React

Introduction:

React (also known as **React.js** or **ReactJS**) is a free and open-source front-end JavaScript library for building user interfaces based on components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies. React can be used to develop single-page, mobile, or server-rendered applications with frameworks like Next.js. Because React is only concerned with the user interface and rendering components to the DOM, React applications often rely on libraries for routing and other client-side functionality.

React is named React because of its ability to react to changes in data. React is called React because it was designed to be a declarative, efficient, and flexible JavaScript library for building user interfaces. The name React was chosen because the library was designed to allow developers to "react" to changes in state and data within an application, and to update the user interface in a declarative and efficient manner. React is a JavaScript-based UI development library. Facebook and an open-source developer community run it.

Features:

1. Solid base architecture
2. Extensible architecture
3. Component based library
4. JSX based design architecture
5. Declarative UI library
6. Single Page Applications
7. The Virtual DOM
8. Unidirectional Data Flow

Benefits:

1. Easy to learn
2. Easy to adept in modern as well as legacy application
3. Faster way to code a functionality
4. Availability of large number of ready-made component
5. Large and active community

React adheres to the declarative programming paradigm. Developers design views for each state of an application, and react updates and renders components when data changes. This is in contrast with imperative programming

declarative programming is a programming paradigm a style of building the structure and elements of computer programs that expresses the logic of a computation without describing its control flow.

Official documentation:

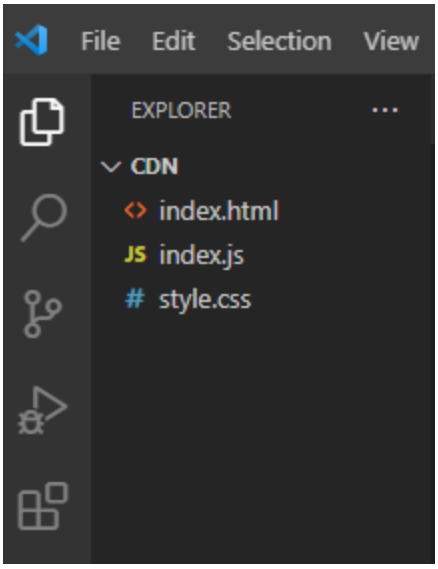
https://create-react-app.dev/

https://legacy.reactjs.org/

https://react.dev/

Run React (CDN)

Folder structure:



CDN links:

```
  <script crossorigin
src="https://unpkg.com/react@18/umd/react.development.js"></scr
ipt>
<script crossorigin src="https://unpkg.com/react-
dom@18/umd/react-dom.development.js"></script>
<script src="https://unpkg.com/babel-
standalone@6/babel.min.js"></script>
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
    <title>React CDN</title>
</head>
<body>
     <div id="root"></div>
     <script src="./index.js" type="text/babel"></script>
</body>
</html>
```

Index.js

```
ReactDOM.render(<h1>hello world</h1>,document.getElementById('root'));
```

