Styling in React

Prior Knowledge

- Be able to import css files into a React project.
- Be able to write css classes.
- Be able to apply classNames in React.

Learning Objectives

- Understand how to build websites with a mobile-first approach.
- Understand how to use media-queries to alter css classes.
- Understand the BEM (block element modifier) naming convention.
- Understand how to scope classes with css modules.
- Understand how to dynamically change classNames in React.
- Understand when UI libraries are appropriate to use.

Mobile First Design

When designing the style of our website we have to consider the various different screen sizes that our sites will be viewed on. The user could be on a laptop, mobile, tablet, etc and these devices come in a very wide variety of screen sizes.

A common approach when dealing with this is to design `mobile-first`. The principle behind this is that if you design your site around the smallest device it will be used on first, then you make sure that every aspect of the site will be functional and accessible on that small device. As the screen size increases you can them re-arrange the existing elements to take advantage of the increased area.

Contrast this to doing things the other way round. You could design a wonderful site when viewed on a desktop monitor, but when viewing that site on a smaller screen will you be able to fit on all of the elements you created for the desktop view. This is definitely possible, but much harder to implement and runs the risk of considering mobile screen sizes as something extra, when they are a huge part of todays market.

In order to check the current screen size in CSS, we can use a media query to apply new styles.

Consider the example below. On a mobile screen (less than 600px wide) the App class displays it's content in a single column and 3 rows.

When the screen size is at least 600px then additional styles are applied from the media query. This changes the layout to be 2 columns, one 25% of the screens width, the second 75%. The rows are reduced from 3 to 2 with the `nav` area in the left hand column and `content` in the right whilst the `header` takes up the entire first row.

```
.App {
  display: grid;
  grid-template-columns: 100%;
  grid-template-rows: min-content auto auto;
  grid-template-areas:
    'header'
    'nav'
    'content';
}

@media only screen and (min-width: 600px) {
  .App {
    grid-template-columns: 25% 75%;
    grid-template-rows: min-content auto;
    grid-template-areas:
      'header header'
      'nav content';
  }
}
```

This example uses grid but we can change any styles we like. For more information on the specific grid properties used check out this excellent guide by CSS Tricks.

BEM (Block__Element--Modifier)

We can import css files directly into React and webpack will bundle them as part of our app to be applied. As css files can get quite long a common approach is to give each component it's own css file and import them to the relevant components.

```
// App.js
import './App.css';

// Nav.js
import './Nav.css';
```

CSS styles are **global** however and any styles we define in one file are available in every other. This is where a good naming convention to prevent us duplicating or writing conflicting styles comes in handy.

One widely adopted convention is BEM, or 'Block Element Modifier'.

This defines 3 distinct parts of each className:

1. **Block** A standalone entity that is meaningful on it's own. In React components are often considered blocks.

e.g. `header`, `nav`, `input`, `button`

2. **Element** A part of a block that is semantically tied to it's block.

e.g. `nav__link`, `header__username`

3. **Modifier** A flag to change the appearance of a block or element.

e.g. `button--disabled`, `nav__link--active`, `input--big`

```
<form className="form">
  <input type="text" className="input form__input" />
  <button
    className={`button form__button ${
      isFormValidated ? 'form__button--active' : 'form__button--inactive'
    }`}
  >
    Place my order!
  </button>
</form>
```

This keeps our classes organised and consistent which is invaluable when working in larger teams or on more complicated projects.

CSS modules

When we import a css file those styles are available everywhere, not just the component we imported the file into.

A nice solution to this are CSS modules.

This is a small library that is included in `create-react-app` by default and as such we can use it straight out of the box in our React projects.

Instead of importing CSS files directly, extend your file name with `.module` and import the default export from that file.

```
/* Header.module.css */
.header {
  /* some lovely styles */
}

.companyLogo {
  /* some lovely styles */
}

.companyName {
  /* some lovely styles */
}

.contactButton {
  /* some lovely styles */
}
```

```
import styles from './Header.module.css'; // object with keys of header, companyLogo, etc...
```

The imported styles object will have a key for each css class. We can then apply these as the `className` for elements in our component. Each class on the object will be suffixed with a unique string which acts as an `id` for the module. This way the classes are guaranteed to be unique and we don't have to worry about repeating class names in each of our css files.

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

const Header = () => {
  return (
    <header className={styles.header}>
      <h1 className={styles.companyName}>Not Fake Incorporated</h1>
      <img src="https://example.com/logo.jpg" className={styles.companyLogo}/>
    </header>
  );
};
```

Dynamic classNames in React

Sometimes we will want to dynamically apply styles to our elements based on some state from React. As our classes are applied through the `className` prop we can update that prop as necessary to apply different styles. As the className prop is a string we can generate that string using Javascript, string templates are very useful here, and use that as our className.

Consider the component below that renders a list of pokemon cards. (In reality the state would be fetched from an api with an effect but has been hard coded here for the sake of example.)

```
const PokeList = () => {
  const [pokemon, setPokemon] = useState([
    {
      name: 'bulbasaur',
      sprite:
        'https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/1.png',
    },
    // etc..
  ]);
  const [highlightedName, setHighlightedName] = useState('bulbasaur');

  return (
    <ul className="PokeList">
      {pokemon.map(({ name, sprite }) => (
        <li
          key={name}
          className={`PokeCard ${
            name === highlightedName ? 'PokeCard--highlight' : ''
          }`}
          onClick={() => setHighlightedName(name)}
        >
          <img src={sprite} alt={name}></img>
        </li>
      ))}
    </ul>
  );
};
```

```
.PokeCard {
  background-color: white;
  margin: 1em;
  padding: 1em;
  cursor: pointer;
}

.PokeCard--highlight {
  background-color: rgb(252, 242, 203);
  transition: 200ms;
}
```

When assigning the className the component uses a ternary to check each `name` against the `highlightedName`. If they match the className becomes `PokeCard PokeCard--highlight` and if not, just `PokeCard`.

This fits well with our `BEM modifier` but the same principle applies when using CSS modules.

```
className={name === highlightedName ? styles.pokeCard : styles.pokeCardHighlight}
```

We can use any condition we like, and if the logic for working out the classNames becomes complicated then it can always be extracted to a util function and built with TDD.

React UI Libraries

There are a number of CSS libraries available that can provide us with pre-defined styles. A lot of the larger libraries offer react specific versions that come in the form of pre-built components with all the styles and functionality already built. For getting projects up and running quickly they can be very beneficial but abstract a lot of the functionality for you.

Some of the more popular libraries are:

- MUI (formerly known as Material UI)
- React Materialize
- Grommet
- React Bootstrap
- Bulma