**React**

**Day 60th**

**Why Module CSS??**

CSS has a series of design principles (global scope and the cascade) which have some fundamental problems in large web applications. As applications grow so does the amount of CSS needed to style a web application, over time it becomes harder to maintain as CSS changes for a new feature may have unintended consequences. Over time the web development community has come up with techniques and conventions to scale CSS like OOCSS, BEM or SMACSS which amongst other things focus on creating component based CSS (tied a portion of the DOM, a component). All of the above are based on following naming conventions that are verbose and hard to enforce consistently.

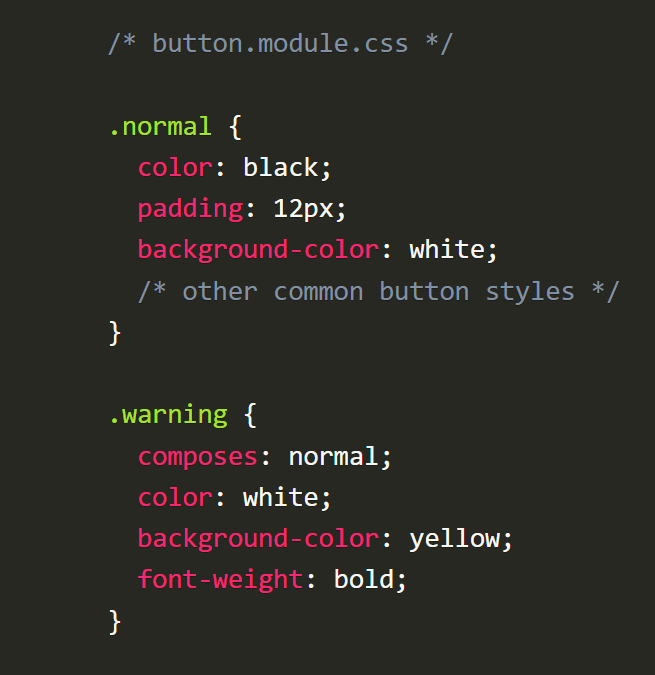
CSS modules are another approach that aims to address these CSS limitations and are designed with component based styles in mind:

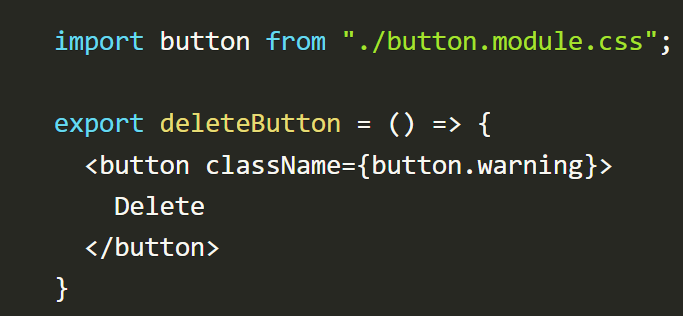
* They are locally scoped (So they ensure there’s no problems with style collisions or overrides)
  + No uncertainty about which components get affected when changing some CSS
  + No uncertainty about whether some CSS is unused or not
  + Since CSS defined a module is only used within a small scope we can use short CSS names and take advantage of the context provided by the CSS module file itself. E.g. button\_warning is no longer necessary to prevent collisions between warning like components since we already are in a button.module.css module with local scope we can rely on a .warning CSS class as a name.
* The explicitly define dependencies. So you can very quickly see which styles are applied to given component or template
* They provide a mechanism for composition that allows to
  + Reuse styles
  + Define a higher level semantic language we can rely on to promote consistency and reuse of styles within an application. E.g. instead of everyone defining specific font styles using basic CSS rules we can composes: large from ‘./typography.module.css’ to have all components with a large font use the exact same CSS rules.

