Scalable Front-End Architecture using SaSS

What is SaSS?

- a CSS Preprocessor
- Sass has lots of interesting features
  - Variables
  - Nesting
  - Mixins
  - Selector inheritance
  - and more
- Sass has two syntaxes. The main one being SCSS

Common SaSS Mistakes

- Over use of nested selectors.
- Unintentional Duplication of CSS Properties.

Common SaSS Mistakes

- Over use of nested selectors.
- Unintentional Duplication of CSS Properties.

Over use of nested selectors.

Say no to nesting.

An example of nesting could look something like this:

```
.module {
  .hd {}
  .bd {
    ul {
      li {
      }
    }
  }
}
```

Over use of nested selectors.

this would compile to the following:

```
.module {}
.module .header {}
.module .bd {}
.module .bd ul {}
.module .bd ul li {}
```

This implementation is less then flexible:

- Need a few reasons here.

Over use of nested selectors.

A more flexible approach could look something like this:

.module {}
.module-hd {}
.module-bd {}
.module-list {}
.module-item {}

The benefits being:

- Need a few reasons here.

Common SaSS Mistakes

- Over use of nested selectors.
- Unintentional Duplication of CSS Properties.

Unintentional Duplication of CSS Properties

Take the following sass snippet:

```
@mixin inline-block { display: inline-block; }
.module-a { @include inline-block; }
.module-b { @include inline-block; }
```

which compiles to:

```
.module-a { display: inline-block; }
.module-b { display: inline-block; }
```

notice that the properties are duplicated between both rules.
This is not ideal.

Unintentional Duplication of CSS Properties

Instead, create a base class and extend from it.

```
.ibf { display: inline-block; }
.my-module-a { @extends .ibf; }
.my-module-b { @extends .ibf; }
.my-module-c { @extends .ibf; }
```

which compiles to:

```
.ibf,
.my-module-a,
.my-module-b,
.my-module-c { display: inline-block; }
```

Unintentional Duplication of CSS Properties

If the base class is not used in your markup than, use a placeholder selector.

```
%ibf { display: inline-block; }
.my-module-a { @extends %ibf; }
.my-module-b { @extends %ibf; }
.my-module-c { @extends %ibf; }
```

which compiles to:

```
.my-module-a,
.my-module-b,
.my-module-c { display: inline-block; }
```

notice that .ibf is not in the compiled set of rules.

A few best practices

SCSS Directory Structure

- layout/ -- collections of "modules"
- module/ -- self-contained "modules"
- _base.scss -- structure of your sites layout
- _helpers.scss -- common utilities used for building a layout
- _state.scss -- global, layout, or module state changes
- _theme.scss -- visual theme of your layout
- style.scss -- inherits everything above.

Thinking about layouts

- A Layout structures the way modules are laid out on a page
- No color style should be applied directly on a layout
- things like margin, padding, width, height, transitions media queries should be used for style in layouts.
- layouts should be prefixed with an "L" to specify its a layout.
  - example: .l-constrained

Thinking about modules

- A module structures the way content is laid out on a page
- Any style can be applied to modules.
- if your module name is .test you should prefix every child class of .test
  - example: .test-header
- use placeholder selectors for base modules that are commonly used that you don't need to access in your markup.

Thinking about _base.scss

- _base.scss is used to style the structure of your layout.
- things that are considered structural properties:
  font, box and float properties.

Thinking about _helpers.scss

- _helpers.scss is used for common patterns to help you with the structure of your layout.
- things you would consider making into helpers:
  text-alignment, floats, vertical-alignment, etc.
- use placeholder selectors if you don't plan to use them in your markup.

Thinking about _state.scss

- _state.scss is used for anything that modifies the original state of any part of your layout.
- you can prefix your sites layout (html, body, main content layer) with a class of something like .is-active or .is-valid
- the reason for the prefix of the state classes is so you can make a direct connection in your markup as to what kind of class is being applied to the particular element in question.
  - example "This element IS active", so now you can immediately know which file to check for this class.
- If you have a module state change, put those states in that module.

Thinking about _theme.scss

- _theme.scss is used for anything that directly correlates to the visual style of your site.
- properties that are considered theme:
  background color, text color, text-shadow, box-shadow, etc.
- One reason why you should do this is if you have a designer that would like to work on the sites theme, they can without the worry of messing anything up, because this file only contains the colors of the site, so the structure of each module will stay perfectly intact.

Thinking about style.scss

- style.scss is where all of your modules, layouts, helpers, etc get inherited
- Optionally include a css reset file here.

Other considerations

- If you need to namespace your modules so others can use them on their sites, you can just create a separate CSS file and include the modules under an ID or class to keep your selectors from clashing with theirs.
- If you'd like to support IE but also have the option to drop support later with ease, you could create a _ie.scss file that would have all the abstracted bug fixes for your styles and then you can remove the include for _ie.scss in your style.scss to easily drop support.

Tools of the trade

- Zen Coding
- Sublime Text 2 (it's awesome)
  - Package Manager, Python Interpreter, TextMate Bundle Support
- Photoshop Extensions
  - CSS Hat
  - Guide Guide
- Dwarf - On Screen Rulers / Guides

It's time for the code!