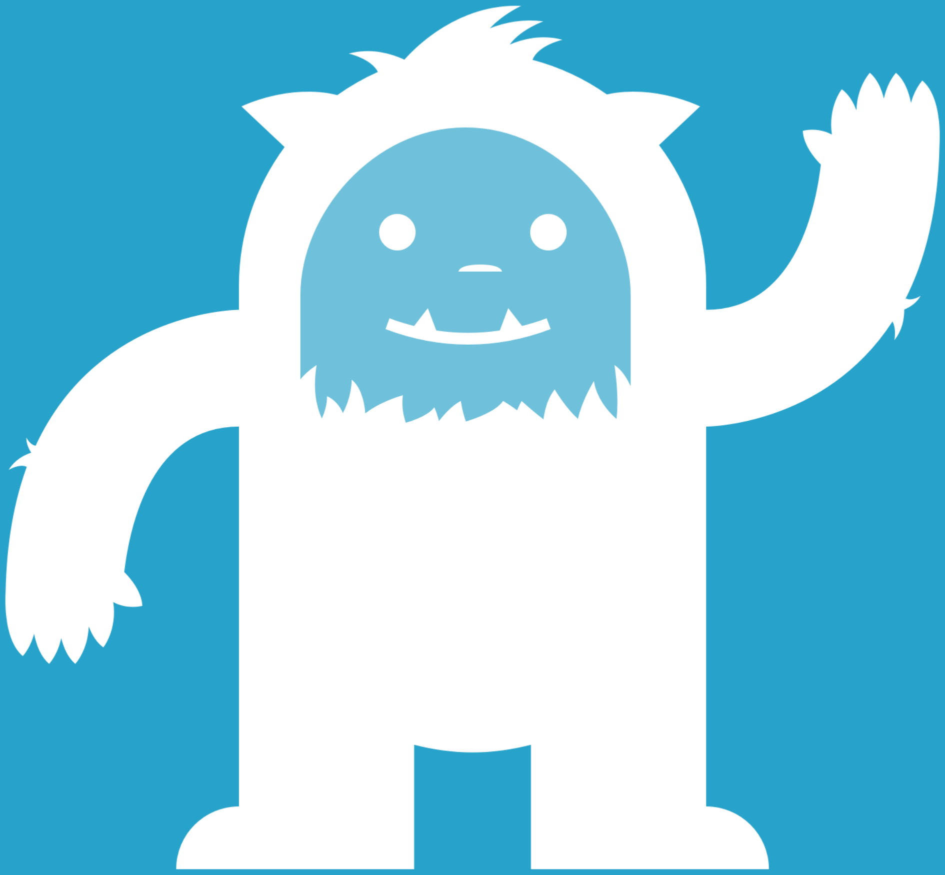


ZURB Foundation

Front End Performance Guide

James Michael Stone

ZURB Foundation Front End Performance Guide

Take your site or app to the next level

James Michael Stone

This book is for sale at http://leanpub.com/zurb-foundation-performance-guide

This version was published on 2014-06-15

Contents

Introduction to ZURB and Foundation

ZURB

ZURB is a self professed team of "T-Shaped Designers that help grow businesses" and doesn't fit into typical agency categories. They have worked with a ton of startups since 1998 and have help build or refine some of the greatest UX you have experienced across names big and small.

They are situated geographically in the city of Campbell, which is a small community in the South Bay Area which is in the heart of the Silicon Valley. One of the things that makes ZURB very unique is how transparent they are in their process and their tools.

If you are interested in Responsive Web Design or or even Sketching Wireframes ZURB has you covered. They not only post amazing articles that offer a lot of value and insight but they also host free events such as their Soapbox and the Foundation Meetup allowing the community to tap into their hive mind and get a glimpse at how the masters work at their craft.

ZURB Foundation

So you might be thinking that Foundation looks a lot like Bootstrap. Well there is a lot of history there and a common linage that leads back to ZURB. (If you want to learn more, listen to the Changelog Podcast, episode 112[1] with Jonathan Smiley and Mark Hays)

A common misconception is that Foundation is for designers and needs a lot of design to work well, where as Bootstrap just looks great out of the box. One can certainly argue that Bootstrap is a little more "designed" out of the box and Foundation looks a little more bare bones.

At first it appears that Bootstrap seems to be clear winner when making a choice. The problem comes later, after implementing Bootstrap on a project and you want to make some large sweeping changes. Sure, this is possible with Bootstrap but it often leads to a lot of overridden CSS or other less graceful solutions.

Foundation uses Sass and allows you to easily and quickly change the color, appearance and style of elements. It is designed to be configurable, flexible and extensible.

Can Web Development be Fun?

In my experience, ZURB Foundation is fun to work with. But, lets see if I can qualify why I think it is fun.

[1]http://5by5.tv/changelog/112

You hear a lot of these same types of comparisons with the Ruby language and the Ruby on Rails framework that popularized it. Things like Syntactic Sugar, Magic and Readable are thrown around. It is something that is hard to explain to people who 1) have not programmed in another language that is not so fun like C or Java and 2) moved to Ruby and Ruby on Rails and 3) experienced the culture of Rails.

Coming back to Foundation it is both the syntax, structure of the framework but most importantly the culture that surrounds it that makes it so much fun.