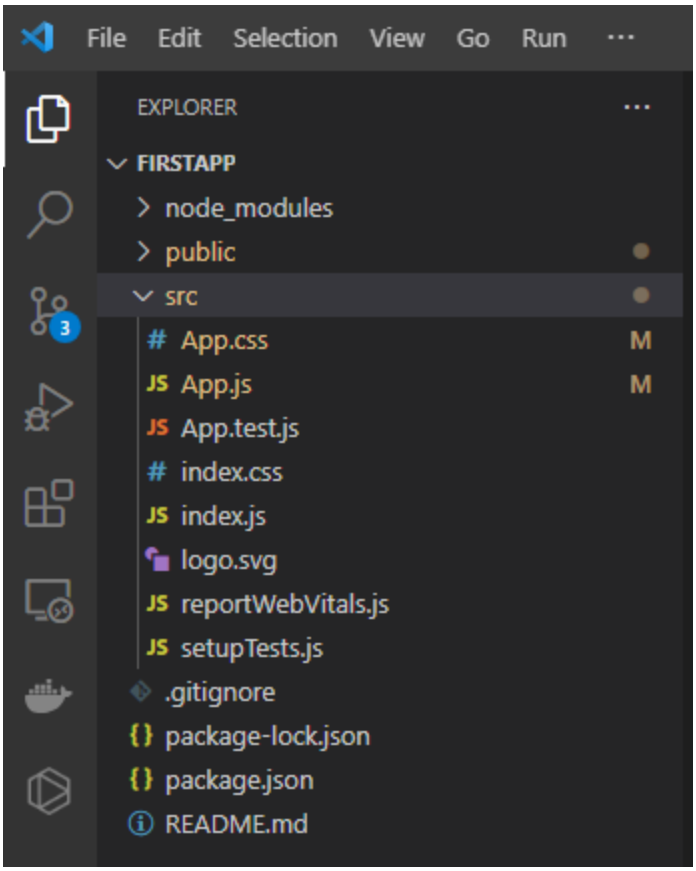
Installation/project setup:

On terminal

npm i -g create-react-app (First Time)

```
npx create-react-app my-app
cd my-app
npm start
```

Folder Structure:



The ***public* folder** - Contains the core file, *index.html* and other web resources like images, logos, robots, etc., *index.html* loads our react application and render it in user's browser.

The *src* folder - Contains the actual code of the application.

The *index.js* - Entry point of our application. It uses *ReactDOM.render* method to kick-start and start the application.

React.StrictMode is a build-in component used to prevent unexpected bugs by analysing the component for unsafe lifecycle, unsafe API usage, depreciated API usage, etc., and throwing the relevant warning.

App is our first custom and root component of the application. All other components will be rendered inside the *App* component.

The *index.css* - Used to styles of the entire application

App.css **-** Used to style the *App* component.

App.test.js **-** Used to write unit test function for our component

setupTests.js **-** Used to setup the testing framework for our application.

reportWebVitals.js **-** Generic web application startup code to support all browsers.

Create React app using Vite(veet) build tool

https://vitejs.dev/

on CLI type this command:

```
npm create vite@latest
```

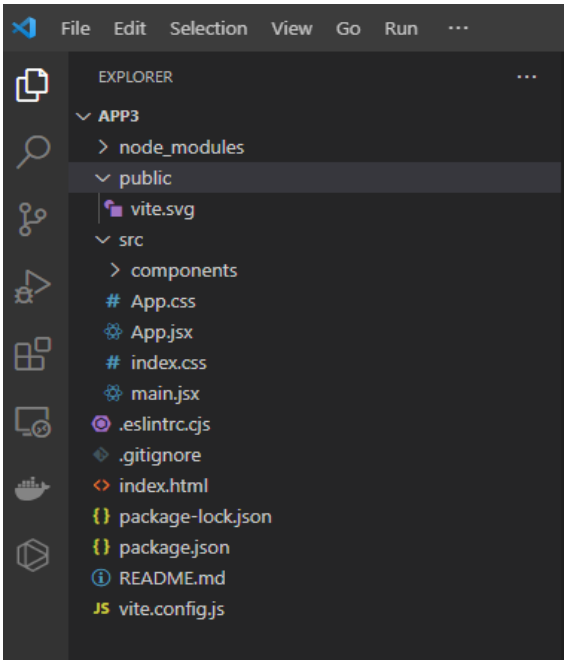
follow instruction

```
cd my-project
```

```
npm install
```

```
npm run dev
```

Folder Structure:



Components:

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return (JSX).A React component represents a small chunk of user interface in a webpage. The primary job of a React component is to render its user interface and update it whenever its internal state is changed. In addition to rendering the UI, it manages the events belongs to its user interface. To summarize, React component provides below functionalities.

