React CSS

CSS in React is used to style the React App or Component. The **style** attribute is the most used attribute for styling in React applications, which adds dynamically-computed styles at render time. It accepts a JavaScript object in **camelCased** properties rather than a CSS string. There are many ways available to add styling to your React App or Component with CSS. Here, we are going to discuss mainly **four** ways to style React Components, which are given below:

1. Inline Styling
2. CSS Stylesheet
3. CSS Module
4. Styled Components

1. Inline Styling

The inline styles are specified with a JavaScript object in camelCase version of the style name. Its value is the style?s value, which we usually take in a string.

Example

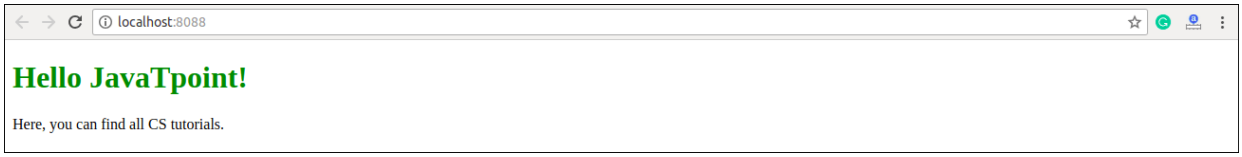
App.js

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

class App extends React.Component {
  render() {
    return (
      <div>
      <h1 style={{color: "Green"}}>Hello JavaTpoint!</h1>
      <p>Here, you can find all CS tutorials.</p>
      </div>
    );
  }
}
export default App;
```

Note: You can see in the above example, we have used two curly braces in: **<h1 style={{color: "Green"}}>Hello JavaTpoint!</h1>**.

It is because, in JSX, JavaScript expressions are written inside curly braces, and JavaScript objects also use curly braces, so the above styling is written inside two sets of curly braces {{}}.

Output



camelCase Property Name

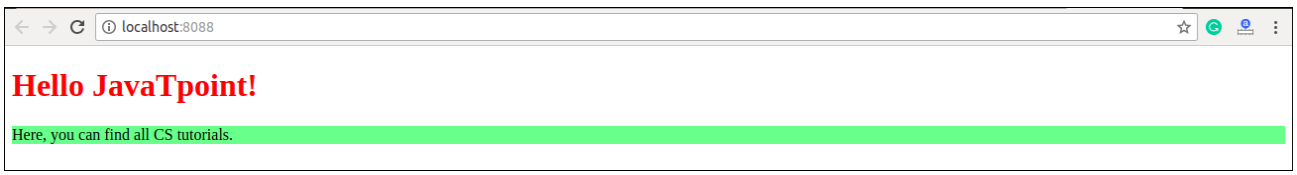
If the properties have two names, like **background-color**, it must be written in camel case syntax.

Example

App.js

```
1. import React from 'react';
2. import ReactDOM from 'react-dom';
3.
4. class App extends React.Component {
5.   render() {
6.     return (
7.       <div>
8.       <h1 style={{color: "Red"}}>Hello JavaTpoint!</h1>
9.       <p style={{backgroundColor: "lightgreen"}}>Here, you can find all CS tutorials.</
   p>
10.      </div>
11.    );
12.  }
13. }
14. export default App;
```

Output



Using JavaScript Object

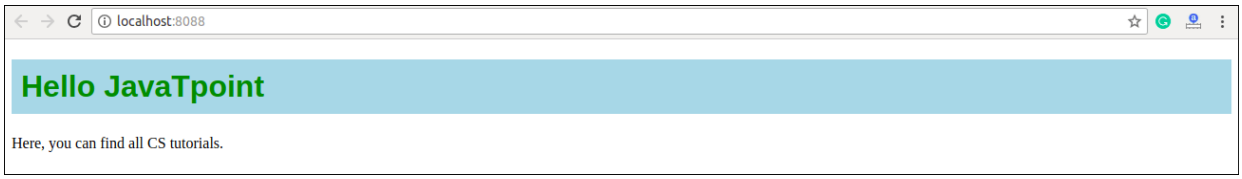
The inline styling also allows us to create an object with styling information and refer it in the style attribute.

Example

App.js

