CSS LESS Style Guide

dcneiner edited this page on Nov 22, 2014 · 4 revisions

Note: This is copied and adapted from Medium's LESS Guidelines.

LeanKit's naming conventions are adapted from the work being done in the SUIT CSS framework. Which is to say, it relies on *structured class names* and *meaningful hyphens* (i.e., not using hyphens merely to separate words). This is to help work around the current limits of applying CSS to the DOM (i.e., the lack of style encapsulation) and to better communicate the relationships between classes.

Table of contents

JavaScript

Note: The following syntax should normally not be needed with React.js/Lux code. However, for new work on the existing infrastructure, this pattern has been included.

syntax: `js-<targetName>`

JavaScript-specific classes reduce the risk that changing the structure or theme of components will inadvertently affect any required JavaScript behavior and complex functionality. It is not necessary to use them in every case, just think of them as a tool in your utility belt. If you are creating a class, which you dont intend to use for styling, but instead only as a selector in JavaScript, you should probably be adding the `js-` prefix. In practice this looks like this:

```
<a href="/login" class="btn btn-primary js-login"></a>
```

Again, JavaScript-specific classes should not, under any circumstances, be styled.

Utilities

LeanKit's utility classes are low-level structural and positional traits. Utilities can be applied directly to any element; multiple utilities can be used together; and utilities can be used alongside component classes.

Pages 11

Home

Common Global NPM Modules at LeanKit

CSS LESS Style Guide

Formatting JavaScript

Install Node.js

LeanKit Git Workflow

Linting JavaScript

Setting Up Atom

Setting Up Sublime Text 3

Setting Up Visual Studio 2013

Update launchd.conf to Set Your PATH

Clone this wiki locally

https://github.com/LeanKit-

Clone in Desktop

Utilities exist because certain CSS properties and patterns are used frequently. For example: floats, containing floats, vertical alignment, text truncation. Relying on utilities can help to reduce repetition and provide consistent implementations. They also act as a philosophical alternative to functional (i.e. non-polyfill) mixins.

```
<div class="u-clearfix">
  <p class="u-textTruncate">{$text}</p>
  <img class="u-pullLeft" src="{$src}" alt="">
  <img class="u-pullLeft" src="{$src}" alt="">
  <img class="u-pullLeft" src="{$src}" alt="">
</div>
```

u-utilityName

Syntax: `u-<utilityName>`

Utilities must use a camel case name, prefixed with a `u` namespace. What follows is an example of how various utilities can be used to create a simple structure within a component.

```
<div class="u-clearfix">
  <a class="u-pullLeft" href="{$url}">
    <img class="u-block" src="{$src}" alt="">
  </a>
  <p class="u-sizeFill u-textBreak">
    ...
  </p>
</div>
```

Components

Syntax: `<componentName>[--modifierName|-descendantName]`

Component driven development offers several benefits when reading and writing HTML/React and CSS:

- It helps to distinguish between the classes for the root of the component, descendant elements, and modifications.
- It keeps the specificity of selectors low.
- It helps to decouple presentation semantics from document semantics.

You can think of components as custom elements that enclose specific semantics, styling, and behavior.

ComponentName

The component's name must be written in camel case.

```
.myComponent { /* ... */ }
```

```
<article class="myComponent">
  ...
</article>
```

componentName--modifierName

A component modifier is a class that modifies the presentation of the base component in some form. Modifier names must be written in camel case and be separated from the component name by two hyphens. The class should be included in the HTML *in addition* to the base component class.

```
/* Core button */
.btn { /* ... */ }
```

```
/* Default button style */
.btn--default { /* ... */ }
```

```
<button class="btn btn--primary">...</button>
```

componentName-descendantName

A component descendant is a class that is attached to a descendant node of a component. It's responsible for applying presentation directly to the descendant on behalf of a particular component. Descendant names must be written in camel case.

```
<article class="tweet">
  <header class="tweet-header">
    <img class="tweet-avatar" src="{$src}" alt="{$alt}">
    ...
  </header>
  <div class="tweet-body">
    ...
  </div>
</article>
```

componentName.is-stateOfComponent

Use `is-stateName` for state-based modifications of components. The state name must be Camel case. **Never style these classes directly; they should always be used as an adjoining class.**

JS can add/remove these classes. This means that the same state names can be used in multiple contexts, but every component must define its own styles for the state (as they are scoped to the component).

```
.tweet { /* ... */ }
.tweet.is-expanded { /* ... */ }
```

```
<article class="tweet is-expanded">
  ...
</article>
```

Variables

Placeholder

Colors

Placeholder

z-index scale

Please use the following z-index scale:

`@zIndex-1 - @zIndex-9` are provided. Nothing should be higher then `@zIndex-9`.

Font Weight

Placeholder

Line Height

Placeholder

Letter spacing

Placeholder

Polyfills

Polyfills are provided by auto-prefixer and are (or should be) part of the build process on the LESS source files.

Formatting

The following are some high level page formatting style rules.

Spacing

CSS rules should be comma separated but live on new lines:

Right:

```
.content,
.content-edit {
  ...
}
```

Wrong:

```
.content, .content-edit {
  ...
}
```

CSS blocks should be separated by a single blank line.

Right:

```
.content {
  ...
}

.content-edit {
  ...
}
```

Wrong:

```
.content {
  ...
}
.content-edit {
  ...
}
```

Wrong:

```
.content {
  ...
}



.content-edit {
  ...
}
```

Quotes

Quotes are optional in CSS and LESS. We use double quotes as it is visually clearer that the string is not a selector or a style property.

Right:

```
background-image: url("/img/you.jpg");
font-family: "Helvetica Neue Light", "Helvetica Neue", Helvetica, Arial;
```

Wrong:

```
background-image: url(/img/you.jpg);
font-family: Helvetica Neue Light, Helvetica Neue, Helvetica, Arial;
```

Performance

Specificity