fat Follow
computer loser / captioned co-founder (captioned.com)
Aug 28, 2014 · 9 min read

Medium's CSS is actually pretty f***ing good.

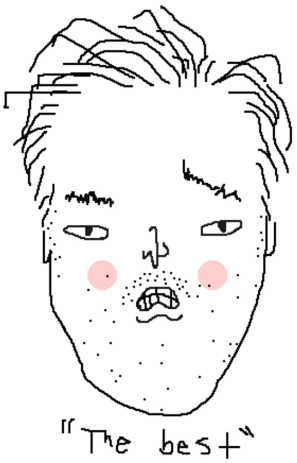
Listen to the audio version above? 🔈 😱

> *I always believe that to be the best, you have to smell like the best, dress like the best, act like the best. When you throw your trash in the garbage can, it has to be better than anybody else who ever threw trash in the garbage can.*
>
> *— Lil Wayne*

I've been meaning to write something about the CSS at Medium for a while because I'm not completely ashamed of it…

So how did that happen? What did we do differently? OMG, how can you do the same thing? Or learn from us, or something?

What follows are some notes on our CSS, the steps we've taken to get it to where it is, and the world we live in today.



The best at drawing pictures of myself

. . .

In the beginning (some history)

Roughly 2 years ago I joined Obvious Corp. to work on the software application, *medium.com* (which, you're ~hopefully~ reading on).

At the time, Medium had already gone through a series of "re-styles" (*A re-style is where designers ruin your life by coming up with something prettier than they had previously come up with, resulting in you having to rewrite a bunch of CSS/LESS/SASS/etc*). Because of these re-styles, there was a lot of built up cruft – the company was leaning pretty heavily on LESS's language features, with page driven semantics, non-sprited/non-retina image assets, and other such things.

I dug through git and managed to pull out our old profile page implementation from back in 2012 ~shudder~. Check it out below and notice some of these unfortunate patterns:

- everything nested under a .**profile-page** class, with zero reusable components (also ~almost~ everything is prefixed with **.profile**)
- super generic variable names like **@link-color** which weren't scoped to anything, but presumably only to be used in **profile-page.less**

- deep nesting (**.profile-page .profile-posts .home-post-summary .home-post-collection-image img** is an actual selector being generated below—super rough on perf)
- no image spriting
- no z-index scale, no font scale, no color scale, no scalesss…

```
// Copyright 2012 The Obvious Corporation.

@column-margin: 9px;
@link-color: rgb(78, 78, 78);
@link-hover-color: rgb(160, 160, 160);

.profile-page {
  margin: 0 0 0 55px;
  width: auto;
  font-family: helvetica, arial, sans-serif;
  line-height: 1.2em;

  .profile-left {
    width: 925px;
    float: left;

    h1 {
      height: 55px;
      line-height: 55px;
      text-transform: uppercase;
      font-size: 15px;
      font-weight: 700;
      padding-left: 30px;
    }

    .profile-details {
      background-color: rgb(47, 47, 47);
      padding: 47px 37px;
      position: relative;

      .profile-settings-link {
        position: absolute;
        width: 48px;
        height: 48px;
        right: 37px;
        top: 22px;
        text-indent: 1000px;
        overflow: hidden;
        background: url(/img/placeholder-gray.png) no-repe
      }

```

```
42          .profile-image {
43            float: left;
44            margin-right: 47px;
45
46            img {
47              width: 128px;
48              height: 128px;
49              .border-radius(64px);
50            }
51          }
52
53          .profile-name {
54            font-family: georgia, "Times New Roman", serif;
55            font-size: 32px;
56            font-style: italic;
57            font-weight: 400;
58            color: white;
59            text-shadow: 0 1px 0 rgba(0, 0, 0, 0.6);
60            text-transform: capitalize;
61          }
62
63          .profile-bio {
64            margin-left: 175px; // = 128 + 47
65            font-size: 16px;
66            line-height: 1.2em;
67            font-weight: 400;
68            color: rgb(224, 224, 224);
69          }
70        }
71
72        .profile-stats {
73          .clearfix;
74          height: 66px;
75          line-height: 66px;
76          padding-left: 28px;
77          border: 1px solid rgb(211, 211, 211);
78          border-top: none;
79          background: rgb(245, 245, 245);
80
81          .profile-stats-label {
82            float: left;
83            height: 66px;
```

```
84          line-height: 66px;
85          text-transform: uppercase;
86          font-size: 15px;
87          font-weight: 700;
88          color: rgb(77, 77, 77);
89        }
90
91        .profile-stats-item {
92          width: 139px;
93          height: 66px;
94          line-height: 66px;
95          border-left: 1px solid rgb(211, 211, 211);
96          float: right;
97          text-align: center;
98          font-family: georgia, "Times New Roman", serif;
99          font-size: 14px;
100         font-weight: 700;
101         color: rgb(153, 153, 153);
102         background: transparent url(/img/placeholder-gray.
103         padding-top: 16px;
104         .box-sizing(border-box);
105       }
106     }
107
108     .profile-posts {
109       border: 1px solid rgb(211, 211, 211);
110       border-top: none;
111       background: white;
```

https://gist.github.com/fat/a4af78882d0003d2345e

. . .

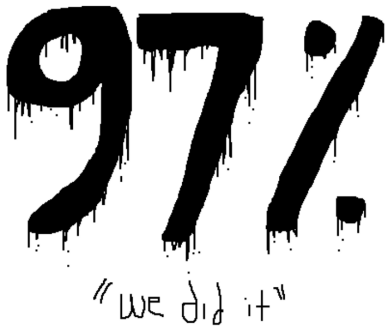
The 1st Project: OMG Images

At the time, I had been working a lot on library code (Bootstrap, Ratchet, etc.) and sweating all the details, trying to write the best CSS humanly possible.

Medium's CSS was clearly different. Not a neutral difference, but a soul crushing, shitty difference. And I wanted to fix that.

Looking at all there was to do, the first thing I tackled was images. I remember being pretty appalled that we weren't doing any sort of spriting in 2012… accumulating hundreds of image assets like placeholders, arrows, icons, etc. in this crazy /img/ directory which was something like a graveyard for icons.

To get away from that world I did two things—first, I wrote a little CSS utility script called SUS (which we're still using today and I've just open sourced here: https://github.com/medium/sus). This was largely written in IRC with the help of Guillermo Rauch, and is used to extract images from stylesheets and lazy load them in a separate, data-uri file. Not an original concept, just something simple that could be done to help us get to where we needed to go.



The second thing was Geoff Teehan and I sat down and created Medium's first icon font. At the time, we largely had no idea what we were doing… but with the help of icomoon, a github blog post, and a few long nights, we were able to ship something ~pretty~ good to Medium. This was huge because it meant we could delete like 97% of the img folder, everything looked great on my retina MacBook Pro, and we were requesting significantly fewer resources.

. . .

The 2nd Project: Scales

The next major project for me was the z-index scale. Z-index is one of those things which can get out of hand super easily and has always driven me crazy. I didn't want this to be the same at Medium.

Before this project began, it was very common to see one element with a **z-index: 5**, next to a sibling with a **z-index: 1000000;** and another sibling **z-index: 1000001;** and another **z-index: 99999;**.

The styles for these were spread all across the code base and there was no clear way of figuring out how to sort these.

So I took on the laborious task of manually auditing the entire codebase for z-index values. I then introduced a private scale (private to z-index.less) which could be used for z-indexing components (limiting it to 1-10). And finally, I moved all the actual assignments of this scale internal to z-index.less, so you could see how different components were situated relative to each other (which is actually pretty handy).

Below is the z-index file we are using on medium.com today.

```
// Copyright 2014 A Medium Corporation
//
// z-index.less
// Medium.com's z-index scale. Z-index values should always
// allows us to at a glance determine relative layers of ou
// arrising from arbitrary z-index values. Do not edit the
// scoped z-index values.


// Z-Index Scale (private vars)
// ---------------------------------------------------
@zIndex-1:   100;
@zIndex-2:   200;
@zIndex-3:   300;
@zIndex-4:   400;
@zIndex-5:   500;
@zIndex-6:   600;
@zIndex-7:   700;
@zIndex-8:   800;
@zIndex-9:   900;
@zIndex-10: 1000;


// Z-Index Applications
// ---------------------------------------------------
@zIndex-1--screenForeground:        @zIndex-1;
@zIndex-1--followUpVisibility:      @zIndex-1;
@zIndex-1--prlWelcome:              @zIndex-1;
@zIndex-1--appImageDropdown:        @zIndex-1;
@zIndex-1--surfaceUnder:            @zIndex-1;
@zIndex-1--blockGroup:              @zIndex-1;

@zIndex-2--surfaceOver:             @zIndex-2;
@zIndex-2--imagePickerControl:      @zIndex-2;
@zIndex-2--collectionCardButton:    @zIndex-2;
@zIndex-2--blockGroupButtonGroup:   @zIndex-2;
@zIndex-2--blockGroupFocused:       @zIndex-2;
@zIndex-2--blockGroupOverlay:       @zIndex-2;

@zIndex-3--caption:                 @zIndex-3;
@zIndex-3--blockInsertControl:      @zIndex-3;
```

```
42
43  @zIndex-5--figureOverlay:           @zIndex-5;
44  @zIndex-5--highlightMenu:           @zIndex-5;
45  @zIndex-5--metabar:                 @zIndex-5;
46  @zIndex-5--profileAvatar:           @zIndex-5;
47  @zIndex-5--noteRecommendations:     @zIndex-5;
48  @zIndex-5--collectionLogo:          @zIndex-5;
49
```

