Gulp, SASS, Optimizations, and Bootstrap 4

CS-554 – WEB PROGRAMMING

Easy Optimizations

AN EXERCISE IN USING TERRIBLE STOCK IMAGES TO ILLUSTRATE POINTS

Declutter your head

The head tag blocks the page while assets are loading. You should basically never load JavaScript in your head tag, and only load small amounts of CSS.

To our right is an example of clutter. Please do not let your head tag get this cluttered.

https://unsplash.com/photos/Kw_zQBAChws

Minify, Uglify, Optimize your Code

Minification is the process of removing whitespace and other unnecessary bloat from your code. This saves you many, many kilobytes.

Uglifying / Optimizing your code rewrites the code to remove as much as possible.

You should strive to have as few bytes as possible sent down. Even things that are usually large (huge JavaScript files, the pineapple to our right) can be made small.

https://unsplash.com/photos/s74-lwoDFOw

Remove Dead Code / Reduce Overhead / Reduce Requests

You should make sure that all code that is unused is constantly removed. In general, you should make sure that you use as little code as possible. When possible, don't import huge libraries; import as small parts as possible.

In addition, you should reduce the total number of requests as much as possible. Requests are very expensive operations.

https://unsplash.com/photos/PRCWaGDZAYY

Use CDNS and Caching

Cached resources load near instantly, and CDNS have the dual purpose of speeding up load time at a network level (by having multiple distribution centers), and by allowing scripts to be cached across many websites.

https://unsplash.com/photos/CyFBmFEsytU

Compress Text with Gzip, Optimize Images

Squash down every byte possible by using Gzip on your server to reduce the size of assets sent. You should also optimize images to make them as small as possible.

https://expressjs.com/en/advanced/best-practice-performance.html#use-gzip-compression

http://nginx.org/en/docs/http/ngx_http_gzip_module.html

https://imageoptim.com

https://unsplash.com/photos/mSzdau1JOPE

Sidebar: Stock Photography

Using Photos Online

Sometimes, you will need images. You should use online stock photography resources, so that you can either purchase the right to use an image or use a free stock photography services to get images that you are **allowed** to use.

You are often told that the "real world" is different than academia, and it is, but one thing is for certain: passing off someone else's work as your own is **very, very bad**. Stock photography that you buy or are licensed to use means you have permission to do that in an acceptable way.

Free (be a good person, give credit)

- https://unsplash.com/
- https://pixabay.com/

Paid:

- https://www.gettyimages.com/
- https://www.shutterstock.com/

SASS

What is SASS?

SASS is a pre-processed CSS language

- It gets compiled into CSS

It must be built before the browser can render it

- We will build with Gulp

It's just syntactical sugar over CSS

- http://sass-lang.com/guide
- SASS does not do anything that CSS does not, it just makes it more convenient to write

What are SASS Features?

Variables

Nesting

Imports

Mixins

Extending

Inheritance

SCSS SYNTAX

```
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

Variables

We can define variables simply by using the dollar sign in front of a string name and assigning a value

Variables have scope; they are accessible in the block they are defined

```
.container { width: 100%; }


article[role="main"] {
  float: left;
  width: 600px / 960px * 100%;
}

aside[role="complementary"] {
  float: right;
  width: 300px / 960px * 100%;
}
```

Operators

SASS supports many basic operators – and, it supports mixing units.

This is a really, really, **really** good thing.

```
$name: foo;
$attr: border;
p.#{$name} {
  #{$attr}-color: blue;
}
```

is compiled to:

```
p.foo {
  border-color: blue; }
```

String interpolation

We can interpolate variables easily, which is very useful in mixins and making dynamic class names.

```
#main p {
  color: #00ff00;
  width: 97%;

  .redbox {
    background-color: #ff0000;
    color: #000000;
  }
}
```

is compiled to:

```
#main p {
  color: #00ff00;
  width: 97%; }
  #main p .redbox {
    background-color: #ff0000;
    color: #000000; }
```

Nesting

You can nest SASS selectors in order to create a nested CSS output:

You can use a parent selector to reference the selector and add rules before or after that selector rule-set.

