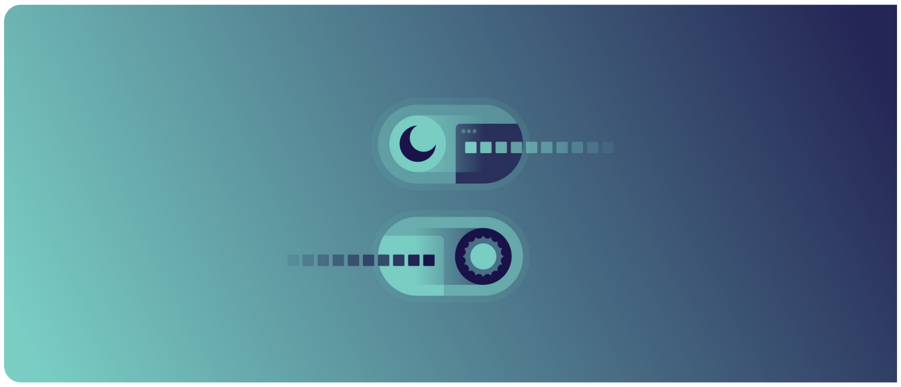
Dynamic Theming with CSS Variables



David Leger

Sep 5, 2019

Share

A common trend on the web these days is dark theming. Twitter, YouTube, Facebook, and many more all have a dark theme option now. It seems every website is adding this capability. In this article, I explore how to create a simple theming system with CSS variables. Let's get started!

CSS variables

CSS variables (a.k.a. custom properties) are now supported in most browsers. Nearly 92% of people worldwide use a browser that supports them, so it's about time we finally start using them with confidence.

CSS variables have been a long-awaited feature of the web platform. Preprocessors like LESS and SASS have been showing us, for years now, the benefits of having variables in CSS. However, these preprocessors are limited in what they can do with variables. These variables are replaced with a static value and therefore can't be changed during runtime. This often leads to a lot of code duplication when run through a preprocessor, because each permutation must be generated at build time.

As it relates to theming, this means we need to create an entirely new stylesheet for each different theme. This also means that themes aren't customizable without introducing a lot of complex JavaScript to update CSS values on-the-fly.

CSS variables allow us to create dynamic stylesheets without relying heavily on JavaScript. This makes theming a lot easier to implement and keeps much of the styling in CSS, where it should be.

Infinite shades of gray

The toughest part of theming is creating a grayscale that works well regardless of the background color. The grayscale is a subtle piece of the UI but, it's an important piece to get right when creating a flexible theming system.

Not all grayscales can be easily adapted for dark themes. Let's discuss the limitations of this grayscale:

```css
:root {
  --offWhite: #f8f8f8;
  --lightest-gray: #f3f3f3;
  --lighter-gray: #eee;
  --light-gray: #ddd;
  --gray: #888;
  --dark-gray: #444;
  --darker-gray: #222;
  --darkest-gray: #111;
  --black: #000;
}
```

To create a dark theme, all these values need to be inverted, but lightest-gray, for example, needs to be one of the darkest colors. So the whole naming convention breaks down with each color being inaccurately named. This naming convention also makes it tough to insert a new gray values between any two of these colors, because there won't be a good name for it. What do you name a color that's between light-gray and lighter-gray?

A better approach is to use RGBA values, name the variables based on their alpha value, and use the term grayscale as a neutral name:

```
:root {
  --grayscale-03: rgba(0, 0, 0, 0.03);
  --grayscale-05: rgba(0, 0, 0, 0.05);
  --grayscale-10: rgba(0, 0, 0, 0.1);
  --grayscale-20: rgba(0, 0, 0, 0.2);
  --grayscale-50: rgba(0, 0, 0, 0.5);
  --grayscale-80: rgba(0, 0, 0, 0.8);
  --grayscale-90: rgba(0, 0, 0, 0.9);
  --grayscale-90: rgba(0, 0, 0, 0.95);
  --grayscale-100: rgba(0, 0, 0, 1);
}
```

This allows new intermediate values to be added to the grayscale as needed, and the names still make sense when white is used as a base.

Now that we're using alpha values to determine shade, we can clean this up a bit by pulling out the base color (0, 0, 0) into a variable:

```
:root {
  --grayscale-base: 0, 0, 0; /* black */

  --grayscale-03: rgba(var(--grayscale-base), 0.03); /* most transparent */
  --grayscale-05: rgba(var(--grayscale-base), 0.05);
  --grayscale-10: rgba(var(--grayscale-base), 0.1);
  --grayscale-20: rgba(var(--grayscale-base), 0.2);
  --grayscale-50: rgba(var(--grayscale-base), 0.5);
  --grayscale-80: rgba(var(--grayscale-base), 0.8);
  --grayscale-90: rgba(var(--grayscale-base), 0.9);
  --grayscale-95: rgba(var(--grayscale-base), 0.95);
  --grayscale-100: rgba(var(--grayscale-base), 1); /* most opaque */
}
```

Now we just need to change the base color to change the entire theme.

Inverting the grayscale

We've solved the naming convention problem and made the grayscale easy to modify. However, we've introduced a new problem now that we're using alpha values to determine the shade. There are no white values!

To solve this problem, we need an inverted grayscale. This looks almost identical to the regular grayscale, but with a different base color and the addition of an i (*inverted*) at the end of the variable names:

```css
:root {
  /* grayscale */
  --grayscale-03: rgba(var(--grayscale-base), 0.03); /* most transparent */
  --grayscale-05: rgba(var(--grayscale-base), 0.05);
  --grayscale-10: rgba(var(--grayscale-base), 0.1);
  --grayscale-20: rgba(var(--grayscale-base), 0.2);
  --grayscale-50: rgba(var(--grayscale-base), 0.5);
  --grayscale-80: rgba(var(--grayscale-base), 0.8);
  --grayscale-90: rgba(var(--grayscale-base), 0.9);
  --grayscale-95: rgba(var(--grayscale-base), 0.95);
  --grayscale-100: rgba(var(--grayscale-base), 1); /* most opaque */

  /* inverted grayscale */
  --grayscale-03i: rgba(var(--grayscale-base-inverted), 0.03); /* transparent */
  --grayscale-05i: rgba(var(--grayscale-base-inverted), 0.05);
  --grayscale-10i: rgba(var(--grayscale-base-inverted), 0.1);
  --grayscale-20i: rgba(var(--grayscale-base-inverted), 0.2);
  --grayscale-50i: rgba(var(--grayscale-base-inverted), 0.5);
  --grayscale-80i: rgba(var(--grayscale-base-inverted), 0.8);
  --grayscale-90i: rgba(var(--grayscale-base-inverted), 0.9);
  --grayscale-95i: rgba(var(--grayscale-base-inverted), 0.95);
  --grayscale-100i: rgba(var(--grayscale-base-inverted), 1); /* opaque */


  /* grayscale bases */
  --grayscale-base: 0, 0, 0;
  --grayscale-base-inverted: 255, 255, 255;
}
```

We've also moved the grayscale base colors to the bottom, which doesn't affect their behavior. These base values don't need to be defined before they're used, which is an important detail to keep in mind for when we want to override the default theme values later on.

Using the theme

Now that we've defined our theme, let's use it!

Here's how the theme can be applied to a button class:

```css
.button {
  background: var(--grayscale-100); /* black */
  color: var(--grayscale-100i); /* white */
}
```

Any elements that use these theme values will automatically update as the base colors change.

Creating a dark theme

Now that the hard part is done, we can simply redefine the base colors with their values reversed to create a dark theme:

```
:root {
  --grayscale-base: 255, 255, 255; /* white */
  --grayscale-base-inverted: 0, 0, 0; /* black */
}
```

You can also experiment with base values other than pure black and white to find a unique color scheme for your website or app.

Going further with theming

So far, we've focused on grayscale because it's one of the more complex problems in theming. CSS variables can handle any type of property though. Here's an example of a few more properties we can add to our theming system to make it even more customizable:

```
:root {
  /* Grayscale bases */
  --grayscale-base: 255, 255, 255;
  --grayscale-base-inverted: 0, 0, 30;

  /* Contextual colors */
  --color-primary: darkorange;
  --color-success: lime;
  --color-warn: gold;
  --color-error: red;

  /* buttons */
  --button-radius: 4px;
  --button-border: 1px solid var(--grayscale-20);
}
```

These values can be used to theme any element.

The main challenge with theming is balancing complexity with customizability. Exposing too many properties will make a theme difficult to maintain, while too few will make it rigid and defeat the purpose of theming.

Try it yourself!

It's hard to fully grasp the power of theming with CSS variables without experiencing it firsthand, so I've set up a demo project that implements everything discussed in this article.

Check out the demo app on CodePen.

Theming at Manifold

Manifold's Marketplace-as-a-Service uses the same grayscale schema discussed in this article.

Our embeddable web components integrate with a variety of developer platforms and use theming to accommodate the unique design aesthetic of each platform.