

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:

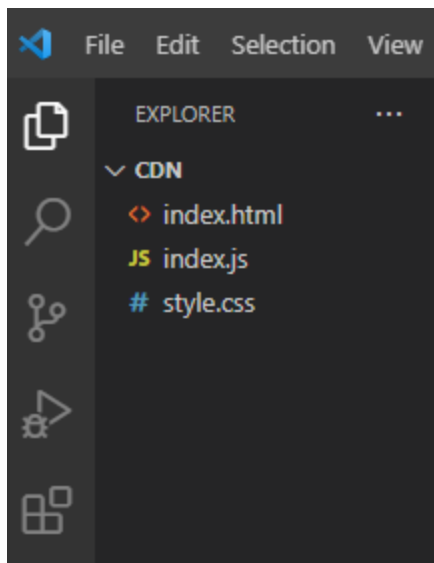
<https://create-react-app.dev/>

<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"></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>
  <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'));
```

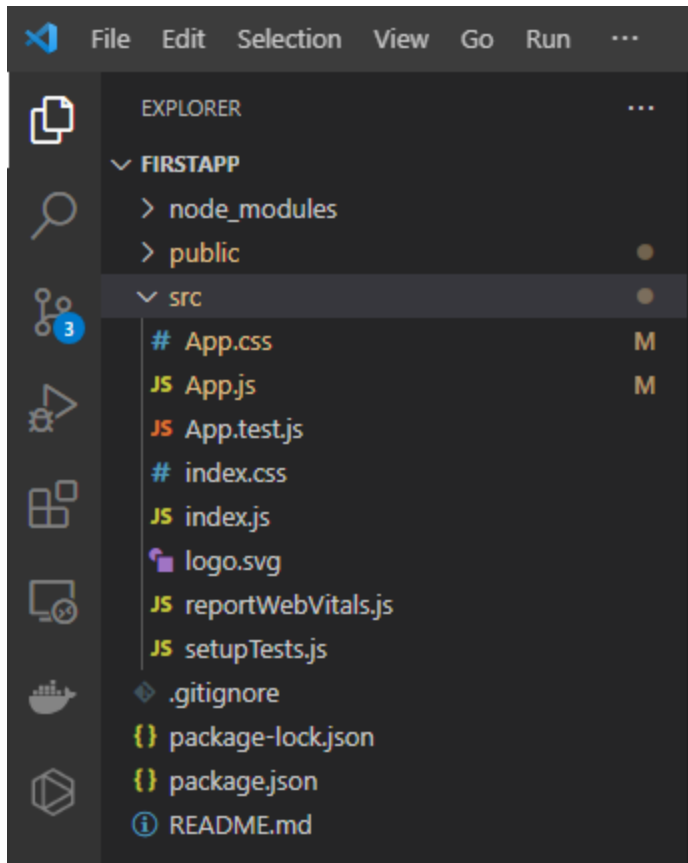
Installation/project setup:

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, deprecated 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
```

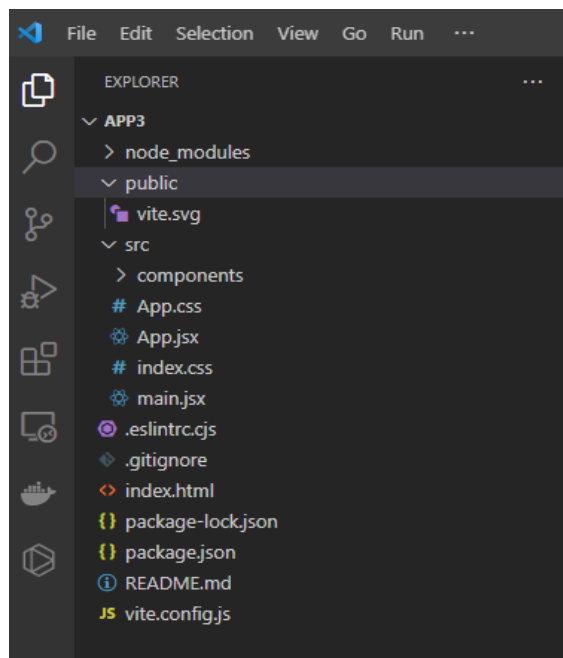
follow instruction

```
cd my-project
```

```
npm install
```

```
npm run dev
```

Folder Structure:



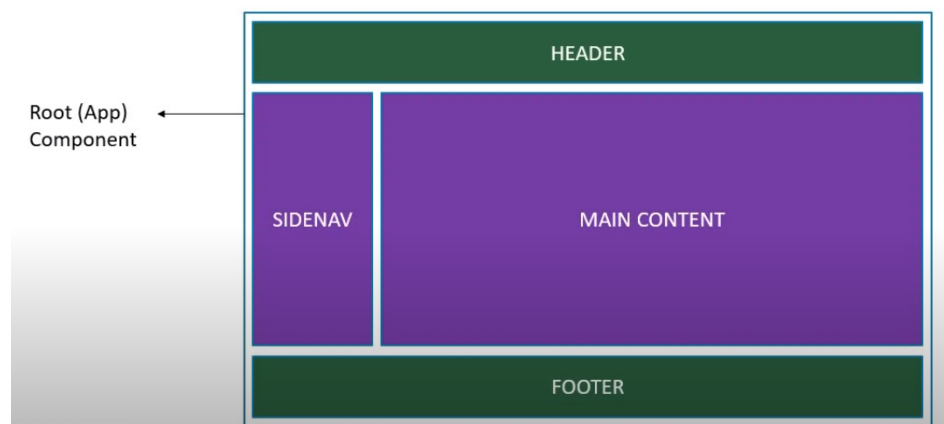
Components:

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

```

Stateless Functional Component

JavaScript Functions

```

function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

```

Stateful Class Component

Class extending Component class
Render method returning HTML

```

class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}

```

Functional Components



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

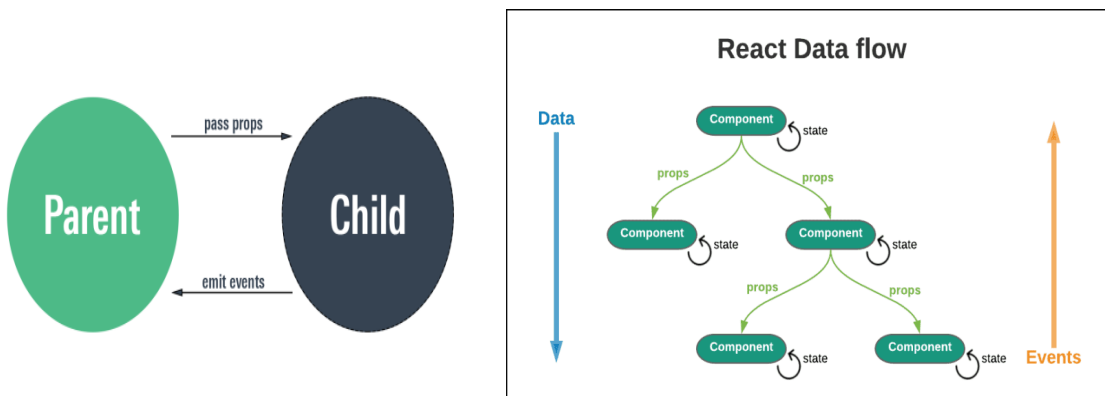
JSX

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

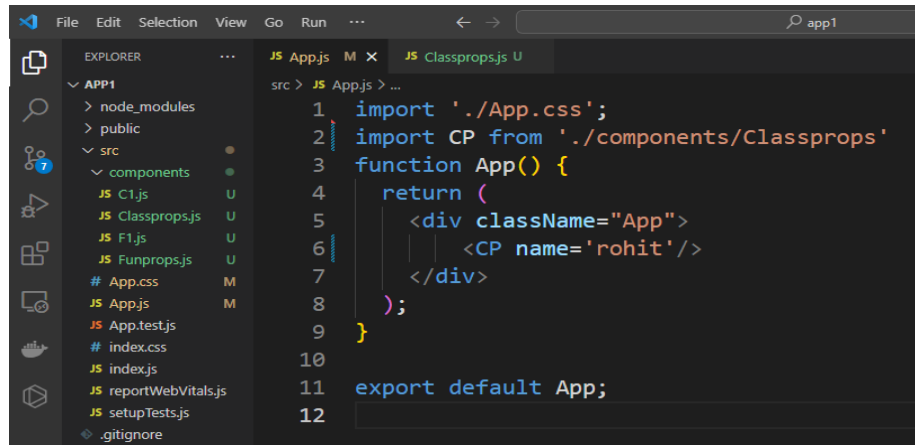
```
File Edit Selection View Go Run ... app1
EXPLORER
  APP1
    > node_modules
    > public
    > src
      > components
        JS C1.js U
        JS Classprops.js U
        JS F1.js U
        JS Funprops.js U
        # App.css M
        JS App.js M
        JS App.test.js
        # index.css
        JS index.js
        JS reportWebVitals.js
        JS setupTests.js
        .gitignore
      JS App.js M
      JS Classprops.js U
      JS Funprops.js U
src > JS App.js > ...
1 import './App.css';
2 import FP from './components/Funprops'
3 function App() {
4   return (
5     <div className="App">
6       <FP name='rohan' />
7     </div>
8   );
9 }
10
11 export default App;
12
```

Funprops.js

