this, this is our sum total of all the assets that are being loaded.

Now, a lot of these things are pretty small, but some of them could be smaller. And we could combine some things to create some optimizations that will help things run just that much faster and reduce the number of separate requests. We go back over to our code, and we're going to open up the starting version of that About Page.



And we're just going to scroll through and take a look. So our icons are loading. Those are fine. We don't have to worry about that. There's—really pretty minimal.

So we get down here, you can see we're loading a number of different files. Now, the Normalize and Setup CSS files are minified, but they are separate requests. Now we have the icon font loading from a separate source. But it is on a content distribution network, so that should be pretty fast.

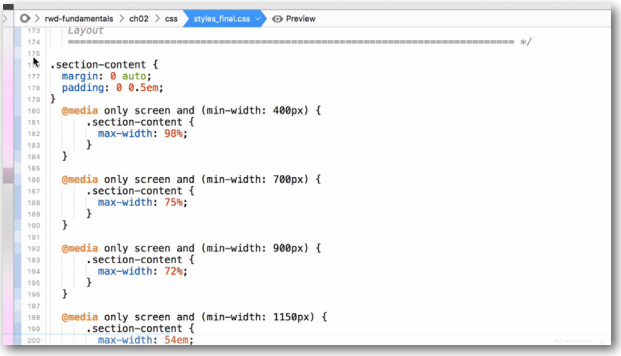
We've got our web fonts loading, and then we've got a few different JavaScript files. We've got Modernizer, Font Face Observer. And then if we scroll all the way down to the bottom, well, we can't forget Picturefill—just in case we need to support with older browsers.

And down at the bottom, we're loading in jQuery. Now, the way this is working is it's going to load it from a jQuery CDN. So if you've been to another website that's loaded it from the jQuery site, they probably won't have to load it again. This next line actually looks to see if it's loaded. And if not, it then writes out, using JavaScript, a line to load it locally.

So this is just a way to minimize the need to load it again in case it's already been cached by the browser. Then it's grabbing a plug-ins file, Widowtamer to fix up our typography a little bit, a classless polyfill for browsers that don't support that in the native JavaScript, and then our main JavaScript file that handles tasks for navigation and that sort of thing.

So there's a bunch of different assets here. And we can't forget that if we look back in the middle, we're loading a bunch of different image assets for the photo of Tristan, the sunset—or a sunrise, rather—and the other little illustrations in the logo. So we've got a bunch of different assets that are being loaded—images, JavaScript, stylesheets. Those are the ones that we're going to focus on.

Coming back over to the browser, we're going to go look at our web page test and take a look. And if we click on this little waterfall graphic, you can see there's a couple of things that are blocking elements.

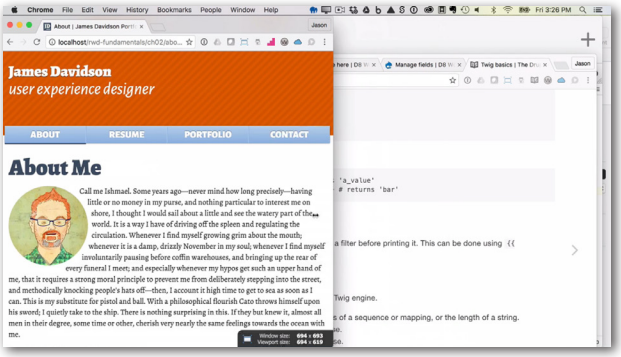


So you can see—during this loading process, we've got a bunch of font files that don't load until everything else has loaded. And then we've got image files that have been blocked until all of these other elements are loaded in, and

so forth. Here you see JavaScript being loaded later. All of these things adding up to delays in getting your page to load.

Now, what we want to do is remove some of those blockers. There's a few different ways that we're going to be able to go about that, based on how well we compress things and also how well we set the timing with which things are acted upon. So we've spent a bunch of time looking at this waterfall graphic of showing the order in which things are loading.

And what this really is getting at at the heart of it is—what is this page rendering process? Now, going back over, looking at our page and inspecting this—again, going to the Network view and loading it. When we talk about the page rendering process, it has to do with how things are received by the browser and in what order and how it can act on them.



So you can see it starts off by loading the HTML page, and then it moves on to whatever order things are listed in the page—loading in some CSS, loading in some JavaScript, loading in web fonts. Now, what you have to understand is there are certain things that can be loaded asynchronously, and certain things that are considered blocking events.

