PUBLISHED BY JAMES STONE CONSULTING

VOL
01

James Stone

DESIGN SYSTEMS ENGINEERING

HOW SYSTEMS ARE CHANGING THE FACE OF WEB DESIGN

Introduction

In the past few years there has been a trend in web design. This trend is that we are moving from design deliverables such as the PhotoShop or Sketch files towards programmatic systems of design.

What this means for designers is that we must go beyond simply aesthetics.

We must start to think in terms of modules that are interchangeable, that modulate based on their environment (devices), and most importantly have a consistency and structure that will ease the product development cycle.

Furthermore, designers that function on teams, must learn and embrace new technology such as shared symbol libraries. In addition to working in a graphic tool, they must accept the tooling that allows the sharing and standardization of design assets with an engineering team. This allows assets to be easily maintained and quickly rolled out with minimum effort.

We are at the point where web projects have risen high enough in complexity to have to embrace the same processes and methodologies which have been the norm for decades in other industries. Just like the design of a car, airplane or building a miscalculation early on can cause projects budgets to quickly skyrocket in cost and deadlines to be missed. Today the same can be said for web projects.

The following is a collection of articles written by James Stone over the first half of 2017, chronicling some details of his workflow along with a rally cry for a systems based approach. This is not simply theory, but rather the culmination of his experience working on actual industry projects and helping to translate the design intent and best possible user experience to diverse engineering teams with a variety of backgrounds.

Contents

A real-world example of a coded AKA living styleguide

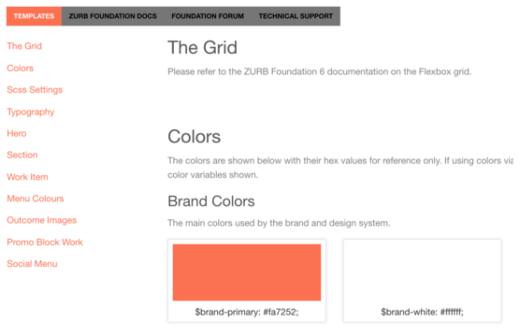
There is often focus and attention on large public styleguides such as MailChimp and Salesforce Lightning. There also seems to be many people telling you you should create one.

For smaller companies and projects there are not a lot of examples about how a styleguide should look.

Here I want to showcase a living / coded styleguide I am creating for Marcus Handa's new site. Here I used the ZURB Template and ZURB Panini to generate one.

This isn't as automated as it could be, but it is a good first step for most projects.

Here is an example of the navigation in the styleguide.



The Grid

Please refer to the ZURB Foundation 6 documentation on the Flexbox grid.

Colors

The colors are shown below with their hex values for reference only. If using colors vi
color variables shown.

Brand Colors

The main colors used by the brand and design system.

$brand-primary: #fa7252;

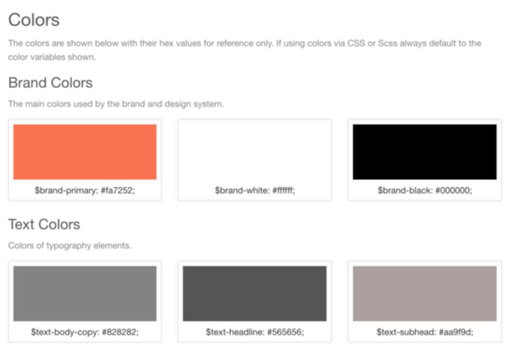
$brand-white: #ffffff;

This menu along the left side of the screen for easy access to any of the components in the project.

With Panini a heading level 1 (#) and with a section with separated by several line breaks will be added to the menu at the top left of the document.

Also, in this case I have used the ZURB Foundation framework, so I am leaning on their documentation for any standard components. This is why you see the note under the Grid section and buttons linking back to the official docs.

This styleguide shows custom components and how they are used alongside some developer documentation that might be useful should the project be handed off to another dev or team.


Colors

The colors are shown below with their hex values for reference only. If using colors via CSS or Scss always default to the color variables shown.

Brand Colors

The main colors used by the brand and design system.

$brand-primary: #fa7252;

$brand-white: #ffffff;

$brand-black: #000000;

Text Colors

Colors of typography elements.

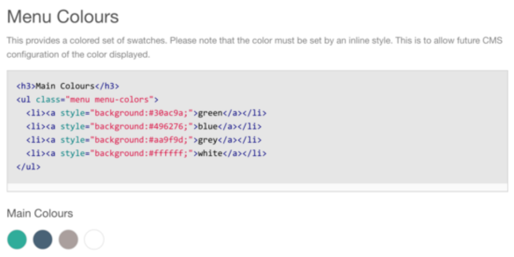
$text-body-copy: #828282;

$text-headline: #565656;

$text-subhead: #aa9f9d;

This first example displays color swatches.