```
a {
  font-weight: bold;
  text-decoration: none;
  &:hover { text-decoration: underline; }
  body.firefox & { font-weight: normal; }
}
```

is compiled to:

```
a {
  font-weight: bold;
  text-decoration: none; }
  a:hover {
    text-decoration: underline; }
  body.firefox a {
    font-weight: normal; }
```

The parent selector

We can use the & symbol to state “for the selector I am nested inside **with** these other selectors” or “with this selector than the current selector”

Imports

You can have SASS import different files by using the @import ’filename.scss’ syntax.

This is useful for including other related data.

```
.sidebar {
  width: 300px;
  @media screen and (orientation: landscape) {
    width: 500px;
  }
}
```

is compiled to:

```
.sidebar {
  width: 300px; }
  @media screen and (orientation: landscape) {
    .sidebar {
      width: 500px; } }
```

Nested Media Queries

We can nest media queries inside of rulesets and have the media query bubble up and wrap the updated ruleset.

```
$type: monster;
p {
  @if $type == ocean {
    color: blue;
  } @else if $type == matador {
    color: red;
  } @else if $type == monster {
    color: green;
  } @else {
    color: black;
  }
}
```

General control

We can use many common control types with SASS, such as *if* and *for*

For and each

```
@for $i from 1 through 3 {
  .item-#{$i} { width: 2em * $i; }
}
```

is compiled to:

```
.item-1 {
  width: 2em; }
.item-2 {
  width: 4em; }
.item-3 {
  width: 6em; }
```

```
@each $animal in puma, sea-slug, egret, salamander {
  .#{$animal}-icon {
    background-image: url('/images/#{$animal}.png');
  }
}
```

is compiled to:

```
.puma-icon {
  background-image: url('/images/puma.png'); }
.sea-slug-icon {
  background-image: url('/images/sea-slug.png'); }
.egret-icon {
  background-image: url('/images/egret.png'); }
.salamander-icon {
  background-image: url('/images/salamander.png'); }
```

```
@mixin box-shadow($shadows...) {
  -moz-box-shadow: $shadows;
  -webkit-box-shadow: $shadows;
  box-shadow: $shadows;
}

.shadows {
  @include box-shadow(0px 4px 5px #666, 2px 6px 10px #999);
}
```

is compiled to:

```
.shadows {
  -moz-box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;
  -webkit-box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;
  box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;
}
```

Mixins

Mixins are styles that can have parameters.

We **include** the mixins, rather than extend them.

```
$grid-width: 40px;
$gutter-width: 10px;

@function grid-width($n) {
  @return $n * $grid-width + ($n - 1) * $gutter-width;
}

#sidebar { width: grid-width(5); }
```

Becomes:

```
#sidebar {
  width: 240px; }
```

Functions

We can write our own functions in SASS, which allow us to very easily create our own libraries full of rules and utility methods that we can use across an entire project.

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.seriousError {
  @extend .error;
  border-width: 3px;
}
```

is compiled to:

```
.error, .seriousError {
  border: 1px #f00;
  background-color: #fdd;
}

.seriousError {
  border-width: 3px;
}
```

Extending

We can extend content so that we can reduce redundancy in our HTML and no longer have to remember combinations of classes to define.

Placeholders

```
// This ruleset won't be rendered on its own.
#context a%extreme {
  color: blue;
  font-weight: bold;
  font-size: 2em;
}
```

However, placeholder selectors can be extended, just like classes and ids. The extended selectors will be generated, but the base placeholder selector will not. For example:

```
.notice {
  @extend %extreme;
}
```

Is compiled to:

```
#context a.notice {
  color: blue;
  font-weight: bold;
  font-size: 2em; }
```

Building Bootstrap 4

Bootstrap 4 Changes

Bootstrap 4 is the most recent major version of Bootstrap.

The most major differences from Bootstrap 3 (which was the active version for many years)

- Dropped IE8 support; IE9+
- Changed from LESS to SASS
- Now using *rem* as the primary unit
- New, even tinier grid added
- Flexbox grid added
- Panels, thumbnails, and wells are now *cards*
- General improvements

A full changelog can be found:

- https://getbootstrap.com/docs/4.0/migration/#global-changes

Why build Bootstrap?

Bootstrap was created in order to make building websites more easily.

There's a strange complication that resulted because of that: it is now used **everywhere**.