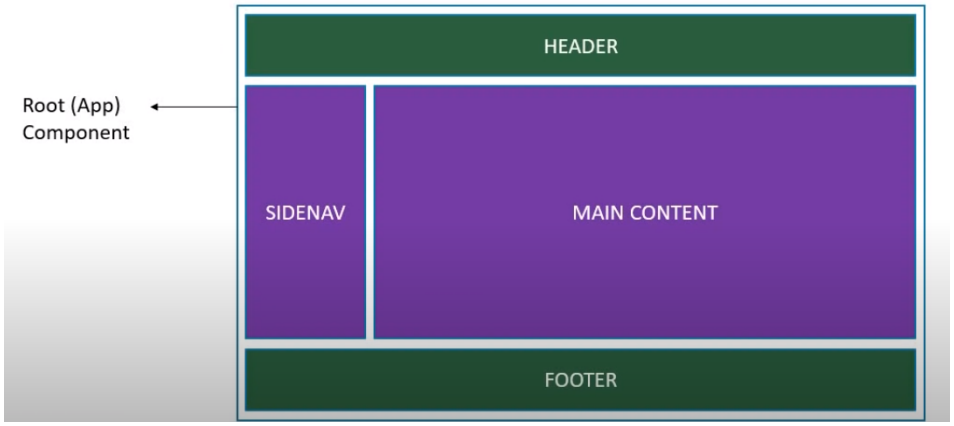
1. Initial rendering of the user interface.
2. Management and handling of events.
3. Updating the user interface whenever the internal state is changed.

React component accomplish these features using three concepts:

1. **Properties -** Enables the component to receive input.

2. **Events -** Enable the component to manage DOM events and end-user interaction.

3. **State -** Enable the component to stay stateful. Stateful component updates its UI with respect to its state.



Components come in two types, Class components and Function components.

Class Components:

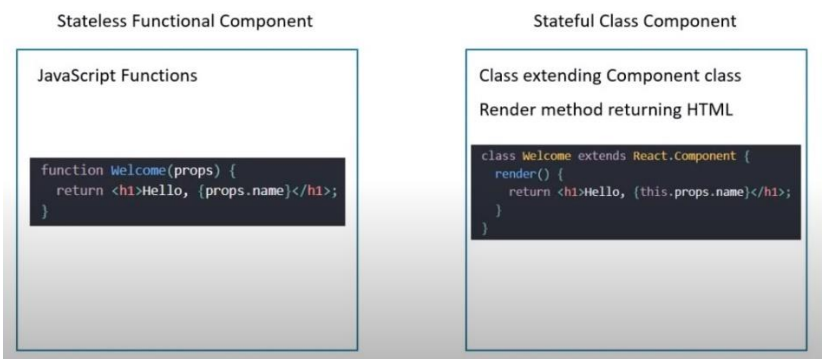
A class component must include the extends React.Component statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.The component also requires a render() method, this method returns HTML.

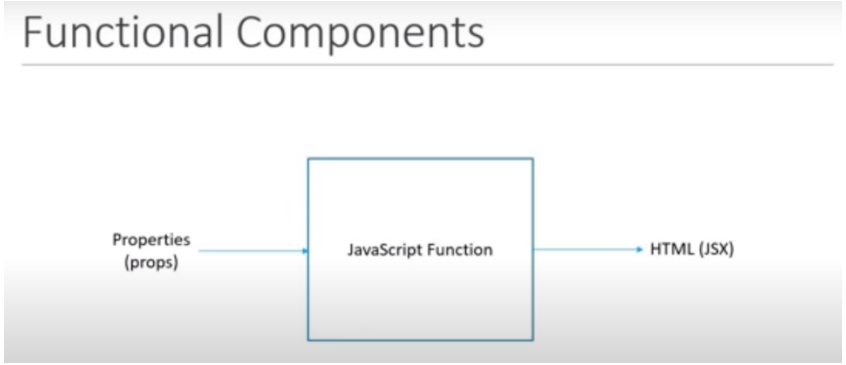
```
JS Home.js U ●
components > JS Home.js > ...
1  import React, { Component } from 'react'
2
3  export default class home extends Component {
4    render() {
5      return (
6        <div>home</div>
7      )
8    }
9  }
```

Function Components:

A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand

```
JS Home.js U ●
components > JS Home.js > ...
1  import React from 'react'
2
3  export default function home() {
4    return (
5      <div>home</div>
6    )
7  }
8
```





Functional components are simple JavaScript function which accept properties as parameter and return html(jsx)

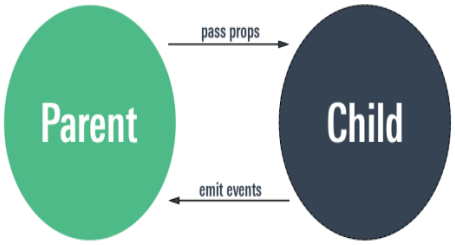
JSX

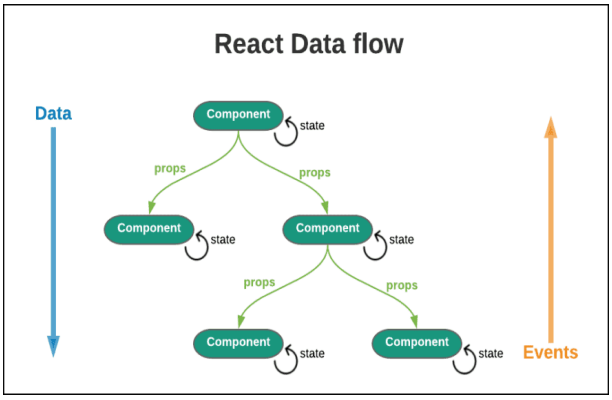
JSX, or JavaScript Syntax Extension, is an extension to the JavaScript language syntax. Similar in appearance to HTML, JSX provides a way to structure component rendering using syntax familiar(html)

```
class App extends React.Component {
    render() {
      return (
        <div>
          <p>Header</p>
          <p>Content</p>
          <p>Footer</p>
        </div>
      );
    }
  }
```

React — Properties (*props*)

In React props are a way to pass the data or properties from one component to other components.

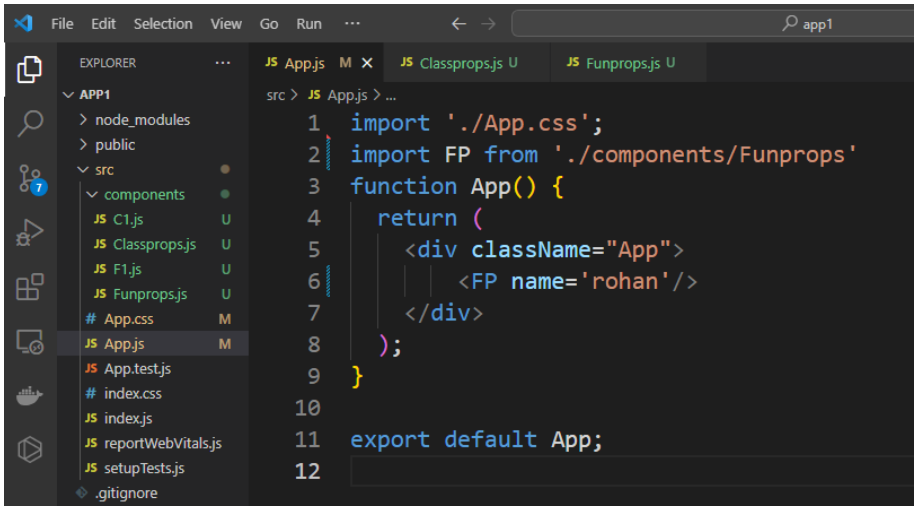




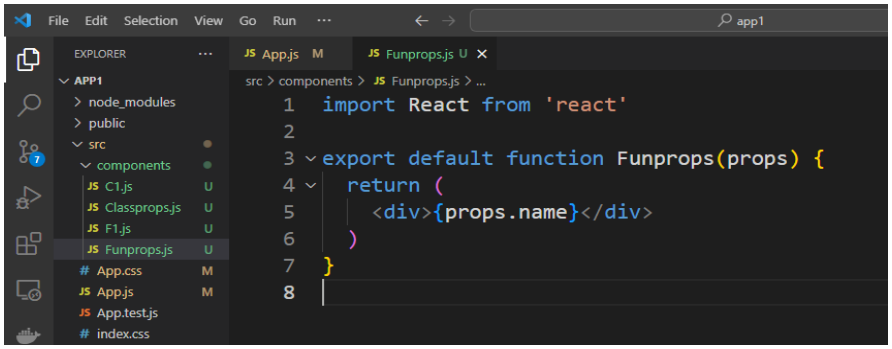
React enables developers to create dynamic and advanced component using properties. Every component can have attributes similar to HTML attributes and each attribute's value can be accessed inside the component using properties (props).

Props in Functional components:

App.js



Funprops.js



Props in Class components:

App.js

```
import './App.css';
import CP from './components/Classprops'
function App() {
  return (
    <div className="App">
        <CP name='rohit'/>
    </div>
  );
}

export default App;
```

Classprops.js

```
import React, { Component } from 'react'

export default class Classprops extends Component {
    constructor (props){
        super(props);
    }
  render() {
    return (
      <div>{this.props.name}</div>
    )
  }
}
```

Send obj as props:

App.js

```
import './App.css';
import FP from './components/Funprops'
function App() {
  const obj={name:'rohan',age:27}
  return (
    <div className="App">
        <FP {...obj}/>
    </div>
  );
}

export default App;
```

Funprops.js

```
import React from 'react'

export default function Funprops(props) {
    return (
        <div>{props.name}</div>
    )
}
```

Or (destructured obj)

```
import React from 'react'
export default function Funprops({name,age}) {
    return (
        <>
        <div>{name}</div>
        <div>{age}</div>
        </>
    )
}
```

Send array as props:

App.js

```
import './App.css';
import FP from './components/Funprops'
function App() {
    const arr=['rohan',27]
    return (
        <div className="App">
            <FP {...arr}/>
        </div>
    );
}

export default App;
```

Funprops.js

```
import React from 'react'
export default function Funprops(props) {
  return (
    <>
    <div>{props[0]}</div>
    <div>{props[1]}</div>
    </>
  )
}
```

Send number as props:

App.js

```
import './App.css';
import FP from './components/Funprops'
function App() {
  return (
    <div className="App">
        <FP num={10}/>
    </div>
  );
}

export default App;
```

Funprops.js

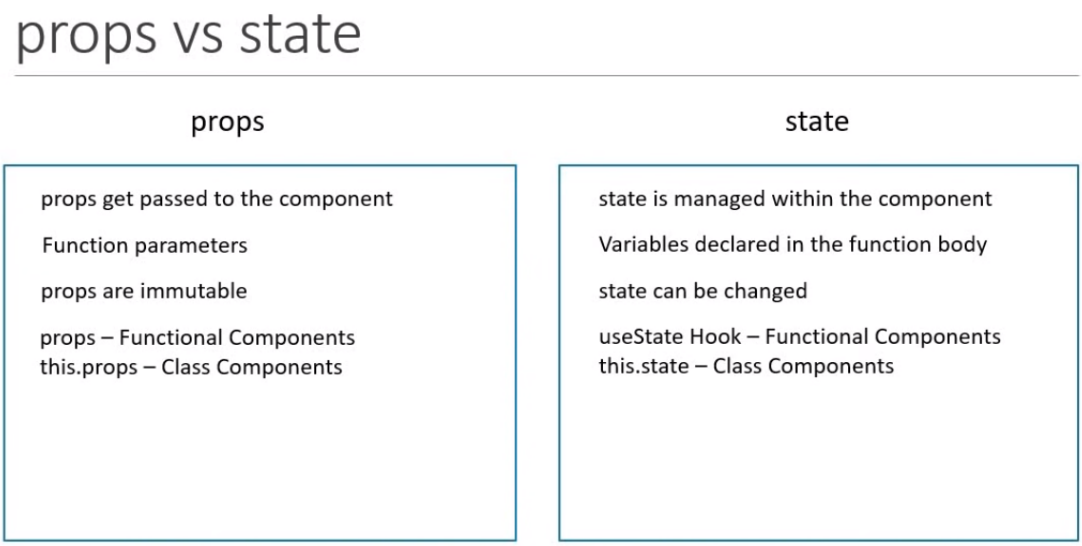
```
import React from 'react'
export default function Funprops(props) {
  return (
    <>
    <div>{props.num}</div>
    </>
  )
}
```

State:

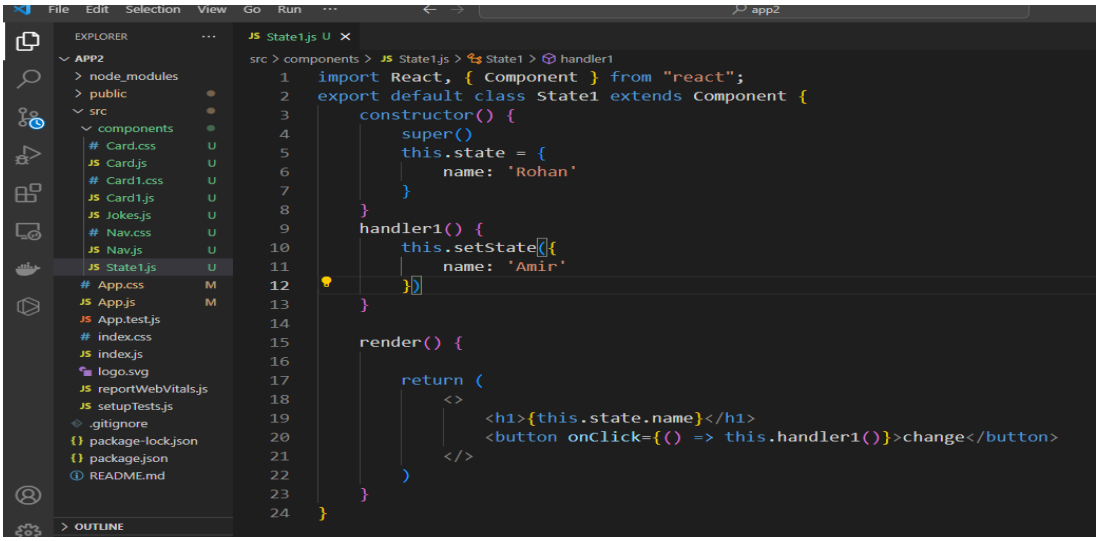
The state is a built-in React object that is used to contain data or information about the component. A component's state can change over time; whenever it changes, the component re-renders.

In React the state is the real-time data available (dynamic properties) to use within that only component.

State represents the value of a dynamic properties of a React component at a given instance. React provides a dynamic data store for each component. The internal data represents the state of a React component and can be accessed using *this.state* member variable of the component. Whenever the state of the component is changed, the component will re-render itself by calling the *render()* method along with the new state.



State in Class Components:

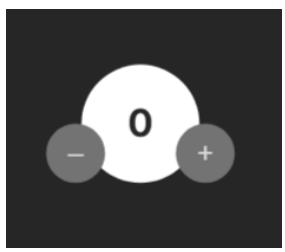


State in Functional Components:

```
import React,{useState} from 'react'

export default function State2() {
    const [name,setname]=useState('rohan');
    const handler=()=>{
            setname('amir')
    }
 return (
    <>
    <h1>{name}</h1>
     <button onClick={handler}>Change</button>
 </>
  )
}
```

Counter Challenge



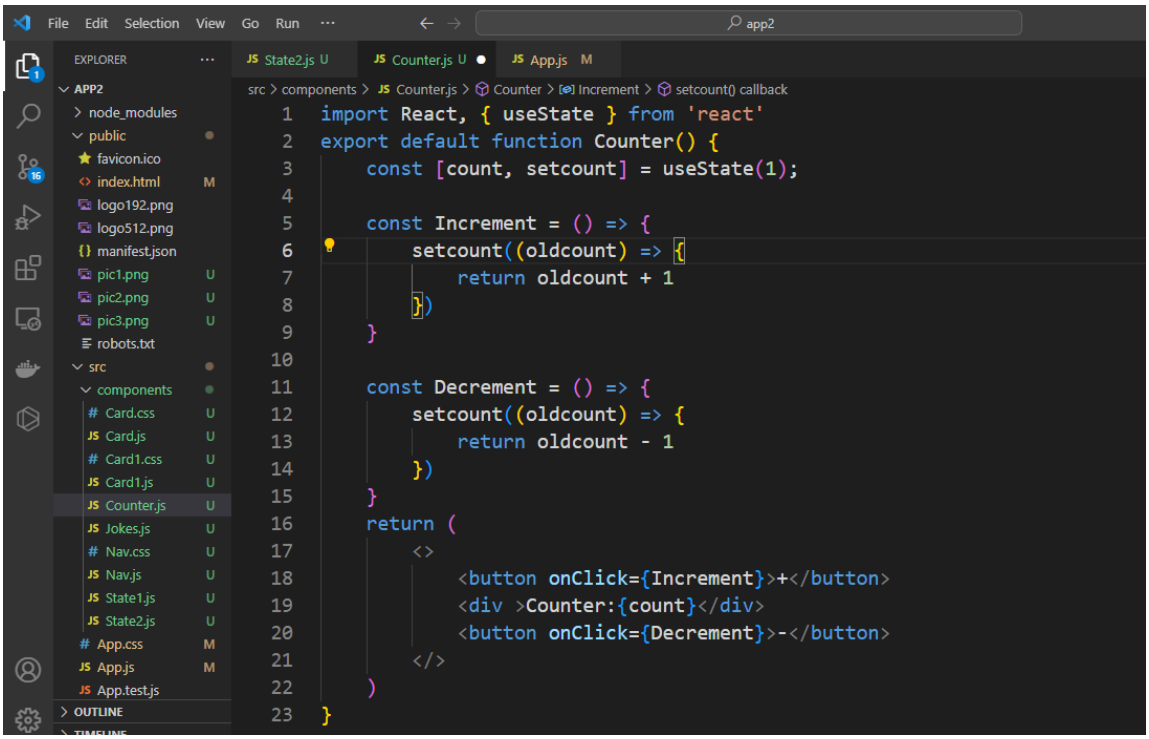
First Method:

```
import React, { useState } from 'react'
export default function Counter() {
    const [count, setcount] = useState(1);

    const Increment = () => {
        setcount(count +1)
    }

    const Decrement = () => {
        setcount(count -1)
    }
    return (
        <>
            <button onClick={Increment}>+</button>
            <div >Counter:{count}</div>
            <button onClick={Decrement}>-</button>
        </>
    )
}
```

Second Method:



Events:

Events in React JS allow developers to respond to user interaction within their applications quickly and efficiently. Events are triggered whenever certain actions occur – from clicks to key presses - which then fire off callback functions so that appropriate responses can be made accordingly. In React, events are handled using event handlers. An event handler is a function that is called when an event occurs.

React also has built-in events to handle common interactions such as onMouseEnter, onChange, onFocus, onBlur, onKeyDown, onKeyUp, onSubmit, etc.