https://gist.github.com/fat/1f6da6b3bd0311a1f8a0

Of course, this z-index scale was quickly followed by a color scale (variables for blacks, grays, whites, brand-colors, etc) and a type scale (variables for font-sizes, weights, letter-spacing, line-heights, etc.).

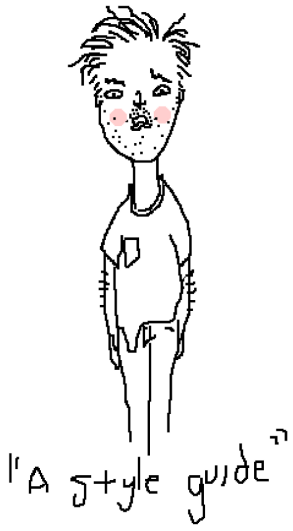
Also, you might notice the variable names have taken on some stricter, semantic value – more on that later…

. . .

The 3rd Project: New Style Guide

Not long after introducing scales to Medium, we started in on a big re-style code named “Cocoon”.

Cocoon deprecated several post templates (Medium originally had image templates and quote templates, not just a single article template) and also introduced post lists instead of post “cards.”



We took this as a chance to step back and write a new style guide for Medium.

This initial effort was shared between myself, Dave Gamache, Dustin Senos, and the talented Chris Erwin.

We took some time to really tighten up our thinking around writing production

CSS/LESS at Medium—the major updates being:

- Limit LESS to variables and mixins (no nesting, guards, extend, etc.)—while these language features ~can~ be powerful, we found that inexperienced LESS developers often got in trouble with them fairly easily. We also just preferred the visual aesthetic of pure CSS (and the consistency it afforded) and wanted to move our codebase as far in that direction as reasonable.
- Classes and IDs are lowercase with words separated by a dash—this is how most of us had been writing CSS at the time with Bootstrap, Skeleton, Ratchet, etc. And we thought it made sense to do the same here. Another reason is it followed the naming conventions set forth by the CSS language itself, i.e. border-radius-top-left, etc.
- Prefer components over page level styles—we wanted our front end to feel like library code, and began to break files like profile.less into smaller more focused files like button.less, dialog.less, tooltip.less, etc.

Here it is in its entirety:

LESS Coding Guidelines

The Obvious Corporation uses LESS as its CSS preprocessor. It's helpful to become familiar with its dynamic behavior before reading these guidelines.

Naming Conventions

Classes and IDs are lowercase with words separated by a dash:

Right:

```
.user-profile {}
.post-header {}
#top-navigation {}
```

Wrong:

```
.userProfile {}
.postheader {}
#top_navigation {}
```

Image file names are lowercase with words separated by a dash:

Right:

```
icon-home.png
```

Wrong:

```
iconHome.png
icon_home.png
iconhome.png
```

Image file names are prefixed with their usage.

Right:

```
icon-home.png
bg-container.jpg
bg-home.jpg
sprite-top-navigation.png
```

Wrong:

```
home-icon.png
container-background.jpg
bg.jpg
top-navigation.png
```

IDs vs. Classes

You should almost never need to use IDs. Broken behavior due to ID collisions are hard to track down and annoying.

Color Units

When implementing feature styles, you should only be using color variables provided by colors.less.

When adding a color variable to colors.less, using RGB and RGBA color units are preferred over hex, named, HSL, or HSLA values.

Right:

```
rgb(50, 50, 50);
rgba(50, 50, 50, 0.2);
```

Wrong:

```
#FFF;
#FFFFFF;
```

```
white;
hsl(120, 100%, 50%);
hsla(120, 100%, 50%, 1);
```

z-index scale

Please use the z-index scale defined in z-index.less. `@z-index-1 - @z-index-9` are provided. @z-index-9 is the `kill -9` of our z-index scale and really should never be used.

