Tailwind CSS Framework

Nathan Warner

Northern Illinois
University

Computer Science
Northern Illinois University
United States

Contents

Getting started

1.1 Using the CDN (Quick Start)

If you're just experimenting or making a small demo, create an HTML file and add the following

```
<script src="https://cdn.tailwindcss.com"></script>
```

To <head>... The full html setup might look something like

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
    ↪  initial-scale=1.0" />
    <title>My Tailwind Project</title>
    <!-- Tailwind via CDN (for prototyping only) -->
    <script src="https://cdn.tailwindcss.com"></script>
  </head>
  <body>
    ...
  </body>
</html>
```

This approach quickly gets you started but is limited in customization and performance optimizations.

1.2 Setting Up a Full Project with Tailwind (npm-Based Installation)

For a production-ready project, follow these steps

1. **Install Node.js and npm**: Make sure you have Node.js installed since npm is bundled with it. You can check installation by running:

   ```
   node -v
   npm -v
   ```

2. **Initialize Your Project**: In your project directory, initialize a new Node.js project

   ```
   npm init -y
   ```

3. **Install Tailwind**:

```
npm install tailwindcss @tailwindcss/cli
```

4. **Import tailwind in your CSS**: Add to your main CSS file

```
@import "tailwindcss";
```

5. **Start the Tailwind CLI build process**:

```
npx @tailwindcss/cli -i ./src/input.css -o ./src/output.css
↪ --watch
```

6. **Start using Tailwind in your HTML**

```
<link href="./output.css" rel="stylesheet">
```

1.3 What is Tailwind?

Tailwind CSS is fundamentally built around a comprehensive collection of CSS utility classes.

Utility classes are self-descriptive, single-purpose CSS classes

```
.flex {
    display: flex;
}
```

Developers use these functional classes to build without writing additional CSS because if the style is in the library, you can use it over and over and over

```
<div class="flex"> </div>
```

A more involved example may look something like

```css
* {
    margin: 0;
}
.fs {
    height: 100vh;
    width: 100vw;
    margin: 0;
}

.bg-pink {
    background-color: pink;
}

.flex {
    display: flex;
}

.justify-center {
    justify-content: center;
}

.align-center {
    align-items: center;
}

.box-red {
    width: 20vw;
    height: 20vh;
    background-color: red;
}
```

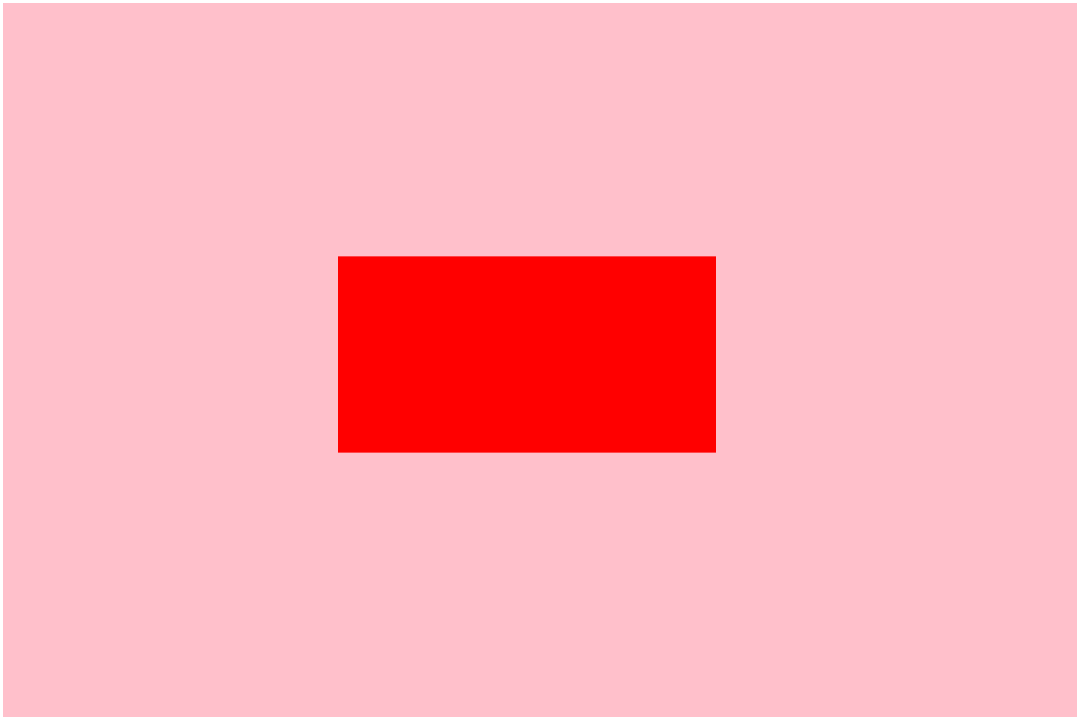
Then, in the html document, we can put them to use

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
    ↪  initial-scale=1.0">
    <title>test2</title>
    <link rel="stylesheet" href="styles.css">
    <link href='https://unpkg.com/boxicons@2.1.4/css/boxicons.mi
    ↪  n.css'
    ↪  rel='stylesheet'>
    <!-- <meta http-equiv="refresh" content="1"> -->
</head>

<body>
    <div class="fs flex justify-center align-center bg-pink">
        <div class="box-red"> </div>
    </div>
</body>

</html>
```



Variants and plugins

Variants are modifiers that allow your utility classes to respond to different states or conditions. Essentially, they generate alternative versions of your utilities that apply under specific circumstances, such as:

- **State-based conditions:** e.g., hover, focus, active, disabled
- **Responsive breakpoints:** e.g., sm:, md:, lg:, xl:
- **Contextual conditions:** e.g., group-hover, first, last, odd, even
- **Dark mode:** e.g., dark:

For example, writing hover:bg-blue-500 tells Tailwind to apply the blue background color only when the user hovers over the element. This system eliminates the need for writing custom CSS pseudo-class selectors by hand.

Plugins

Plugins are modules that extend the core functionality of Tailwind CSS. They let you add new utility classes, components, or even entirely new variant generators without modifying the Tailwind core. Plugins are especially powerful when you need to enforce a design pattern or add a set of repeated, custom styles across projects.

Media queries (Responsive design)

There are five breakpoints by default, inspired by common device resolutions:

Breakpoint prefix	Minimum width	CSS
sm	640px	@media (min-width: 640px) ...
md	768px	@media (min-width: 768px) ...
lg	1024px	@media (min-width: 1024px) ...
xl	1280px	@media (min-width: 1280px) ...
2xl	1536px	@media (min-width: 1536px) ...

To add a utility but only have it take effect at a certain breakpoint, all you need to do is prefix the utility with the breakpoint name, followed by the : character:

```
<!-- Width of 16 by default, 32 on medium screens, and 48 on
↪  large screens -->
<img class="w-16 md:w-32 lg:w-48" src="...">
```

This works for every utility class in the framework, which means you can change literally anything at a given breakpoint — even things like letter spacing or cursor styles.

4.1 Mobile first

By default, Tailwind uses a mobile first breakpoint system, similar to what you might be used to in other frameworks like Bootstrap.

Where this approach surprises people most often is that to style something for mobile, you need to use the unprefixed version of a utility, not the sm: prefixed version. Don't think of sm: as meaning "on small screens", think of it as "at the small breakpoint".

For this reason, it's often a good idea to implement the mobile layout for a design first, then layer on any changes that make sense for sm screens, followed by md screens, etc.

Tailwind's breakpoints only include a min-width and don't include a max-width, which means any utilities you add at a smaller breakpoint will also be applied at larger breakpoints.

If you'd like to apply a utility at one breakpoint only, the solution is to undo that utility at larger sizes by adding another utility that counteracts it.

Here is an example where the background color is red at the md breakpoint, but green at every other breakpoint

```
<div class="bg-green-500 md:bg-red-500 lg:bg-green-500">
  <!-- ... -->
</div>
```

You can completely customize your breakpoints in your tailwind.config.js file:

```
 // tailwind.config.js
module.exports = {
  theme: {
    screens: {
      'tablet': '640px',
      // => @media (min-width: 640px) { ... }

      'laptop': '1024px',
      // => @media (min-width: 1024px) { ... }

      'desktop': '1280px',
      // => @media (min-width: 1280px) { ... }
    },
  }
}
```

Hover, Focus, and Other States

Not all state variants are enabled for all utilities by default due to file-size considerations, but we've tried our best to enable the most commonly used combinations out of the box.

5.1 Hover

Add the hover: prefix to only apply a utility on hover.

```
<button class="bg-red-500 hover:bg-red-700 ...">
    Hover me
</button>
```

By default, the hover variant is enabled for the following core plugins:

- backgroundColor
- backgroundOpacity
- borderColor
- borderOpacity
- boxShadow
- gradientColorStops
- opacity
- rotate
- scale
- skew
- textColor
- textDecoration
- textOpacity
- translate

You can control whether hover variants are enabled for a plugin in the variants section of your tailwind.config.js file:

```
// tailwind.config.js
module.exports = {
    // ...
    variants: {
        extend: {
            padding: ['hover'],
        }
    },
}
```

5.2 Focus

Add the focus: prefix to only apply a utility on focus.

```
<input class="focus:ring-2 focus:ring-blue-600 ...">
```

By default, the focus variant is enabled for the following core plugins:

- accessibility
- backgroundColor
- backgroundOpacity
- borderColor
- borderOpacity
- boxShadow
- gradientColorStops
- opacity
- outline
- placeholderColor
- placeholderOpacity
- ringColor
- ringOffsetColor
- ringOffsetWidth
- ringOpacity
- ringWidth
- rotate
- scale
- skew
- textColor
- textDecoration
- textOpacity
- translate
- zIndex

You can control whether focus variants are enabled for a plugin in the variants section of your tailwind.config.js file:

```
// tailwind.config.js
module.exports = {
    // ...
    variants: {
        extend: {
            maxHeight: ['focus'],
        }
    },
}
```

5.3 Active

Add the active: prefix to only apply a utility when an element is active.

```
<button class="bg-green-500 active:bg-green-700 ...">
    Click me
</button>
```

By default, the active variant is not enabled for any core plugins.

You can control whether active variants are enabled for a plugin in the variants section of your tailwind.config.js file:

5.4 Visited

Add the visited: prefix to only apply a utility when a link has been visited.

```
<a href="#" class="text-blue-600 visited:text-purple-600
↪  ...">Link</a>
```

By default, the visited variant is not enabled for any core plugins.

You can control whether visited variants are enabled for a plugin in the variants section of your tailwind.config.js file:

5.5 Disabled

Add the disabled: prefix to only apply a utility when an element is disabled.

```
<button class="disabled:opacity-50 ...">
    Submit
</button>
<button class="disabled:opacity-50 ..." disabled>
    Submit
</button>
```

By default, the disabled variant is not enabled for any core plugins.

You can control whether disabled variants are enabled for a plugin in the variants section of your tailwind.config.js file:

5.6 Checked

Add the checked: prefix to only apply a utility when a radio or checkbox is currently checked.

```
<input type="checkbox" class="appearance-none
↪  checked:bg-blue-600 checked:border-transparent ...">
```

By default, the checked variant is not enabled for any core plugins.

You can control whether checked variants are enabled for a plugin in the variants section of your tailwind.config.js file:

5.7 Dark Mode

Now that dark mode is a first-class feature of many operating systems, it's becoming more and more common to design a dark version of your website to go along with the default design.

To make this as easy as possible, Tailwind includes a dark variant that lets you style your site differently when dark mode is enabled:

```
<div class="bg-white dark:bg-gray-800">
  <h1 class="text-gray-900 dark:text-white">Dark mode is
  ↪  here!</h1>
  <p class="text-gray-600 dark:text-gray-300">
    Lorem ipsum...
  </p>
</div>
```

It's important to note that because of file size considerations, the dark mode variant is not enabled in Tailwind by default.

To enable it, set the darkMode option in your tailwind.config.js file to media:

```
// tailwind.config.js
module.exports = {
    darkMode: 'media',
    // ...
}
```

By default, when darkMode is enabled dark variants are only generated for color-related classes, which includes text color, background color, border color, gradients, and placeholder color.

Container

The container class sets the max-width of an element to match the min-width of the current breakpoint. This is useful if you'd prefer to design for a fixed set of screen sizes instead of trying to accommodate a fully fluid viewport.

To center a container, use the mx-auto utility:

```
<div class="container mx-auto">
    <!-- ... -->
</div>
```

To add horizontal padding, use the px-{size} utilities:

```
<div class="container mx-auto px-4">
  <!-- ... -->
</div>
```

Display

Class	Properties
block	display: block;
inline-block	display: inline-block;
inline	display: inline;
flex	display: flex;
inline-flex	display: inline-flex;
table	display: table;
inline-table	display: inline-table;
table-caption	display: table-caption;
table-cell	display: table-cell;
table-column	display: table-column;
table-column-group	display: table-column-group;
table-footer-group	display: table-footer-group;
table-header-group	display: table-header-group;
table-row-group	display: table-row-group;
table-row	display: table-row;
flow-root	display: flow-root;
grid	display: grid;
inline-grid	display: inline-grid;
contents	display: contents;
list-item	display: list-item;
hidden	display: none;

Floats

Utilities for controlling the wrapping of content around an element.

- Use float-right to float an element to the right of its container.
- Use float-left to float an element to the left of its container.
- Use float-none to reset any floats that are applied to an element. This is the default value for the float property.

Object Fit

Class	Properties
object-contain	object-fit: contain;
object-cover	object-fit: cover;
object-fill	object-fit: fill;
object-none	object-fit: none;
object-scale-down	object-fit: scale-down;

Object Position

Class	Properties
object-bottom	object-position: bottom;
object-center	object-position: center;
object-left	object-position: left;
object-left-bottom	object-position: left bottom;
object-left-top	object-position: left top;
object-right	object-position: right;
object-right-bottom	object-position: right bottom;
object-right-top	object-position: right top;
object-top	object-position: top;

Overflow

Class	Properties
overflow-auto	overflow: auto;
overflow-hidden	overflow: hidden;
overflow-visible	overflow: visible;
overflow-scroll	overflow: scroll;
overflow-x-auto	overflow-x: auto;
overflow-y-auto	overflow-y: auto;
overflow-x-hidden	overflow-x: hidden;
overflow-y-hidden	overflow-y: hidden;
overflow-x-visible	overflow-x: visible;
overflow-y-visible	overflow-y: visible;
overflow-x-scroll	overflow-x: scroll;
overflow-y-scroll	overflow-y: scroll;

Position

Class	Properties
static	position: static;
fixed	position: fixed;
absolute	position: absolute;
relative	position: relative;
sticky	position: sticky;

Visibility

Class	Properties
visible	visibility: visible;
invisible	visibility: hidden;

Z-Index

Class	Properties
z-0	z-index: 0;
z-10	z-index: 10;
z-20	z-index: 20;
z-30	z-index: 30;
z-40	z-index: 40;
z-50	z-index: 50;
z-auto	z-index: auto;