CSS Structure

Your classnames should be readable by humans but no longer than they have to be. Wherever possible, choose modular, reusable CSS over ultra-specific CSS. Selector inheritance should be *managed* with two levels of specificity wherever possible. Classes should generally *not* be used to attach javascript behaviours to components. Instead these should invoked/attached via data- attributes.

Collections

Use a plural/singular approach.

```
.columns
.column
```

Blocks

Use a BEM approach.

```
.article
.article__head
.article__title
```

Modifiers

Used a mixed approach. Very specific modifiers should be used like BEM dictates. Try and keep modifiers specific in scope and obvious to the reader.

```
.button--bordered
```

Common, global or near-global modifiers can use the following format.

```
.-text-lg
```

State

We borrow the “is” convention from SMACCS, with a twist. Is it a state? Anything that can be toggled is state. (active, hidden, open, closed etc.)

```
.-is-active
```

Responsive

Classes tied to Breakpoints are signified by @xs, @sm, @md, @lg, @xl

```
.-width-1/2@md
```

Format

To ensure that stylesheets are easier to read we follow a specific format when writing CSS. It's a good idea to get into the habit, future framework versions may include JSLINT to enforce them.

- Use four spaces to indent your code
- Utilize spaces to make your code easier to read (after commas, curly braces, colons)
- Break multiple selectors onto separate lines. Break multiple declarations onto separate lines.
- Avoid redundant values whenever possible

```
/* nicely formatted block */
.selector,
.selector2 {
    overflow: visible;
    background-color: rgba(255, 0, 0, 0.5);
    transform: translate(-50%, -50%);
}

/* not so much */
.selector,.selector2 {
    overflow:visible;background-color:rgba(255,0,0,0.5);
    transform:translate(-50%,-50%);
}
```

Selector Specificity

We're using a well-managed and self-*enforcing* CSS selector philosophy. Most rules should have a specificity of 2 and modifiers classes should appear after the core rules within the components/blocks stylesheet.

```
.button.button {
	/* two */
}

.button.button--bordered {
	/* two */
}
```

Javascript (Future)

Javascript event should be attached via data-attributes, not classes. Data attributes offer more flexibility in terms of configuration and can accept string or, for more complex behaviour, JSON objects.

```
data-tooltip='{
      "label": "Here's My tooltip",
      "Position": "bottom-center"
}'
```

Responsive

Media Queries

Keep media queries out of of blocks default behaviour whenever possible. Stick to using using classes to modify. This helps make elements more flexible and more reusable. One modifier for each type of modification:

```
.nav--stacked@sm
.nav--fixed@sm
```

vs.

```
.nav {
	@media only screen and (max-width: 823px) {
		/* mobile nav styles */
	}
}
```

Breakpoints

Stick to the existing breakpoints whenever possible. On the odd situation where these breakpoints will not work, add your media query to skin.less/skin.css or shame.css.

Class Modifier	Media	Min-Width	Max-width
@xs	screen	-	479px
@sm	screen	480px	767px
@md	screen	768px	991px
@lg	screen	992px	1999px
@xl	screen	1999px	-

Files & Locations

skin.css | skin.less

All things specific to the skin.

shame.css

Hacks, Things that require explanation, code that may need a refactor or is temporary.

More to Come...