- https://trends.builtwith.com/docinfo/Twitter-Bootstrap

The problem with that is that most people do not particularly customize Bootstrap, creating a situation that many websites now look like carbon copies of each.

Simply put, this has become a "big" small issue. We've created an internet where everything looks the same.

- It's not *wrong*, but it's definitely not *great*

What's the compromise?

We can compromise by still using Bootstrap but customizing and building our own version of Bootstrap.

There are 2 general strategies:

- Download Bootstrap's compiled version and modify it accordingly.
- Download Bootstrap's source code, change variables, and build it yourself.

By downloading Bootstrap's source, we can create our own copy of its style

- By keeping a reference to source repository, you can pull in updates constantly from the official Bootstrap repository.
- You can modify all variables and cleanly modify the entire display with minimal tweaks to the actual source logic.

How do we compile Bootstrap?

Bootstrap 4 is written in SASS, so we can just customize it by updating the SASS source files.

We can easily create Gulp setup to go through the process of building it ourselves.

You can install the bootstrap through NPM, and reference the SASS files in your gulp script if you have simple needs

- `npm install bootstrap`

That being said, to achieve full customizability over Bootstrap, we will download and compile our own copy of the source files.

- https://getbootstrap.com/docs/4.1/getting-started/download/#source-files

Selectively Including Components

One of the easiest optimizations to ever make is removing things you don't need.

Bootstrap's primary file *bootstrap.scss* is actually just a series of includes that allow you to selectively enable and disable components from being included.

If you know that your page only needs a certain subset of features, you can simply remove those includes from this file and enjoy the benefits of:

- Smaller downloads
- Shorter build times
- Simpler code base with fewer things to worry about

Tweaking Bootstrap: Variables

The next thing to look for when tweaking Bootstrap is the _variables.scss file

- This can be seen as a “helper”; underscores are really just naming conventions

This file contains a series of variables that can be updated that each are propagated throughout the entire bootstrap source.

You’ll notice that there are many branding-related concepts. By tweaking a single variable, we can change the look and feel of many components to all be matching.

Tweaking Bootstrap: Modifying the Content

Bootstrap runs off a series of includes, meaning that it's designed in an incredibly modular fashion.

Each included file builds out one single component, meaning that you can modify components and not affect other styles.

We can also follow their design pattern and leverage their common utility classes to create new components of our own. We can write our components using the Bootstrap variables and mixins to quickly create highly consistent components.

Gulp

What is Gulp?

Gulp is, simply put, a task runner.

- You think of tasks. It runs these tasks.

Gulp is a node module that is centralized on the concept of using streams and manipulating their data

Gulp is a command line tool, that is installed through a global node module known as *Gulp*.

What will our first task do?

Our first task will take a bunch of SASS code and turn it into a single CSS file

- Grab all scss files we want to compile
- Initialize a sourcemap
- Compile the SASS into CSS
- Add prefixes to increase browser support
- Concatenate all the files
- Minify the result
- Save the result to file

What modules?

gulp-sass

- Compiles SASS into CSS using libsass
- https://www.npmjs.com/package/gulp-sass

gulp-sourcemaps

- Creates a map from old code to newcode
- https://www.npmjs.com/package/gulp-sourcemaps

gulp-concat

- Concatenates files.
- https://www.npmjs.com/package/gulp-concat

gulp-clean-css

- Minifies our CSS
- https://www.npmjs.com/package/gulp-clean-css

gulp-autoprefixer

- Adds useful vendor prefixes for browser support
- https://www.npmjs.com/package/gulp-autoprefixer

Optimizations we will use

clean-css

- clean-css will remove redundancies and manipulate the CSS output to be as minimal as possible with perspective of units and rulesets.
- It can be configured to perform an extremely high degree of manipulation.
- One of its other uses are to minimize all whitespaces and make the CSS

auto-prefixer

- auto-prefixing allows developers to write code without vendor prefixes, and automatically add them to the output.

concatenation

- concatenation is the process of taking many files and them into one giant file so that the browser does not have to make many, many HTTP requests. concatening

Source Maps

A source map is a way to inform browsers how to interpret optimized code and allow developers to debug it as if it were the non-debugged version of the code.

Source maps are usually stored in a secondary file so that it does not need to be downloaded during the initial run of the code.