Styling React Components

This lesson introduces some common methods used to style React components.

WE'LL COVER THE FOLLOWING

React is a view layer. It kind of controls the markup rendered in the browser. And we know that the styling with CSS is tightly connected to the markup on the page. There are couple of approaches for styling React applications and in this section we will go through the most popular ones.

The good old CSS class

JSX syntax is pretty close to HTML syntax. As such we have almost the same tag attributes and we may still style using CSS classes. Classes which are defined in an external `.css` file. The only caveat is using `className` and not `class`. For example:

```
<h1 className='title'>Styling</h1>
```

Inline styling

The inline styling works just fine. Similarly to HTML we are free to pass styles directly via a `style` attribute. However, while in HTML the value is a string, in JSX it must be an object.

```
const inlineStyles = {
```

```
  color: 'red',
  fontSize: '10px',
  marginTop: '2em',

  'border-top': 'solid 1px #000'
};

<h2 style={ inlineStyles }>Inline styling</h2>
```

Since we write the styles in JavaScript we have some limitations from a syntax point of view. If we want to keep the original CSS property names we have to put them in quotes. If not then we have to follow the camel case convention. However, writing styles in JavaScript is quite interesting and may be a lot more flexible than the plain CSS. Like for example inheriting of styles:

```
const theme = {
  fontFamily: 'Georgia',
  color: 'blue'
};
const paragraphText = {
  ...theme,
  fontSize: '20px'
};
```

We have some basic styles in `theme` and we mix them with what is in `paragraphText`. Shortly, we are able to use the whole power of JavaScript to organize our CSS. What matters at the end is that we generate an object that goes to the `style` attribute.

CSS modules

CSS modules is building on top of what we said so far. If we don't like the JavaScript syntax then we may use CSS modules and we will be able to write plain CSS. Usually this library plays its role at bundling time. It is possible to hook it as part of the transpilation step but normally is distributed as a build system plugin.

Here is a quick example to get an idea how it works:

```
/* style.css */
.title {
  color: green;
}

// App.jsx
import styles from "./style.css";
```

```
function App() {
  return <h1 style={ styles.title }>Hello world</h1>;
}
```

That is not possible by default but with CSS modules we may import directly a plain CSS file and use the classes inside.

And when we say *plain CSS* we don't mean that it is exactly like the normal CSS. It supports some really helpful composition techniques. For example:

```
.title {
  composes: mainColor from "./brand-colors.css";
}
```

Styled-components

Styled-components took another direction. Instead of inlining styles the library provides a React component. We then use this component to represent a specific look and feel. For example, we may create a `Link` component that has certain styling and use that instead of the `<a>` tag.

```
const Link = styled.a`
  text-decoration: none;
  padding: 4px;
  border: solid 1px #999;
  color: black;
`;

<Link href='http://google.com'>Google</Link>
```

There is again a mechanism for extending classes. We may still use the `Link` component but change the text color like so:

```
const AnotherLink = styled(Link)`
  color: blue;
`;

<AnotherLink href='http://facebook.com'>Facebook</AnotherLink>
```

For me styled-components are probably by far the most interesting approach for styling in React. It is quite easy to create components for everything and forget about the styling. If your company has the capacity to create a design system and build a product with it then this option is probably the most suitable one.

Final thoughts

There are multiple ways to style your React application, there is no right or wrong way. As is the case with most things in JavaScript today, you have to pick the one that fits better in your context.

Here is the final code for all the different methods we learned to style our components in:

```
import React from 'react';
import ReactDOM from 'react-dom';

const inlineStyles = {
  color: 'red',
  fontSize: '10px',
  marginTop: '2em',
  'border-top': 'solid 1px #000'
};

const theme = {
  fontFamily: 'Georgia',
  color: 'blue'
};
const paragraphText = {
  ...theme,
  fontSize: '20px'
};

function App() {
  return (
    <section>
      <h1 className='title'>Styling</h1>
      <h2 style={inlineStyles}>Inline styling</h2>
      <p style={paragraphText}>This is a paragraph here!</p>
    </section>
  );
}

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

Now that we have learned multiple ways to style React components, in the next lesson we will move forward to look into integrating third-party libraries within our applications.