```
1. import React from 'react';
2. import ReactDOM from 'react-dom';
3.
4. class App extends React.Component {
5.   render() {
6.     const mystyle = {
7.       color: "Green",
8.       backgroundColor: "lightBlue",
9.       padding: "10px",
10.      fontFamily: "Arial"
11.    };
12.    return (
13.      <div>
14.      <h1 style={mystyle}>Hello JavaTpoint</h1>
15.      <p>Here, you can find all CS tutorials.</p>
16.      </div>
17.    );
18.  }
19. }
20. export default App;
```

Output



2. CSS Stylesheet

You can write styling in a separate file for your React application, and save the file with a .css extension. Now, you can **import** this file in your application.

Example

App.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './App.css';

class App extends React.Component {
  render() {
    return (
      <div>
      <h1>Hello JavaTpoint</h1>
      <p>Here, you can find all CS tutorials.</p>
      </div>
    );
  }
}
export default App;
```

App.css

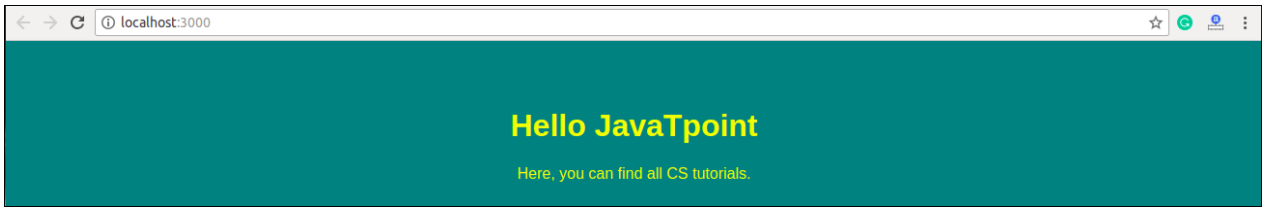
```
body {
  background-color: #008080;
  color: yellow;
  padding: 40px;
  font-family: Arial;
  text-align: center;
}
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
```

```
3.   <head>
4.     <meta charset="utf-8" />
5.     <meta name="viewport"
6.       content="width=device-width, initial-scale=1" />
7.     <title>React App</title>
8.   </head>
9.   <body>
10.    <div id="app"></div>
11.  </body>
12. </html>
```

Output



3. CSS Module

CSS Module is another way of adding styles to your application. It is a **CSS file** where all class names and **animation** names are scoped locally by default. It is available only for the component which imports it, means any styling you add can never be applied to other components without your permission, and you never need to worry about name conflicts. You can create CSS Module with the **.module.css** extension like a **myStyles.module.css** name.

Example

App.js

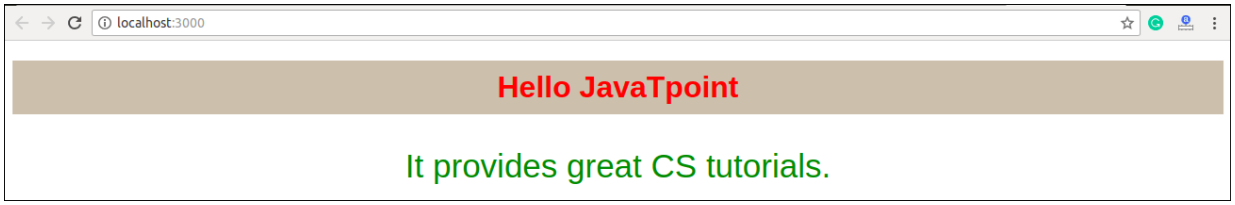
```
1.  import React from 'react';
2.  import ReactDOM from 'react-dom';
3.  import styles from './myStyles.module.css';
4.
5.  class App extends React.Component {
6.    render() {
7.      return (
8.        <div>
9.        <h1 className={styles.mystyle}>Hello JavaTpoint</h1>
```

```
10.       <p className={styles.parastyle}>It provides great CS tutorials.</p>
11.       </div>
12.     );
13.   }
14. }
15. export default App;
```

myStyles.module.css

```
1.  .mystyle {
2.    background-color: #cdc0b0;
3.    color: Red;
4.    padding: 10px;
5.    font-family: Arial;
6.    text-align: center;
7.  }
8.
9.  .parastyle{
10.   color: Green;
11.   font-family: Arial;
12.   font-size: 35px;
13.   text-align: center;
14. }
```

Output



4. Styled Components

Styled-components is a **library** for React. It uses enhance CSS for styling React component systems in your application, which is written with a mixture of JavaScript and CSS.

The styled-components provides:

- Automatic critical CSS

- No class name bugs
- Easier deletion of CSS
- Simple dynamic styling
- Painless maintenance

Installation

The styled-components library takes a single command to install in your React application. which is:

1. $ npm install styled-components --save

Example

Here, we create a variable by selecting a particular HTML element such as ** **, **<Title>**, and **<paragraph>** where we store our style attributes. Now we can use the name of our variable as a wrapper **<Div></Div>** kind of React component.

App.js

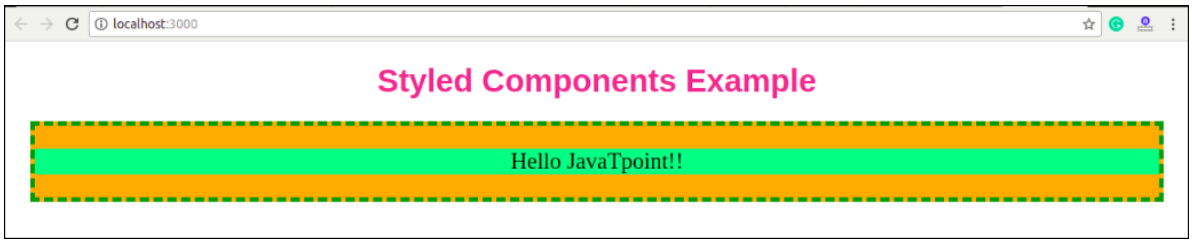
```
import React from 'react';
import ReactDOM from 'react-dom';
import styled from 'styled-components';

class App extends React.Component {
  render() {
    const Div:any = styled.div`
          margin: 20px;
          border: 5px dashed green;
          &:hover {
          background-color: ${(props:any) => props.hoverColor};
          }
          `;
    const Title = styled.h1`
          font-family: Arial;
          font-size: 35px;
          text-align: center;
          color: palevioletred;
          `;
```

```
20.    const Paragraph = styled.p`
21.            font-size: 25px;
22.            text-align: center;
23.            background-Color: lightgreen;
24.            `;
25.    return (
26.       <div>
27.            <Title>Styled Components Example</Title>
28.            <p></p>
29.            <Div hoverColor="Orange">
30.                <Paragraph>Hello JavaTpoint!!</Paragraph>
31.            </Div>
32.        </div>
33.    );
34.  }
35. }
36. export default App;
```

Output

Now, execute the App.js file, we will get the output as shown below.

When we move the mouse pointer over the image, its color will be changed, as shown in the below image.



React Animation

The animation is a technique in which images are manipulated to appear as moving images. It is one of the most used technique to make an interactive web application. In React, we can add animation using an explicit group of components known as the **React Transition Group**.

React Transition Group is an add-on component for managing component states and useful for defining **entering** and **exiting** transitions. It is not able to animate styles by itself. Instead, it exposes transition states, manages classes and group elements, and manipulates the DOM in useful ways. It makes the implementation of visual transitions much easier.

React Transition group has mainly **two APIs** to create transitions. These are:

1. **ReactTransitionGroup:** It uses as a low-level API for animation.
2. **ReactCSSTransitionGroup:** It uses as a high-level API for implementing basic CSS transitions and animations.

Installation

We need to install **react-transition-group** for creating animation in React Web application. You can use the below command.

1. $ npm install react-transition-group --save

React Transition Group Components

React Transition Group API provides **three** main components. These are:

1. Transition
2. CSSTransition
3. Transition Group

Transition

It has a simple component API to describe a transition from one component state to another over time. It is mainly used to animate the **mounting** and **unmounting** of a component. It can also be used for in-place transition states as well.

We can access the Transition component into four states:

- entering
- entered
- exiting
- exited

CSSTransition

The CSSTransition component uses CSS stylesheet classes to write the transition and create animations. It is inspired by the **ng-animate** library. It can also inherit all the props of the transition component. We can divide the "CSSTransition" into **three** states. These are:

- Appear
- Enter
- Exit

CSSTransition component must be applied in a pair of class names to the child components. The first class is in the form of **name-stage** and the second class is in the **name-stage-active**. For example, you provide the name fade, and when it applies to the 'enter' stage, the two classes will be **fade-enter** and **fade-enter-active**. It may also take a prop as Timeout which defines the maximum time to animate.

TransitionGroup

This component is used to manage a set of transition components (Transition and CSSTransition) in a list. It is a state machine that controls the **mounting** and **unmounting** of components over time. The Transition component does not define any animation directly. Here, how 'list' item animates is based on the individual transition component. It means, the "TransitionGroup" component can have different animation within a component.

Let us see the example below, which clearly help to understand the React Animation.

Example

App.js

In the App.js file, import react-transition-group component, and create the CSSTransition component that uses as a wrapper of the component you want to animate. We are going to use **transitionEnterTimeout** and **transitionLeaveTimeout** for CSS Transition. The

Enter and Leave animations used when we want to insert or delete elements from the list.

```
import React, { Component } from 'react';
import { CSSTransitionGroup } from 'react-transition-group';

class App extends React.Component {
   constructor(props) {
   super(props);
   this.state = {items: ['Blockchain', 'ReactJS', 'TypeScript', 'JavaTpoint']};
   this.handleAdd = this.handleAdd.bind(this);
  }

 handleAdd() {
   const newItems = this.state.items.concat([
     prompt('Enter Item Name')
   ]);
   this.setState({items: newItems});
 }

 handleRemove(i) {
   let newItems = this.state.items.slice();
   newItems.splice(i, 1);
   this.setState({items: newItems});
 }

 render() {
   const items = this.state.items.map((item, i) => (
     <div key={item} onClick={() => this.handleRemove(i)}>
       {item}
     </div>
   ));

   return (
     <div>
   <h1>Animation Example</h1>
         <button onClick={this.handleAdd}>Insert Item</button>
```

```
35.             <CSSTransitionGroup
36.               transitionName="example"
37.             transitionEnterTimeout={800}
38.               transitionLeaveTimeout={600}>
39.               {items}
40.             </CSSTransitionGroup>
41.       </div>
42.     );
43.   }
44. }
45. export default App;
```

Main.js

```
1.  import React from 'react';
2.  import ReactDOM from 'react-dom';
3.  import App from './App.js';
4.
5.  ReactDOM.render(<App />, document.getElementById('app'));
```

style.css

Add style.css file in your application, and add the following CSS styles. Now, to use this CSS file, you need to add the **link** of this file in your HTML file.

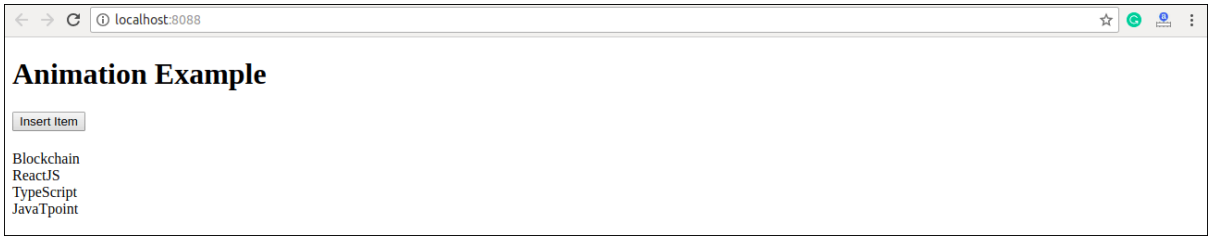
```
1.  .example-enter {
2.    opacity: 0.01;
3.  }
4.
5.  .example-enter.example-enter-active {
6.    opacity: 1;
7.    transition: opacity 500ms ease-in;
8.  }
9.
10. .example-leave {
11.   opacity: 1;
12. }
13.
14. .example-leave.example-leave-active {
```

```
15.  opacity: 0.01;
16.  transition: opacity 300ms ease-in;
17. }
```

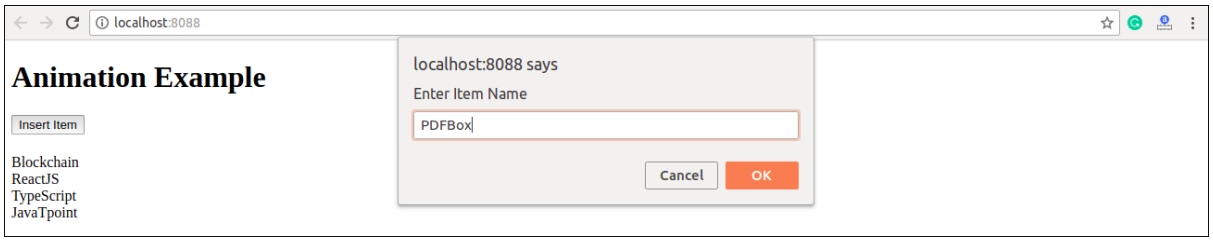
In the above example, the animation durations are specified in both the **CSS** and **render** method. It tells React component when to remove the animation classes from the list and if it is leaving when to remove the element from the DOM.

Output

When we execute the above program, it gives the below output.



Click on '**Insert Item**' button, the following screen appears.



Once we insert the item and press **Ok**, the new item can be added in the list with fade in style. Here, we can also delete any item from the list by clicking on the particular link.

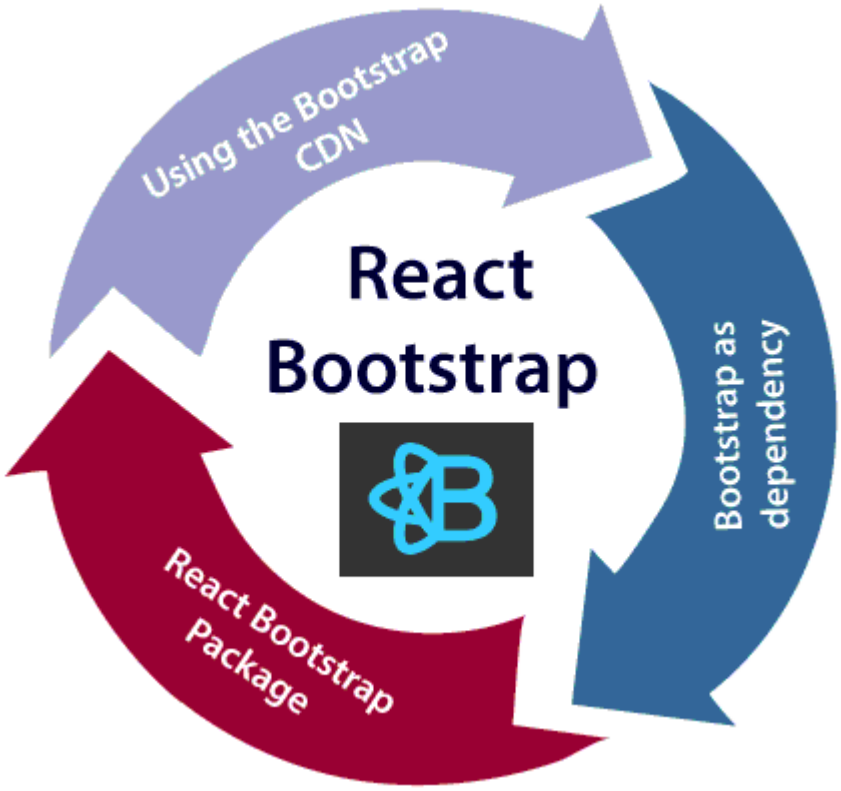
React Bootstrap

Single-page applications gaining popularity over the last few years, so many front-end frameworks have introduced such as Angular, React, Vue.js, Ember, etc. As a result, jQuery is not a necessary requirement for building web apps. Today, React has the most used JavaScript framework for building web applications, and Bootstrap become the most popular CSS framework. So, it is necessary to learn various ways in which Bootstrap can be used in React apps, which is the main aim of this section.

Adding Bootstrap for React

We can add Bootstrap to the React app in several ways. The **three** most common ways are given below:

1. Using the Bootstrap CDN
2. Bootstrap as Dependency
3. React Bootstrap Package



Using the Bootstrap CDN

It is the easiest way of adding Bootstrap to the React app. There is no need to install or download Bootstrap. We can simply put an **<link>** into the **<head>** section of the **index.html** file of the React app as shown in the following snippet.

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
```

If there is a need to use Bootstrap components which depend on JavaScript/jQuery in the React application, we need to include **jQuery**, **Popper.js**, and **Bootstrap.js** in the document. Add the following imports in the **<script>** tags near the end of the closing **</body>** tag of the **index.html** file.

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>
```

In the above snippet, we have used jQuery's slim version, although we can also use the full version as well. Now, Bootstrap is successfully added in the React application, and we can use all the CSS utilities and UI components available from Bootstrap in the React application.

Bootstrap as Dependency

If we are using a build tool or a module bundler such as Webpack, then importing Bootstrap as dependency is the preferred option for adding Bootstrap to the React

application. We can install Bootstrap as a dependency for the React app. To install the Bootstrap, run the following commands in the terminal window.

```
$ npm install bootstrap --save
```

Once Bootstrap is installed, we can import it in the React application entry file. If the React project created using the **create-react-app** tool, open the **src/index.js** file, and add the following code:

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

Now, we can use the CSS classes and utilities in the React application. Also, if we want to use the JavaScript components, we need to install the **jquery** and **popper.js** packages from **npm**. To install the following packages, run the following command in the terminal window.

```
$ npm install jquery popper.js
```

Next, go to the **src/index.js** file and add the following imports.

```
import $ from 'jquery';
import Popper from 'popper.js';
import 'bootstrap/dist/js/bootstrap.bundle.min';
```

Now, we can use Bootstrap JavaScript Components in the React application.

React Bootstrap Package

The React Bootstrap package is the most popular way to add bootstrap in the React application. There are many Bootstrap packages built by the community, which aim to rebuild Bootstrap components as React components. The **two** most popular Bootstrap packages are:

1. **react-bootstrap:** It is a complete re-implementation of the Bootstrap components as React components. It does not need any dependencies like bootstrap.js or jQuery. If the React setup and React-Bootstrap installed, we have everything which we need.
2. **reactstrap:** It is a library which contains React Bootstrap 4 components that favor composition and control. It does not depend on jQuery or Bootstrap JavaScript. However, react-popper is needed for advanced positioning of content such as Tooltips, Popovers, and auto-flipping Dropdowns.

React Bootstrap Installation

Let us create a new React app using the **create-react-app** command as follows.

```
$ npx create-react-app react-bootstrap-app
```

After creating the React app, the best way to install Bootstrap is via the npm package. To install Bootstrap, navigate to the React app folder, and run the following command.

```
$ npm install react-bootstrap bootstrap --save
```

Importing Bootstrap

Now, open the **src/index.js** file and add the following code to import the Bootstrap file.

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

We can also import individual components **like import { SplitButton, Dropdown } from 'react-bootstrap';** instead of the entire library. It provides the specific components which we need to use, and can significantly reduce the amount of code.

In the React app, create a new file named **ThemeSwitcher.js** in the **src** directory and put the following code.

```
import React, { Component } from 'react';
import { SplitButton, Dropdown } from 'react-bootstrap';

class ThemeSwitcher extends Component {

  state = { theme: null }

  chooseTheme = (theme, evt) => {
    evt.preventDefault();
    if (theme.toLowerCase() === 'reset') { theme = null }
    this.setState({ theme });
  }

  render() {
    const { theme } = this.state;
    const themeClass = theme ? theme.toLowerCase() : 'default';
```

```
17.
18.   const parentContainerStyles = {
19.     position: 'absolute',
20.     height: '100%',
21.     width: '100%',
22.     display: 'table'
23.   };
24.
25.   const subContainerStyles = {
26.     position: 'relative',
27.     height: '100%',
28.     width: '100%',
29.     display: 'table-cell',
30.   };
31.
32.   return (
33.     <div style={parentContainerStyles}>
34.       <div style={subContainerStyles}>
35.
36.         <span className={`h1 center-block text-center text-
    ${theme ? themeClass : 'muted'}`} style={{ marginBottom: 25 }}>{theme || 'Default'}</
    span>
37.
38.         <div className="center-block text-center">
39.           <SplitButton bsSize="large" bsStyle={themeClass} title={`${theme || 'Default B
    lock'} Theme`}>
40.             <Dropdown.Item eventKey="Primary Block" onSelect={this.chooseTheme}>
    Primary Theme</Dropdown.Item>
41.             <Dropdown.Item eventKey="Danger Block" onSelect={this.chooseTheme}>
    Danger Theme</Dropdown.Item>
42.             <Dropdown.Item eventKey="Success Block" onSelect={this.chooseTheme}>
    Success Theme</Dropdown.Item>
43.             <Dropdown.Item divider />
44.             <Dropdown.Item eventKey="Reset Block" onSelect={this.chooseTheme}>De
    fault Theme</Dropdown.Item>
45.           </SplitButton>
46.         </div>
```

```
47.         </div>
48.       </div>
49.     );
50.   }
51. }
52. export default ThemeSwitcher;
```

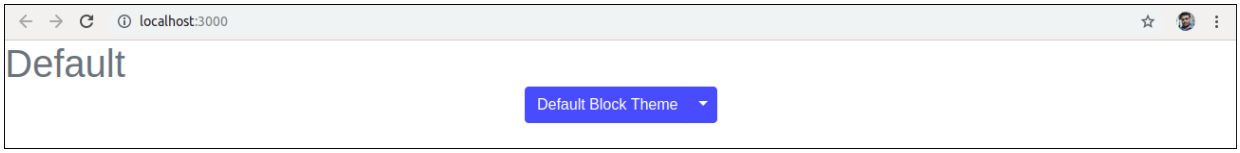
Now, update the **src/index.js** file with the following snippet.

Index.js

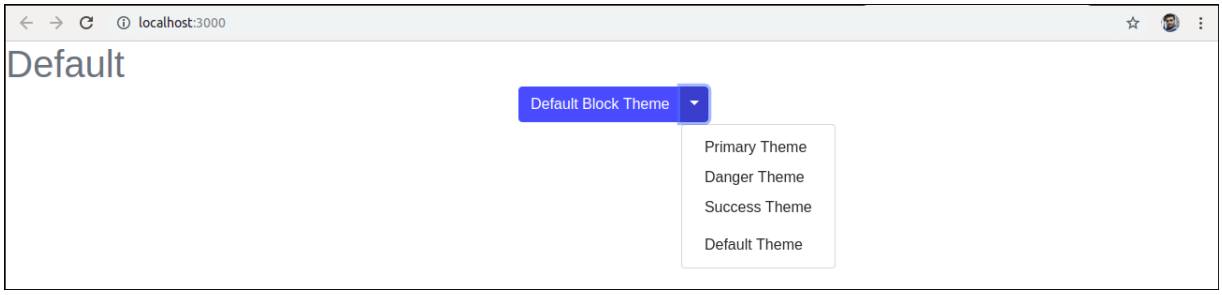
```
1. import 'bootstrap/dist/css/bootstrap.min.css';
2. import React from 'react';
3. import ReactDOM from 'react-dom';
4. import App from './App.js';
5. import './index.css';
6. import ThemeSwitcher from './ThemeSwitcher';
7.
8. ReactDOM.render(<ThemeSwitcher />, document.getElementById('root'));
```

Output

When we execute the React app, we should get the output as below.



Click on the dropdown menu. We will get the following screen.



Now, if we choose the **Success Theme**, we will get the below screen.

Using reactstrap

Let us create a new React app using the create-react-app command as follows.

```
$ npx create-react-app reactstrap-app
```

Next, install the **reactstrap** via the npm package. To install reactstrap, navigate to the React app folder, and run the following command.

```
$ npm install bootstrap reactstrap --save
```

Importing Bootstrap

Now, open the **src/index.js** file and add the following code to import the Bootstrap file.

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

We can also import individual components **like import { Button, Dropdown } from 'reactstrap';** instead of the entire library. It provides the specific components which we need to use, and can significantly reduce the amount of code.

In the React app, create a new file named **ThemeSwitcher.js** in the **src** directory and put the following code.

```
import React, { Component } from 'react';
import { Button, ButtonDropdown, DropdownToggle, DropdownMenu, DropdownItem } from 'reactstrap';

class ThemeSwitcher extends Component {

  state = { theme: null, dropdownOpen: false }

  toggleDropdown = () => {
    this.setState({ dropdownOpen: !this.state.dropdownOpen });
  }

```

```
12.  resetTheme = evt => {
13.    evt.preventDefault();
14.    this.setState({ theme: null });
15.  }
16.
17.  chooseTheme = (theme, evt) => {
18.    evt.preventDefault();
19.    this.setState({ theme });
20.  }
21.  render() {
22.    const { theme, dropdownOpen } = this.state;
23.    const themeClass = theme ? theme.toLowerCase() : 'secondary';
24.
25.    return (
26.      <div className="d-flex flex-wrap justify-content-center align-items-center">
27.
28.        <span className={`h1 mb-4 w-100 text-center text-
    ${themeClass}`}>{theme || 'Default'}</span>
29.
30.        <ButtonDropdown isOpen={dropdownOpen} toggle={this.toggleDropdown}>
31.          <Button id="caret" color={themeClass}>{theme || 'Custom'} Theme</Button>
32.          <DropdownToggle caret size="lg" color={themeClass} />
33.          <DropdownMenu>
34.            <DropdownItem onClick={e => this.chooseTheme('Primary', e)}>Primary The
    me</DropdownItem>
35.            <DropdownItem onClick={e => this.chooseTheme('Danger', e)}>Danger The
    me</DropdownItem>
36.            <DropdownItem onClick={e => this.chooseTheme('Success', e)}>Success The
    me</DropdownItem>
37.            <DropdownItem divider />
38.            <DropdownItem onClick={this.resetTheme}>Default Theme</DropdownItem
    >
39.          </DropdownMenu>
40.        </ButtonDropdown>
41.
42.      </div>
43.    );
```

```
44.   }
45. }
46. export default ThemeSwitcher;
```

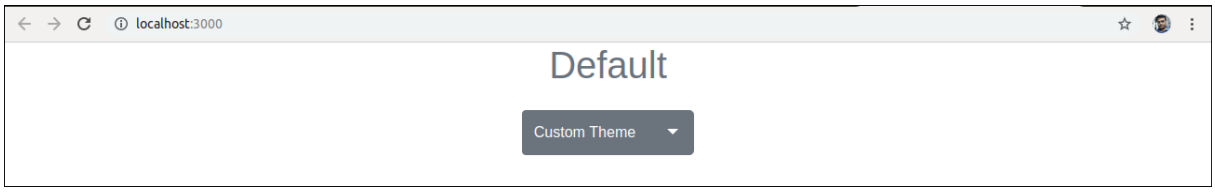
Now, update the **src/index.js** file with the following snippet.

Index.js

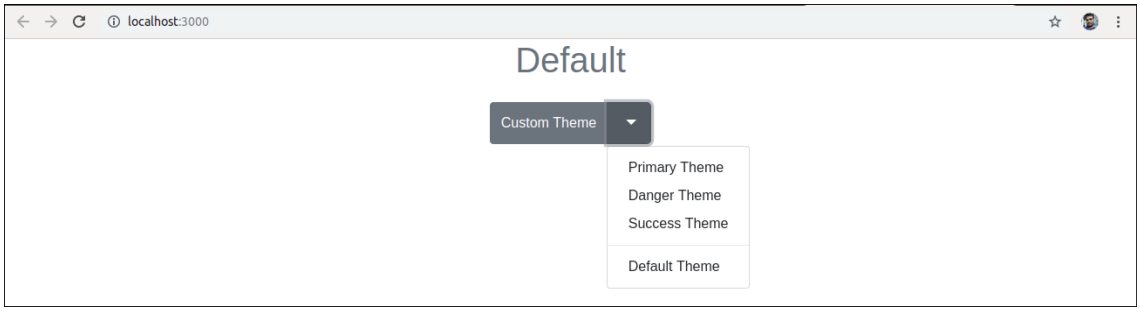
```
1. import 'bootstrap/dist/css/bootstrap.min.css';
2. import React from 'react';
3. import ReactDOM from 'react-dom';
4. import App from './App.js';
5. import './index.css';
6. import ThemeSwitcher from './ThemeSwitcher';
7.
8. ReactDOM.render(<ThemeSwitcher />, document.getElementById('root'));
```

Output

When we execute the React app, we should get the output as below.



Click on the dropdown menu. We will get the following screen.



Now, if we choose the **Danger Theme**, we will get the below screen.