Important to note is that I provide the Sass variable name (for example $brand-primary) for each color alongside its hex code. The goal is to provide the hex code as a reference but emphasize the use of the color variable.


Menu Colours

This provides a colored set of swatches. Please note that the color must be set by an inline style. This is to allow future CMS configuration of the color displayed.

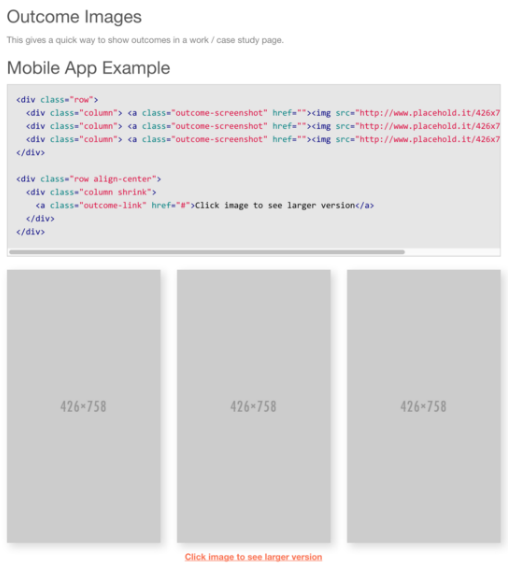
```
<h3>Main Colours</h3>
<ul class="menu menu-colors">
  <li><a style="background:#30ac9a;">green</a></li>
  <li><a style="background:#496276;">blue</a></li>
  <li><a style="background:#aa9f9d;">grey</a></li>
  <li><a style="background:#ffffff;">white</a></li>
</ul>
```

Main Colours

This example above shows a custom component that extends the existing ZURB Foundation .menu component.

This is called an example code pair, and it provides a coded example (the HTML above) along with the browser rendered output.

This browser rendered output is key because it will be automatically updated with any updates to the system. If I were to include an image here, it could become quickly outdated and would add to maintenance of the project.



Outcome Images

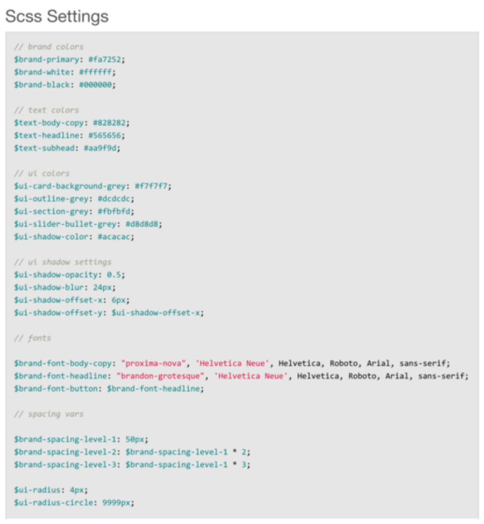
This gives a quick way to show outcomes in a work / case study page.

Mobile App Example

```
<div class="row">
  <div class="column"> <a class="outcome-screenshot" href=""><img src="http://www.placehold.it/426x7
  <div class="column"> <a class="outcome-screenshot" href=""><img src="http://www.placehold.it/426x7
  <div class="column"> <a class="outcome-screenshot" href=""><img src="http://www.placehold.it/426x7
</div>

<div class="row align-center">
  <div class="column shrink">
    <a class="outcome-link" href="#">Click image to see larger version</a>
  </div>
</div>
```

In this example I am showing a blended component. It comprises several ZURB Foundation components alongside some custom components for the images and the link below. I am only showing a mobile app example here but there is another variation available in the styleguide.

I wanted to showcase some additional engineer documentation alongside my thought process.

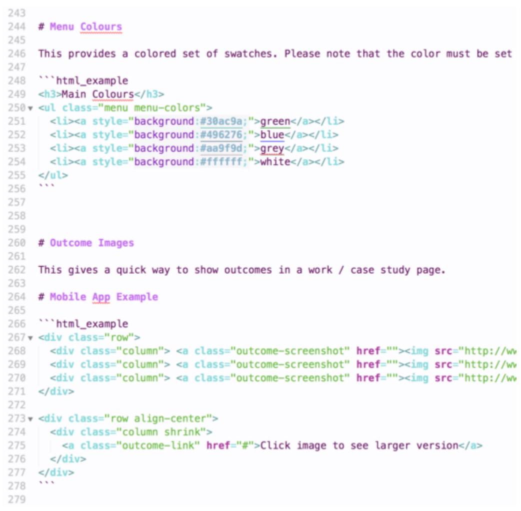


Scss Settings

```scss
// brand colors
$brand-primary: #fa7252;
$brand-white: #ffffff;
$brand-black: #000000;

// text colors
$text-body-copy: #828282;
$text-headline: #565656;
$text-subhead: #aa9f9d;

// ui colors
$ui-card-background-grey: #f7f7f7;
$ui-outline-grey: #dcdcdc;
$ui-section-grey: #fbfbfd;
$ui-slider-bullet-grey: #d8d8d8;
$ui-shadow-color: #acacac;

// ui shadow settings
$ui-shadow-opacity: 0.5;
$ui-shadow-blur: 24px;
$ui-shadow-offset-x: 6px;
$ui-shadow-offset-y: $ui-shadow-offset-x;

// fonts

$brand-font-body-copy: "proxima-nova", 'Helvetica Neue', Helvetica, Roboto, Arial, sans-serif;
$brand-font-headline: "brandon-grotesque", 'Helvetica Neue', Helvetica, Roboto, Arial, sans-serif;
$brand-font-button: $brand-font-headline;

// spacing vars

$brand-spacing-level-1: 50px;
$brand-spacing-level-2: $brand-spacing-level-1 * 2;
$brand-spacing-level-3: $brand-spacing-level-1 * 3;

$ui-radius: 4px;
$ui-radius-circle: 9999px;
```

In this example I am highlighting all the variables from the _brand-design-tokens.scss. Although this information is available in the Sass build, it is important to highlight these top level tokens. This will give a good overview of how variables are used and what could be modified with a heavy UX buildout. All without having to dig into any code.

I want to show what the example code pairs look like in ZURB Panini. Let's take a look inside the markdown source file.



~~~

Menu Colours

This provides a colored set of swatches. Please note that the color must be set

```html_example
<h3>Main Colours</h3>
<ul class="menu menu-colors">
  <li><a style="background:#30ac9a;">green</a></li>
  <li><a style="background:#496276;">blue</a></li>
  <li><a style="background:#aa9f9d;">grey</a></li>
  <li><a style="background:#ffffff;">white</a></li>
</ul>
```


Outcome Images

This gives a quick way to show outcomes in a work / case study page.

Mobile App Example

```html_example
<div class="row">
  <div class="column"> <a class="outcome-screenshot" href=""><img src="http://ww
  <div class="column"> <a class="outcome-screenshot" href=""><img src="http://ww
  <div class="column"> <a class="outcome-screenshot" href=""><img src="http://ww
</div>

<div class="row align-center">
  <div class="column shrink">
    <a class="outcome-link" href="#">Click image to see larger version</a>
  </div>
</div>
```

~~~

When you use Panini, you can use a special GitHub style code fence (“‘) annotated with html_example. This creates an element to separate and syntax color your source code, but also displays an in document rendered version right below it.

This saves a ton of time because you only have one set of html to deal with and the example will match the source.

How many times have you copied an example online only to find an error. No fun. You want to make sure your documentation is air tight.

Creating the styleguide is a simple process if you have kept your project organized. You go through your templates and look for patterns with custom components. Then copy and paste the examples.

It is also possible to make other examples from scratch, or document patterns that didn't make it into your templates.

The goal here should be to make sure your engineering team has everything they need. Think about what they need to use or understand the components that make up the system. Since we are resting on the ZURB Foundation documentation here, there is no need to reinvent the wheel. A simple link back to the official docs is all that is needed. Adding a few tips (in this case colors and Sass variables) can help point your team in the right direction.

This is a pretty small project but the gains by creating a proper set of front-end code and a coded or living styleguide are immense.

Just imagine that none of this existed and Marcus' hired a new team to build out an UX heavy change that used a lot of the same patterns.

He would only have access to the final site which was built into a CMS. Your new team would have to spend a bunch of time to figure out how the system worked. Sometimes, teams might just copy and paste the output of the Sass files into a static CSS file. You might end up with a bunch of HTML with CMS garbage thrown in and a standard CSS file that would be a burden to make sweeping changes in.

This is a very real proposition because I have inherited many projects over the years just like this or worse.

Expanding on the ideas above the benefits of building a proper set of front-end code are:

- UX heavy changes can be handled within the design system and make changes without effecting the live site.
- Separation of the front-end code from the back end implementation. This makes it faster to make changes.
- A sleek Sass build pipeline that allows you to make large sweeping changes quickly.
- The ability to bolt on tools like auto-prefixer (you don't have to have CSS prefixes AKA duplicate code) and uncss (a last resort performance add on).
- Gulp (or otherwise) will allow you to change your Sass files and re-compile (and live browser load) your changes on the fly.
- Everything can be checked into a git repo so it is easy to create feature branches or refer to older versions of the system.
- Once you make your UX heavy changes, you have a clear set of example HTML for your team to refer to.
- You can show progress to stakeholders much faster by just chang-

ing the front-end code. It will also be easier to forecast the time to implement the changes on the back end since you can compare the differences.

And the benefits of creating a living or coded styleguide include:

- A clear path for documentation related to the project. If your team needs to add something that isn't covered, this will point them in the right direction.
- Examples for how components should look. Think about alerts, notifications, and other elements. On the live site it might not be so easy to trigger these. They also might not be included in the front-end code if you are limiting the number of templates you build out (and I would recommend you limit them). The coded styleguide is a perfect place to showcase specialized components and their options. You can even make changes and check the styleguide for visual updates since the share the same Sass build pipeline.
- Specialized engineer documentation. Since this will have a single home everyone on your team will get used to looking for it here.

The advantages for skipping these steps: None. Nada. Nothing whatsoever.

People will claim you will save time by skipping these steps. I have never-not once-seen this to be the case.

What I have seen is projects bursting with bugs, overrun budgets, beaucoup inconsistencies, and deadlines missed that just make you want to cry.

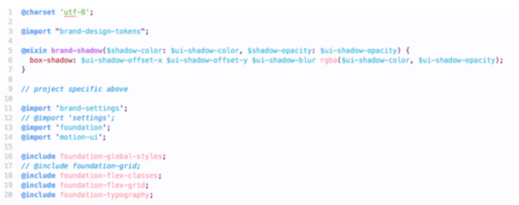
This is the realm of the design systems engineer. To save you and your team from making these mistakes and get your project(s) back on track.

A sneak peak into one of my project Sass files

Let's take the look under the hood at my main scss file and show you how I am structuring things in a typical web project. This example is from Marcus' new site, but the approaches I show here I use universally.

First lets take a look at the top of my ~/scss/app.scss file. This is the default name for a ZURB Foundation project, but it could literally have any name.

What is more important are the contents of this file.



```scss
@charset 'utf-8';

@import "brand-design-tokens";

@mixin brand-shadow($shadow-color: $ui-shadow-color, $shadow-opacity: $ui-shadow-opacity) {
  box-shadow: $ui-shadow-offset-x $ui-shadow-offset-y $ui-shadow-blur rgba($ui-shadow-color, $ui-shadow-opacity);
}

// project specific above

@import 'brand-settings';
// @import 'settings';
@import 'foundation';
@import 'motion-ui';

@include foundation-global-styles;
// @include foundation-grid;
@include foundation-flex-classes;
@include foundation-flex-grid;
@include foundation-typography;
```

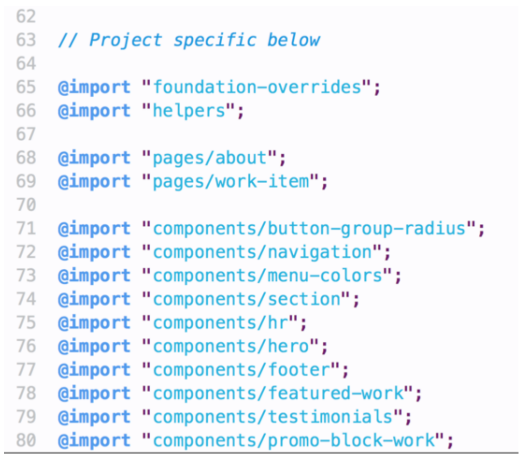
You will notice the first thing I do is import a file called _brand-design-tokens.scss. I try to use names that are descriptive and literal. Here I place all of my tokens or variables for the project.

Below I have a small mixin called brand-shadow. You notice that it has a set of defaults that come from the brand-design-tokens file. If this was a much larger project, this mixin would have found itself into its own file as part of the build sequence.

Next up you will see the import of the _brand-setting.scss file. This is a copy of the built in ZURB Foundation _settings.scss file I change to take variables from my _brand-design-tokens file.

After this, if you are familiar with ZURB Foundation, you can see that I switch from the float based grid system to the new flexbox based one. Then I have a blanket import of the foundation components that I am using in the project.

Now I want to skip down the file to show you the project specific imports.

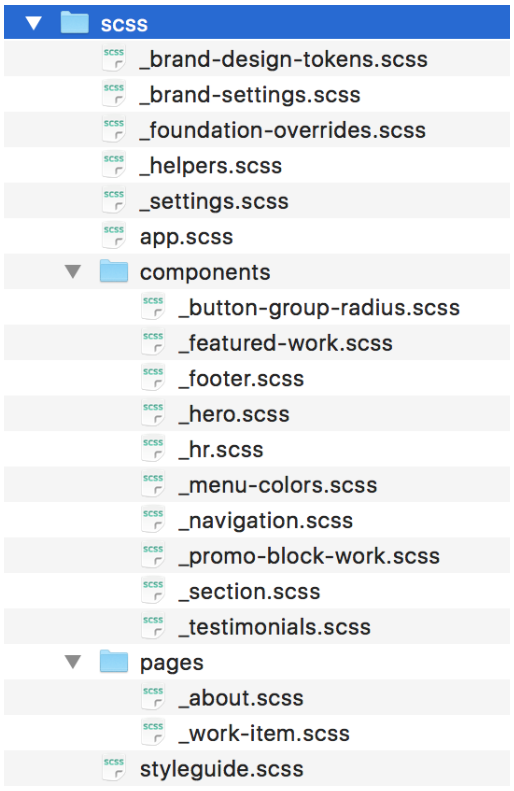

```

// Project specific below

@import "foundation-overrides";
@import "helpers";

@import "pages/about";
@import "pages/work-item";

@import "components/button-group-radius";
@import "components/navigation";
@import "components/menu-colors";
@import "components/section";
@import "components/hr";
@import "components/hero";
@import "components/footer";
@import "components/featured-work";
@import "components/testimonials";
@import "components/promo-block-work";
```

Here I am using line spaces to create logical groups. Foundation Overrides and Helpers are exactly what they sound like. Places I override the base framework with additional CSS and also small classes that don't fit elsewhere.

Next up I have my page specific imports. These are pages that have specific components or spacing patterns that are not shared.

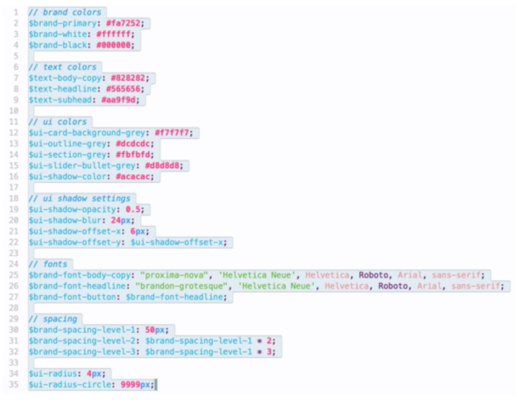
Below I have my general components. These could be used anywhere in the project but they are named in such a way to correspond with the CSS class names and the documentation in the living / coded styleguide.

Another way to look at this information is by taking a look at the actual file structure that also matches the imports above.



```
▼ scss
    _brand-design-tokens.scss
    _brand-settings.scss
    _foundation-overrides.scss
    _helpers.scss
    _settings.scss
    app.scss
  ▼ components
      _button-group-radius.scss
      _featured-work.scss
      _footer.scss
      _hero.scss
      _hr.scss
      _menu-colors.scss
      _navigation.scss
      _promo-block-work.scss
      _section.scss
      _testimonials.scss
  ▼ pages
      _about.scss
      _work-item.scss
    styleguide.scss
```

Now lets take a deeper look at my _brand-design-tokens.scss file. These are not formally design tokens, but have been arranged so they could be integrated with a 3rd party design token system such as Brand.ai or Salesforce Theo. The name indicates that future role.



```scss
// brand colors
$brand-primary: #fa7252;
$brand-white: #ffffff;
$brand-black: #000000;

// text colors
$text-body-copy: #828282;
$text-headline: #565656;
$text-subhead: #aa9f9d;

// ui colors
$ui-card-background-grey: #f7f7f7;
$ui-outline-grey: #dcdcdc;
$ui-section-grey: #fbfbfd;
$ui-slider-bullet-grey: #d8d8d8;
$ui-shadow-color: #acacac;

// ui shadow settings
$ui-shadow-opacity: 0.5;
$ui-shadow-blur: 24px;
$ui-shadow-offset-x: 6px;
$ui-shadow-offset-y: $ui-shadow-offset-x;

// fonts
$brand-font-body-copy: "proxima-nova", 'Helvetica Neue', Helvetica, Roboto, Arial, sans-serif;
$brand-font-headline: "brandon-grotesque", 'Helvetica Neue', Helvetica, Roboto, Arial, sans-serif;
$brand-font-button: $brand-font-headline;

// spacing
$brand-spacing-level-1: 50px;
$brand-spacing-level-2: $brand-spacing-level-1 * 2;
$brand-spacing-level-3: $brand-spacing-level-1 * 3;

$ui-radius: 4px;
$ui-radius-circle: 9999px;
```

Here you see that I use comments and white space to break this file up into logical sections.

1. Colors: Brand, Text, and UI
2. Shadow Settings
3. Fonts
4. Spacing

You can see I put a lot of thought into how to organize this project. I also spent time to make sure I stayed consistent throughout.

Your styleguide stinks

The internet is littered with ton’s of people talking about styleguides. And I say, who cares! It seems like one floats across designer or hacker news on an almost weekly basis. The sad news, I hate to break it to you, is that most people don’t care about your styleguide. Not at all. Not even for a second.

Here’s why:

- You gotta get beyond the colors. No one cares.
- Don’t just post some icons. Don’t just say something obvious like, these icons should be black.
- You gotta get beyond posting fonts. Fonts! Who cares.
- Go beyond the logo and how it is used. No one cares about your logo.

And maybe that last one stings a bit.

Now this might sound harsh and I get it. You wear your heart on your sleeve sometimes when you put your design out in the world.

This is the real world of fast paced web development.

To just create a styleguide that resembles print brand guidelines (similar to the above example) is a waste of time. That is because unlike print, where you can change everything willy-nilly, for the web you want a more regimented and consistent output. Also engineering time is crazy expensive.

First off, the logo isn't going to be dropped all over the place.

It might only end up in one place, on the top left of your app. And maybe there is a variation for a mobile format. That would be way more useful than describing every situation in which the logo could be used. I would say for me on average, I spend maybe 10 minutes total dealing with the logo. Which is why you won't find it in the styleguides that I create.

And trust me, I love, love, looove me a good logo.

What would make a styleguide more useful?

In short, understanding the problem better.

As designers we are master problem solvers. And engineers are master problem solvers as well. But we approach the problem from different angles. We have different processes. We also arrive at a different set of deliverables.

In design, it is the design document. In engineering, it is the production site.

The part in the middle. That is where things get interesting…

How do you redesign a site like Capital One with 2500 pages?

In the book Responsive Design: Patterns & Principals by Ethan Marcotte and there are two passages in particular that stand out.

> "In describing how they rebuilt Capital One's layout in four weeks, developer Scott Childs argued that **a focus on components, not pages, was key.**" (Kindle Loc 182)

It continues with a direct quote from Scott Childs...

> "Even though it's this massive number of pages, when **you look at 2,500 pages over 4,000 page configurations, everything boils down to a couple things really**. How many different components do we have total? It's around twenty components" (Kindle Loc 182)

2,500 pages with 4,000 page configurations; four weeks. That's no small job!

I have spoken before about how the face of web design + development has changed over the years on the Experts & Influencers Podcast Episode 15 around mark 4:58.

In the podcast I said,

> "**[Today web] projects are much more complex**… The downside of that is now applications are so much more complex and so much larger than they used to be. [Now] you are back to spending 2-weeks with a room full of engineers to accomplish much more, it's just **you have a much higher expectation for what you have to accomplish**."

So today's teams have a huge amount of responsibility so we have to work smarter.

Capital One did that by being smart about their process and using a component (rather than a page) driven method for their redesign.

While smaller sites and apps might give you the luxury of building out every template.

Larger systems require a different approach.

This is where the coded or living styleguide comes in. It provides a home for these components to be used in the redesign. It provides context and allows you to see what they look like under the hood. Most important, it gives your engineers just enough documentation they can use every component interchangeably.

Using a styleguide allows you to work smarter and not harder.

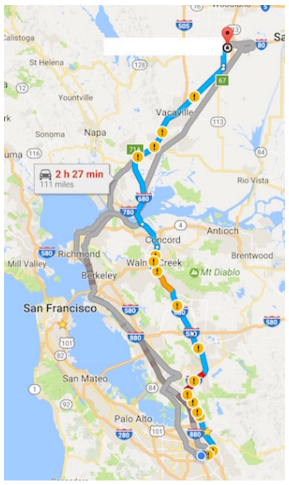
If you have done a site or app audit and can boil everything down to a handful of components that in itself doesn't necessarily make the job easier or faster. It is all about how you name, structure, and document these components so they can be used efficiently and in parallel across your team.

What does traffic have to do with design systems?

So there I was, huddled up in a cafe in Fremont refreshing Google Maps about every 5-10 minutes. The traffic in the Bay Area is just terrible. Sometimes I feel like I might arrive faster if I wait for the traffic to die down.

In fact, this day the traffic is what I would call moderate.

You will notice below how the 680 freeway is riddled with accidents. These are denoted with an (!) which seems appropriate.



I am no expert, but I have spent several years traveling around this terrible Bay Area traffic.

I have found the following things to be true:

1. **A freeway has a specific maximum capacity.** It can provide flow at maximum speed until that capacity is exceeded by just one car.
2. If there is an accident or emergency vehicle, **every car will slow down to look at it then speed away** (at least in California).
3. **The HOV or Carpool lanes are useless** when they have to merge back into the other traffic.
4. **It only takes one mistake to disable an entire freeway** when it is operating at or near capacity. Gridlock.

Now this is just my observation having driven the same routes literally hundreds of times in similar conditions and similar times. Perhaps you have come to some conclusions of your own.

This got me thinking to how having a proper set of design systems can help streamline your project.

Every time an engineer has to question what needs to be done (from a visual standpoint) it is just like one of these accidents.

Even thought it might not be a blocking task-meaning the engineer can not proceed with any other work-there still is the tendency to slow down and look.

This often means asking questions, talking to the design department, talking to other engineers, googling things, jumping on stack overflow. This is the equivalent of slowing down and taking a look.

Even though a quick solution is certainly possible, no one can resist slowing down for a gander.

Having a great set of documentation (a proper set of front-end code and a coded / living styleguide) is like having a magic wand.

Imagine the next time you are stuck in traffic, just to pass a minor fender bender on the side of the road, and speed up to normal speed again that you had this magic wand.

Poof.

Just with one wave of the wand you could prevent or remove the accident from the side of the road. There would be no reason for anyone to slow down. There would be no delays. Your trip would continue in the more efficient way possible.

Design Systems Engineering is your magic wand for fast-paced web projects.

If approached correctly, it gives you the super power to get slowdowns, meetings, chats, and googling down to a minimum. It allows you do to your best work possible by being able to work at near capacity.

Creating a minimal feature demo to ensure stakeholder buy-in

SnowflakeStories.com is very complex under the hood. There is a lot of complexity in the product. In order to mask a lot of that complexity, a lot of business logic has to be written in order to handle different conditions and edge cases. The books are automatically translated into several languages, which requires specific roles and other data to be gathered by the end user.

In short, it is a lot of work to keep things simple for the end user. The result is the process seems easy. That is no accident, there is a lot of effort put into making it seem simple.

The downside of that is that the project is a bit complex to maintain.

When adding a new feature it is critical that I get stakeholder buy in up and to be very clear about how things will work. Before spending the time to implement this across 3 different products (with subtle variations) and 4 different character types (all including subtle variations), I just created an MVP form of the feature.

I created this on a separate git branch and I added the feature for the easiest to show character. Once I was done, it was important to showcase how this works.

Sure I could have thrown this over the wall to the founder, with some text explaining how it works. But that would not have been the best experience. Furthermore, it would have left a lot of things up for misinterpretation.

Creating a short screencast video allowed me to clearly isolate and communicate the feature. This also served as a basis for future communication about the feature.

You can watch out the video here: http://dropshare.jamesstone.com/NameUX.mp4

You will note that it is cropped tightly around the feature, it has no sound recorded, and it goes through a simple sequence to show how the feature works.

When you are working on a fast paced web product, it is critical to communicate clearly with all members of your team. It is kind of like the snowball effect. Every corner you cut, every word you leave that is vague, every hour you waste using the wrong tool can add up fast.

Woes of the CMS: the open source theme or starter project

The final reason that a CMS buildout is challenging is that often starter themes and starter projects were not created with your project in mind. Often these themes are “labors of love” started by a lone designer or developer with a passion for CMS + something else (like Bootstrap) and they just went ahead and put it together.

Most of the time they might have a lot of knowledge on one area, but they are often not an expert at both.

This is because rather than use a tried and true solution, they took their latest interest and tried to integrate it with their long time passion. This usually results in poorly documented, over engineered (on one end), and extremely opinionated examples.

In some cases projects are created for agency use, or released by a commercial core team. Although on the surface this might sound like great news as apposed to the hobbyist, but these are also not without their faults.

The issue is that most agencies or commercially driven open source projects usually have some sort of hidden (or not so hidden agenda). That is great if it aligns with yours, but in this day and age of the pivot, this can change in an instant.

The solution to this problem is quite simple.

It doesn't matter if you have an agency that does a buildout across 5 different platforms (which is crazy) or you are a WooCommerce specialist cranking out the next greatest e-commerce WordPress sites. The solution is to create your front-end code and CSS (Sass) in isolation, to make short work the issues presented with a CMS buildout.

What livin’ in the desert taught me

Last year, I remember being at first excited and then a little bit terrified when my wife said she had been offered an internship at NASA AFRC.

I was excited because it was NASA. Who wouldn’t be excited about that.

But then I realized we were going to be living near a military base out in the middle of the Mojave Desert. The fear started to settle in...

If you have ever been sick and googled your symptoms, I am pretty sure you came across a bunch of pages that told you that you have something terrible like cancer.

In a similar way, you will hear horror stories and terrible things that happen out in the desert thanks to the negative nature of the Google search.

It certainly helps to be prepared, but to be honest it wasn't as bad as I thought.

First off, this place is like ground zero for aerospace. It is like the who's who of much of the important things that have happened. Like breaking the sound barrier. <– pretty important stuff.

So it shouldn't come as a surprise that there are a lot of crazy smart things you can learn, just by getting access to such a strange isolated place full of smart and talented people.

Much of this might sound obvious once you know about it, but if you haven't lived in the desert, you might not think about things in a peculiar type of way.

Take this first example: park your car facing away from the sun.

I have lived in a pretty hot place, Sacramento where it often climbs near or over 110 degrees in the summer. And I have one of silver reflecting things that you put on the windshield.

This isn't just to keep the temperature down in the car, but it will get so hot on those days that you won't be able to touch the steering wheel without some sort of protection. It will literally burn your hands.

But in the desert, you have a special kind of hot and a crazy kind of sun. It is really like nothing else. And you are out on the playa (a dry lake that is similar in smoothness to glass), so it is like being on the beach. Somewhat like having 2 or 4 or maybe even 8 suns.

IT IS HOT! No joke.

And so one day my wife explained to me when we were on the base, you just park away from the sun. And really, everyone does it.

It's so simple.

You have the largest piece of glass on the front of the car (the reason for the reflector) which is the biggest way a car heats up.

If you just point that sucker away from the sun, which coincidentally puts that gigantic piece of glass in a shadow. This in turn drastically reduces the heat that accumulates in the car.

Its amazing and super simple.

Now when I am in Sacramento - I often don’t even use that reflector. Its so much easier to pay more attention to how I park the car.

You just take a look around, look at which way the shadows are pointing, and then angle your car appropriately. It doesn’t have to be exact, just an approximation.

No more fumbling for that crazy reflector, which has a habit of smacking everything and everybody as you put it up.

This is such a simple and elegant solution, but I had never thought of it.

And trust me, I do not like a hot car in the summer.

Just like this simple trick, sometimes you just need to be introduced to the right idea at just the right time. Sure you could wait like me your whole life to have that aha moment, but there is a better way.

This is where having a guide comes in. They are someone that can just show you the way. You no longer have the pressure of having to invent it all yourself and this can make a huge difference.

What is it like to win an award?

I have spoken before about my process behind building the award-winning site SnowflakeStories.com. Today I want to talk about what it is like to win an award. And also what that process is like.

1. Having something that is worth entering

2. Submit your entry ahead of the deadline

3. Wait for quite a long time. **Oh and you pretty much hear nothing back. And for a very long time.**

For the duration of our UI heavy buildout (AKA the redesign) we knew of the award timelines. We had hoped to apply to many of them. Like many projects with an aggressive deadline, it got a little hectic towards the end as we tried to have every little detail nailed down.

Remembering back we applied and then you pretty much hear nothing…

It is nerve-wracking.

You get a confirmation email, yes you have applied. But then you hear nothing.

Even worse, often the award organizations send you strange little emails. Each time you see one light up your mailbox you open it instantly to see if it is the results.

So it is this rollercoaster ride of highs and lows. And lots of waiting.

I still remember the email I got from Jill on April 8th, 2016 telling me that we had won.

So what does this feel like? It feels great.

In an instant you go from x to award-winning x. You feel awesome about that project and it provides a lot of validation for all of your hard work.

Why offshore teams need Design Systems Engineering

Recently I worked on a project where most of the dev team was housed offshore. That in itself is not a problem, I love my offshore buddies, but it presents an interesting set of problems. Problems that can be largely solved by using Design Systems Engineering.

One of the biggest issues is the time difference.

Even if you are just separated by a few time zones, it makes meetings just that much more difficult.

Additionally, East Coaster at the end of their day might feel and operate a bit differently that us West Coasters who just hit up our local favorite coffee shop.

One might be thinking about how they are going to spend time with their family, the other might be fired up about getting some serious work done.

Now when you go across many time zones, even if you match your work hours, you will still run into issues.

So meetings are tough; you have to make the most of them.

One important way to do that is to make sure everyone is working from the same playing field.

One of the best things you can create is **a maintained set of front-end code**. At a minimum you should have **at least 5 major templates**.

Additionally, your team will run into issues where they need an extra modal, or an alert, or a badge. **This is where a living (or coded) styleguide can come in really handy**.

Even if you are extending a framework such as ZURB Foundation or Bootstrap, it is important to display common elements in their final coded form.

Ideally you want the trifecta of truth:

1. A coded example
2. What that example looks like rendered (bonus points for it being html / css based)
3. Some basic documentation about what it is, what it does, and how to use it (extra classes, options, etc.)

Now, imagine having these tools in place.

No longer will you have to answer questions like, how do I code that button. Or, where is that icon for the alert. Or, what font should I use.

Focus on what your team does best. Build on your individual strengths.

In American Football (which can be quite confusing to the uninitiated) you have specialized roles.

You have the quarterback. He pretty much throws the ball based on the play. He is awesome at throwing that ball. He can throw it faster and more accurately that anyone else.

But what about field goals (kicking the football through the goal), punts (just kicking the ball down the field) or kickoffs (pretty much the same as a punt but from a defined spot)?

Sure, you could put your quarterback there. **Why not? He can kick.**

But the problem is that he can't kick as well as the kicker.

They become a better team and win more games by focusing their directed roles through a common strategy.

This is what Design Systems Engineering can do for your team.

It helps you to simply do better work.

About James Stone

James Stone is a Design Systems Engineer and a top contributor to Open Source ZURB Foundation CSS front-end framework. He is most well known for his videos on YouTube which have had over 294k views to date. He has written for UX Pin and ZURB University and is an Adjunct Professor at Penn State's School of Visual Art, where he teaches an online introduction to scripting course targeted at artists and designers.

If you have any questions about the book you can reach James via email at james@jamesstone.com

JamesStone.com