First off, ZURB is a fun company. Look at the bright colors or the playful and energetic writing style they use.

They are passionate, quirky and fun. Just take a look at their about page or one of their ZURB Wired projects where they build out a project for a deserving charity in a company wide 48 hour marathon.

When you look at Bootstrap in comparison, it is not that it is wrong or right it is just that it is a different culture. Bootstrap presents itself as a mature, corporate and mild mannered framework.

Whats the deal with the Yeti?

Which brings us to the Yeti. Whats up with the Yeti and what does that mean anyways?

Well, ZURB offers a explanation[2], but I think the Yeti is much more.

The Yeti is really the mascot for ZURB Foundation and is lovingly crafted by some of its best designers. There is even ZURB swag if you are lucky enough to get your hands on it. There are stickers, T-Shirts and other Yeti sightings.

If you want to learn more about ZURB, Yetis and ZURB Foundation I suggest taking a look at their official history[3].

The Mobile First Philosophy

Sometimes Foundation gets a bad rap for taking a Mobile First approach. Let me clear up one important thing first: it is completely optional. Don't like Mobile First? Prefer Desktop First, Tablet First or Paper First? No problem, ZURB Foundation has you covered.

What is important to know is that ZURB Foundation allows a Mobile First approach in its workflow and it is very simple to implement. Basically you can start from your Mobile Design (previewing on a mobile device or a small width resized browser window) by using the small grid only such as .small-4.columns.

[2]http://foundation.zurb.com/learn/about.html
[3]http://foundation.zurb.com/learn/about.html

Then, after you have created your mobile interface, you can continue refine your desktop and tablet layouts adding medium[4] and large grids to your existing small grid divs. Or, it is quite possible that you will like the simplicity of your mobile design and just leave it.

The Mobile First implementation in ZURB Foundation stems from a book of the same name written by ZURB Advisor Luke Wroblewski[5] published by A Book Apart. For more details, be sure to check out my review of Mobile First[6].

The basic premise is this: if you design for desktop you will add all sorts of features and widgets adding a lot of complexity to your web app. By approaching your design from the mindset that it will be used on a mobile device it forces you to simplify your UX and UI.

This generally leads to more usable products. Specifically thinking about small screen size, large touch targets for buttons, simplified information displays, a distracted audience and only being able to press something with a single finger limit your options. Or in other words, apply huge constraints. But, as Luke explains in his book, this is a wonderful thing.

Obviously this has had a tremendous impact on ZURB's philosophy as well as the philosophy of ZURB Foundation. You will see that a lot of elements (navbar, buttons, sections, clearing) provide an enhanced and mobile friendly UX and UI right out of the box.

- The navbar collapses to a hamburger menu and provides large touch targets.
- The buttons get expanded to full width on mobile.
- Sections change orientation to work better on a mobile device.
- The Clearing Gallery has larger targets and is even swipe gesture enabled.

You get this all automatically and you get all this for free. All without having to write a single extra line of code.

This all adds up to much faster prototyping, a much better UX and generally a much better workflow for rapid prototyping especially when taking a Mobile First approach.

So Whats the Problem?

Well with all of that ZURB goodness comes a price: Bloat.

CSS Bloat. JavaScript Bloat. Potentially Awesome Retina Image Bloat.

And the real problem adds up to a site that is sluggish, loads slow and can affect your conversions. [need citation] That is what this book is about. ZURB Foundation is fun, but it can be fast too. We are going to take your basic ZURB Foundation site and make it run as fast as possible.

Before we can do that though, we must first learn how to measure, document and understand what fast is.

[4]http://www.manofstone.com/blog/zurb-foundation-5-medium-grid
[5]http://www.lukew.com/about
[6]http://www.manofstone.com/blog/mobile-first-by-luke-wroblewski

Tools for Performance Analysis

In order to better understand the hurdles you must overcome, it is important to get a good assesment of your situation and be able to track your progress. Here are some tools that are useful in testing your website performance as well as measuring important metrics.

Pingdom

http://tools.pingdom.com[7]

Pingdom runs a comprehensive test against your website and provides you a detailed report about what elements are loading, in which order and how much size they occupy.

When you test your site you will have your overview at the top and four tabs of detailed information.

Here are some general guidelines for testing your site.

- If you are hitting a performance milestone, choose make test public so you can compare your results later. If you are experimenting then leave this feature turned off so you do not clutter the graph.
- Choose a server that makes sense. If you are hosting in the United States then this is most likely Dallas. If you are in Europe or Asia then Amsterdam is a better choice. Some hosts are located in NYC such as Heroku, so choose NYC if that is the case. If you are not sure ask you hosting company where their data center is located and chose the host geographically closest to your host.
- If the test errors out or doesn't finish, likely with more than 150 connections, then you have a lot of work to do. I have frequently seen WordPress sites that use many plugins have this issue. Don't panic. Even a partially run test can give you great insight.

So, how do you make sense of this report and what can you do with the information?

First it gives you some good general benchmarks. Ranking, Time to render, Number of connections and Size of Page. Below it will tell you at what percentile your page is at against all tested pages.