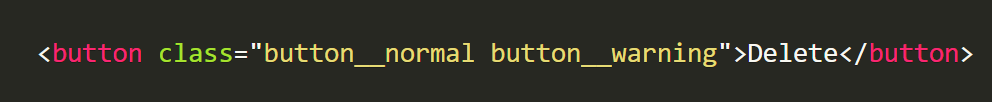
**CSS modules feature**

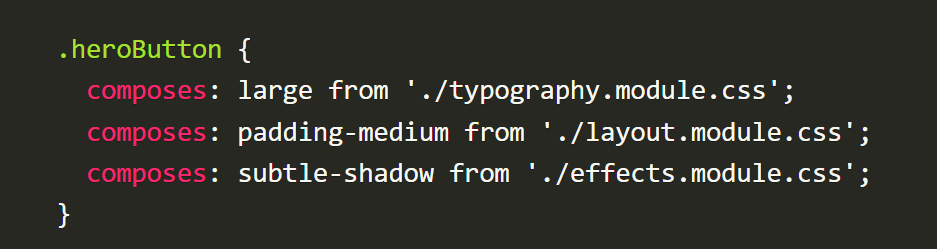
1. Local Scope by default: CSS in CSS modules is locally scoped. This is achieved because CSS modules are a form of CSS-in-JS that uses a build step to generate uniquely identified classes.
2. Compositions: CSS classes in CSS modules can be composed to increase reuse and modularity
3. Values: Values can be defined in CSS modules and exported in a similar fashion to LESS and SASS variables
4. Global scope escape hatch: You can refer to CSS classes in global scope using ‘:global’
5. Theming: CSS modules can be used for theming by having a component obtain a set of styles (the theme) via prop instead of importing the CSS module from the component itself.

**Compositions**

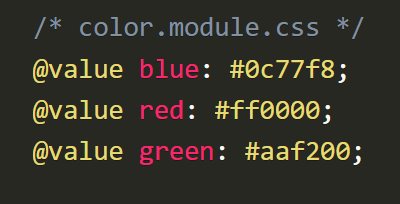
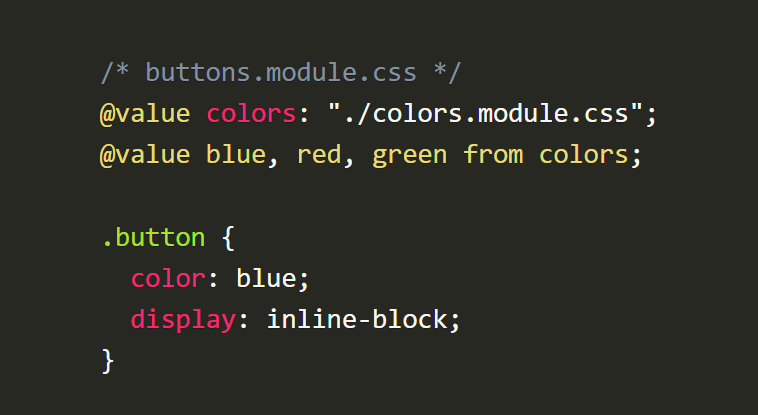
CSS modules support style composition through the use of the composes keyword. To have one style compose with another style you use composes like so:

We can then apply the normal and warning classes in JavaScript components.

As a result of the CSS module composition, the generated component includes both normal and warning classes.

It is also possible to compose classes from other CSS modules. This enables the possibility of writing a series of common higher level styles that can be reused within your application for higher consistency and reuse.