Font Weight

With the additional support of web fonts `font-weight` plays a more important role than it once did. Different font weights will render typefaces specifically created for that weight, unlike the old days where `bold` could be just an algorithm to fatten a typeface. Obvious uses the numerical value of `font-weight` to enable the best representation of a typeface. The following table is a guide:

Raw font weights should not be specified. Instead, use the appropriate font mixin:

`.wf-sans-i7, .wf-sans-n7, etc.`

The suffix defines the weight and style:

```
n = normal
i = italic
4 = normal font-weight
7 = bold font-weight
```

Refer to type.less for type size, letter-spacing, and line height. Raw sizes, spaces, and line heights should be avoided outside of type.less.

```
ex:

@type-micro
@type-smallest
```

```
@type-smaller
@type-small
@type-base
@type-large
@type-larger
@type-largest
@type-jumbo
```

See Mozilla Developer Network — font-weight for further reading.

Componentizing

Always look to abstract components. Medium has a very strong, very consistent style and the reuse of components across designs helps to improve this consistency at an implementation level.

A name like `.homepage-nav` limits its use. Instead think about writing styles in such a way that they can be reused in other parts of the app. Instead of `.homepage-nav`, try instead `.nav` or `.nav-bar`. Ask yourself if this component could be reused in another context (chances are it could!).

Components should belong to their own less file. For example, all general button definitions should belong in buttons.less.

Name-spacing

Name-spacing is great! But it should be done at a component level – never at a page level.

Also, namespacing should be made at a descriptive, functional level. Not at a page location level. For example, `.profile-header` could become `.header-hero-unit`.

Wrong:

```
.nav,
```

```
.home-nav,
.profile-nav,
```

Right:

```
.nav,
.nav-bar,
.nav-list
```

Style Scoping

Medium pages should largely be reusing the general component level styles defined above. Page level name-spaces however can be helpful for overriding generic components in very specific contexts.

Page level overrides should be minimal and under a single page level class nest.

```
.home-page {
  .nav {
    margin-top: 10px;
  }
}
```

Nesting

Don't nest. Ever.

Nesting makes it harder to tell at a glance where css selector optimizations can be made.

Wrong:

```
.list-btn {
  .list-btn-inner {
    .btn {
      background: red;
    }
    .btn:hover {
      .opacity(.4);
    }
  }
}
```

Right:

```
.list-btn .btn-inner {
  background: red;
}
.list-btn .btn-inner:hover {
  .opacity(.4);
}
```

Spacing

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

Right:

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

Wrong:

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

CSS blocks should be seperated by a single new line. not two. not 0.

Right:

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

https://gist.github.com/fat/b27700946c777adacdf4

It wasn't perfect by any means, but it cleaned up a few basic ideas and got us headed towards what felt like the right direction.

Unfortunately, even after this style update, people were still struggling with when to make a component, when to make a subcomponent, etc… And we were getting the occasional unfocused, run-on classname like: **.nav-on-light-background-button** or **.button-primary-sidebar-over-blur.**

People weren't prefixing classes at a page level anymore (which is great), but instead they were stringing together really long lists of arbitrary words separated by dashes. The evolution being: **.button → .button-primary → .button-primary-dark → .button-primary-dark-container → .button-primary-dark-container-label**, ad nauseam.

. . .

The 4th Project: Identifying the Future

About this time I started writing ~a lot~ about CSS internally at Medium with the lofty goal of it becoming the best in the world. I didn't really know what that meant, but I knew our current direction just wasn't working.

> *People were confused. And what's worse, they were thinking they were writing CSS really well, when in reality they weren't.*

So, I started looking around—poking at frameworks, trying different tools, philosophies, talking to friends, talking to friends of friends, etc. I soon identified three major projects that I wanted to tackle in order to get our CSS to where I thought it needed to be.

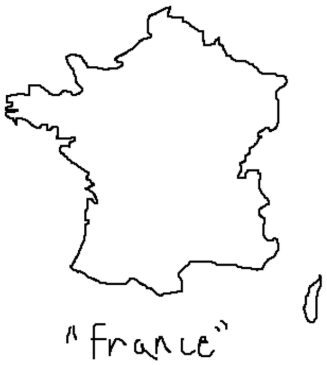
1. Introduce new CSS variable, mixin, and classname semantics to avoid run-on classnames and improve readability
2. Move us off of LESS and onto Rework for more powerful mixin support and a syntax even closer to vanilla CSS
3. Add tooling around CSS performance (load time, fps, layout time, etc)—to make it easier to track style changes and regressions over time

. . .