```
File Edit Selection View Go Run ... app1
EXPLORER
  APP1
    > node_modules
    > public
    > src
      > components
        JS C1.js U
        JS Classprops.js U
        JS F1.js U
        JS Funprops.js U
        # App.css M
        JS App.js M
        JS App.test.js
        # index.css
      JS Funprops.js U
src > components > JS Funprops.js > ...
1 import React from 'react'
2
3 export default function Funprops(props) {
4   return (
5     <div>{props.name}</div>
6   )
7 }
8
```

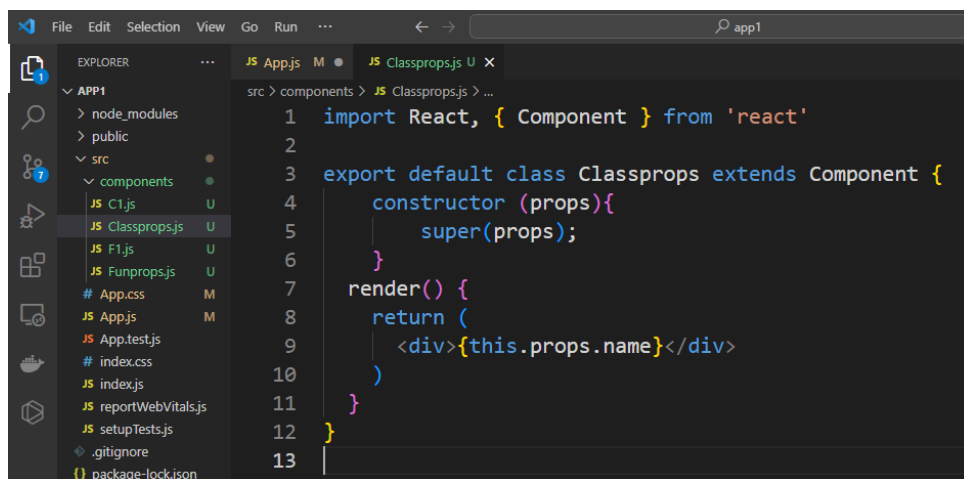
Props in Class components:

App.js

A screenshot of the Visual Studio Code editor. The Explorer sidebar on the left shows a project structure with folders 'node_modules', 'public', and 'src'. Inside 'src', there is a 'components' folder containing 'C1.js', 'Classprops.js', 'F1.js', and 'Funprops.js'. The main editor area shows the 'App.js' file. The code in 'App.js' imports './App.css' and a component 'CP' from './components/Classprops'. It defines a function 'App()' that returns a JSX element: a 'div' with 'className="App"' containing a 'CP' component with 'name="rohit"'. The function is then exported as the default export.

```
1 import './App.css';
2 import CP from './components/Classprops'
3 function App() {
4   return (
5     <div className="App">
6       <CP name="rohit"/>
7     </div>
8   );
9 }
10
11 export default App;
12
```

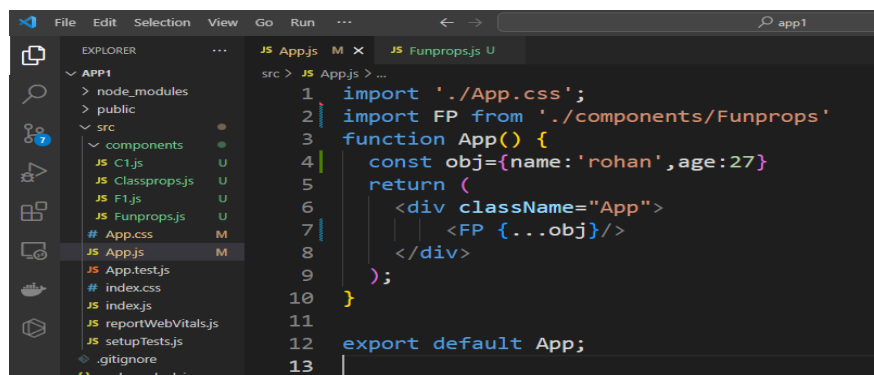
Classprops.js

A screenshot of the Visual Studio Code editor. The Explorer sidebar on the left shows the same project structure as the previous image. The main editor area shows the 'Classprops.js' file. The code imports 'React' and 'Component' from 'react'. It defines a class 'Classprops' that extends 'Component'. The class has a 'constructor' that takes 'props' and calls 'super(props)'. It also has a 'render()' method that returns a JSX element: a 'div' containing 'this.props.name'. The class is exported as the default export.

```
1 import React, { Component } from 'react'
2
3 export default class Classprops extends Component {
4   constructor (props){
5     super(props);
6   }
7   render() {
8     return (
9       <div>{this.props.name}</div>
10     )
11   }
12 }
13
```

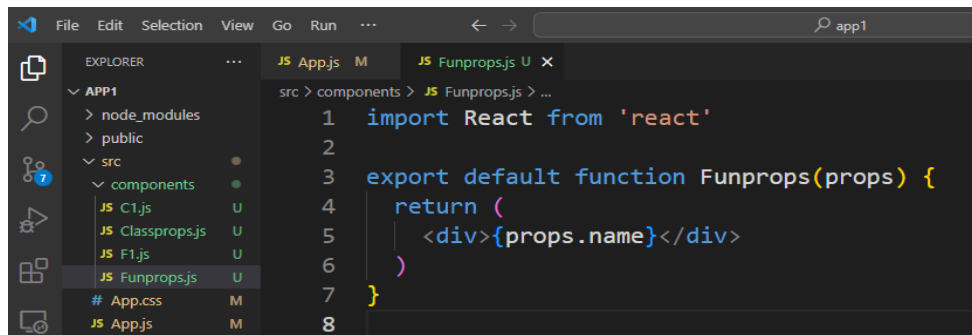
Send obj as props:

App.js

A screenshot of the Visual Studio Code editor. The Explorer sidebar on the left shows the same project structure. The main editor area shows the 'App.js' file. The code imports './App.css' and a component 'FP' from './components/Funprops'. It defines a function 'App()' that creates a constant object 'obj' with 'name: 'rohan'' and 'age: 27'. It then returns a JSX element: a 'div' with 'className="App"' containing an 'FP' component with the spread props '{...obj}'. The function is then exported as the default export.

```
1 import './App.css';
2 import FP from './components/Funprops'
3 function App() {
4   const obj={name: 'rohan',age:27}
5   return (
6     <div className="App">
7       <FP {...obj}/>
8     </div>
9   );
10 }
11
12 export default App;
13
```

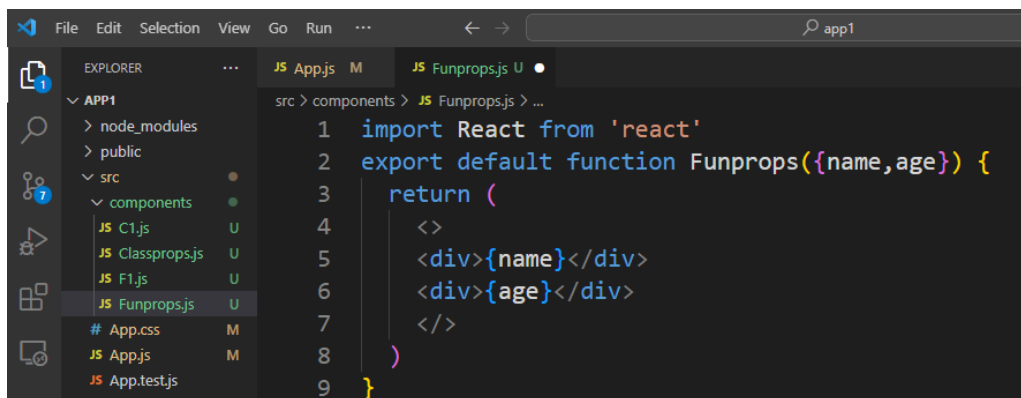
Funprops.js



A screenshot of the Visual Studio Code editor. The Explorer sidebar on the left shows a project structure with 'APP1' containing 'node_modules', 'public', and 'src'. Inside 'src', there is a 'components' folder with files 'C1.js', 'Classprops.js', 'F1.js', and 'Funprops.js'. The 'Funprops.js' file is selected and open in the editor. The code in the editor is as follows:

```
1 import React from 'react'
2
3 export default function Funprops(props) {
4   return (
5     <div>{props.name}</div>
6   )
7 }
8
```

Or (destructured obj)

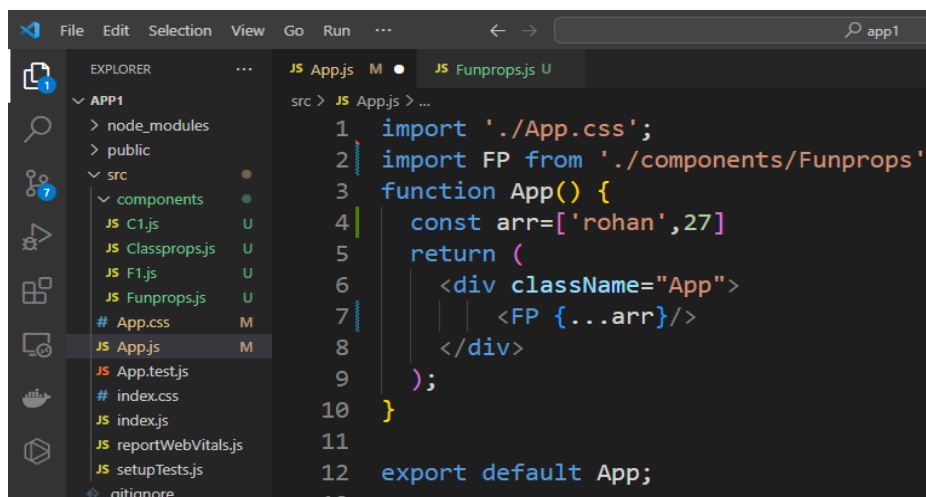


A screenshot of the Visual Studio Code editor showing the 'Funprops.js' file. The code is updated to use destructured props:

```
1 import React from 'react'
2 export default function Funprops({name,age}) {
3   return (
4     <>
5     <div>{name}</div>
6     <div>{age}</div>
7     </>
8   )
9 }
```

Send array as props:

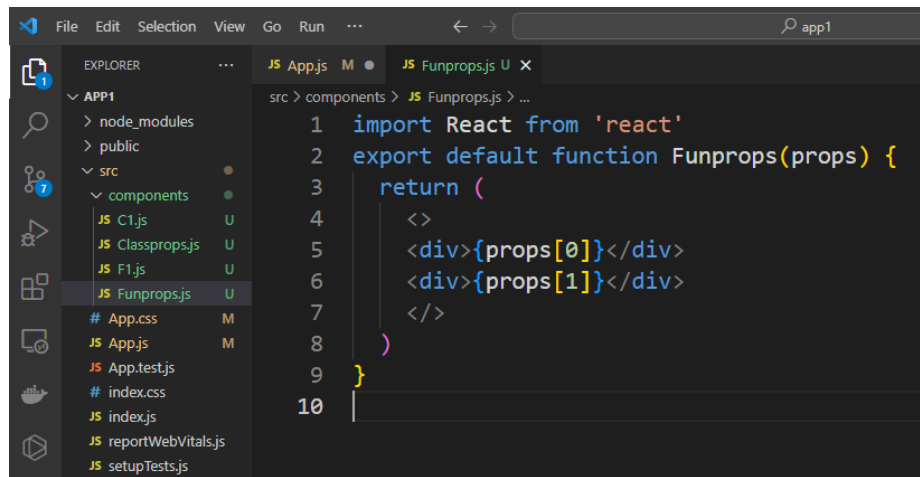
App.js



A screenshot of the Visual Studio Code editor showing the 'App.js' file. The code is as follows:

```
1 import './App.css';
2 import FP from './components/Funprops'
3 function App() {
4   const arr=['rohan',27]
5   return (
6     <div className="App">
7       <FP {...arr}/>
8     </div>
9   );
10 }
11
12 export default App;
```

Funprops.js

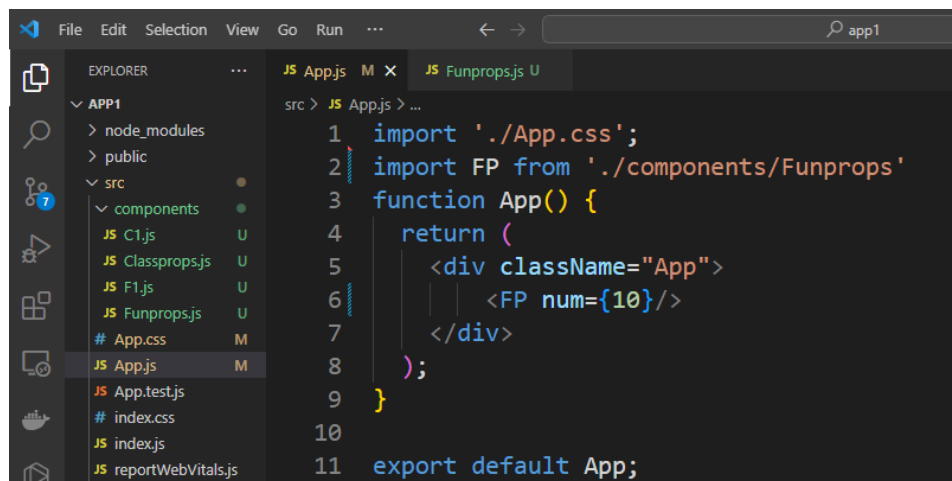


The image shows a VS Code editor window with the Explorer sidebar on the left. The Explorer shows a project structure with a 'src' directory containing a 'components' subdirectory. The 'components' directory contains several files: 'C1.js', 'Classprops.js', 'F1.js', 'Funprops.js', 'App.css', 'App.js', 'App.test.js', 'index.css', 'index.js', 'reportWebVitals.js', and 'setupTests.js'. The 'Funprops.js' file is selected and its content is displayed in the main editor area. The code in 'Funprops.js' is as follows:

```
1 import React from 'react'
2 export default function Funprops(props) {
3   return (
4     <>
5     <div>{props[0]}</div>
6     <div>{props[1]}</div>
7     </>
8   )
9 }
10
```

Send number as props:

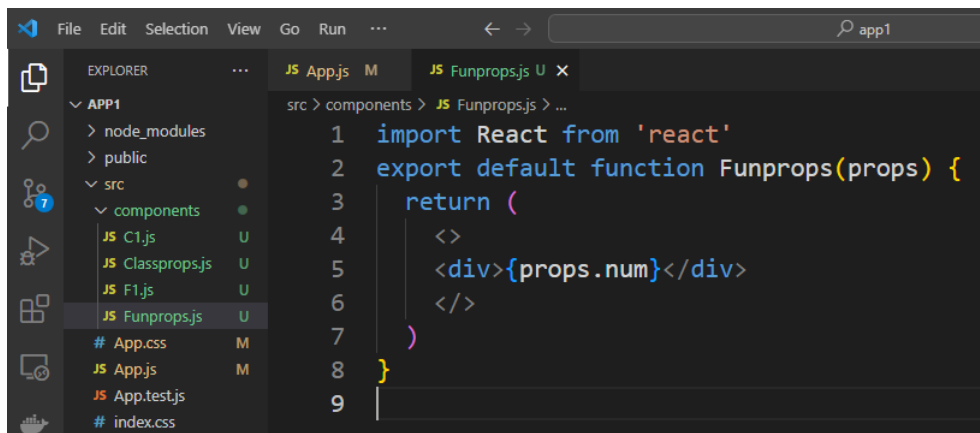
App.js



The image shows a VS Code editor window with the Explorer sidebar on the left. The Explorer shows the same project structure as the previous image. The 'App.js' file is selected and its content is displayed in the main editor area. The code in 'App.js' is as follows:

```
1 import './App.css';
2 import FP from './components/Funprops'
3 function App() {
4   return (
5     <div className="App">
6       <FP num={10}/>
7     </div>
8   );
9 }
10
11 export default App;
```

Funprops.js



The image shows a VS Code editor window with the Explorer sidebar on the left. The Explorer shows the same project structure as the previous images. The 'Funprops.js' file is selected and its content is displayed in the main editor area. The code in 'Funprops.js' is as follows:

```
1 import React from 'react'
2 export default function Funprops(props) {
3   return (
4     <>
5     <div>{props.num}</div>
6     </>
7   )
8 }
9
```

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.

props vs state

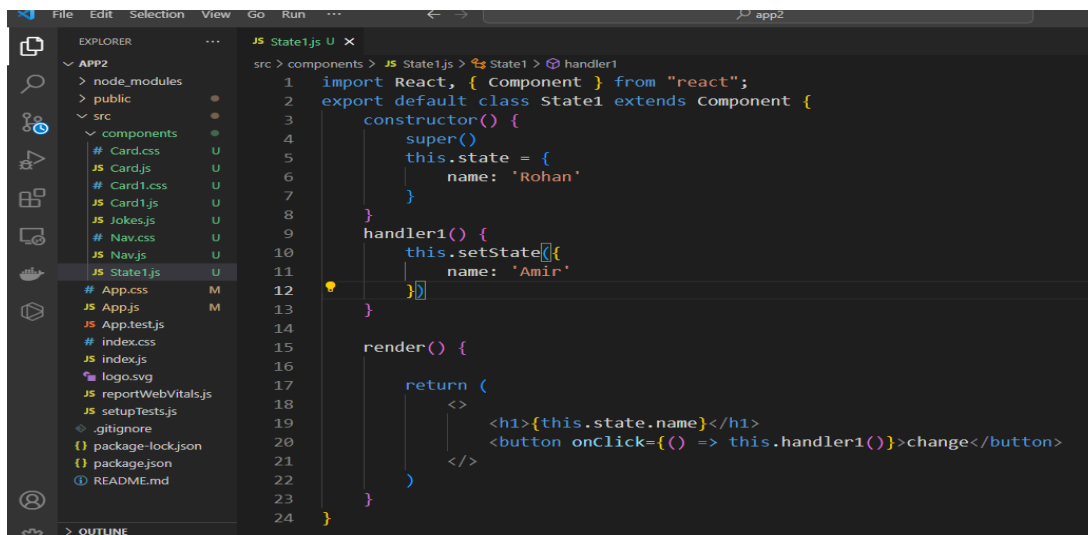
props

- props get passed to the component
- Function parameters
- props are immutable
- props – Functional Components
- this.props – Class Components

state

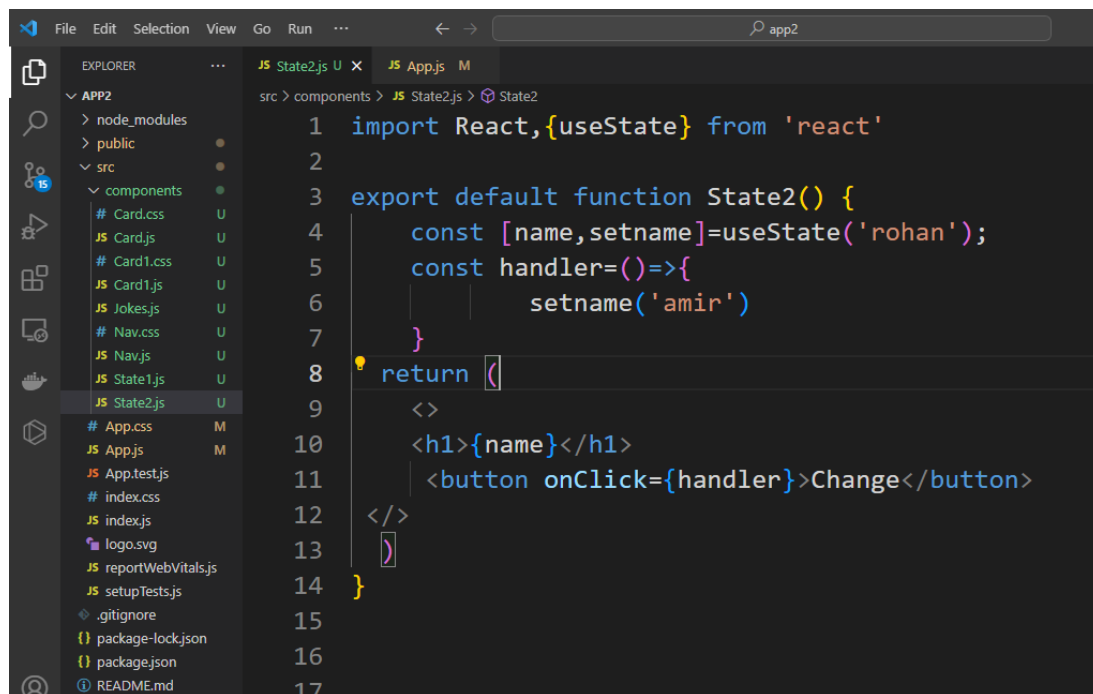
- state is managed within the component
- Variables declared in the function body
- state can be changed
- useState Hook – Functional Components
- this.state – Class Components

State in Class Components:



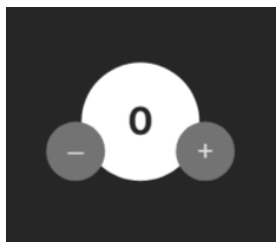
```
1 import React, { Component } from "react";
2 export default class State1 extends Component {
3   constructor() {
4     super();
5     this.state = {
6       name: 'Rohan'
7     }
8   }
9   handler1() {
10    this.setState({
11      name: 'Amir'
12    })
13  }
14  render() {
15    return (
16      <>
17        <h1>{this.state.name}</h1>
18        <button onClick={() => this.handler1()}>change</button>
19      </>
20    )
21  }
22 }
23
24 }
```

State in Functional Components:

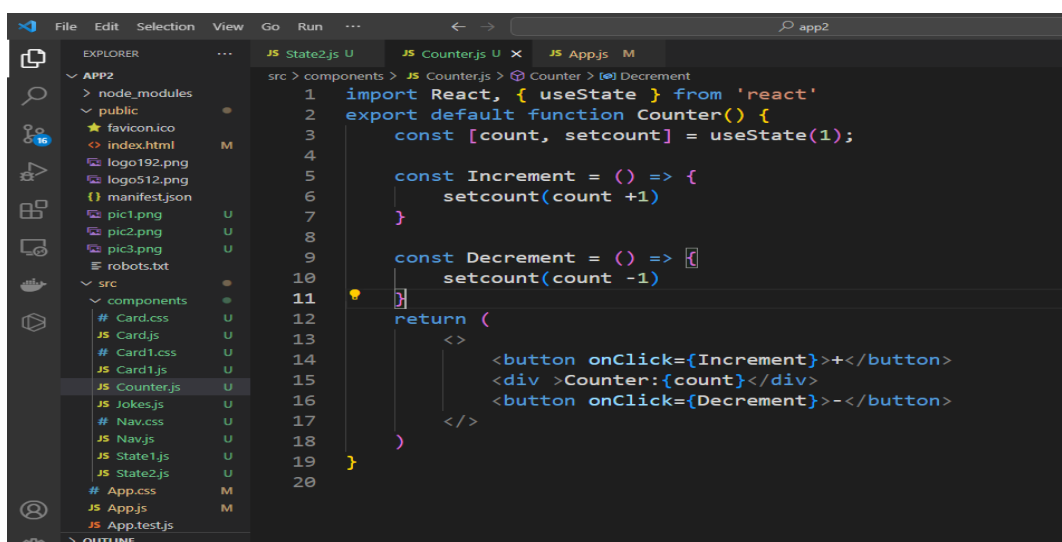


```
1 import React,{useState} from 'react'
2
3 export default function State2() {
4   const [name,setname]=useState('rohan');
5   const handler={()={
6     setname('amir')
7   }}
8   return (
9     <>
10    <h1>{name}</h1>
11    <button onClick={handler}>Change</button>
12  </>
13 )
14 }
```

Counter Challenge

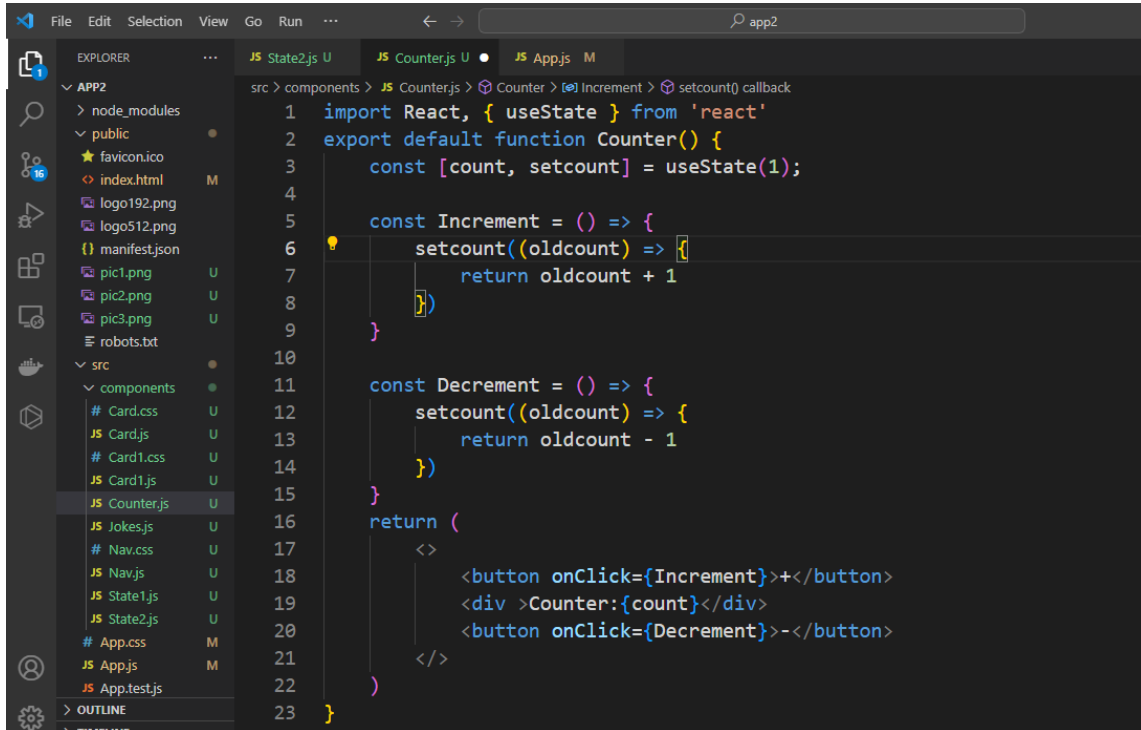


First Method:



```
1 import React, { useState } from 'react'
2 export default function Counter() {
3   const [count, setcount] = useState(1);
4
5   const Increment = () => {
6     setcount(count + 1)
7   }
8
9   const Decrement = () => {
10    setcount(count - 1)
11  }
12  return (
13    <>
14    <button onClick={Increment}>+</button>
15    <div>Counter:{count}</div>
16    <button onClick={Decrement}>-</button>
17  </>
18 )
19 }
```

Second Method:

A screenshot of a code editor interface, likely VS Code, showing a React component file named Counter.js. The Explorer sidebar on the left shows a project structure with folders like 'public' and 'src', and various files including images, JSON, and JavaScript files. The main editor area displays the following code:

```
1 import React, { useState } from 'react'
2 export default function Counter() {
3   const [count, setcount] = useState(1);
4
5   const Increment = () => {
6     setcount((oldcount) => {
7       return oldcount + 1
8     })
9   }
10
11  const Decrement = () => {
12    setcount((oldcount) => {
13      return oldcount - 1
14    })
15  }
16  return (
17    <>
18      <button onClick={Increment}>+</button>
19      <div>Counter:{count}</div>
20      <button onClick={Decrement}>-</button>
21    </>
22  )
23 }
```

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.