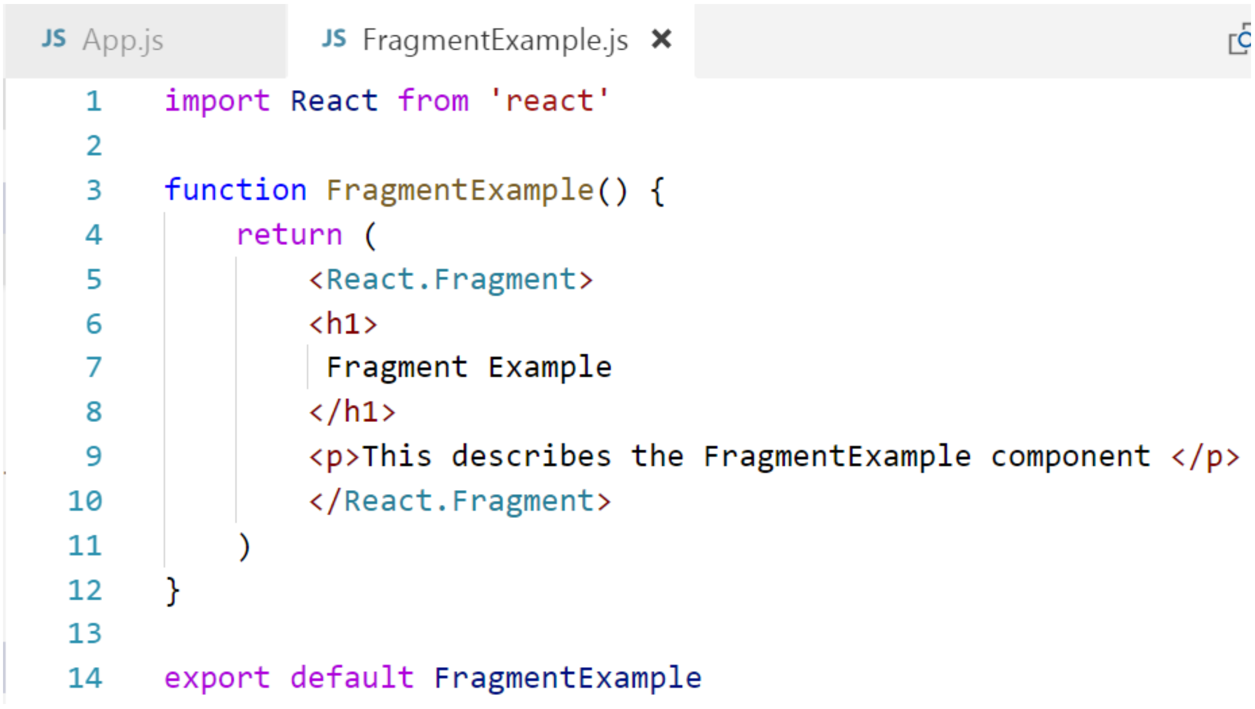
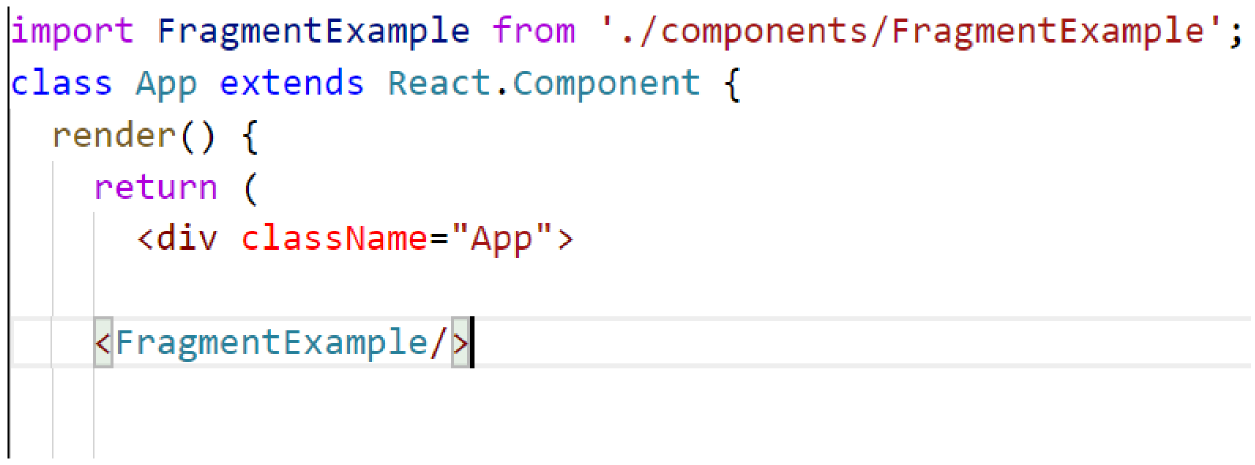
Fragments:

A common pattern in React is for a component to return multiple elements. **Fragments let you group a list of children without adding extra nodes to the DOM.**



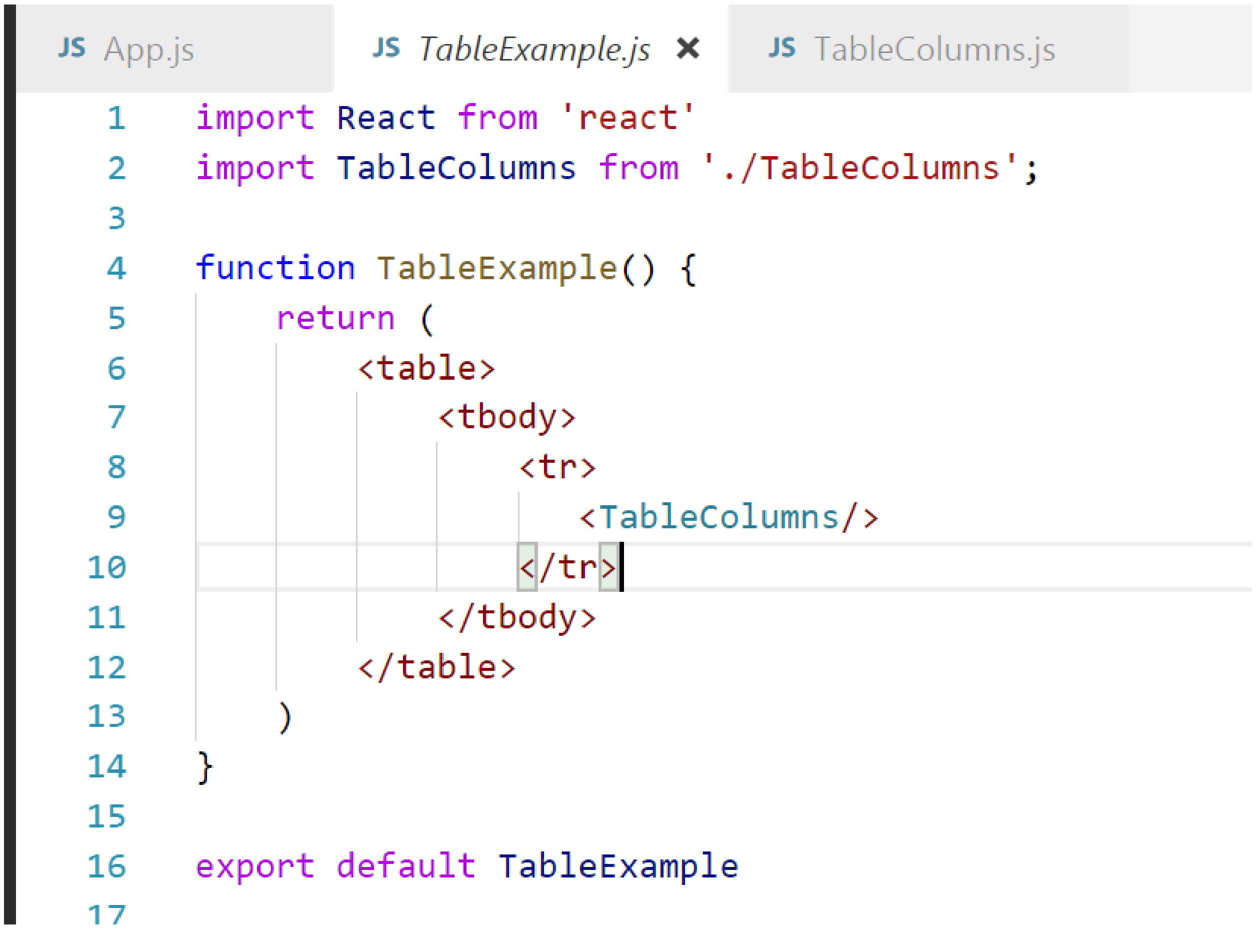
```
import React from 'react'

function FragmentExample() {
    return (
        <React.Fragment>
        <h1>
         Fragment Example
        </h1>
        <p>This describes the FragmentExample component </p>
        </React.Fragment>
    )
}

export default FragmentExample
```

App.js



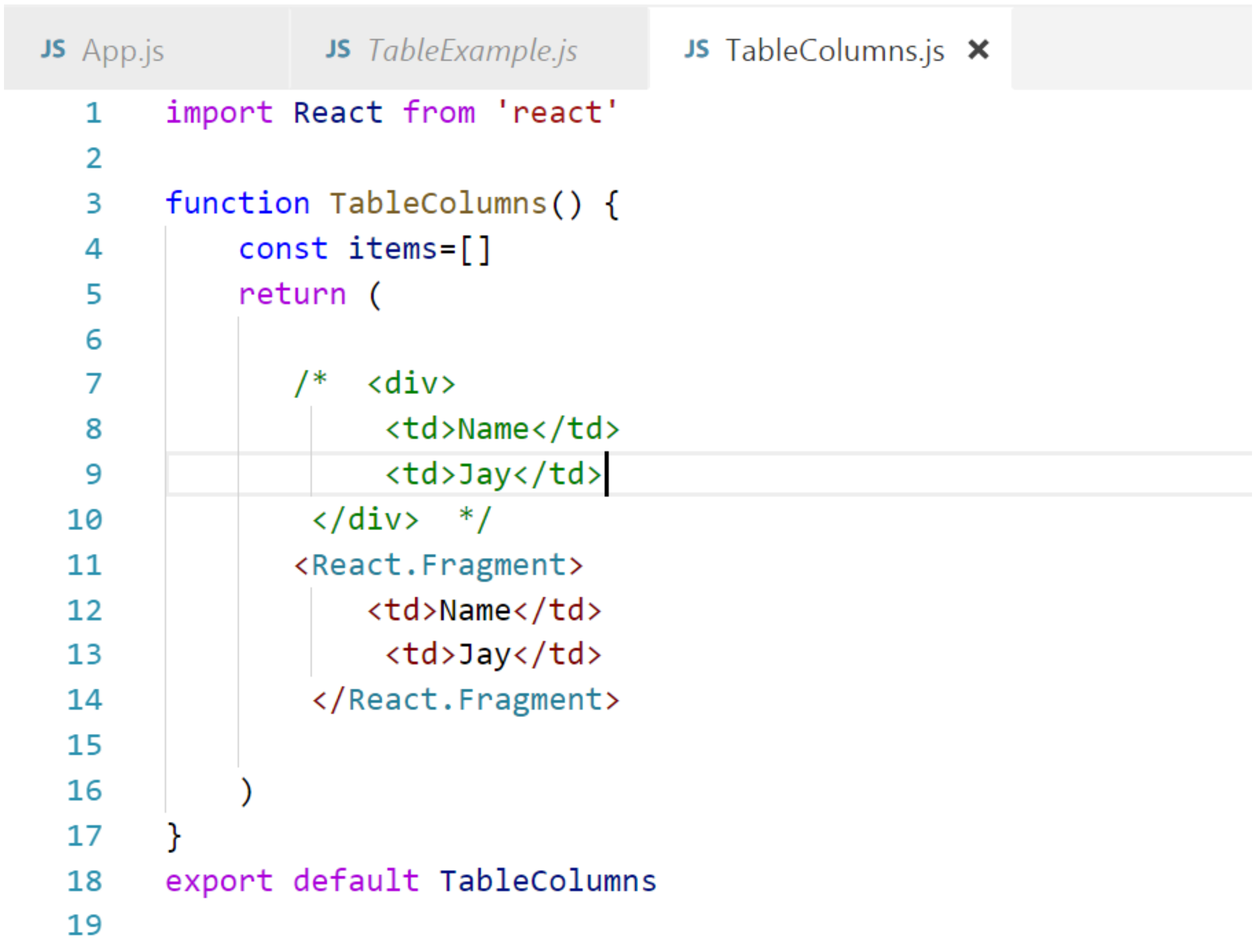
```
import FragmentExample from './components/FragmentExample';
class App extends React.Component {
  render() {
    return (
      <div className="App">

    <FragmentExample/>
```

TableExample.js


JS App.js JS TableExample.js ✖ JS TableColumns.js

```
import React from 'react'
import TableColumns from './TableColumns';

function TableExample() {
    return (
        <table>
            <tbody>
                <tr>
                    <TableColumns/>
                </tr>
            </tbody>
        </table>
    )
}

export default TableExample

```

TableColumns.js



```
import React from 'react'

function TableColumns() {
    const items=[]
    return (

       /*  <div>
             <td>Name</td>
             <td>Jay</td>
        </div>  */
       <React.Fragment>
           <td>Name</td>
            <td>Jay</td>
        </React.Fragment>

    )
}
export default TableColumns

```

App.js

<TableExample/>

Pure Components:

Pure Components are introduced for performance enhancement. You can use this optimization to improve the performance of your components.

The major difference between React.PureComponent and React.Component is PureComponent does a shallow comparison on state change. It means that when comparing scalar values it compares their values, but when comparing objects it compares only references. It helps to improve the performance of the app

You should go for React.PureComponent when you can satisfy any of the below conditions.

- State/Props should be an immutable object
- State/Props should not have a hierarchy
- You should call forceUpdate when data changes

If you are using React.PureComponent you should make sure all child components are also pure.

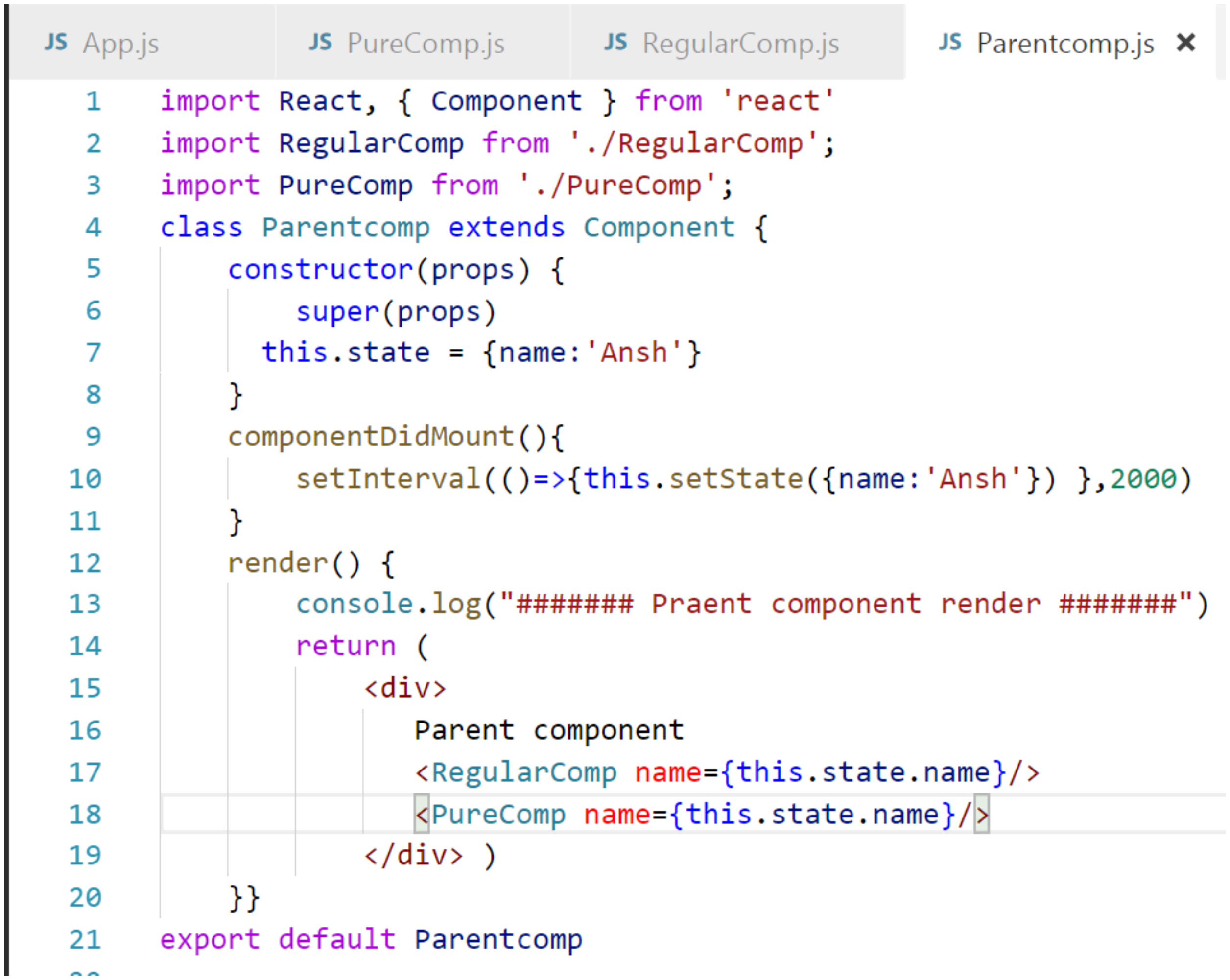
PureComponent is exactly the same as Component except that it handles the shouldComponentUpdate method for you.
When props or state changes, PureComponent will do a *shallow comparison* on both props and state. Component on the other hand won't compare current props and state to next out of the box. Thus, the component will re-render by default whenever shouldComponentUpdate is called.
Shallow Comparison

When comparing previous props and state to next, a shallow comparison will check that primitives have the same value (eg, 1 equals 1 or that true equals true) and that the references are the same between more complex javascript values like objects and arrays.

React does the shallow comparisons of current state and props with new props and state to decide whether to continue with next update cycle or not.

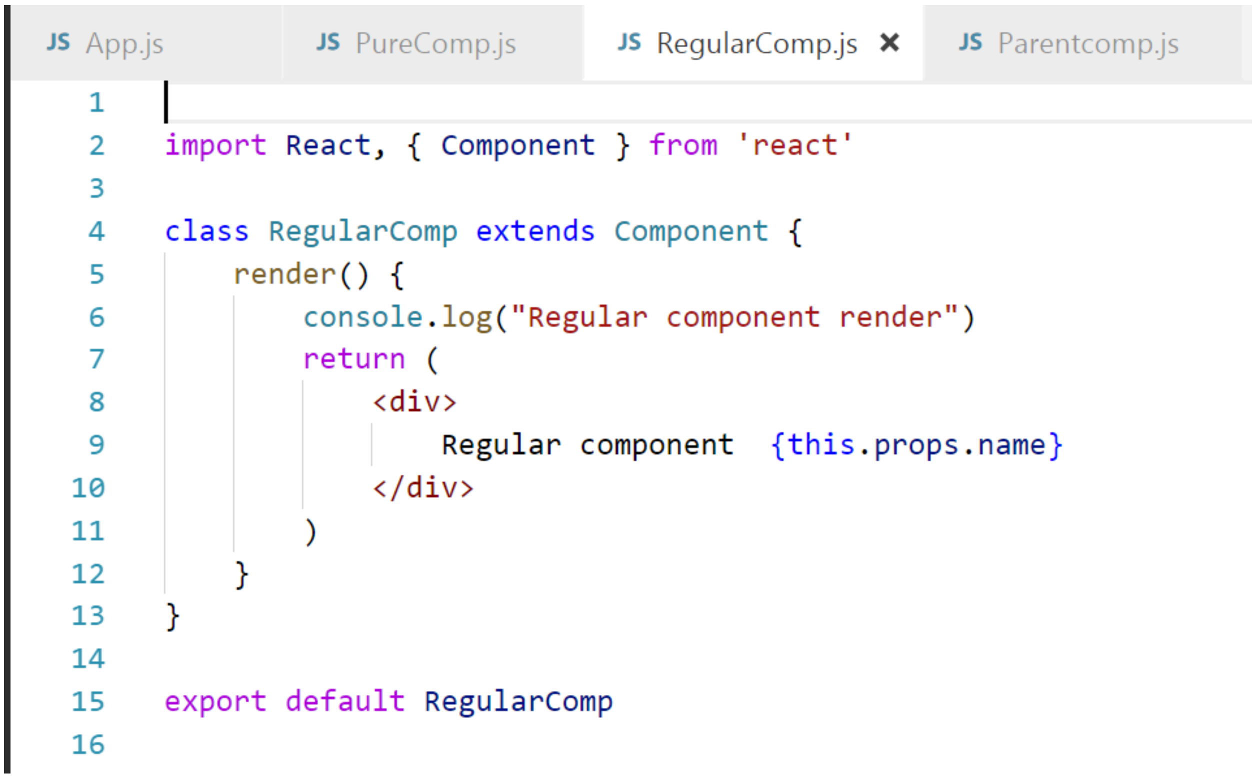
Example:

Parentcomp.js



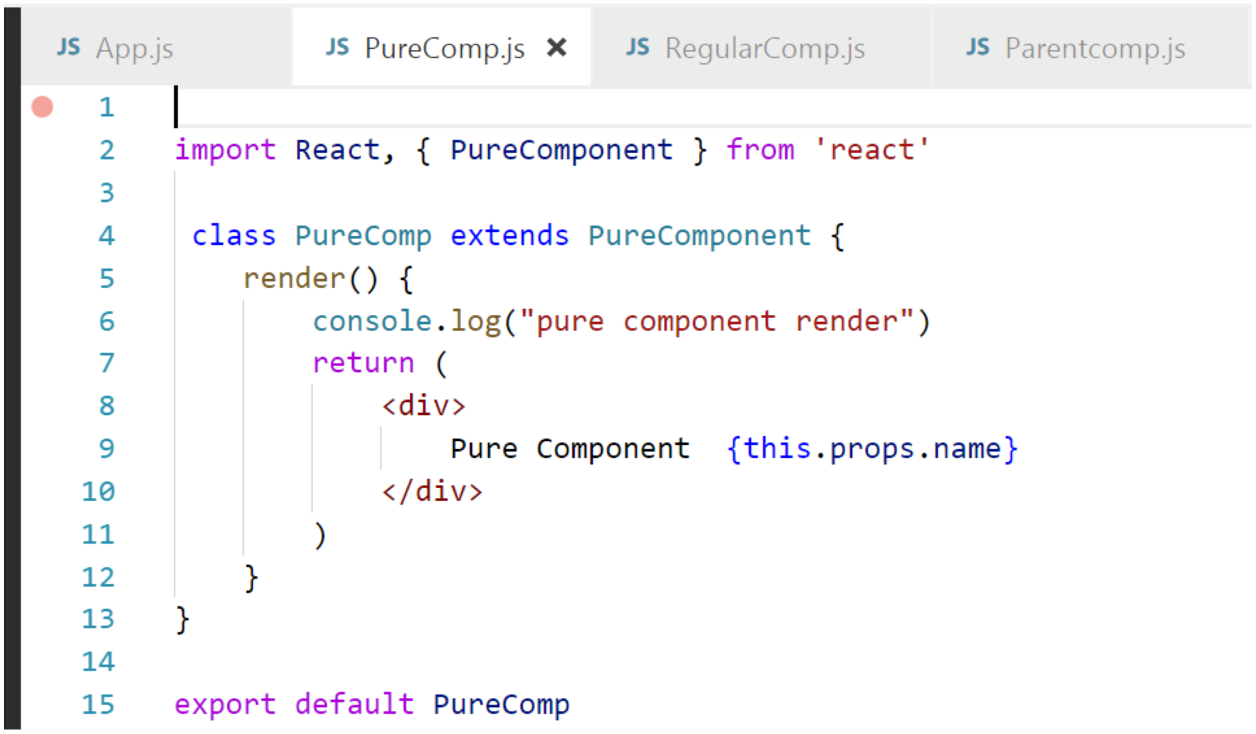
```
import React, { Component } from 'react'
import RegularComp from './RegularComp';
import PureComp from './PureComp';
class Parentcomp extends Component {
    constructor(props) {
        super(props)
      this.state = {name:'Ansh'}
    }
    componentDidMount(){
        setInterval(()=>{this.setState({name:'Ansh'}) },2000)
    }
    render() {
        console.log("####### Praent component render #######")
        return (
            <div>
               Parent component
               <RegularComp name={this.state.name}/>
               <PureComp name={this.state.name}/>
            </div> )
    }}
export default Parentcomp
```

Regularcomp.js



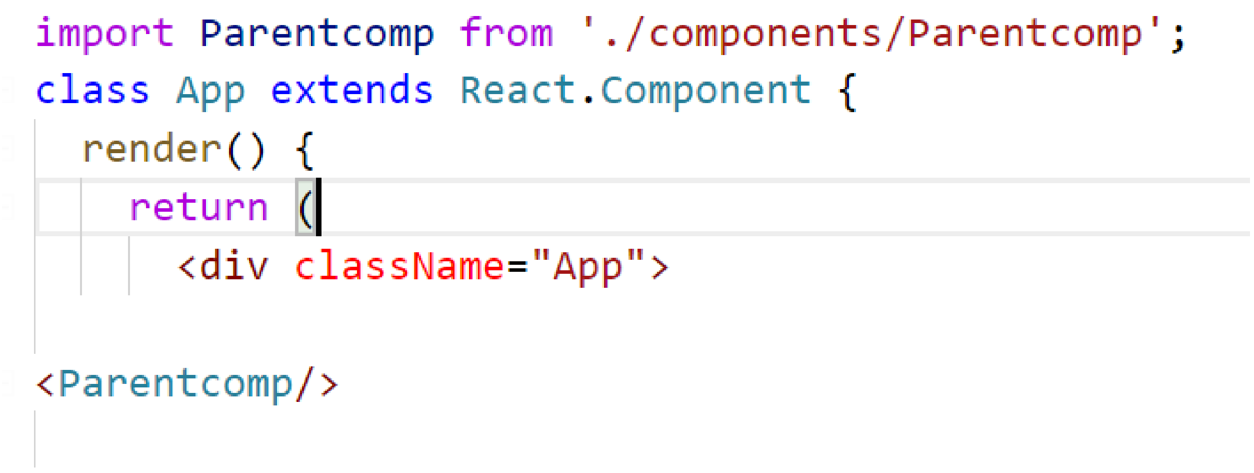
```

import React, { Component } from 'react'

class RegularComp extends Component {
    render() {
        console.log("Regular component render")
        return (
            <div>
                Regular component  {this.props.name}
            </div>
        )
    }
}

export default RegularComp

```

Purecomp.js



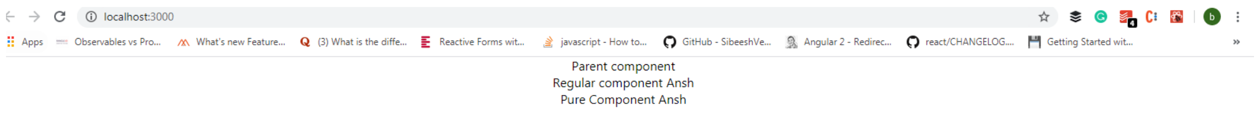
```

import React, { PureComponent } from 'react'

 class PureComp extends PureComponent {
    render() {
        console.log("pure component render")
        return (
            <div>
                Pure Component  {this.props.name}
            </div>
        )
    }
}

export default PureComp
```

App.js



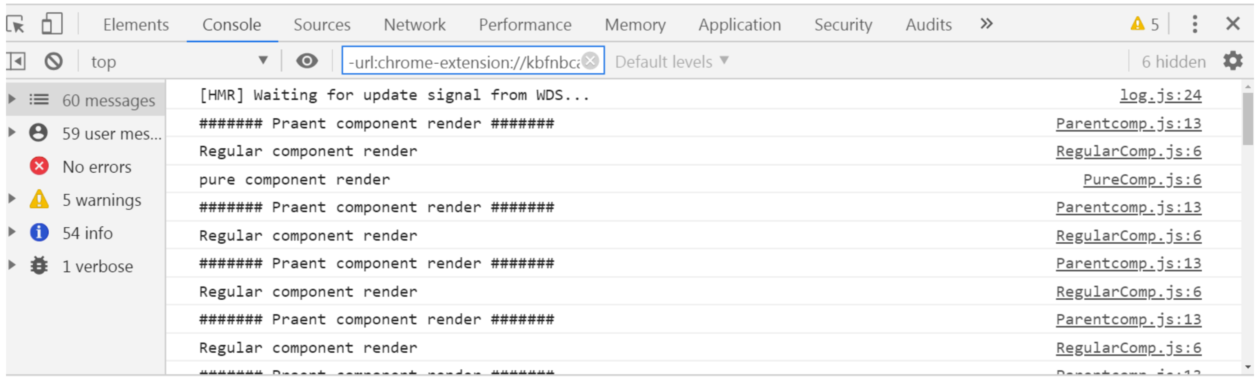
```
import Parentcomp from './components/Parentcomp';
class App extends React.Component {
  render() {
    return (
      <div className="App">

<Parentcomp/>
```

Output:



localhost:3000

```
Parent component
Regular component Ansh
Pure Component Ansh
```



```
[HMR] Waiting for update signal from WDS...          log.js:24
####### Praent component render #######              Parentcomp.js:13
Regular component render                             RegularComp.js:6
pure component render                                PureComp.js:6
####### Praent component render #######              Parentcomp.js:13
Regular component render                             RegularComp.js:6
####### Praent component render #######              Parentcomp.js:13
Regular component render                             RegularComp.js:6
####### Praent component render #######              Parentcomp.js:13
Regular component render                             RegularComp.js:6
```

Above example parent component and regular component will update every 2 second

Memo (memoization**):**

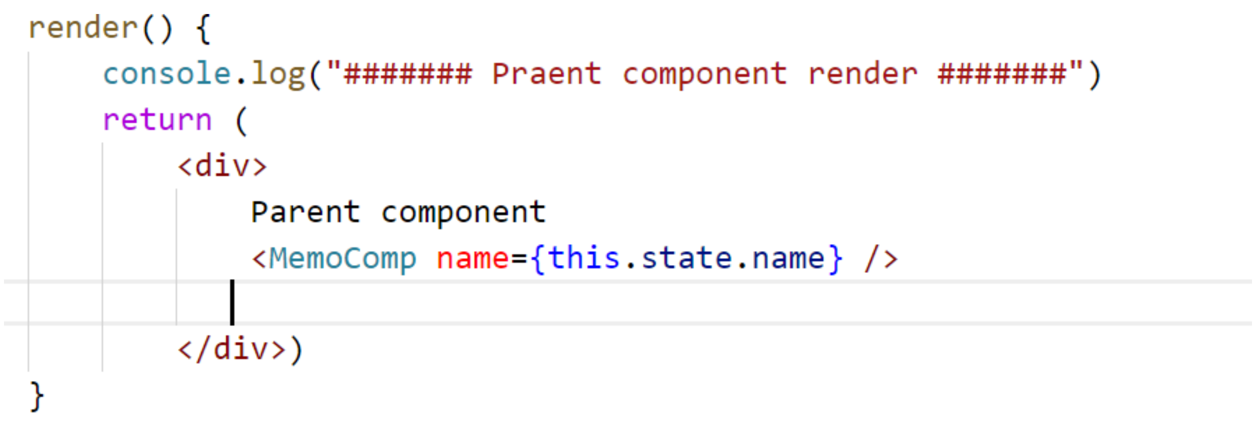
PureComponent works with **classes**. React.memo() works with **functional components**.

React.memo() is similar to PureComponent in that it will **help us control when our components rerender**.

In computing, memoization or memoisation is an optimization technique used primarily to speed up computer programs by storing the results of expensive **function** calls and returning the cached result when the same **inputs** occur again.

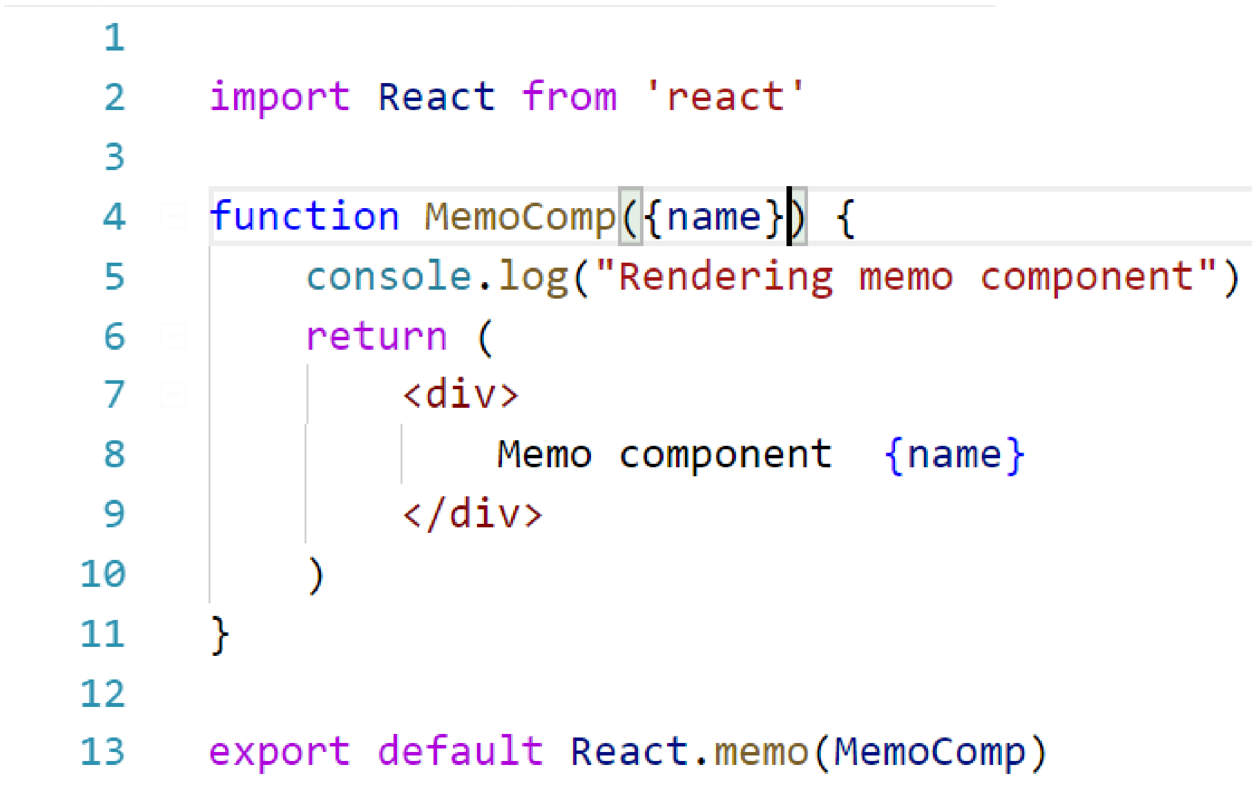
React.memo() was introduce in version 16.6

Example: In above parentcomp.js will change in render()-



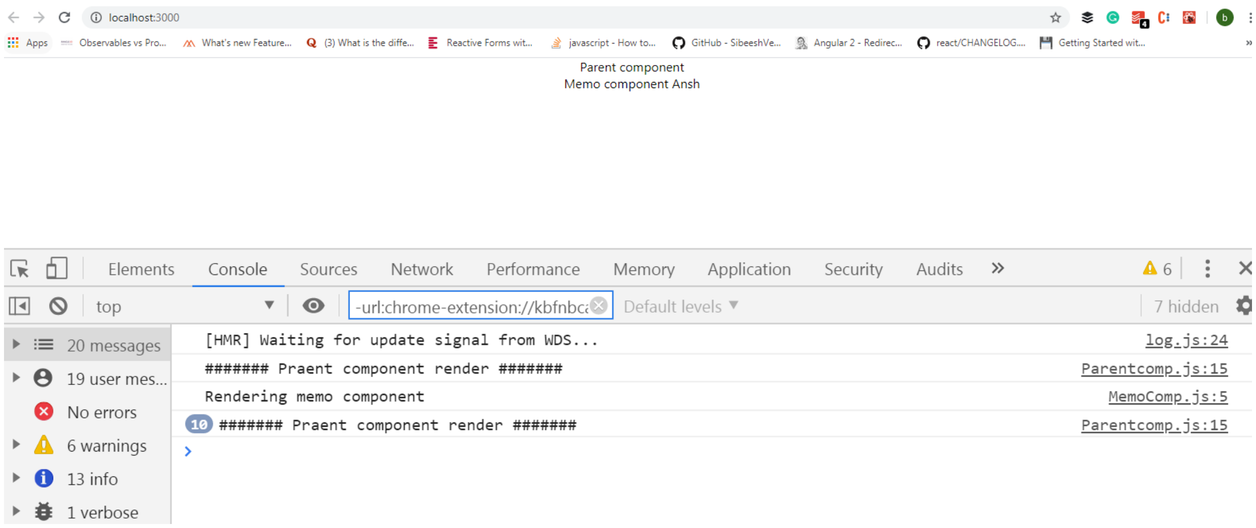
```
render() {
    console.log("####### Praent component render #######")
    return (
        <div>
            Parent component
            <MemoComp name={this.state.name} />

        </div>)
}
```

memoComp.js



```

import React from 'react'

function MemoComp({name}) {
    console.log("Rendering memo component")
    return (
        <div>
            Memo component  {name}
        </div>
    )
}

export default React.memo(MemoComp)
```

Output:



Parent component
Memo component Ansh

```
[HMR] Waiting for update signal from WDS...                    log.js:24
####### Praent component render #######                  Parentcomp.js:15
Rendering memo component                                   MemoComp.js:5
10 ####### Praent component render #######               Parentcomp.js:15
```

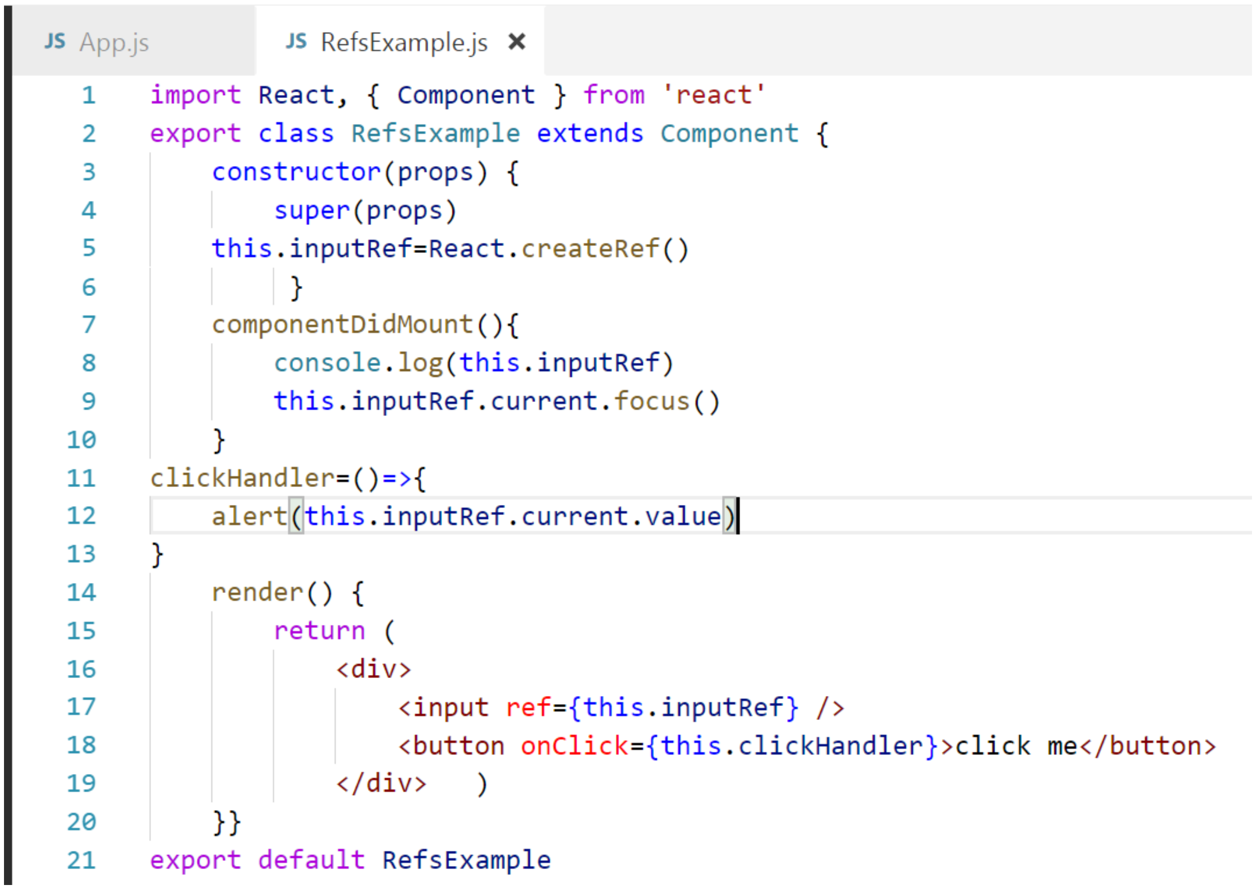
Refs:

Refs make it possible to access DOM nodes directly within React.

React provides a way to get references to DOM nodes by using React.createRef()

Example: Focusing an <input>

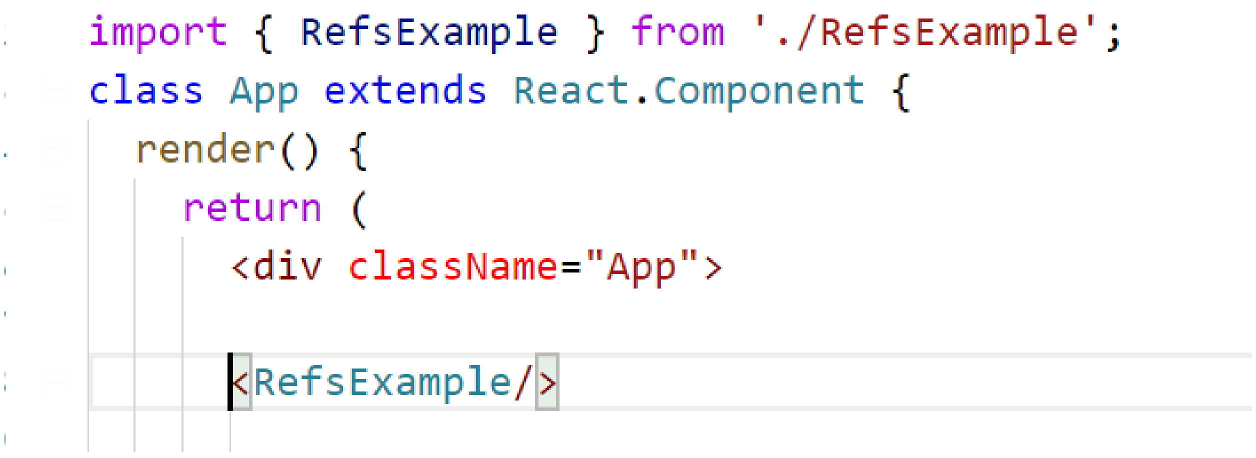
We could start interacting with the <input> DOM node



JS App.js　JS RefsExample.js

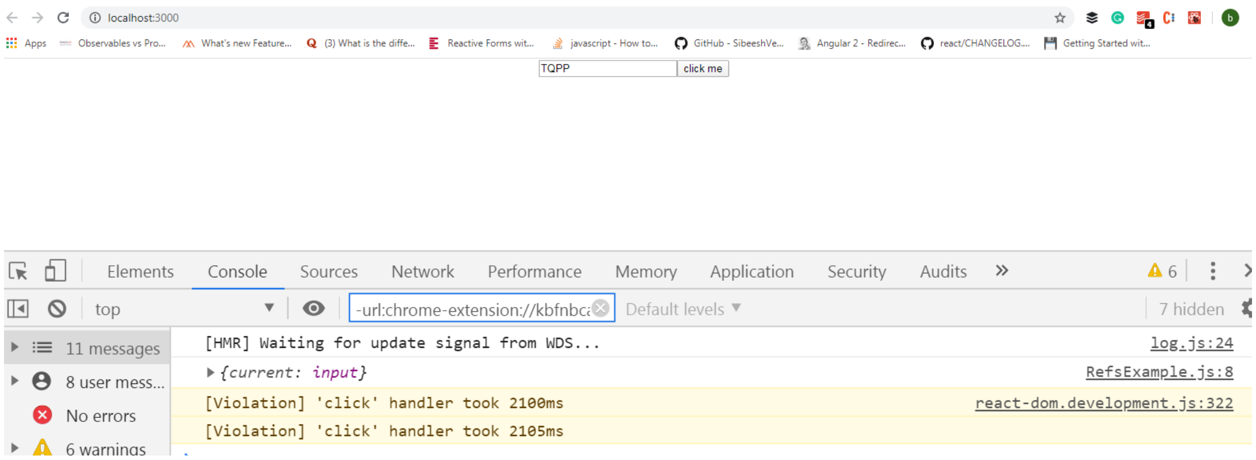
```
import React, { Component } from 'react'
export class RefsExample extends Component {
    constructor(props) {
        super(props)
    this.inputRef=React.createRef()
          }
    componentDidMount(){
        console.log(this.inputRef)
        this.inputRef.current.focus()
    }
clickHandler=()=>{
    alert(this.inputRef.current.value)
}
    render() {
        return (
            <div>
                <input ref={this.inputRef} />
                <button onClick={this.clickHandler}>click me</button>
            </div>    )
    }}
export default RefsExample
```

App.js



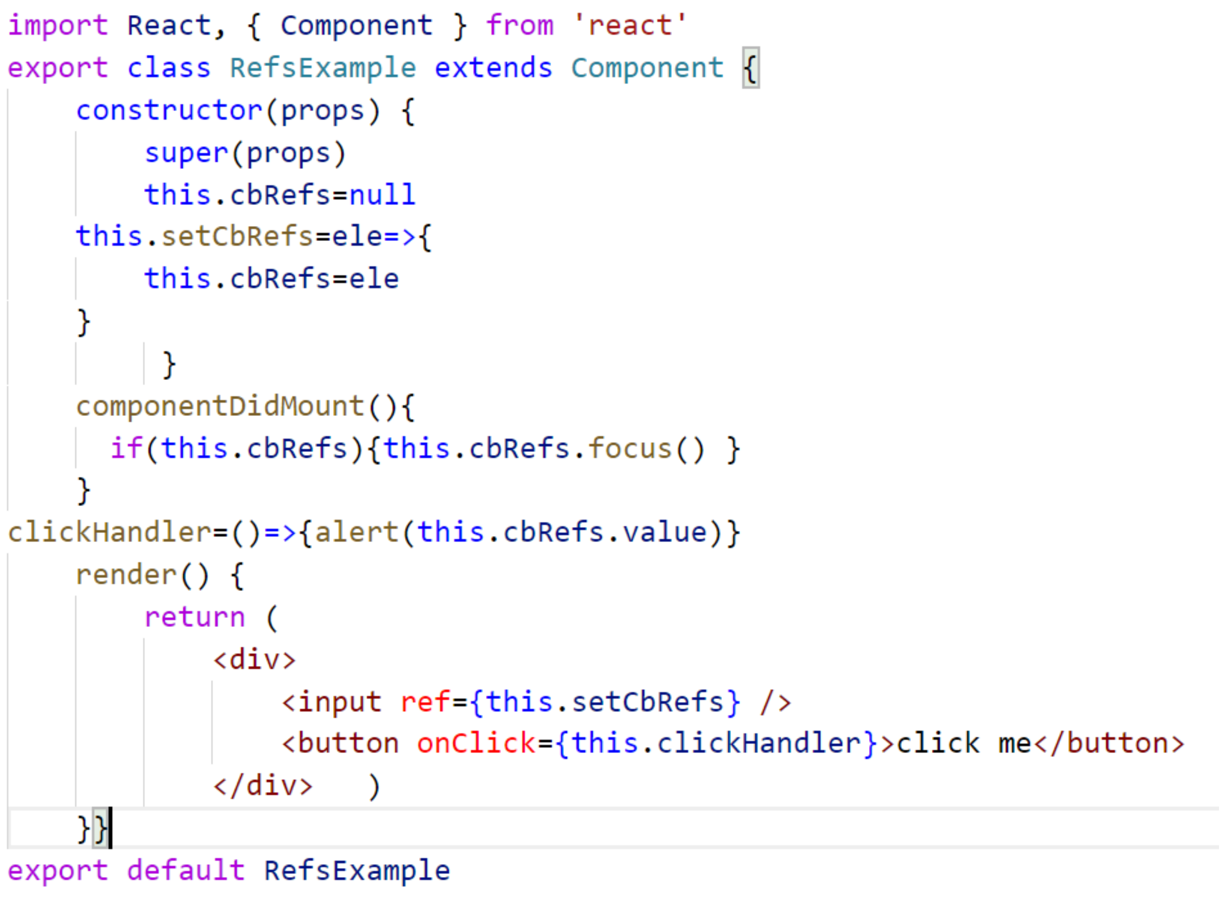
```
import { RefsExample } from './RefsExample';
class App extends React.Component {
  render() {
    return (
      <div className="App">

      <RefsExample/>
```

Output:



```
[HMR] Waiting for update signal from WDS...                              log.js:24
▸{current: input}                                                 RefsExample.js:8
[Violation] 'click' handler took 2100ms                   react-dom.development.js:322
[Violation] 'click' handler took 2105ms
```

Another way to set Refs which is called as **callback Refs**

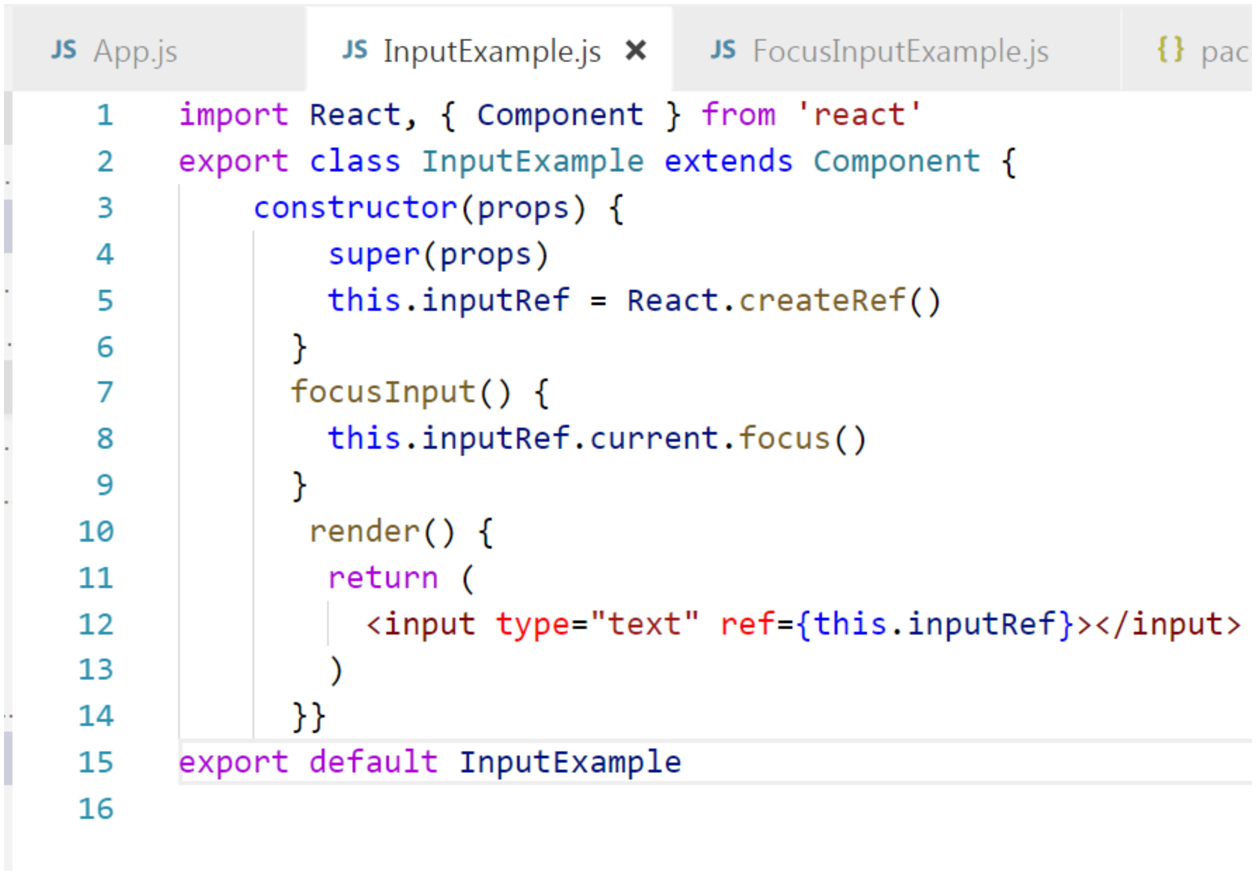


```
import React, { Component } from 'react'
export class RefsExample extends Component {
    constructor(props) {
        super(props)
        this.cbRefs=null
    this.setCbRefs=ele=>{
        this.cbRefs=ele
    }
          }
    componentDidMount(){
      if(this.cbRefs){this.cbRefs.focus() }
    }
clickHandler=()=>{alert(this.cbRefs.value)}
    render() {
        return (
            <div>
                <input ref={this.setCbRefs} />
                <button onClick={this.clickHandler}>click me</button>
            </div>    )
    }}
export default RefsExample
```

Refs with Class Components:

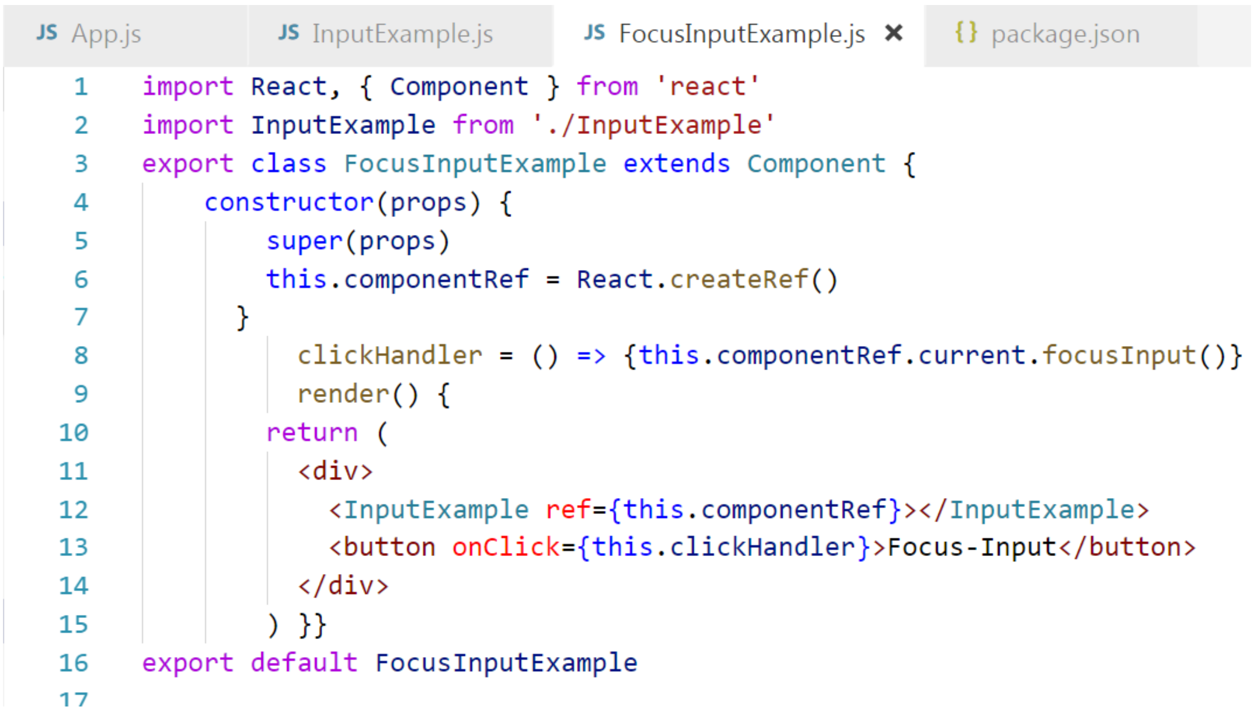
Example

InputExample.js



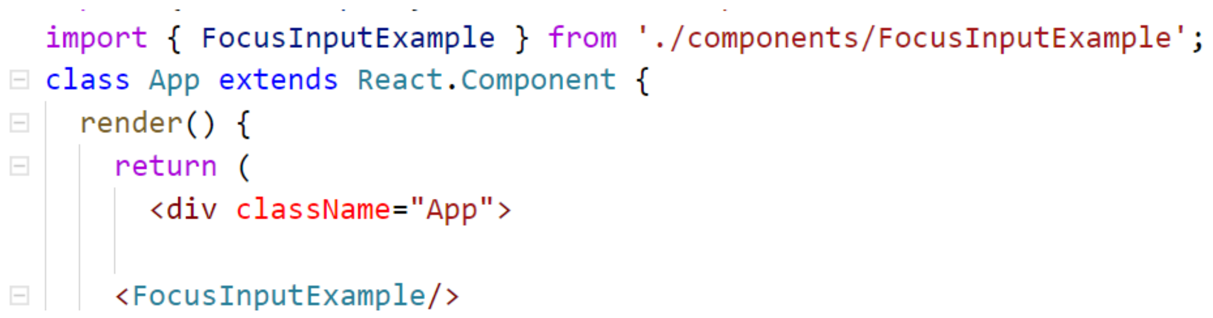
```
import React, { Component } from 'react'
export class InputExample extends Component {
    constructor(props) {
      super(props)
      this.inputRef = React.createRef()
   }
   focusInput() {
     this.inputRef.current.focus()
   }
    render() {
     return (
       <input type="text" ref={this.inputRef}></input>
     )
   }}
export default InputExample
```

FocusInputExample.js


App.js InputExample.js FocusInputExample.js package.json

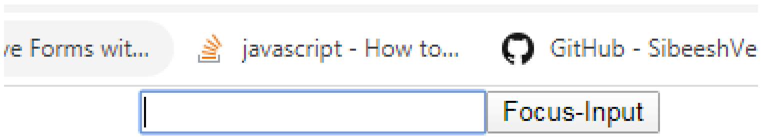
```
import React, { Component } from 'react'
import InputExample from './InputExample'
export class FocusInputExample extends Component {
    constructor(props) {
        super(props)
        this.componentRef = React.createRef()
      }
        clickHandler = () => {this.componentRef.current.focusInput()}
        render() {
        return (
        <div>
          <InputExample ref={this.componentRef}></InputExample>
          <button onClick={this.clickHandler}>Focus-Input</button>
        </div>
        ) }}
export default FocusInputExample
```

App.js



```
import { FocusInputExample } from './components/FocusInputExample';
class App extends React.Component {
  render() {
    return (
      <div className="App">

    <FocusInputExample/>
```

Output:



When to Use Refs

There are a few good use cases for refs:

- Managing focus, text selection, or media playback.
- Triggering imperative animations.
- Integrating with third-party DOM libraries.

Avoid using refs for anything that can be done declaratively.
For example, instead of exposing open() and close() methods on
a Dialog component, pass an isOpen prop to it.

You may not use the ref attribute on function components because they don't have instances.

Forwarding Refs

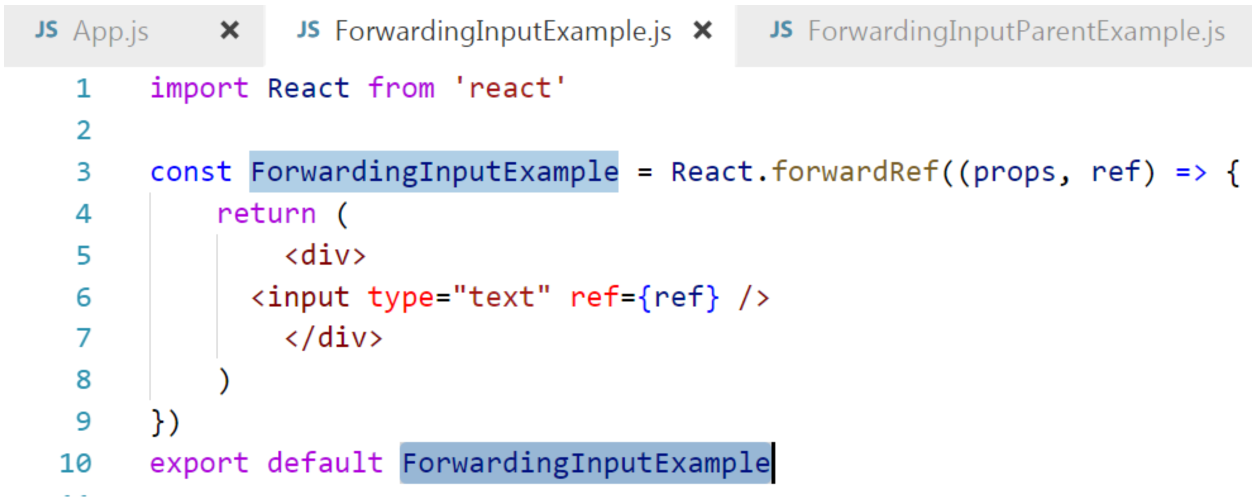
Ref forwarding is a technique for automatically passing a **ref** through a component to one of its children. This is typically not necessary for most components in the application. However, it can be useful for some kinds of components, especially in reusable component libraries.

Ref forwarding is a technique to automatically pass a ref to a child component, allowing the parent component to access that child component's element and read or modify it in some way.

React provide us with extra boilerplate specifically for ref forwarding whereby we wrap a component with React.forwardRef()

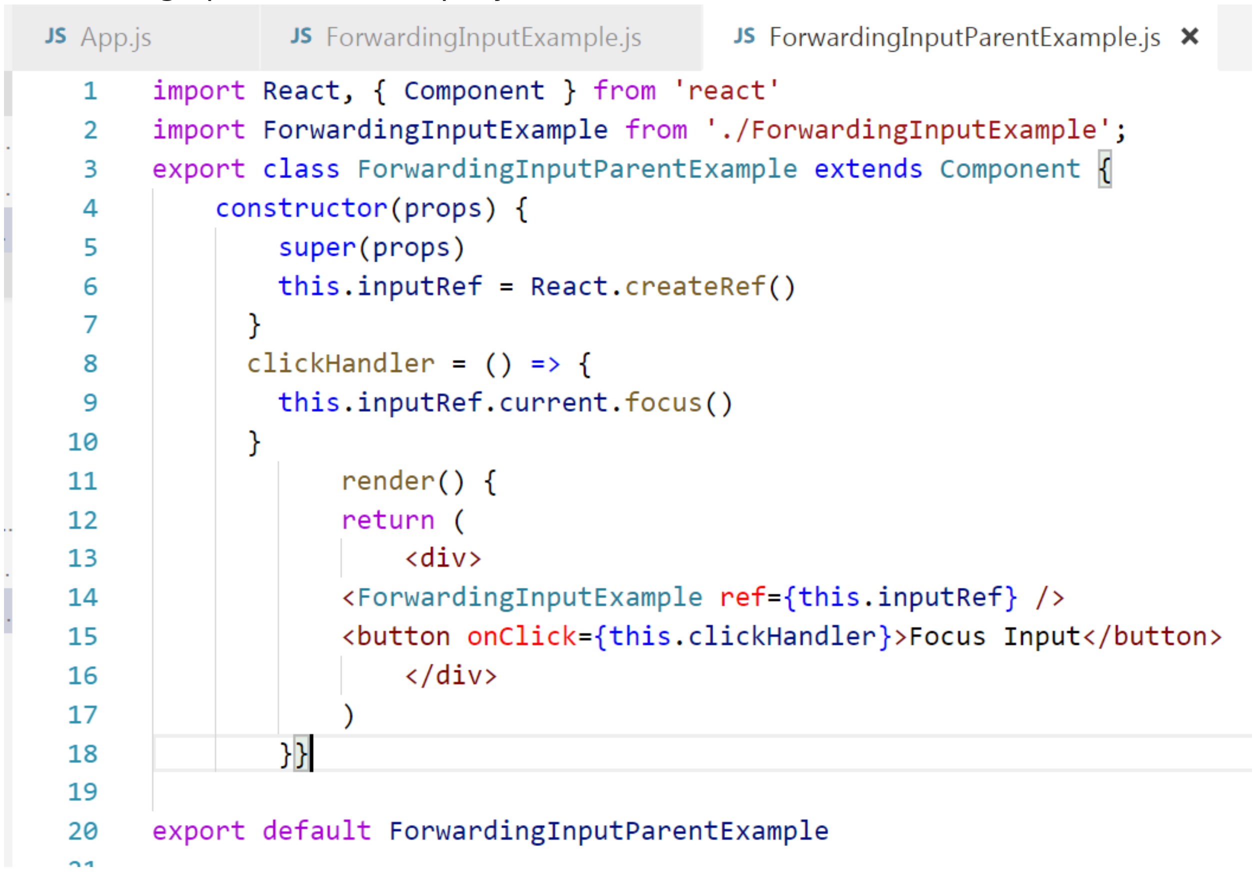
Example:

ForwardingInputExample.js


JS App.js ✖ JS ForwardingInputExample.js ✖ JS ForwardingInputParentExample.js

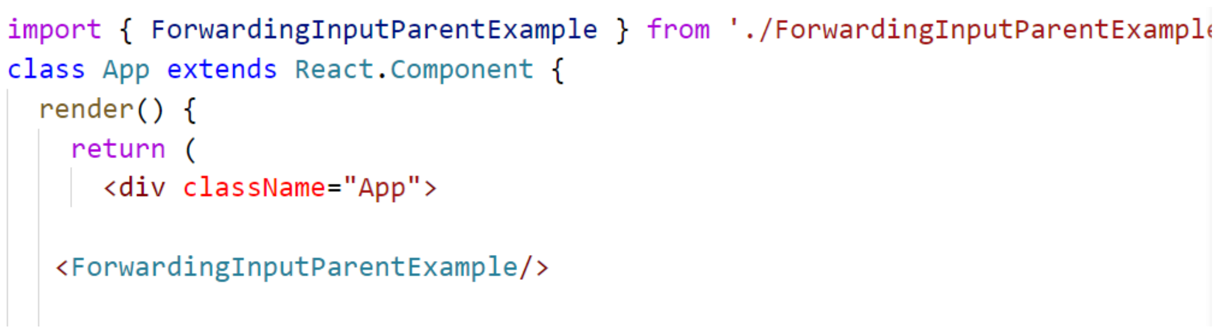
```
import React from 'react'

const ForwardingInputExample = React.forwardRef((props, ref) => {
    return (
        <div>
      <input type="text" ref={ref} />
        </div>
    )
})
export default ForwardingInputExample
```

ForwardingInputParentExample.js


JS App.js JS ForwardingInputExample.js JS ForwardingInputParentExample.js ✖

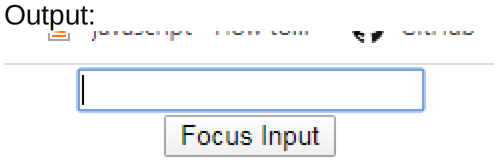
```
import React, { Component } from 'react'
import ForwardingInputExample from './ForwardingInputExample';
export class ForwardingInputParentExample extends Component {
    constructor(props) {
        super(props)
        this.inputRef = React.createRef()
      }
      clickHandler = () => {
        this.inputRef.current.focus()
      }
            render() {
            return (
                <div>
            <ForwardingInputExample ref={this.inputRef} />
            <button onClick={this.clickHandler}>Focus Input</button>
                </div>
            )
        }}

export default ForwardingInputParentExample
```

App.js



```
import { ForwardingInputParentExample } from './ForwardingInputParentExampl
class App extends React.Component {
  render() {
    return (
      <div className="App">

   <ForwardingInputParentExample/>
```

Output:



Portals:

Portals provide a way to render children into a DOM node that exists outside the DOM hierarchy of the parent component i.e., in a separate component.

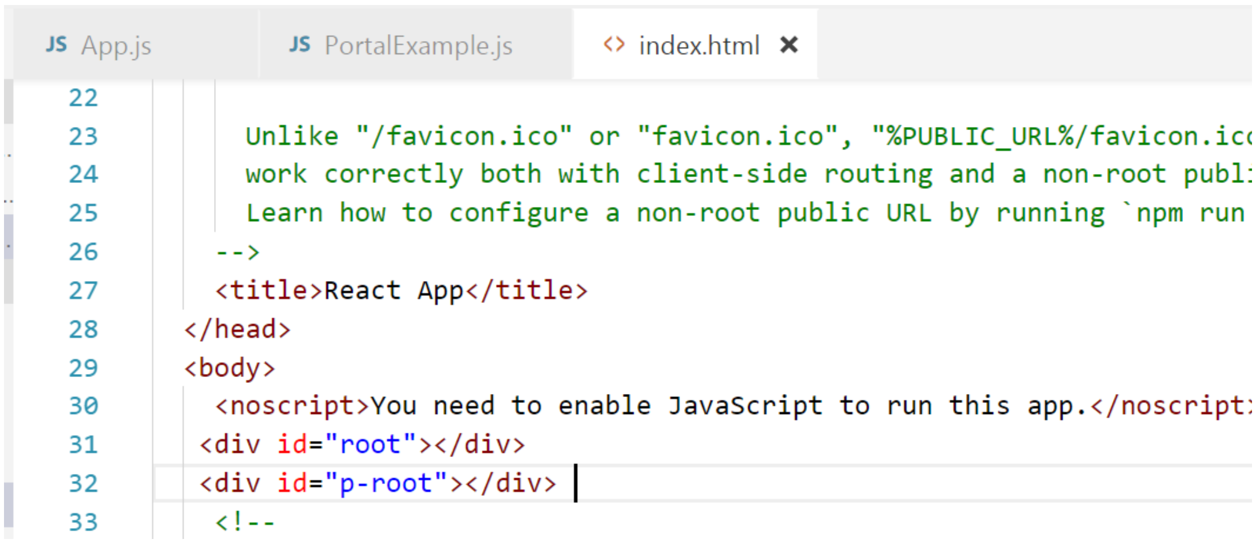
In React, portals can be used to render an element outside of its parent component’s DOM node while preserving its position in the React hierarchy, allowing it to maintain the properties and behaviors it inherited from the React tree.

When to use?

- Modals
- Tooltips
- Floating menus
- Widgets

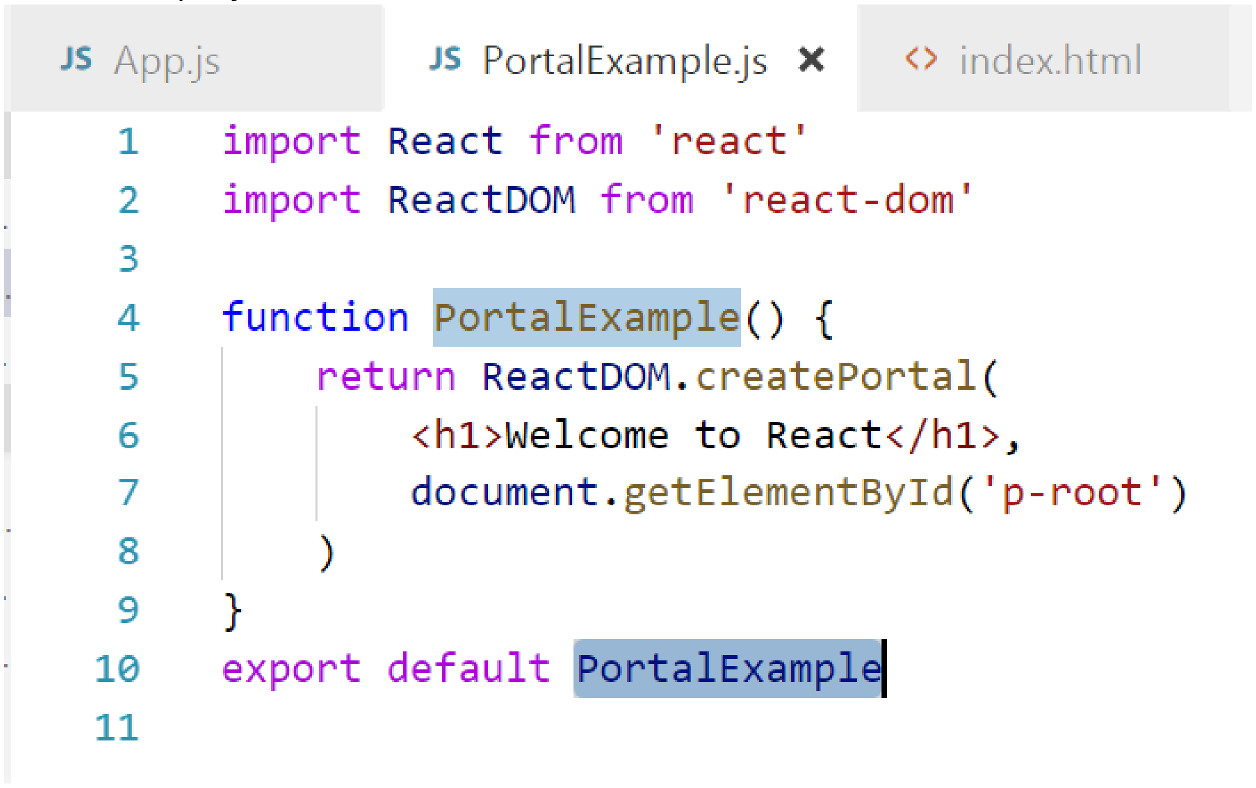
Example:

Index.html



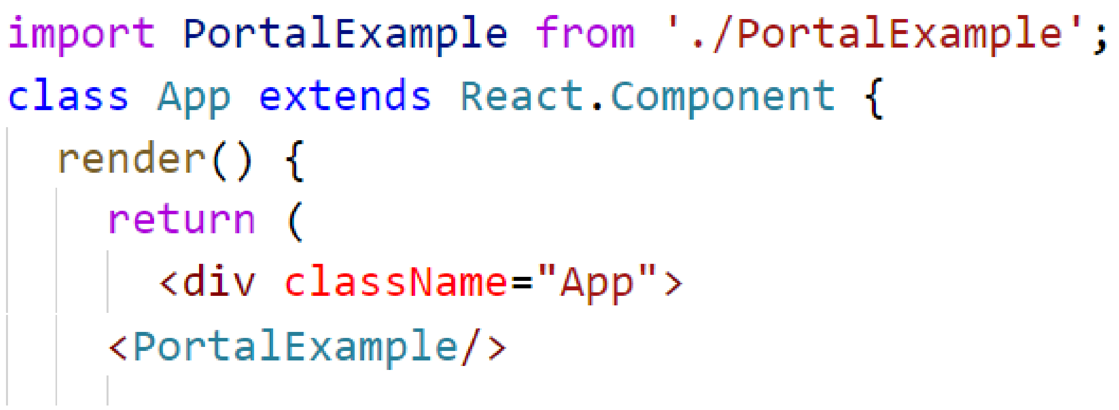
```
        Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ic
        work correctly both with client-side routing and a non-root publ
        Learn how to configure a non-root public URL by running `npm run
      -->
      <title>React App</title>
    </head>
    <body>
      <noscript>You need to enable JavaScript to run this app.</noscript
     <div id="root"></div>
     <div id="p-root"></div>
      <!--
```

PortalExample.js



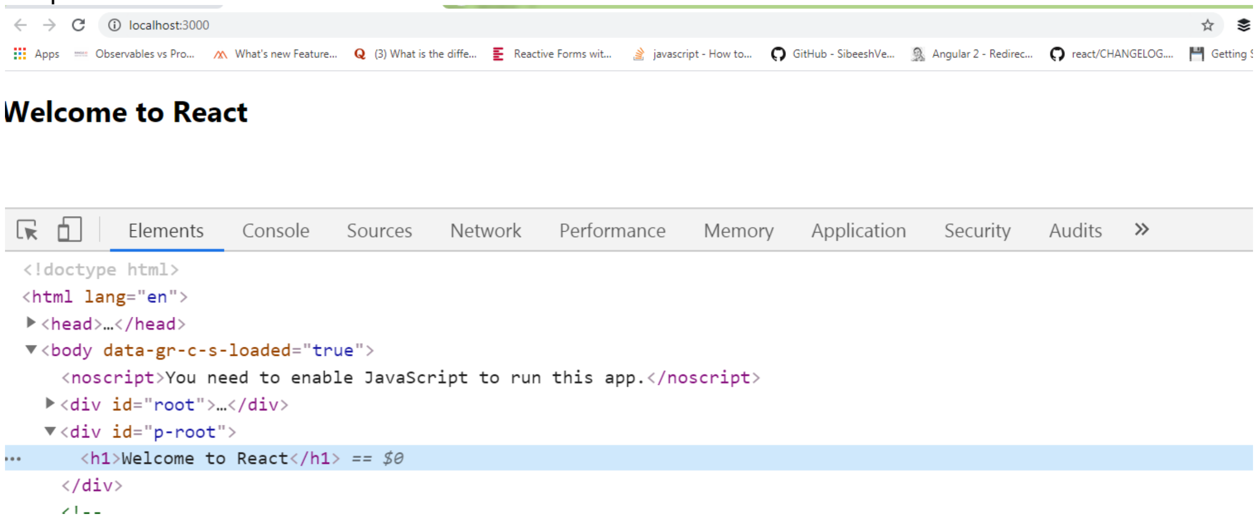
```
import React from 'react'
import ReactDOM from 'react-dom'

function PortalExample() {
    return ReactDOM.createPortal(
        <h1>Welcome to React</h1>,
        document.getElementById('p-root')
    )
}
export default PortalExample
```

App.js



```
import PortalExample from './PortalExample';
class App extends React.Component {
  render() {
    return (
      <div className="App">
    <PortalExample/>
```

Output:



Welcome to React

```
<!doctype html>
<html lang="en">
▶<head>…</head>
▼<body data-gr-c-s-loaded="true">
    <noscript>You need to enable JavaScript to run this app.</noscript>
  ▶<div id="root">…</div>
  ▼<div id="p-root">
      <h1>Welcome to React</h1> == $0
    </div>
    <!--
```

Error Boundary:

A JavaScript error in a part of the UI shouldn't break the whole app. To solve this problem using an "error boundary".
Error boundaries are React components that **catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI** instead of the component tree that crashed. Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.

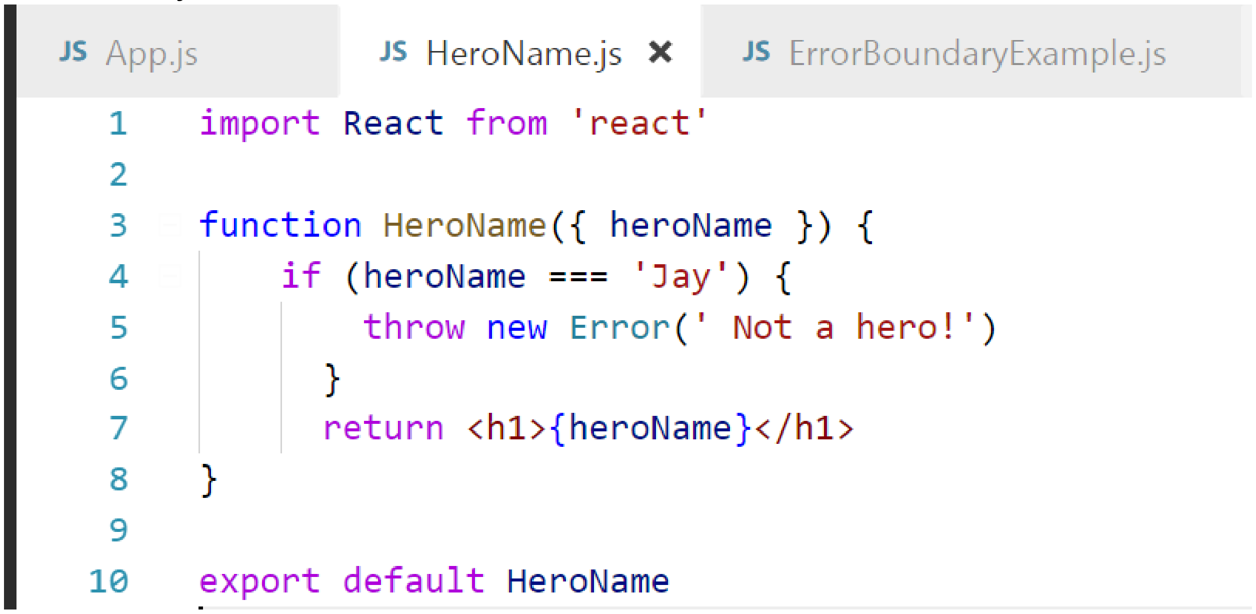
Note
Error boundaries do **not** catch errors for:

- Event handlers (learn more)
- Asynchronous code (e.g. setTimeout or requestAnimationFrame callbacks)
- Server side rendering

- Errors thrown in the error boundary itself (rather than its children)

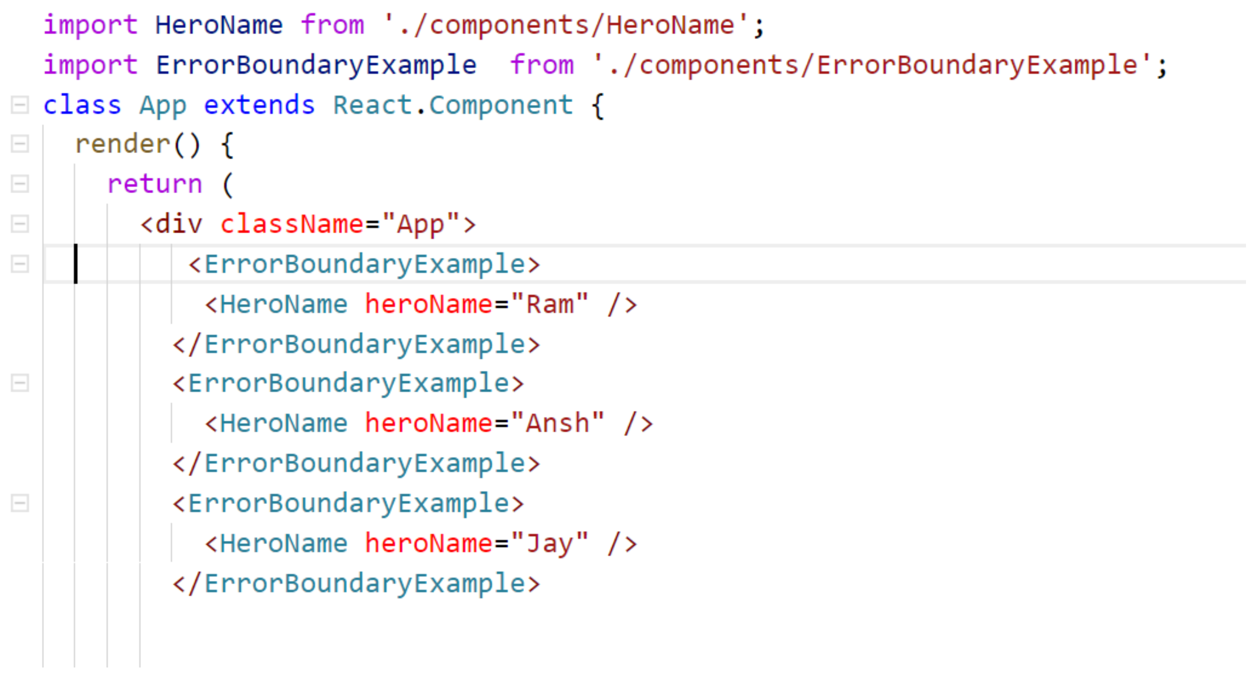
A class component becomes an error boundary if it defines either (or both) of the lifecycle methods static
getDerivedStateFromError() or componentDidCatch(error, info). Use static getDerivedStateFromError() to render a fallback UI after an error has been thrown. Use componentDidCatch() to log error information.

Example:
HeroName.js



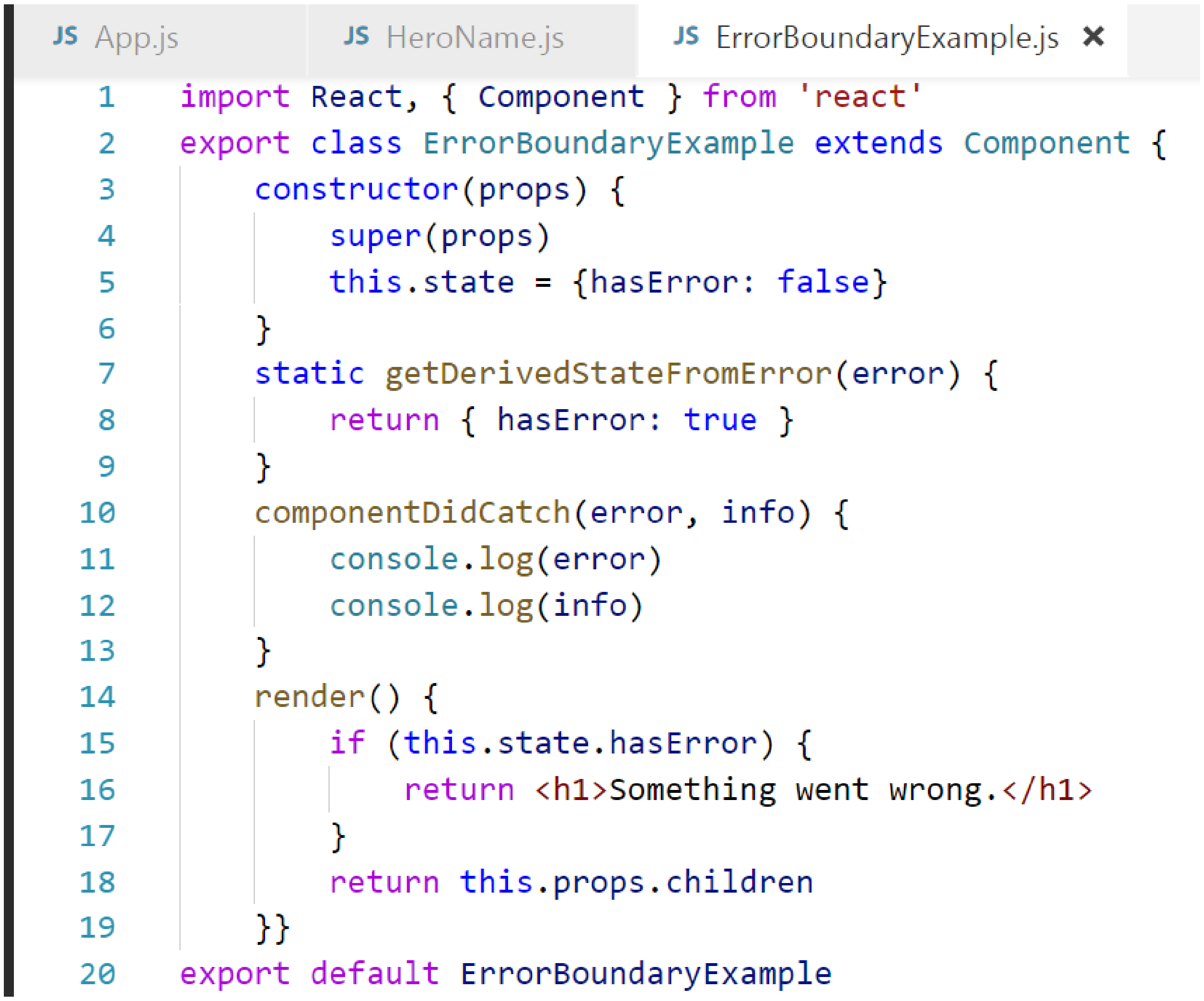
```
import React from 'react'

function HeroName({ heroName }) {
    if (heroName === 'Jay') {
        throw new Error(' Not a hero!')
    }
    return <h1>{heroName}</h1>
}

export default HeroName
```

App.js



```
import HeroName from './components/HeroName';
import ErrorBoundaryExample  from './components/ErrorBoundaryExample';
class App extends React.Component {
  render() {
    return (
      <div className="App">
         <ErrorBoundaryExample>
          <HeroName heroName="Ram" />
        </ErrorBoundaryExample>
        <ErrorBoundaryExample>
          <HeroName heroName="Ansh" />
        </ErrorBoundaryExample>
        <ErrorBoundaryExample>
          <HeroName heroName="Jay" />
        </ErrorBoundaryExample>
```

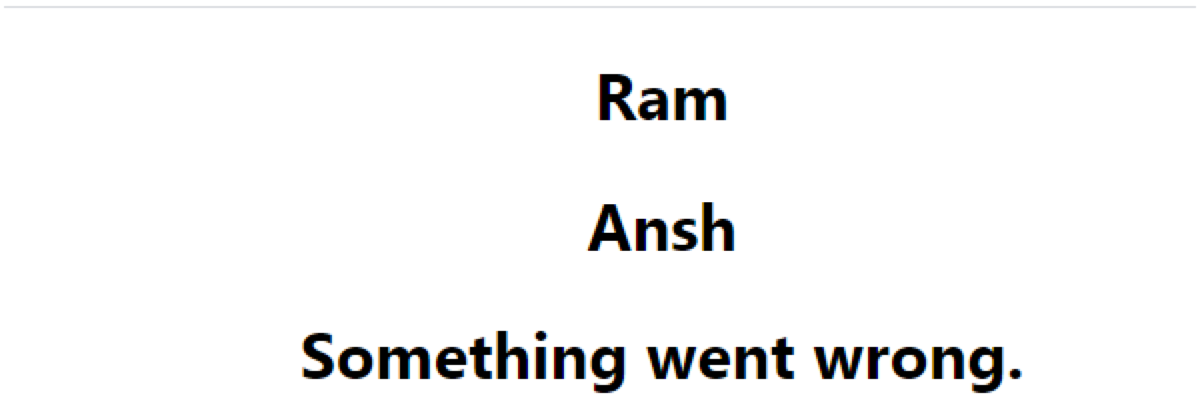
ErrorBoundaryExample.js



JS App.js JS HeroName.js JS ErrorBoundaryExample.js

```
import React, { Component } from 'react'
export class ErrorBoundaryExample extends Component {
    constructor(props) {
        super(props)
        this.state = {hasError: false}
    }
    static getDerivedStateFromError(error) {
        return { hasError: true }
    }
    componentDidCatch(error, info) {
        console.log(error)
        console.log(info)
    }
    render() {
        if (this.state.hasError) {
            return <h1>Something went wrong.</h1>
        }
        return this.props.children
    }}
export default ErrorBoundaryExample
```

Output:



```
Ram

Ansh

Something went wrong.
```

Higher order component:

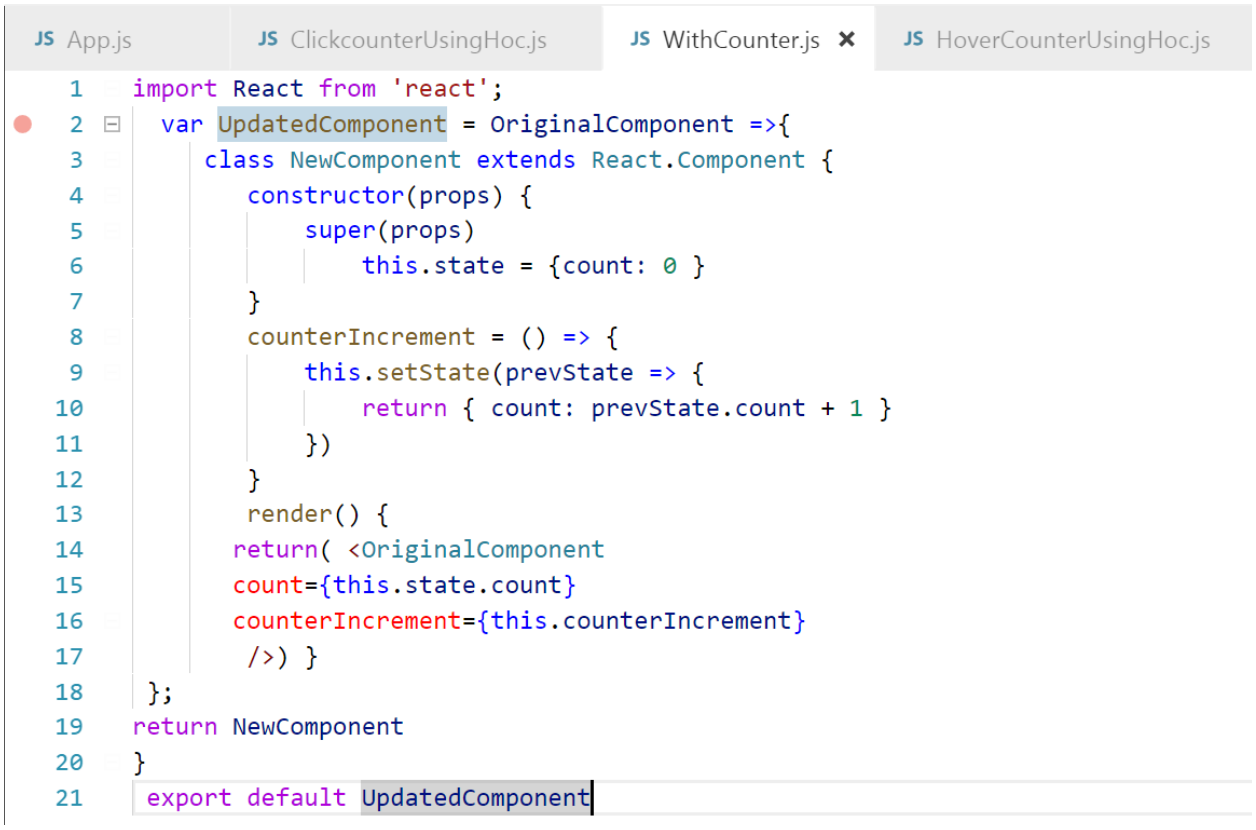
To share common functionality or logic between components without repeating code
A higher-order component is a function that takes a component and returns a new component.

When should you use HOC?

HOC is useful when you want to inject additional behaviours to the existing Component. You can use HOC to inject:

- React Lifecycle (eg. execute code in componentWillMount)
- State (eg. react-redux's connect)
- Component (Parent Component, Child Component, Sibling Component)
- Style

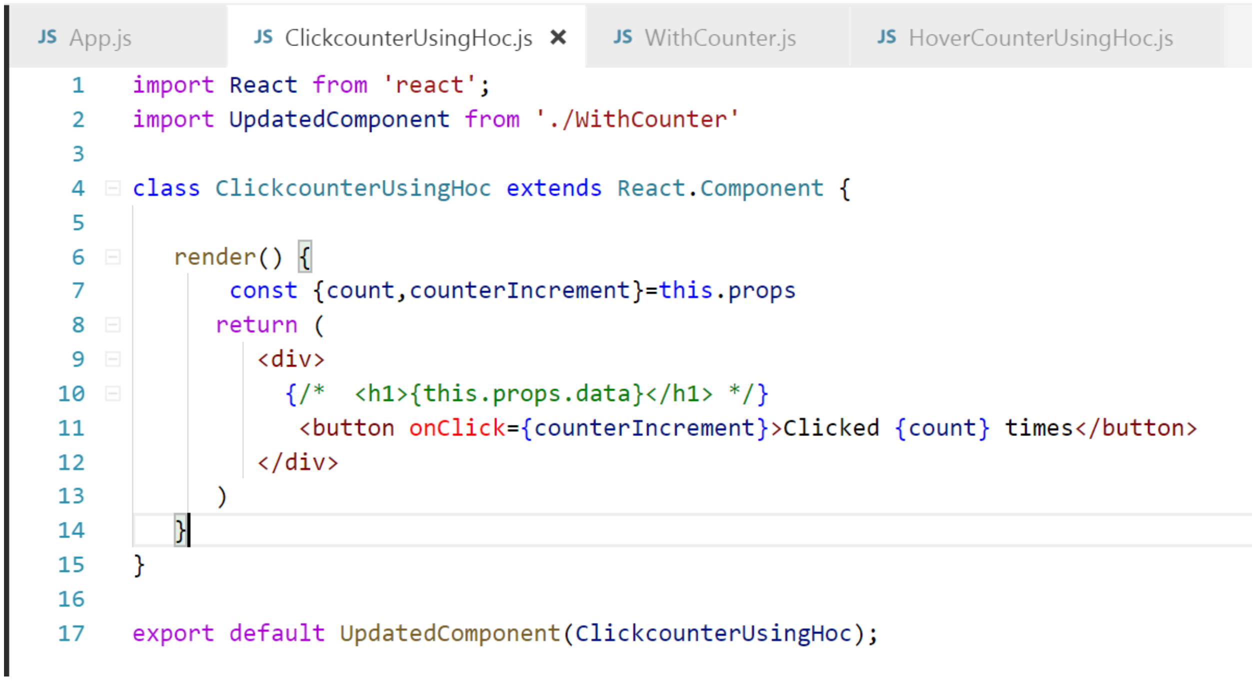
Example:
Withcounter.js



App.js | ClickcounterUsingHoc.js | WithCounter.js | HoverCounterUsingHoc.js

```
import React from 'react';
  var UpdatedComponent = OriginalComponent =>{
    class NewComponent extends React.Component {
       constructor(props) {
           super(props)
               this.state = {count: 0 }
       }
       counterIncrement = () => {
           this.setState(prevState => {
               return { count: prevState.count + 1 }
           })
       }
       render() {
      return( <OriginalComponent
      count={this.state.count}
      counterIncrement={this.counterIncrement}
       />) }
 };
return NewComponent
}
 export default UpdatedComponent
```

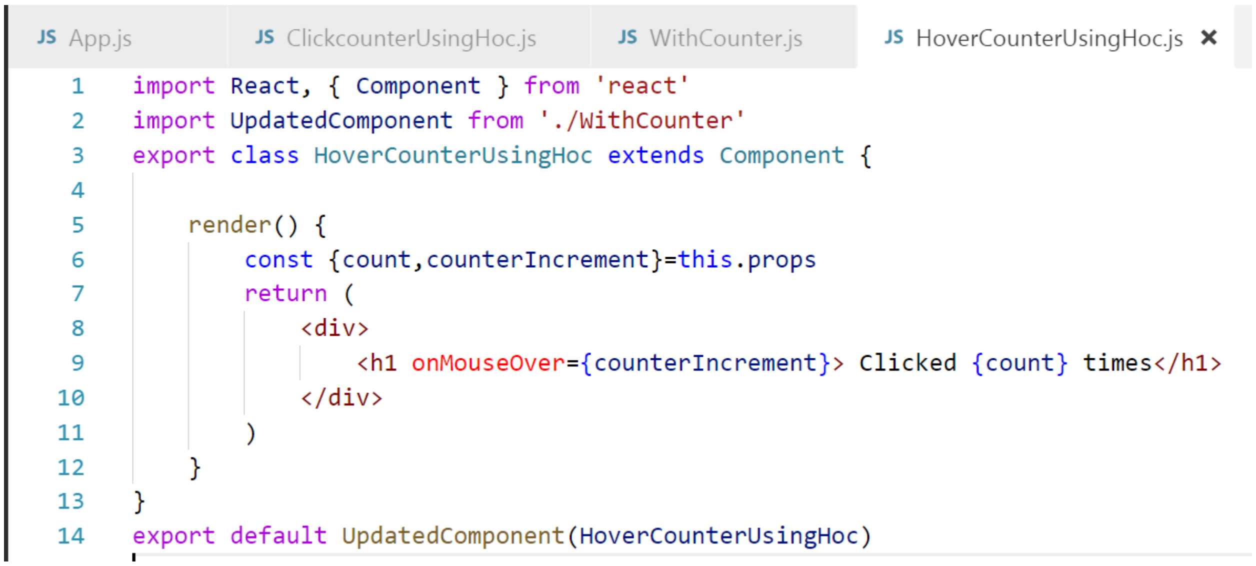
ClickcounterUsingHoc.js



App.js | ClickcounterUsingHoc.js | WithCounter.js | HoverCounterUsingHoc.js

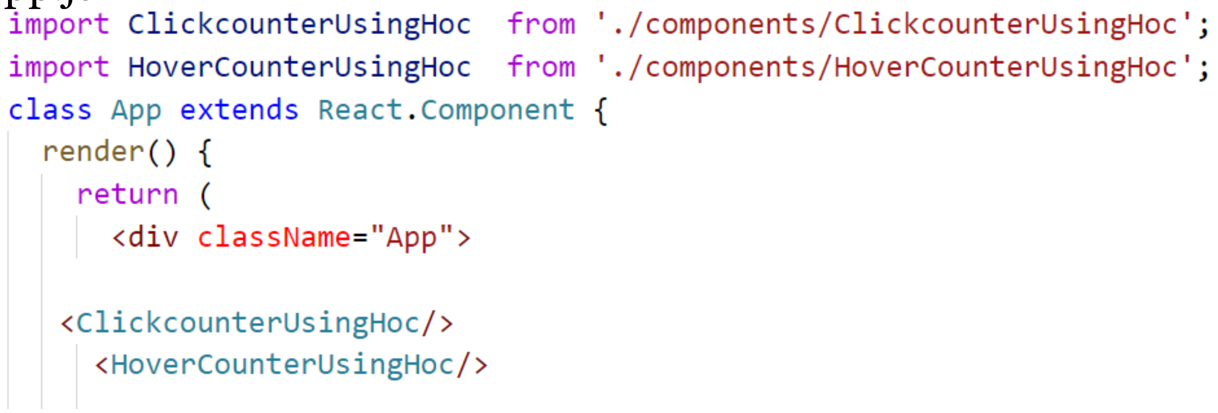
```
import React from 'react';
import UpdatedComponent from './WithCounter'

class ClickcounterUsingHoc extends React.Component {

   render() {
       const {count,counterIncrement}=this.props
      return (
         <div>
           {/*  <h1>{this.props.data}</h1> */}
            <button onClick={counterIncrement}>Clicked {count} times</button>
         </div>
      )
   }
}

export default UpdatedComponent(ClickcounterUsingHoc);
```

HoverCounterUsingHoc.js



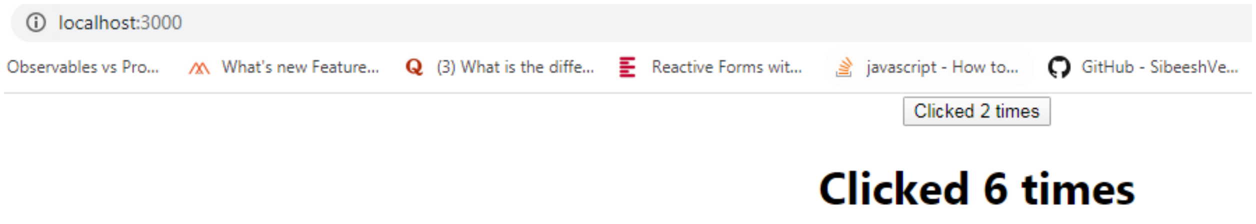
```
import React, { Component } from 'react'
import UpdatedComponent from './WithCounter'
export class HoverCounterUsingHoc extends Component {

    render() {
        const {count,counterIncrement}=this.props
        return (
            <div>
                <h1 onMouseOver={counterIncrement}> Clicked {count} times</h1>
            </div>
        )
    }
}
export default UpdatedComponent(HoverCounterUsingHoc)
```

App.js



```
import ClickcounterUsingHoc  from './components/ClickcounterUsingHoc';
import HoverCounterUsingHoc  from './components/HoverCounterUsingHoc';
class App extends React.Component {
  render() {
    return (
      <div className="App">

     <ClickcounterUsingHoc/>
       <HoverCounterUsingHoc/>
```

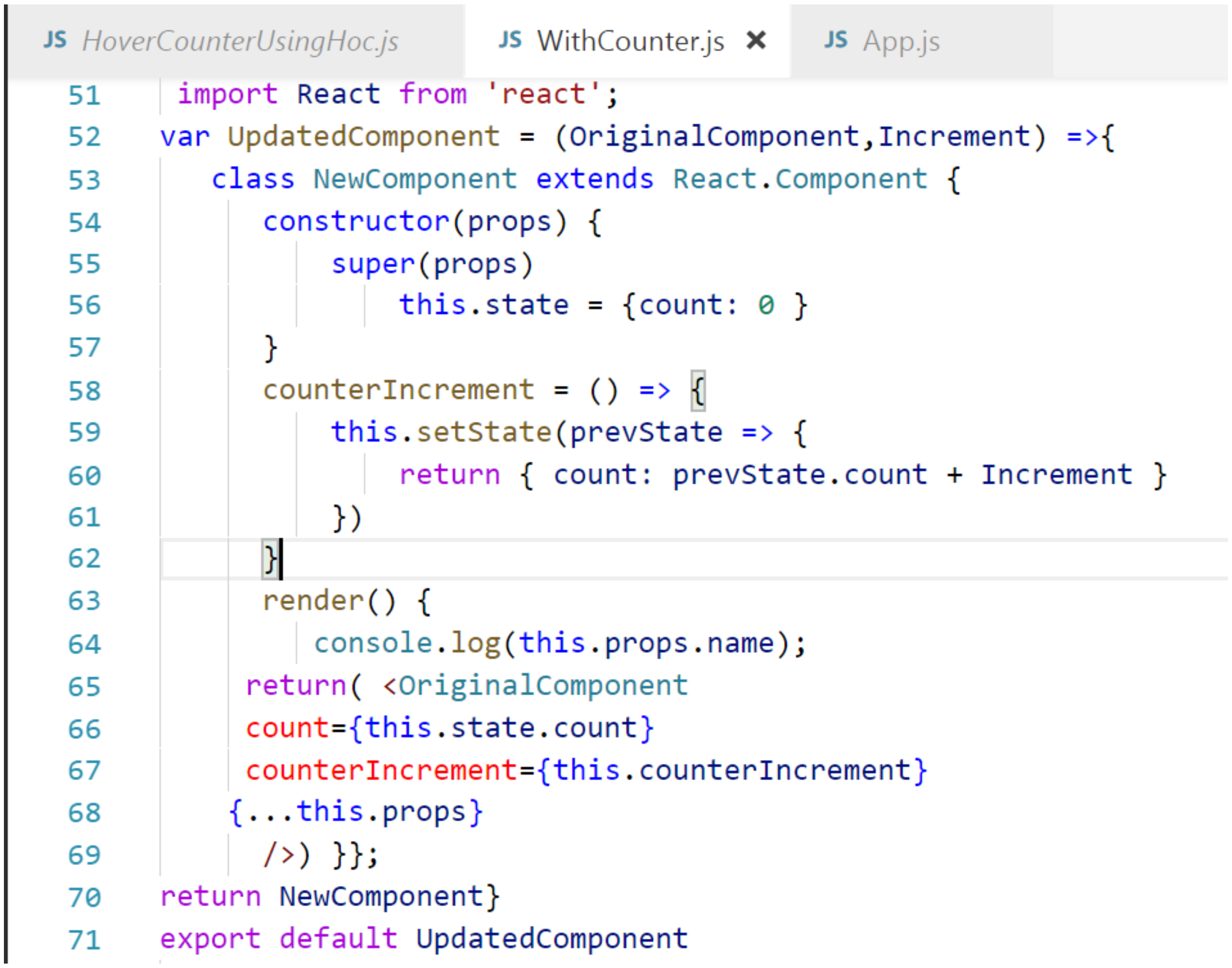
OutPut:



```
Clicked 6 times
```

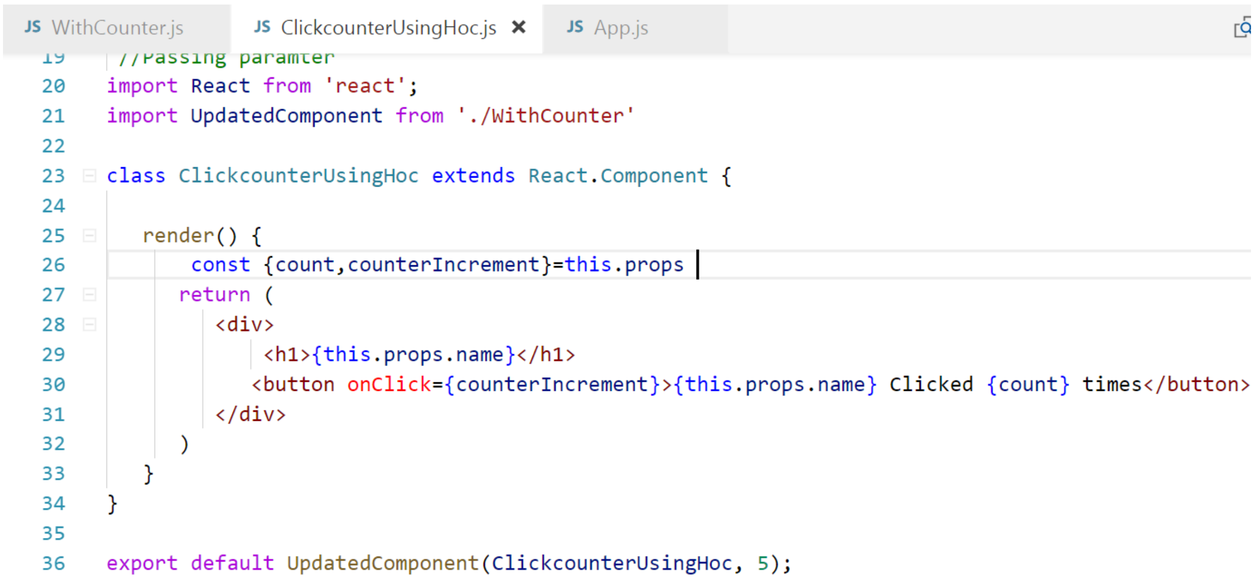
Passing parameter and default value in HOC

WithCounter.js



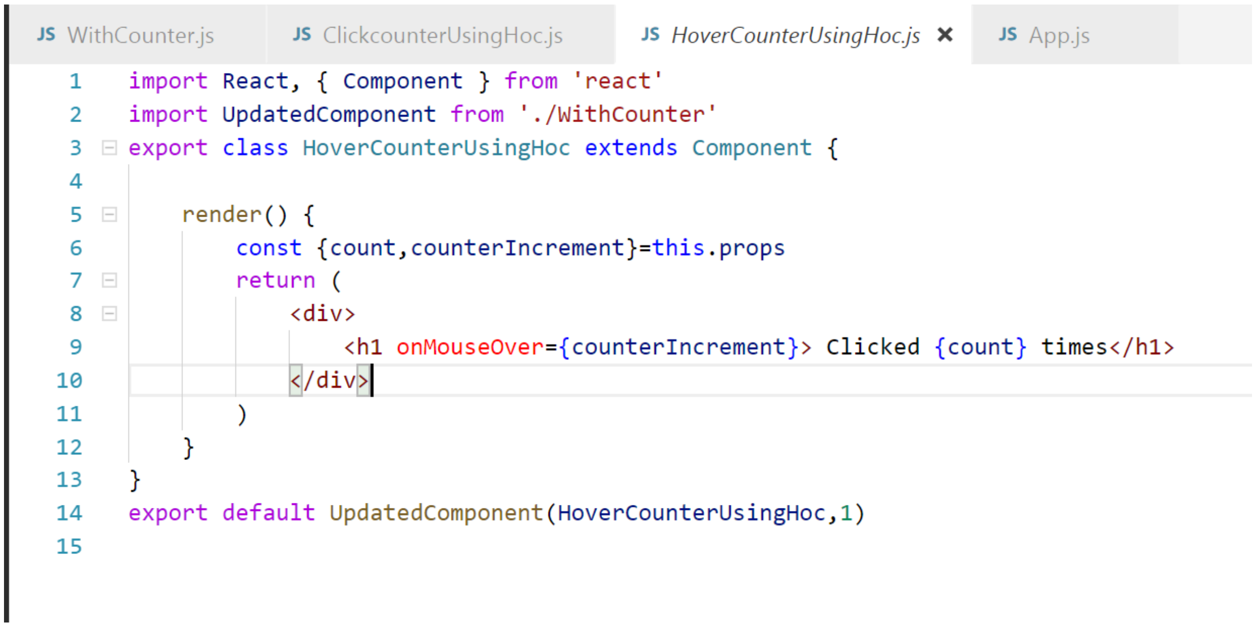
```
 import React from 'react';
var UpdatedComponent = (OriginalComponent,Increment) =>{
   class NewComponent extends React.Component {
      constructor(props) {
          super(props)
              this.state = {count: 0 }
      }
      counterIncrement = () => {
          this.setState(prevState => {
              return { count: prevState.count + Increment }
          })
      }
      render() {
         console.log(this.props.name);
     return( <OriginalComponent
     count={this.state.count}
     counterIncrement={this.counterIncrement}
    {...this.props}
      />) }};
return NewComponent}
export default UpdatedComponent
```

ClickcounterUsingHoc.js



```
 //Passing paramter
import React from 'react';
import UpdatedComponent from './WithCounter'

class ClickcounterUsingHoc extends React.Component {

   render() {
       const {count,counterIncrement}=this.props
      return (
         <div>
            <h1>{this.props.name}</h1>
           <button onClick={counterIncrement}>{this.props.name} Clicked {count} times</button>
         </div>
      )
   }
}

export default UpdatedComponent(ClickcounterUsingHoc, 5);
```

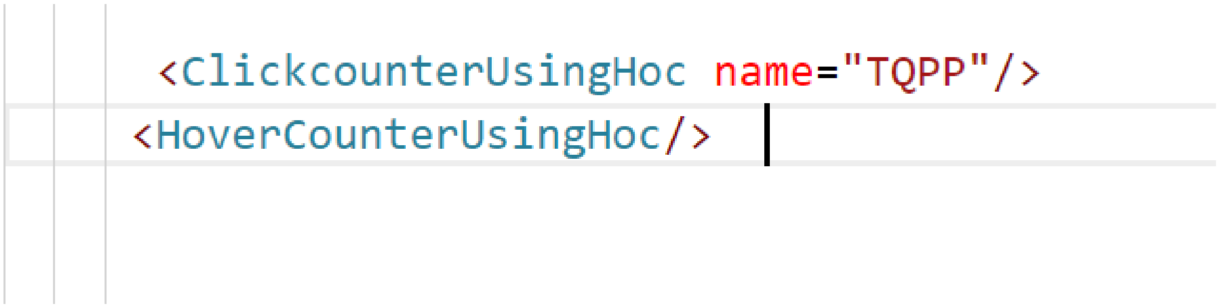
HovercounterUsingHoc.js



WithCounter.js | ClickcounterUsingHoc.js | HoverCounterUsingHoc.js | App.js

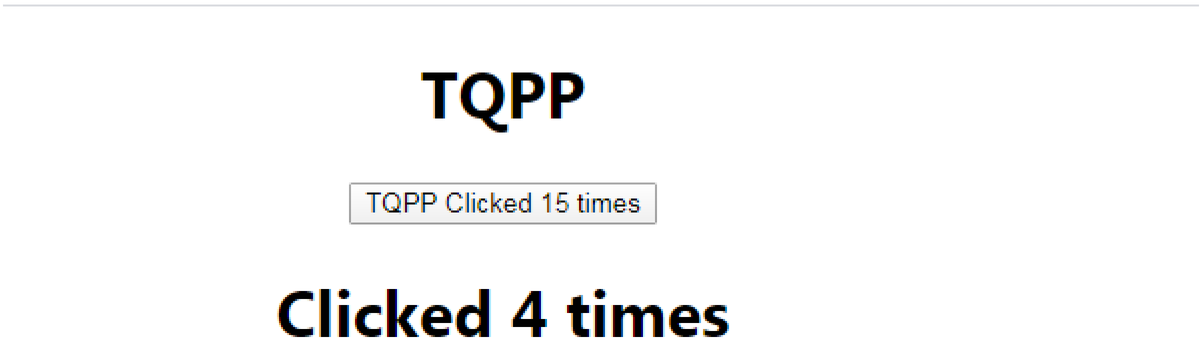
```
import React, { Component } from 'react'
import UpdatedComponent from './WithCounter'
export class HoverCounterUsingHoc extends Component {

    render() {
        const {count,counterIncrement}=this.props
        return (
            <div>
                <h1 onMouseOver={counterIncrement}> Clicked {count} times</h1>
            </div>
        )
    }
}
export default UpdatedComponent(HoverCounterUsingHoc,1)

```

App.js



```
      <ClickcounterUsingHoc name="TQPP"/>
     <HoverCounterUsingHoc/>
```

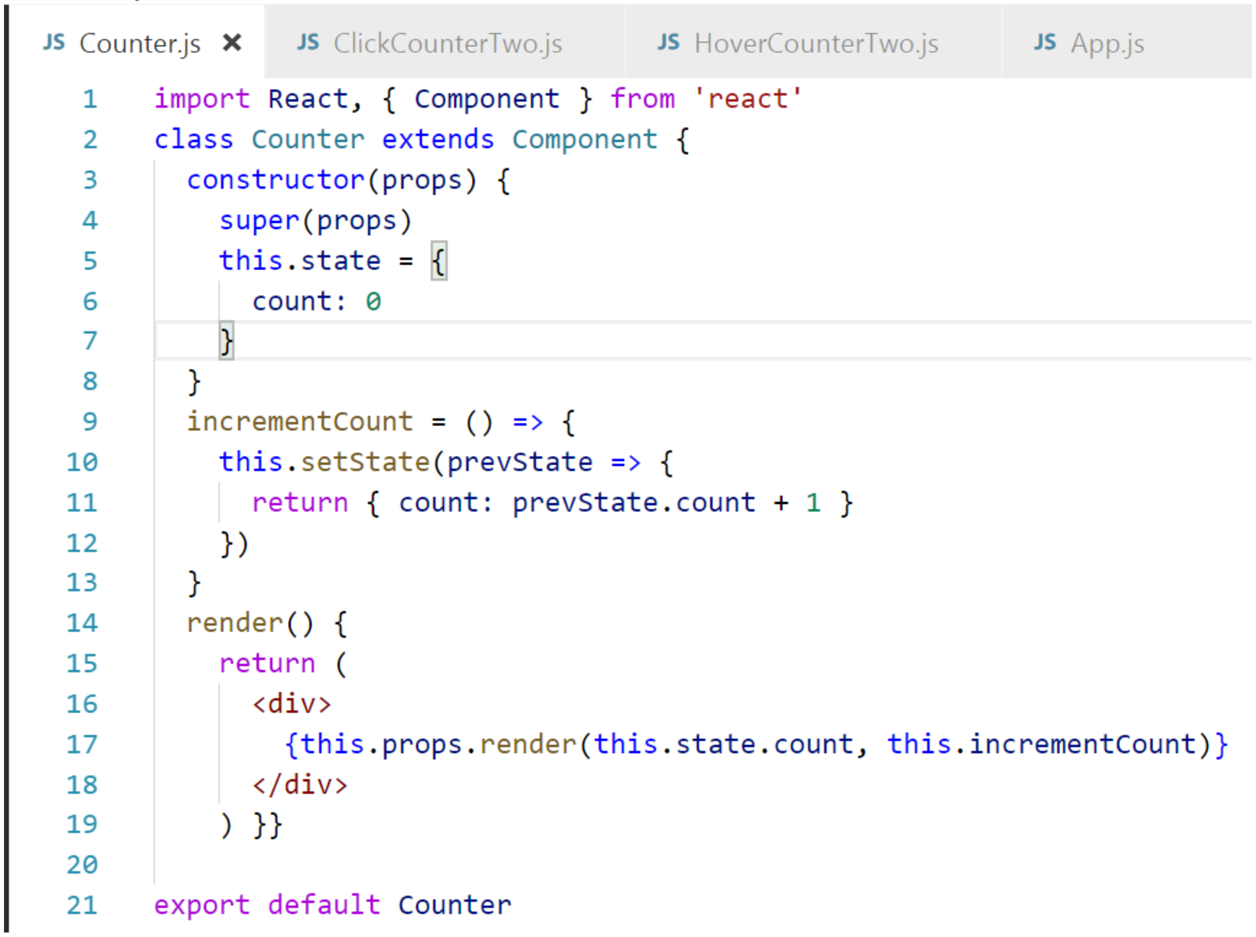
OutPut:



```
TQPP

TQPP Clicked 15 times

Clicked 4 times
```

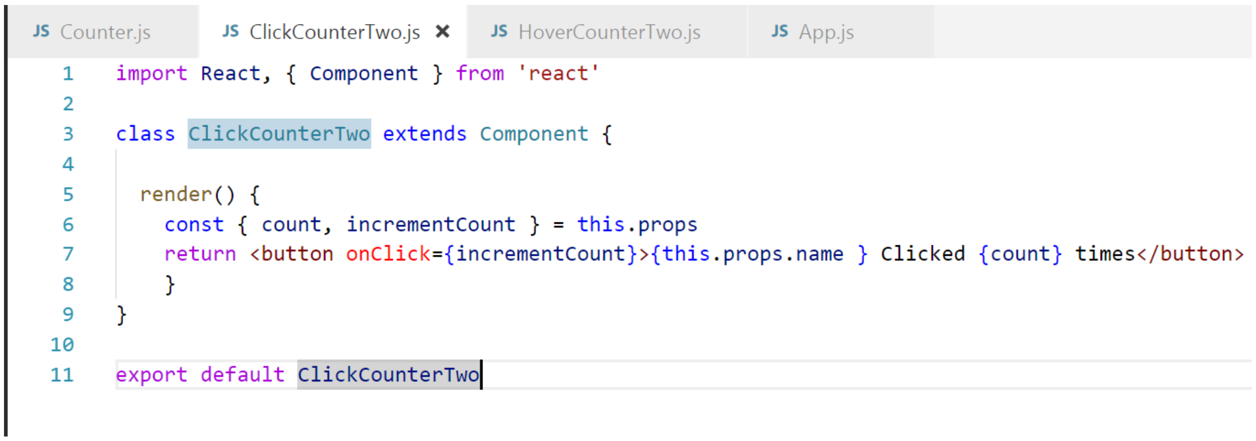
Render Props:

To Share code between react components which render props It is similar to HOC
The term “render prop” refers to a technique for **sharing code** between React components using a **prop whose value is a function**.
Example:
Counter.js



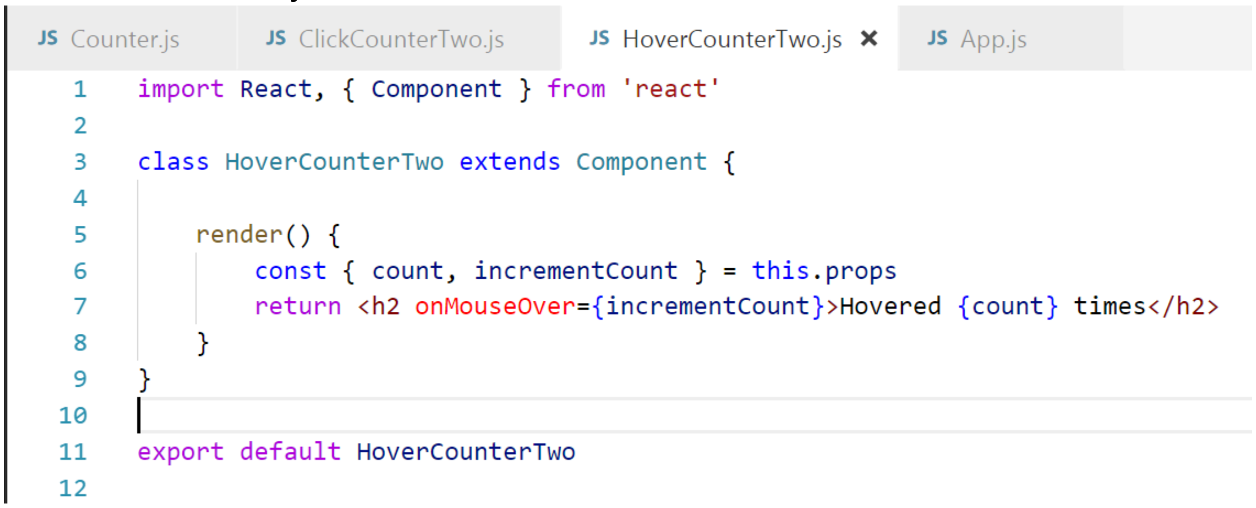
```
import React, { Component } from 'react'
class Counter extends Component {
  constructor(props) {
    super(props)
    this.state = {
      count: 0
    }
  }
  incrementCount = () => {
    this.setState(prevState => {
      return { count: prevState.count + 1 }
    })
  }
  render() {
    return (
      <div>
        {this.props.render(this.state.count, this.incrementCount)}
      </div>
    ) }}

export default Counter
```

ClickCounterTwo.js



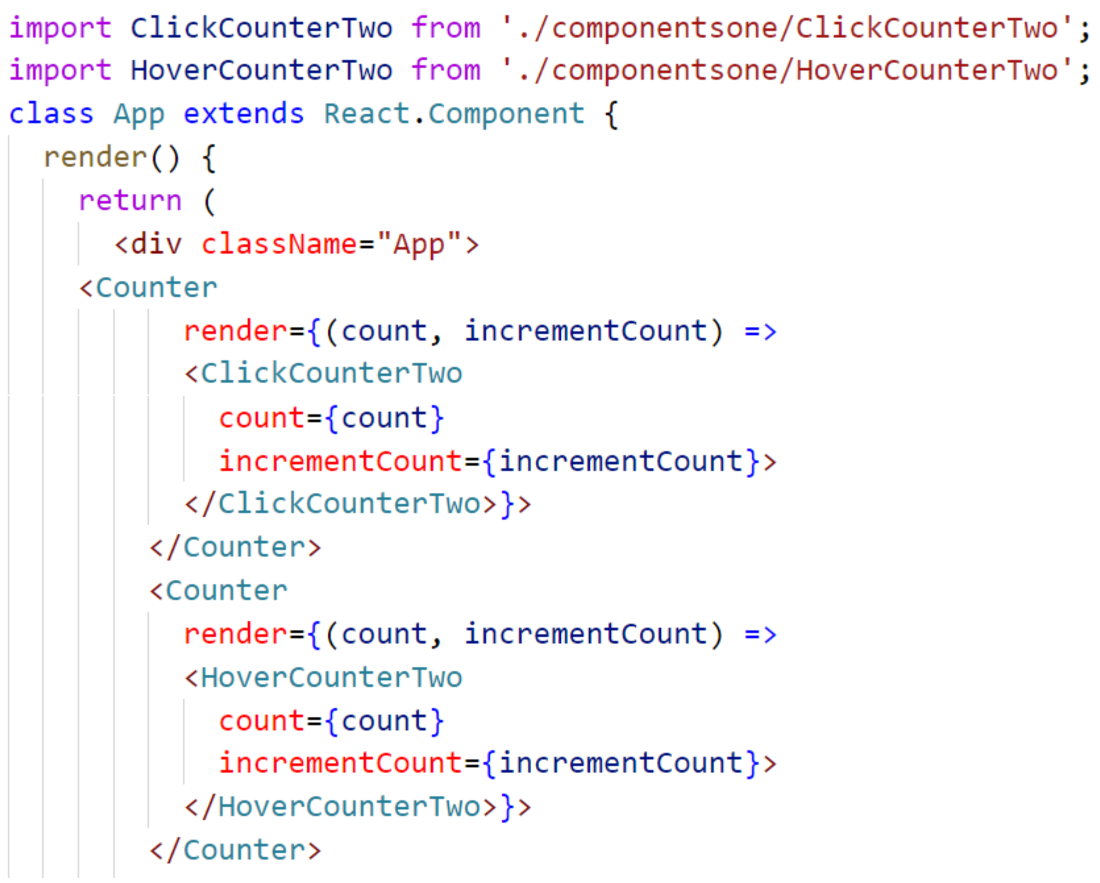
```
import React, { Component } from 'react'

class ClickCounterTwo extends Component {

  render() {
    const { count, incrementCount } = this.props
    return <button onClick={incrementCount}>{this.props.name } Clicked {count} times</button>
    }
}

export default ClickCounterTwo
```

HoverCounterTwo.js



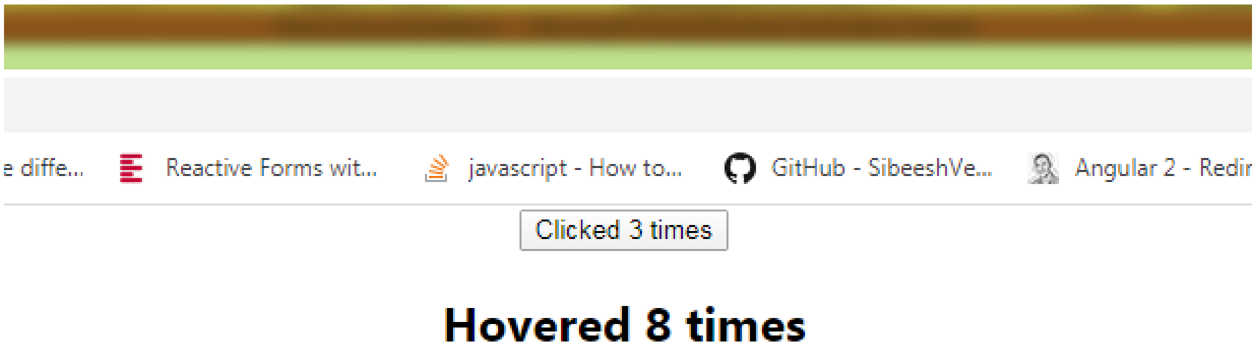
```
import React, { Component } from 'react'

class HoverCounterTwo extends Component {

    render() {
        const { count, incrementCount } = this.props
        return <h2 onMouseOver={incrementCount}>Hovered {count} times</h2>
    }
}

export default HoverCounterTwo

```

App.js



```
import ClickCounterTwo from './componentsone/ClickCounterTwo';
import HoverCounterTwo from './componentsone/HoverCounterTwo';
class App extends React.Component {
  render() {
    return (
      <div className="App">
    <Counter
          render={(count, incrementCount) =>
          <ClickCounterTwo
            count={count}
            incrementCount={incrementCount}>
          </ClickCounterTwo>}>
        </Counter>
        <Counter
          render={(count, incrementCount) =>
          <HoverCounterTwo
            count={count}
            incrementCount={incrementCount}>
          </HoverCounterTwo>}>
        </Counter>
```

OutPut:



e diffe... Reactive Forms wit... javascript - How to... GitHub - SibeeshVe... Angular 2 - Redir

```
Clicked 3 times

Hovered 8 times
```

React Context:

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

In a typical React application, data is passed top-down (parent to child) via props, but this can be cumbersome for certain types of props (e.g. locale preference, UI theme) that are required by many components within an application. Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.

When to Use Context

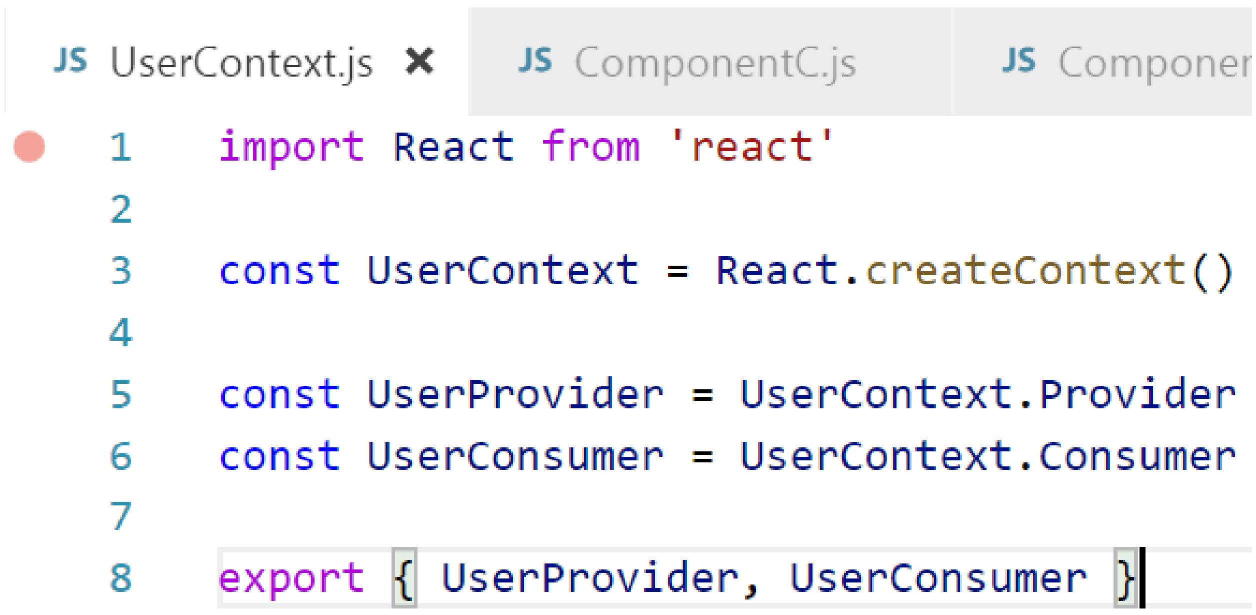
Context is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language.

Three steps:

1. Create the Context
2. Provide the context Value
3. Consume the context Value

Example:

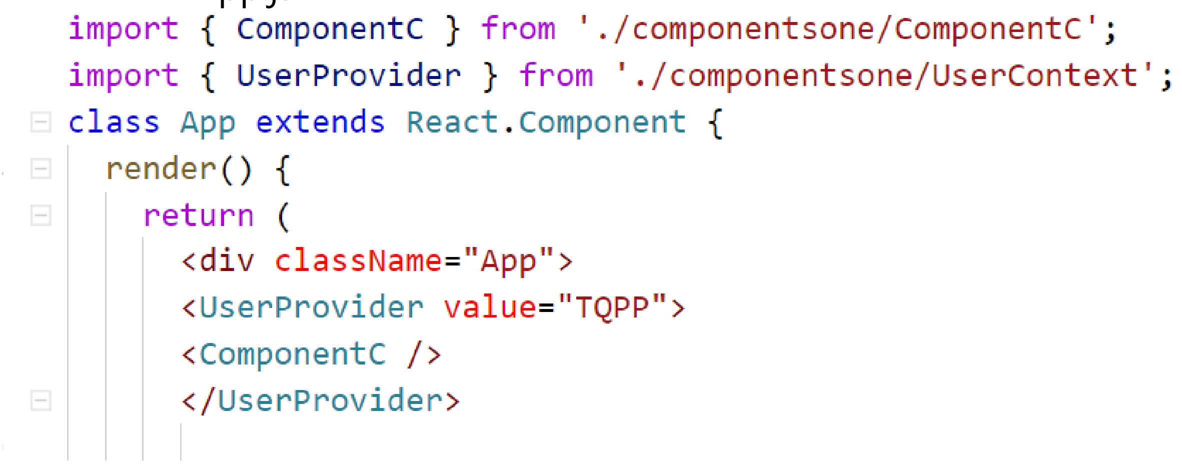
1. **Create the Context**



```
import React from 'react'

const UserContext = React.createContext()

const UserProvider = UserContext.Provider
const UserConsumer = UserContext.Consumer

export { UserProvider, UserConsumer }
```

2. **Provide the context Value**

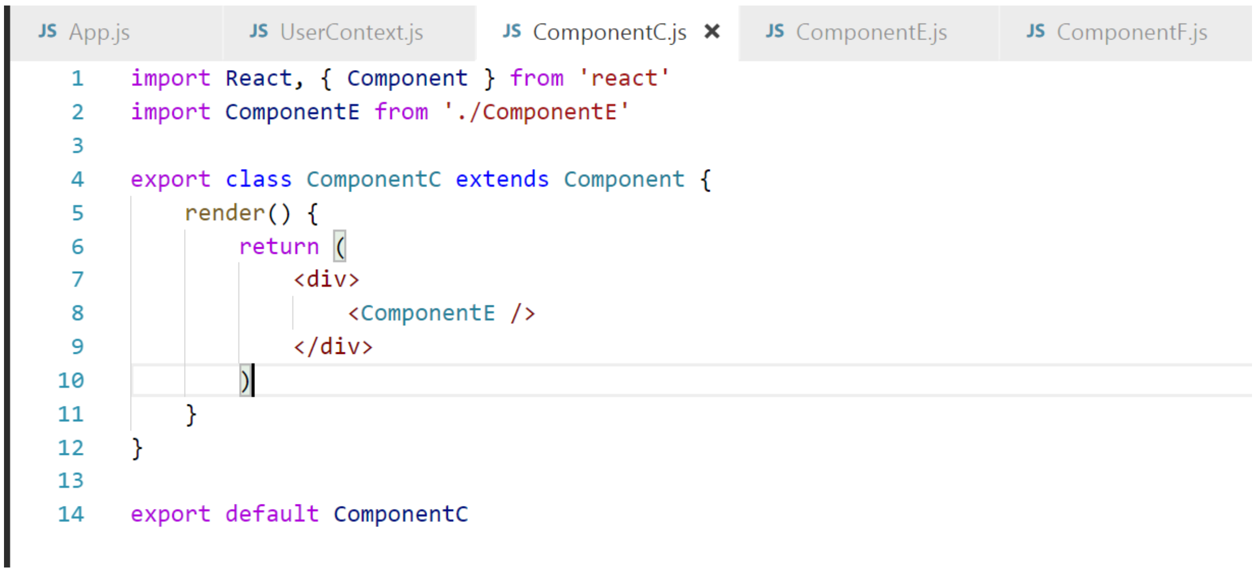
   App.js



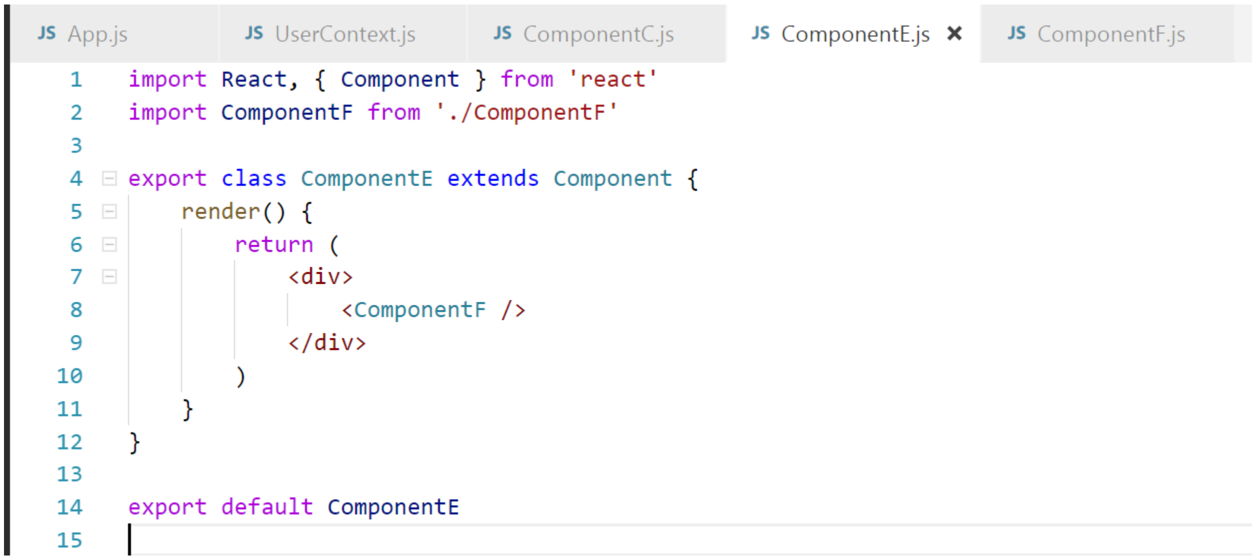
```
import { ComponentC } from './componentsone/ComponentC';
import { UserProvider } from './componentsone/UserContext';
class App extends React.Component {
  render() {
    return (
      <div className="App">
      <UserProvider value="TQPP">
      <ComponentC />
      </UserProvider>
```

3. **Consume the context Value**

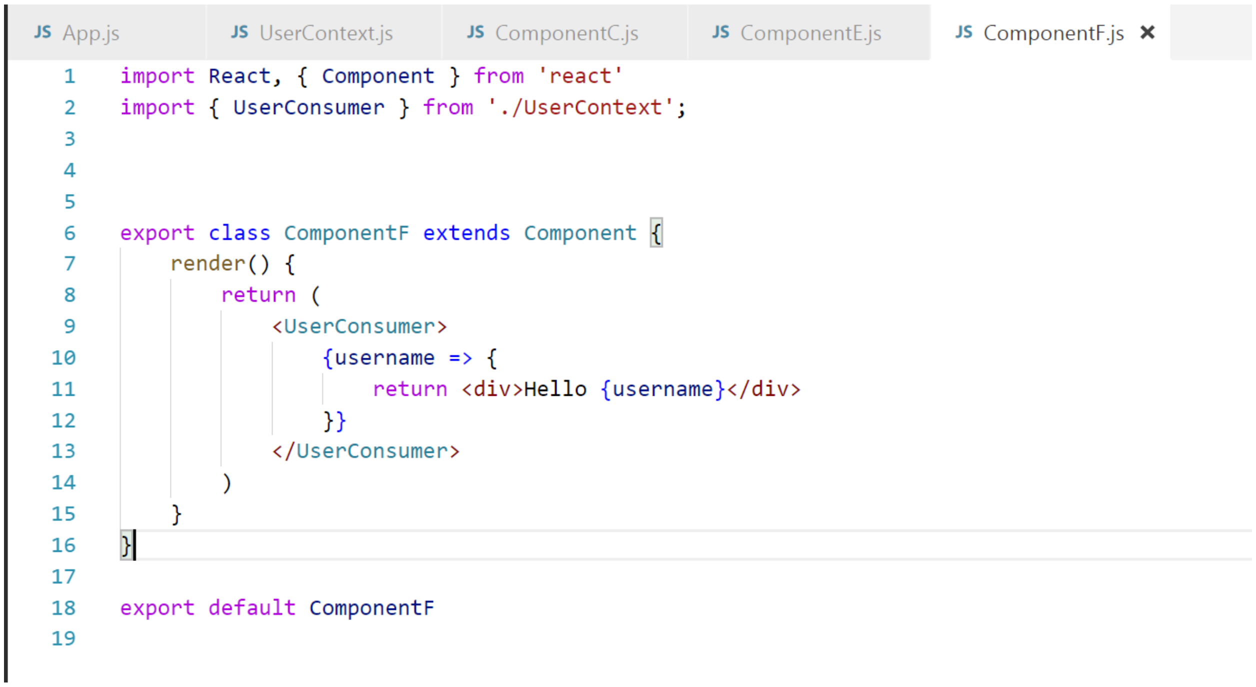
   In componentF



```
import React, { Component } from 'react'
import ComponentE from './ComponentE'

export class ComponentC extends Component {
    render() {
        return (
            <div>
                <ComponentE />
            </div>
        )
    }
}

export default ComponentC
```

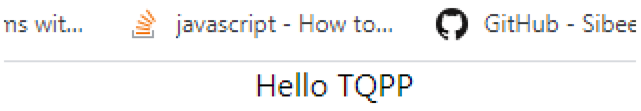


```
import React, { Component } from 'react'
import ComponentF from './ComponentF'

export class ComponentE extends Component {
    render() {
        return (
            <div>
                <ComponentF />
            </div>
        )
    }
}

export default ComponentE

```



```
import React, { Component } from 'react'
import { UserConsumer } from './UserContext';



export class ComponentF extends Component {
    render() {
        return (
            <UserConsumer>
                {username => {
                    return <div>Hello {username}</div>
                }}
            </UserConsumer>
        )
    }
}

export default ComponentF

```

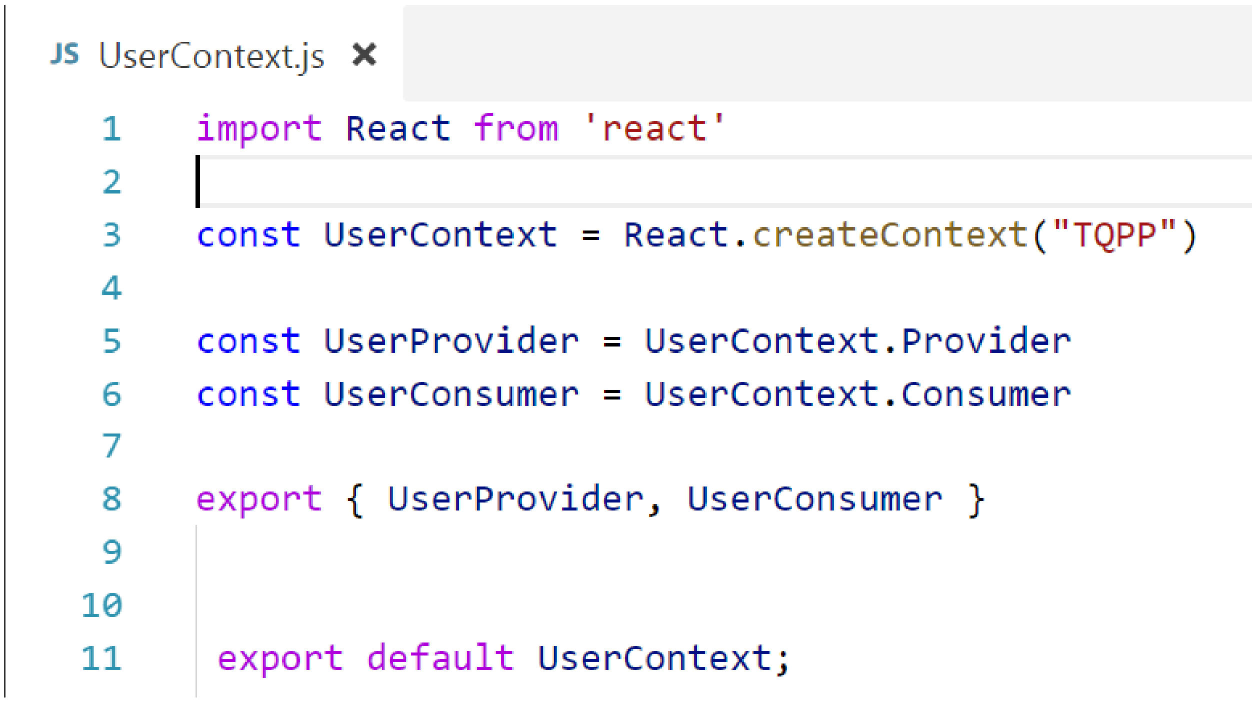
Output:



```
Hello TQPP
```

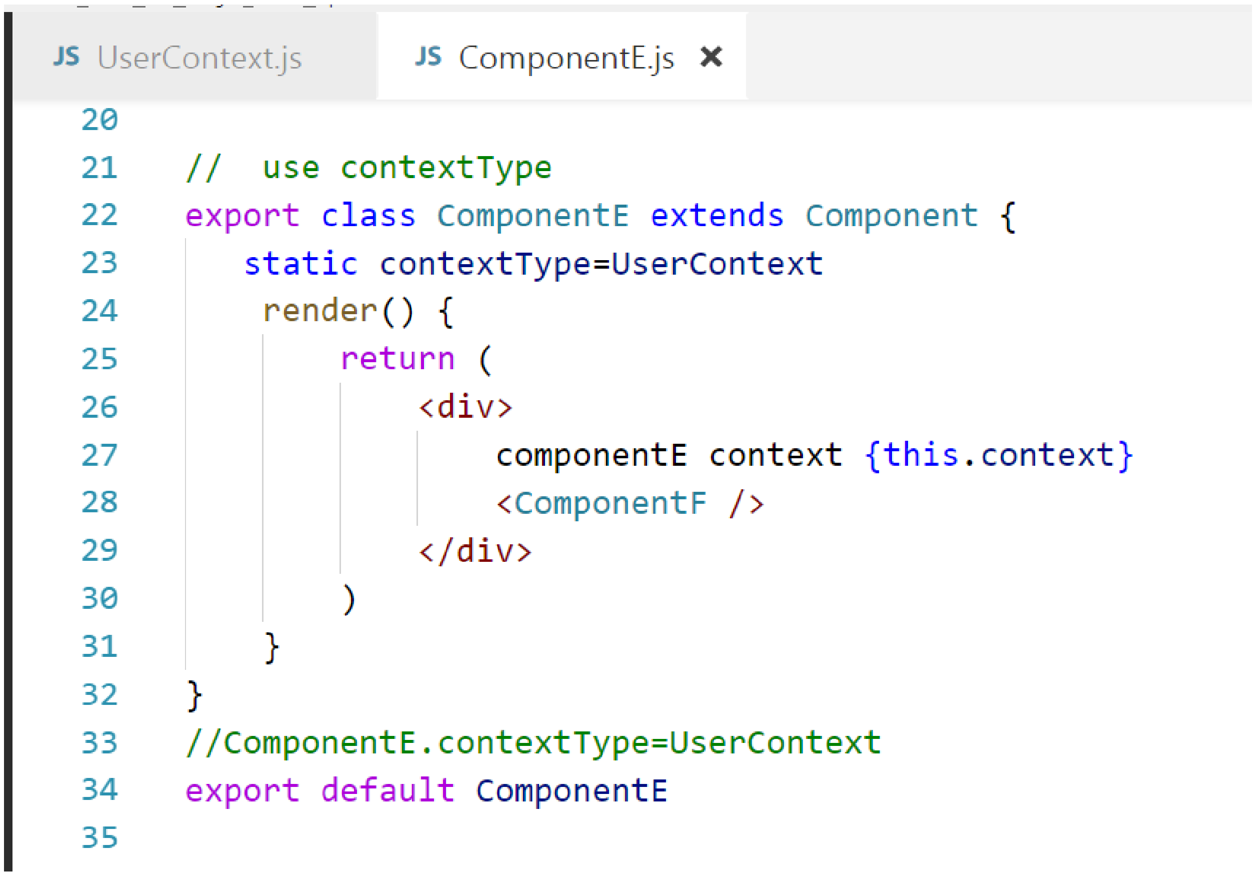
Set default value to context
const UserContext = React.createContext("TQ")

context type properties


UserContext.js

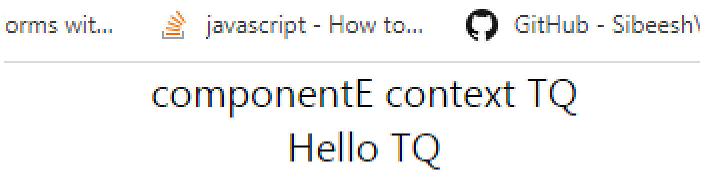
```
import React from 'react'

const UserContext = React.createContext("TQPP")

const UserProvider = UserContext.Provider
const UserConsumer = UserContext.Consumer

export { UserProvider, UserConsumer }


 export default UserContext;
```

ComponentE.js


UserContext.js ComponentE.js

```

//  use contextType
export class ComponentE extends Component {
   static contextType=UserContext
    render() {
        return (
            <div>
                componentE context {this.context}
                <ComponentF />
            </div>
        )
    }
}
//ComponentE.contextType=UserContext
export default ComponentE

```

OutPut:


orms wit... javascript - How to... GitHub - Sibeesh\

componentE context TQ

Hello TQ

HTTP in React

How can I make an AJAX call?

You can use any AJAX library you like with React. Some popular ones are Axios, jQuery AJAX, and the browser built-in window.fetch.

Where in the component lifecycle should I make an AJAX call?

You should populate data with AJAX calls in the componentDidMount lifecycle method. This is so you can use setState to update your component when the data is retrieved.

There are popular library to handle request we can use **axios and** also use **Fetch API** is also good to fetch data but we use axios

There are many HTTP libraries we can use to fetch data from a endpoint:

fetch, axios, superagent

Check in package.json file axios is install or not

First install axios API in your application

npm install axios

let us have to make get request using axios and render fetch data in react component

We can read fake data into our application from following url use

https://jsonplaceholder.typicode.com/

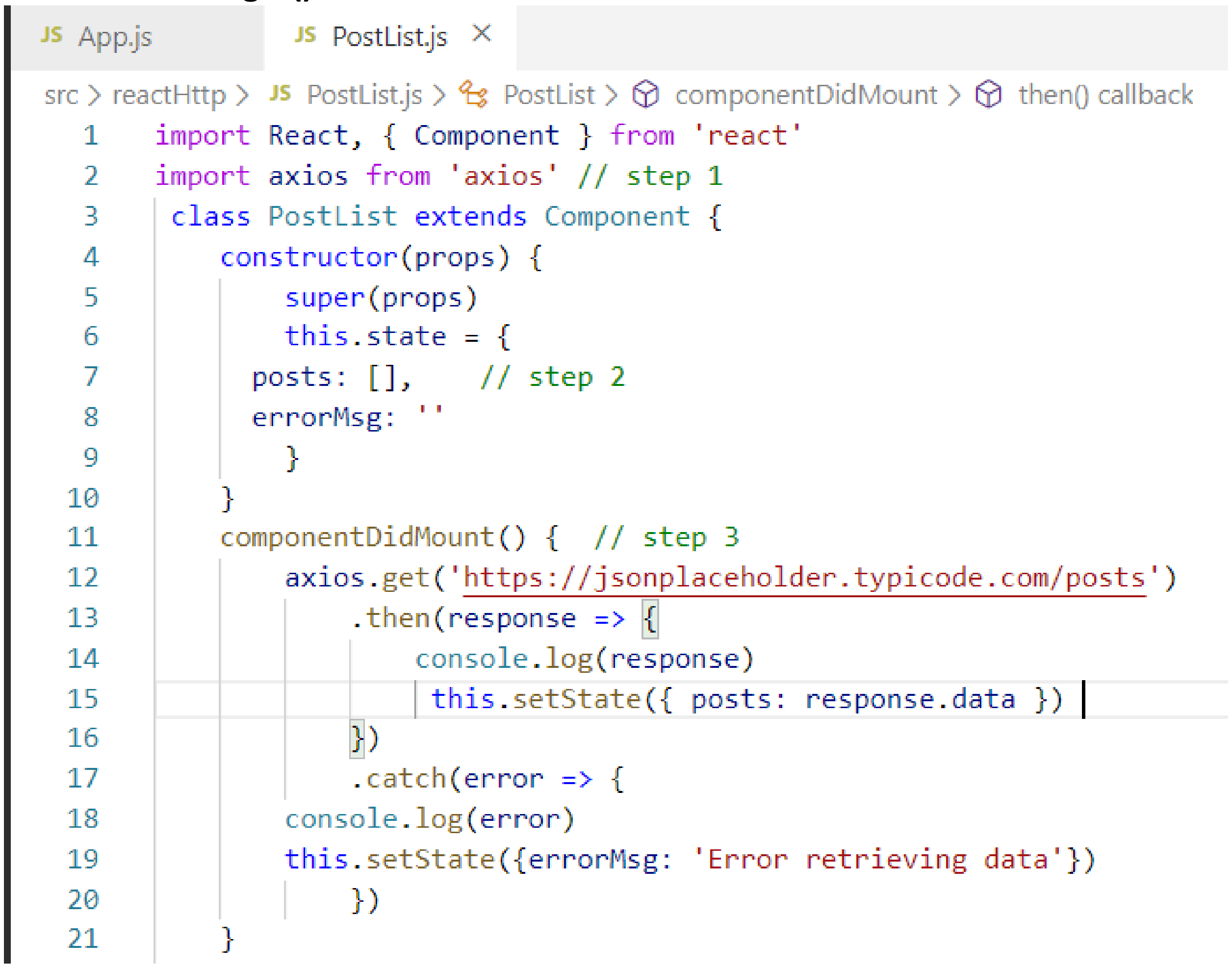
Axios is promise based libaray so we can add then() and catch() blocks

Example: Using AJAX results to set local state

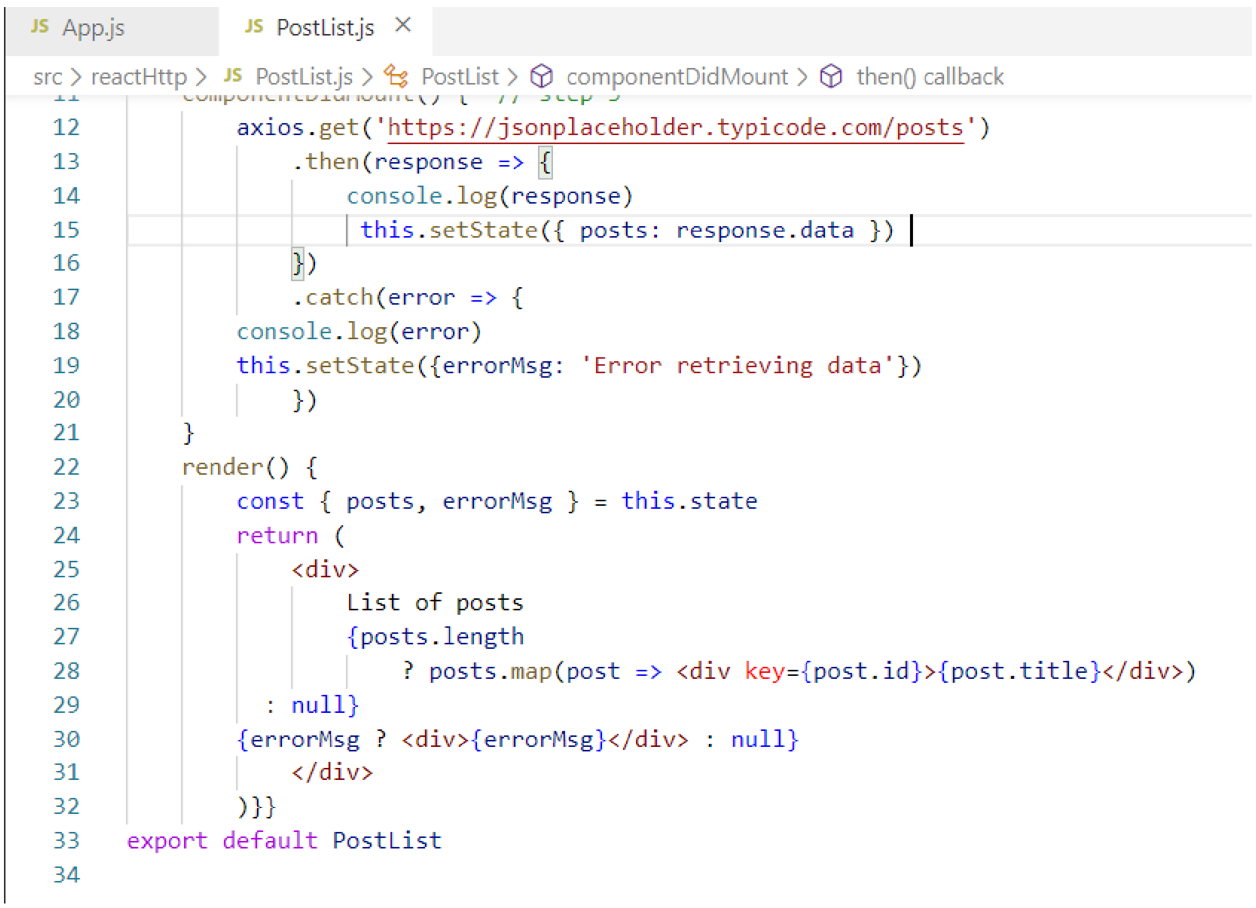
The component below demonstrates how to make an AJAX call in componentDidMount to populate local component state.

axios is allows you to send an asynchronous request to REST endpoints.

use Axios API - get()

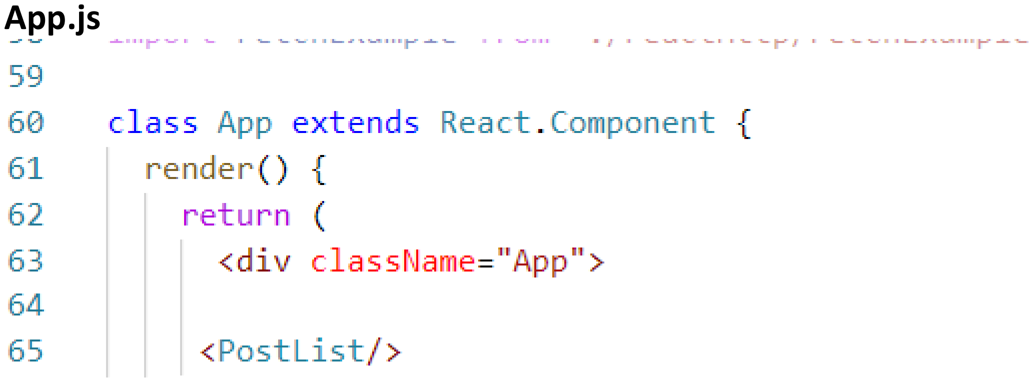

App.js
PostList.js
src > reactHttp > PostList.js > PostList > componentDidMount > then() callback

```
import React, { Component } from 'react'
import axios from 'axios' // step 1
 class PostList extends Component {
    constructor(props) {
        super(props)
        this.state = {
      posts: [],     // step 2
      errorMsg: ''
        }
    }
    componentDidMount() {  // step 3
        axios.get('https://jsonplaceholder.typicode.com/posts')
            .then(response => {
                console.log(response)
                 this.setState({ posts: response.data })
            })
            .catch(error => {
        console.log(error)
        this.setState({errorMsg: 'Error retrieving data'})
            })
    }
```


App.js
PostList.js
src > reactHttp > PostList.js > PostList > componentDidMount > then() callback

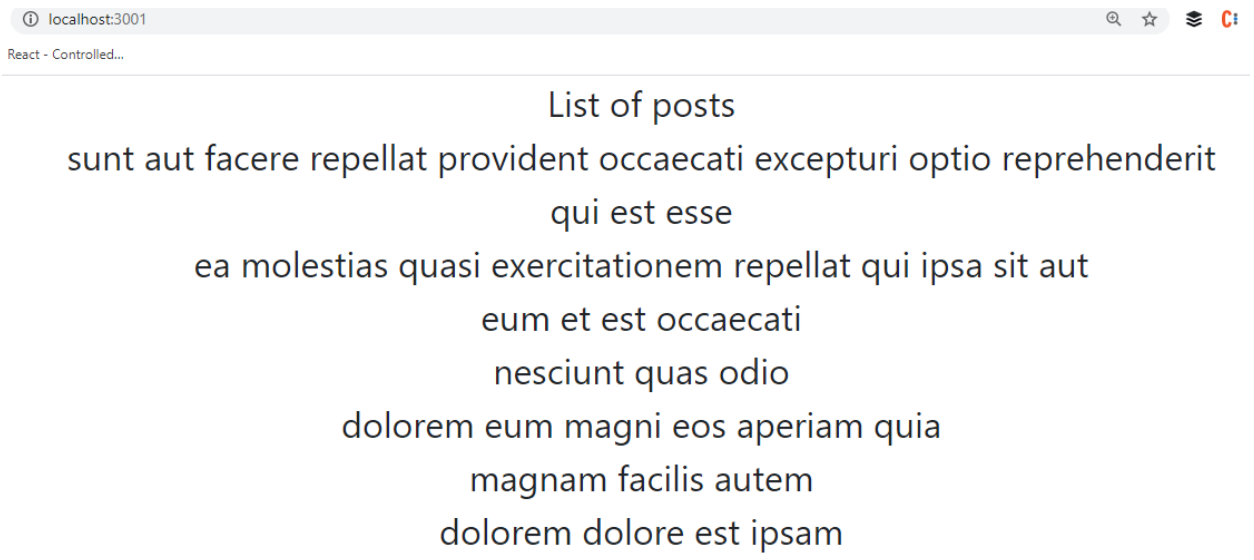
```
    componentDidMount() { // step 3
        axios.get('https://jsonplaceholder.typicode.com/posts')
            .then(response => {
                console.log(response)
                this.setState({ posts: response.data })
            })
            .catch(error => {
        console.log(error)
        this.setState({errorMsg: 'Error retrieving data'})
            })
    }
    render() {
        const { posts, errorMsg } = this.state
        return (
            <div>
                List of posts
                {posts.length
                    ? posts.map(post => <div key={post.id}>{post.title}</div>)
              : null}
        {errorMsg ? <div>{errorMsg}</div> : null}
            </div>
        )}}
export default PostList
```

App.js



```
import FetchExample from './reactHttp/FetchExample'

class App extends React.Component {
  render() {
    return (
      <div className="App">

     <PostList/>
```

Output



List of posts

sunt aut facere repellat provident occaecati excepturi optio reprehenderit

qui est esse

ea molestias quasi exercitationem repellat qui ipsa sit aut

eum et est occaecati

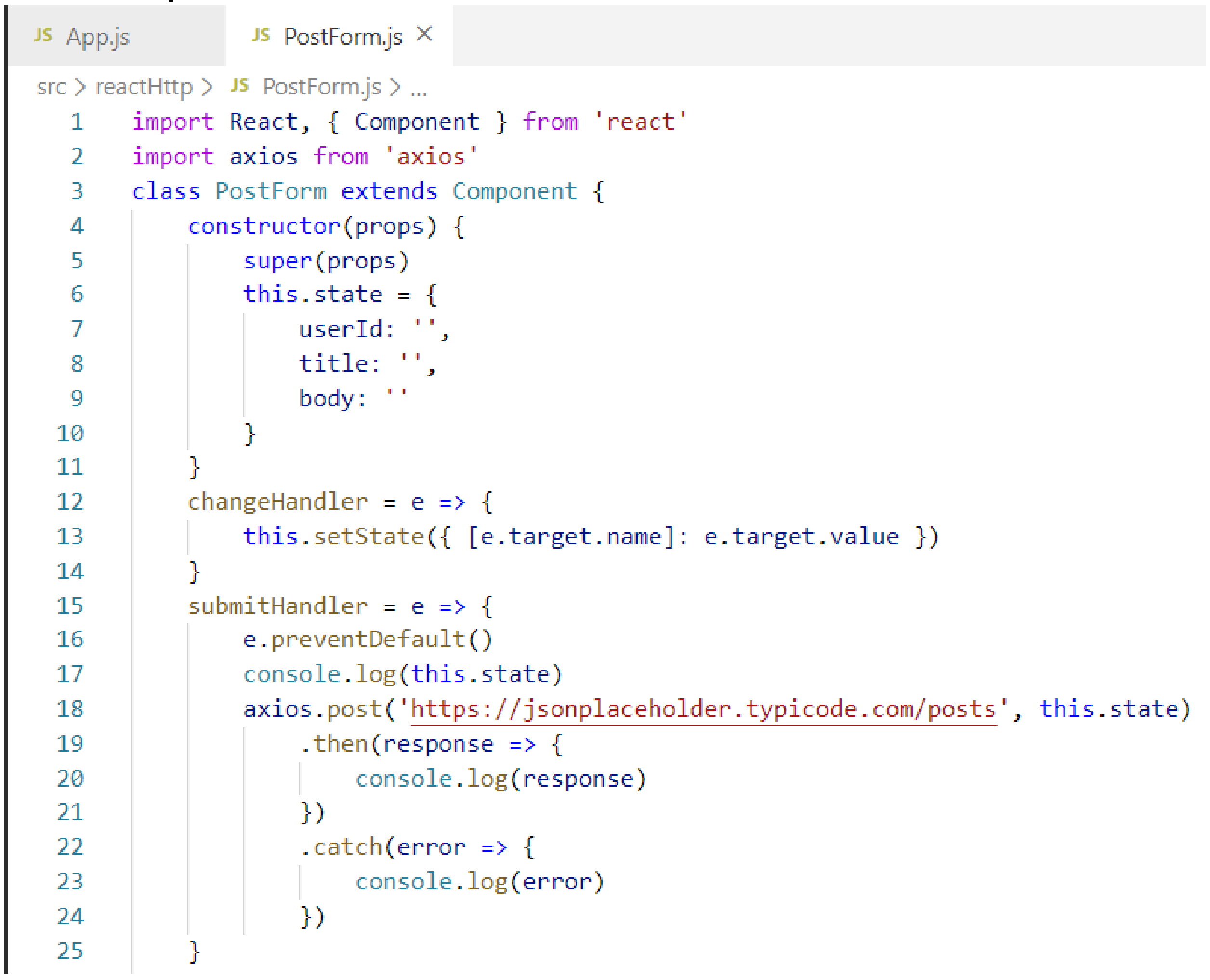
nesciunt quas odio

dolorem eum magni eos aperiam quia

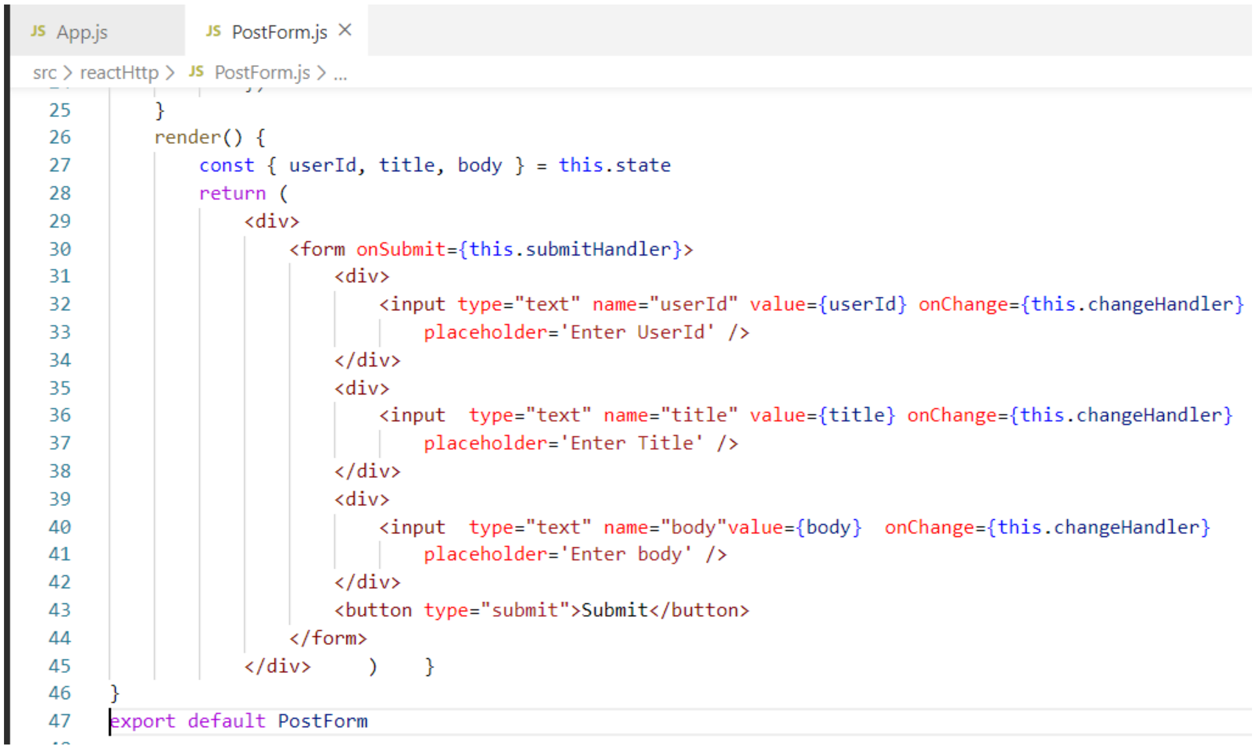
magnam facilis autem

dolorem dolore est ipsam

Post example



```
import React, { Component } from 'react'
import axios from 'axios'
class PostForm extends Component {
    constructor(props) {
        super(props)
        this.state = {
            userId: '',
            title: '',
            body: ''
        }
    }
    changeHandler = e => {
        this.setState({ [e.target.name]: e.target.value })
    }
    submitHandler = e => {
        e.preventDefault()
        console.log(this.state)
        axios.post('https://jsonplaceholder.typicode.com/posts', this.state)
            .then(response => {
                console.log(response)
            })
            .catch(error => {
                console.log(error)
            })
    }
```

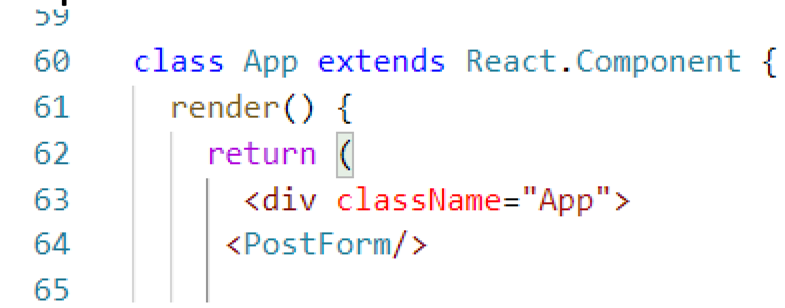


JS App.js   JS PostForm.js ×

src > reactHttp > JS PostForm.js > ...

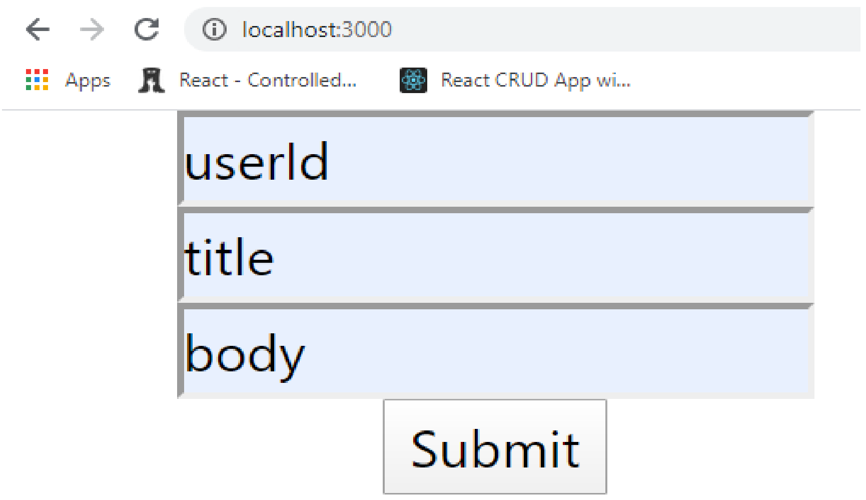
```
    }
    render() {
        const { userId, title, body } = this.state
        return (
            <div>
                <form onSubmit={this.submitHandler}>
                    <div>
                        <input type="text" name="userId" value={userId} onChange={this.changeHandler}
                            placeholder='Enter UserId' />
                    </div>
                    <div>
                        <input  type="text" name="title" value={title} onChange={this.changeHandler}
                            placeholder='Enter Title' />
                    </div>
                    <div>
                        <input  type="text" name="body"value={body}  onChange={this.changeHandler}
                            placeholder='Enter body' />
                    </div>
                    <button type="submit">Submit</button>
                </form>
            </div>     )     }
}
export default PostForm
```

Output:



```
class App extends React.Component {
  render() {
    return (
      <div className="App">
     <PostForm/>
```

Output:



localhost:3000

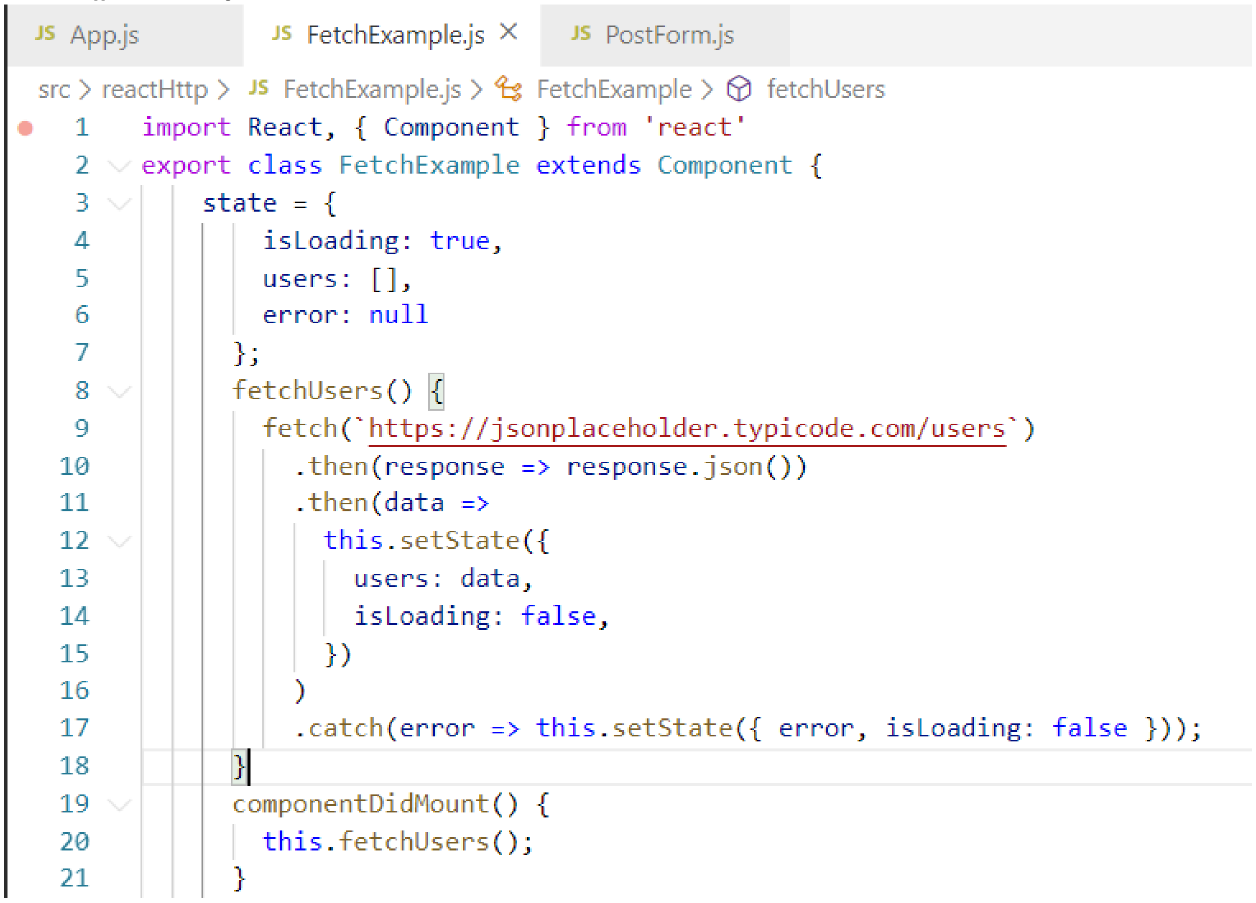
Apps   React - Controlled...   React CRUD App wi...

Check response in browser console

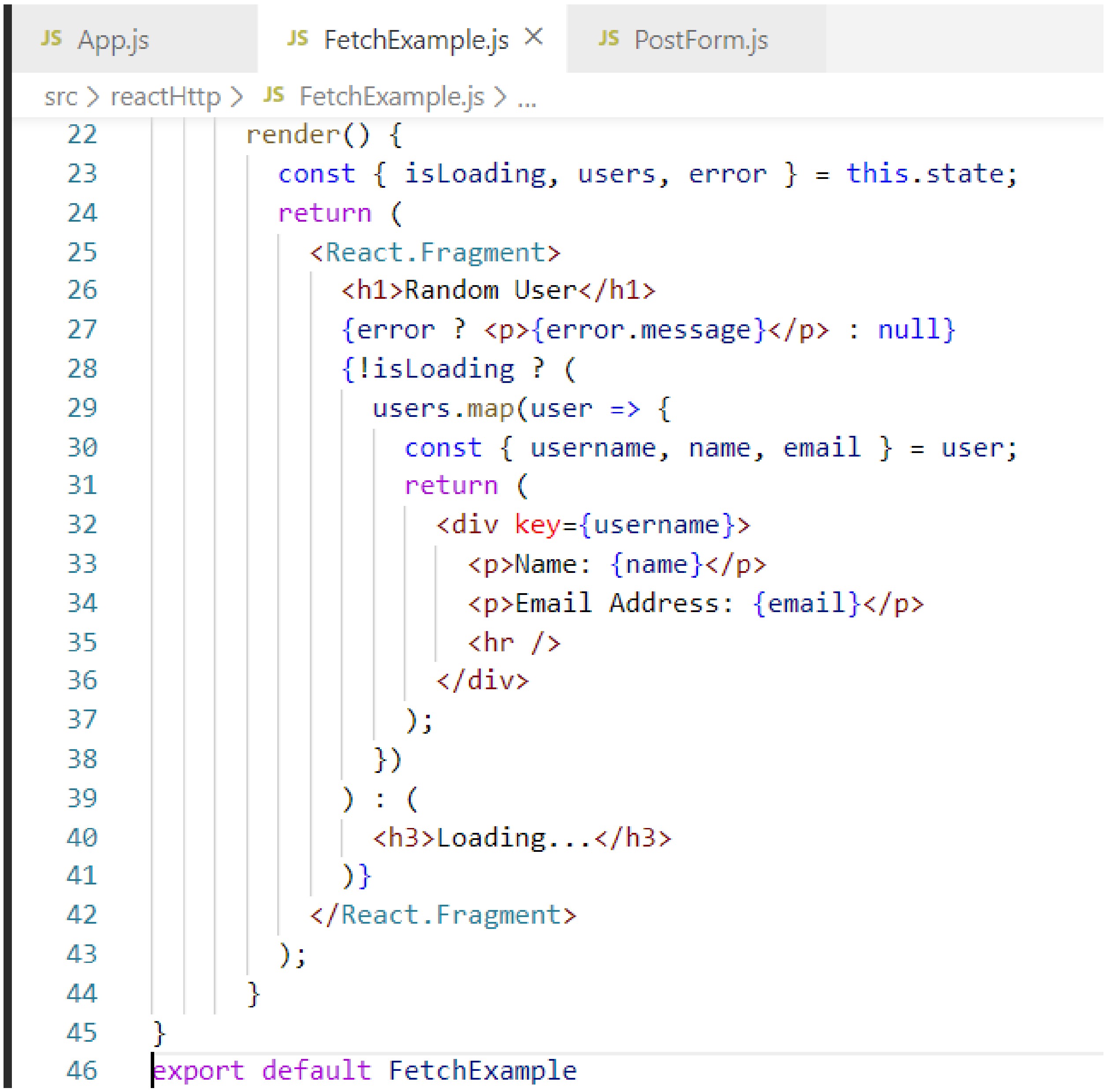
The Fetch API

The Fetch API provides an interface for fetching resources. We'll use it to fetch data from a third-party API and see how to use it when fetching data from an API built in-house.

Fetch() : example


App.js
FetchExample.js
PostForm.js
src > reactHttp > FetchExample.js > FetchExample > fetchUsers

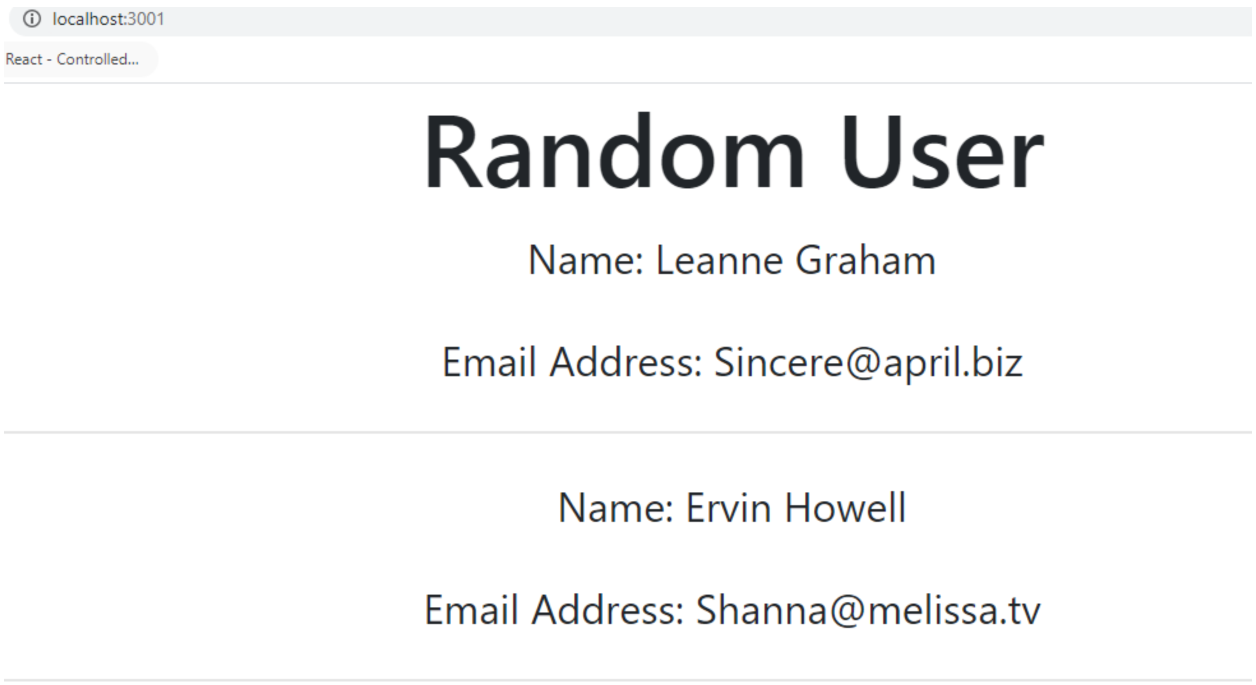
```
import React, { Component } from 'react'
export class FetchExample extends Component {
    state = {
        isLoading: true,
        users: [],
        error: null
    };
    fetchUsers() {
        fetch(`https://jsonplaceholder.typicode.com/users`)
          .then(response => response.json())
          .then(data =>
            this.setState({
              users: data,
              isLoading: false,
            })
          )
          .catch(error => this.setState({ error, isLoading: false }));
    }
    componentDidMount() {
        this.fetchUsers();
    }
```


JS App.js   JS FetchExample.js ×   JS PostForm.js

src > reactHttp > JS FetchExample.js > ...

```
  render() {
    const { isLoading, users, error } = this.state;
    return (
      <React.Fragment>
        <h1>Random User</h1>
        {error ? <p>{error.message}</p> : null}
        {!isLoading ? (
          users.map(user => {
            const { username, name, email } = user;
            return (
              <div key={username}>
                <p>Name: {name}</p>
                <p>Email Address: {email}</p>
                <hr />
              </div>
            );
          })
        ) : (
          <h3>Loading...</h3>
        )}
      </React.Fragment>
    );
  }
}
export default FetchExample
```

Output:



Random User

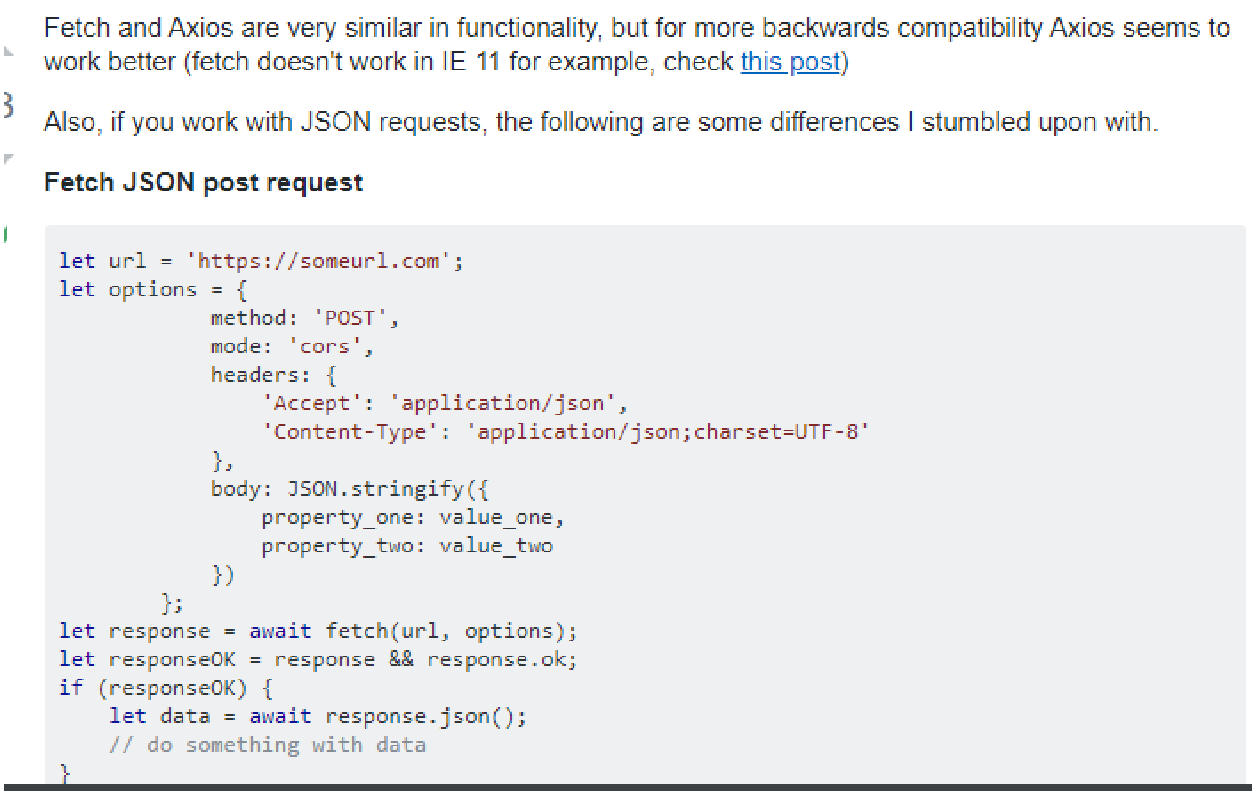
Name: Leanne Graham

Email Address: Sincere@april.biz

Name: Ervin Howell

Email Address: Shanna@melissa.tv

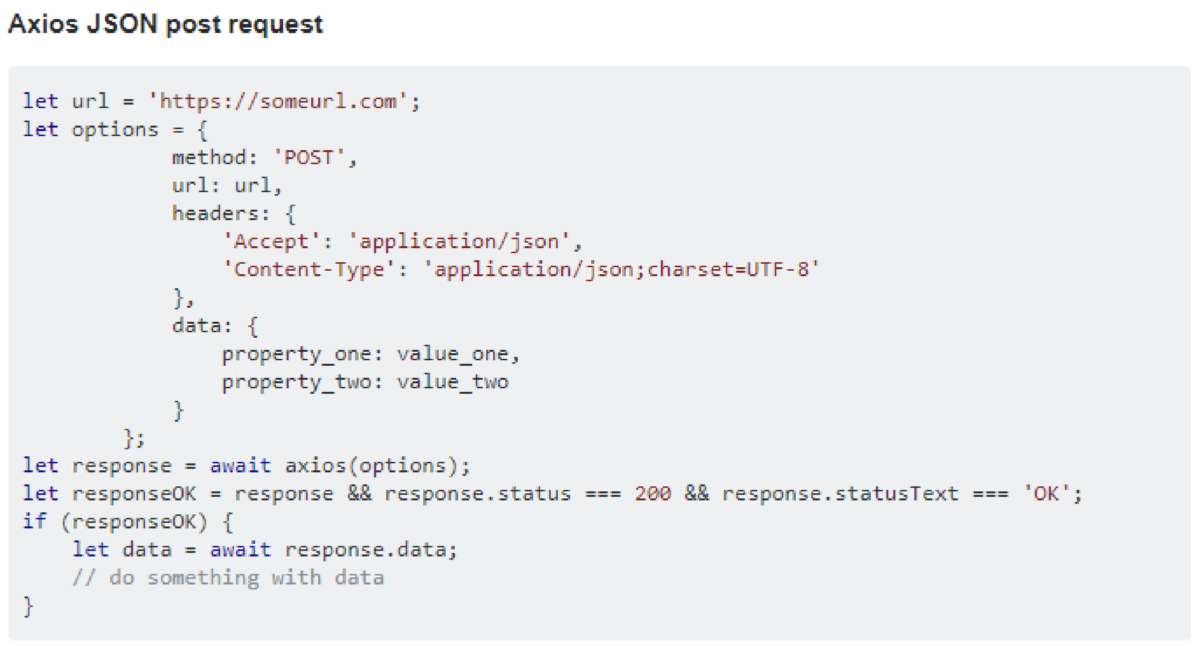
What is difference between Axios and Fetch?


Fetch and Axios are very similar in functionality, but for more backwards compatibility Axios seems to work better (fetch doesn't work in IE 11 for example, check this post)

Also, if you work with JSON requests, the following are some differences I stumbled upon with.

Fetch JSON post request

```
let url = 'https://someurl.com';
let options = {
            method: 'POST',
            mode: 'cors',
            headers: {
                'Accept': 'application/json',
                'Content-Type': 'application/json;charset=UTF-8'
            },
            body: JSON.stringify({
                property_one: value_one,
                property_two: value_two
            })
        };
let response = await fetch(url, options);
let responseOK = response && response.ok;
if (responseOK) {
    let data = await response.json();
    // do something with data
}
```



Axios JSON post request

```
let url = 'https://someurl.com';
let options = {
            method: 'POST',
            url: url,
            headers: {
                'Accept': 'application/json',
                'Content-Type': 'application/json;charset=UTF-8'
            },
            data: {
                property_one: value_one,
                property_two: value_two
            }
        };
let response = await axios(options);
let responseOK = response && response.status === 200 && response.statusText === 'OK';
if (responseOK) {
    let data = await response.data;
    // do something with data
}
```

So:

- Fetch's **body** = Axios' **data**
- Fetch's body has to be **stringified**, Axios' data contains the **object**
- Fetch **has no url** in request object, Axios **has url** in request object
- Fetch request function includes the **url as parameter**, Axios request function **does not include the url as parameter**.
- Fetch request is **ok** when response object contains the **ok property**, Axios request is **ok** when **status is 200** and **statusText is 'OK'**
- To get the json object response: in fetch call the **json() function** on the response object, in Axios get **data property** of the response object.

Issues with Fetch API

1. Handing error with fetch api.
2. Getting api response in 2 steps.
3. No timeout functionality available.
4. Cancelling request.
5. Fetch does not support upload progress.
6. No cookies by default

How to include bootstrap css and js in reactjs app?

If you are new to React and using **create-react-app cli** setup. Then run the below NPM command to include the latest version of bootstrap.

npm install --save bootstrap
or

npm install --save bootstrap@4.0.0-alpha.6

Then add the following import statement to **index.js** file

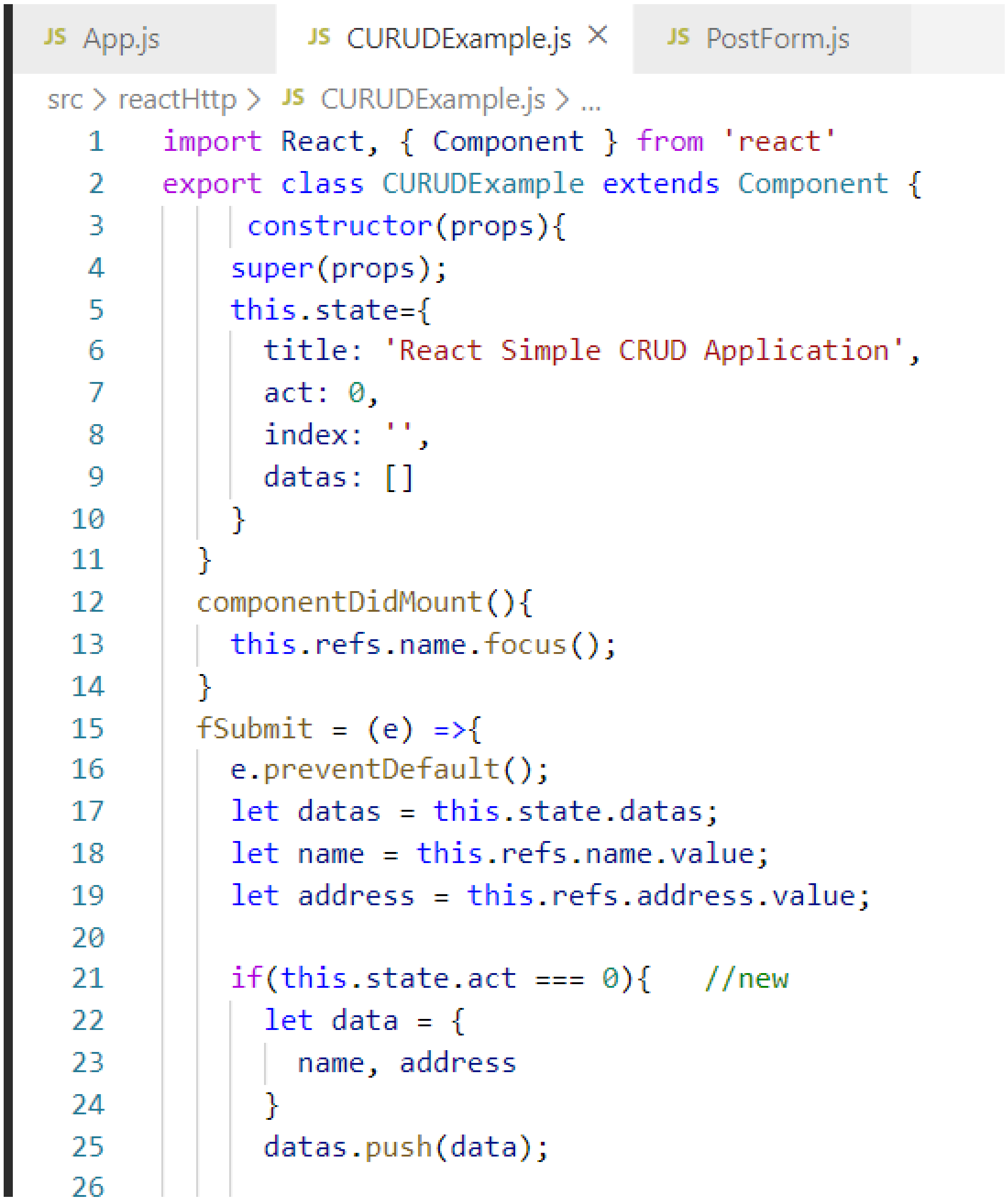
```
import '../node_modules/bootstrap/dist/css/bootstrap.min.css';
```

or

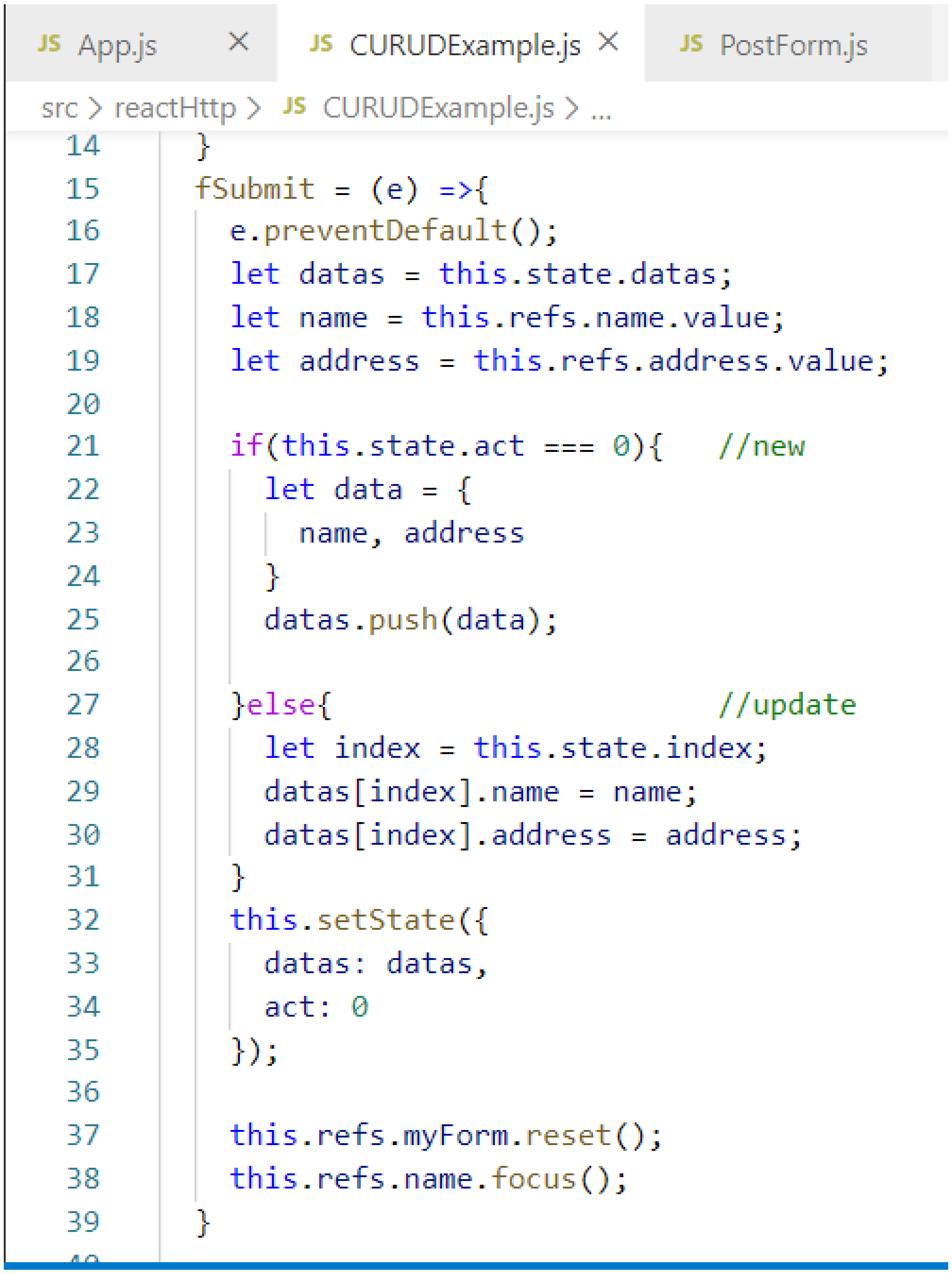
```
import 'bootstrap/dist/css/bootstrap.min.css';
```

don't forget to use **className** as attribute (react uses **className** as attribute instead of **class**)

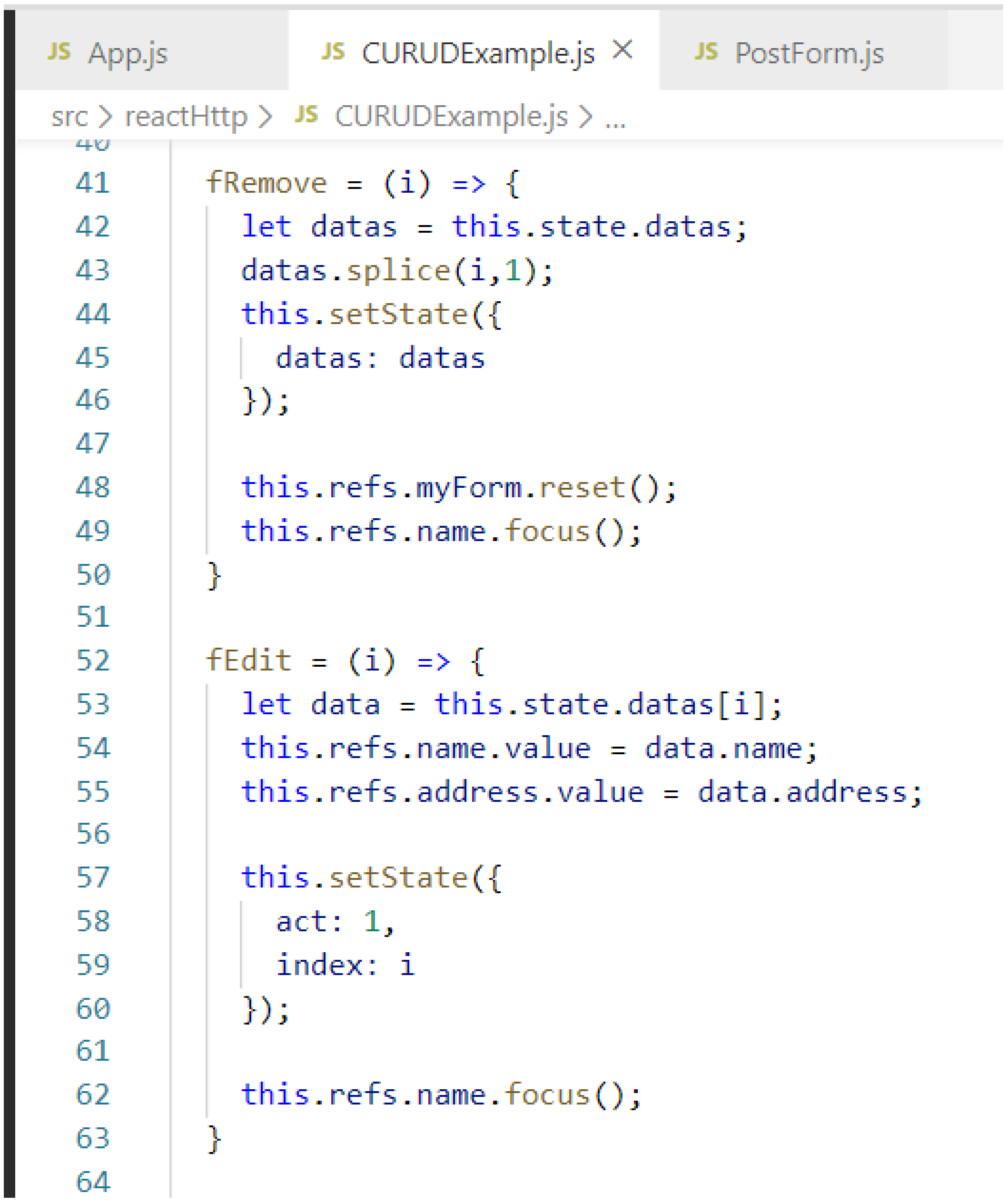
CRUD Example with array



```
import React, { Component } from 'react'
export class CURUDExample extends Component {
        constructor(props){
        super(props);
        this.state={
          title: 'React Simple CRUD Application',
          act: 0,
          index: '',
          datas: []
        }
      }
      componentDidMount(){
        this.refs.name.focus();
      }
      fSubmit = (e) =>{
        e.preventDefault();
        let datas = this.state.datas;
        let name = this.refs.name.value;
        let address = this.refs.address.value;

        if(this.state.act === 0){   //new
          let data = {
            name, address
          }
          datas.push(data);

```



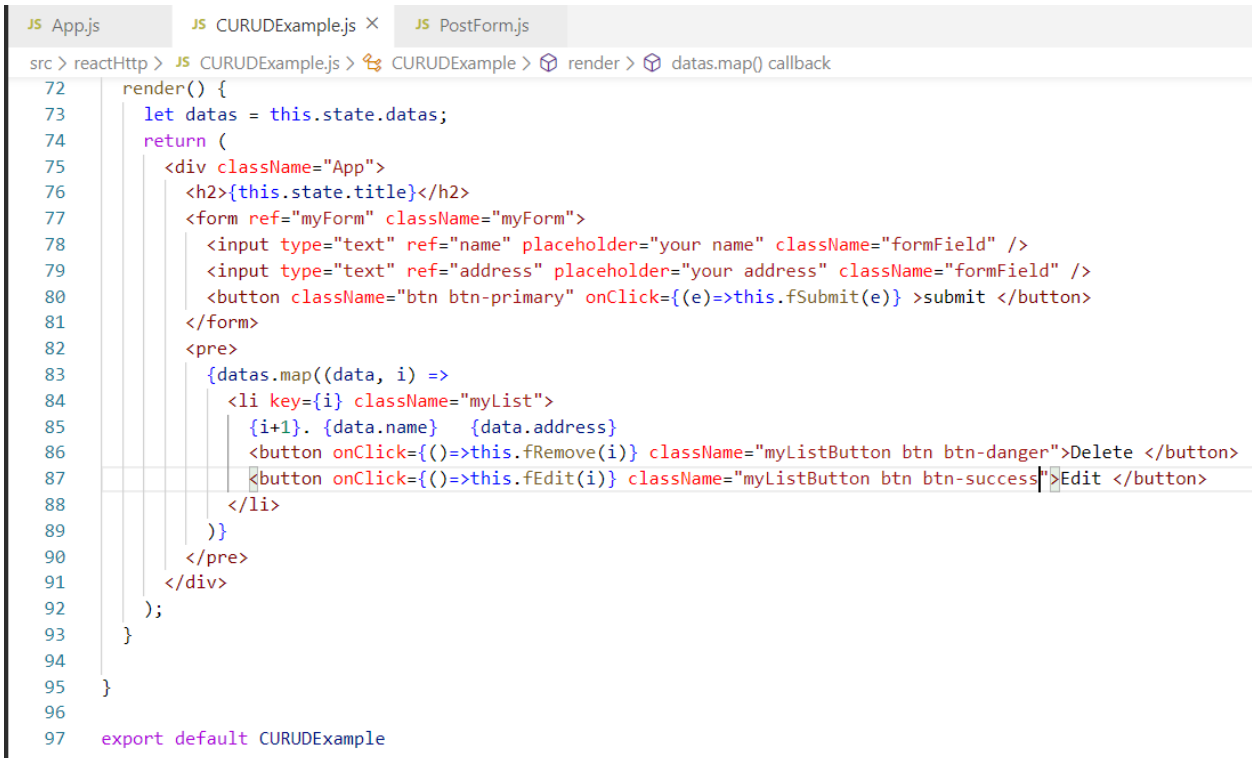
```
    }
    fSubmit = (e) =>{
      e.preventDefault();
      let datas = this.state.datas;
      let name = this.refs.name.value;
      let address = this.refs.address.value;

      if(this.state.act === 0){   //new
        let data = {
          name, address
        }
        datas.push(data);

      }else{                      //update
        let index = this.state.index;
        datas[index].name = name;
        datas[index].address = address;
      }
      this.setState({
        datas: datas,
        act: 0
      });

      this.refs.myForm.reset();
      this.refs.name.focus();
    }
```



JS App.js JS CURUDExample.js × JS PostForm.js

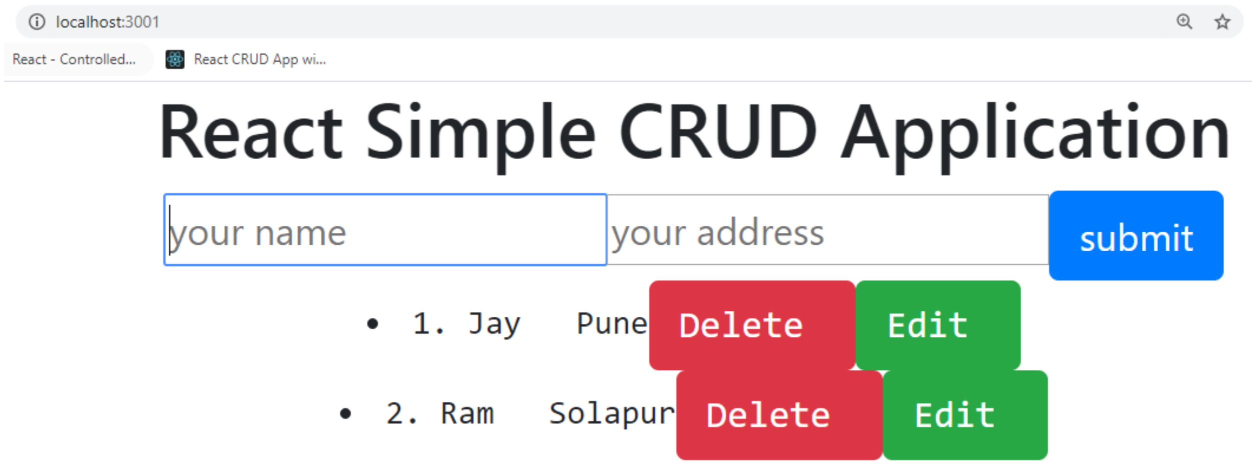
src > reactHttp > JS CURUDExample.js > ...

```
  fRemove = (i) => {
    let datas = this.state.datas;
    datas.splice(i,1);
    this.setState({
      datas: datas
    });

    this.refs.myForm.reset();
    this.refs.name.focus();
  }

  fEdit = (i) => {
    let data = this.state.datas[i];
    this.refs.name.value = data.name;
    this.refs.address.value = data.address;

    this.setState({
      act: 1,
      index: i
    });

    this.refs.name.focus();
  }

```


App.js
CURUDExample.js
PostForm.js
src > reactHttp > CURUDExample.js > CURUDExample > render > datas.map() callback

```
  render() {
    let datas = this.state.datas;
    return (
      <div className="App">
        <h2>{this.state.title}</h2>
        <form ref="myForm" className="myForm">
          <input type="text" ref="name" placeholder="your name" className="formField" />
          <input type="text" ref="address" placeholder="your address" className="formField" />
          <button className="btn btn-primary" onClick={(e)=>this.fSubmit(e)} >submit </button>
        </form>
        <pre>
          {datas.map((data, i) =>
            <li key={i} className="myList">
              {i+1}. {data.name}   {data.address}
              <button onClick={()=>this.fRemove(i)} className="myListButton btn btn-danger">Delete </button>
              <button onClick={()=>this.fEdit(i)} className="myListButton btn btn-success">Edit </button>
            </li>
          )}
        </pre>
      </div>
    );
  }

}

export default CURUDExample
```

OutPut:



React Simple CRUD Application

```
• 1. Jay    Pune
• 2. Ram    Solapur
```