Here are some general numbers to shoot for:

[7] http://tools.pingdom.com

rank	> 75
time to render	< 2.0 sec
connections w/ async services**	< 30*
connections, static F5 site	4*
size of page	< 1 mb

- not including image connections, subtract these from you connections ** including things such as Google Analytics, Discus Comments, Embeds and Social Sharing Scripts

So now you know what to shoot for. Lets look at things in more detail.

Server Health

Below your stats you will see the waterfall. The first thing to look is the HTML page being returned. To judge sever health look at the wait time. For static served pages this should be less than 50 ms. If your response time is over 0.5 seconds you should consider static or cached pages on key pages or look at scaling or upgrading your hosting capability.

Remember if you are running a web app, you want to have high perceived speed on your home, landing and marketing pages. Consider caching or preloading you assets for your app on the login page instead where it is less noticeable.

If you serve both to visitors in the USA and Europe, switch servers to get an idea of the speed difference. You might consider a CDN to improve your speed in this case.

The rest of your graph shows everything loading in order.

Large Files

Select the sort by size option and you should now see the largest files near the top. Sometimes this list is not exactly in order so just skip down a bit to the largest file shown? Do you have anything above 100k? These files are good places to start. Make sure you make a note of them. If not you are in better shape but make a note of the largest files as it will help you focus your efforts.

Everything Else

You can flip through the other tabs which can have some useful insights such as the percentage of asset size by script, HTML, image and CSS. You can also get a breakdown of the number of connections by type.

The other useful page is the ranking page. It gives you actionable items you can correct. Many of these will be server configurations and in some cases are not possible with your use case. My site has a score of 82 but ranks in the top 2% of sites tested consistently.

Other tools to look at

- JSpeed[8] - helps you optimize your JavaScript speed.
- CSS Stats[9] - shows you statistics about where your CSS is being used and in what frequency.
- Style Stats[10] - a node library to collect CSS statistics.
- Stress CSS[11] - runs a test via JavaScript to test with CSS classes are hindering performance.

[8]https://code.google.com/p/jspeed
[9]http://www.cssstats.com
[10]https://github.com/t32k/stylestats
[11]https://github.com/andyedinborough/stress-css

Front End Best Practices

Here is a list of what I consider current modern best practices for the Front End. This is an opinionated list and you may develop your own preferences that differ from mine, but nonetheless it is a good place to start. It is also a good place to anchor our discussion about how we can get ZURB Foundation Websites and Apps to run faster.

- As few HTTP calls as possible
- As few DNS lookups as possible
- A single external CSS file called in the head of the document
- A single external JavaScript call at the end of the body with the Defer attribute set
- CSS and JavaScript assets loaded from the root domain1[12]
- Images that are as small in file size as possible

Advanced:

- Images loaded from a single CDN domain, if you have a lot of images
- Everything in the top 2000 pixels should be designed to load and paint as fast as possible
- CSS Sprites or Web Fonts for commonly used graphics across the site: logos, UI elements, etc.

So generally speaking, you want to:

- have as few files as possible
- as few hosts as possible
- the smallest file size possible per file

As with everything, there is a trade off. You will have to find a happy medium which is a hard balance to find and this is especially true with images. Since images make up a large percentage of a sites size it is a good thing to consider what your best practice is.

With everything else, JavaScript, HTML, CSS, etc. you can often reduce the size of your site without a significant change in either functionality or visual quality.

[12]AddingaCDNlatermightbebetterforyourusecase.

How this Applies to ZURB Foundation

ZURB Foundation is a Front-end CSS and JavaScript/jQueryFramework and it can benefit from many of the same performance techniques that apply to other websites. ZURB has changed how Foundation is constructed from time to time so I will go the different variations you might find today in F5.

Everything On

As apposed to advanced frameworks such as InuitCss, Foundation is defaulted with everything-on. This is great for quickly getting up and running quickly with the framework and for fast iterative prototyping but this is terrible for performance.

What usually happens is that this leads to a massive amount of bloat that must be controlled before you push your site or app to production. Before doing this, we must understand exactly what the underlying components and dependancies of Foundation are and how we can modify them to better suit our use cases.

Foundation Sass Overview

By default Foundation creates two Sass files: scss/app.scss and scss/_settings.scss.

The underscore prevents Compass / Sass from generating a file for _settings.scss and it is included as a partial in app.scss.

When you run compass compile, watch or run grunt it generates another file called app.css that is linked in the index.HTML.

Lets take a look at the app.scss and understand how it is an everything-on framework by default.

default scss/app.scss as of Foundation 5.2.1:

```
@import "settings";
@import "foundation";

// Or selectively include components
// @import
//   "foundation/components/accordion",
//   "foundation/components/alert-boxes",
//   "foundation/components/block-grid",
//   "foundation/components/breadcrumbs",
//   "foundation/components/button-groups",
//   "foundation/components/buttons",
//   "foundation/components/clearing",
//   "foundation/components/dropdown",
//   "foundation/components/dropdown-buttons",
//   "foundation/components/flex-video",
//   "foundation/components/forms",
//   "foundation/components/grid",
//   "foundation/components/inline-lists",
//   "foundation/components/joyride",
//   "foundation/components/keystrokes",
//   "foundation/components/labels",
//   "foundation/components/magellan",
//   "foundation/components/orbit",
//   "foundation/components/pagination",
//   "foundation/components/panels",
//   "foundation/components/pricing-tables",
//   "foundation/components/progress-bars",
//   "foundation/components/reveal",
//   "foundation/components/side-nav",
//   "foundation/components/split-buttons",
//   "foundation/components/sub-nav",
//   "foundation/components/switch",
//   "foundation/components/tables",
//   "foundation/components/tabs",
//   "foundation/components/thumbs",
//   "foundation/components/tooltips",
//   "foundation/components/top-bar",
//   "foundation/components/type",
//   "foundation/components/offcanvas",
//   "foundation/components/visibility";
```

So, by default all this file does is load scss/_settings.scss and also bower_components/foundation/scss/_foundation.scss

You will notice that there is a comment about how you can selectively add or remove components. We will get to that soon, but the process is to comment out or remove the main foundation import and then uncomment each component that you would like to use.

Since this is Sass there is no performance issue with including extra comments so it is just fine to leave the commented lines in. They will be automatically removed when Compass / Sass or Libsass builds your CSS files.

NOTE: you might be wondering how Compass knows how to grab the _foundation.scss file automatically. It does that with the setting in config.rb

default ~/config.rb

```
# Require any additional compass plugins here.
add_import_path "bower_components/foundation/scss"
```

Foundation JavaScript Overview

Although Sass provides this very elegant way of making performance modifications, JavaScript is a completely different beast. In the past, ZURB Foundation has included each and every JavaScript file independently and it was up to the end user to optimize them.

Now, ZURB includes a single minified foundation.min.js file that includes all of the functionality by default. The best way to see how this works is to look our generated index.html file.

default ~/index.html

```
<script src="bower_components/modernizr/modernizr.js"></script>
```

The first line is modernizr. This is loaded in the head because you want it to run prior to DOM ready and it is the exception. There are times where you might want to consider omiting Moderinzr but for now I would suggest keeping it.

default ~/index.html continued

```
<script src="bower_components/jquery/dist/jquery.min.js"></script>
<script src="bower_components/foundation/js/foundation.min.js"></script>
<script src="js/app.js"></script>
```

Next, at the bottom of our file we are loading a minified jQuery library from bower, a minimized full version of the Foundation JavaScript Library and finally app.js where we have one line of JavaScript that initializes Foundation.

Notice how we do not have the defer data attribute set. I will discuss this more in detail in the Chapter on JavaScript Optimization.

Development Workflow Explained

In this chapter I will describe my own development workflow with ZURB Foundation to give better insight into what decisions I am making and why. Every person / shop has their own way of doing things. Feel free to take what you like and leave the rest. The key takeaway is that this is my approach so the rest of the optimization methods are organized around it. Feel free to remix appropriately in your own workflow.

Development, Staging and Production Environments

In my work I typically have 3 environments. This is to provide the fastest development time and the most optimized output. This process is automatic and is well suited to team environments. I won't talk about topics such as testing and CI because they are beyond the scope of this book.

Development

This is your local machine and where you do your own development. Often this is a specific directory with a git repo, where I version all of my files and also can either fire up the server from the command line or can point something like Apache / MAMP quickly for development.

The point is that this should be able to run everything in a safe environment that is completely disconnected from your site. In fact, you should ideally be able to run everything without even having an internet connection. So this means if you have databases or custom configurations you will have to figure out a way to duplicate this on your local machine.

I work on many different projects simultaneously so this can mean a different thing for each project. One might be ruby middleman running from the command line, another might be running mamp quickly and pointing it at a specific directory. In more advanced cases you might be running a full copy of the production environment in a vm. The important thing is that it should be able to separately work on each project without it impacting any of the others.

With that said there are some other important considerations. Lets start with JavaScript.

In my development environment I use only development JavaScript. This is an uncompressed, ideally commented version of the library or file. The reason for this is readability. If you use a compressed JavaScript file it becomes much more difficult to debug as everything is on one line. The files are clean, readable and easier to debug.

For CSS I use Sass exclusively, unless the project calls for something else. And my sass I mean Scss files. The same thing goes for sass, I use uncompressed Scss files with comments and I leave all of the line number and debugging options on in compass.

Why not use Sass instead of Scss? One reason is that you can not use libsass with sass files. Another reason is that although I prefer the syntax of sass, it is just another layer of complexity. All of the other libraries that I work with are Scss so I just keep that the same.

Let me show you how I set up a typical Foundation project by showing you my app.scss file.

[app.scss example]

So what about things like images? I try and keep the images in a consistent top level directory, something like ~/images/ I do this because it makes the process of automatically uploading and changing the references to your images to a CDN easier in the future.

Staging

The staging server is a complete copy of the production server. If I am deploying production to Heroku I will create a staging server on a free instance on Heroku as well. If I am using a Digital Ocean instance I will create a duplicate server and spin it up while testing and destroy it when I am one. On RackSpace I follow a similar process.

The important thing here is that it is something off of your dev box out on the internet. That way you can test it in real world conditions and also get feedback from stakeholders in the project.

Once you have staging the way you want it you can push to production.

Production

This is your live sever that is in ideal situations being used by real people. Make sure you push this in a way that doesn't impact many people and that the process is as fast as possible.

For static sites rebuilt with rsync, you might be able to do this on the fly but for other tech stacks you might need more time and need to do it in off peak hours.

Summary

So for our purposes Staging and Production are identical but differently named processes. If you are using Rails or Middleman with the Sprocket Asset Pipeline a lot of your automation will be done automatically. For many other technologies you will need to create a build script.

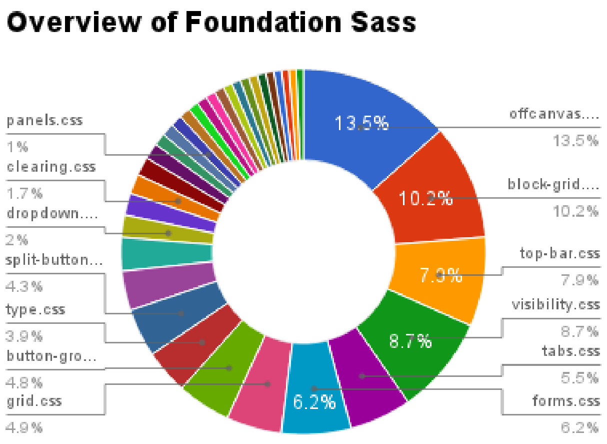
As you go through the following chapters, be thinking about which things you could automate and which you cannot.

There are things that are easy to do with automation and there are things that will take a great amount of effort for very little return.

At the end I will show you how to create a variety of build scripts as well as how to approach this automation in tools such as Rails 4 and Ruby Middleman.

Removing Unused Scss

One of the biggest things you can do to immediately reduce the CSS footprint of your Foundation install is to remove unused components. Lets take a look at exactly how much space that each of the Foundation components takes up.



Overview of Sass Component Size

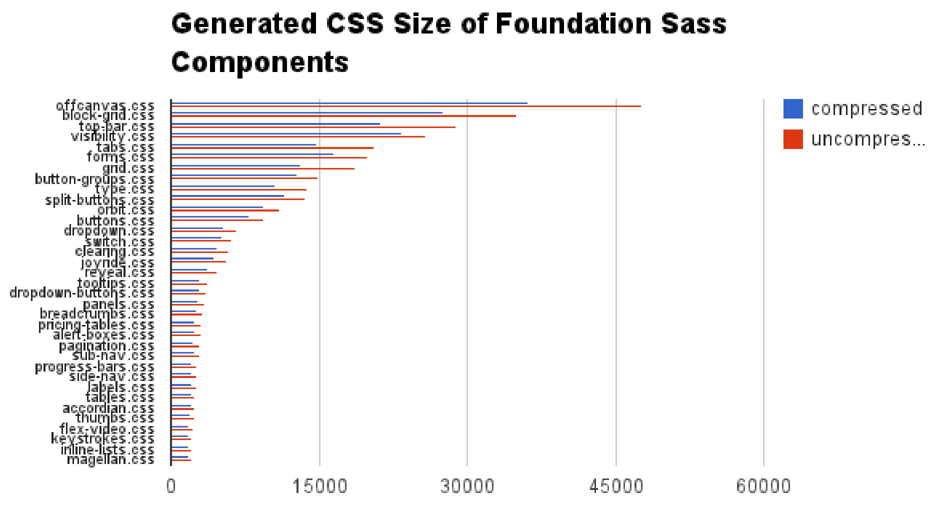


Chart Showing Compressed and Uncompressed Sass File Sizes

file	compressed	uncompressed
offcanvas.css	36220	47722
block-grid.css	27528	35050
top-bar.css	21254	28785
visibility.css	23391	25804
tabs.css	14742	20515
forms.css	16574	20018
grid.css	13167	18586
button-groups.css	12776	14979
type.css	10559	13833
split-buttons.css	11502	13568
orbit.css	9380	11100
buttons.css	7941	9448
dropdown.css	5301	6595
switch.css	5227	6228
clearing.css	4676	5899
joyride.css	4433	5611
reveal.css	3810	4656
tooltips.css	2937	3667
dropdown-buttons.css	2995	3517
panels.css	2785	3332
breadcrumbs.css	2612	3190
pricing-tables.css	2466	3022
alert-boxes.css	2470	3002
pagination.css	2324	2843
sub-nav.css	2357	2842

file	compressed	uncompressed
progress-bars.css	2155	2635
side-nav.css	2111	2547
labels.css	2092	2542
tables.css	2058	2490
accordian.css	2029	2480
thumbs.css	1933	2358
flex-video.css	1825	2220
keystrokes.css	1757	2139
inline-lists.css	1734	2120
magellan.css	1748	2117

A Minimal Sass Project Example

If you are architecting a new ZURB Foundation project one strategy is to try and be as lean as possible to start. By deciding early on exactly what components you want to use from the framework you can drastically increase the performance of your site.

In the example below I have cut most of the JavaScript components and components that have a large CSS size but might not be crucial to our business goals. Offcanvas for example is a very slick component, however, that comes with a cost.

It is worth evaluating closely the first 10 - 15 line items to decide if they are really necessary.

minimal-app.scss

```
@import "settings";
// comment out Founadtion components that are not used below
// @import "foundation/components/accordion";
// @import "foundation/components/alert-boxes";
// @import "foundation/components/block-grid";
@import "foundation/components/breadcrumbs";
// @import "foundation/components/button-groups";
@import "foundation/components/buttons";
// @import "foundation/components/clearing";
// @import "foundation/components/dropdown";
// @import "foundation/components/dropdown-buttons";
// @import "foundation/components/flex-video";
@import "foundation/components/forms";
@import "foundation/components/grid";
// @import "foundation/components/inline-lists";
// @import "foundation/components/joyride";
// @import "foundation/components/keystrokes";
```

```
@import "foundation/components/labels";
// @import "foundation/components/magellan";
// @import "foundation/components/orbit";
@import "foundation/components/pagination";
@import "foundation/components/panels";
// @import "foundation/components/pricing-tables";
@import "foundation/components/progress-bars";
// @import "foundation/components/reveal";
// @import "foundation/components/side-nav";
// @import "foundation/components/split-buttons";
// @import "foundation/components/sub-nav";
// @import "foundation/components/switch";
// @import "foundation/components/tables";
// @import "foundation/components/tabs";
// @import "foundation/components/thumbs";
// @import "foundation/components/tooltips";
@import "foundation/components/top-bar";
@import "foundation/components/type";
// @import "foundation/components/offcanvas";
// @import "foundation/components/visibility";
```

By initially limiting our components to the following list:

- breadcrumbs
- buttons
- forms
- grid
- labels
- pagination
- panels
- progress-bars
- top-bar
- type

We have retained a lot of the components and design decisions that will make our app development go much more quickly. This does not come with any loss in flexibility and you can still override all of the defaults in our _settings.scss file as well as add our own Sass to the mix.

From our original compressed files size of 147k we have dropped to 50k compressed which is about 1/3 the size of what we started with. Not bad.

CSS Minification + Optimization

First lets take a look at what we are starting with. Recently ZURB has updated Foundation to use a minified CSS file which is a step in the right direction by default, but if you are planning on using Sass it is not going to help much.

Nonetheless, it is a good place to start our comparison.

Bower Defaults	Size
foundation.css	190k
foundation.min.css	155k

Now, lets take a look at what can be done just with CSS optimization alone. These are just changing the compass settings in the config.rb file and for comparison using an online version of the YUI Compressor. We are still using the ZURB Foundation framework in its entirety.

Compression Settings	Size
Basic Full Foundation install from CLI	321k
line_comments = false	183k
output_style = :compact & line_comments = false	170k
output_style = :compress & line_comments = false	148k
using online yui compressor [1]	148k

Here we are able to get to 46% of the original app.css file size. All of these files have exactly the same functionality. The only difference is the compression and the readability of the file.

This can be even further reduced by enabling gzip compression which I will touch on in the chapter dealing with the server.

[1] http://refresh-sf.com/yui

Semantic Markup with Sass Mixins

Coming soon…

Going Sass Only

So you might be thinking, what is Sass only? It is exactly what it sounds like, it is just using the Sass components of Foundation without any of the ones that JavaScript to function.

This will limit you to what features that you might use, however, the trade-off in compatibility and file size is enormous.

ZURB Foundation Features Available Sass Only:

- Block Grid
- Breadcrumbs
- Button Groups
- Buttons
- Flex Video
- Forms
- Grid
- Inline Lists
- Keystrokes
- Labels
- Media Queries
- Pagination
- Panels
- Pricing Tables
- Progress Bars
- Right-to-Left Support
- Side Nav
- Sub Nav
- Tables
- Thumbnails
- Type
- Utility Classes
- Visibility

So as long as you stick to the above list you are going to be compatible with anything that allows you to use CSS, with one caveat. You must make sure that none of the Foundation CSS class names conflict with any other technologies you are planning on using.

Furthermore, if you follow this technique to remove all of the unused CSS Classes you will be left with a single CSS file that is quite compact.

This technique can also be combined with only using Sass Mixins which gives you the best in both Semantic HTML, compact CSS and better compatibility with other JavaScript frameworks.

Just in case you are wondering on what you might be missing out on, here is a list of the ZURB Foundation components that require JavaScript to fully function. Remember it is not only the JavaScript files but also jQuery which is [need file size] itself minified.

ZURB Foundation Features Requiring JavaScript:

- Abide Validation
- Accordions
- Alerts
- Clearing Lightbox
- Dropdown and Split Buttons
- Equalizer
- Interchange
- Joyride
- Magellan Sticky Nav
- Orbit Slider
- Reveal Modals
- Sliders
- Tabs
- Tooltips
- Topbar

Removing Unused JavaScript

Before jumping into specifics of how to optimize our JavaScript, lets quickly introduce the libraries that make up the entirety of the ZURB Foundation 5 package.

Modernizr

Modernizr is a library that detects features that are available in older browsers and in some cases allows you to load libraries or polyfills to provide such support. The question is do you need Modernizr and the answer is it depends. It depends on what browsers you want to support. Keep in mind that ZURB Foundation 5 only supports the following browsers, so you might find diminishing returns in using this library: Chrome, Firefox, Safari, IE9+ , iOS Safari (iPhone + iPad), Android 2, 4 (Phone + Tablet), Windows Phone 7+, and Surface.

As of this writing the following browsers are supported by Modernizr and its feature detection: IE6+, Firefox 3.5+, Opera 9.6+, Safari 2+, Chrome, iOS's mobile Safari, Android's WebKit browser, Opera Mobile, Firefox Mobile and possibly Blackberry 6+.

Note: Modernizr is loaded in the <head> so you it is a special case when considering performance.

http://modernizr.com[13]

jQuery

jQuery is a dependency of Foundation so if you use any of the JS components you will need to include it. jQuery is a library that makes things like changing the DOM, attaching events and using async functionality easy. In addition it makes these things work across many different browsers which is why many projects use it. Less code and browser compatibility.

The downside is that the library is quite large. If you can get away with not using the any of the Foundation JS components my recommendation is to try and not use it unless you are doing very complex things in JavaScript.

If performance is not a concern or you are prototyping, by all means start with jQuery. Then consider refactoring before you go to production.

http://jquery.com/[14]

[13]http://modernizr.com
[14]http://jquery.com/

FastClick

This library removes the 200 ms delay on clicking. This makes your pages perceived speed much faster. You can use this without any other libraries or JavaScript code. There can be issues using it with single page apps or things like Rails Turbo Links so make sure you test on a device before you roll it out into production.

https://github.com/ftlabs/fastclick[15]

jQuery Cookie

This is a simple lightweight JavaScript library for setting and reading cookies. It is used by Foundation JoyRide. If you are not using JoyRide you do not need this file.

https://github.com/carhartl/jquery-cookie[16]

jQuery Placeholder

This is a Polyfill that allows form input placeholder behavior on browsers that do not support it.

https://github.com/mathiasbynens/jquery-placeholder[17]

Foundation

This acts like a pseudo jQuery plugin loader for the Foundation JavaScript components. It will load any Foundation components that you load via JavaScript as well as tell them to run at startup.

This is the foundation.init call you see in every new Foundation install and is a requirement if you want to use any of the Foundation JS Components. In the official docs, it is components marked with a JS.

It has jQuery as a dependency. So if you want to use even one F5 component you will need this file as well as jQuery. If you find yourself in that position, consider looking for a JavaScript library with the same functionality that does not require jQuery and that is smaller in size that the other 3.

The Foundation Components

These function like jQuery plugins bit are loaded up by the foundation.js file and loaded with the foundation.init call.

Leave them all on while prototyping and remove the unused ones when you go to production. By default Foundation links to a minified foundation file that contains everything.

[15]https://github.com/ftlabs/fastclick
[16]https://github.com/carhartl/jquery-cookie
[17]https://github.com/mathiasbynens/jquery-placeholder

```
<script src="bower_components/jquery/dist/jquery.min.js"></script>
<script src="bower_components/foundation/js/foundation.min.js"></script>
<script src="js/app.js"></script>
```

I recommend replacing that with the following code in all of your files.

```
<script src="bower_components/jquery/dist/jquery.js"></script>
<script src="bower_components/foundation/js/fastclick/lib/fastclick.js"></script>
<script src="bower_components/foundation/js/jquery.cookie/jquery.cookie.js"></scr\
ipt>
<script src="bower_components/foundation/js/jquery-placeholder/jquery.placeholder\
.js"></script>
<script src="bower_components/foundation/js/foundation/foundation.js"></script>
<script src="bower_components/foundation/js/foundation/foundation.abide.js"></scr\
ipt>
<script src="bower_components/foundation/js/foundation/foundation.accordion.js"><\
/script>
<script src="bower_components/foundation/js/foundation/foundation.alert.js"></scr\
ipt>
<script src="bower_components/foundation/js/foundation/foundation.clearing.js"></\
script>
<script src="bower_components/foundation/js/foundation/foundation.dropdown.js"></\
script>
<script src="bower_components/foundation/js/foundation/foundation.equalizer.js"><\
/script>
<script src="bower_components/foundation/js/foundation/foundation.interchange.js"\
></script>
<script src="bower_components/foundation/js/foundation/foundation.joyride.js"></s\
cript>
<script src="bower_components/foundation/js/foundation/foundation.magellan.js"></\
script>
<script src="bower_components/foundation/js/foundation/foundation.offcanvas.js"><\
/script>
<script src="bower_components/foundation/js/foundation/foundation.orbit.js"></scr\
ipt>
<script src="bower_components/foundation/js/foundation/foundation.reveal.js"></sc\
ript>
<script src="bower_components/foundation/js/foundation/foundation.slider.js"></sc\
ript>
<script src="bower_components/foundation/js/foundation/foundation.tab.js"></scrip\
t>
<script src="bower_components/foundation/js/foundation/foundation.tooltip.js"></s\
cript>
<script src="bower_components/foundation/js/foundation/foundation.topbar.js"></sc\
ript>
<script src="js/app.js"></script>
```

Why change this? Three reasons.

1. You use developer uncompressed JavaScript that is easier to debug. Since they are broken up by file it will help you to locate the offending component.
2. It is a simple process to remove specific components when you go to production.
3. You will want to combine / concatenate all of these files together into one file at some point so make it easy to do so from the start.

app.js

This is just boilerplate with the Foundation init call. You will notice that it is the last one in the file. That is because it requires everything else to be loaded first.

This is also a good place to add any of your own JavaScript for smaller projects or while prototyping.

