Styling in React

Inline Styling

To style an element with the inline style attribute, the value must be a JavaScript object:

```
function MyComponent(){
        return <div style={{ color: 'blue', lineHeight : 10, padding: 20 }}> Inline Styled
Component</div>
}
```

Notice that the value of padding does not have a unit as React appends a 'px' suffix to some numeric inline style properties. In cases where you need to use other units, such as 'em' or 'rem', specify the unit with the value explicitly as a string. Applying that to the padding property should result in **padding: '1.5em'**.

Style attribute

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. For example:

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

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

Importing css file/stylesheet in react

Create a Button.css file and add write your css there. For e.g. inside the style.css write:

```
.Button {
  padding: 20px;
}
```

Now inside the JS file you can import this css file using.

```
import React, { Component } from 'react';
import './Button.css'; // Tell webpack that Button.js uses these styles

class Button extends Component {
  render() {
    // You can use them as regular CSS styles
    return <div className="Button" />;
  }
}
```

Adding a CSS Modules Stylesheet

In the React application, we usually create a single .css file and import it to the main file so the CSS will be applied to all the components. In the large project there can be name collision between two css files. But using CSS modules helps to create separate CSS files for each component and is local to that particular file and **avoids class name collision**.

A css module class is created using the [name].module.css file naming convention. CSS Modules allows the scoping of CSS by automatically creating a unique classname of the format [filename]_[classname]__[hash].

Step 1:- Create Button.module.css

```
.error {
  background-color: red;
}
```

Step 2:- inside the Button.js import like this

```
import React, { Component } from 'react';
import styles from './Button.module.css'; // Import css modules stylesheet as styles
import './another-stylesheet.css'; // Import regular stylesheet

class Button extends Component {
  render() {
    // reference as a js object
    return <button className={styles.error}>Error Button</button>;
  }
}
```

Result:

```
<!-- This button has red background but not red text -->
<button class="Button_error_ax7yz">Error Button</button>
```

Styled Components

Styled-components is a library built for React and React Native developers. It allows you to use component-level styles in your applications. Styled-components leverage a mixture of JavaScript and CSS using a technique called CSS-in-JS.

Styled-components are based on tagged template literals, meaning actual CSS code is written between backticks when styling your components. This gives developers the flexibility of reusing their CSS code from one project to another.

With styled-components, there is no need to map your created components to external CSS styles.

Advantages of using Styled-components

Below are some of benefits of using styled-components:

- Eliminates class name bugs: styled-components provide unique class names for your styles, thus eliminating the problems with class names duplications, misspellings, and overlaps.
- Easier management of CSS: With every bit of styling tied to a specific component, it is easier to know which CSS is applied This makes it easy to delete unused component styles.
- Simple and dynamic styling: Through props and global themes supported in styled-components, styling is simple without manually managing dozens of classes.
- Reproducible styles: When you style with styled-components, you can import your styles into other project areas no matter how big or small your codebase is.

Installation:

```
npm install styled-components
```

Import:

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

Open up an existing React project you're working on, and open up one of your existing components.

Here, you can add your first Styled Component.

Now that you have styled imported, here's how you get started:

```
// 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;
```

Let’s break this down:

- Just like writing a React functional component, declare the name of the component with const Button
- styled is what we imported above, and 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 in a separate file and import them into the current file.

Dynamic styling with styled-components

The point of dynamic styling is saving time and writing less CSS.

Imagine that you have a **primary** and a **secondary** button. They are very similar, but you want your primary button to have a flashy color so people actually click on it.

You can do that by adding a primary attribute to your <Button /> and handling that new attribute on your styled component, exactly like you would with component props!

```
export default function App() {
  return (
    <Wrapper>
      <Button>Hello, I am a Button</Button>
      <Button primary>Hello, I am a Primary Button</Button>
    </Wrapper>
  );
}
```

```
const Button = styled.button`
  background: ${props => props.primary ? "#6495ED" : "#2b2b2b"};
  color: white;
  font-size: 24px;
  padding: 12px;
  cursor: pointer;
`;
```

It may happen that your buttons need to be extra flexible, while still sharing some basic styles. Maybe you need to have a **primary** button that also has **rounded corners** and a fancy **box-shadow**. For such case you can "lift" the props and do something like that:

```
const Button = styled.button(
  ({ primary, round, shadow }) => `
    background: ${primary ? "#6495ED" : "#2b2b2b"};
    border-radius: ${round ? "4px" : "0"};
    box-shadow: ${shadow ? "2px 2px 2px rgba(0, 0, 0, 0.5)" : "none"};
    color: white;
    font-size: 24px;
    padding: 12px;
    cursor: pointer;
  `
);
```

Radium

Radium is a popular third package application used to add inline styling and pseudo selectors such as **:hover, :focus, :active**, etc. to a JSX element.

Installation:

```
npm install --save radium
```

Usage:

Start by wrapping your component class with Radium(), like export default Radium(Component), or Component = Radium(Component), which works with classes, createClass, and stateless components (functions that take props and return a ReactElement). Then, write a style object as you normally would with inline styles, and add in styles for interactive states and media queries. Pass the style object to your component via style={...} and let Radium do the rest!

Example

```
import Radium from 'radium';
import React from 'react';
import color from 'color';

class Button extends React.Component {
  static propTypes = {
    kind: PropTypes.oneOf(['primary', 'warning']).isRequired
  };

  render() {
      return (
      <button style={[styles.base, styles[this.props.kind]]}>
        {this.props.children}
      </button>
    );
  }
}

Button = Radium(Button);

// You can create your style objects dynamically or share them for
// every instance of the component.
var styles = {
  base: {
    color: '#fff',

    // Adding interactive state couldn't be easier! Add a special key to your
    // style object (:hover, :focus, :active, or @media) with the additional rules.
    ':hover': {
      background: color('#0074d9')
```

```
        .lighten(0.2)
        .hexString()
    }
  },

  primary: {
    background: '#0074D9'
  },

  warning: {
    background: '#FF4136'
  }
};
```

This is all about styling in React.