mrmrs / **scalable-css-draft.md**
Last active 2 days ago

WIP thoughts on my last few years thinking about how to scale css for large and small teams working on large and small web applications.

`scalable-css-draft.md`

How not to scale css

Several years ago I got curious about how css worked at scale. When I first started out, there weren't nearly as many learning resources as there are now. CSS zen garden was amazing, at the time it showed how much you could change a design without altering the html.

In the beginning, that's what people sold me as a feature. By writing css, you could make a change one place and have it propagate everywhere. In principle this sounds pretty good. I'm lazy so I like doing things one time. But eleven years later, my experience on both large and small teams is that this is the most terrifying thing about css.

https://twitter.com/thomasfuchs/status/493790680397803521

In the past few years a lot of very smart people have been thinking more about CSS and this has lead to some fascinating discussions around how to build 'scalable' ui and how that relates to CSS. When I first started to think about scalability I naturally started to read every blog post and watch every tech talk I could get my hands on. There are a number of suggested 'best practices' around how sass/less/stylus etc. will help you build maintainable css. How mixins can make your code more DRY. How extends will keep your markup clean and pretty to look at. How BEM will make your code so perfect you want to cry.

But what is reality? What actually gets shipped to production? What do all of these tips and tricks do to our production css? How does it affect the whole team? More important, how are your end users affected?

When it comes to designing and shipping products I like to think about reality. Which can be challenging sometimes. I've sat in too many meetings where people don't want to hear or talk about reality. They talk in empty phrases about hypothetical possibilities. I like to do drugs and talk like that too sometimes. But, if you don't accurately assess where you currently are and what your reality is, it can be pretty tough to figure out what your problems are. And the chance of finding solutions to unidentified problems is, well, not good.

So there I was, in the fall of 2013, a new employee at an award winning tech company that had a website. And that website had a lot of css. Wanting to familiarize myself with the codebase - I opened up dev tools and started to read the site's css.

Line by line. From start to finish.

Some of the code made me laugh.

```
/* Start of reusable styles here */
```

Some of the code made me cry.

```
.homepage-promos .grid-50 {
    width: 100%
}
.homepage-promos .grid-33 {
    width: 100%
}
```

```
.homepage-promos .grid-34 {
    width: 100%
}
```

Some of it made me laugh and cry at the same time.

```
#seo-container {
    display: none;
}
```

Some of it made me wonder where numbers come from

```
.product-tab {
  height: 530px;
  width: 99.7%;
}
```

Some of it made me thankful that I have read the spec

```
  .container-inner.promo-status {
     float: center;
  }
```

Eventually, I stumbled across this class of code.

```
.left {
  float: left!important;
}
```

I saw this class and thought - well at least you know what it does. Unlike some of the other css it seemed like it was very reusable. By anyone who needed to float something left.

I kept reading and stumbled across this:

```
.left-max-scr1,
.left-max-scr2,
.left-max-scr3,
.left-only-scr1 {
    float: left;
}
```

…followed by this very similar block of code…

```
.left-min-scr2,
.left-min-scr3,
.left-max-scr3,
.left-only-scr3 {
    float: left;
}
```

I found myself wondering what the story was behind this next code block. Is it for .header-nav-list elements that aren't nested inside a .header-nav-container element?

```
.header-nav-container .header-nav-list {
    float: left;
}
```

```
.CA .header-nav-list.second {
    float: left;
}

#nav.challenger-a .submenu-3col li,
#nav.challenger-a .submenu-3col li {
    float: left;
}
```

Here we start to see a mixture of content-semantic (.submenu) with visually semantic class names (-3col).

```
.ie6 #footer-content .flex-control-nav li a,
.ie7 #footer-content .flex-control-nav li a,
.ie8 #footer-content .flex-control-nav li a {
    float: left;
}

#nav.challenger-a li.menu-products {
    float: left;
}
```

CSS is interesting because unlike other types of code, you are most likely to find the most recently towards the bottom of the file. Reading a css file from start to finish will often times reveal in chronological order how a defined system breaks down.

In the above examples we see things breaking down to increasingly longer selectors that have more and more weight that were only setting one property to one value: float: left.

Mind you, there are only three options for float. Contrary to one of the code examples above, there is no float center. Just left, right, and none.

Reading the css made me think about a lot of things. One thing I couldn't get out of my head was - why all this work to change one thing? Why write all that code to just float something left? To me, this is a reflection of the mental model most people are in when they are writing front-end code. They are generally trying to change one or two things in an interface and that's it. And when you are in this mode of development you want to limit the amount of things you break - while also making the change that is being asked for in some jira ticket that got assigned to you.

I also started to think about communication and how it moves in multiple directions. If I'm reading html, I want to know what the css is going to do. If I'm reading css I want to know what will happen if I apply it to a block of html. In a great system there is a two way street of information. If you look at the CSS, you can tell what will happen. If you look at the html, you know what the code will do.

