React-CSS Modules

CSS Modules: A Better Way to Style Your React Components

In traditional CSS, styles are global and can cause naming conflicts. To avoid this, React introduced **CSS Modules**, a way to scope styles to individual components, making it easier to manage and reuse code.

Key Features of CSS Modules:

1. **Scoped Styles**: Styles are attached to the component's scope, ensuring no naming conflicts.
2. **Locally-Defined Namespaces**: Components can define their own namespace for styles using a custom module name.
3. **Efficient Code Splitting**: Only required styles are loaded, reducing bundle size.

Example Usage:

Let's say we have a `Header` component with its own styles in a separate file (`Header.module.css`):

```
 /* src/components/Header/Header.module.css */
.header-container {
    display: flex;
    justify-content: space-between;
}

.logo {
    width: 20px;
}
```

In our React component, we can import the styles using ES6 syntax:

```
 // src/components/Header.js
import React from 'react';
import styles from './Header.module.css';

const Header = () => {
    return (
        <div className={styles.headerContainer}>
            <img src="logo.png" alt="Logo" className={styles.logo} />
            <nav>
                <ul>
                    {/* Navigation items */}
                </ul>
            </nav>
        </div>
    );
};

export default Header;
```

Benefits:

- **No Naming Conflicts**: Styles are scoped to the component, reducing naming conflicts.
- **Efficient Code Splitting**: Only required styles are loaded, minimizing bundle size.

Best Practices:

- Use a consistent naming convention for your CSS modules (e.g., `Component.module.css`).
- Keep related components and their styles in the same directory.
- Avoid mixing global and local styles; use global styles sparingly or in specific cases where required.

Curated by Brajesh Kumar