Chapter# 10: Tailwind in Action

In this chapter, we will learn how to style our application using Tailwind CSS.

Different ways to style components in React

**External CSS file**: CSS for components is written in a central file, like index.css.

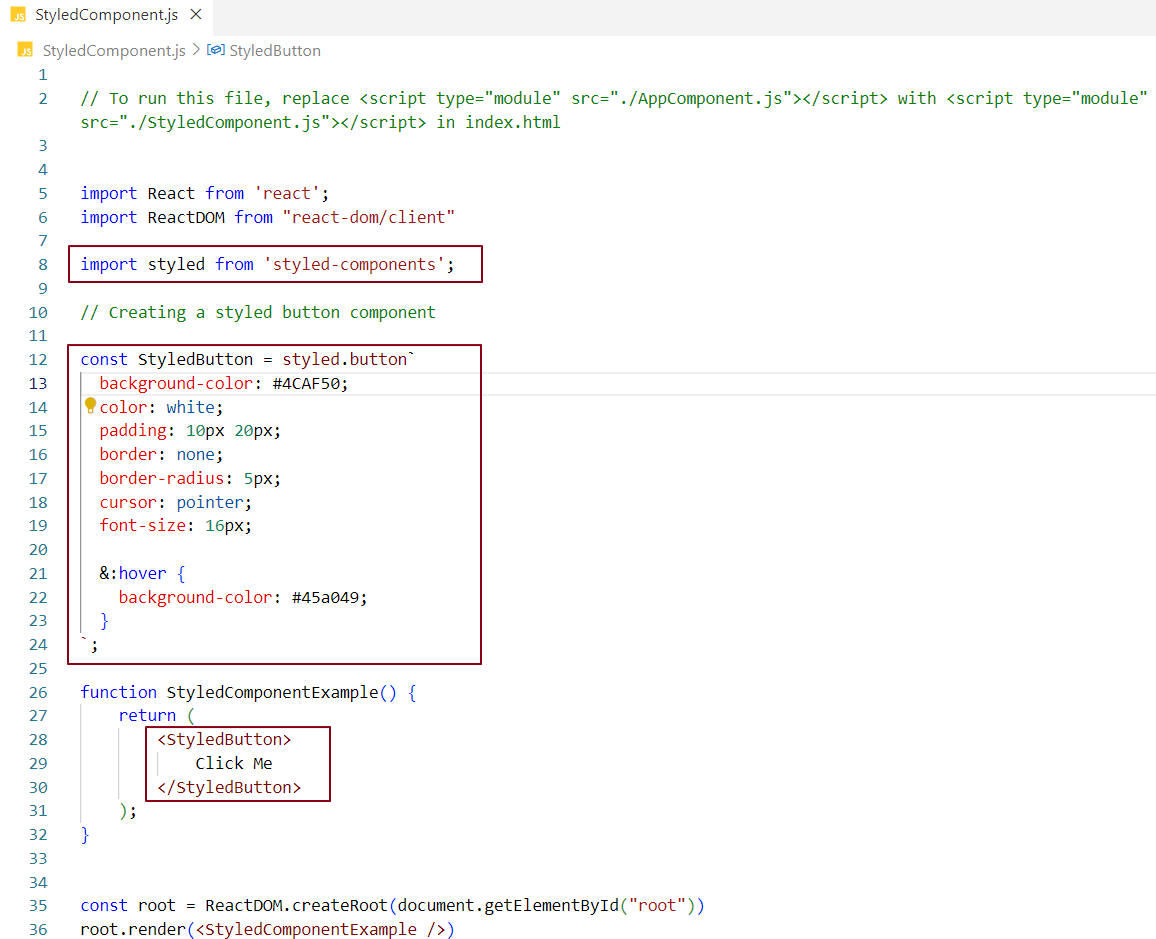
**External SCSS file**: SCSS for components is written in a central file, like index. scss.

**Inline CSS**: Styles are defined within a JavaScript object and applied to the component’s style attribute in JSX.



**Styling libraries**: Libraries like Material UI and Ant Design provide pre-styled components and utilities.

**Styled Components**: Styled Components let you write CSS directly inside JavaScript, similar to how JSX lets you write HTML-like code in JavaScript. This approach makes it easy to create reusable components with their own styles.



**Importing styled-components:** We import the styled component from the styled-components library to create styled elements in React.



**Creating Styled Components:** You can access various HTML elements using styled. <elementName>, where <elementName> is any valid HTML tag (e.g., button, div, section, etc.), and then define styles for those elements.

Syntax: const StyledElement = styled.button` /\* CSS styling here \*/ `;



Here, StyledElement is a styled button StyledButton, and you can replace button with other HTML tags to create different styled components.

**Using Styled Components:** Once a styled component is created, you can use it in your JSX just like any other React component. This allows you to apply the styles directly within the component.



In this example, StyledButton is a styled component that you can reuse throughout your application.

Advantages and disadvantages of using styling libraries and frameworks

**Advantages:**

* Easy to use and reusable.
* Saves a lot of time, speeding up development.
* Provides automatic themes.
* Ensures a consistent UI (A consistent UI means that all UI elements, such as buttons and textboxes, look the same, enhancing the user experience. We don’t want different button styles on different pages; the design should be uniform across the app).
* Takes care of responsiveness, ensuring the UI is compatible with all devices.

**Disadvantages:**

* Can result in a heavy bundle size.
* Limits control over design customization. For example, using Material UI buttons throughout the app makes it difficult to customize them as per personal preferences. While development is easier and faster, customization requires more effort and time.
* Difficult to integrate into heavily customized legacy front ends.
* May be overly complex for simple projects.
* Updates can introduce bugs.
* Some library features may not be supported on older devices and mobile devices.

How did we use to style our components?

Earlier, we maintained a single CSS file, index.css, where we wrote CSS for all components of the project. This file served as the central place for styling. We would then include index.css in our main index.html to apply the styles to the application. However, this approach has several drawbacks:

**Disadvantages:**

* All components were styled together in one central file, index.css, which causes the file to grow in size with every styling update.
* As the file grows, it becomes difficult to maintain and test.
* The CSS written is not optimized.
* This approach impacts reusability. If we want to reuse a style from one component in another, it is not straightforward.
* It leads to hardcoded styling.
* Processing inline CSS can be heavy for the browser to handle efficiently.

To address these issues, Tailwind CSS offers a better solution.

What is Tailwind CSS?

Tailwind is an open-source CSS framework that allows us to write CSS directly within the same component JSX file, similar to inline styling. This eliminates the need to switch between the CSS file and the component JSX file.

**Perks / Why Choose Tailwind:**

* **CSS on the go**: Tailwind enables faster development by allowing us to write styles directly in the JSX, saving time.
* **Reusability**: Tailwind promotes the reuse of utility classes across components.
* **Smaller bundle size**: Tailwind provides minimal, optimized CSS, resulting in smaller bundle sizes.
* **Flexibility and customization**: Unlike Material UI, Tailwind offers greater flexibility and customization.
* **Prebuilt classes**: It provides a wide range of utility classes, reducing the need to write custom CSS.
* **Less code and easier debugging**: Since styles are defined in JSX, there’s less code to manage, making it easier to debug.
* **No duplicate CSS**: Tailwind’s bundler automatically removes duplicate CSS, preventing redundancy.
* **Dynamic styling**: It supports dynamic styling with custom CSS values.

**Disadvantages of Tailwind:**

* **Initial learning curve**: It may be challenging for new developers to learn, as it introduces a new way of styling.
* **Code readability**: Using many utility classes to style an element can make the code harder to read and cluttered.

**Note:** Inline styling and other styling methods each have their drawbacks. Tailwind inherits only the beneficial aspects of these approaches. For instance, "CSS on the go" is a feature that Tailwind borrowed from inline styling, enabling faster development.

How to include Tailwind into our project?

**Method 1**: Use the CDN link to allow Tailwind to control the styling of your UI elements and components.

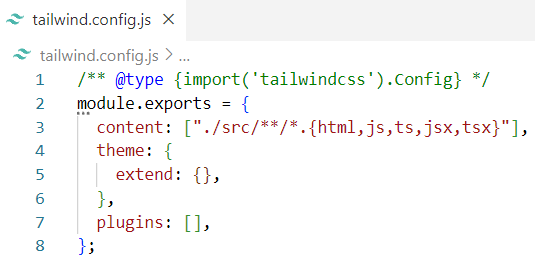


**Method 2 - Tailwind Installation with Parcel**: [Tailwind Installation Guide](https://tailwindcss.com/docs/guides/parcel)

* Install Tailwind CSS by running the command: npm install -D tailwindcss
* After installing Tailwind, install PostCSS using the command: npm install -D postcss

**Why PostCSS?**  
PostCSS is needed because it allows Parcel to understand that we are using Tailwind, so it can compile the Tailwind class properties into CSS that browsers can interpret.

* Configure Tailwind by running the command: npx tailwindcss init

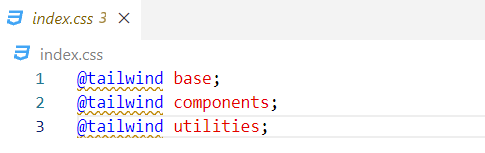


In the content section, we specify a regular expression for the files where Tailwind will be used. This configuration ensures that the specified files can use Tailwind's CSS.

* Create a new file named **.postcssrc** in the project root directory.  
  The **.postcssrc** file should contain the following configuration, which tells Parcel that our project will use many Tailwind classes. This ensures that when bundling, Parcel compiles Tailwind's CSS into regular CSS.



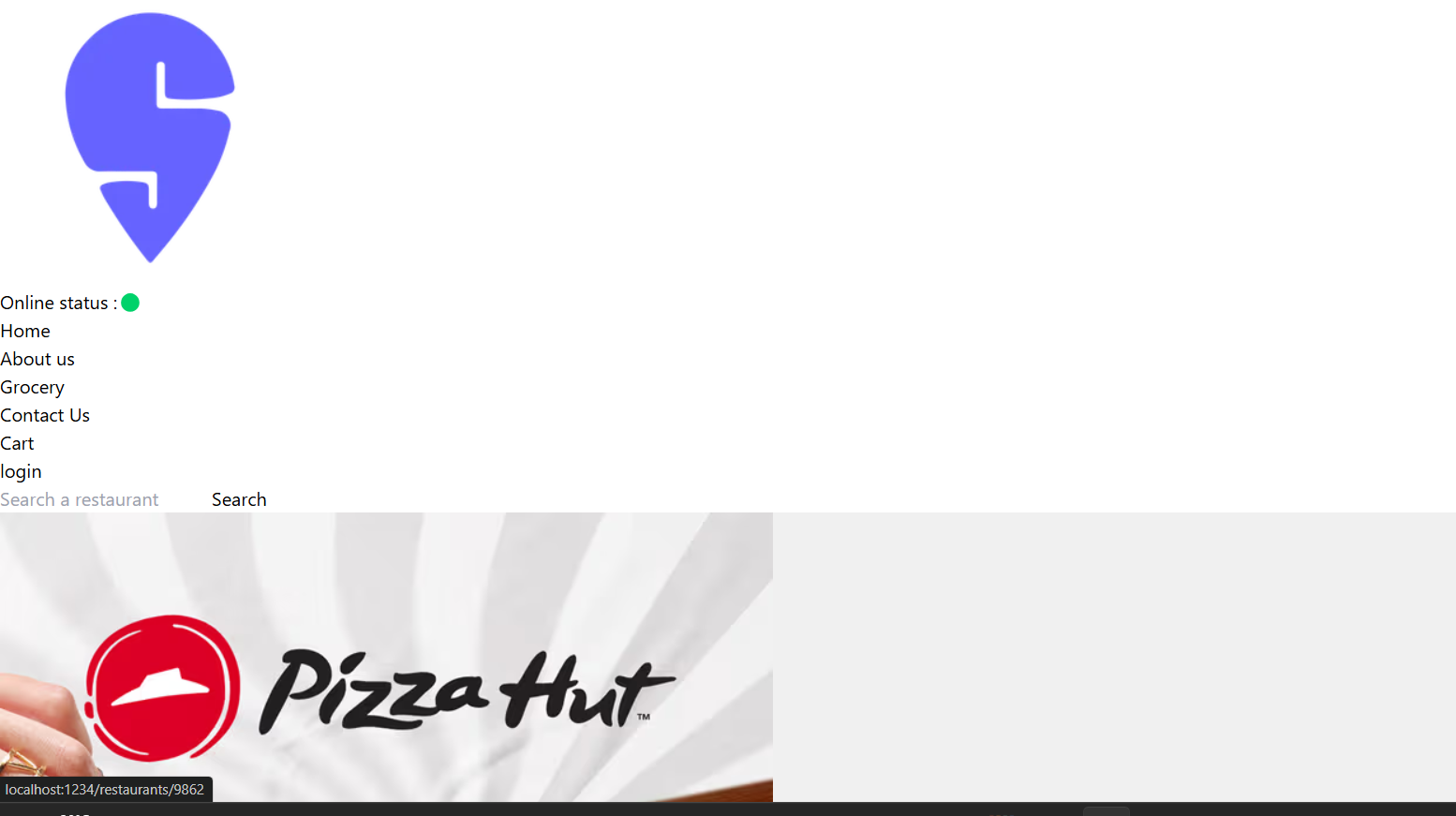
* Add the following three lines to index.css. This ensures that when Parcel processes index.css, it knows that Tailwind is being used for base components and utilities in the project.



We're done! Now, we can use Tailwind classes throughout our components.

Tailwind Controlling UI Elements

When we run npm start, Tailwind styles will override the default HTML styles of the UI.



Note – Tailwind provides a className for every CSS rule we want to apply inside our app.

Resources and Home Work

**Resources**

* Styled Components documentation: <https://styled-components.com/docs/basics>
* Tailwind documentation: <https://v2.tailwindcss.com/docs>
* VS Code Extension for Tailwind: Tailwind CSS IntelliSense

**Homework:**

Make your app look good using Tailwind classes.

* **Answer:** I have already implemented this in my project. Please refer to the "NamasteReact\_Tailwind\_In\_Action" project on GitHub.

After using Tailwind in my project, the UI looks like this:

