CSS Modules in React

CSS Modules are an advanced **CSS-in-CSS** approach. The CSS file stays the same, where you could apply CSS extensions like Sass, but its use in React components changes. To enable CSS modules in create-react-app, rename the *src/App.css* file to *src/App.module.css*. This action is performed in the command line from your project's directory:

```
mv src/App.css src/App.module.css
```

> **Note:** This command is to be executed when you run this on your local machine. You can see the live execution of code at the end of this lesson.

In the renamed *src/App.module.css*, start with the first CSS class definitions, as before:

```css
.container {
  height: 100vw;
  padding: 20px;

  background: #83a4d4; /* fallback for old browsers */
  background: linear-gradient(to left, #b6fbff, #83a4d4);

  color: #171212;
}

.headlinePrimary {
  font-size: 48px;
  font-weight: 300;
  letter-spacing: 2px;
}
```

src/App.module.css

Import the *src/App.module.css* file with a relative path again. This time, import it as a JavaScript object where the name of the object (here `styles`) is up to you:

```
import React from 'react';
import axios from 'axios';

import styles from './App.module.css';
```

src/App.js

Instead of defining the `className` as a string mapped to a CSS file, access the CSS class directly from the `styles` object, and assigns it with a JavaScript in JSX expression to your elements.

```
const App = () => {
  ...

  return (

    <div className={styles.container}>
      <h1 className={styles.headlinePrimary}>My Hacker Stories</h1>


      <SearchForm
        searchTerm={searchTerm}
        onSearchInput={handleSearchInput}
        onSearchSubmit={handleSearchSubmit}
      />

      {stories.isError && <p>Something went wrong ...</p>}

      {stories.isLoading ? (
        <p>Loading ...</p>
      ) : (
        <List list={stories.data} onRemoveItem={handleRemoveStory} />
      )}
    </div>
  );
};
```

src/App.js

There are various ways to add multiple CSS classes via the `styles` object to the element's single `className` attribute. Here, we use JavaScript template literals:

```
const Item = ({ item, onRemoveItem }) => (

  <div className={styles.item}>
    <span style={{ width: '40%' }}>

      <a href={item.url}>{item.title}</a>
    </span>

    <span style={{ width: '30%' }}>{item.author}</span>
    <span style={{ width: '10%' }}>{item.num_comments}</span>
    <span style={{ width: '10%' }}>{item.points}</span>
    <span style={{ width: '10%' }}>

      <button
        type="button"
        onClick={() => onRemoveItem(item)}

        className={`${styles.button} ${styles.buttonSmall}`}

      >
        Dismiss
      </button>
    </span>
  </div>
);
```

src/App.js

We can also add inline styles as more dynamic styles in JSX again. It's also possible to add a CSS extension like Sass to enable advanced features like CSS nesting. We will stick to native CSS features though:

```css
.item {
  display: flex;
  align-items: center;
  padding-bottom: 5px;
}

.item > span {
  padding: 0 5px;
  white-space: nowrap;
  overflow: hidden;
  white-space: nowrap;
  text-overflow: ellipsis;
}

.item > span > a {
  color: inherit;
}
```

src/App.css

Then the button CSS classes in the *src/App.module.css* file:

```css
.button {
  background: transparent;
  border: 1px solid #171212;
  padding: 5px;
  cursor: pointer;

  transition: all 0.1s ease-in;
}

.button:hover {
  background: #171212;
  color: #ffffff;
}

.buttonSmall {
  padding: 5px;
}

.buttonLarge {
  padding: 10px;
}
```

src/App.module.css

There is a shift toward pseudo BEM naming conventions here, in contrast to `button_small` and `button_large` from the previous section. If the previous naming convention holds true, we can only access the style with `styles['button_small']` which makes it more verbose because of JavaScript's limitation with object underscores. The same shortcomings would apply for classes defined with a dash (`-`). In contrast, now we can use `styles.buttonSmall` instead (see: Item component):

```
const SearchForm = ({ ... }) => (
```

```
  <form onSubmit={onSearchSubmit} className={styles.searchForm}>

    <InputWithLabel ... >

      <strong>Search:</strong>
    </InputWithLabel>

    <button
      type="submit"
      disabled={!searchTerm}

      className={`${styles.button} ${styles.buttonLarge}`}

    >
      Submit
    </button>
  </form>
);
```

src/App.js

The SearchForm component receives the styles as well. It has to use string interpolation for using two styles in one element via JavaScript's template literals. One alternative way is the classnames library, which is installed via the command line as project dependency:

```
import cs from 'classnames';

...

// somewhere in a className attribute
className={cs(styles.button, styles.buttonLarge)}
```

src/App.js

The library offers conditional stylings as well. Finally, continue with the InputWithLabel component:

```
const InputWithLabel = ({ ... }) => {
  ...

  return (
    <>

      <label htmlFor={id} className={styles.label}>

        {children}
      </label>
       
      <input
        ref={inputRef}
        id={id}
        type={type}
        value={value}
```

```
        onChange={onInputChange}

        className={styles.input}

      />
    </>
  );
};
```

src/App.js

And finish up the remaining style in the *src/App.module.css* file:

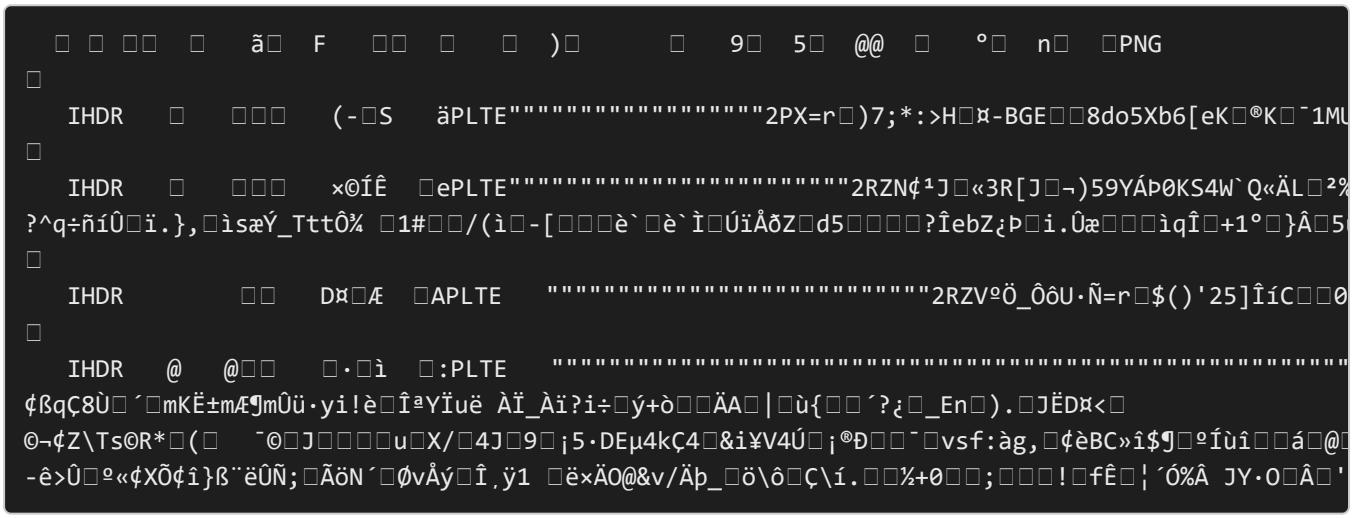
```css
.searchForm {
  padding: 10px 0 20px 0;
  display: flex;
  align-items: baseline;
}

.label {
  border-top: 1px solid #171212;
  border-left: 1px solid #171212;
  padding-left: 5px;
  font-size: 24px;
}

.input {
  border: none;
  border-bottom: 1px solid #171212;
  background-color: transparent;

  font-size: 24px;
}
```

src/App.module.css

The same caution applies as the last section: some of these styles like `input` and `label` might be more efficient in a global *src/index.css* file without CSS modules.

Again, CSS Modules–like any other CSS-in-CSS approach–can use Sass for more advanced CSS features like nesting. The advantage of CSS modules is that we receive an error in the JavaScript each time a style isn't defined. In the

standard CSS approach, unmatched styles in the JavaScript and CSS files might go unnoticed.



Exercises:

- Confirm the changes from the last section.
- Read more about CSS Modules in create-react-app.