Styled Components Cheat Sheet

Styled Components is a library for React & React Native to write and manage your CSS.

It's an extremely popular solution for managing CSS in React, with around 8 million npm downloads/month and 30k stars in Github.

A familiarity and understanding of React is recommended before diving into Styled Components.

Benefits of styled components:

- It's plain CSS. Yes, you're writing the CSS in a JS file, but the CSS syntax is unchanged.
- Vendor prefixes are automatically added when using Styled Components, improving performance across browsers.
- All unused CSS and styling gets removed automatically
- You don't write any class names, whatsoever. Class names are generated automatically, so there's no need to manage a CSS class naming methodology like BEM.

Installing styled-components

To get started with styled-components, you first need to install it into your project:

```
npm install styled-components
```

And on every file that you use styled-components, you'll need to add this import:

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

Building your first styled component

Inside a React component, import styled and add the following code:

```
// App.js
import React from 'react';
import styled from 'styled-components';

// Button component that'll render an <a> tag with some styles
const Button = styled.a`
  background-colour: teal;
  color: white;
  padding: 1rem 2rem;
`

const App = () => {
  return (
    <Button>I am a button</Button>
  )
}

export default App;
```

Understand the code:

- Just like writing a React functional component, declare the name of the component with `const Button`
- importing `styled` gives us the Styled Components functionality
- Notice the `a` after styled? This represents the anchor HTML element: <a>. When declaring a Styled Component, you can use any HTML element here (e.g. <div>, <h1>, <section> etc.)

You can also create styled components as standalone component files:

```
// Button.js
import styled from 'styled-components';

const Button = styled.a`
  background-colour: teal;
  color: white;
  padding: 1rem 2rem;
`

export default Button;
```

Understand the code:

- Importing React is not required here, as you're not writing any JSX
- Make sure you export the component so it can now be used in other components.

Using props to customize Styled Components

Imagine you have a <Button /> component, and you need to style different variants of that button (primary, secondary, danger, etc).

Styled Components have an elegant solution for this, where you leverage props to make your component styles dynamic:

```
// App.js
import React from 'react';
import styled from 'styled-components';
import Button from './Button';

const App = () => {
  return (
    <>
      <Button>I am a button</Button>
      <Button primary>I am a primary button</Button>
    </>
  )
}

export default App;
```

Understand the code:

- Here, I've imported the Button components (which is a Styled Component)
- Notice how one of the <Button /> component passes a 'primary' prop. We'll use this to make the Button styles dynamic

And now, inside our <Button /> component, we can add the dynamic styles:

```
// Button.js
import styled from 'styled-components';

const Button = styled.a`
  display: inline-block;
  border-radius: 3px;
  padding: 0.5rem 0;
  margin: 0.5rem 1rem;
  width: 11rem;
  border: 2px solid white;
  background: ${props => props.primary ? 'white' : 'palevioletred' }
  color: ${props => props.primary ? 'palevioletred' : 'white' }
`

export default Button;
```

Understand the code:

- What's happening here is you're interpolating a function that is returning a CSS value (using a ternary operator) based on the props.
- To put it simply: you're conditionally changing the CSS values based on the props passed.

Handling props like this works in some use-cases, but it can get messy if you have multiple props (e.g. primary, secondary, danger, etc.) as well as multiple lines of CSS.

Often, it makes more sense to import { css } from styled-components like this:

```
// Button.js
import styled, { css } from 'styled-components';

const Button = styled.a`
  display: inline-block;
  border-radius: 3px;
  padding: 0.5rem 0;
  margin: 0.5rem 1rem;
  width: 11rem;
  background: transparent;
  color: white;
  border: 2px solid white;

  ${props => props.primary && css`
    background: white;
    color: palevioletred;
  `}
`

export default Button;
```

Understand the code:

- Notice how { css } is imported from styled-components. This is required to write multiple lines of CSS
- Writing conditional styles in this way keeps your dynamic srtles separate for different props.

Using media-queries to make your styled-components responsive

Thankfully, making your Styled Components responsive is super simple.

Add media queries inside your template literal, like this:

```
// Button.js
import styled from 'styled-components';

const Button = styled.a`
  display: inline-block;
  border-radius: 3px;
  padding: 0.5rem 0;
  margin: 0.5rem 1rem;
  width: 9rem;
  background: transparent;
  color: white;
  border: 2px solid white;

  @media (min-width: 768px) {
    padding: 1rem 2rem;
    width: 11rem;
  }

  @media (min-width: 1024px) {
    padding: 1.5rem 2.5rem;
    width: 13rem;
  }
`

export default Button;
```

Understand the code:

- To add media queries, simply nest the media query inside the template literal strings.

Handling hover states and other pseudo-selectors with Styled Components

Similarly to adding media queries to your Styled Components, adding pseudo selectors is pretty straightforward.

For example, adding a hover state to our <Button /> component would look like this:

```
// Button.js
import styled from 'styled-components';

const Button = styled.a`
  display: inline-block;
  border-radius: 3px;
  padding: 0.5rem 0;
  margin: 0.5rem 1rem;
  width: 9rem;
  background: transparent;
  color: white;
  border: 2px solid white;

  :hover {
    border-color: green;
  }
`

export default Button;
```

Understand the code:

- To add psuedo-selectors, simply nest pseudo-selectors inside your template literal strings

Creating global styles

You might want to set global styles, like:

- Set a font-family for all your typography
- Set the background color on every page
- Override some browser default styling

Styled Components has a solution for global styles using the createGlobalStyle function.

First, navigate to the component which is at the top of your React tree.

```
// App.js
import React from 'react';
import styled, { createGlobalStyle } from 'styled-components';
import { Container, Nav, Content } from '../components';

const GlobalStyle = createGlobalStyle`
  body {
    margin: 0;
    padding: 0;
    background: teal;
    font-family: Open-Sans, Helvetica, Sans-Serif;
  }
`;

const App = () => {
  return (
    <Container>
      <Nav />
      <Content />
    </Container>
  )
}

export default App;
```

Understand the code:

- You need to apply global styles at the top of your React tree, so find your component at the top
- You'll need to import 'createGlobalStyle' into your project
- Create a component for your global styles (in our example, the component is <GlobalStyle />)

This isn't going to apply the styles to the project yet.

Now we need to use the GlobalStyle component to apply the global styles to the application.

```
// App.js
import React from 'react';
import styled, { createGlobalStyle } from 'styled-components';
import { Container, Nav, Content } from '../components';

const GlobalStyle = createGlobalStyle`
  body {
    margin: 0;
    padding: 0;
    background: teal;
    font-family: Open-Sans, Helvetica, Sans-Serif;
  }
`;

const App = () => {
  return (
    <>
      <GlobalStyle />
      <Container>
        <Nav />
        <Content />
      </Container>
    </>
  )
}

export default App;
```

Understand the code:

- Add your <GlobalStyle /> component at the top of your React tree
- You'll need to add a React fragment around your <GlobalStyle /> and it's sibling components