How to configure a Next.js app with TypeScript and Tailwind CSS?

Feb 3, 2022 , by Akhil G Krishnan

3 minute read

In this article, we will talk about how to create an application using Next.js, Tailwind CSS and TypeScript. This blog will also provide a walkthrough on how to use Purgecss to remove unused CSS styles. Let's get started-

Step 1: Bootstrap the project with Create Next App

In order to set up the boilerplate for the Next.js web app, we have to use Create Next app - the Next.js equivalent of Create React App. We will be using this along with with-typescript from the Next.js example project repo to set up TypeScript. Use the following code-

```
npx create-next-app next-tailwind-typescript-starter --example with-typescript

cd next-tailwind-typescript-starter
```

Step 2: Install Tailwind CSS Dependencies

Now, let's install the Tailwind packages.

```
# With Yarn
yarn add -D postcss-preset-env tailwindcss

# With Npm
npm i -D postcss-preset-env tailwindcss
```

We can also optionally create a `tailwind.config.js` file with the help of npx to run the Tailwind CLI.

```
npx tailwind init
```

Step 3: Setup PostCSS Build

Once Tailwind is installed, we have to set it up to be created with PostCSS.

First, we have to configure the config file.

```
touch postcss.config.js
```

Now, we have to add Tailwind and the PostCSS Webpack preset to the plugins section.

```
module.exports = {
  plugins: [
    'tailwindcss',
    'postcss-preset-env',
  ],
}
```

The PostCSS plugin system requires the plugins to be added as the string type.

Step 4: Add Tailwind to CSS file

Now, that the setup is done, we can focus on getting the CSS into Next.js. First, we need to configure a styles directory and an `index.css` file.

```
mkdir styles; touch styles/index.css
```

We can use the `@tailwind` directive to insert the base, components, and utilities styles within the CSS file.

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Step 5: Import Global CSS

Now, we have to import `index.css` file into a component so Tailwind can be utilized all over the app.

Note: We can only import stylesheets in the `pages/_app.{js,ts,jsx,tsx}` file.

Since we're using TypeScript, we have to build an `app.tsx` file.

```
touch pages/_app.tsx
```

Within the `_app.tsx` file, we have to create a custom App component that will be passed as `AppProps` and will import the `index.css` file.

```
import React from 'react'
import { AppProps } from 'next/app'

import '../styles/index.css'

function MyApp({ Component, pageProps }: AppProps) {
  return <Component {...pageProps} />
}

export default MyApp;
```

Now the entire web app can access the Tailwind CSS classes!

Step 6: Configure Purgecss (Optional)

Here, the major trick is to reduce the CSS file size. One of the simplest ways to reduce the file size is to leverage Purgecss to remove any unrequired styles from the final CSS file.

To set up Purgecss, first, we have to install it, using the following code.

```
# With Yarn
yarn add -D @fullhuman/postcss-purgecss

# With npm
npm i -D @fullhuman/postcss-purgecss
```

Once done, we can add the plugin and make it run through the CSS class names in all JavaScript and TypeScript files in the pages and components directories. If there is a class name that isn't used in any of these files, it will simply remove the respective styles from the CSS file.

```
module.exports = {
  plugins: [
    'tailwindcss',
    process.env.NODE_ENV === 'production' ?
      [ '@fullhuman/postcss-purgecss', {
          content: [
            './pages/**/*.{js,jsx,ts,tsx}',
            './components/**/*.{js,jsx,ts,tsx}',
          ],
          defaultExtractor: content => content.match(/[\w-/:]+(?<!:)/g) || [],`
        },
      ] : undefined,
    'postcss-preset-env',`
  ],
}
```

We can also ensure that Purgecss is only added in production.

Finally, we have to select the base and components styles and mark them as positive, so that Purgecss doesn't remove them.

Here, we can use a comment to mark them positive in both the sections at the same time.

```
/* purgecss start ignore */
@tailwind base;
@tailwind components;
/* purgecss end ignore */
@tailwind utilities;
```

Final Thoughts

We hope this blog was helpful! To know more, check out the Github repo to view the source code for the final boilerplate.

Share this post!

If you enjoyed this post, you might also like:

The Ultimate Guide to Gemfile and Gemfile.lock
August 16, 2022

What's New in Tailwind CSS v3?
January 6, 2022

Handling attachments in Action Text in Rails 6
November 12, 2019