**Create a New React App**

Create a new React app using the following command:

npx create-react-app my-app

Next, open the app.js file and remove all the code present. We will write our counter code here. To start the local server or run the app, use the following command:

npm start

**Get Started**

Now we will create a simple functional component in the app.js page and also import the useState hook. Add the following code inside the file:

import {React, useState } from 'react' export default function App() { return ( )}

**Create HTML Elements**

Now we will create the HTML code for the form. The styling will be done later. We need to create two buttons and a *span* tag to show the output. There will be three buttons: +, - and reset. Add the following code within the app.js file inside the return() function:

return ( <div className="counter"> <h1>React Counter</h1> <span className="counter\_\_output"></span> <div className="btn\_\_container"> <button className="control\_\_btn" >+</button> <button className="control\_\_btn">-</button> <button className="reset">Reset</button> </div> </div>)

**Create States and Functions**

Now we will create a state called *counter* that will store the counter value. The state will be initially set to 0. Every time user will click either on + or -, the value will change. The value will be displayed in the span tag. There will be three functions used to control the value.

1. **increase** - this will increment the value
2. **decrease** - this will decrement
3. **reset** - to reset the counter back to zero.

The complete code is given below:

import { React, useState } from 'react' export default function App() { const [counter, setCounter] = useState(0); //increase counter const increase = () => { setCounter(count => count + 1); }; //decrease counter const decrease = () => { setCounter(count => count - 1); }; //reset counter const reset = () =>{ setCounter(0) } return ( <div className="counter"> <h1>React Counter</h1> <span className="counter\_\_output">{counter}</span> <div className="btn\_\_container"> <button className="control\_\_btn" onClick={increase}>+</button> <button className="control\_\_btn" onClick={decrease}>-</button> <button className="reset" onClick={reset}>Reset</button> </div> </div> );}

In the above code, we change the value of the state using setCounter(). To display the result, we use {count} inside the span. On click events are added to each button containing the corresponding functions.

**Validation Check**

Now we will add an if statement to prevent the value of the counter from going into negative digits. This is not necessary for this tutorial, as the counter is completely functional. But it is a useful thing, as it is not practical for a counter to have a negative value.

We will use an if statement to make the minimum value 0. The if statement will be added in the decrease() function.

//decrease counterconst decrease = () => { if (counter > 0) { setCounter(count => count - 1); }};

In the above code, the value will decease only if the current value of the counter is more than 0.

**CSS: The Icing on the Cake**

Right now, the counter is completely functional, but it looks terrible. Now we will do some CSS to style it.

Create a new file called app.css (if it is not already create by default) and add the following code:

**App.css**

.counter { width: 100%; display: flex; align-items: center; flex-direction: column; text-align: center; row-gap: 20px;} .counter h1 { color: rgb(16, 0, 54); font-size: 40px; font-family: cursive;} .counter\_\_output { font-size: 40px; color: rgb(116, 7, 7);} .btn\_\_container { display: flex; justify-content: center; flex-direction: row; column-gap: 20px;} .control\_\_btn { font-size: 20px; padding: 10px 20px; background-color: transparent; color: rgb(16, 0, 54); border: 1px solid rgb(16, 0, 54); cursor: pointer; transition: 0.2s ease-in-out;} .control\_\_btn:hover { background-color: rgb(16, 0, 54); color: rgb(255, 255, 255); border: 1px solid rgb(16, 0, 54);} .reset { width: 100px; font-size: 18px; padding: 10px 20px; background-color: transparent; color: rgb(16, 0, 54); border: 1px solid rgb(16, 0, 54); cursor: pointer; transition: 0.2s ease-in-out;} .reset:hover { background-color: rgb(16, 0, 54); color: rgb(255, 255, 255); border: 1px solid rgb(16, 0, 54);}

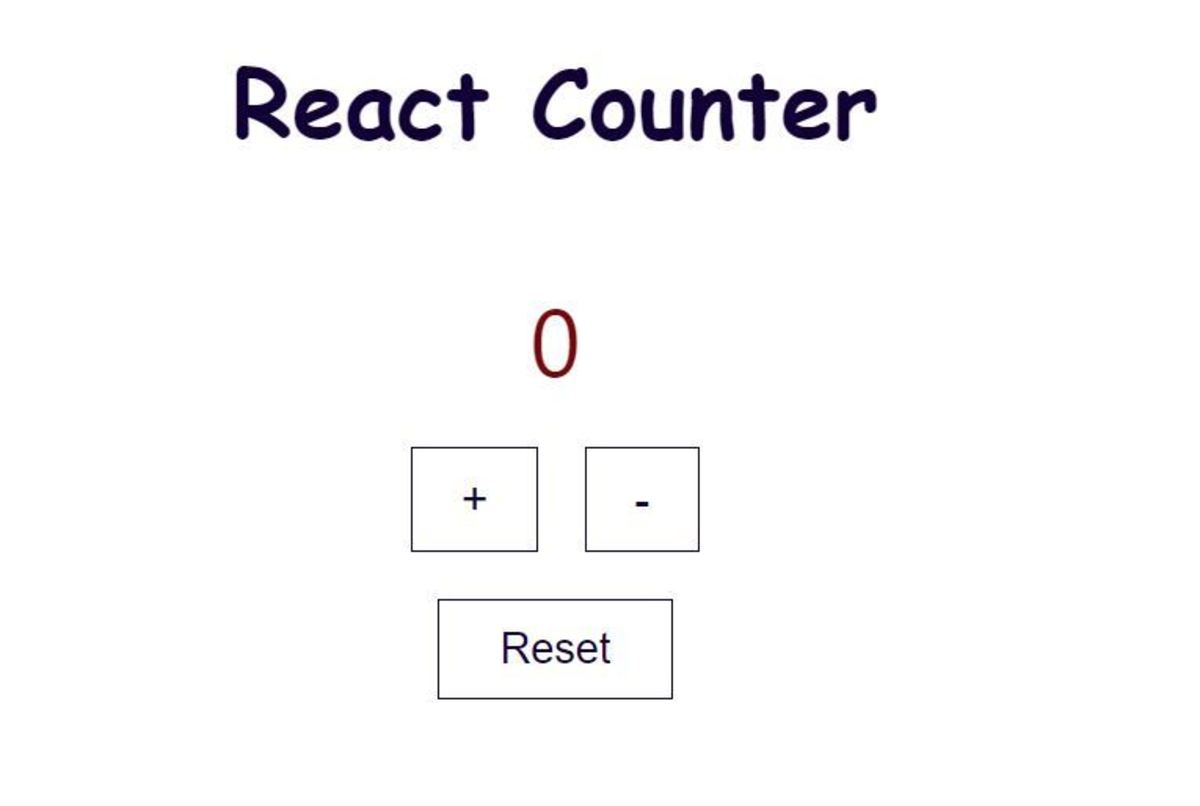
**Apply the Styles**

Now import this CSS to the app.js file using the following code.

import "./App.css"

**The Final Output**

Below is an image of the browser view of this app. As you can see, it looks god visually and is also functional.



//https://owlcation.com/stem/reactCounter