Order of JavaScript

These libraries are loaded in the order that they appear, so make sure you load any libraries that are used by foundation before it is loaded. Then load all of your Foundation JS Components after it. Finally, load your app.js file at the end with the Foundation init call.

Minification and Optimization of JavaScript

Coming soon…

Image Optimization

Retina Graphics and Image Bloat

Image Bloat is one the single largest contributors to website bloat in the last few years. One of the driving fators is the adoption of retina or high pixel density displays. The first thing you need to decide is if you want to support retina graphics and to what level.

This is the crux of the problem: when you resize an image to be double the size (by width and or height) you are increasing the number of pixels 4 times.

Jpeg compression can combat this to an extent, but first lets look at the common ways to handle these different display resolutions in a responsive framework like ZURB Foundation.

1.5 or 2x Images

The big debate is whether you should have 1.5x or 2x images. Of course these are just rules of thumb and it is quite possible to have 1.6x or 1.3x graphics for that matter.

Apple purists and people that really want to cater to these displays will side with 2x images. So much in fact, that it is a default output option on most web graphics software, such as Sketch by Bohemian Coding.

This ideally gives you the type of density to really make these retina images shine. If you want the highest quality images, this is by no doubt the best option for you. However, it is not with out its drawbacks however, which usually results in much higher image file size.

If you have only a few images on a page it is less of an issue, but if you are creating a site or an app with many images displayed per page, such as a blog or image sharing app, small design decisions such as this can really impact the file size and therefore the actual and perceived speed of it.

The option that I prefer to use if I want to support higher pixel density displays is 1.5x. In my opinion and through my testing I did not find a big enough difference in quality between 1.5x and 2x images to justify the additional performance hit of 2x.

Really Squashing your Jpegs

So whether or not you use 1.5x or 2x it is critical that you really bump up your Jpeg compression level. I usually start around 60% compression and will lower it if I do not see too much artifacting. With non-retina images I usually start around 80%.

I have found by increasing the compression level I can often get a similar looking image, at non-retina resolution at about the same file size. You may have to experiment with this on a case by case basis until you find settings that work well for you.

Not only do you get a similar looking image but it also looks great at higher resolution. Since the resolution is so high, the Jpeg artifacts are more difficult to see and often this is a trade-off that is well worth the performance increase.

You might be wondering, won't this create a performance issue with slow mobile devices. This is possible and quite true. However, when using a responsive framework the images are going to resize proportional to the screen anyways, so you will have this issue regardless of the image size.

Taking it to the Next Level

So exporting your images from your app with a higher level of compression is a good start, but lets take it further. By using an app such as Image Optim you can further reduce the size of your images, often without any visible change in the quality.

If you would like to learn more about Image Optim I would suggest this great article on Smashing Magazine[18].

[18] http://www.smashingmagazine.com/2013/12/17/imageoptim-cli-batch-compression-tool/

Dealing with the Server

Coming soon…

Working with Sass Standalone

Up to this point we have been talking about techniques with the Sass Standalone version, or the version of ZURB Foundation that is installed with the CLI (Command Line Interface), but in this chapter we will tie everything together.

We will take the techniques from the previous chapters and implement them in a build script that you can modify for your future projects.

This will allow you to continue to use development versions of JavaScript and CSS while you work on your local machine. Then, when you are ready to go to production, you will run grunt or rake and it will generate a clean and optimized version ready for production.

Finally, we will discuss how to use rsync to upload to your server and how to integrate it with your build script.

What is Grunt

If you have used the new libsass version of foundation you have already used Grunt as a replacement for compass. Grunt is capable uf much mcuh more. In this chapter we willtie together everything we have learned to create an optimized porduction version of our site from our unoptimized dev site. It will do this quickly and on command.

Note: if you are a Ruby shop you might want to skip to the Rake Build Script below.

What is Rake

Coming soon…

Setting up Your Rake Build Script

Coming soon…

Optimizing Sass

Coming soon…

Optimizing JavaScript

Coming soon…

Optimizing Images

Coming soon…

Deploying via Rsync

Coming soon…

Working with the Rails Asset Pipeline

If you have used Ruby on Rails then you have likely used the Asset Pipeline or Sprockets. The asset pipeline is a great tool for allowing you to work with development versions of files on your development machine and then to optimize everything automatically for a production environment. In this chapter we will look at setting up a Rails 4 project with ZURB Foundation 5 and optimizing it for production. In addition we will also discuss specifically deploying to Heroku and what additional steps and measures can boost your performance

Creating a ZURB Foundation 5 Project with Rails 4

Coming soon…

Optimizing Sass

Coming soon…

Optimizing JavaScript

Coming soon…

Optimizing Images

Coming soon…

Deploying to Heroku, Special Considerations

Coming soon…

About James Stone

James Stone

James Stone is the author of ZURB Foundation Blueprints and is a top contributor to ZURB Foundation. James has spoken at the HTML5 Developers Conference, TEDxPSU and the UMD Design | Cultures + Creativity Honors College. He holds an MFA in Studio Art and served as a University Fellow and Instructor at the School of Visual Arts at Penn State. Stone posts screencasts and tutorials from his blog regularly where he hopes to demystify the techniques of modern web development at manofstone.com[19]

[19]http://www.manofstone.com