**Values**

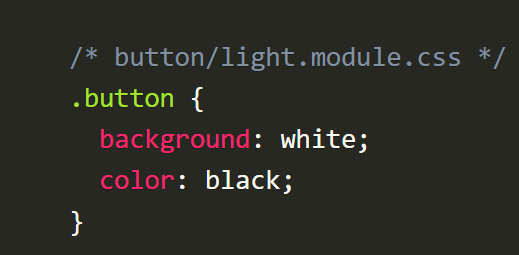
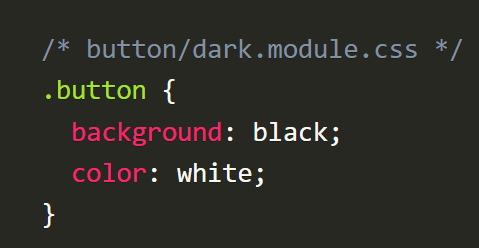
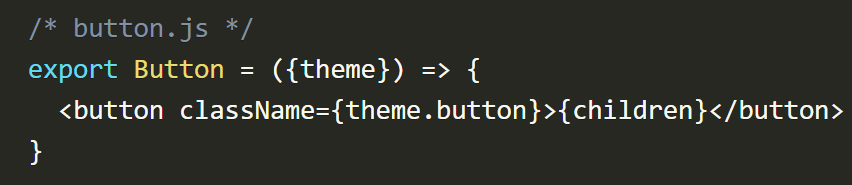
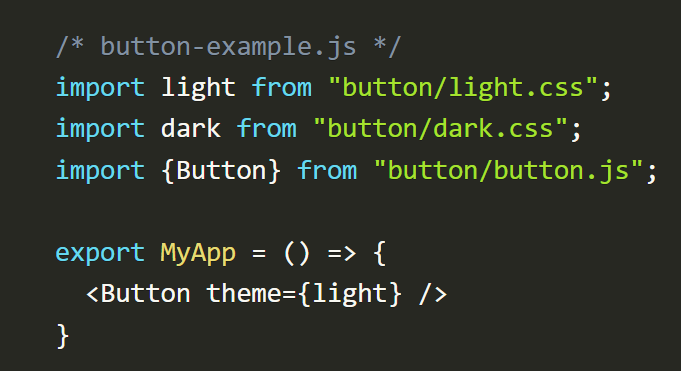
CSS modules support values in a similar fashion to LESS and SASS variables.

Values don’t need to be single CSS properties. For example these are valid values too.

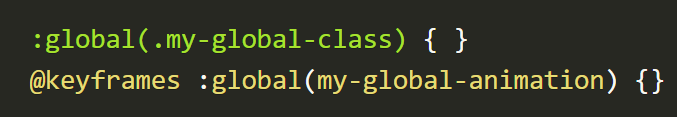
****/\* media.module.css \*/

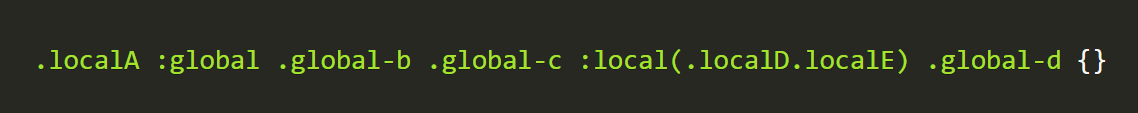
@value laptop: only screen and (min-device-width : 768px) and (max-device-width : 1024px)

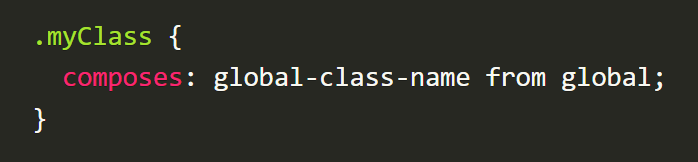
**Theming**

****The natural way to work with CSS modules is that one designs self-contains components where the styles are scoped and contained within the component itself. This gives us all the benefits that we’ve described above on simplicity and maintainability. There’s one case where it makes sense to style a component from the outside, and that’s theming, i.e. the ability to have a component be able to have multiple styles or themes. CSS modules are really good at this because at the end of the day they are a collection of classes and rules that can be passed to a component via props making it a really good fit for theming in React applications:

**Global Styles in CSS modules**

To create global styles in CSS modules use the :global switch.

To refer to global styles in CSS modules use :global like so

To use a global class name when composing styles: