Style React Components 40th Day

React components are the building blocks of our applications' UI in 2020. As visual elements, styling them is a big part of how applications actually meet our users, and composes the way our brand and product looks and feels.
Choosing the right method for styling components isn't an absolute truth. It's a relative choice that should serve your use case, personal preferences and above all architectural goals of the way you work: Global namespacing, dependencies, reusability, scalability, dead–code elimination and so on.

It is important to note that:

As React components become the building blocks of our application's UI, styling them becomes a critical part of our development workflow.

When scaling our codebase, our styling method is the key to balancing modularity and avoiding common scaling issues which can compromise our development velocity.

Remember that down the line, there is no right or wrong way to style components. It really depends on how you work, the tools you use and on understanding what it is you're really trying to achieve. Hope you find this useful, and please feel free to comment and suggest your own insights.]

The choiceof the method to use is yours, and please feel free to add your own experience, insights, and suggestions.

Here will will just discuss 5 methods but that is not exhaustively.

1). **Inline CSS**

React lets you add CSS inline, written as attributes and passed to elements.

In React, inline styles are not specified as a string. Instead they are specified with an object whose key is the camelCased version of the style name, and whose value is the style's value, usually a string.

The style attribute accepts a JavaScript object with camelCased properties rather than a CSS string. This is consistent with the DOM style JavaScript property, is more efficient, and prevents XSS security holes

Example:

```
const divStyle = {
  color: 'blue',
  backgroundImage: 'url(' + imgUrl + ')',
};

function HelloWorldComponent() {
  return <div style={divStyle}>Hello World!</div>;
```

Or :

```
function HelloWorldComponent() {
  return <div style={{color: “blue”}}>Hello World!</div>;
```

You can pass the styling directly or create a variable that stores style properties and then pass it to the element. With inline styles, you also have the option to combine CSS syntax with JSX code (a preprocessor step that adds XML syntax to JavaScript).

This approach makes it easier to colocate styles in components and understanding what styles each component has. Leveraging the abilities of JS also makes it easier to work with more complex styling when needed.

2. **CSS in JS**

A particularly interesting concept is using **CSS in JS** to abstract CSS to the component level itself, using JavaScript to describe styles in a declarative and maintainable way. With the release of the popular styled-components project by Max Stoiber, this concept is more mainstream today than ever

So what's the difference between CSS-in-JS and Inline styles? let's take it to the ground level. Here's a simple explanation that makes things clearer at the most down-to-earth level possible.

Link:

https://hackernoon.com/all-you-need-to-know-about-css-in-js-984a72d48ebc

In React, CSS-in-JS lets you think and design styles in abstraction in the component level, leveraging the principles of modularity and isolation, unit-testing, DRY principle and so on. It's a mind-bender, but often a practical one.

Note that the official React docs still state that "React does not have an opinion about how styles are defined; if in doubt, a good starting point is to define your styles in a separate *.css file as usual and refer to them using className."

3). **Styled Components**

While effectively a CSS-in-JS library, styled components deserves a position of its own. Take a look at some of these UI libraries using styled-components.

Link: https://blog.bitsrc.io/9-react-styled-components-ui-libraries-for-2018-4e1a0bd3e179

An idea born in an Australian Whisky bar turned into an 20K stars project. This project makes it easier to use CSS in React components, by defining styled-components with encapsulated styles without CSS classes as a mediator layer.

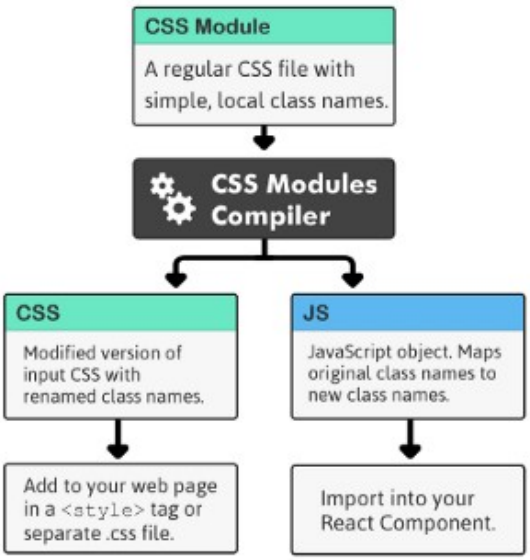
Styled-components are created by defining components using the ES6 template literal notation. CSS properties can be added to the component just like you would do using CSS. When the JS is parsed, styled-components will generate unique class names, and inject the CSS into the DOM.

Find more about styled component:

https://github.com/styled-components/styled-components

4). **CSS Modules**

A CSS Module is a CSS file in which all class names and animation names are scoped locally– by default.



In React, each React component gets its own CSS file, which is scoped to that file and component. For a React component that you'd like to style, simply create a CSS file that'll contain the styles for that component.

At build time local class names are mapped and exported as a JS object literal for React– as well as a modified version of input CSS with renamed class names. The result is, you don't have to mess as much with global styles. When scaling your projects, you have less overrides and trouble on your hands.

Processors: Sass, SCC and Less

Sass is a preprocessor scripting language that is interpreted or compiled into Cascading Style Sheets (CSS). The original syntax uses a syntax similar to Haml and uses indentation to separate code blocks and newline characters to separate rules.

Find more about it and try out some examples.

5). **Stylable**

Much like CSS modules, Stylable was built for SCC styling in large projects with components while tackling scaling issues without running to CSS in JS. It enables you to write reusable, highly-performant components. Each component exposes a style API that maps its internal parts so you can reuse components across teams without sacrificing component "stylability".

Stylable scopes styles to components so they don't "leak" and clash with other styles. It enables custom pseudo-classes and pseudo-elements that abstract the internal state and structure of a component. These can then be styled externally. For example, you can style the label inside a button, or style the play button of a video player from outside these components.

At build time, the preprocessor converts the Stylable CSS into flat, static, valid, vanilla CSS that works cross-browser. Here's a useful review and demo.

Base your styling method based on your app architecture.

More resources: https://www.w3schools.com/react/react_css.asp

Best wishes. Lux Tech Academy