Tailwind CSS - 5 reasons I choose Tailwind over CSS - Why I consistently choose Tailwind for new projects

Written by Seth Corker on Benevolent Bytes

Good defaults

Tailwind strikes a good balance between by giving you a good set of defaults. It's not too much that you're overwhelmed, and it never feels limiting. By having consistent naming, it becomes second nature as you remember the units, colours, and modifiers.

Defaults Tailwind offers

Tailwind has good default colours, spacing, breakpoints and other useful tokens already. This provides a nice base to build upon. It's the foundation of a design system that's been well-thought-out.

Colours In the documentation, they say: > Tailwind includes an expertly-crafted default color palette out-of-the-box that is a great starting point if you don't have your own specific branding in mind. This is just it. It's a great starting point when prototyping, if you're building with a brand in mind, then you can customise the colours to match. What you get for free is a good range of colours across the spectrum at predefined steps.

Colours are defined with a name and a value, so `green-700` is `#15803d`. Now you know the name of the token, you can combine it with prefixes to apply the colour in various places, like `bg-green-700` for a green background.

Read more on the Tailwind docs about colours.

Spacing Spacing is fundamental in a design system. It ties everything together and is one of the most important things for consistency. There are many times I've used utilities in organisations to use consistent spacing during development. These are a great start, but they only get as far as defining the base units and making a nice helper to multiply a number by the base units. Tailwind goes one step further and defines a scale. 1 unit is 4px but in practice, you have fine grain control for smaller units—from 0 to 12. Beyond that, the steps become a little larger, so the last few units are 72, 80 and 96.

Take a look at the delightful visual representation of the spacing scale on the Tailwind docs.

Breakpoints Whenever I used to start a project when using plain CSS, there would be two paths I'd go down. Either, copying a set of breakpoints from another project or elsewhere on the web or, patching media queries together as needed. Breakpoints are an important part of responsive design, and tailwind covers the majority of use cases without any additional customisation.

The great thing is that Tailwind gives you modifiers, which makes working with media queries much easier when thinking about responsive design at a component level.

Check out a small example here:

```
<div class="grid gap-1 w-52 grid-cols-1 sm:grid-cols-3">
  <div class="bg-red-300 h-16 w-16"></div>
  <div class="bg-green-300 h-16 w-16"></div>
  <div class="bg-blue-300 h-16 w-16"></div>
</div>
```

I arranged three squares in a column for small viewports, and anything above 640px wide uses 3 columns. It's easy to see what modifications happen based on viewport size or even other media queries like for print CSS or dark mode.

Take a look at how the breakpoints are defined on the Tailwind docs.

How does this compare with CSS?

CSS has the ultimate flexibility, but you need to build everything up or find a framework to give you a starting point. If you're starting from scratch, you have to create the world. Define good rules for spacing, define your colour palette and figure out breakpoints. It's not difficult, but it does require a lot of thought and preparation before you can become productive. When I want to move fast on a web project that spans more than a single page, I often reach for a framework that someone's already put time and effort into perfecting. For anything

smaller or for projects that have a well-defined style guide or design system already, then CSS is often the best primitive to use.

Easy dark mode, print CSS and responsive design

Variants are a smart way to approach dark mode, responsive design and print styling in Tailwind. In plain CSS, you use media queries directly, but it can be easy to miss different viewport sizes because the base style and the media queries are in different places. Tailwind makes this more intuitive because your base styles and variations are colocated. I find that when I'm rapidly prototyping a UI, it's easier to build up the layers of responsiveness because everything is right there.

Adding a dark mode variant with Tailwind

To use a modifier, just prefix a class you want to apply when a particular condition is true. So, to support light and dark mode we could use the `dark` modifier, here's an interactive playground too:

```
<div class="bg-white dark:bg-slate-800 p-10 m-8 border rounded-md">
  <h1 class="text-xl text-blue-600 dark:text-blue-100">Hola</h1>
</div>
```

If the user's browser prefers a dark colour scheme, then the following class will be applied to the container `bg-slate-800` and the text will receive `text-blue-100` because we used the classes `dark:bg-slate-800` and `dark:text-blue-100` respectively.

This same principle applies for `print` and responsive modifiers too!

How to use responsive modifiers

Tailwind's modifiers are easy to use, just add the modifier and a colon as a prefix to the class you want to apply. There are a few modifiers to choose from depending on which viewport you need to target like `sm`, `md`, `lg`, `xl`, `2xl`.

Here's an example of changing the size of a rectangle based on the viewport width,there's a sandbox link as well:

```
<div class="bg-indigo-400 w-16 h-16 rounded-xl m-4 md:w-28 md:h-28 lg:w-48 lg:h-48"></div>
```

Good plugins like typography and forms

What does Tailwind's Typography plugin do?

The typography plugin styles your standard HTML tags like headings and paragraphs nicely, including font sizing and spacing. At first glance, this seems to go against everything Tailwind is trying to do, but there's a good use case. Content you don't control. This is very useful for content from a CMS, maybe you have a blog written in Markdown and you want to render it on your blog. There's no easy way we can apply the classes we need to the markup, so instead, the typography plugin has a good set of rules to do the job.

What does Tailwind's Form plugin do?

Like the typography plugin, this applies some good default styling to form inputs. If you're using a component-based UI framework like React, Vue, Svelte, etc. there's probably not much reason to use this plugin because you can apply the classes you want directly in the component and reuse that component everywhere you need. The four plugin is useful in places where you want to provide a solid foundation to your forms without having to manually add the same set of classes to each input. It could also be used when you're generating a form and don't have control of each individual input.

Rapid prototyping

How Tailwind helps with creating UIs quickly

There are common tasks you often aim to achieve when styling a user interface. Common things like rounded corners, box shadows, focus and hover states. Tailwind includes classes for these common cases at a level which is at arm's reach. It's really a system that combines a consistent naming standard with modifiers and related entities like colours or spacing. If I want to a box shadow of a particular colour, let's say this particular red — `#fecaca`. In Tailwind, that's the colour token `red-200`. So for a box shadow colour, the class is `shadow-red-200`. What if we want the text to be the same colour? `text-red-200`. Background? `bg-red-200`. You get the idea. It's consistent, and you build up speed over time.

The joy of copy-paste

Tailwind controversially brings HTML and CSS closer together. It's criticised because you end up adding numerous classes to your markup instead of hiding it away in a CSS file. The downside is that your styling concerns are out in the open, but a side effect of this is that copy and paste is simpler than ever. This really helps with prototyping because you can copy markup and styling together, make some modifications and move on. It removes the busywork of copying the markup, going back to your CSS and file and copying the selector and rules, making modifications and giving it a new name then updating your markup one final time. It's not a lot of work, but I've felt Tailwind gets out of the way and allows me to enter a flow state.

Consistency and flexibility

CSS is flexible, consistency requires discipline

Tailwind has constraints that CSS doesn't, but this is a good thing. Adding some constraints results in more consistency. In CSS, if you wanted to ensure your spacing is consistent, you'd need discipline. On a team, you might need linting rules and best practices documentation. In Tailwind, the default setup gives you a well-designed set of constrains that make it easier to be consistent.

If you wanted to make a drop shadow, this is what Tailwind specifies:

- `shadow-sm`
- `shadow`
- `shadow-md`
- `shadow-lg`
- `shadow-xl`
- `shadow-2xl`
- `shadow-inner`

These are the sizes you'll use, as they're the path of least resistance. This makes for a much more consistent process of building UI. Compare this to CSS:

- `box-shadow: 0 1px 2px 0 rgb(0 0 0 / 0.05);`
- `box-shadow: 0 1px 0.125em 0 rgba(0, 0, 0, 0.05);`
- `box-shadow: 0 1px 0.5mm 0 #0000000d;`

These are roughly equivalent, but it's not immediately obvious. CSS provides flexibility, but it's much easier to create inconsistencies if you're not careful.

Banner photo by Johannes Plenio on Unsplash

Resources

- https://tailwindcss.com/docs/responsive-design
- https://tailwindcss.com/docs/dark-mode
- https://tailwindcss.com/docs/typography-plugin