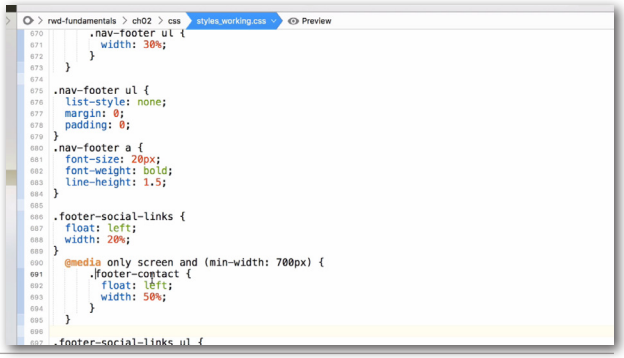
In the page rendering process, CSS and JavaScript are—by default—blocking events, meaning that when the browser gets to that point in the file, it won't do anything else until it has loaded the entire file, can process it, and then move on. Now, stylesheets—there's not too much you can do about that. There are some advanced techniques that you can play around with.

However, most of the time, you want to make sure that your styles have loaded in order to move on with laying out the page. JavaScript, on the other hand, very often doesn't need to be loaded immediately, as long as it takes care of firing off the events that it needs to in order to enable the behaviors that you want.

It used to be very common practice to import the CSS and JavaScript at the same time in the head of the page. It turns out, most of the time, it's just fine to shift the bulk of the calls to JavaScript down to the footer, which is what we've been doing all along. We're going to look a little bit at why some are at the top of the page and why some can come later.

So now we've cataloged everything that we're loading and we've talked about the impact of different assets on that page rendering process. Let's move on to seeing what we can do about it.

CHAPTER FOUR: HOW YOU'RE LOADING IT

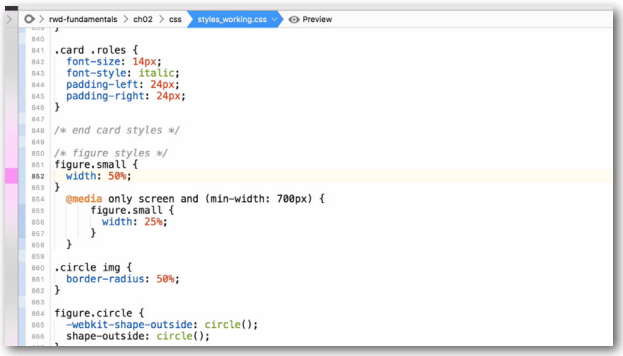


Now we're going to get into some code. We're going to figure out how to unblock that path to page render, and we're going to figure out what we can optimize, what we can compress, and what we can delay.

So digging back into our code, if you look through this folder you'll notice that this time, we've got everything in this

directory here. We've got, in the CSS folder, we've got our CSS start and working files and final versions. In JavaScript, we've got them here. We've loaded our images in. Everything is here so we can focus on seeing what it is we can optimize.

So we're going to start off by thinking through what are the aspects that we want to hit first. We're going to open up this working file and we're going to look again at what's in this about page. First off, we're loading three different CSS files here, normalized, set up, and the working styles. First thing we could do is simply combine them. The second part is minify it in order to remove all the extra white space and comments, and that will help give us a much more compact file.



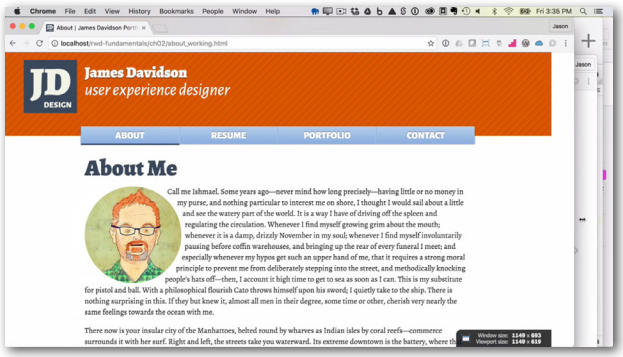
So let's open up our working CSS file. Now this is only the site files. Now what we want to do is actually bring in both of the other files. So we take a look and it's back out in our base CSS directory. And we want to get normalize and the setup file. We scroll down, open up this folder, I'm going to open up normalize, and we're simply going to copy this and bring it into our working CSS.

And I'll put a comment up here that this is our normalized CSS. Make sure we close our comment. And then we want to get the set up file and do the same. Grab this. So we have this extra chunk of CSS in here. Now this is part of what we want to do. We still want to minimize it.

```
figure.small {
   width: 50%;
   -webkit-transition: all linear 0.2s;
   -moz-transition: all linear 0.2s;
   transition: all linear 0.2s;
}

@media only screen and (min-width: 700px) {
   figure.small {
      width: 25%;
   }
}
```

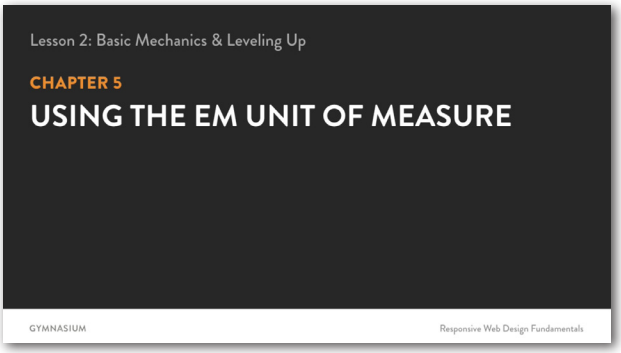
So I found a website that we can use. If we go to minifier.org, it's a really simple little web app that just gives you a box that you can paste stuff into, click a button, and get back minimized code—minified code. So I'm going to Select All, Copy, Paste. Make sure you select CSS. Go ahead and minify it. And you can see it just makes everything more compact. It removes all the unnecessary whitespace. Select everything in that box and go back over here, Paste.

Now I'm going to go ahead and save this as a minimized file. Now all this is doing is really, it's just a hint to myself. So I'm going to stylesworking.min. And this is going to save in that same folder. Now what we can do on our About Working page, we can change this to .min and remove the other two references. And we must reduced by quite a bit the amount that we're loading.



Now if we go and look at our working file that was unchanged and we take a look, it's 33k. And scroll down, we're going to do the same to this minimized file, that one's 4k. And this one is another 4k. So we've got about 41k total CSS together there. And our minified file here is only 25. So we've almost cut it in half just with that one optimization.

So we've removed two requests. So it's two fewer requests going back to the web server. And we've cut the size nearly in half. Now notice I'm skipping over a couple of things we're leaving in the head here. There's some JavaScript from Moderniser and Font Face Observer. We're leaving those here at the top of the page because we need those to load in order to style the page properly and start the process of loading the web fonts.



So we're going to leave those alone. And we're going to go back down to the bottom of the page and we're going to take a look at the JavaScript that we're loading. We have our main JavaScript file. And then we also have plug-ins, then Widowtamer, then Class List. And there's one more that we use on the Portfolio page. So we're going to get all of these and make them a part of what it is that we're loading.

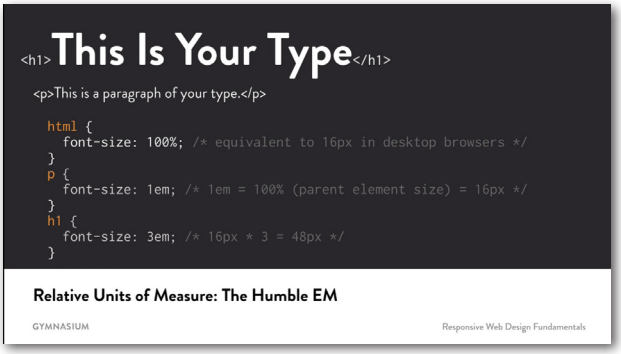
So the first thing, we'll open the plug-ins file. We'll copy this, put that up at the top. We know we needed Widowtamer. And bring that one in next. The next was Class List. We grab this one. Let's go take a look, see if there's anything else.

Now the one other one that I know that we load, if we take a look at the Portfolio Project page, we know we're using FlexSlider there. We want to make sure we add this one in, too. So it's jquery.flexslider-min. Where So let's take a look in here. jquery.flexslider. We'll select all of this. Again, go back to our main file.

Now before we were only loading that on the Portfolio page. However, when you take the time to minify and aggregate everything, sometimes it's better to just go with that one minified request even though you're not using it on all the pages. I think this is one of those cases. And we're also going to go back and add that into the CSS because I forgot to do that before.

So we paste this in here. We've got all the JavaScript that we might possibly be using. Now I'm going to save this as a working file so we don't overwrite the main one. Leave that in the same place. And now we're going to do the same thing we did before. I'm going to select everything, go back over to the browser, paste it in this box—we know it's JavaScript—hit Minify, select everything, and paste it in here and save this as the minified version.

So we now have our one JavaScript file that we can load. Get our naming right—there we go—just so we're consistent. And now we can go back into these pages and know that we're loading working.min. And that means we can get rid of all of these other lines here. And go ahead and save.



So if we look again, let's see, if we look at our working file, it's 16k of JavaScript and the minified version is down to 12k. So not as big a savings here, but every little bit counts. We've removed three or four other calls and we've reduced the file size. So everything taken together is going to make a huge difference.

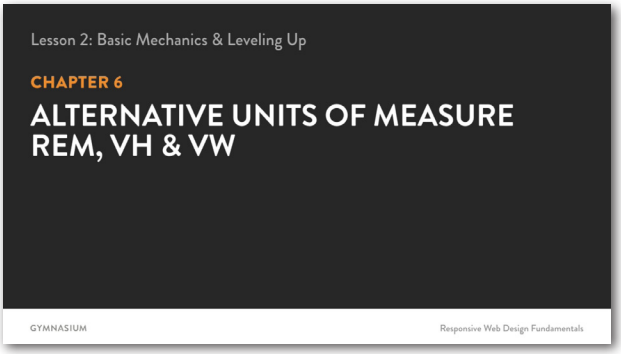
We've optimized this as best we can for the external CSS and JavaScript. We need to make sure we do the same thing on all the other pages of the site. So I'm just going to do it to this other one here at the moment. We know we have all of these factored in. But on this Portfolio page, we're using FlexSlider. Remember to go back up to the top here. We're loading some FlexSlider CSX here that we need to make sure that we catch.

So let's go back to our root CSS folder, grab our FlexSlider CSS, all of that, and we go back to our working CSS file. It's a good thing we saved that. And we put this in right above the site styles, save it, select all copy, and we'll minify this again. And go back and add this to our working file.

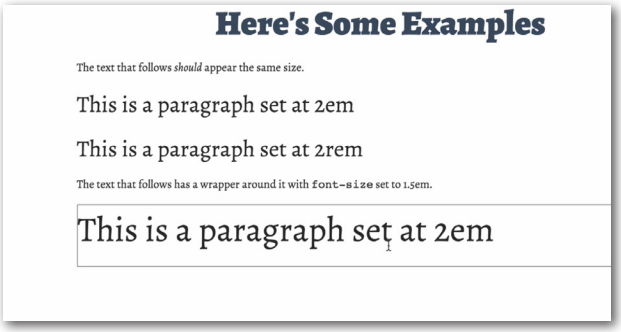
```
.section-content {
   margin: 0 auto;
   padding: 0 0.5em; /* 8 / 16 = 0.5 */
   max-width: 54em; /* 864 / 16 = 54 */
   width: 90%;
}

.main-nav {
   max-width: 54em; /* 864 / 16 = 54 */
   width: 90%;
}
```

So we now have two complete minified files. And on our portfolio page, that means we can reference only the minified CSS in addition to only the minified JavaScript, and remove these other three. Scroll back down to the bottom, make sure we got this here. Good. So these two pages now are loading all optimized CSS and JavaScript. And close out the rest of these files. And we've removed a whole bunch of dependencies there.

Now, one of the things that we haven't touched yet are images. So I've already downloaded the ImageOptim application. When we start up the application, we've got a little drag and drop window here. I've got two folders in here. I have an already minified version that I've been referencing. And we're going to go ahead—this is a copy of the same images that are already out in that main image folder.



I'm going to select all of these and we just drag them right over here. And you can see all of the compression happening. You can see what the file savings look like. And you can see there's a fairly significant amount of file savings going on. Now the one thing that you want to remember is, by default, this ImageOptim is going to use a lossless compression. Meaning it's not going to compress JPEGs in the same manner that you might want to.



If you look under the Preferences under Quality, you can enable lossy minification. That's really just saving JPEG quality the way you normally would. I bumped it up to a 90% quality setting and I couldn't see any difference when I was looking at these derived images. So you can play around with this with your own site. Generally speaking, with JPEG quality, somewhere around 85% to 90% is the most level of quality you would need to maintain in order to not really be able to tell a difference and obtain some really substantial file savings.

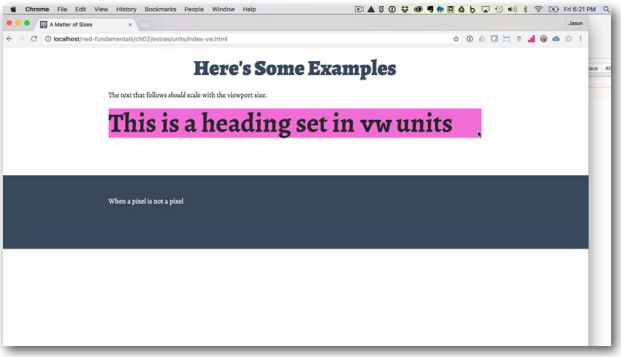
So if we go ahead and take a look here and compare, I'm just going to look at the file size for the sunrise, the largest version. That comes in at 164k. The uncompressed one, if we look back out here in the root of our

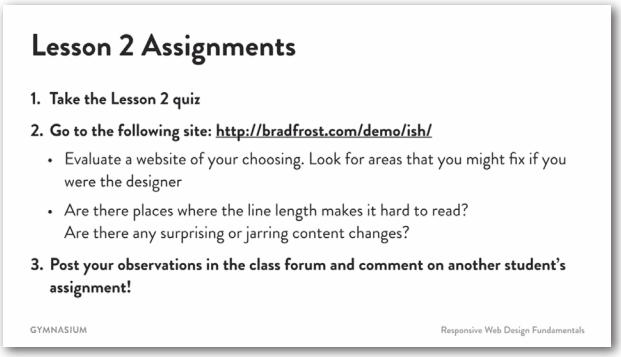
project and do the same thing, 455k. So we've taken nearly 300k away from the size of that image. So we're going to have some massive savings when we run this optimization test.

We have, in our About Working file, we've combined all our CSS. We've combined all our JavaScript. We've minified everything. And we've gone through and we've optimized all of our images. So let's take a look and see exactly what this is doing. Now all of our image references here are pointing to that outer folder. But let's take a look and see.

If we do a little search and replace here and reference this local image folder—I'm just going to do this on this one page right now—and if we go back into our browser here and load the Working page. Now before I do this, when we looked at this before we had 1.1 megabytes being transferred. And if we looked at the images being loaded, we had 936k of images being loaded on this page. Watch what happens when we go view the other page.

We've taken almost 300k of image size away. And if we look at the total requests, it's down to 925k loading in—we'll, locally—in less than a second. But more importantly, when we go and take a look at some of these optimizations in the browser, I already ran a web page comparison on these. From our starting point, we had a 3.1 second load time, a 1.1 second start render, a speed index of 1141, and about 1.5 megabytes of images and assets being loaded, all told, in again, 3 and 1/2 seconds.





With the optimized version, our speed index was down to 1024, our start render was 1 second flat, our total rendered—total page fully loaded time was 2.74 seconds, and only 872k. So we cut the load of, in file size, almost in half. We took, looks like about almost a full second, 8/10 of a second away from the full page load time. And we've taken a fair chunk away in our speed index as well.

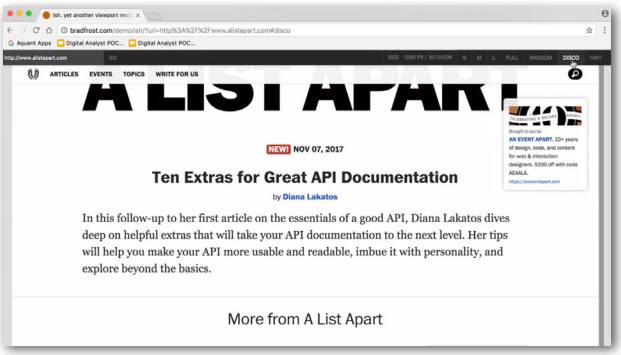
And that start render time, again, coming in not a lot faster, just a tenth of a second. But a tenth of a second is noticeable. So this was a fairly simple page. When you think about applying this across a much larger web site, you can quickly see that those benefits are really going to add up.

We've got carefully written HTML. We've got minified CSS and JavaScript. We've got highly optimized images. And there's a couple of other things that we can do to speed up the process even more, simply by changing the way our JavaScript is being loaded. And all of that goes to getting stuff on the screen faster.

So the last thing that we'll want to do—and you'll want to do this across all the pages—is look down here at the bottom. Very simply, JavaScript has the option to be loaded in particular ways. And without any extra keyword—it just loads normally, has a regular blocking event—you can also specify asynchronous or defer. Async means execute this file whenever it loads.

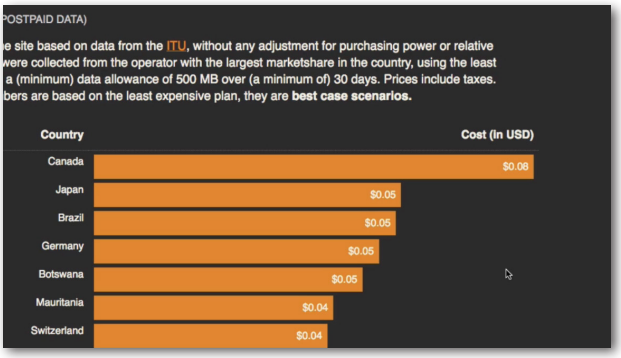
Defer, which is what we're going to add, means go about rendering the page and then execute these JavaScript files in order, once they have all been loaded entirely. So the important thing to note there is, you still make sure that each file is executed in order. So you have to load jquery before you can execute jquery plug-ins. If you load them asynchronously, it's quite possible that one could finish loading before the other.

Now the place where we can absolutely use an asynchronous loading process is up at the top where we're loading our web fonts. And that's exactly what we want to do. Moderniser and Font Face Observer we want to load and make sure we get the whole script in there first. It's then going to go using this Promise syntax to load the web fonts and change things once it can. But still, this is not blocking the rendering of the page.

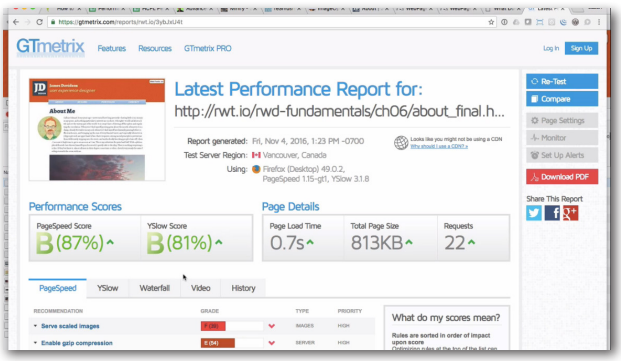


And Picturefill, which is the polyfill we've added in here to ensure support for the picture element in older browsers, we are loading that asynchronously because we know we simply want that to execute as soon as it's ready. But we can still load the images as they are normally. So these are the optimizations that we can do within our project to ensure that this page loads as quickly as possible. And you can see those results as we test them out on the web.

And we can see that we're substantially lowering the cost of people viewing our site. So we noticed that when we tested it previously, it was about $0.18 or so. It was about $0.18 in Canada to look at that page. We've taken $0.10 off the cost by optimizing our images and minifying all of our CSS and JavaScript. So it's a pretty substantial savings.

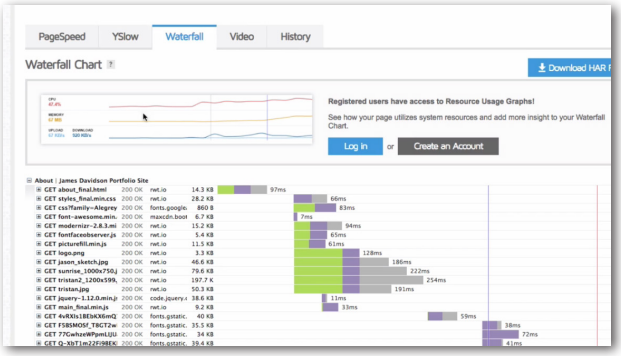


There's one other site that I wanted to take a look at for doing a little test on our site, and that's gtmetrix.com. So G-T-M-E-T-R-I-X .com. And we can just go there and paste in any URL. So grabbing our optimized one, it's going to analyze this both with page speeds and also with YSlow. And you can see that it's being a little picky about the scaled images, mainly because it's analyzing it at a larger screen size.



So it's loading the original image, which is actually what we want. It's going to serve a much smaller one for the smaller screens. And that's fine. The one thing that's really not doing too well here that we can't control just because we're serving this off just a regular HTML site is that we have not enabled gzip compression for the CSS and JavaScript that's being served.

That's something that's normally handled in the server configuration. Usually, you don't have to set that. But if you are responsible for it, it's a fairly simple thing to set up using Apache or most web servers. A lot of times that's going to be on by default. So the only real ding that we have in our grade is something that would normally be handled by standard server configuration.



Now the nice thing in looking at this is, if there are any issues, then it actually will give you a bunch of cues. Again, it also gives you a good waterfall view of what's loading on your page, and when. So you can take a look and see what blocking events there might be going on. And if there is an issue, you can click on it and it'll give you some extra tips to figure out what you might be able to do.

And now it's mentioning using a content delivery network. We're going to get into some more of these things that you can research a little further, in the next chapter. So to wrap up, in this chapter we've optimized what we can. We've combined and compressed our CSS and JavaScript. We've squashed the heck out of all of our images. And we figured out how to load things in a way that will unblock the rendering process and get things on screen as quickly as we can.

And in the last chapter, we're going to take a look at things coming down the pike and some ways for you to look further into your optimization journey and try some more advanced techniques and keep up to speed with everything else going on in the industry.

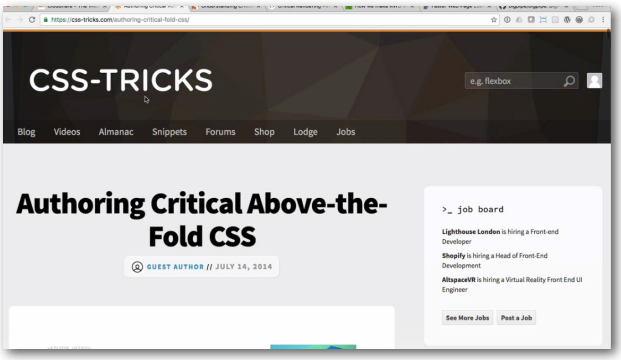
CHAPTER 5: FURTHER RESEARCH

In this last chapter, we're going to take a look at server configuration considerations, some of which we talked about in the last chapter. We're going to get a little bit more in depth with content distribution networks like CloudFlare. We're going to take a look at some options for in-lining CSS and JavaScript and what that all means, and then talk a little bit about some emerging technologies that are going to take performance even further.

A content distribution network is, simply put, a way for assets like images and CSS and JavaScript to be served geographically closer to where the end user is requesting it. And that content distribution network is a way of taking your files and caching them out on these networks and rerouting the request for those files to go through the CDN in order to serve those assets a little bit quicker to where the people are asking for them.

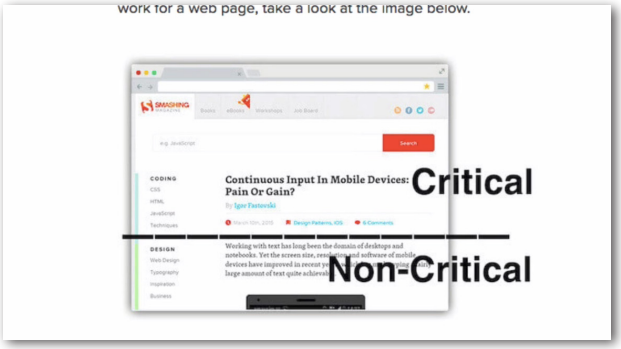
Now, there are a bunch of services that you can use. If you're hosting on Media Temple or some other web hosting providers, many of them have deals with CloudFlare or services like them so that you can try it out for free, depending on how things are set up. I've used CloudFlare.

There's Akamai. There's a whole bunch of other ones out there. They tend to be pretty easy to set up. Lots of content management systems have plug-ins or modules that make integrating them really easy. I know WordPress.com offers that service, as well.



So it's generally just a few clicks to set up. Quite often, it doesn't require any new coding. But the way that it works really does help speed up the end delivery of those assets, because they don't have to come from wherever it happens to be that you're hosting your site. If someone's in Australia, it can make a really big difference in load time.

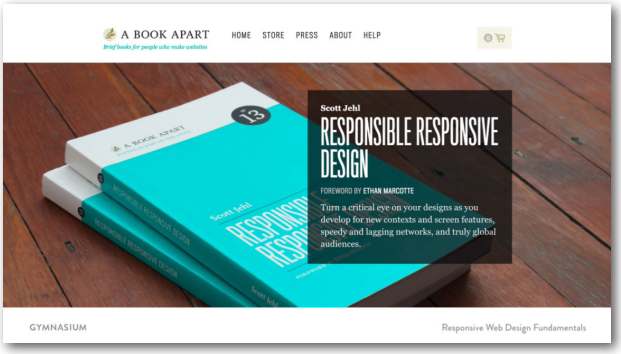
Another thing that's worth talking about is loading critical CSS or JavaScript for above-the-fold content. There's this post, "Authoring Critical Above-the-Fold CSS," by Ben Edwards on CSS-Tricks. And there's also a great post by Dean Hume on Smashing Magazine called "Understanding Critical CSS." And this has a really great illustration to show you exactly what that means.



So when we scroll down here and take a look at this image, you can see that the idea of critical CSS is—what CSS do we need to load in order to lay out this portion of the page that gets seen first? By taking some of that CSS that's used to just do the basic layout and styling of the initial elements on the page and actually putting that in-line, rather than as an external link, then you'd end up getting that stuff being able to render that much faster.

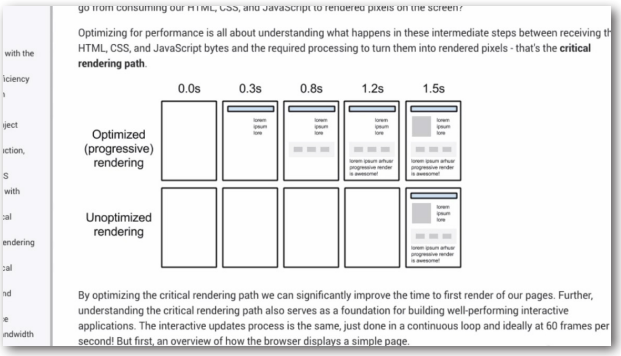
Now, the tricky part is figuring out exactly what you need to load. But there are actually some really interesting tools, and we'll provide some links to those so you can try it out for yourself. The Filament Group has written about that pretty extensively.

There's also a great book written by Scott Jehl called Responsible Responsive Design from A Book Apart that also has a whole bunch of these techniques laid out. They get into a whole bunch of layered approaches, and they really are at the forefront of the absolute best and quickest performance that you can get out of your responsive site.



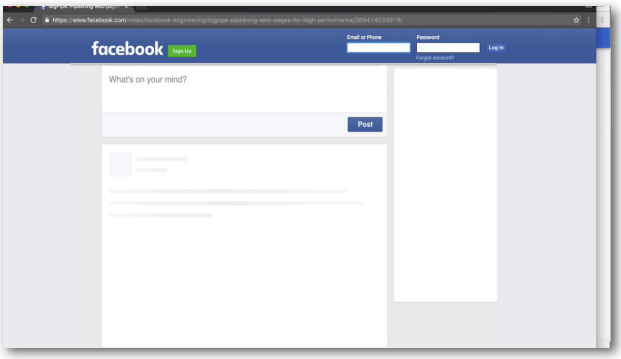
There's also another article that I'm going to point you to here from Ilya Grigorik. Ilya is a developer advocate and web performance guru over at Google, and is really tremendously smart and does a great job of laying this out and shows you what the difference is. If you look at this graphic in the middle of the page here, by optimizing that critical path and undoing some of these blocking things, you can see that you start to get stuff on the screen at 0.3 seconds, 0.8, 1.2, 1.5 seconds to get the whole page.

Versus if you are waiting for all of that CSS and JavaScript to load, there might be a blank screen for a second and a half. So we were able to get the start render time in our project to a second, but it's nicer if it's faster. If you start to see things on the screen, it gets into that whole notion of perceived performance.



Starting to see a change allows you to start to interact with that content. So you can see by the time you get into that 8/10 of a second range, there's already some images and text on that screen that you can start to interact with.

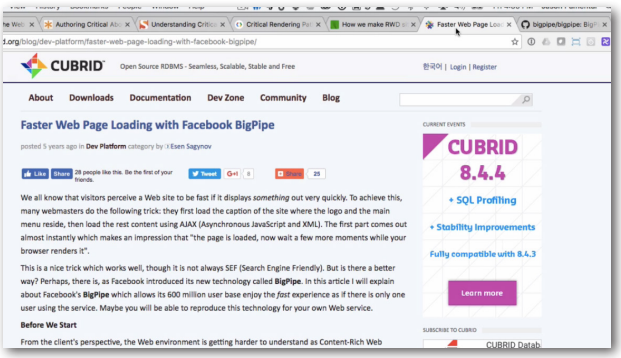
Now, one of the last things that I wanted to talk about is the project that was initiated over at Facebook. Now, if you think about the loading process when you look at your Facebook page, the idea is when that starts to load, you see some placeholders show up and then regular content fills in. And those placeholders generally show up really quickly.



It's actually a JavaScript-based library that's running on the server that alters the way the pages are served. And rather than serving out that whole HTML page and serving that closing body and HTML tags, it actually keeps the connection open and starts to serve out content chunks at a time. And as those chunks come in, then it replaces the placeholder CSS image that's there, and eventually you get that whole page built out.

Again, it's addressing the perceived performance more so than actual, but it really does make a huge difference. And the great thing about this is it's actually all open source. So you can incorporate it in other projects as you see fit.

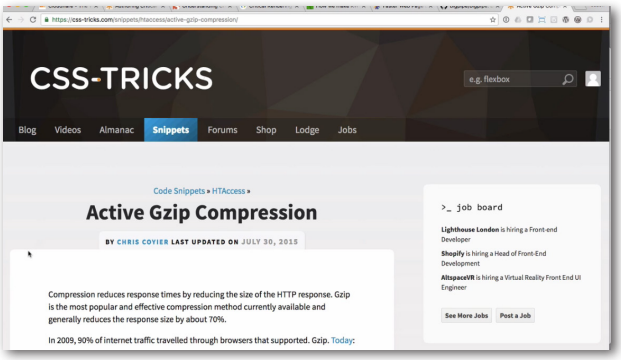
There's a great article over here on—well, I think it's called "Cue-brid," but I'm not really sure how to pronounce it. But we'll give you the link. You can read through it.



It has links to the GitHub repository, which is actually just over at bigpipe.io. You can download the whole thing, or if it so happens that you are working in Drupal 8, it's actually a module that you can just click once and turn on, and you don't have to do anything else. Works really, really well.

Now, one thing that we talked about a little bit in the last chapter was enabling gzip compression on your server. Now, it's not really within the purview of this course to get into advanced configuration, but a lot of us do have to make a few changes here and there in .htaccess files as we work on Unix-based web servers.

There's a great article over on CSS-Tricks about setting up gzip compression, and Chris does a great job of walking you through exactly what you need to add. Most of the time, the necessary components are already there. So there's not too much that you need to do other than make sure you enable it in the .htaccess file.



If it's not there, you might need to do a little bit of extra work. But most of the time, most hosting providers have a lot of these things set up. You just need to make sure it's enabled.

It's going to give you a significant performance boost. It's really worth it. And it also will help out your page speed ranking, simply by having that in place—no matter how much it speeds things up.

So in this chapter, we've taken a look at exactly what we can configure. One of the things that you have to decide is, really, how far you need to go. And we've taken a look at what's coming.

And in making that decision, before you get into these even more advanced techniques, you have to think about going back to your performance budget. What were your targets? What were the numbers you were trying to hit? How are you doing?

If you're meeting those numbers without going into some of these advanced configurations, you're serving it as quickly as you were hoping, maybe you just leave it there and put it on the bucket list to come back to and revisit as you have time to go and tweak it even further as those skills and time become more available to you.

But otherwise, we've really done, I think, a pretty thorough job of going through. We've optimized all our images. We're serving them responsively. We've compressed all our JavaScript and CSS. We've minified everything to eliminate a lot of the extraneous bits and bobs. And overall, I think we've got a pretty fast site. I hope you think so, too.