The 5th Project: Semantics

I wanted stricter semantics for our code base because for a team of our size I felt like it would be easier to have rules for us to lean on. And I'd rather have something a bit more complicated if that also meant people had to be more mindful when creating new CSS classes. At all costs, I wanted to avoid the run-on classname, or at least make it harder to create.

I began talking at length about it with Daryl Koopersmith and my good friend Nicolas Gallagher.



Nicolas and I have an interesting relationship in that Nicolas always tells me something, I say he's wrong, call him names in french (Va te faire foutre, enculé), and dance around for a few weeks, before inevitably realizing he's right, and taking his idea as my own.

This time was no different, and after a few late nights, I was eventually convinced of a style similar to the one Nicolas outlined in SUITCSS. Similar, but ~slightly~ better.

So I began by plagiarizing the hell out of Nicolas. I threw away our old style guide and copy pasted large portions of his, editing a few things here and there.

Eventually I came up with our current guide, which you can read in its entirety below, the main additions being:

- **.js-** prefixed class names for elements being relied upon for javascript selectors
- **.u-** prefixed class name for single purpose utility classes like .**u-underline**, .**u-capitalize**, etc.
- Introduction of meaningful hypens and camelCase—to underscore the separation between component, descendant

components, and modifiers

- **.is-** prefixed classes for stateful classes (often toggled by js) like **.is-disabled**
- New CSS variable semantics: [property]-[value]--[component]
- Mixins reduced to polyfills only and prefixed with **.m-**

LESS Coding Guidelines

Medium uses a strict subset of LESS for style generation. This subset includes variables and mixins, but nothing else (no nesting, etc.).

Medium'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

- Performance
  - Specificity

JavaScript

syntax: `js-<targetName>`

JavaScript-specific classes reduce the risk that changing the structure or theme of components will inadvertently affect any required JavaScript behaviour and complex functionality. It is not neccesarry 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

Medium'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.

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 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 behaviour.

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

Syntax: `<property>-<value>[--componentName]`

Variable names in our CSS are also strictly structured. This syntax provides strong associations between property, use, and component.

The following variable defintion is a color property, with the value grayLight, for use with the highlightMenu component.

```
@color-grayLight--highlightMenu: rgb(51, 51, 50);
```

Colors

When implementing feature styles, you should only be using color variables provided by colors.less.

When adding a color variable to colors.less, using RGB

When adding a color variable to colors.less, using RGB and RGBA color units are preferred over hex, named, HSL, or HSLA values.

Right:

```
rgb(50, 50, 50);
rgba(50, 50, 50, 0.2);
```

Wrong:

```
#FFF;
#FFFFFF;
white;
hsl(120, 100%, 50%);
hsla(120, 100%, 50%, 1);
```

z-index scale

Please use the z-index scale defined in z-index.less.

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

https://gist.github.com/fat/a47b882eb5f84293c4ed

It's one thing to copy paste a style guide, it's another to rewrite all of your CSS application.

Thankfully I was able to convince the company of the importance of this project, if only for my sanity, and was given 2½ weeks to rewrite all the CSS on Medium.com—which if you're following along, is a fraction of the amount of time it took for us to fix CSS underlines.

That said, this rewrite was not only cathartic, it cleaned up loads of dead styles, tightened up and generalized implementations across the site, broke files out into smaller subcomponents, and to my surprise, only caused a handful of rollbacks.

Of course, refactoring CSS meant also refactoring our templates—adding more encapsulation and stricter semantics across the board. Now today, rather than having hundreds of one off <IMG> tags with random avatar classes for example, we have a single centralized avatar template, which accepts boolean options to trigger different modifiers

like size, style, etc. This makes updating styles easier and implementation detail bugs much less frequent.

. . .

X Project: The future?

Undoubtedly we are in a better place then we were 2 years ago. Writing CSS at Medium is actually pretty pleasant and devs with different experience levels are contributing some pretty great stuff.

That said, the next CSS project will have to be around performance. As we continue to grow our story pages and push them to the next level, you can imagine how getting accurate, reliable measuring tools around layout and rendering performance is incredibly important… and kinda just sad that we don't already have in 2014.

So that's it. There's more to do, but I'm feeling pretty good about it all at the moment. Which again, is better than usual I think.

Cheers if you made it to the bottom of this long boring post.

Thanks so much to Katie, Dave, Mark, Koop, Kristofer, Nicolas, and others for helping me write this thing (and fix Medium) without even probably realizing it ❤

PS

(If you enjoy this?! Consider reading my take on React and Redux: