Styling in React

Styling in React

Styling is one of the most important aspects of the React application. There are various ways to follow when planning to style React components. Some of the most popular and modern styling strategies are:

- CSS Stylesheets
- Inline Styling
- Styled Components
- CSS Modules

CSS Style sheets

This is the conventional way of styling websites. In this method, we separate the CSS part into an external file with a .css extension which is simply imported into the React component. After that, we can give className and id to point which styles should point to which element.

Note: **class** attribute is used in HTML, whereas **className** is used in React. This is because class is a reserved keyword in JavaScript and since React uses JSX, which is a syntax extension to JavaScript, we must use className instead of the class attribute.

Example: Styling the Navbar

Basic Structure of the Navbar Component

Navbar.js

```
import "./Navbar.css";
```

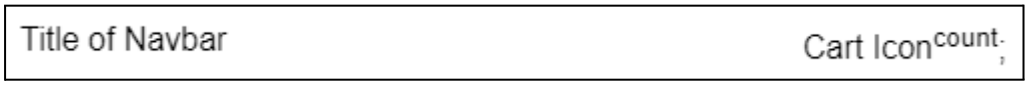
```
const Navbar = () => {
 return (
   <div className="navbar">
     <span>Title of Navbar</span>
     <span>
       Cart Icon<sup>count</sup>
     </span>
   </div>
 );
};

export default Navbar;
```

Navbar.css

```
.navbar {
 display: flex;
 justify-content: space-between;
}
```

Output:



Advantages:

- Styles of numerous documents can be organized from one single file.
- Good performance as it is easy for the browser to optimize and cache the files locally for repeated visits.
- You can very easily rip out the entire stylesheet and create a new one to refresh the look and feel of your app.

Disadvantages:

- If not properly structured, It can become long and difficult to navigate through as the application becomes complex.

- CSS Stylesheets have global scopes and can cause conflicts in styles if the same selector names are used in the codebase

Inline Styling

Inline CSS is the widely preferred but less recommended way to style your website. In React, you will write your style using the style attribute followed by = and then CSS properties enclosed by double curly braces {{ }} instead of quotes “ ”. React uses JSX, In JSX for evaluation of any variable, state object , expression etc has to be enclosed in {}. The style attribute in React only accepts a JavaScript object with camelCased properties and values enclosed with quotes rather than a CSS string. This is the reason there are two pairs of curly braces.

Note: Inline styles have got more priority, and they will overwrite any other styles given to them in any manner.

Example: Styling the Navbar

Navbar.js - Method 1 (inline styling)

```
const Navbar = () => {
 return (
   <div style={{display:"flex", justifyContent:"space-between"}}>
     <span>Title of Navbar</span>
     <span>
       Cart Icon<sup>count</sup>
     </span>
   </div>
 );
};

export default Navbar;
```

Navbar.js - Method 2 (internal style object)

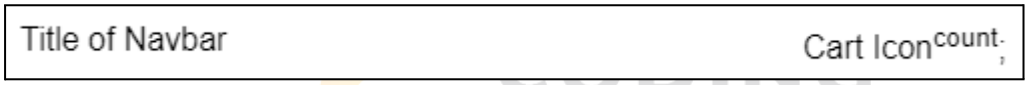
```
const Navbar = () => {
 return (
   <div style={styles}>
```

```
    <span>Title of Navbar</span>
    <span>
      Cart Icon<sup>count</sup>
    </span>
  </div>
 );
};

let styles = {
 display: "flex",
 justifyContent: "space-between"
};

export default Navbar;
```

Output:



Advantages:

- Inline CSS is best suited for learners and when you are testing a particular style.

Disadvantages:

- It cannot be reused, i.e., you must write the same CSS code repeatedly for the same styles.
- It does not provide browser cache advantages.
- Some useful CSS properties like pseudo-codes, pseudo-classes, media queries, etc., cannot be used in inline styles.

Styled Components

`styled-components` is a library for React that allows you to use component-level styles in your application that are written with a mixture of

JavaScript and CSS using a technique called CSS-in-JS. This is done using the tagged template literal syntax. Follow the following steps to implement styling using styles-components:

1. First, we need to installthe styled-components library by running `npm install styled-components`.
2. We then need to import the styled component library into our component by writing `import styled from 'styled-components'`.
3. Now we can create a variable by selecting a particular HTML element where we store our style keys.
4. Then we use the name of the variable we created as a wrapper around our JSX elements.

Example: Styling the Navbar

Navbar.js

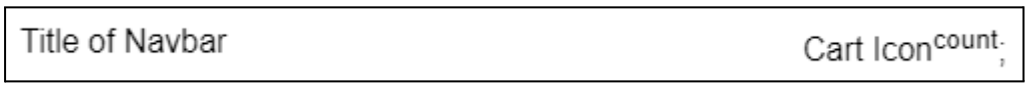
```
import styled from "styled-components";

const Nav = styled.div`
 display: flex;
 justify-content: space-between;
`;

const Navbar = () => {
 return (
   <Nav>
     <span>Title of Navbar</span>
     <span>
       Cart Icon<sup>count</sup>
     </span>
   </Nav>
 );
};

export default Navbar;
```

Output:



Example: Dynamic Styling with props

One of the advantages of styled-components is that the components themselves are functional, as in you can use props within the CSS. You can also use conditional statements to change styles based on a state or prop.

Navbar.js

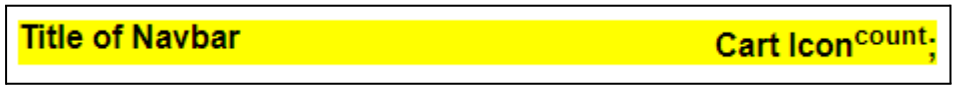
```
import styled from "styled-components";

const Nav = styled.div`
 display: flex;
 justify-content: space-between;
 background-color: ${(props) => props.color};
 font-weight: ${(props) => (props.bold ? "normal" : "bolder")};
`;

const Navbar = () => {
 return (
   <Nav color="yellow" bold="">
     <span>Title of Navbar</span>
     <span>
      Cart Icon<sup>count</sup>
    </span>
  </Nav>
);
};

export default Navbar;
```

Output



Advantages:

- Styled components eliminate specificity problems as it encapsulates CSS inside a component.
- styled-components allow you to combine CSS and JS in the same file.
- You can make use of props to dynamically change the styles in any way.

Disadvantages:

- Writing CSS in JS means separating the two in the future will be difficult, which is terrible for maintainability.
- Differentiating between styled and React components can be difficult
- For applications that use styled components, the browser downloads the CSS and parses it using JavaScript before injecting them into the page. This causes performance issues because the user must download a lot of JavaScript in the initial load.

CSS Modules

A CSS Module is a CSS file with a `.module.css` extension in which all class names and animation names are scoped locally by default. One huge advantage of the CSS modules is that it is locally scoped to the component which prevents conflicting styles because of using the same selector names.

The CSS properties are hashed into unique class names during compilation. You can use CSS Modules by creating a file with extension `.module.css` file and import it into the specific React Component file.

Example: Styling the Navbar

Navbar.js

```
import styles from "./Navbar.module.css";


const Navbar = () => {
```

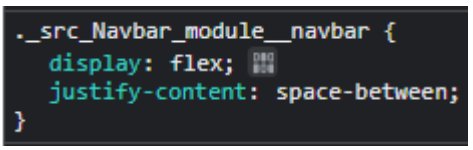
```
return (
  <div className={styles.navbar}>
    <span>Title of Navbar</span>
    <span>
      Cart Icon<sup>count</sup>
    </span>
  </div>
);
};
export default Navbar;
```

Navbar.module.css

```
.navbar {
 display: flex;
 justify-content: space-between;
}
```

Output:

Note: When you check it in the browser. On inspecting, The class name is _src_Navbar_module__navbar which is further transformed into a Unique Identifier. This will remove any chances of name collision in the React App.



Advantages:

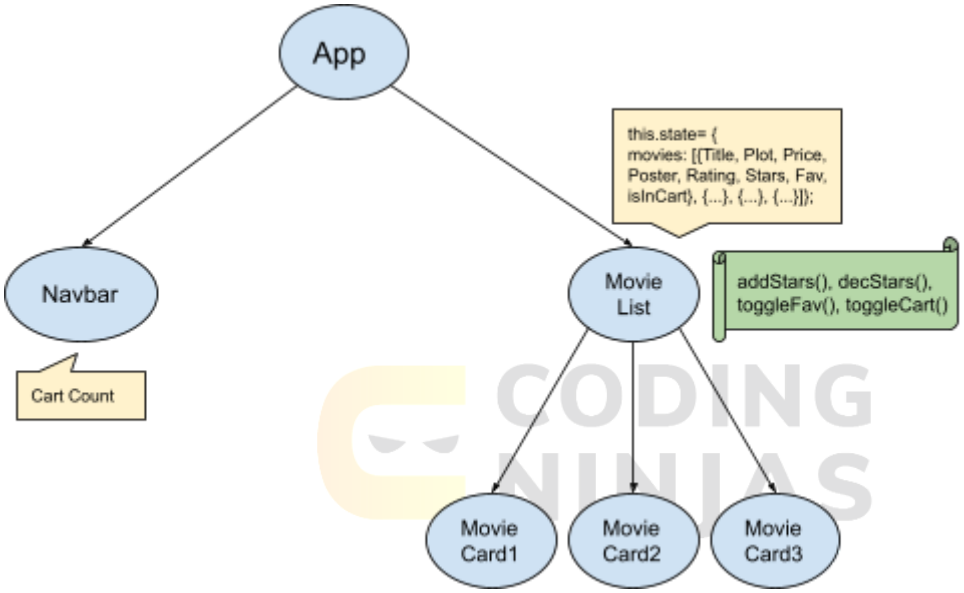
- Modular and reusable CSS
- No more styling conflicts, So, you can use the same CSS class in multiple CSS files

Disadvantages:

- Using the styles object whenever constructing a className is compulsory.
- Only allows usage of camelCase CSS class names.

Lifting up the State

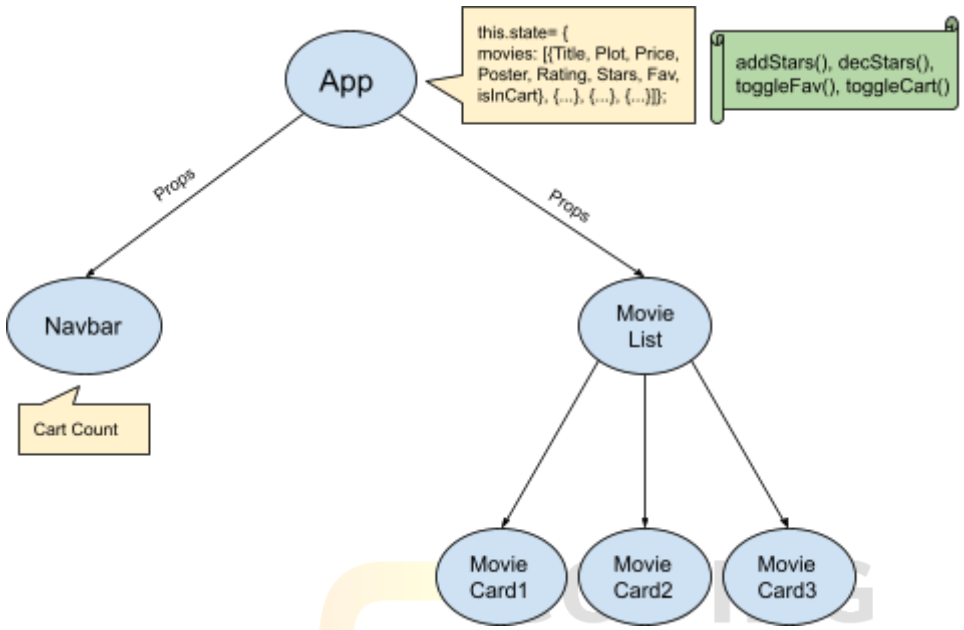
Lifting state up is a common pattern that is essential for React developers to know. It helps you avoid more complex (and often unnecessary) patterns for managing your state. The app structure of the project is shown below:



In `<Navbar>`, we have a **cart count** which we want to increase/ decrease as the add to cart/ remove from cart button is pressed. These buttons are present under `<MovieCards>` but the logic/ event handlers for add to cart and remove from cart are written inside `<MovieList>`. We want to access cartCount in `<Navbar>` component which is a sibling of this component, but we cannot pass the data among siblings as per the rule we can only pass data from parent to child.

```
this.state = {
        movies: movies,
        cartItems :[],
        cartCount:0
    }
```

The solution is **Lifting up the state to App component** and passing the **cartCount** from `<App>` component to `<Navbar>`. So with states we can take all the handlers to the `<App>` component. And then we can pass everything as a **prop** to another component.




```
render(){
 const {movies, cartItems,cartCount} = this.state;
 return(
   <>
   <Navbar cartItems = {cartItems} cartCount={cartCount}/>
   <MovieListS movies={movies}
               decStars = {this.decStars}
               addStars = {this.addStars}
               toggleCart = {this.toggleCart}
               toggleFav = {this.toggleFav} />
   </>

 )
}
```

In the Navbar component we can now access CartCount and in MovieList, we can access all the handlers through props.

Presentational Components

A component that has to have a state, make calculations based on props or manage any other complex app logic is called a container component

A component whose only job is to contain some JSX and render it in UI is called a presentational component. A presentational component must be exported and will never render anything on its own. It will always be rendered by a container component.

All presentation components can be changed from class-based to stateless functional components as shown in the following example.

Navbar.js (class-based)

```
import { Component } from "react";
import styles from "./navbar.module.css";

class Navbar extends Component {
 render() {
   return (
     <div className={styles.navbar}>
       <span>Title of Navbar</span>
       <span>
         Cart Icon<sup>count</sup>
       </span>
     </div>
   );
 }
}
export default Navbar;
```

Navbar.js (stateless functional)

```
import styles from "./navbar.module.css";

function Navbar() {
 return (
   <div className={styles.navbar}>
     <span>Title of Navbar</span>
     <span>
```

```
        Cart Icon<sup>count</sup>
      </span>
    </div>
  );
}
export default Navbar;
```

Note: Functional components can create and maintain their internal state using React hooks. They can also be a container component.

Summarising it

Let's summarise what we have learned in this Lecture:

- Learned about the styling strategies in React.
- Learned about styling React components using CSS Stylesheets.
- Learned about styling React components using Inline Styling.
- Learned about styling React components using styled components.
- Learned about styling React components using CSS Modules.
- Learned about lifting up the state.
- Learned about presentational components.

Some References:

- Styling and CSS in React: link
- Inline CSS vs Raw CSS: link
- Styled Components: link
- Dynamic Styling with styled-components: link
- CSS Modules: link
- Lifting state up in React: link
- Container vs Presentational components: link