CSS Architecture

A high level view of the common css approaches. This work was inspired by the book Enduring CSS by Ben Frian.

Specificty Problem for large scale applications

As developers we are often faced with the challenge of trying to design our system in a way which is easy to read and reason about. The cascading nature of css can create a number of headaches because of how specificty is assigned. The different problems outlined below attempt to solve the specificty issue through a conscious design of the css naming convention.

This table summarizes problem of name clashing

selector	inline	ID	class	type
.widget	0	0	1	0
aside#sidebar .widget	0	1	1	1
.class-on-sidebar .widget	0	0	2	0

1. different weights of selectors are confusing to learn and understand
2. can lead to the use of !important when specificty is not understood, effectively creating a specificty war
3. leads to css styling that is hard to reason about, and bloated selectors which obfuscate the relationship between the DOM element and the actual style applied to it

> Low specificity: If specificity battles start between selectors, the code quality starts to nosedive. Having a low specificity will help maintain the integrity of a large projectâ€™s CSS for a lot longer.

Limitations of IDs as selectors

> In summary, they are far more specific than a class selector - therefore making overrides more difficult. Plus they can only be used once in the page anyway so their efficacy is limited.

However, to keep specificty lower you can do the following:

```
[id="Thing"] {
/* Property/Values Here */
}
```

Object Oriented CSS

The best exmple of this approach is Atomic CSS where the class name is composed of different features which make up or compose the overall styling. The philosophy behind atomic css is to break the css into a variety of building blocks which can be declared inside the element. In this way, the component retains control of the styling, and the user can create variables and customize the stylistic building blocks for the application.

Here is on example:

```
    <div class="Bgc(#0280ae.5) C(#fff) P(20px)">A big cal
```

In this case, Bgc would be background color and the element inside the () is the value for that background color. Taking this one step further, you could even write `Bgc(grey-color)` and abstract the grey-color into a variable which can be used throughout your application.

CSS

Using the shorthand nomenclature for Atomic css, the grid system can be vastly shortened. Here is an example of an atomic css grid system for inline blocks:

```
<div>
   <div class="IbBox W(1/3) P(20px) Bgc(#0280ae.5)">
   Box 1</div>
    <div class="IbBox W(1/3) P(20px) Bgc(#0280ae.8)">
    Box 2</div>
    <div class="IbBox W(1/3) P(20px) Bgc(#0280ae">
    Box 3</div>
</div>
```

Above, the IbBox determines the type to be an inline-block, the width is 1/3, P(20px) = padding 20px, and Bgc is background color and the .5 is opacity 50%.

Pros

- Variables can be passed along in a more intuitive manner, closer to the DOM structure.
- Grid system is more obvious than a standard bootstrap row approach which abstracts the underlying CSS.
- Is more composable and reduces the effort needed to override using stylesheets

Cons

- Responsive Web design is still a pain because you need to add a media breakpoint name to an atomic class in order to apply specific styling. So you end up with something like this:

```
<div class="D(ib)--sm W(50%)--sm W(25%)--lg P(20px) Bg
<div class="D(ib)--sm W(50%)--sm W(25%)--lg P(20px) Bg
<div class="D(ib)--sm W(50%)--sm W(25%)--lg P(20px) Bg
<div class="D(ib)--sm W(50%)--sm W(25%)--lg P(20px) Bg
```

(In which case it is almost better to use media querieis
in css style sheets.)

- sometimes you really don't want this level of detail when composing an element, but a more general component style will suffice such as `label`

Scalable Modular CSS

The idea behind this approach is really to organize your code or group the css into flexible elements which also promote long term maintainability. SMCSS uses the following categorizations for css:

- Base
- Layout
- Module
- State
- Theme

```
<div class="c-card c-card--is-active">
    [â€¦]
</div>
```

In this case it almost seems better to use is-active class to address simple state change since this indicate DOM element change, not a component modifier.

BEM naming enhancements aka BEMIT

Hungarian Notation, you can use c-, for Components, o-, for Objects, u-, for Utilities, and is-/has- for States (there are plenty more detailed in the linked post).

Responsive suffixes using @ breakpoint.

```
<div class="o-media@md  c-user  c-user--premium">
  <img src="" alt="" class="o-media__img@md  c-user__phot
  <p class="o-media__body@md  c-user__bio">...</p>
</div>
```