How to CSS Style in React

MAY 10, 2020 BY ROBIN WIERUCH - EDIT THIS POST

In modern React, there are many ways to style a React application with CSS. Whenever I do a React workshop with aspiring React developers, I show only one of these ways due to the limited time I have for the complete React workshop. But often this one way of styling isn't enough to cover the full scope of this important topic. There are many different strategies (e.g. CSS-in-JS) and many different approaches within these strategies (e.g. Styled Components) to learn about:

- CSS-in-CSS (e.g. CSS, Sass, CSS Modules, or CSS Modules with Sass)
- CSS-in-JS (e.g. Styled Components, Emotion)
- Utility-First-CSS (e.g. Tailwind CSS)

Follow me on this React journey to learn more about these different strategies and approaches in CSS to style your React components. For all of the different ways, we will start with the same React

components:

```
import React from 'react';

function App() {
  const [fruits, setFruits] = React.useState([
    { id: '1', name: 'Apple', isFavorite: false },
    { id: '2', name: 'Peach', isFavorite: true },
    { id: '3', name: 'Strawberry', isFavorite: false },
  ]);

  function handleClick(item) {
    const newFruits = fruits.map((fruit) => {
      if (fruit.id === item.id) {
        return {
          id: fruit.id,
          name: fruit.name,
          isFavorite: !fruit.isFavorite,
        };
      } else {
        return fruit;
      }
    });

    setFruits(newFruits);
  }

  return (
    <div>
      <h3>with no styling</h3>

      <Basket items={fruits} onClick={handleClick} />
    </div>
  );
}

function Basket({ items, onClick }) {
  return (
    <ul>
      {items.map((item) => (
        <li key={item.id}>
          {item.name}
          <button type="button" onClick={() => onClick(item)}>
            {item.isFavorite ? 'Unlike' : 'Like'}
          </button>
        </li>
      ))}
    </ul>
  );
}

export default App;
```

This small React application just renders a list component with a stateful list. A button for each item of the list helps us via a button and its callback handler to like or unlike a list item. In the next steps we will style the button and the list the different CSS styling approaches. We will use a popular folder structure for the React project whenever we have a CSS style file.

TABLE OF CONTENTS

CSS-IN-CSS: CSS IN REACT

The most basic way is to just use vanilla CSS in React with CSS files. Next to each component or to each set of components, you can have a file with the .css extension. For example, the following CSS file defines a CSS class for a button:

```
.button {
  cursor: pointer;
  border: 1px solid #1a202c;
  padding: 8px;
  min-width: 64px;

  background: transparent;

  transition: all 0.1s ease-in;
}

.button:hover {
  background: #1a202c;
  color: #ffffff;
}
```

In the React JavaScript file, we can import the CSS from this style file and implicitly use it:

```
import React from 'react';
import './style.css';
```

```
function App() {
  ...
}

function Basket({ items, onClick }) {
  return (
    <ul>
      {items.map((item) => (
        <li key={item.id}>
          {item.name}
          <button
            type="button"
            className="button"
            onClick={() => onClick(item)}
          >
            {item.isFavorite ? 'Unlike' : 'Like'}
          </button>
        </li>
      ))}
    </ul>
  );
}
```

There is no direct connection -- like a variable -- that let's us define this CSS class in JSX with a className attribute. Instead, by importing the CSS file, all the CSS classes are available here.

Let's continue with styling the list. In the CSS file, we can add two more CSS classes for the list and the list items:

```
.unordered-list {
  margin: 0;
  padding: 0;
  list-style-type: none;
}

.list-item {
  display: flex;
  justify-content: space-between;
  padding: 8px 0;
}
```

Then again we can use them with the CSS className attribute in our React's JSX. Since we imported the CSS file already, we can just use the CSS class right away:

```
function Basket({ items, onClick }) {
  return (
    <ul className="unordered-list">
      {items.map((item) => (
        <li key={item.id} className="list-item">
          {item.name}
```

```
            <button
              type="button"
              className="button"
              onClick={() => onClick(item)}
            >
              {item.isFavorite ? 'Unlike' : 'Like'}
            </button>
          </li>
        ))}
      </ul>
    );
  }
```

There are a few drawbacks of this CSS usage in React. First of all, it's just vanilla CSS and we are missing out a lot of advanced CSS features. We will improve this situation with the next approach called Sass which lives in the same CSS-in-CSS strategy.

CSS-IN-CSS: SASS IN REACT

If you are using create-react-app, you can use Sass after installing it. In contrast, if you are using a custom React with Webpack setup, you need to configure Webpack for it.

Sass (Syntactically Awesome Style Sheets) is a CSS extension that gives you more powerful CSS. For example, you can define reusable CSS variables and you are able to nest your CSS. We will make use of the latter for the button's hover effect:

```
.button {
  cursor: pointer;
  border: 1px solid #1a202c;
  padding: 8px;
  min-width: 64px;

  background: transparent;

  transition: all 0.1s ease-in;

  &:hover {
    background: #1a202c;
    color: #ffffff;
  }
}
```

In vanilla CSS, we had to define another button pseudo hover class. With Sass, we can use the parent selector & which refers to the outer selector (here `.button`). This way, we can neatly nest CSS selectors into each other and reference parents from within these nested selectors.

The new CSS file comes with a Sass file extension. Rename your style file to *style.scss* and import it in your React JavaScript file for further usage:

```
import React from 'react';
import './style.scss';

...
```

All the other styling and usage remains the same like before -- when we used vanilla CSS --, because we are not using any other Sass features here. Just keep in mind that whenever you are using a CSS-in-CSS strategy, make sure to opt-in a CSS extension like Sass to give yourself more features (nested CSS, variables, and special selectors like the parent selector) when using CSS.

There is still another drawback of using CSS -- even with Sass -- in React this way: All the CSS is globally accessible after importing it. Somewhere else in your React project you could reuse the CSS classes for the button, list and list item. Sometimes this may be the desired effect, but most of the times you want to scope your styles/CSS to one JavaScript file or one React component. Let's enter CSS Modules ...

CSS-IN-CSS: CSS MODULES IN REACT

If you are using create-react-app, you can use CSS Modules right away. However, if you are using a custom React with Webpack setup, you need to configure Webpack for it.

CSS Modules can be used with vanilla CSS but also with CSS extensions like Sass. Let's see how a CSS module can be defined in a *style.module.css* (vanilla CSS) or *style.module.scss* file (Sass):

```
.button {
  cursor: pointer;
  border: 1px solid #1a202c;
  padding: 8px;
  min-width: 64px;

  background: transparent;

  transition: all 0.1s ease-in;
}

.button:hover {
  background: #1a202c;
  color: #ffffff;
}
```

If you are using Sass with CSS Modules, you can use the all Sass features like the & parent selector again:

```
.button {
  cursor: pointer;
  border: 1px solid #1a202c;
  padding: 8px;
  min-width: 64px;

  background: transparent;

  transition: all 0.1s ease-in;

  &:hover {
    background: #1a202c;
    color: #ffffff;
  }
}
```

In your React JavaScript file, you can import the *style.module.css* or *style.module.scss* file again, but this time it's an explicit import with an JavaScript style object:



```
import React from 'react';
import styles from './style.module.css';

...
```

If you are using Sass, use the *.scss* instead of the *.css* file extension. This new JavaScript style object, which is nothing else than a regular JavaScript object, holds all your styles from your CSS file. You can use it in your React component's JSX:

```
function Basket({ items, onClick }) {
  return (
    <ul>
      {items.map((item) => (
        <li key={item.id}>
          {item.name}
          <button
            type="button"
            className={styles.button}
            onClick={() => onClick(item)}
          >
            {item.isFavorite ? 'Unlike' : 'Like'}
          </button>
        </li>
      ))}
    </ul>
  );
```

```
}
```

The CSS class is available as property on the imported style object. We can do the same for the list and list item classes. First define them in your CSS file:

```
.unordered-list {
  margin: 0;
  padding: 0;
  list-style-type: none;
}

.list-item {
  display: flex;
  justify-content: space-between;
  padding: 8px 0;
}
```

Since both CSS classes are already imported due to the previous button usage, we can use them straight away in React's JSX:

```
function Basket({ items, onClick }) {
  return (
    <ul className={styles['unordered-list']}>
      {items.map((item) => (
        <li key={item.id} className={styles['list-item']}>
          {item.name}
          <button
            type="button"
            className={styles.button}
            onClick={() => onClick(item)}
          >
            {item.isFavorite ? 'Unlike' : 'Like'}
          </button>
        </li>
      ))}
    </ul>
  );
}
```

CSS classes are usually defined in kebab-case. In case of the button style, you can retrieve it with `styles.button`. However, for the other styles with dashes you need to retrieve them with strings from the object.

In conclusion, CSS Modules with an extension like Sass are the status quo in modern React if you want to use CSS-in-CSS as styling strategy. If you want to use CSS-in-JS instead, you would choose something like Styled Components.

CSS-IN-JS: STYLED COMPONENTS

There is CSS setup needed for styled components, because everything comes with JavaScript. Essentially as the strategy CSS-in-JS already says, we will not need any CSS file, because all CSS is defined in JavaScript. Before you can use Styled Components you need to install them on the command line:

```
npm install styled-components
```

Styled Components takes the approach to create components just from a HTML tag and a style string. Let's see how this looks for a button element which becomes a Button component in our JavaScript file:

```
import React from 'react';
import styled from 'styled-components';

const Button = styled.button`
  cursor: pointer;
  border: 1px solid #1a202c;
  padding: 8px;
  min-width: 64px;

  background: transparent;

  transition: all 0.1s ease-in;

  &:hover {
    background: #1a202c;
    color: #ffffff;
  }
`;
```

The Button variable is a valid React component which can be used in JSX. Any properties like the `onClick` are passed through to the real button HTML element. In addition, a styled component already comes features (here: CSS nesting with parent selector) which we would usually gain from a CSS extension like Sass.

```
function Basket({ items, onClick }) {
  return (
    <ul>
      {items.map((item) => (
        <li key={item.id}>
          {item.name}
          <Button type="button" onClick={() => onClick(item)}>
            {item.isFavorite ? 'Unlike' : 'Like'}
```

```
            </Button>
          </li>
        ))}
      </ul>
    );
  }
```

The syntax of Styled Components isn't very clear for many React beginners. Basically the `styled` object offers you a function for each HTML element (e.g. button, ul, li). The function can be called with JavaScript template literals whereas everything you place into the template literals becomes the style of the component:

```
const UnorderedList = styled.ul`
  margin: 0;
  padding: 0;
  list-style-type: none;
`;

const ListItem = styled.li`
  display: flex;
  justify-content: space-between;
  padding: 8px 0;
`;
```

The styled components can be defined in the same file or somewhere else. After all, they are just regular React components after you have defined them, which makes them exportable or directly usable in your JSX:

```
function Basket({ items, onClick }) {
  return (
    <UnorderedList>
      {items.map((item) => (
        <ListItem key={item.id}>
          {item.name}
          <Button type="button" onClick={() => onClick(item)}>
            {item.isFavorite ? 'Unlike' : 'Like'}
          </Button>
        </ListItem>
      ))}
    </UnorderedList>
  );
}
```

With a CSS-in-JS approach like Styled Components you still need to write CSS, but you write it in JavaScript. In addition, a library like Styled Components already solves many problems that we had to solve with CSS Modules (scoping) and Sass (CSS features) previously.

UTILITY-FIRST-CSS: TAILWIND CSS

Last but not least, next to the CSS-in-CSS and CSS-in-JS strategies, there exists Utility-First-CSS. One of the approaches for Utility-First-CSS is Tailwind CSS. Let's see how this looks like after you have set it up. Note that Tailwind CSS needs some proper setup (in React) before you can use it. Check out the official Tailwind CSS website for instructions. Afterward, you can import Tailwind CSS for your React components:

```
import React from 'react';

import '../tailwind.generated.css';

...
```

When using a Utility-First-CSS strategy with something like Tailwind CSS, you don't need to define your CSS anymore. Tailwind CSS gives you all the preconfigured CSS that you can use right away in your React's classNames. Let's see how this looks like for our button example:

```
function Basket({ items, onClick }) {
  return (
    <ul>
      {items.map((item) => (
        <li key={item.id}>
          {item.name}
          <button
            type="button"
            className="p-2 w-16 border border-solid border-gray-900 transiti
            onClick={() => onClick(item)}
          >
            {item.isFavorite ? 'Unlike' : 'Like'}
          </button>
        </li>
      ))}
    </ul>
  );
}
```

Tailwind CSS comes with preconfigured CSS classes. For example, the `p-2` class gives us a padding to all directions of 0.5rem which -- if nothing else is configured -- translates usually to 8px. It's also possible to use the selectors of pseudo classes (here hover) directly in your JSX className attribute.

The bad part about Tailwind CSS is that you cannot directly apply your CSS knowledge anymore, because you have to learn their syntax for expressing all the CSS properties like width (here w-16) or color (border-gray-900). However, once you have learned Tailwind CSS' available properties (or at

least know how to browser their documentation), you may will find yourself faster than ever developing React components with CSS. Rather than knowing about all the possible key/value pairs from CSS, you can almost just use the value right away in your JSX. In addition, Tailwind CSS comes with lots of sensible defaults like colors or padding/margins which will automatically lead to a better looking application.

Let's see how we can style the list and list item elements with Tailwind CSS:

```
function Basket({ items, onClick }) {
  return (
    <ul>
      {items.map((item) => (
        <li key={item.id} className="flex justify-between py-2">
          {item.name}
          <button
            type="button"
            className="p-2 w-16 border border-solid border-gray-900 transiti
            onClick={() => onClick(item)}
          >
            {item.isFavorite ? 'Unlike' : 'Like'}
          </button>
        </li>
      ))}
    </ul>
  );
}
```

The list item element just receives CSS values for its flexbox styling and a vertical padding for top and bottom. The list itself doesn't receive any CSS classes, because it already looks great with the Tailwind CSS defaults which remove the CSS list style decoration and the margin/padding.

Tailwind CSS is great for solo developers or teams who are willing to learn Tailwind CSS' classes for the sake of a faster development process, because they don't need to define the CSS themselves anymore.

INLINE CSS

Inline CSS (also called inline style) is a little bonus on top, because it shouldn't replace any of the other shown CSS approaches. However, sometimes it's useful to know about it for rapid prototyping or for more dynamic CSS driven by JavaScript. For example, every HTML element comes with a style attribute. You can use the style attribute in React's JSX to pass a style object to it:

```
function Basket({ items, onClick }) {
```

```
  return (
    <ul>
      {items.map((item) => (
        <li key={item.id}>
          {item.name}
          <button
            type="button"
            onClick={() => onClick(item)}
            style={{
              cursor: 'pointer',
              border: '1px solid #1a202c',
              padding: '8px',
              minWidth: '64px',

              background: 'transparent',

              transition: 'all 0.1s ease-in',
            }}
          >
            {item.isFavorite ? 'Unlike' : 'Like'}
          </button>
        </li>
      ))}
    </ul>
  );
}
```

We don't need to define any other styled component or CSS file, because we can directly pass all the style as object to the JSX's HTML element. The same can apply the the list and list item elements:

```
function Basket({ items, onClick }) {
  return (
    <ul
      style={{
        margin: '0',
        padding: '0',
        listStyleType: 'none',
      }}
    >
      {items.map((item) => (
        <li
          key={item.id}
          style={{
            display: 'flex',
            justifyContent: 'space-between',
            padding: '8px 0',
          }}
        >
          {item.name}
          <button
            type="button"
            onClick={() => onClick(item)}
            style={{
```

```
              cursor: 'pointer',
              border: '1px solid #1a202c',
              padding: '8px',
              minWidth: '64px',

              background: 'transparent',

              transition: 'all 0.1s ease-in',
            }}
          >
            {item.isFavorite ? 'Unlike' : 'Like'}
          </button>
        </li>
      ))}
    </ul>
  );
}
```

You can already see the negative impacts of this approach: your JSX becomes unreadable, because all the style is cluttered within your HTML tags. That's why you will only rarely see inline styles in regular React projects. However, as mentioned before, it can come in handy for prototyping or for dynamic CSS based on JavaScript conditions.

. . .

After all, personal taste and features influence the decision of which styling strategy and approach to take for you and your team's React project. In modern React applications you will find the most popular approaches of every strategy: CSS Modules, Styled Components, and Tailwind CSS. You can find all the different approaches within the styling strategies in this GitHub repository.

Show Comments

KEEP READING ABOUT REACT›

HOW TO USE CSS MODULES IN CREATE-REACT-APP?

The article is a short how to use CSS Modules in your create-react-app application . It shows you how to setup CSS Modules, but also how to use them in your components.

After you have setup your...

HOW TO USE CSS MODULES IN REACT?

CSS Modules are one of the most popular ways for styling React components. Whether you are using only CSS or a more advanced pre-processor like SASS, it doesn't matter for CSS Modules: You can write...

PORTFOLIO

Online Courses

Open Source

Tutorials

ABOUT

About me

What I use

How to work with me

How to support me