But most front-end systems are not great. You have what I refer to as one-way streets of information. If I am looking at a block of html and I can't find ALL of the css that will visually affect how it renders in under 5 seconds, I feel that system is failing. So if I ran into the above example: `.container-inner.promo-status` That means I would have to find elements that had both of those classes (but not limited to) on an element. But by looking at this block of css I also have no idea if it has been redefined elsewhere. So I must grep through the system to find definitions for container-inner and promo-status. By the time I found those definitions - I would probably have forgotten what I was trying to do in which case I'd leave my desk and go get coffee.

On the flip side think about this example:

```
.red {
    color: #FF4136;
}
```

I'd presume that this would make an element have red text. I was traditionally told this is a horrible name for a class. But I

quite like it. The only way I'd think it was bad is if it set the color property to purple, blue, or any other color that wasn't red. Or if it was redefined a few times to a few different shades of red. But more on that later.

And if I saw this block of html:

```
<div class="red">Some text</div>
```

I'd know without looking at the css what was going to happen. This to me is a two ion way street of information. It's easy to learn a group of small classes that do one thing well that you an use anywhere. I should have a one to one mapping between a class and it's use. This concept of immutability is not new - but we've ignored it in the css community for far too long. Imagine if you had a function called 'filesize' where you passed it a filename and it returned the filesize. That sounds pretty great. But imagine if sometimes it returned the amount of lines in the file. That doesn't sound great at all does it. Well that's what you're doing anytime you redefine a css class. It's the worst.

When we look at convoluted css, we must remember that behind every css ruleset is a story. Maybe the author was on a time crunch and didn't have time to look up if there was some previously written code they could reuse. Maybe they don't care about css at all and the second something works they save, commit, and go back to writing monads. When I write bad code, it generally isn't because I'm not trying to write good code. There are generally other forces at work. What are the forces that work against writing clean reusable code? How do we get rid of them? I've spent the last few years doing a lot of user testing centered around how people write their own front-end code, consume other peoples front-end code, and think about developing UI in general. It has been a pretty valuable experience (as user-testing generally is). I've learned a lot about the different mental models people use - and how to try to construct systems that allow people of all abilities to spend more time designing / building and less time debugging/fighting/crying.

Sidenote (I don't know where to fit this in) Before I started doing both performance and user testing I did not like the idea of oocss / atomic css. I really liked writing monolithic classes and traversing the dom with my selectors. It made a ton of sense to me and I was able to build out a lot of websites that way. Running tests and testing my assumptions helped inform a lot of how I think about code now. I'm not very interested in what I can do with css. I'm interested at this point in what I can help groups of people do with css.

If you're going to build a new component, or change a piece of UI in your app - what do you do? I don't know anyone that reads all the available css in the app to see if there is something they can reuse. Who has time to read thousands of lines of code before they start working!? Even if people do have the time, I have not found that this is their first instinct on how to get going. I don't blame them.

Even if you did read all of the available css and stumbled upon some code that you think might be reusable - what if someone edits it later? If you are starting from the assumption that your css isn't reusable, your first instinct is to write new css. But most likely you aren't creating a new classification of visual styles. In my experience it is likely that you are replicating visual styles that already exist.

Salesforce may pay people a lot of money. And they may win Forbes awards for being the most innovative. But they are not known for hiring people who are good at css. Their css is anything but innovative. After reading all of the css and going through bits of crying, head-scratching, and laughing out loud - I determined that this problem must only exist at Salesforce. Surely awesome companies like Medium, GitHub, Adobe, and Pinterest didn't have this problem. They've hired some of the most amazing CSS developers I've ever met. They most have figured out how to scale css.

Naturally I opened up their sites and started to read their css too. Line by line. From start to finish. I wanted to see what everyone's reality really was.

I found the exact same things.

Let's take a look at just the rulesets that set things to display none:

GitHub https://gist.github.com/mrmrs/786241f0a5fade0324e2

Pinterest https://gist.github.com/mrmrs/57705f9a9fdce4d3d6f7

Medium https://gist.github.com/mrmrs/07e2ad668bac33e2da67

Adobe https://gist.github.com/mrmrs/2d5592826adb45748bac

That's a lot of code setting things to display: none. If you look at the files above ask yourself some questions: Do they seem like they follow the principles behind DRY? Does this seem like a bunch of code you would know how to reuse?

If I'm going to write css, I want it to be reusable. If someone isn't going to reuse it - it seems pretty useless.

So what does DRY even mean? Because you do have to repeat yourself somewhere! You either repeat yourself in html or in css. But no matter how good you are at html - you can't build out a new component without editing html. But, it is possible to build a new component without writing a single line of css. And this applies to changing UI too. You should be able to change most things, just by editing HTML.

When you repeat yourself in HTML, there isn't any real damage to file size (read: multi-class patterns will not bloat your HTML). A user doesn't have to download every html file from your website to view one page. Yet most web sites are architected in a way that require you to download the css for the entire site when you try to view one page. This is a broken model.

The files I referenced above are the result of css authors generating really long selectors that add a lot of weight to your cascade to do one or two things. And as time goes on there are more and more things in your cascade to override. So your selectors get longer and longer, your file sizes get bigger and bigger, and the amount of time you spend debugging css is going to go up and up and up.

In this model, you will never stop writing css. Refactoring css is hard and time consuming. Deleting css is hard and time consuming. And more often than not - it's not work people are excited to do. So what happens? People keep writing more and more css.

This affects scalability in two ways. One, it makes it harder to work on your app, but it also means that your users have an increasing amount of code to download. And if you think "oh but it will never get that bad" - well you're wrong. Pinterest has more than 1mb of css spread out of 5 files [todo: calculate gzipped file size]. That is a lot of css. 97% of it isn't used on pinterest.com. I'd rather just try and send my users the 3% they do need.

Part of this code bloat is an attachment to authoring 'content-semantic' class names. Honestly I'm surprised you can find people that still think this isn't the worst idea ever. Nicolas Gallagher already wrote the mic drop piece on why this doesn't work: http://nicolasgallagher.com/about-html-semantics-front-end-architecture/

So we come to the crux of the problem I have with any system that requires you to map visual styles to components in your css. Content semantics have NOTHING TO DO WITH VISUAL STYLES. When I used to build things with legos I never thought 'oh this is a piece for an engine block' I thought 'oh cool this is a 1x4 blue lego and I can do anything I want with it'

It's all about lego bits. Because I never needed to re-contextualize my understanding of lego blocks. I could use them across 'projects' and they were always the same. I dream of this world in front-end development. Give me lego blocks that work everywhere.

Outside of some brand specific background-images, gradients, and colors etc., the overwhelming majority of css you would need for your site has already been written. Yet we continue as a community to constantly reinvent the wheel. Which is starting to feel like building a camera from scratch every time you want to take a photo. I totally think building cameras from scratch is cool and a worthwhile endeavor. But I don't want to do it every time I am going to take a photo. I just want to go take photos.

When I read about or listen to ideas on how to scale an app's css - most of the talk is about how to write css. The real way to scale css, is to stop writing css. Abstract out the things you use most - and move to a multi-class pattern where you compose lego bits in your html. You'll be amazed at how quickly your team starts to move.

There are a lot of problems that come a long with people writing css. Inconsistent values, the introduction of magic numbers, duplicate code, non-responsive patterns. I'm sure there are others.

If you create a system that is flexible and powerful, and pull from that, you might find your designs to be more harmonious.

You might spend less time debugging the cascade. Your pages will probably load faster. But who knows. Maybe that won't work for you at all. All I know is the current model is definitely not going to work.

Writing new css should be the exception, not the rule.

lachlanjc commented 2 days ago

DLavin23 commented 2 days ago

Awesome write up! Thank you for taking the time to articulate these thoughts. The notion that our current model encourages writing more CSS was a pretty eye opening slap in the face; I never thought of it like that. Do you know of any companies that are using functional CSS patterns like this at scale?

marcusandre commented 2 days ago

Wow! Amazing work. Sometimes it feels that CSS scales with every team member into the wrong direction. It should be quick and easy. A parallel workflow — just styling. It's hilarious how hard it is.

"The real way to scale css, is to stop writing css." — is a key point.

I love it. Great draft. :)

colepeters commented 2 days ago

> They talk in empty phrases about hypothetical possibilities. I like to do drugs and talk like that too sometimes.
> 😆

<style>

```
CSS is interesting because unlike other types of code, you are most likely to find the most recently towards the bottom o
```

…you are most likely to find the most *recent code* towards… ?

```
You have what I refer to as one-way streets of information.
```

…a one-way *street* of information ?

```
This to me is a two ion way street of information.
```

I think a rogue *ion* got stick up in therre.

```
that do one thing well that you an use anywhere.
```

…that you *can* use anywhere.

```
between a class and it's use
```

its

```
it generally isn't because I'm not trying to write good code.
```

The double negative is confused. Perhaps *it generally isn't because I'm trying to write bad code.* ?

<article>

```
This concept of immutability is not new
```

In this context, it feels like you're talking more about immediacy or understandability, rather than immutability. You can have a one-to-one mapping of how something works, but technically that doesn't mean it's immutable in the programmatic sense (IMHO).

```
Salesforce may pay people a lot of money.
```

I wonder if it's worth mentioning Salesforce here, as they aren't mentioned previously. I see how it flows into where you talk about how you wanted to see how other companies do CSS, but I also wonder if there's a way to cut to the chase here without dipping into this specific case? I think the paragraph would be stronger you just started it as `Working at that job in 2013, I determined that this problem must only exist within this company.` … and continue from there.

```
The real way to scale css, is to stop writing css.
```

You could make this argument even stronger by providing some examples of where this has worked, IMHO. I've found when discussing this topic that readers gain a lot of value from seeing how things have been done differently, and what the direct benefit has been. You've clearly articulated some practical examples of how content-semantic classes are shitty. Providing some examples of how single-purpose classes/multi-class have saved you file size, cut out complexity, and made systems more understandable would be a great thing to include to seal the argument.

This is a great write-up. A few refinements and some practical examples of how your idea works well, and it'll be sitting real pretty.