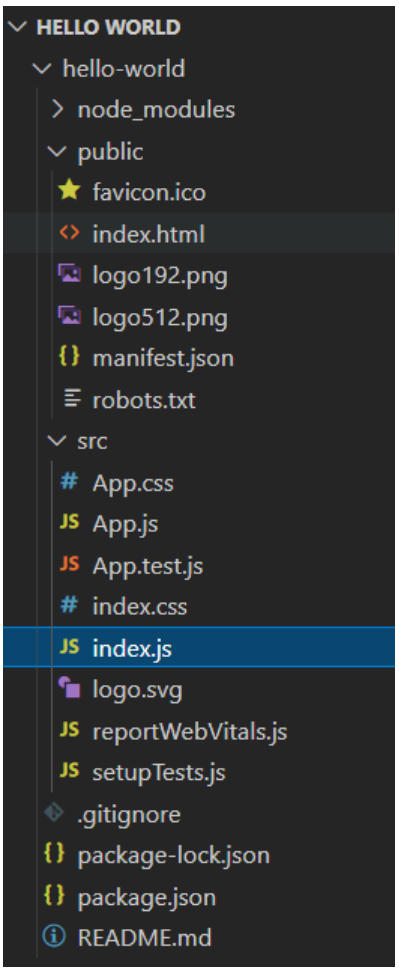
React Course

Starting of the project

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

How the execution flow works?

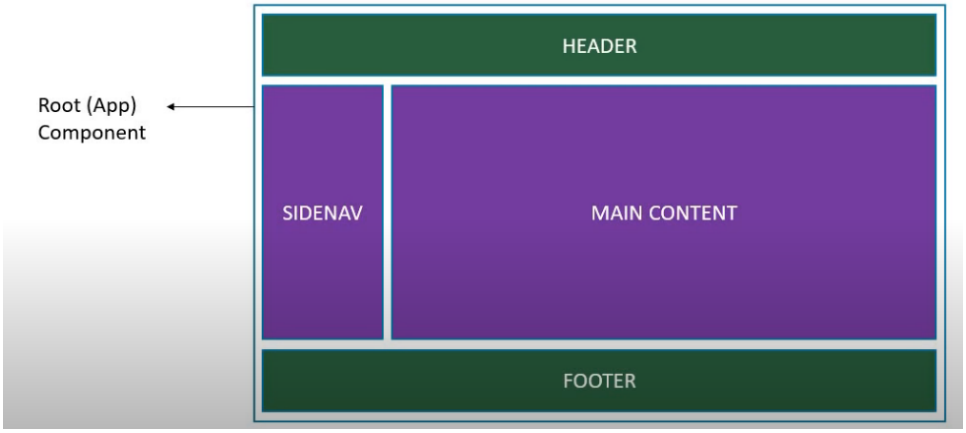


First it will go to the public directory and go to the index HTML file and there you see a root node then it automatically goes to the SRC folder and then open the index JS file and there you see what component you want to render and also it has to point the directory of the public index file.

Component in React

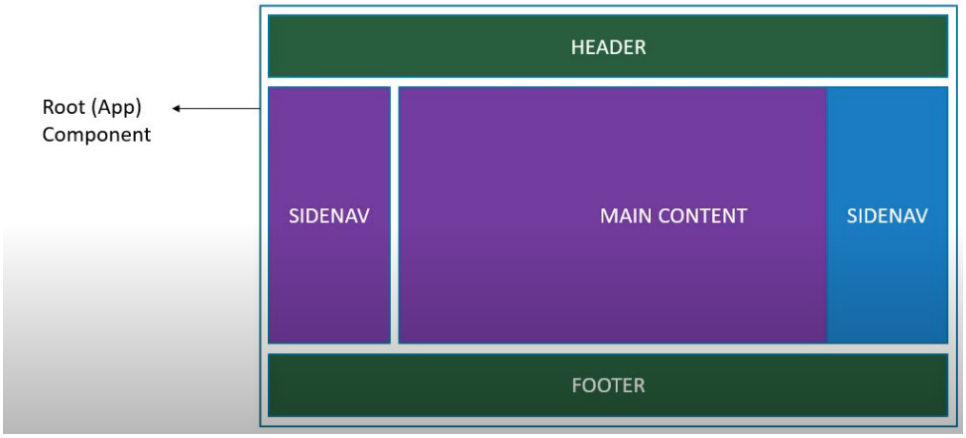
You can have multiple components in the project and one main or root component which encapsulates all the other components.

Components



One component can have other components inside

Components



Component Types

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>;
  }
}
```

Components Summary

Components describe a part of the user interface

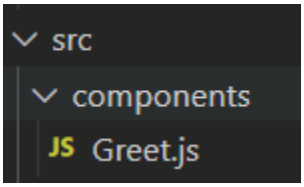
They are re-usable and can be nested inside other components

Two Types –

- Stateless Functional Components
- Stateful Class Components

Functional Components:

Below is the demonstration of the component



Below is the component I created which is using default namespaces.

```
1  import React from "react";
2
3  function Greet() {
4    return <h1>Ahmed</h1>;
5  }
6
7  export default Greet;
8
```

How you are going to use below is the attached image

```
import MyComponent from "./components/Greet";

ReactDOM.render(
  <React.StrictMode>
    <MyComponent />
  </React.StrictMode>,
  document.getElementById("root")
);

// If you want to start measuring performance in your ap
// to log results (for example: reportWebVitals(console.
```

You can use any name while importing the file when you are using the export default but when you are using the const exports then you must have to put the curly braces and exact same name. One more thing we encourage to use the arrow function as it has some additional features let me rewrite the above function into arrow function.

Below is the example if you are using the const export then you need to put the same name where you are using that function with the curly braces as shown in the image below

```
6  import { Greet } from "./components/Greet";
7
8  ReactDOM.render(
9    <React.StrictMode>
10     <Greet />
11   </React.StrictMode>,
12   document.getElementById("root")
13 );
```

Class Components:

It is basically an ES6. It will save the private information to that component.

Instead of just importing the react and component we need to import and we need to specify the render function inside this as shown in the below image

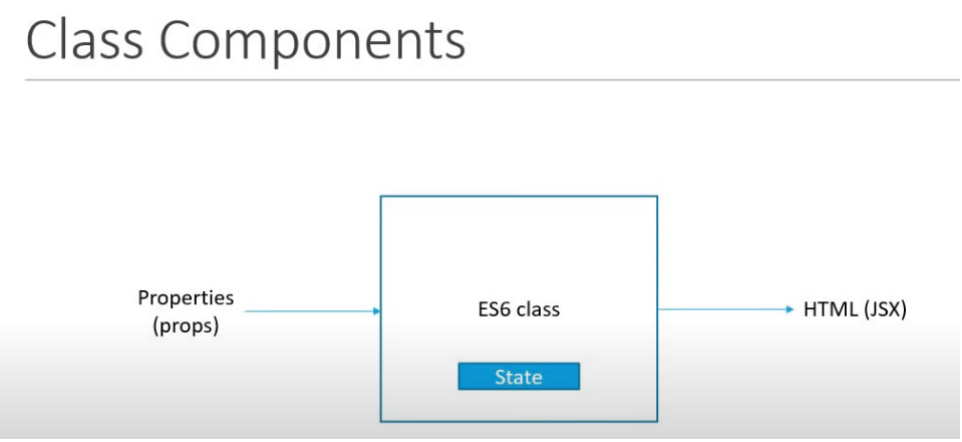
index.js M | App.js M | Welcome.js U | Greet.js U

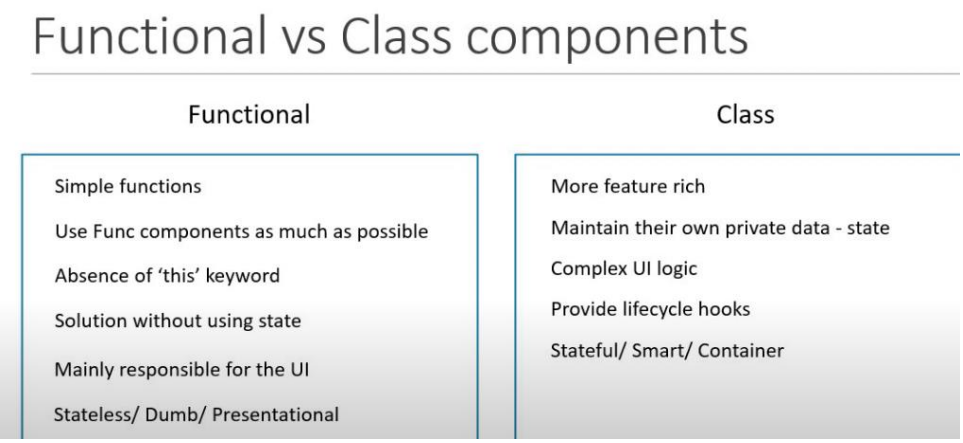
ello-world > src > components > Welcome.js > Welcome > render

```
1  import React, { Component } from "react";
2
3  class Welcome extends Component {
4    render() {
5      return <h1>Class Welcome</h1>;
6    }
7  }
8
9  export default Welcome;
10
```

```
import Welcome from "./components/Welcome";
function App() {
  return (
    <div className="App">
      <Greet />
      <Welcome />
    </div>
  );
```

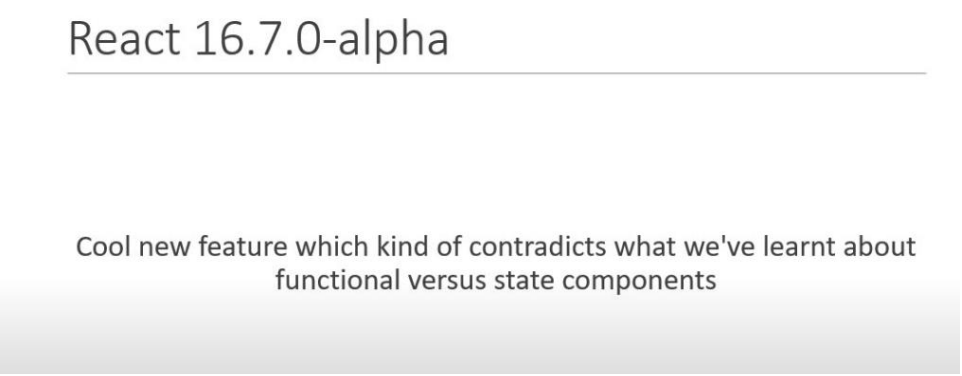
Below some of the use cases of the class based and function-based views





Hooks

It will contradict what we learn about functional views.



It is the feature that you can write the stateful features without writing a class.

Hooks

No breaking changes.

Completely opt-in & 100% backwards-compatible.

What ever we've learned so far in this series still holds good.

Component types - Functional components and Class components.

Using state, lifecycle methods and 'this' binding.

After understanding state, event binding and lifecycle hooks in class components.

How **this**, keyword will help you out and removes the mess which you will be seen in the class-based components.

JSX

JSX

JavaScript XML (JSX) – Extension to the JavaScript language syntax.

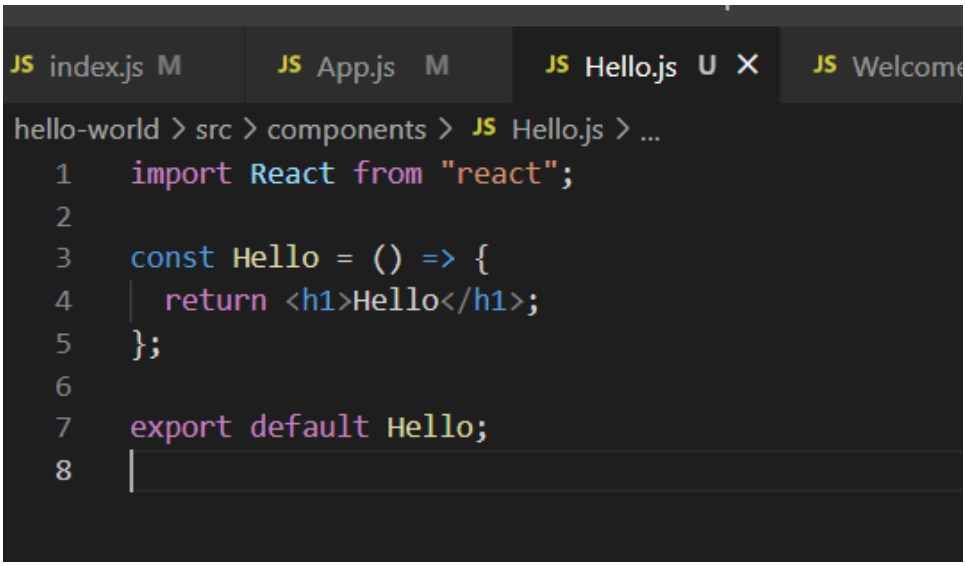
Write XML-like code for elements and components.

JSX tags have a tag name, attributes, and children.

JSX is not a necessity to write React applications.

JSX makes your react code simpler and elegant.

JSX ultimately transpiles to pure JavaScript which is understood by the browsers.



If you are using the JSX, then you can simply do the above code and if you don't want to use the JSX then the image after that image will display that thing.

```jsx
import Hello from "./components/Hello";

function App() {
  return (
    <div className="App">
      {/* <Greet /> */}
      {/* <Welcome /> */}
      <Hello />
    </div>
  );
}
```

index.js M | App.js M | Hello.js U | Welcome.js U | Greet.js U

hello-world > src > components > Hello.js > Hello

```jsx
import React from "react";

const Hello = () => {
  //   return (<h1>Hello</h1>);
  // createElement is to create the element and the next is to pass the values which you gave.
  // and the third parameter is to create the next or inner new element.
  //   return React.createElement("div", null, "<h1>Muhammad Ahmed</h1>");
  return React.createElement(
    "div",
    null,
    React.createElement("h1", null, "Muhammad Ahmed")
  );
};

export default Hello;
```

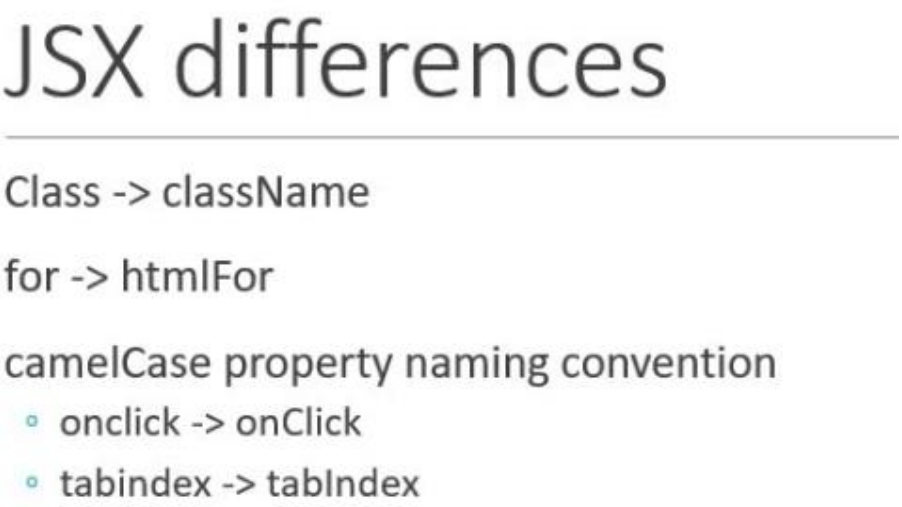
index.js M | App.js M | Hello.js U | Welcome.js U | Greet.js U

hello-world > src > components > Hello.js > Hello

```jsx
import React from "react";

const Hello = () => {
  //   return (<h1>Hello</h1>);
  // createElement is to create the element and the next is to pass the values which you gave.
  // and the third parameter is to create the next or inner new element.
  //   return React.createElement("div", null, "<h1>Muhammad Ahmed</h1>");
  return React.createElement(
    "div",
    { id: "hello", className: "dummyClass" },
    React.createElement("h1", null, "Muhammad Ahmed")
  );
};

export default Hello;
```

The above code is the property we used inside that div for example we want to add the id and the class name we want to add inside that div.

```
▶ <head>…</head>
… ▼ <body> == $0
    <noscript>You need to enable JavaScript to run this app.</noscript>
  ▼ <div id="root">
    ▼ <div class="App">
      ▼ <div id="hello" class="dummyClass">
          <h1>Muhammad Ahmed</h1>
        </div>
      </div>
    </div>
    <!--
```



Props

The Number 1 thing is props are immutable which means you cannot change the property of the variables inside this.

In functional base components you pass the value from the rendering component towards the component where the value is to be used. While on the class-based components you use **this** keyword and with the props you can access these values inside that and one thing if you have more than one property then you can also pass but make you have one parent element present because the render button needs one main div in order to pass the values.

ndex.js M | JS App.js M × | JS Hello.js U

o-world > src > JS App.js > App

```
import logo from "./logo.svg";
import "./App.css";
import Greet from "./components/Gr
import Welcome from "./components/
import Hello from "./components/He

function App() {
  return (
    <div className="App">
      {/* <Greet /> */}
      {/* <Welcome /> */}
      <Hello name="Muhammad" />
      <Hello name="Ahmed" />
      <Hello name="Zahid" />
    </div>
  );
}

export default App;
```

S index.js M | JS App.js M | JS Hello.js U × | JS W

ello-world > src > components > JS Hello.js > Hello

```
1  import React from "react";
2
3  const Hello = (props) => {
4    console.log(props);
5    return <h1>Hello, {props.name}</h1>;
6    // createElement is to create the element
```

```
function App() {
  return (
    <div className="App">
      {/* <Greet /> */}
      <Welcome name="Muhammad" />
      <Welcome name="Mubashir" />

      <Hello name="Muhammad" />
      <Hello name="Ahmed" />
      <Hello name="Zahid" />
    </div>
  );
```

```
class Welcome extends Component {
  render() {
    return <h1>Class Welcome {this.props.name}</h1>;
  }
```

And if there is a scenario that you don't know how many of the properties are passing you can do this as follows:

Props and then with dot it is called children attribute.

x.js M JS App.js M JS Hello.js U

orld > src > JS App.js > App

```jsx
import logo from "./logo.svg";
import "./App.css";
import Greet from "./components/Greet";
import Welcome from "./components/Welco
import Hello from "./components/Hello";

function App() {
  return (
    <div className="App">
      {/* <Greet /> */}
      <Welcome name="Muhammad" />
      <Welcome name="Mubashir" />

      <Hello name="Muhammad">
        <p>This is the children</p>
      </Hello>
      <Hello name="Ahmed">
        <button>click me</button>
      </Hello>
      <Hello name="Zahid" />
    </div>
  );
```

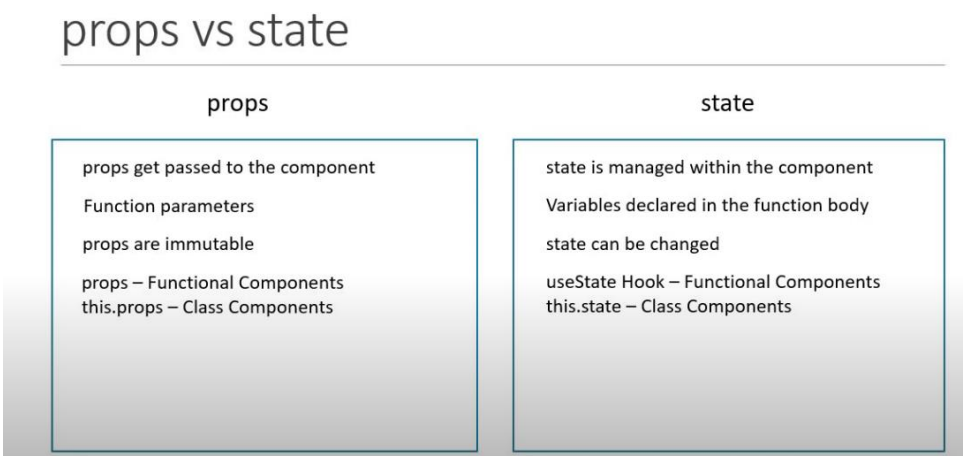
index.js M JS App.js M JS Hello.js U JS W

ello-world > src > components > JS Hello.js > Hello

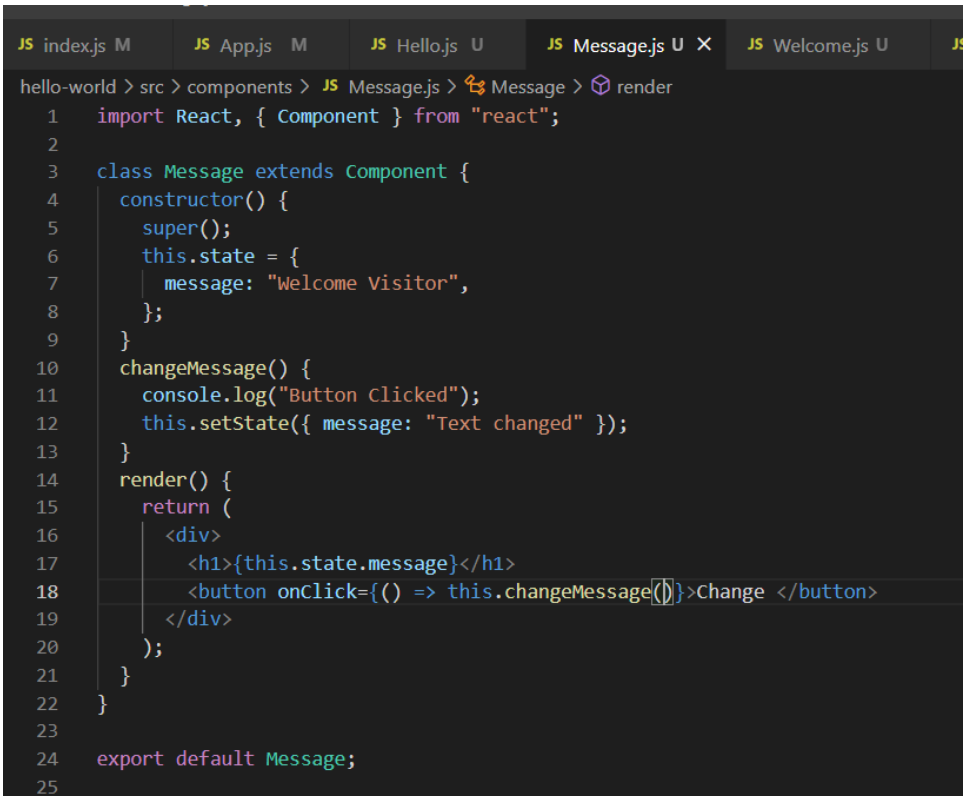
```jsx
1   import React from "react";
2
3   const Hello = (props) => {
4     console.log(props);
5     return (
6       <div>
7         <h1>Hello, {props.name}</h1>
8         {props.children}
9       </div>
10    );
```

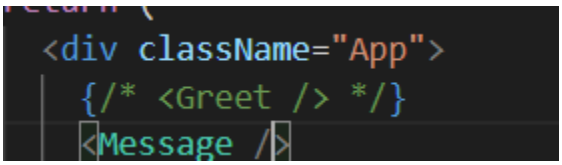
State

Now, let's talk about the state as we know that the props are those properties that cannot change its values.



We can do by using the below code





Set State

Extension ES7, React, Redux in visual studio code.

Rce then tab it will create the all the code snippet.

Rc the react constructor.

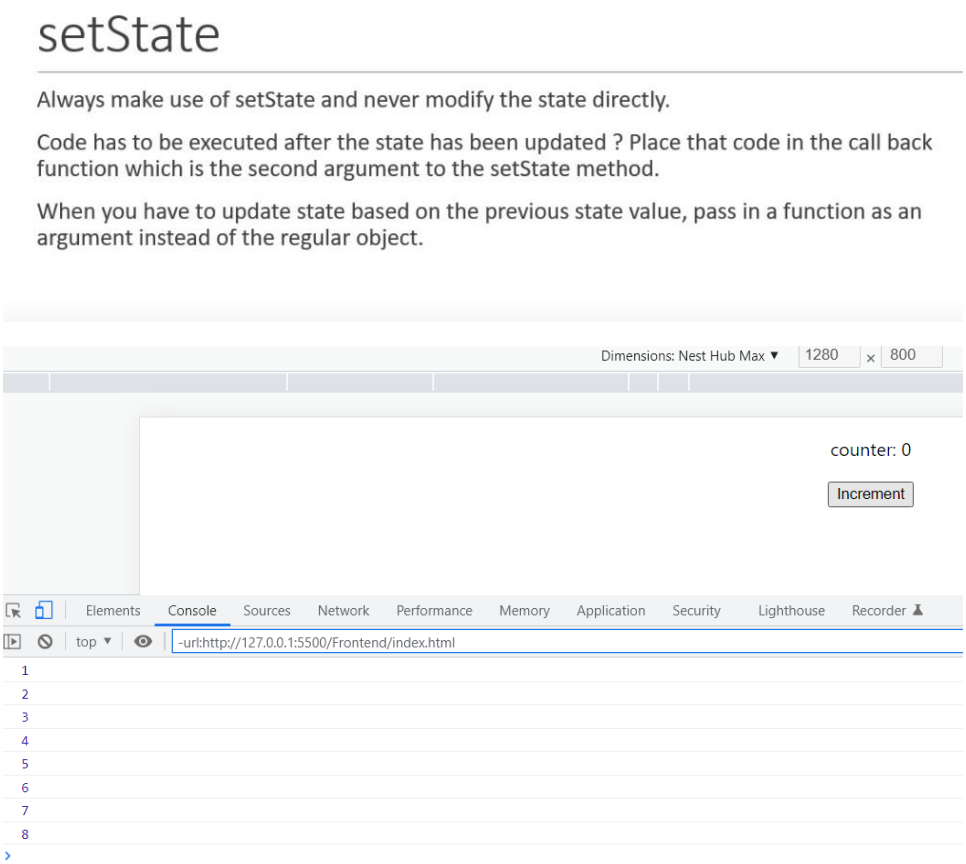
Never change the value of the state directory instead use the set state function because the set state function will update the UI if you just update the value, it cannot change the value inside the UI.

Important thing to note, console value is one is less behind then the webpage value.

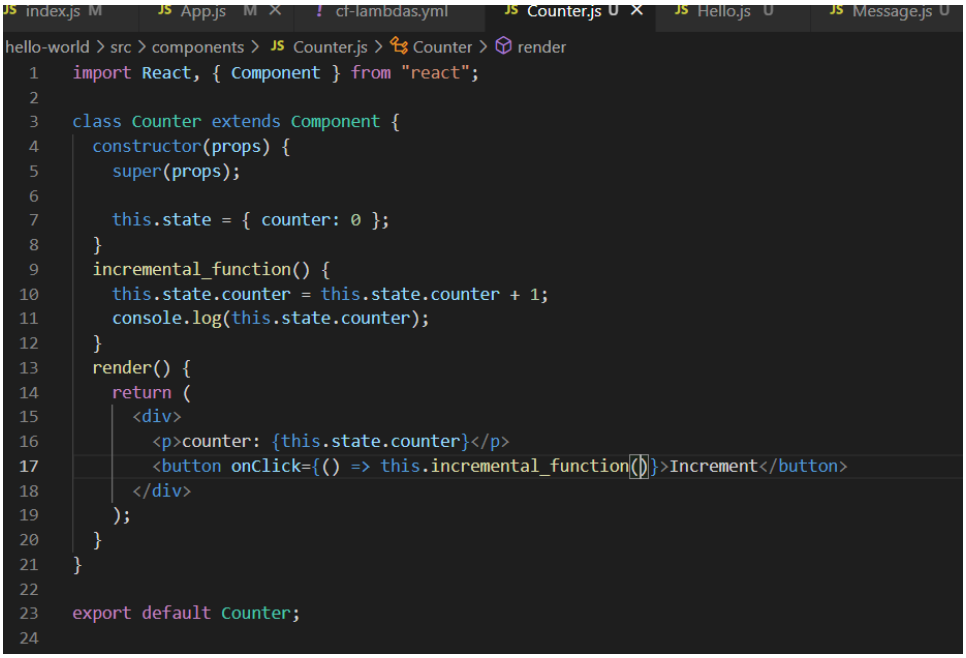
Sometime you want to call a function when a particular state is changed so how you can do this you can pass a call back function which is an arrow function in it. Then the callback value and the display value are one.

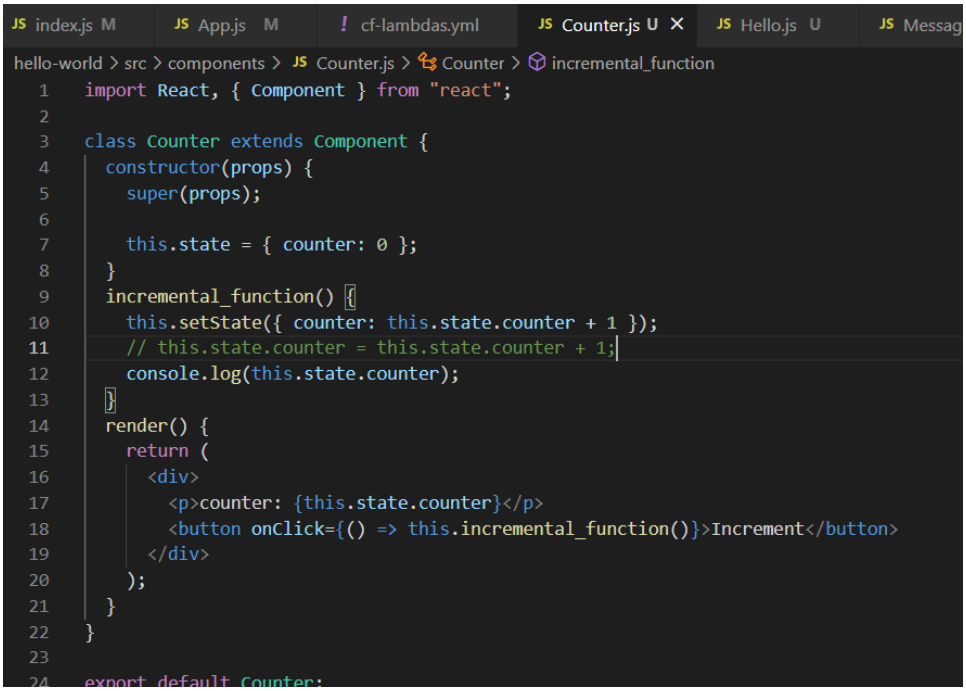
If you want to increment the value based on the previous value then, you can the arrow function if you don't want to use that then the value which you are going to increment does not do that because it is updating the values in parallel.

You can use the prop inside that function also.

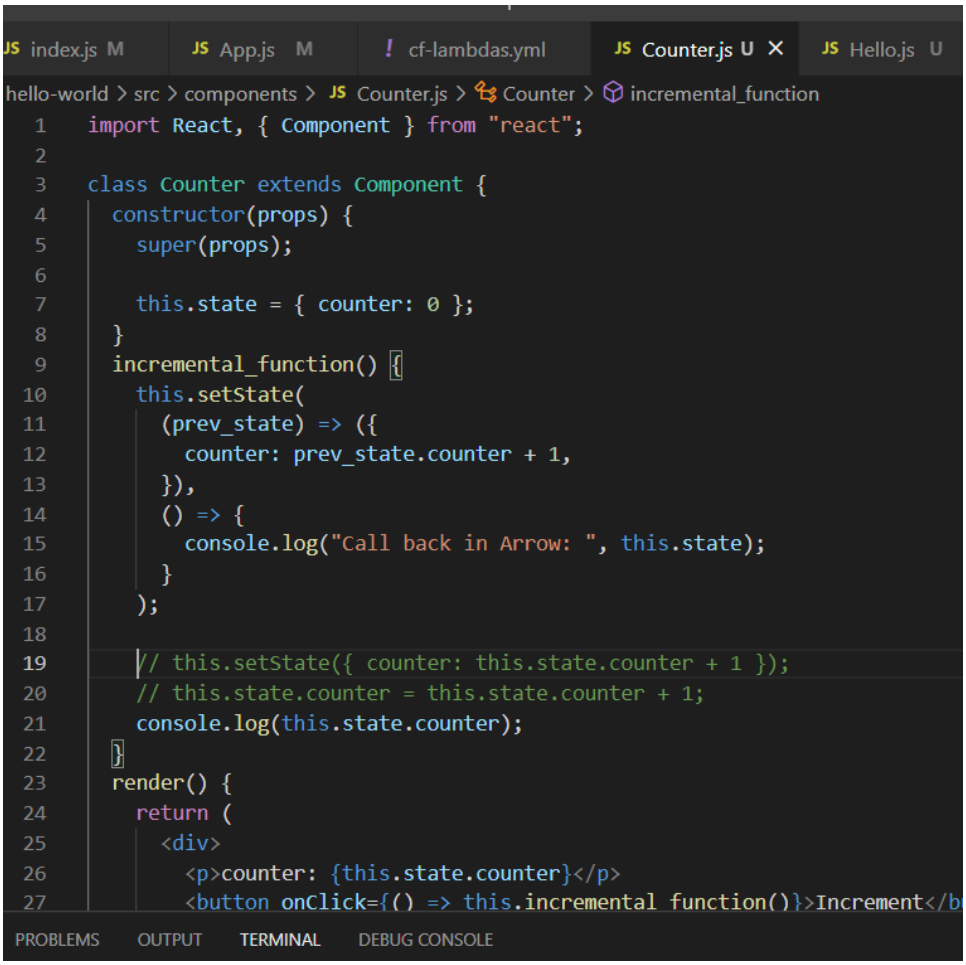


No value is incremented using the below code instead we have to use the set state function if you want to update the value in the UI.





As we described earlier when we are using the previous and incremental into the next then we need to use the arrow function.



This is the right method of doing this. One thing to note that when you referring or changing the value then you need to put the parenthesis and then you can access this value and put the value used.

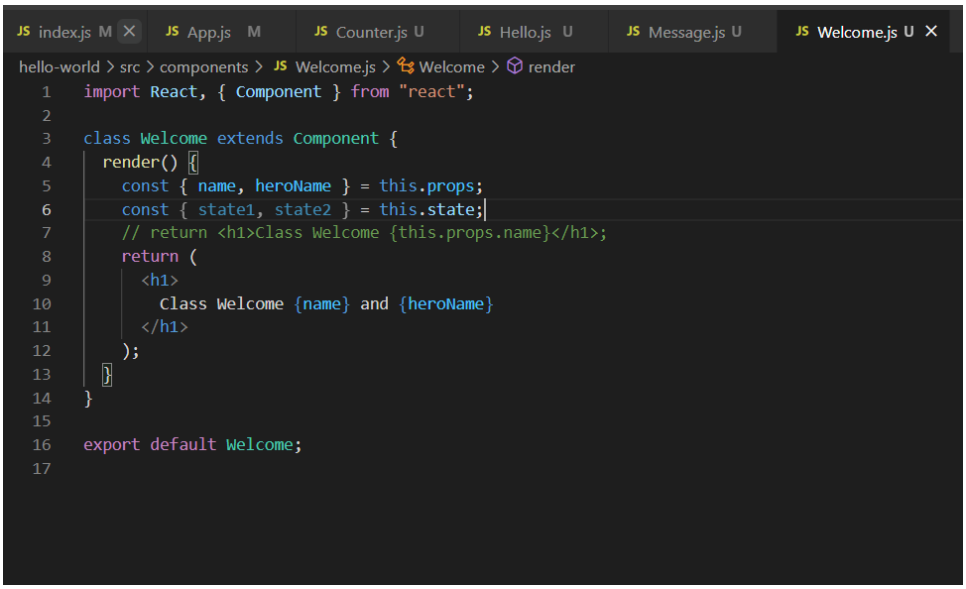
De structuring props and states:

There is an ES6 feature which helps to de structure the array properties into individual elements.

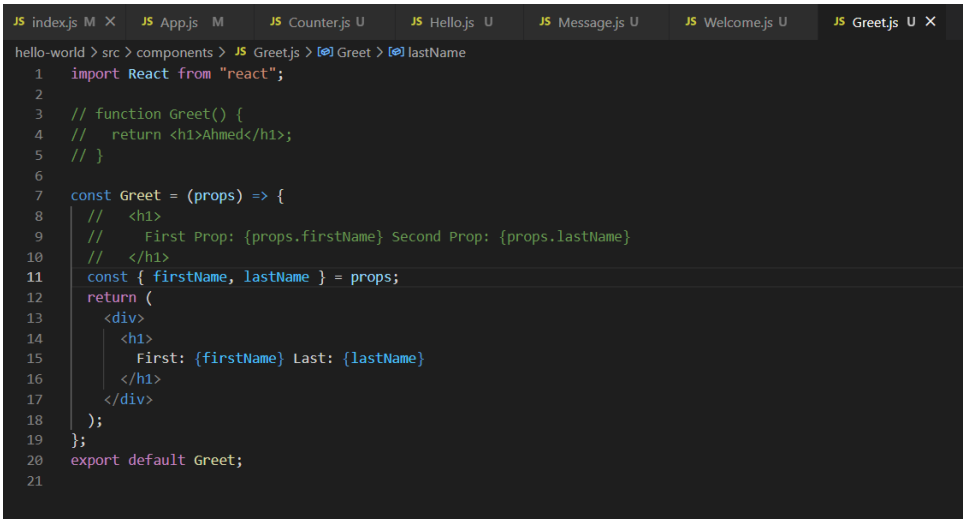
We use curly braces for unpacking the data.

You can have two different ways of de structuring the props in functional component.

Class Based usage:



Functional based components:



Event Handling:

- Event handler is a function, not a function class so you cannot use the parenthesis.

We can put the event handler function inside a class-based component and function-based component also.

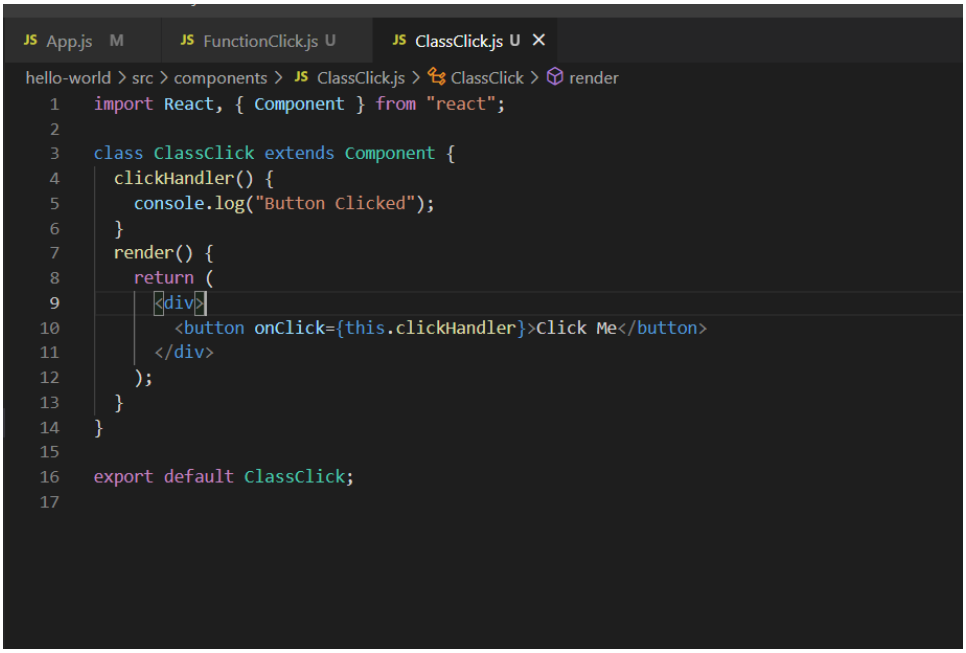
Function based components

p.js M JS FunctionClick.js U ×

world > src > components > JS FunctionClick.js > FunctionClick > clic

```
import React from "react";

function FunctionClick() {
  function clickme() {
    console.log("Function is clicked");
  }
  return (
    <div>
      <button onClick={clickme}>Click me</button>
    </div>
  );
}
export default FunctionClick;
```

Class based components



Binding Event Handlers:

Simple Method with the bind keyword

Second you can use arrow function

Both the above methods have performance issues.

Third one to use bind the event handler inside the constructor instead of button component.

Binding in the constructor is just for one and if you bind in the button component then it will generate each time new binding so the lesson is it is a good approach binding in the constructor.

Or you can have the fourth approach where you can write down the arrow function inside the class.

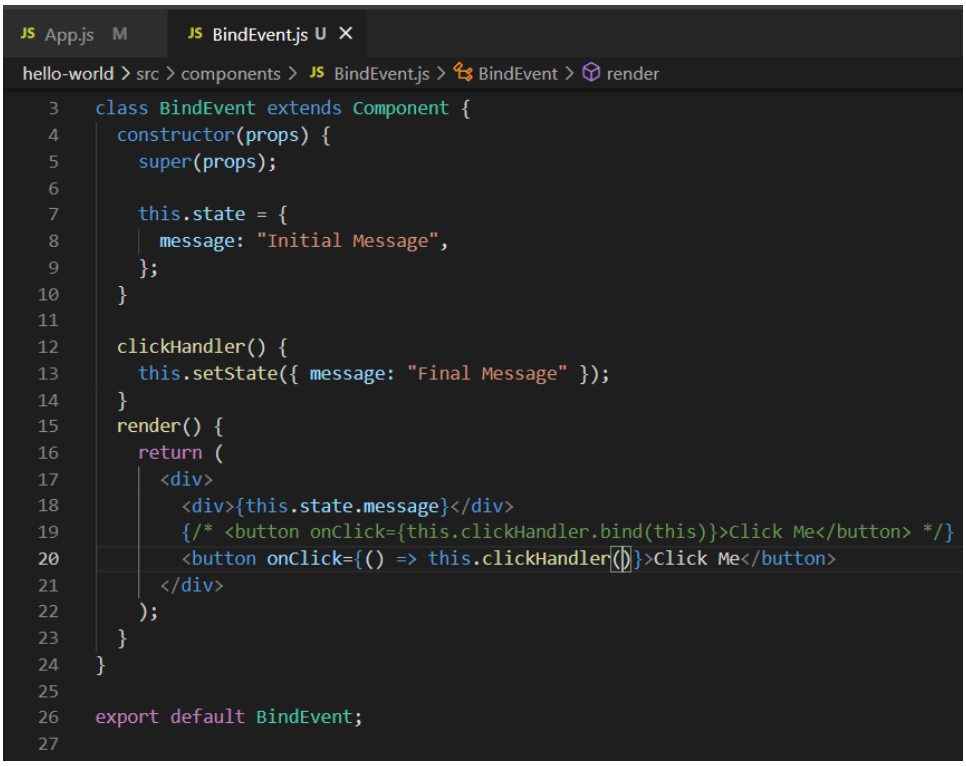
First approach

App.js M BindEvent.js U

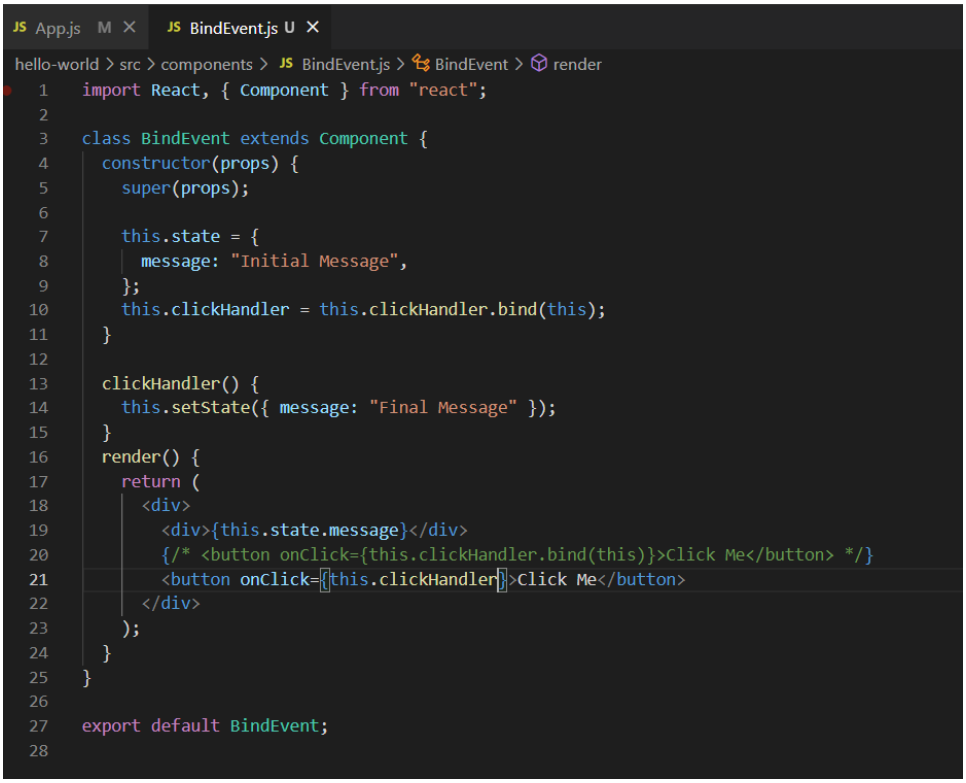
hello-world > src > components > BindEvent.js > BindEvent > render

```
1   import React, { Component } from "react";
2
3   class BindEvent extends Component {
4     constructor(props) {
5       super(props);
6
7       this.state = {
8         message: "Initial Message",
9       };
10    }
11
12    clickHandler() {
13      this.setState({ message: "Final Message" });
14    }
15    render() {
16      return (
17        <div>
18          <div>{this.state.message}</div>
19          <button onClick={this.clickHandler.bind(this)}>Click Me</button>
20        </div>
21      );
22    }
23  }
24
25  export default BindEvent;
26
```

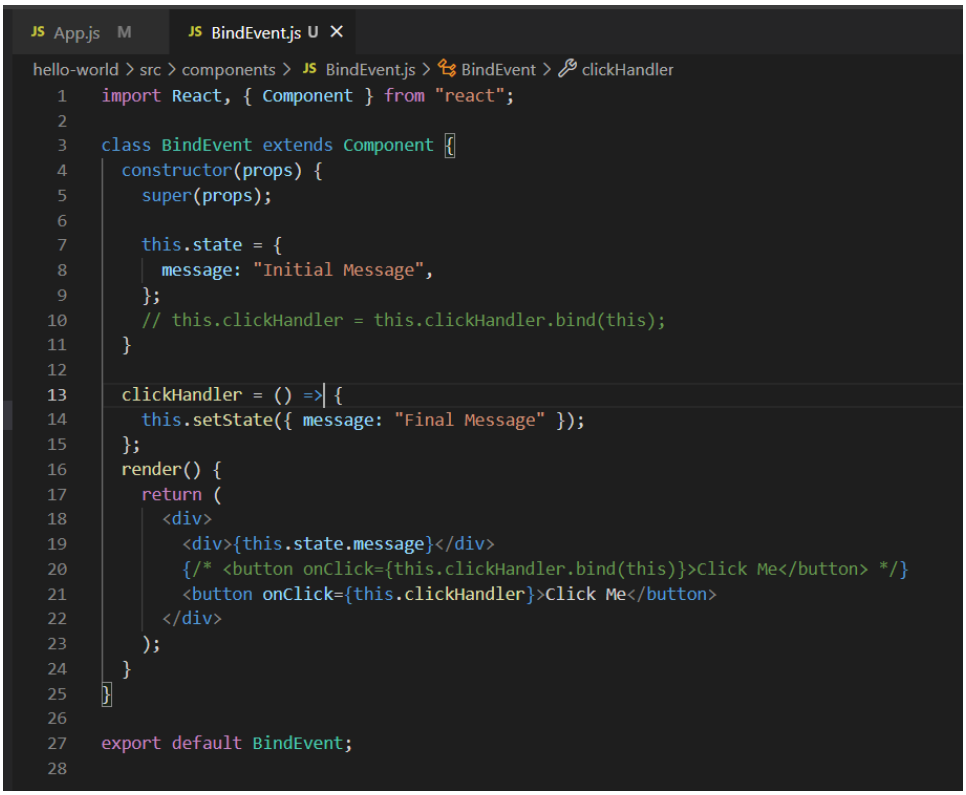
Second approach



Third approach



Fourth approach



Methods as Props

If you want to pass the data from the parent to the child component then you can do it using the properties and what if you want to pass the values from the child values to the parent component.

Something we want to build that when the component of the child clicks then it will call the parent component and display the result.

You can also pass the variable instead of the function. We can use the arrow function and pass as many parameters as you want to the parent component.

ParentComponent.js U | ChildComponent.js U | App.js M

hello-world > src > App.js > App

```
import logo from "./logo.svg";
import "./App.css";
import Greet from "./components/Greet";
import Welcome from "./components/Welcome";
import Hello from "./components/Hello";
import Message from "./components/Message";
import Counter from "./components/Counter";
import FunctionClick from "./components/FunctionClick";
import ClassClick from "./components/ClassClick";
import BindEvent from "./components/BindEvent";
import ParentComponent from "./components/ParentComponent";
function App() {
  return (
    <div className="App">
      <ParentComponent />
      {/* <BindEvent /> */}
      {/* <FunctionClick /> */}
```

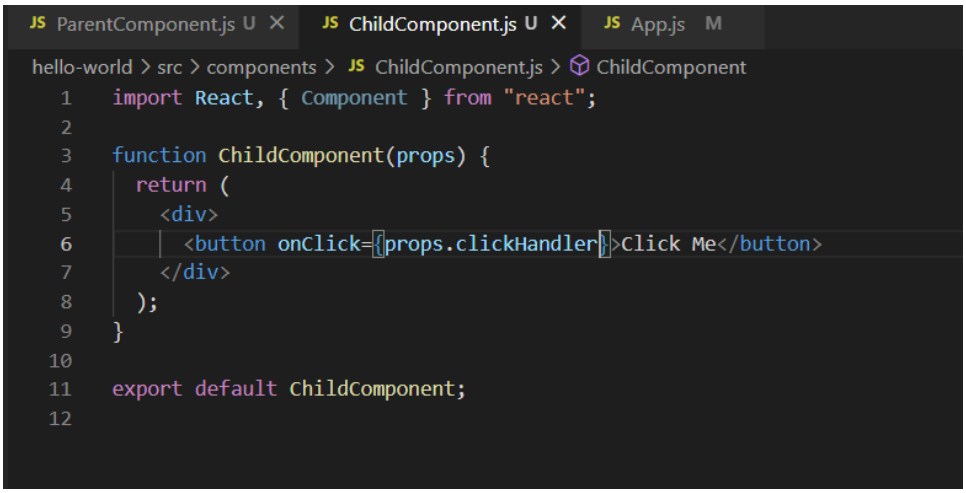
Parent Component

ParentComponent.js U | ChildComponent.js U | App.js M

hello-world > src > components > ParentComponent.js > ParentComponent > clickHandler

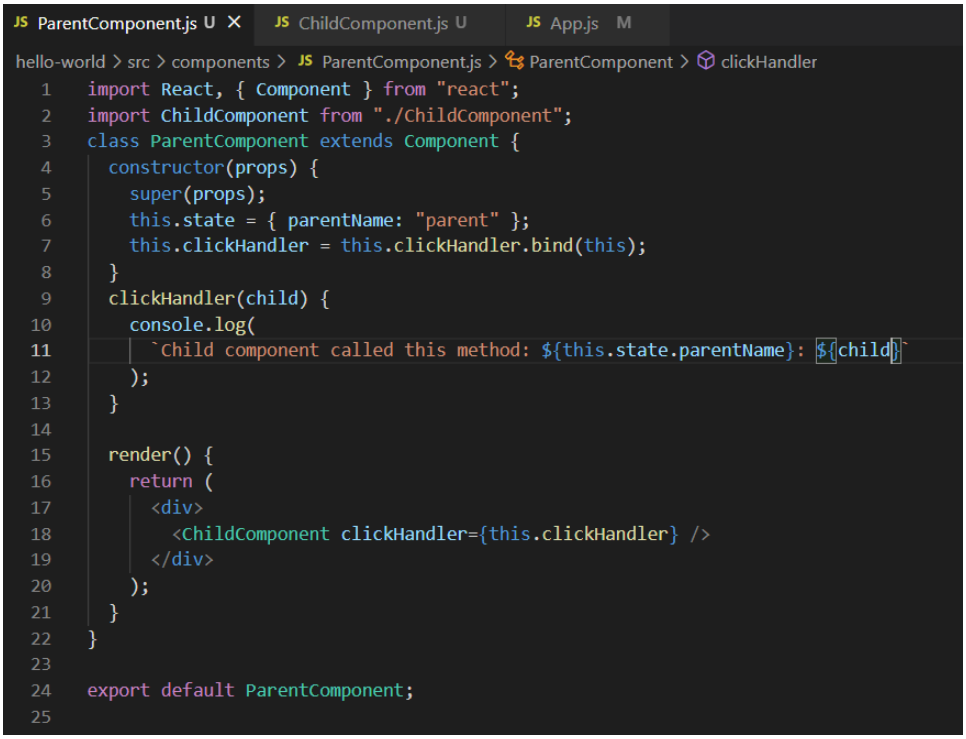
```
import React, { Component } from "react";
import ChildComponent from "./ChildComponent";
class ParentComponent extends Component {
  constructor(props) {
    super(props);
    this.state = { parentName: "parent" };
    this.clickHandler = this.clickHandler.bind(this);
  }
  clickHandler() {
    console.log(`Child component called this method: ${this.state.parentName}`);
  }

  render() {
    return (
      <div>
        <ChildComponent clickHandler={this.clickHandler} />
      </div>
    );
  }
}

export default ParentComponent;
```

Child Component



Now what If we want to pass the values inside that function for that we use arrow function and pass the value inside that.

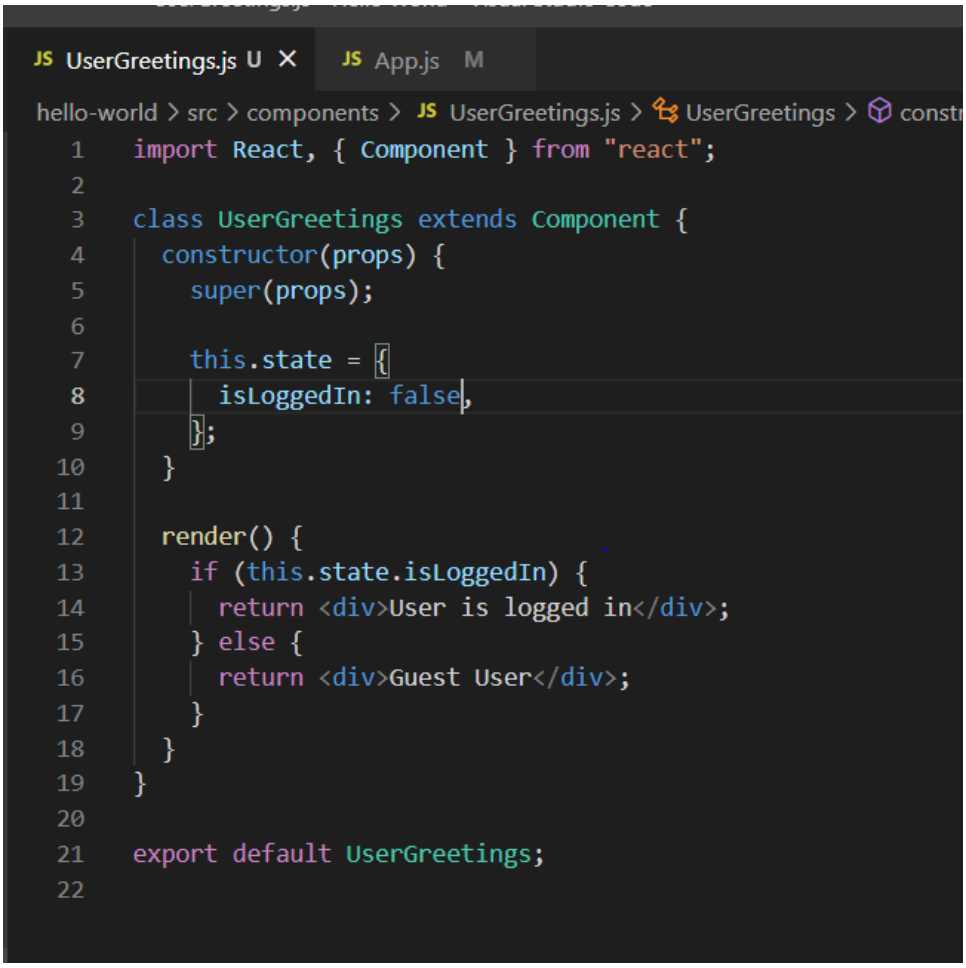
Parent Component



Child Component

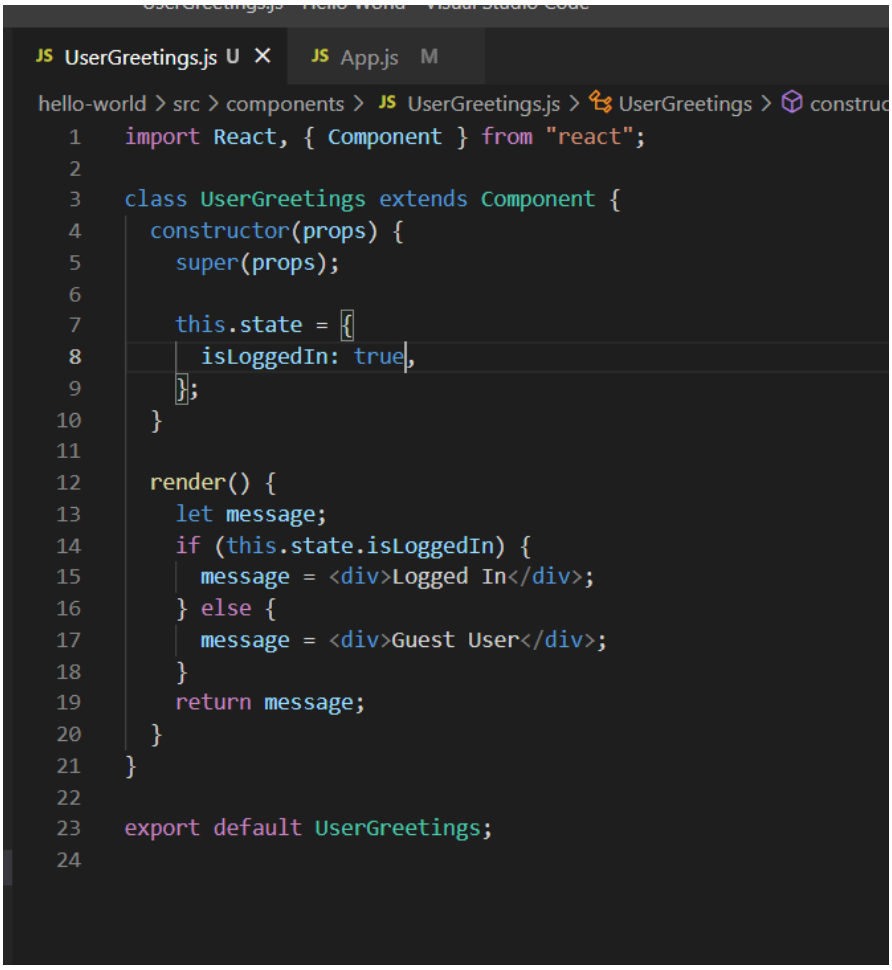
Conditional rendering:

If and else condition:

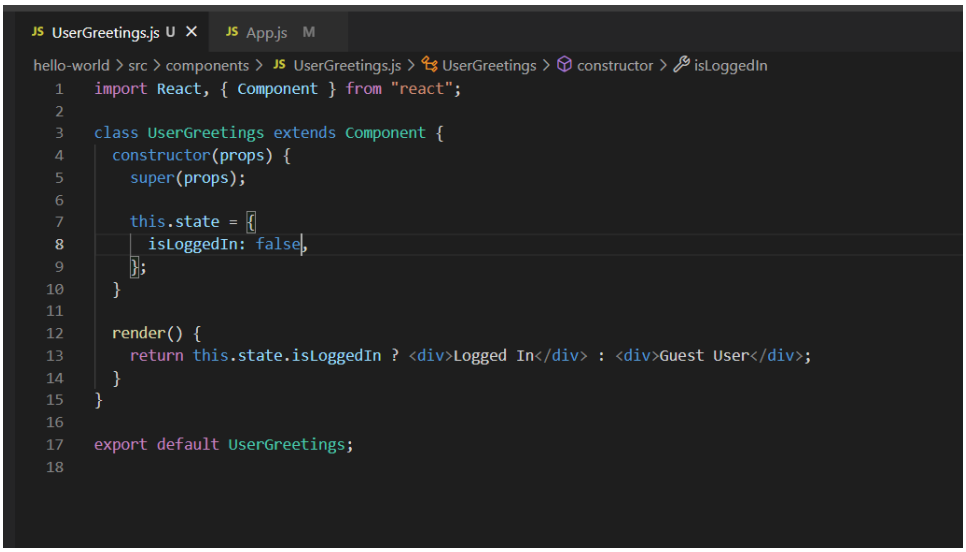


JSX is just the syntax sugar but it cannot used as a conditional statement.

Element Variable:

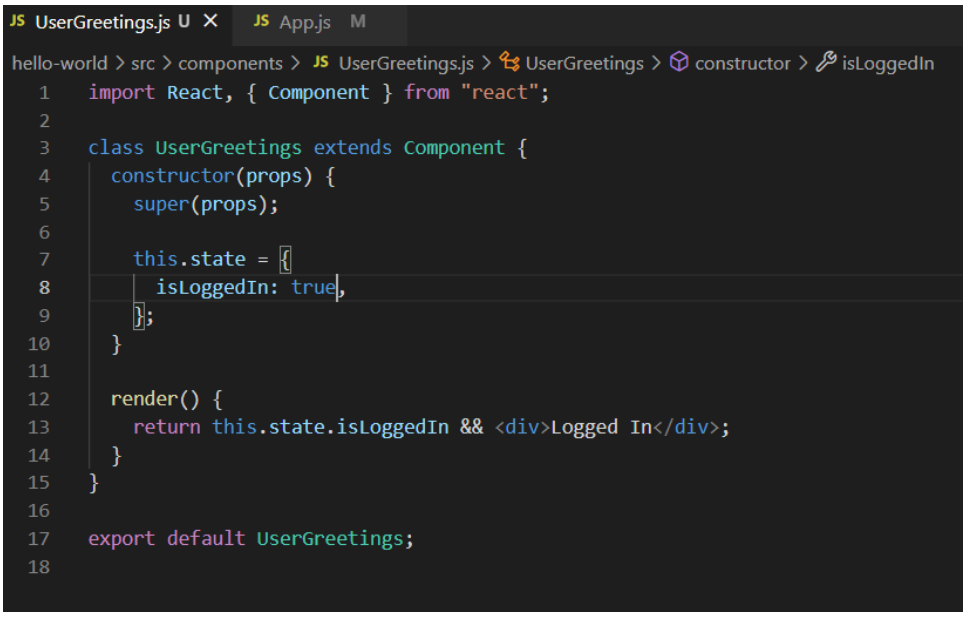


Ternary conditional operator:



Short circuit operator

If you need something like this to render the value or don't render the value for that we use short circuit operator.



List rendering

It is similar as working as the **map** function in the JavaScript. Curly braces are used for evaluating the JavaScript.

1st approach

2nd approach

App.js M NameList.js U

hello-world > src > components > NameList.js > NameList > namelist > nam

```
1   import React from "react";
2
3   function NameList() {
4     const names = ["Muhammad", "Ahmed", "Zahid"];
5     const namelist = names.map((name) => <h1>{name}</h1>);
6     return <div>{namelist}</div>;
7   }
8   export default NameList;
9
```

If you have array of the object, then what you can do

App.js M NameList.js U

hello-world > src > components > NameList.js > NameList

```
1   import React from "react";
2
3   function NameList() {
4     const name_information = [
5       {
6         name: "Muhammad",
7         age: 23,
8         time: "10 years",
9       },
10      {
11        name: "Ahmed",
12        age: 22,
13        time: "12 years",
14      },
15      {
16        name: "Zahid",
17        age: 19,
18        time: "13 years",
19      },
20    ];
21    const namelist = name_information.map((name_index) => (
22      <h1>
23        Name: {name_index.name} Age: {name_index.age} Time: {name_index.time}
24      </h1>
25    ));
26    return <div>{namelist}</div>;
27  }
28  export default NameList;
```

But the true method is to use separate component for that. By doing this we can separate out the Name List and then Person component so the main component is responsible for rendering the list and other component is responsible for rendering the items inside that component.

App.js M | NameList.js U | Person.js U

hello-world > src > components > NameList.js > NameList

```
import React from "react";
import Person from "./Person";
function NameList() {
  const Persons = [
    {
      name: "Muhammad",
      age: 23,
      time: "10 years",
    },
    {
      name: "Ahmed",
      age: 22,
      time: "12 years",
    },
    {
      name: "Zahid",
      age: 19,
      time: "13 years",
    },
  ];
  const personList = Persons.map((person) => <Person person={person} />);
  return <div>{personList}</div>;
}
export default NameList;
```

App.js M | NameList.js U | Person.js U

hello-world > src > components > Person.js > Person

```
import React from "react";

function Person({ person }) {
  console.log("Person: ", person.name);
  return (
    <div>
      <h2>
        I am {person.name}. I am {person.age} years old. I know {person.time}
      </h2>
    </div>
  );
}

export default Person;
```

```
Warning: Each child in a list should have a unique "key" prop.

Check the render method of `NameList`. See https://reactjs.org/link/warning-keys for more information.
    at Person (http://localhost:3000/static/js/bundle.js:1060:5)
    at NameList
    at div
    at App
```

We are facing the above issue while working with the List so how we can solve that is listed below:

List and keys:

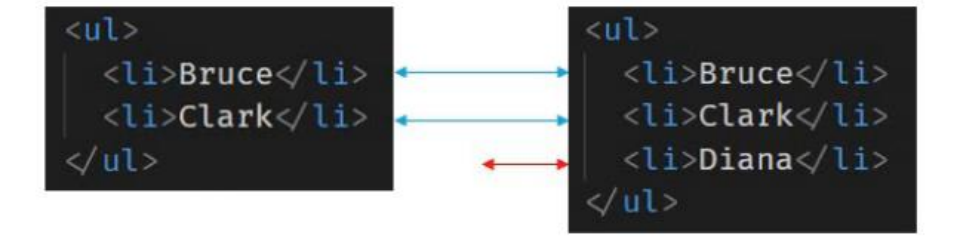
To, remove the above error we need to something like passing a key to the list which is unique.

Now if you see the above error is gone.

Important: Don’t use the key component for rendering the key on the console.

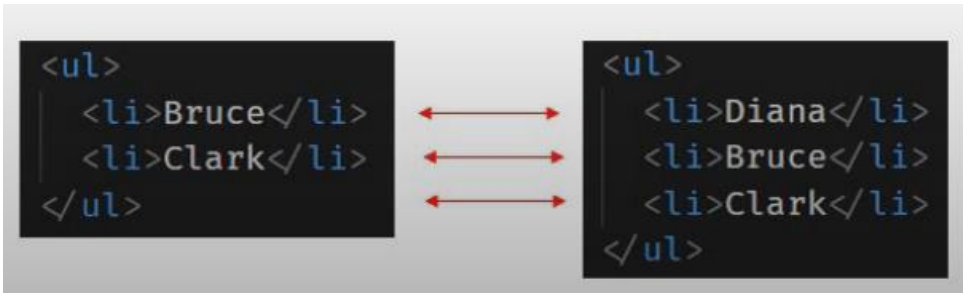
Lists and Keys

List without key attribute



React will check each entry inside the list and if there is a change in the list it will update that item.

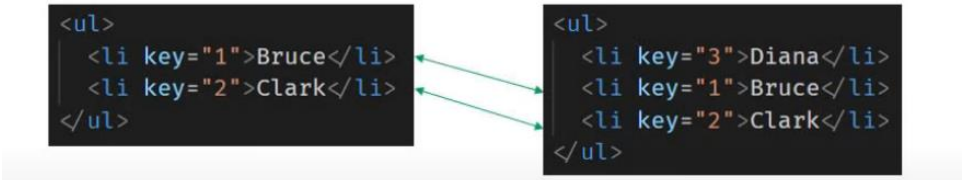
If you seethe below change then it will check the list sequentially and it will change all the list items as it does not come to know which item is to be changed. That is why the react supports the **Key** attribute.



After using the key value

Lists and Keys

List with key attribute



Lists and Keys

A “key” is a special string attribute you need to include when creating lists of elements.

Keys give the elements a stable identity.

Keys help React identify which items have changed, are added, or are removed.

Help in efficient update of the user interface.

Stylesheet in CSS:

Styling React Components

1. CSS stylesheets
2. Inline styling
3. CSS Modules
4. CSS in JS Libaries (Styled Components)

CSS stylesheets

You can use camel case if you want to use the properties inside the element in the UI.

App.js M | StyleSheet.js U | myStyle.css U

hello-world > src > App.js > App

```
import logo from "./logo.svg";
import "./App.css";
import Greet from "./components/Greet";
import Welcome from "./components/Welcome";
import Hello from "./components/Hello";
import Message from "./components/Message";
import Counter from "./components/Counter";
import FunctionClick from "./components/FunctionClick";
import ClassClick from "./components/ClassClick";
import BindEvent from "./components/BindEvent";
import ParentComponent from "./components/ParentComponent";
import UserGreetings from "./components/UserGreetings";
import NameList from "./components/NameList";
import StyleSheet from "./components/StyleSheet";
function App() {
  return (
    <div className="App">
      <StyleSheet />
```

App.js M | StyleSheet.js U | myStyle.css U

hello-world > src > components > StyleSheet.js > ...

```
import React from "react";
import "./myStyle.css";
function styleSheet() {
  return <div className="primary">StyleSheets</div>;
}

export default styleSheet;

```

App.js M | StyleSheet.js U | myStyle.css U

hello-world > src > components > myStyle.css > .primary

```
.primary {
  color: orange;
}

```

And if you want to pass the props into the elements.

App.js M | StyleSheet.js U | myStyle.css U

hello-world > src > App.js > App

```
import logo from "./logo.svg";
import "./App.css";
import Greet from "./components/Greet";
import Welcome from "./components/Welcome";
import Hello from "./components/Hello";
import Message from "./components/Message";
import Counter from "./components/Counter";
import FunctionClick from "./components/FunctionClick";
import ClassClick from "./components/ClassClick";
import BindEvent from "./components/BindEvent";
import ParentComponent from "./components/ParentComponent";
import UserGreetings from "./components/UserGreetings";
import NameList from "./components/NameList";
import StyleSheet from "./components/StyleSheet";
function App() {
  return (
    <div className="App">
      <StyleSheet primary={true} />
```

App.js M | StyleSheet.js U | myStyle.css U

hello-world > src > components > StyleSheet.js > styleSheet

```
import React from "react";
import "./myStyle.css";
function styleSheet(props) {
  let className = props.primary ? "primary" : "";
  return <div className={className}>StyleSheets</div>;
}

export default styleSheet;
```

And let say you want to add one more property than what you can do is below

App.js M | StyleSheet.js U | myStyle.css U

hello-world > src > components > myStyle.css > .font-xl

```
.primary {
  color: orange;
}
.font-xl {
  font-size: 72px;
}
```

App.js M | StyleSheet.js U | myStyle.css U

hello-world > src > components > StyleSheet.js > styleSheet

```
import React from "react";
import "./myStyle.css";
function styleSheet(props) {
  let className = props.primary ? "primary" : "";
  return <div className={`${className} font-xl `}>StyleSheets</div>;
}

export default styleSheet;

```

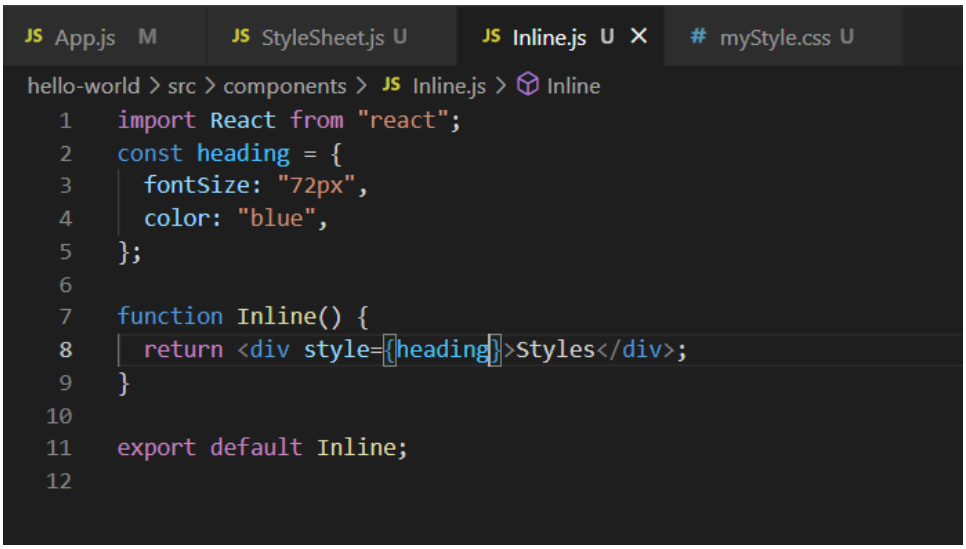
Inline styling

If you want to add the inline styling then what you can do is listed below

App.js M | StyleSheet.js U | Inline.js U | myStyle.css U

hello-world > src > App.js > App

```
import logo from "./logo.svg";
import "./App.css";
import Greet from "./components/Greet";
import Welcome from "./components/Welcome";
import Hello from "./components/Hello";
import Message from "./components/Message";
import Counter from "./components/Counter";
import FunctionClick from "./components/FunctionClick";
import ClassClick from "./components/ClassClick";
import BindEvent from "./components/BindEvent";
import ParentComponent from "./components/ParentComponent";
import UserGreetings from "./components/UserGreetings";
import NameList from "./components/NameList";
import StyleSheet from "./components/StyleSheet";
import Inline from "./components/Inline";
function App() {
  return (
    <div className="App">
      <Inline />
```



Remember one thing which is you can create properties which are camel case.

CSS Module

In this case, we can avoid by having a conflict in the class because we are accessing the variable using the class reference. On the other hand, when we using the class to the simple CSS file then we may be the conflict of naming convention.

App.js M | StyleSheet.js U | Inline.js U | appStyles.css U

hello-world > src > App.js > App

```
import logo from "./logo.svg";
import "./App.css";
import Greet from "./components/Greet";
import Welcome from "./components/Welcome";
import Hello from "./components/Hello";
import Message from "./components/Message";
import Counter from "./components/Counter";
import FunctionClick from "./components/FunctionClick";
import ClassClick from "./components/ClassClick";
import BindEvent from "./components/BindEvent";
import ParentComponent from "./components/ParentComponent";
import UserGreetings from "./components/UserGreetings";
import NameList from "./components/NameList";
import StyleSheet from "./components/StyleSheet";
// import Inline from "./components/Inline";
import "./components/appStyles.css";
import styles from "./components/appStyle.module.css";
function App() {
  return (
    <div className="App">
      <h1 className="error">Error</h1>
      <h1 className={styles.success}>Success</h1>
```

App.js M | appStyles.css U | appStyle.module.css U

hello-world > src > components > appStyles.css > .error

```
.error {
  color: red;
}

```

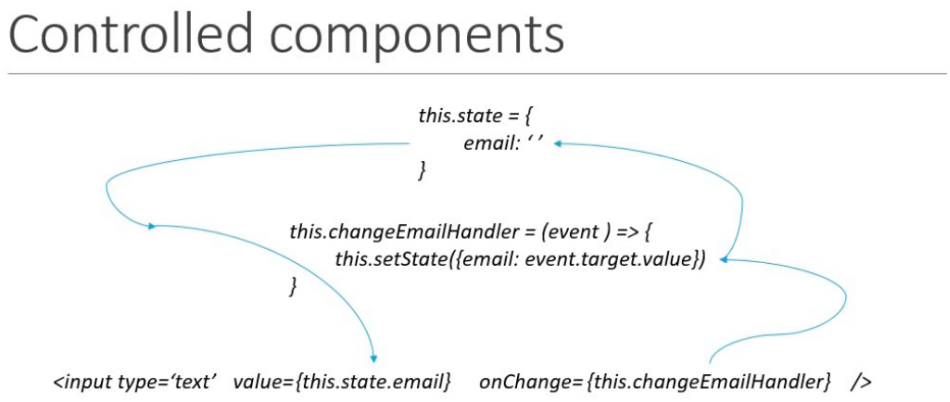
App.js M | appStyles.css U | appStyle.module.css U

hello-world > src > components > appStyle.module.css > .success

```
.success {
  color: green;
}

```

Basics of Form Handling

If the value of the form is controlled by react is known as controlled component.

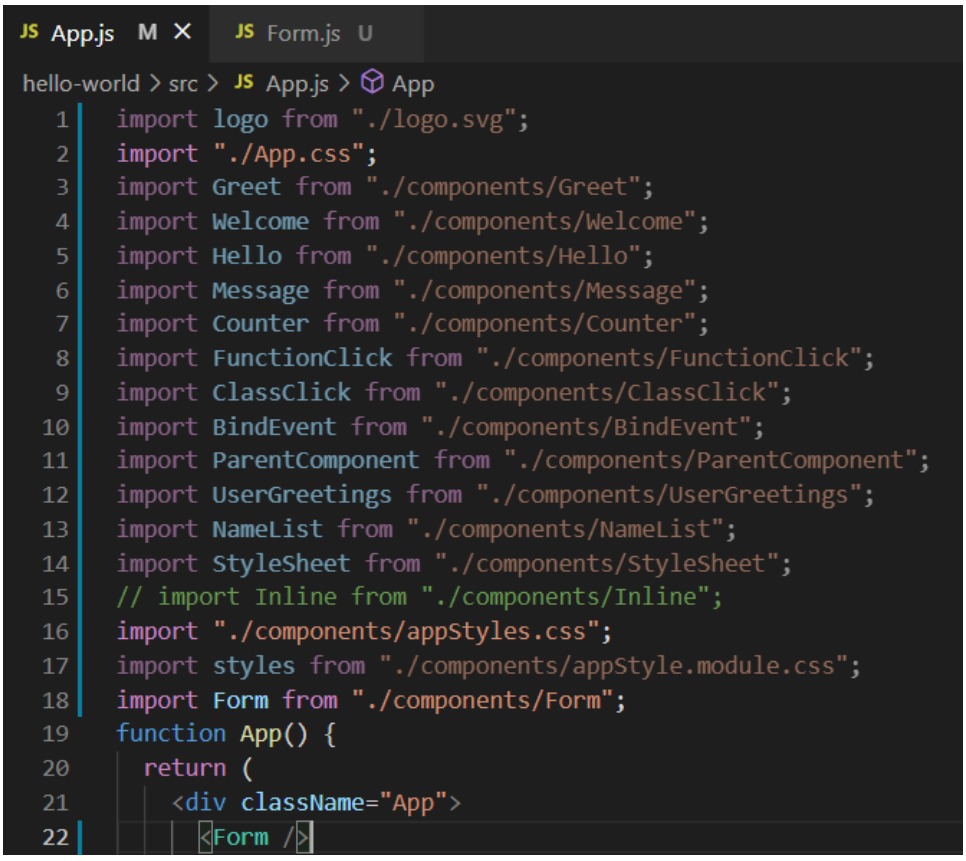


There are two steps there:

1. Setting the state
2. On change where we provide the arrow function

Normally, the submit button is to handle inside the JavaScript.

You can also create the same functionality with the button with that function attached to it the benefit of using the **submit** button is to add the enter functionality.


App.js M
Form.js U
hello-world > src > App.js > App
1 import logo from "./logo.svg";
2 import "./App.css";
3 import Greet from "./components/Greet";
4 import Welcome from "./components/Welcome";
5 import Hello from "./components/Hello";
6 import Message from "./components/Message";
7 import Counter from "./components/Counter";
8 import FunctionClick from "./components/FunctionClick";
9 import ClassClick from "./components/ClassClick";
10 import BindEvent from "./components/BindEvent";
11 import ParentComponent from "./components/ParentComponent";
12 import UserGreetings from "./components/UserGreetings";
13 import NameList from "./components/NameList";
14 import StyleSheet from "./components/StyleSheet";
15 // import Inline from "./components/Inline";
16 import "./components/appStyles.css";
17 import styles from "./components/appStyle.module.css";
18 import Form from "./components/Form";
19 function App() {
20 return (
21
22 <Form />

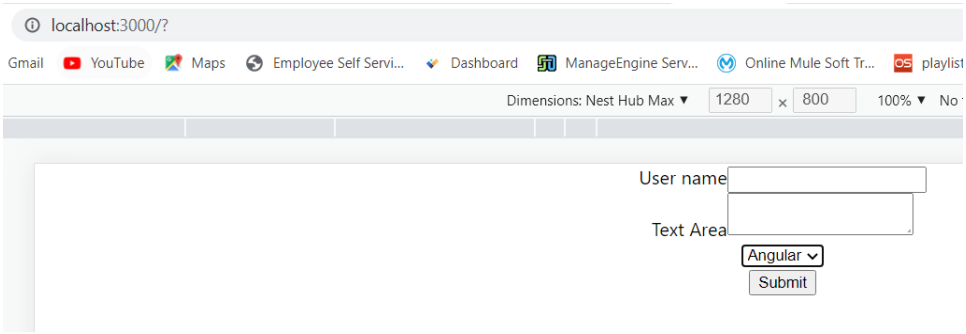
So, if you can see we are setting the two functionalities one is to set the state and the other is the function which handle the values inside the field.

App.js M Form.js U

hello-world > src > components > Form.js > Form > render

```
    this.state = {
      username: "",
      comment: "",
    };
  }
  usernameHandler = (event) => {
    this.setState({ username: event.target.value });
  };
  commentHandler = (event) => {
    this.setState({ comment: event.target.value });
  };

  render() {
    return (
      <form>
        <div>
          <div>
            <label>User name</label>
            <input
              type="text"
              value={this.state.username}
              onChange={this.usernameHandler}
            ></input>
          </div>
          <div>
            <label>Text Area</label>
            <textarea type="text" onChange={this.commentHandler}></textarea>
          </div>
        </div>
      </form>
    );
  }
}
```

Now let's handle the submit button.



App.js M Form.js U

hello-world > src > components > Form.js > Form > render

```
15    commentHandler = (event) => {
16      this.setState({ comment: event.target.value });
17    };
18    topicHandler = (event) => {
19      this.setState({ topic: event.target.value });
20    };
21
22    render() {
23      return (
24        <form>
25          <div>
26            <div>
27              <label>User name</label>
28              <input
29                type="text"
30                value={this.state.username}
31                onChange={this.usernameHandler}
32              ></input>
33            </div>
34            <div>
35              <label>Text Area</label>
36              <textarea type="text" onChange={this.commentHandler}></textarea>
37            </div>
38            <div>
39              <select value={this.state.topic} onChange={this.topicHandler}>
40                <option>React</option>
41                <option>Angular</option>
42                <option>Vue</option>
43              </select>
44            </div>
45          </div>
46
47          <button type="submit">Submit</button>
48        </form>
```

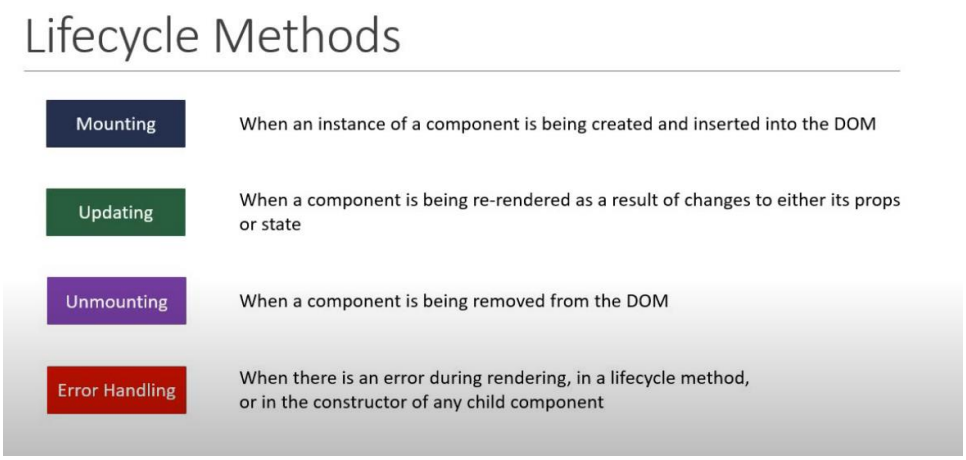
Now if you want to prevent that on submit button your page did not refresh then you can stop by using prevent Default.

```
  this.setState({ topic: event.target.value });
};
formHandler = (event) => {
  alert(`${this.state.username} ${this.state.comment} ${this.state.topic}`);
};

render() {
  return (
    <form onSubmit={this.formHandler}>
      <div>
        <div>
```

```
formHandler = (event) => {
  alert(`${this.state.username} ${this.state.comment} ${this.state.topic}`);
  event.preventDefault();
};

render() {
  return (
    <form onSubmit={this.formHandler}>
      <div>
```

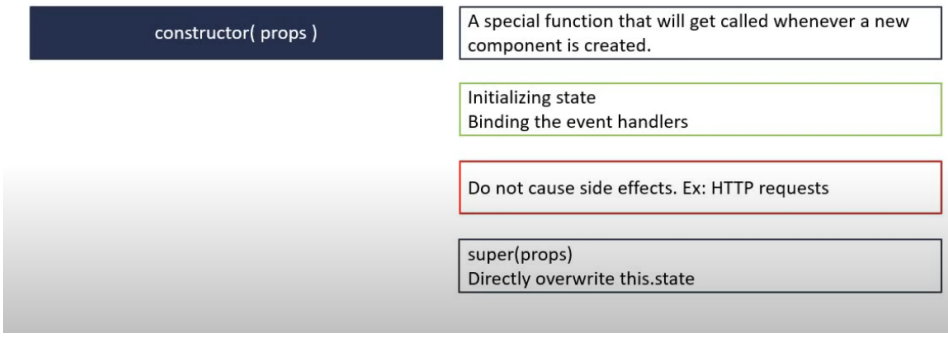
Component Lifecycle Methods

It works with the class-based components but in the proposal of functional based components we can say that it works with that also.

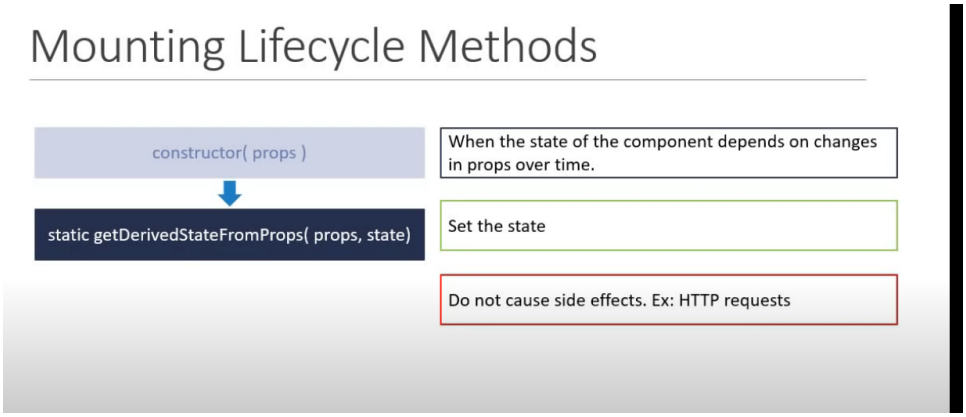


Component Mounting Lifecycle Methods

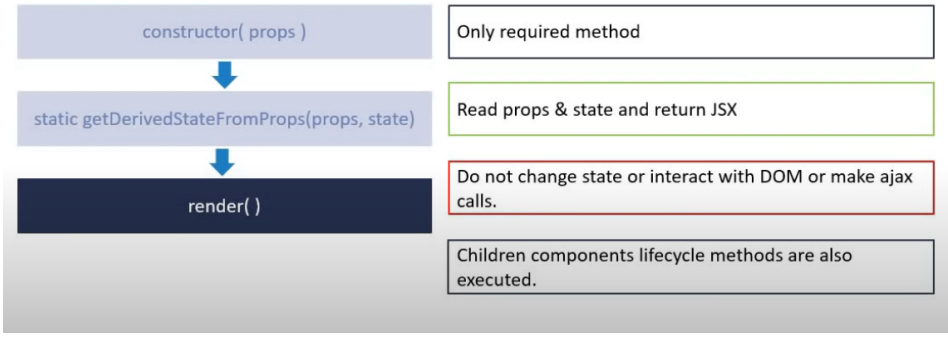
Mounting Lifecycle Methods



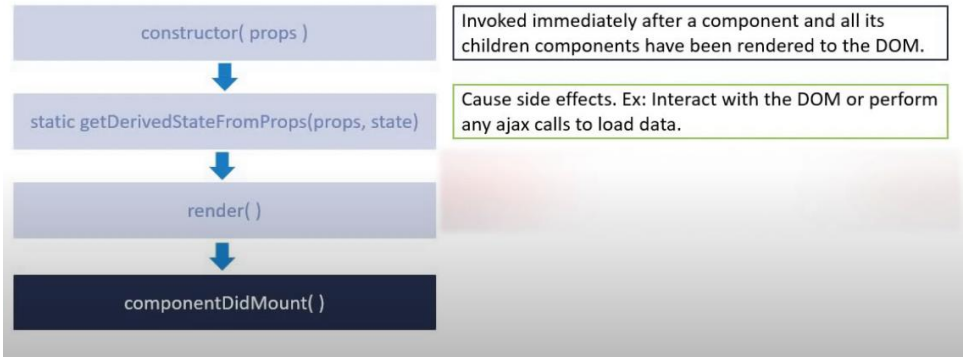
Mounting Lifecycle Methods



Mounting Lifecycle Methods

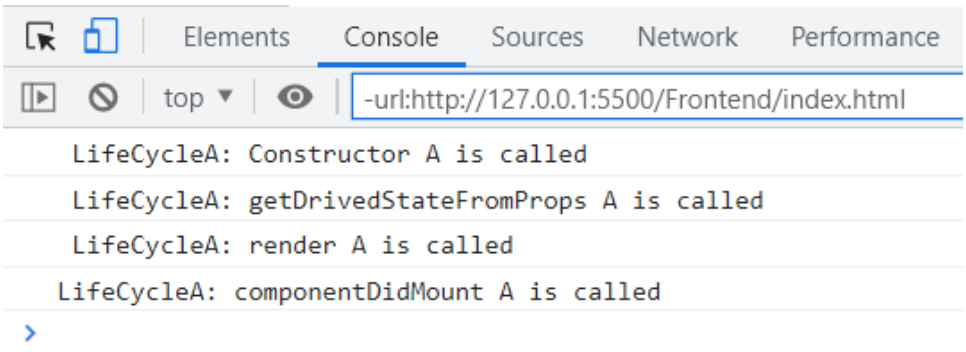


Mounting Lifecycle Methods

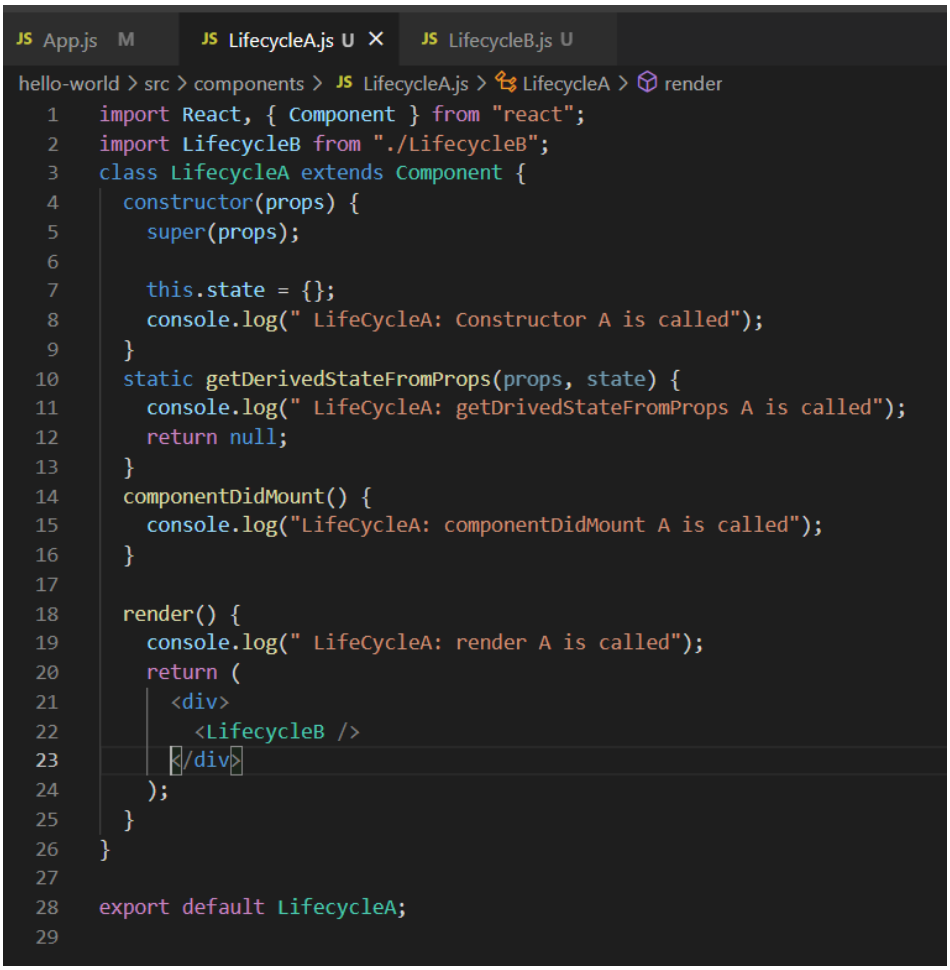


App.js M
LifecycleA.js U
hello-world > src > components > LifecycleA.js > LifecycleA > getDerivedStateFromProps

```
1   import React, { Component } from "react";
2
3   class LifecycleA extends Component {
4     constructor(props) {
5       super(props);
6
7       this.state = {};
8       console.log(" LifeCycleA: Constructor A is called");
9     }
10    static getDerivedStateFromProps(props, state) {
11      console.log(" LifeCycleA: getDrivedStateFromProps A is called");
12      return null;
13    }
14    componentDidMount() {
15      console.log("LifeCycleA: componentDidMount A is called");
16    }
17
18    render() {
19      console.log(" LifeCycleA: render A is called");
20      return <div>Hello</div>;
21    }
22  }
23
24  export default LifecycleA;
25
```



If you created one parent class and one child class so what is the execution of the flow

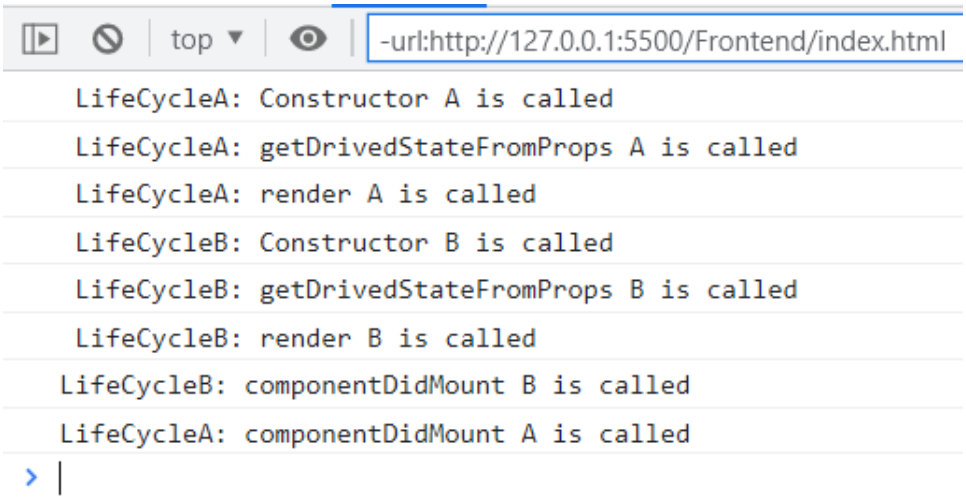


App.js M | LifecycleA.js U | LifecycleB.js U

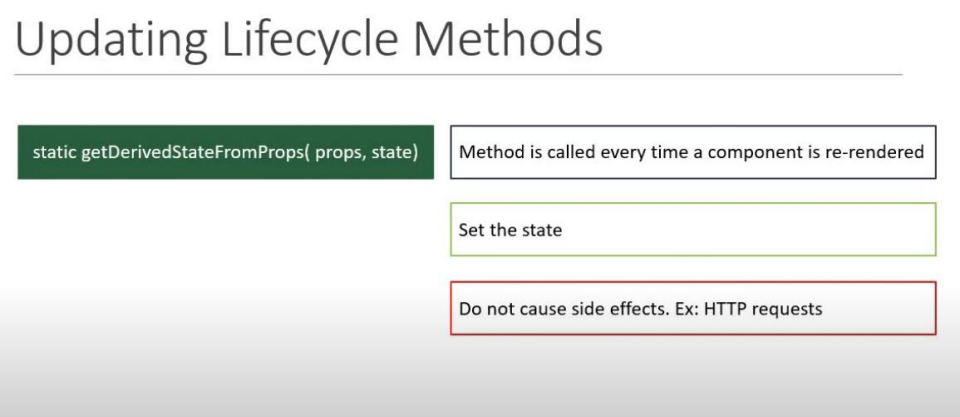
hello-world > src > components > LifecycleB.js > LifecycleB > constructor

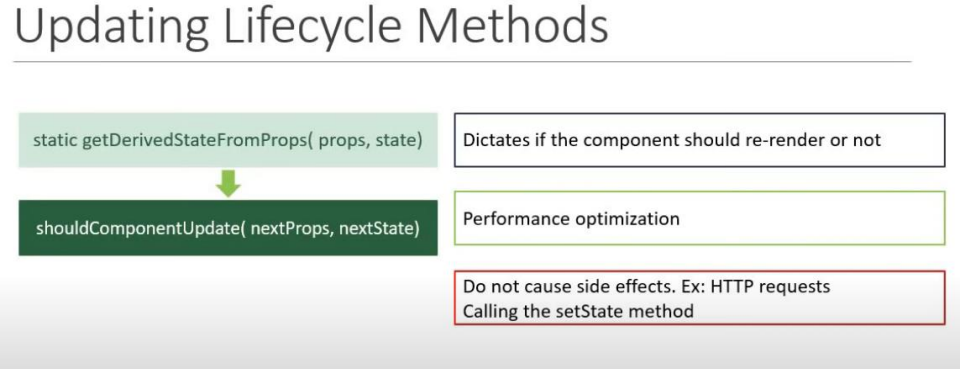
```
1   import React, { Component } from "react";
2
3   class LifecycleB extends Component {
4     constructor(props) {
5       super(props);
6
7       this.state = {};
8       console.log(" LifeCycleB: Constructor B is called");
9     }
10    static getDerivedStateFromProps(props, state) {
11      console.log(" LifeCycleB: getDrivedStateFromProps B is called");
12      return null;
13    }
14    componentDidMount() {
15      console.log("LifeCycleB: componentDidMount B is called");
16    }
17
18    render() {
19      console.log(" LifeCycleB: render B is called");
20      return <div>Hello</div>;
21    }
22  }
23
24  export default LifecycleB;
25
```

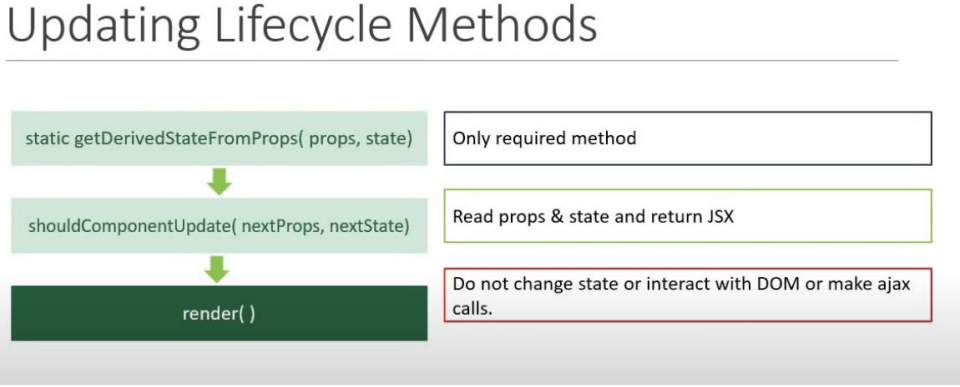
If you see when the render A is executing then it is calling the child process and the further execution is going. Moreover, the exact result is listed below.



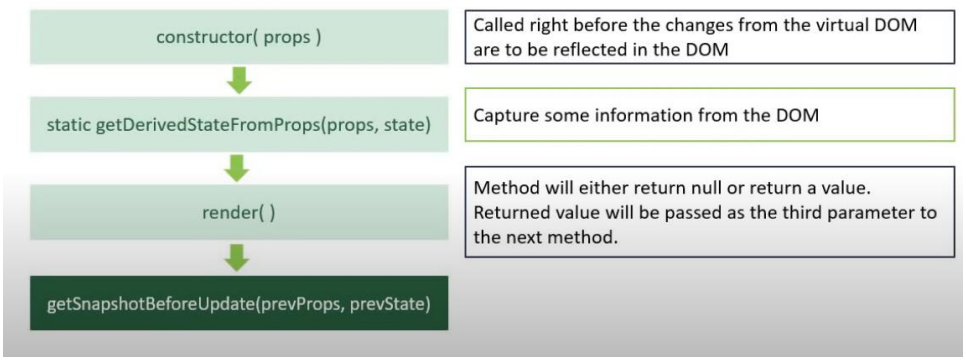
Component Updating Lifecycle Methods



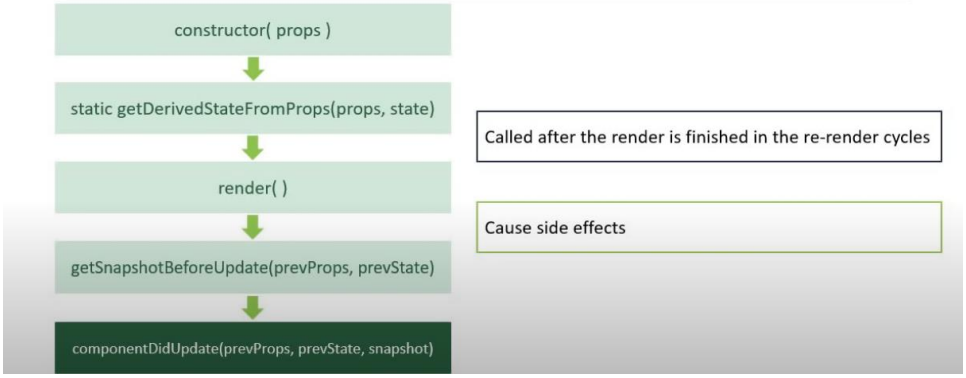




Updating Lifecycle Methods



Updating Lifecycle Methods




App.js M
LifecycleA.js U
LifecycleB.js U
hello-world > src > components > LifecycleA.js > LifecycleA > getSnapshotBeforeUpdate
constructor(props) {
super(props);
this.state = {};
console.log(" LifeCycleA: Constructor A is called");
}
static getDerivedStateFromProps(props, state) {
console.log(" LifeCycleA: getDrivedStateFromProps A is called");
return null;
}
componentDidMount() {
console.log("LifeCycleA: componentDidMount A is called");
}
shouldComponentUpdate() {
console.log("LifeCycleA: shouldComponentUpdate");
return true;
}
getSnapshotBeforeUpdate(PrevProps, prevState) {
console.log("LifeCycleA: getSnapshotBeforeUpdate");
return null;
}
componentDidUpdate() {
console.log("LifeCycleA: componentDidUpdate");
}
changeState = () => {
this.setState({ name: "Code" });
};
render() {
console.log(" LifeCycleA: render A is called");
return (

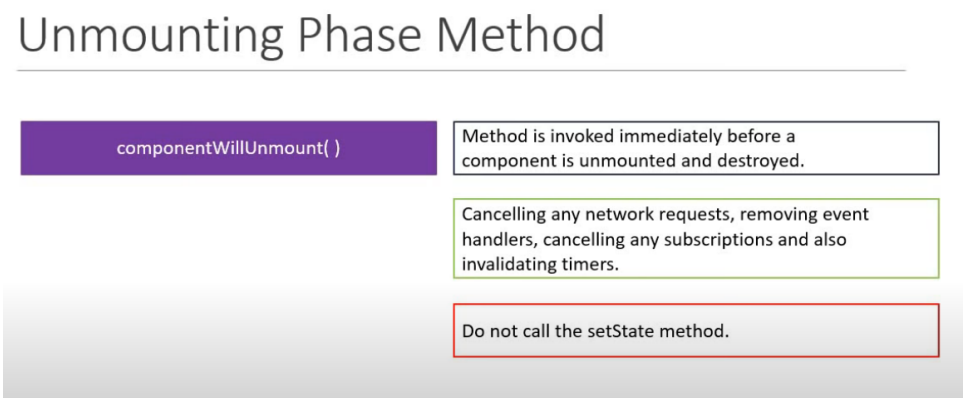
<button onClick={this.changeState}>Change state</button>
<LifecycleB />

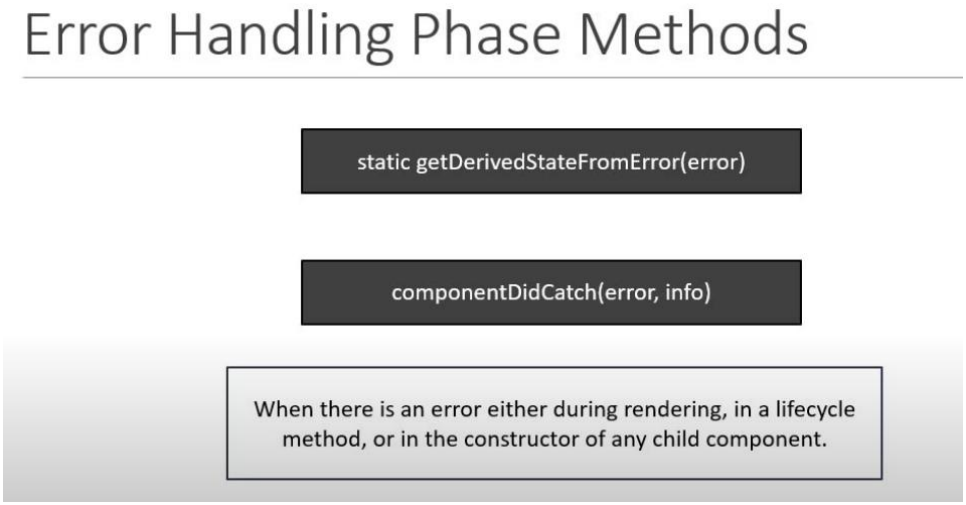
App.js M | LifecycleA.js U | LifecycleB.js U

hello-world > src > components > LifecycleB.js > LifecycleB > getSnapshotBeforeUpdate

```
1  import React, { Component } from "react";
2
3  class LifecycleB extends Component {
4    constructor(props) {
5      super(props);
6
7      this.state = {};
8      console.log(" LifeCycleB: Constructor B is called");
9    }
10   static getDerivedStateFromProps(props, state) {
11     console.log(" LifeCycleB: getDrivedStateFromProps B is called");
12     return null;
13   }
14   componentDidMount() {
15     console.log("LifeCycleB: componentDidMount B is called");
16   }
17   shouldComponentUpdate() {
18     console.log("LifeCycleB: shouldComponentUpdate");
19     return true;
20   }
21   getSnapshotBeforeUpdate(PrevProps, prevState) {
22     console.log("LifeCycleB: getSnapshotBeforeUpdate");
23     return null;
24   }
25   componentDidUpdate() {
26     console.log("LifeCycleB: componentDidUpdate");
27   }
28
29   render() {
30     console.log(" LifeCycleB: render B is called");
31     return <div>Hello</div>;
32   }
33 }
```

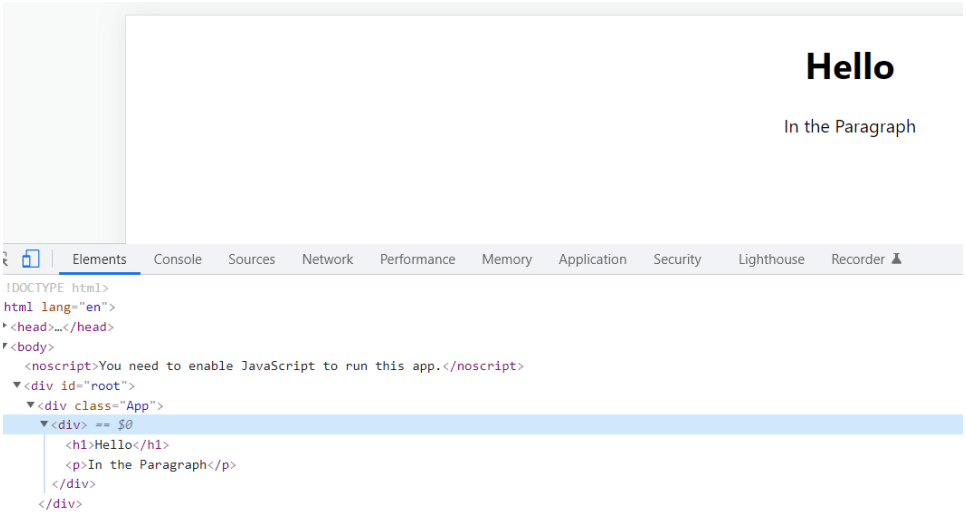
```
LifeCycleA: getDrivedStateFromProps A is called
LifeCycleA: shouldComponentUpdate
LifeCycleA: render A is called
LifeCycleB: getDrivedStateFromProps B is called
LifeCycleB: shouldComponentUpdate
LifeCycleB: render B is called
LifeCycleB: getSnapshotBeforeUpdate
LifeCycleA: getSnapshotBeforeUpdate
LifeCycleB: componentDidUpdate
LifeCycleA: componentDidUpdate
>
```

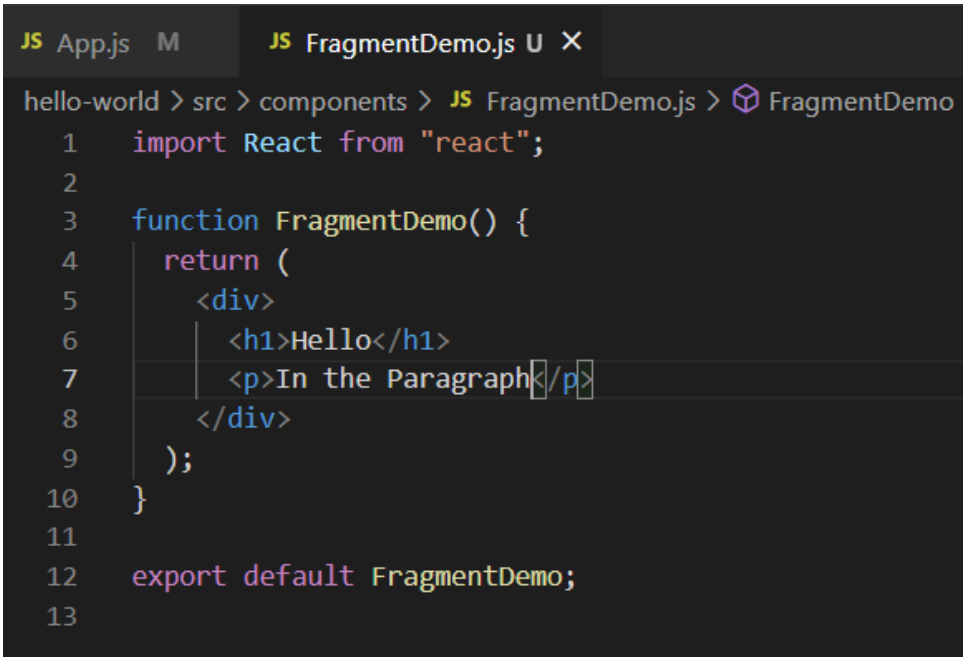




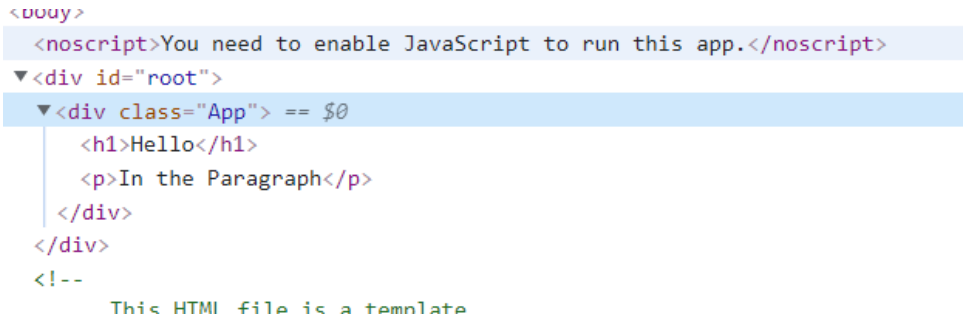
Fragments

In the render method you have only one **div** present to the parent node and inside that node you can create other nodes and all the nodes is enclosed to one and when you write **React.Fragment** then create the individual component.





But what if we want to do render the components individual instead of a **div** then we can use Fragment.

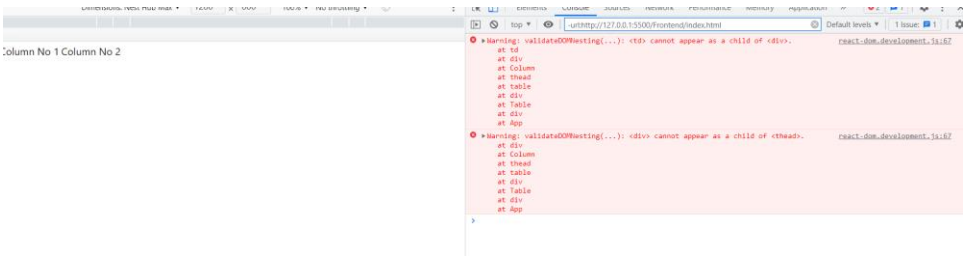


App.js M FragmentDemo.js U

hello-world > src > components > FragmentDemo.js > FragmentDemo

```
1   import React from "react";
2
3   function FragmentDemo() {
4     return (
5       <React.Fragment>
6         <h1>Hello</h1>
7         <p>In the Paragraph</p>
8       </React.Fragment>
9     );
10  }
11
12  export default FragmentDemo;
13
```

You can also replace the React following the Fragment part with the <> but the one limitation with that you cannot pass the key variable because it is used in case of map function.



App.js M | Table.js U | Column.js U

hello-world > src > components > Table.js > ...

```
1   import React from "react";
2   import Column from "./Column";
3   function Table() {
4     return (
5       <div>
6         <table>
7           <thead>
8             <Column />
9           </thead>
10          <tbody></tbody>
11        </table>
12      </div>
13    );
14  }
15
16  export default Table;
17
```

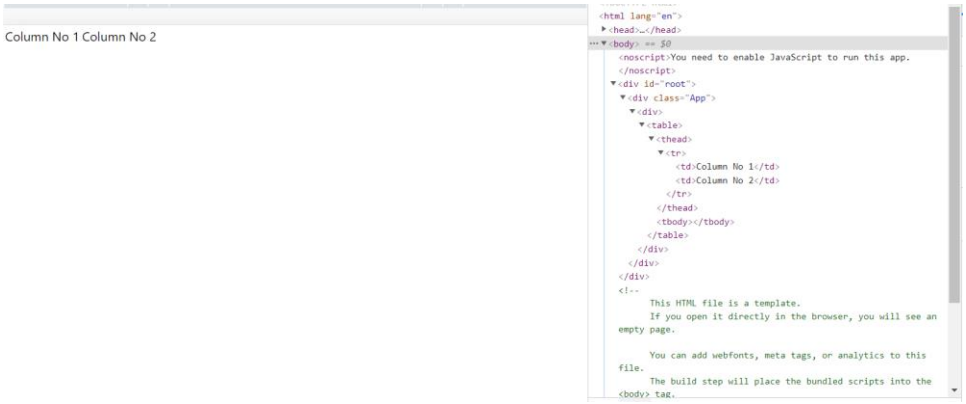
App.js M | Table.js U | Column.js U

hello-world > src > components > Column.js > Column

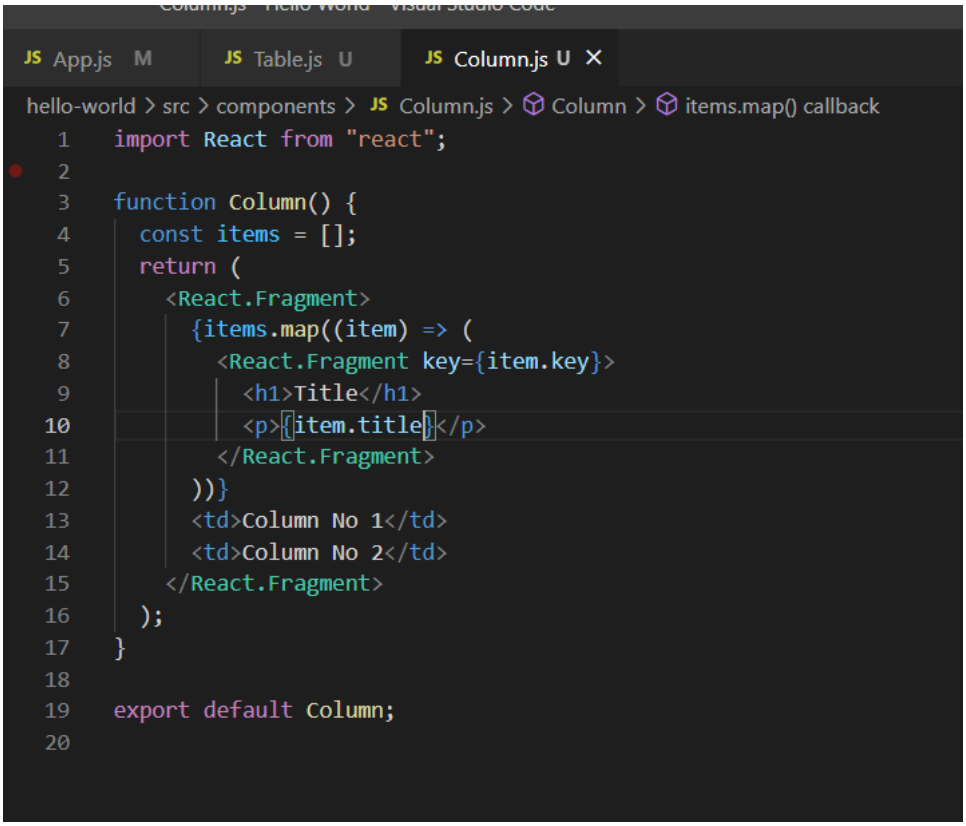
```
1   import React from "react";
2
3   function Column() {
4     return (
5       <div>
6         <td>Column No 1</td>
7         <td>Column No 2</td>
8       </div>
9     );
10  }
11
12  export default Column;
13
```

Because there is no div element if the node is table head, then it must be inside td or tr

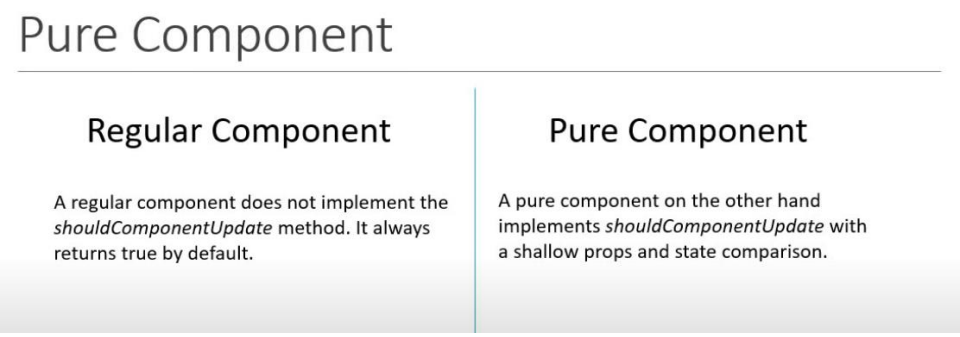
We can solve this using the React and Fragment.



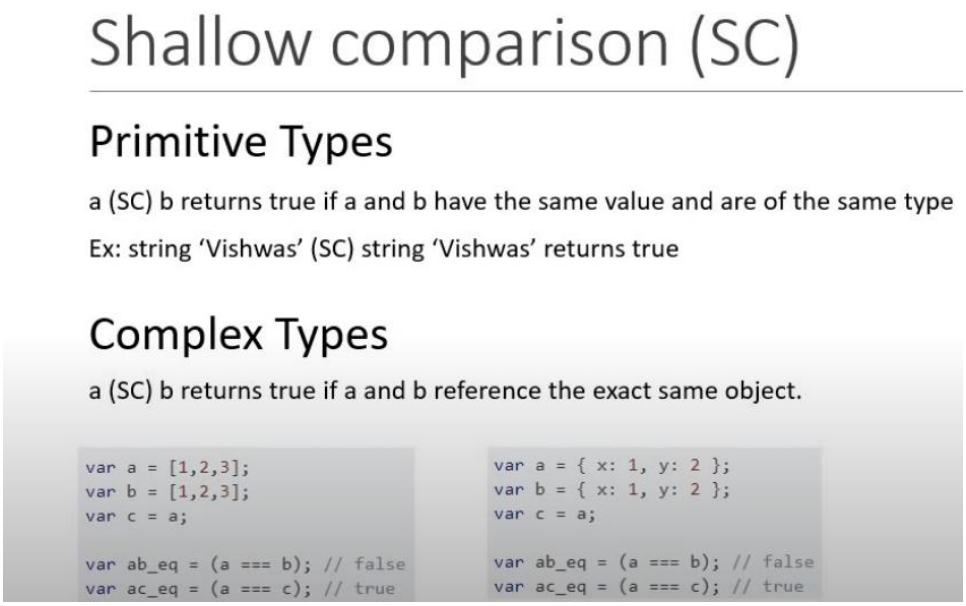
If you are working with the list then you can do the below code



Pure Components



There is no apparent change in regular and pure component. One difference is that for pure component if the value does not change, from the older than it will not re-render the method while in the class method its re-render.



If the parent does not re-render, then the children component also not re-render.

Always return object when working with the pure component.

Summary

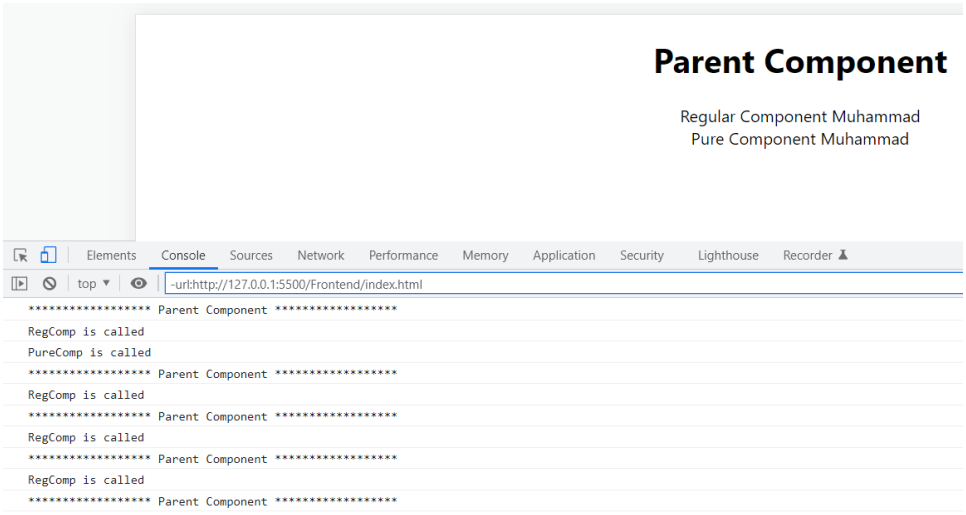
We can create a component by extending the PureComponent class.

A PureComponent implements the *shouldComponentUpdate* lifecycle method by performing a shallow comparison on the props and state of the component.

If there is no difference, the component is not re-rendered – performance boost.

It is a good idea to ensure that all the children components are also pure to avoid unexpected behaviour.

Never mutate the state. Always return a new object that reflects the new state.



App.js M | PureComp.js U | RegComp.js U | ParentComp.js U

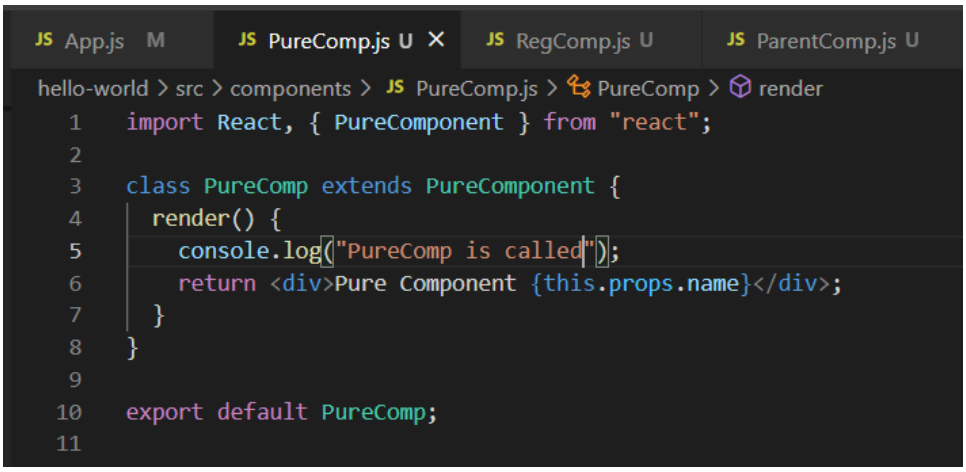
hello-world > src > components > ParentComp.js > ParentComp > render

```
1  import React, { Component, PureComponent } from "react";
2  import PureComp from "./PureComp";
3  import RegComp from "./RegComp";
4
5  class ParentComp extends Component {
6    constructor(props) {
7      super(props);
8
9      this.state = {
10       name: "Muhammad",
11     };
12   }
13   componentDidMount() {
14     setInterval(() => {
15       this.setState({ name: "Muhammad" });
16     }, 2000);
17   }
18   render() {
19     console.log("****************** Parent Component ******************");
20     return (
21       <div>
22         <h1>Parent Component</h1>
23         <RegComp name={this.state.name} />
24         <PureComp name={this.state.name} />
25       </div>
26     );
27   }
28 }
29
30 export default ParentComp;
31
```

App.js M | PureComp.js U | RegComp.js U | ParentComp.js U

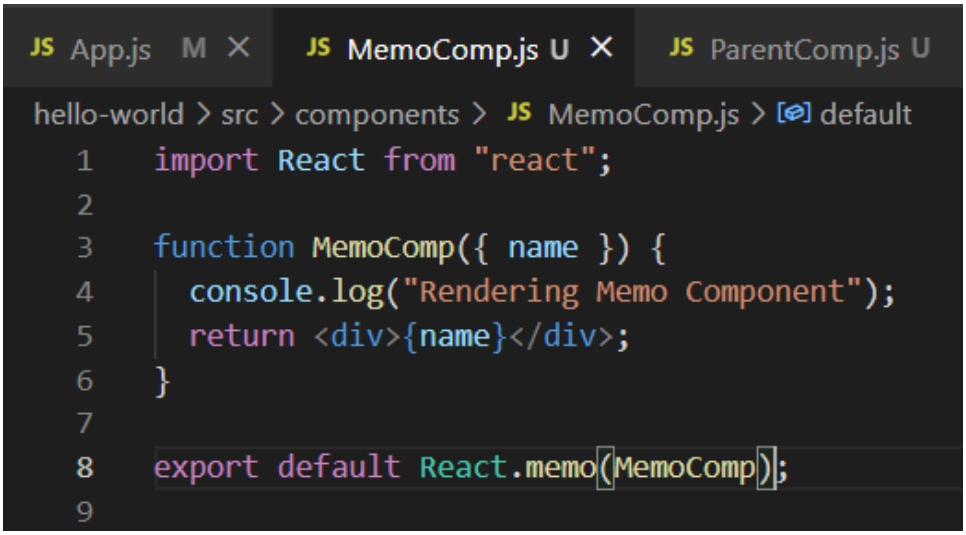
hello-world > src > components > RegComp.js > RegComp > render

```
1  import React, { Component } from "react";
2
3  class RegComp extends Component {
4    render() {
5      console.log("RegComp is called");
6      return <div>Regular Component {this.props.name}</div>;
7    }
8  }
9
10 export default RegComp;
11
```



memo

If we want to achieve same functionality with the function-based component then we need to use memo. React



App.js M | MemoComp.js U | ParentComp.js U

hello-world > src > components > ParentComp.js > ParentComp > render

```
import React, { Component, PureComponent } from "react";
import MemoComp from "./MemoComp";
import PureComp from "./PureComp";
import RegComp from "./RegComp";

class ParentComp extends Component {
  constructor(props) {
    super(props);

    this.state = {
      name: "Muhammad",
    };
  }
  componentDidMount() {
    setInterval(() => {
      this.setState({ name: "Muhammad" });
    }, 2000);
  }
  render() {
    console.log("***************** Parent Component *****************");
    return (
      <div>
        <h1>Parent Component</h1>
        {/* <RegComp name={this.state.name} /> */}
        {/* <PureComp name={this.state.name} /> */}
        <MemoComp name={this.state.name} />
      </div>
    );
  }
}

export default ParentComp;

```

Refs

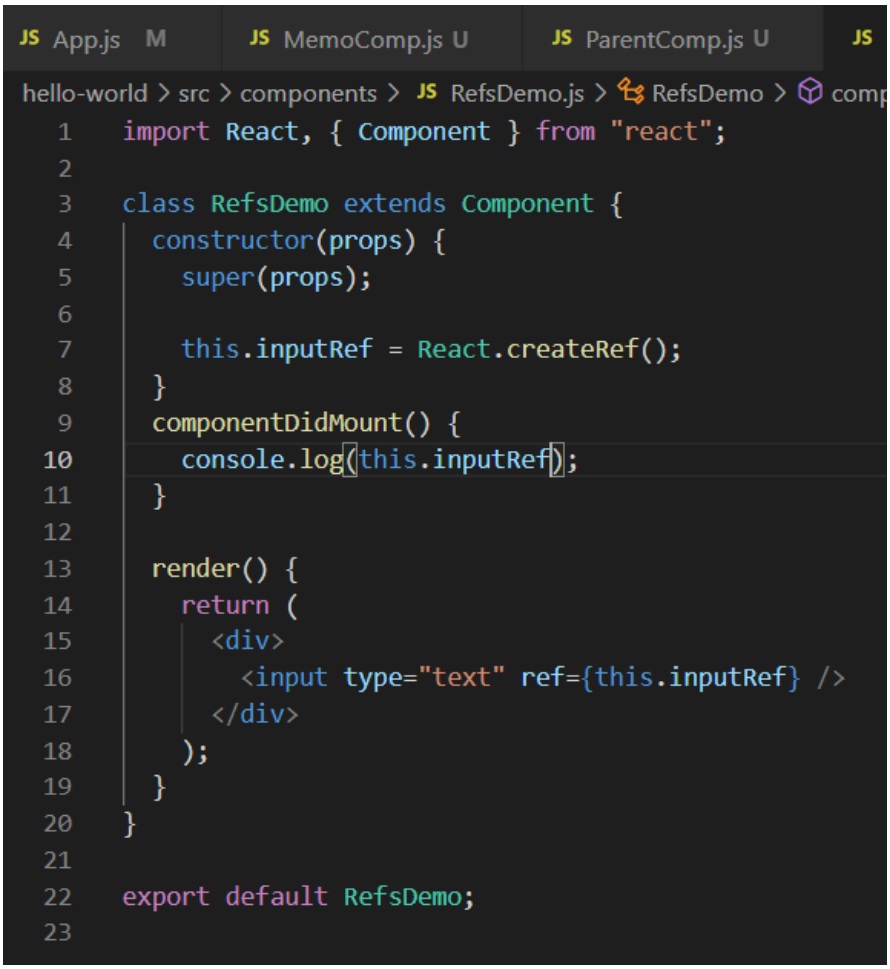
Accessing DOM nodes directly in react.

You can focus on input node.

You can fetch data from input user.

One way is to use the create ref method.

Other way is to use the call back refs


App.js M
MemoComp.js U
ParentComp.js U
hello-world > src > components > RefsDemo.js > RefsDemo
import React, { Component } from "react";
class RefsDemo extends Component {
constructor(props) {
super(props);
this.inputRef = React.createRef();
}
componentDidMount() {
console.log(this.inputRef);
}
render() {
return (

<input type="text" ref={this.inputRef} />

);
}
}
export default RefsDemo;


App.js M
MemoComp.js U
ParentComp.js U
hello-world > src > components > RefsDemo.js > RefsDemo
import React, { Component } from "react";
class RefsDemo extends Component {
constructor(props) {
super(props);
this.inputRef = React.createRef();
}
componentDidMount() {
this.inputRef.current.focus();
console.log(this.inputRef);
}
render() {
return (

<input type="text" ref={this.inputRef} />

);
}
}
export default RefsDemo;

2nd Approach

Refs with Class Components

If you want to do something like when you click on the button and the focus goes to the input field and you cannot use the component did mount.

App.js M | MemoComp.js U | ParentComp.js U | RefsDemo.js U | Input.js U

hello-world > src > components > Input.js > Input > focusInput

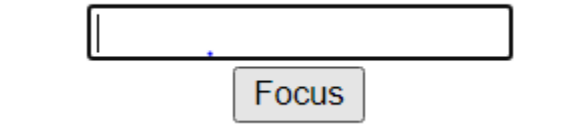
```
1   import React, { Component } from "react";
2
3   class Input extends Component {
4     constructor(props) {
5       super(props);
6
7       this.inputRef = React.createRef();
8     }
9     focusInput() {
10      this.inputRef.current.focus();
11    }
12    render() {
13      return (
14        <div>
15          <input type="text" ref={this.inputRef}></input>
16        </div>
17      );
18    }
19  }
20
21  export default Input;
22
```

App.js M | MemoComp.js U | ParentComp.js U | RefsDemo.js U | Input.js U | FocusInput.js U

hello-world > src > components > FocusInput.js > FocusInput > clickHandler

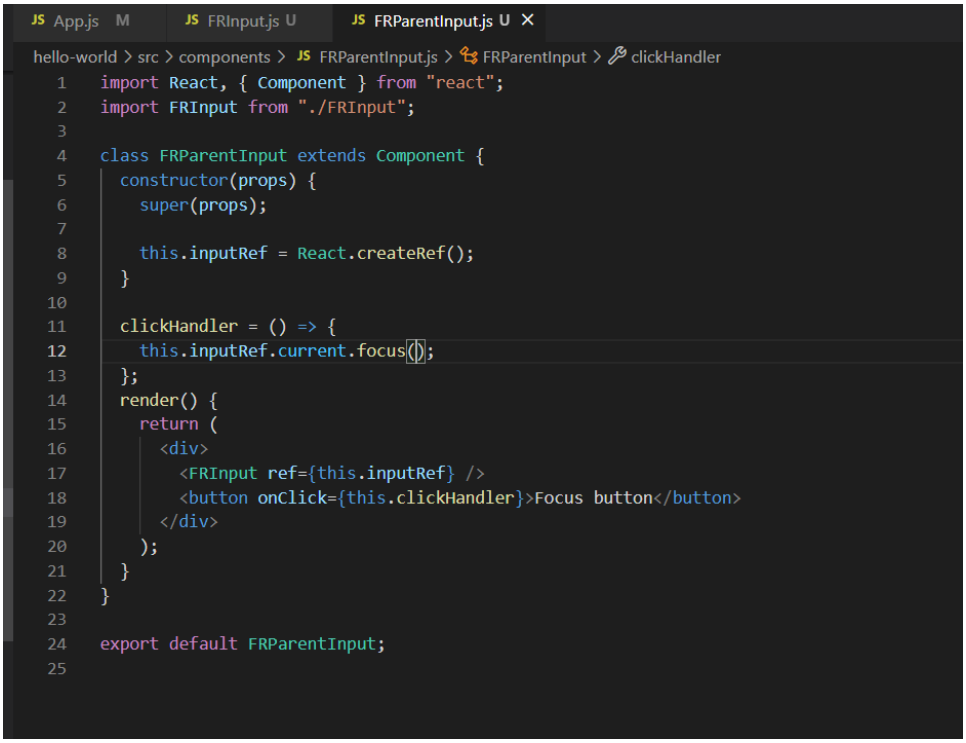
```
1   import React, { Component } from "react";
2   import Input from "./Input";
3
4   class FocusInput extends Component {
5     constructor(props) {
6       super(props);
7
8       this.componentRef = React.createRef();
9     }
10    clickHandler = () => {
11      this.componentRef.current.focusInput();
12    };
13    render() {
14      return (
15        <div>
16          <Input ref={this.componentRef} />
17          <button onClick={this.clickHandler}>Focus</button>
18        </div>
19      );
20    }
21  }
22
23  export default FocusInput;
24
```

```
25  function App() {
26    return (
27      <div className="App">
28        <FocusInput />
```



Forwarding Refs

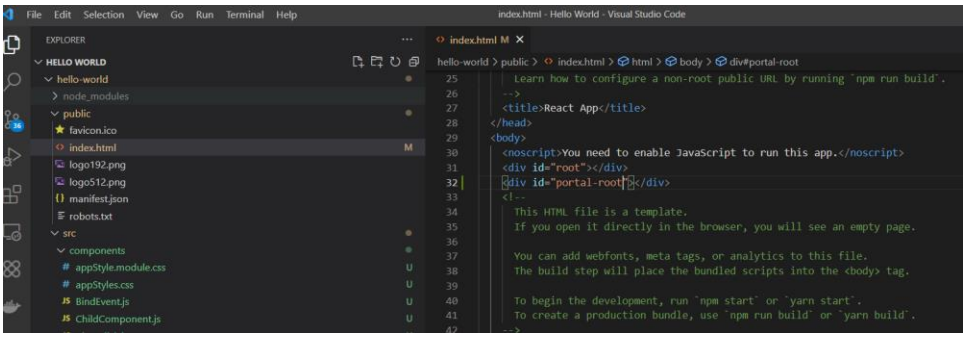
It is a process by which you pass your refs from parent component to its child component.



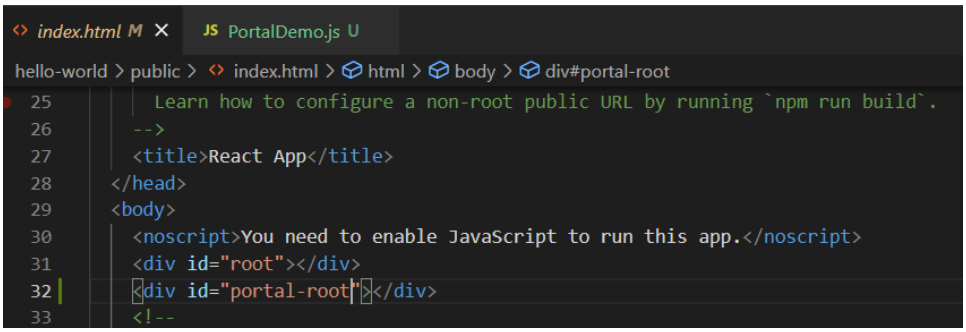
Portals

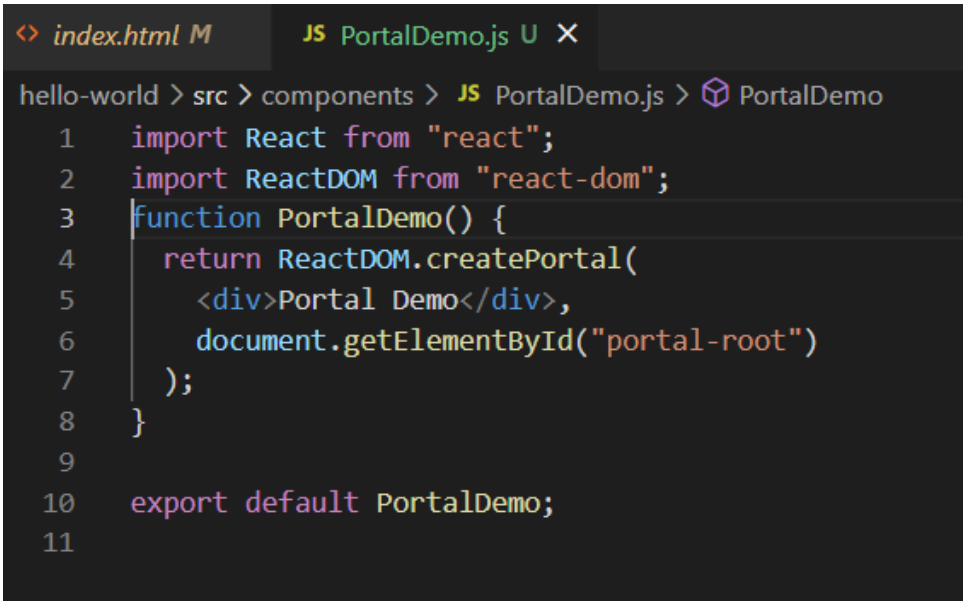
You can create elements which are not under the **root** element and render those elements.

Portals have all the properties of the react component similarly. Below is the code how you can do that:



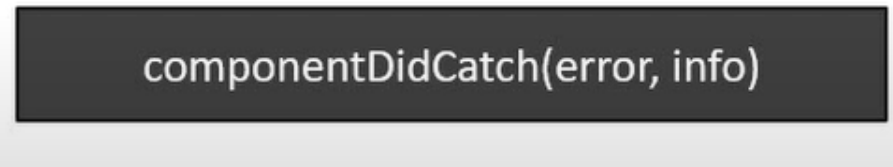
First you need to create the element because it needs a react element where it needs to render and the element which it really wants to render.

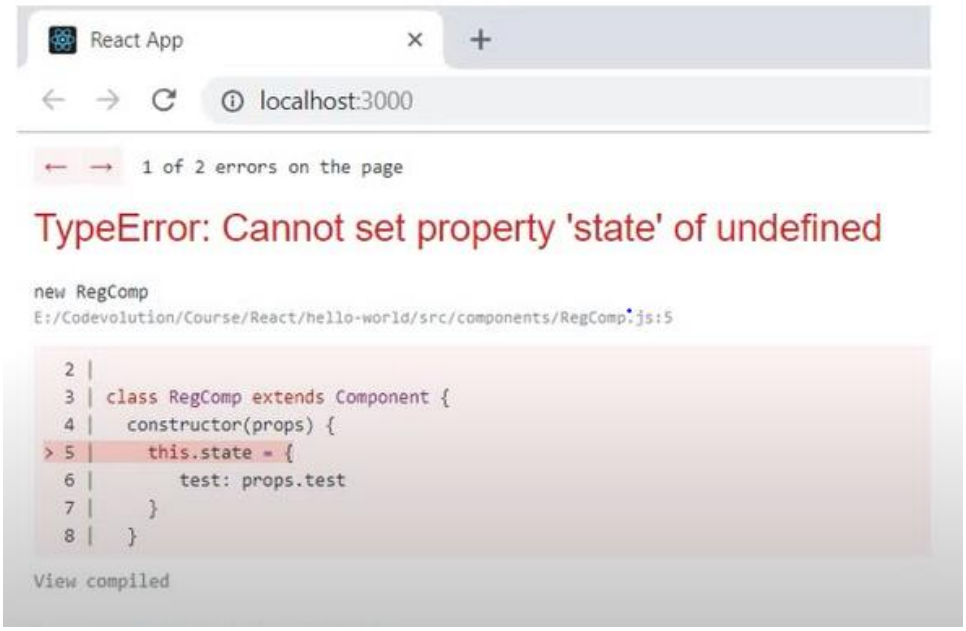




This is beneficial when you want to render outside the app node.

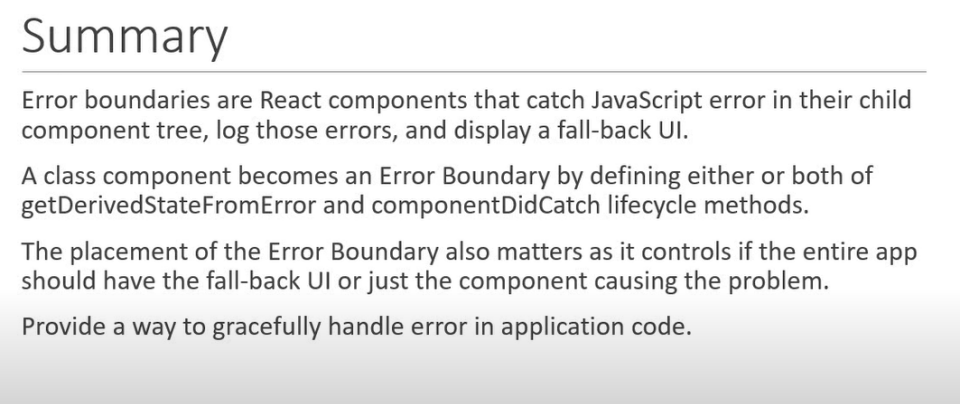
Error Boundary



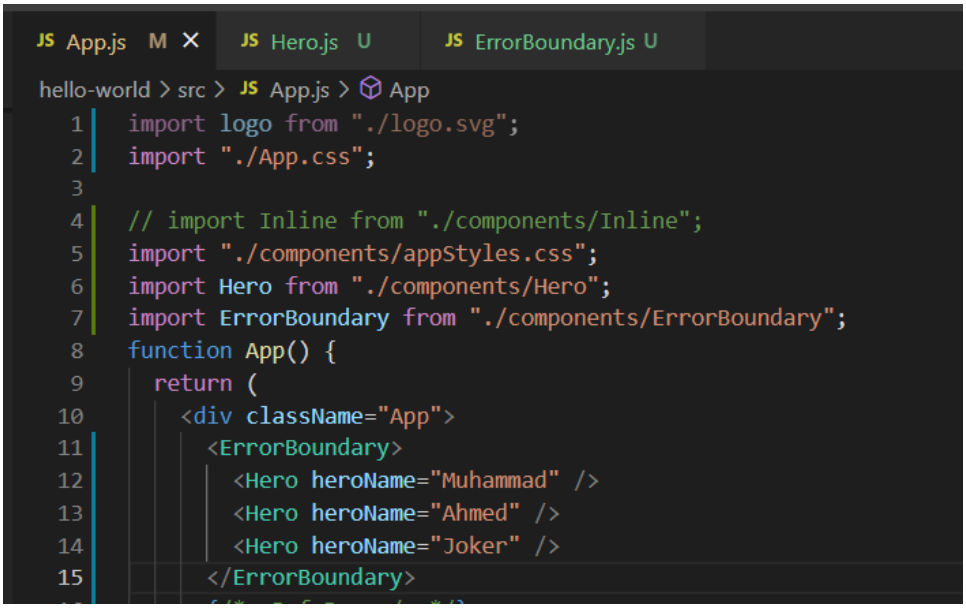


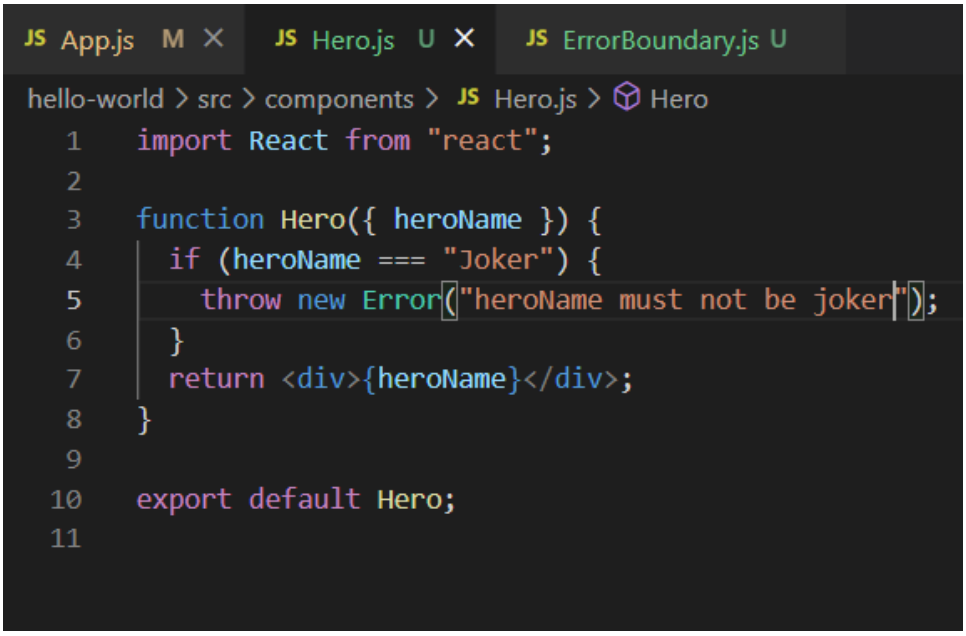
Because at this moment we can say that when something is wrong our whole application break so we don't want our application to break instead we want it shows the particular error instead.

It does not catch error which are under event handlers.



Let’s write the code to handle that





App.js M | Hero.js U | ErrorBoundary.js U

hello-world > src > components > ErrorBoundary.js > ErrorBoundary > render

```
1   import React, { Component } from "react";
2
3   class ErrorBoundary extends Component {
4     constructor(props) {
5       super(props);
6
7       this.state = {
8         hasError: false,
9       };
10    }
11    static getDerivedStateFromError() {
12      return {
13        hasError: true,
14      };
15    }
16
17    render() {
18      if (this.state.hasError) {
19        return <h1>Something went wrong</h1>;
20      }
21      return this.props.children;
22    }
23  }
24
25  export default ErrorBoundary;
26
```

App.js M | Hero.js U | ErrorBoundary.js U

hello-world > src > components > ErrorBoundary.js > ErrorBoundary > componentDidCatch

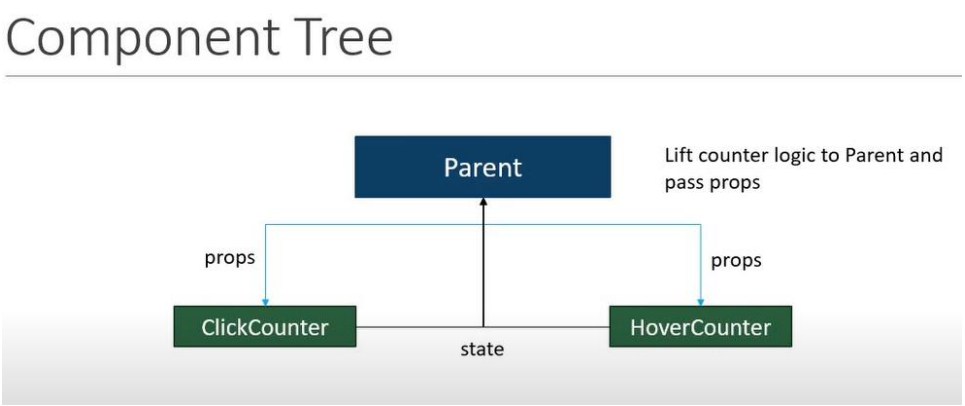
```
15    }
16    componentDidCatch(error, info) {
17      console.log(error);
18      console.log(info);
19    }
```

```
Error: heroName must not be joker
    at Hero (Hero.js:5:1)
    at renderWithHooks (react-dom.development.js:14985:1)
    at mountIndeterminateComponent (react-dom.development.js:17811:1)
    at beginWork (react-dom.development.js:19049:1)
    at HTMLUnknownElement.callCallback (react-dom.development.js:3945:1)
    at Object.invokeGuardedCallbackDev (react-dom.development.js:3994:1)
    at invokeGuardedCallback (react-dom.development.js:4056:1)
    at beginWork$1 (react-dom.development.js:23964:1)
    at performUnitOfWork (react-dom.development.js:22776:1)
    at workLoopSync (react-dom.development.js:22707:1)

▼{componentStack: '\n    at Hero (http://192.168.10.6:3000/static/js/b…197aca0.hot-update.js:25:5)\n    at div\n    at App'}
   componentStack: "\n    at Hero (http://192.168.10.6:3000/static/js/bundle.js:266:5)\n    at ErrorBoundary (http://192.168.10.6:3000/main.3
  ▶[[Prototype]]: Object
```

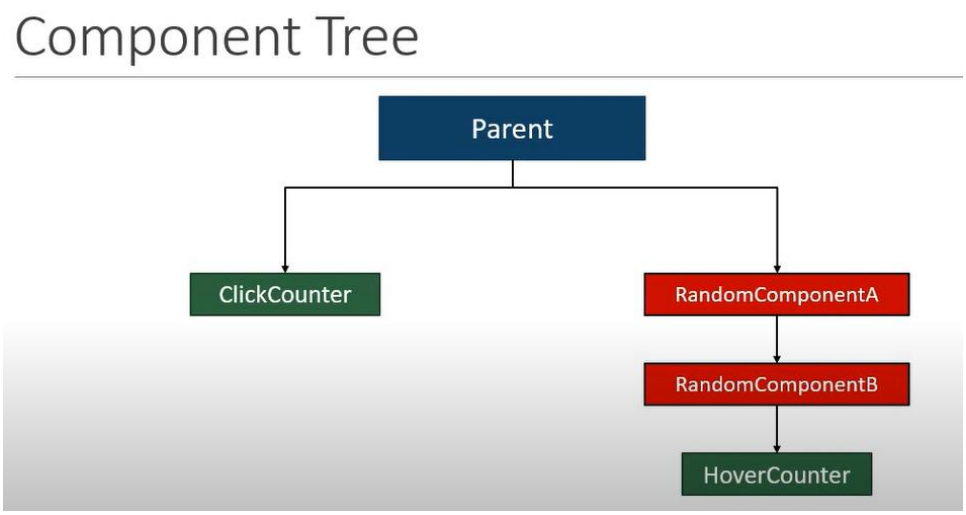
Higher Order Components (Part 1)

Concretely, **a higher-order component is a function that takes a component and returns a new component.**

Repeating functionality which the component used. So, how you can re-write the code that various component uses the same code



You can think of something like this what if we have some like the below one



Below is the demo code for describing the functionality

A basic counter functionality

What If someone says that you need to achieve that some function below is that function

If the same code using various components, then how we can handle that this is how we can do that

Higher Order Components (Part 2)

To share common functionality between components

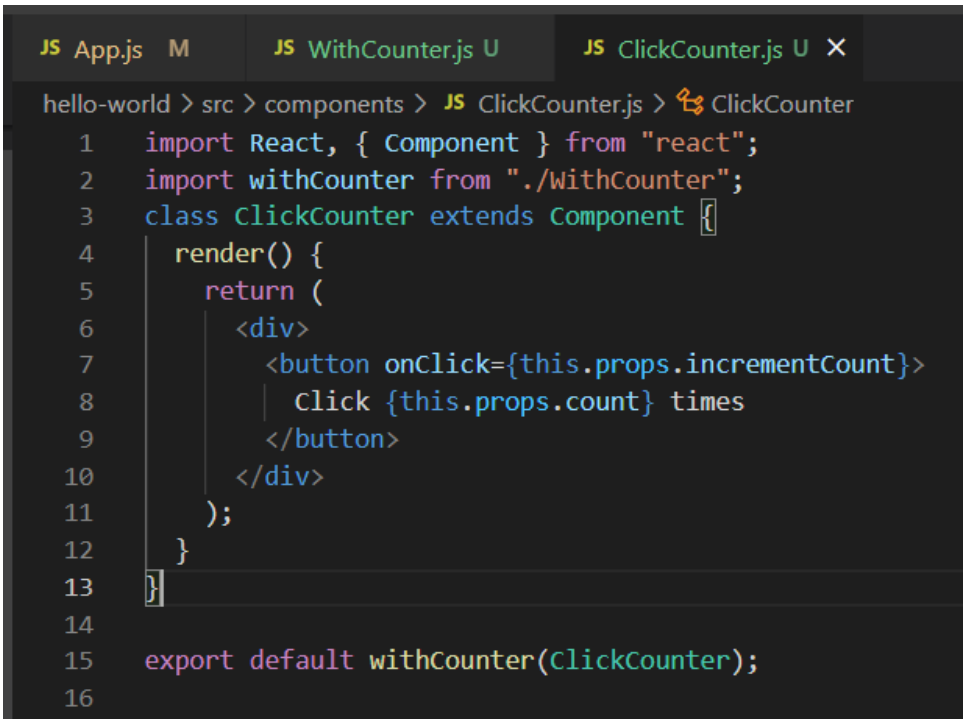
Higher Order Components - HOC

A pattern where a function takes a component as an argument and returns a new component.

const NewComponent = higherOrderComponent(originalComponent)

const EnhancedComponent = higherOrderComponent(originalComponent)

const IronMan = withSuit(TonyStark)


App.js M
WithCounter.js U
ClickCounter.js U
hello-world > src > components > ClickCounter.js > ClickCounter
1 import React, { Component } from "react";
2 import withCounter from "./WithCounter";
3 class ClickCounter extends Component {
4 render() {
5 return (
6
7 <button onClick={this.props.incrementCount}>
8 Click {this.props.count} times
9 </button>
10
11);
12 }
13 }
14
15 export default withCounter(ClickCounter);
16

App.js M | WithCounter.js U | ClickCounter.js U

hello-world > src > components > WithCounter.js > withCounter

```
import React, { Component } from "react";
const withCounter = (WrappedComponent) => {
  class WithCounter extends Component {
    constructor(props) {
      super(props);

      this.state = {
        count: 0,
      };
    }
    incrementCount = () => {
      this.setState((prevCount) => {
        return { count: prevCount.count + 1 };
      });
    };
    render() {
      return (
        <WrappedComponent
          count={this.state.count}
          incrementCount={this.incrementCount}
        />
      );
    }
  }
  return WithCounter;
};

export default withCounter;

```

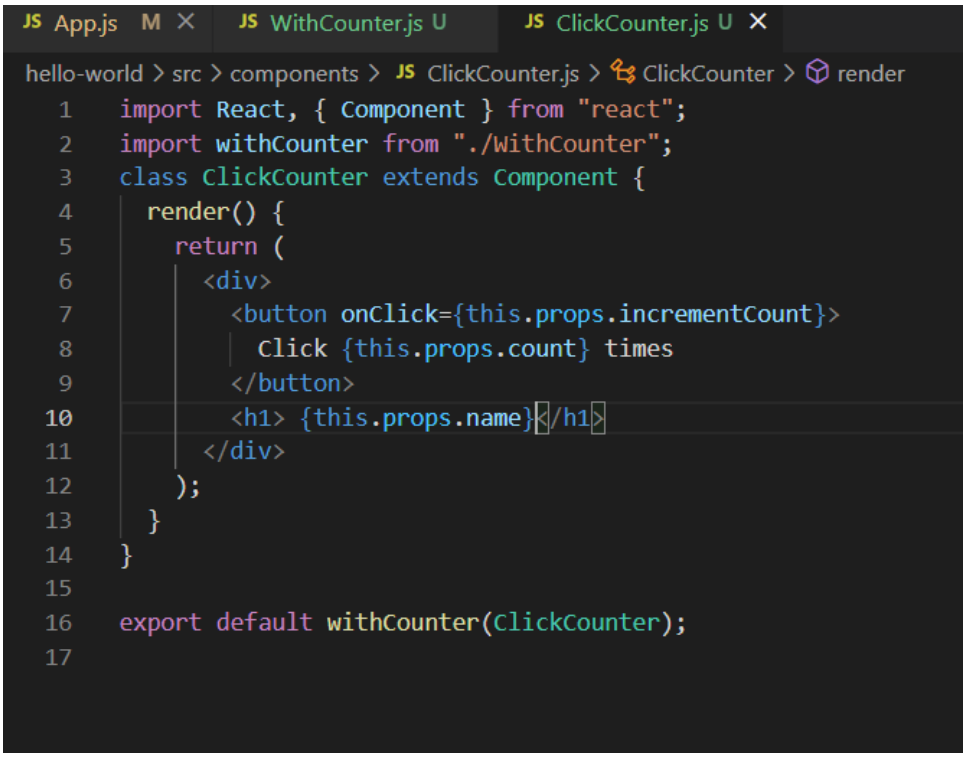
Higher Order Components (Part 3)

Passing down the prop.

hello-world > src > App.js > App

```
import logo from "./logo.svg";
import "./App.css";

// import Inline from "./components/Inline";
import "./components/appStyles.css";
import ClickCounter from "./components/ClickCounter";
import HoverCounter from "./components/HoverCounter";

function App() {
  return (
    <div className="App">
      <ClickCounter name="Muhammad" />
```

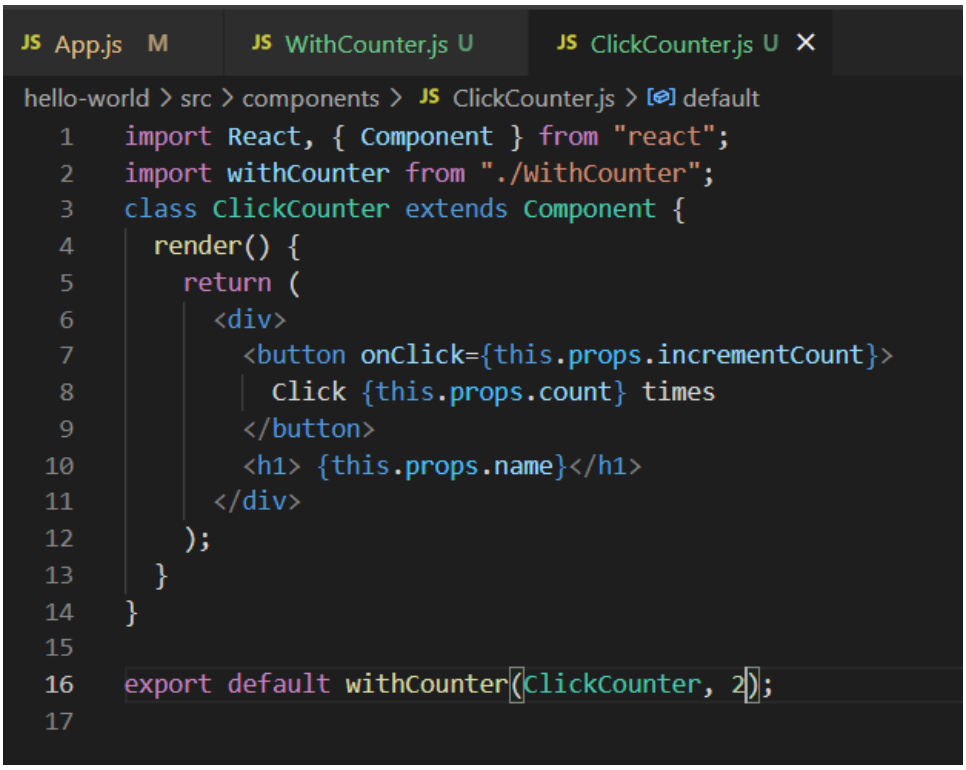
If you pass the prop and wants to use. If you want simply then it does not render



Basically, it passes the prop to the with counter and when it returns back then there is no prop so in order to do that, we need to return the prop from the with counter.

2nd thing

Passing parameters to the HOC function. We want to increment the counter by different number.


App.js M
WithCounter.js U
ClickCounter.js U
hello-world > src > components > ClickCounter.js > default
1 import React, { Component } from "react";
2 import withCounter from "./WithCounter";
3 class ClickCounter extends Component {
4 render() {
5 return (
6
7 <button onClick={this.props.incrementCount}>
8 Click {this.props.count} times
9 </button>
10 <h1> {this.props.name}</h1>
11
12);
13 }
14 }
15
16 export default withCounter(ClickCounter, 2);
17

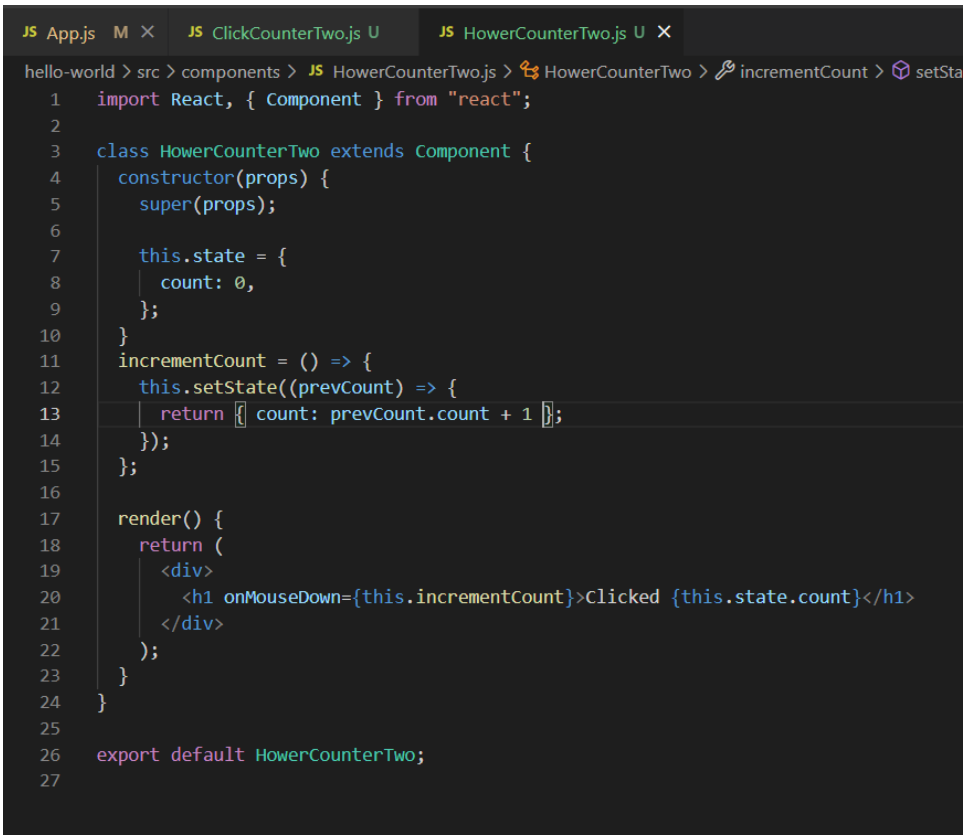
Render Props (Part 1)

It a second method sharing the code between different components. **Render props pattern** it is also used for sharing the code between the components.

App.js M ClickCounterTwo.js U

hello-world > src > components > ClickCounterTwo.js > ClickCounterTwo > incrementCount

```
1   import React, { Component } from "react";
2
3   class ClickCounterTwo extends Component {
4     constructor(props) {
5       super(props);
6
7       this.state = {
8         count: 0,
9       };
10    }
11    incrementCount = () => {
12      this.setState((prevState) => {
13        return { count: prevState.count + 1 };
14      });
15      console.log(this.state.count);
16    };
17
18    render() {
19      return (
20        <div>
21          <button onClick={this.incrementCount}>
22            button clicked {this.state.count}
23          </button>
24        </div>
25      );
26    }
27  }
28
29  export default ClickCounterTwo;
```

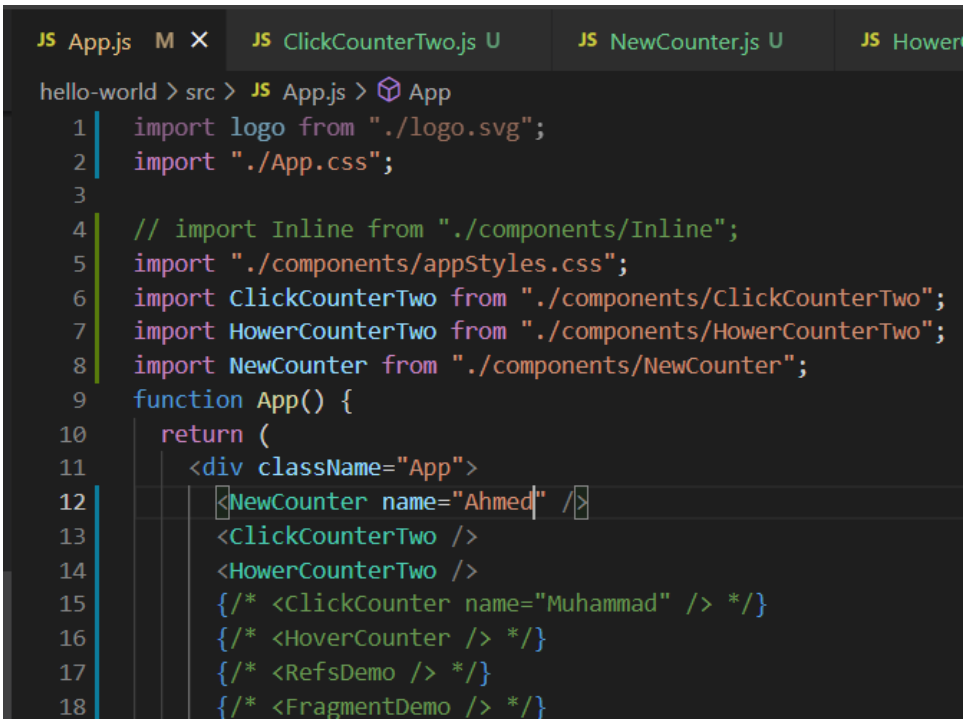


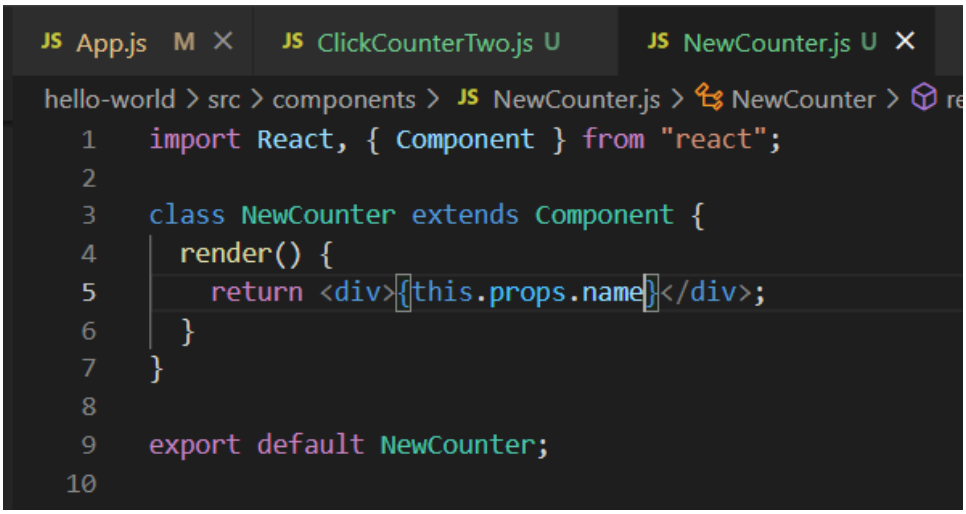
If you can see that the code is repeated **render props pattern**

Render props

The term “render prop” refers to a technique for **sharing code** between React components using a **prop whose value is a function**.

Let’s first consider a simple example





What if I want to get the same using the function

App.js M NewCounter.js U

hello-world > src > App.js > App

```
1  import logo from "./logo.svg";
2  import "./App.css";
3
4  // import Inline from "./components/Inline";
5  import "./components/appStyles.css";
6  import ClickCounterTwo from "./components/ClickCounterTwo";
7  import HowerCounterTwo from "./components/HowerCounterTwo";
8  import NewCounter from "./components/NewCounter";
9  function App() {
10   return (
11     <div className="App">
12       <NewCounter name={() => "Ahmed"} />
13       <ClickCounterTwo />
14       <HowerCounterTwo />
```

App.js M NewCounter.js U

hello-world > src > components > NewCounter.js > NewCounter > render

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

What if we want to pass a variable inside this

App.js M NewCounter.js U

hello-world > src > App.js > App

```
1  import logo from "./logo.svg";
2  import "./App.css";
3
4  // import Inline from "./components/Inline";
5  import "./components/appStyles.css";
6  import ClickCounterTwo from "./components/ClickCounterTwo";
7  import HowerCounterTwo from "./components/HowerCounterTwo";
8  import NewCounter from "./components/NewCounter";
9  function App() {
10   return (
11     <div className="App">
12       <NewCounter
13         name={(isLoggedUser) => (isLoggedUser ? "LoggedUser" : "Guest")}
14       />
```

App.js M NewCounter.js U

hello-world > src > components > NewCounter.js > NewCounter > render

```
1   import React, { Component } from "react";
2
3   class NewCounter extends Component {
4     render() {
5       return <div>{this.props.name(false)}</div>;
6     }
7   }
8
9   export default NewCounter;
10
```

Similarly, the render prop is using the same concept and they are passing into the render.

App.js M NewCounter.js U

hello-world > src > App.js > App

```
1   import logo from "./logo.svg";
2   import "./App.css";
3
4   // import Inline from "./components/Inline";
5   import "./components/appStyles.css";
6   import ClickCounterTwo from "./components/ClickCounterTwo";
7   import HowerCounterTwo from "./components/HowerCounterTwo";
8   import NewCounter from "./components/NewCounter";
9   function App() {
10    return (
11      <div className="App">
12        <NewCounter
13          render={(isLoggedUser) => (isLoggedUser ? "LoggedUser" : "Guest")}
14        />
```

App.js M NewCounter.js U

hello-world > src > components > NewCounter.js > NewCounter > render

```
1   import React, { Component } from "react";
2
3   class NewCounter extends Component {
4     render() {
5       return <div>{this.props.render(false)}</div>;
6     }
7   }
8
9   export default NewCounter;
10
```

Now, if we want to apply the concept inside the code reusability.

App.js M | HowerCounterTwo.js U | ClickCounterTwo.js U | NewCounter.js U

hello-world > src > App.js > App

```
import logo from "./logo.svg";
import "./App.css";

// import Inline from "./components/Inline";
import "./components/appStyles.css";
import ClickCounterTwo from "./components/ClickCounterTwo";
import HowerCounterTwo from "./components/HowerCounterTwo";
import NewCounter from "./components/NewCounter";
function App() {
  return (
    <div className="App">
      <NewCounter
        render={(count, incrementCount) => (
          <ClickCounterTwo count={count} incrementCount={incrementCount} />
        )}
      />
      <NewCounter
        render={(count, incrementCount) => (
          <HowerCounterTwo count={count} incrementCount={incrementCount} />
        )}
      />
```

App.js M | HowerCounterTwo.js U | ClickCounterTwo.js U | NewCounter.js U

hello-world > src > components > HowerCounterTwo.js > HowerCounterTwo > render

```
import React, { Component } from "react";

class HowerCounterTwo extends Component {
  render() {
    const { count, incrementCount } = this.props;
    return (
      <div>
        <h1 onMouseDown={incrementCount}>Clicked {count}</h1>
      </div>
    );
  }
}

export default HowerCounterTwo;
```

App.js M | HowerCounterTwo.js U | ClickCounterTwo.js U | NewCounter.js U

hello-world > src > components > ClickCounterTwo.js > ClickCounterTwo > render

```
1  import React, { Component } from "react";
2
3  class ClickCounterTwo extends Component {
4    render() {
5      return (
6        <div>
7          <button onClick={this.props.incrementCount}>
8            button clicked {this.props.count}
9          </button>
10       </div>
11     );
12   }
13 }
14
15 export default ClickCounterTwo;
16
```

App.js M | HowerCounterTwo.js U | ClickCounterTwo.js U | NewCounter.js U

hello-world > src > components > NewCounter.js > NewCounter > render

```
1  import React, { Component } from "react";
2
3  class NewCounter extends Component {
4    constructor(props) {
5      super(props);
6
7      this.state = {
8        count: 0,
9      };
10   }
11   incrementCount = () => {
12     this.setState((prevCount) => {
13       return { count: prevCount.count + 1 };
14     });
15   };
16   render() {
17     return (
18       <div>{this.props.render(this.state.count, this.incrementCount)}</div>
19     );
20   }
21 }
22
23 export default NewCounter;
24
```

In some cases, you even did not use the render method instead you use the **children** directly.

App.js M | HowerCounterTwo.js U | ClickCounterTwo.js U | NewCounter.js U

hello-world > src > App.js > App

```
import logo from "./logo.svg";
import "./App.css";

// import Inline from "./components/Inline";
import "./components/appStyles.css";
import ClickCounterTwo from "./components/ClickCounterTwo";
import HowerCounterTwo from "./components/HowerCounterTwo";
import NewCounter from "./components/NewCounter";
function App() {
  return (
    <div className="App">
      <NewCounter>
        {(count, incrementCount) => (
          <ClickCounterTwo count={count} incrementCount={incrementCount} />
        )}
      </NewCounter>
      <NewCounter>
        {(count, incrementCount) => (
          <HowerCounterTwo count={count} incrementCount={incrementCount} />
        )}
      </NewCounter>
```

App.js M | HowerCounterTwo.js U | ClickCounterTwo.js U | NewCounter.js U

hello-world > src > components > NewCounter.js > NewCounter > render

```
import React, { Component } from "react";

class NewCounter extends Component {
  constructor(props) {
    super(props);

    this.state = {
      count: 0,
    };
  }
  incrementCount = () => {
    this.setState((prevCount) => {
      return { count: prevCount.count + 1 };
    });
  };
  render() {
    return (
      <div>{this.props.children(this.state.count, this.incrementCount)}</div>
    );
  }
}

export default NewCounter;
```

App.js M | HowerCounterTwo.js U | ClickCounterTwo.js U | NewCounter.js U

hello-world > src > components > ClickCounterTwo.js > ClickCounterTwo > render

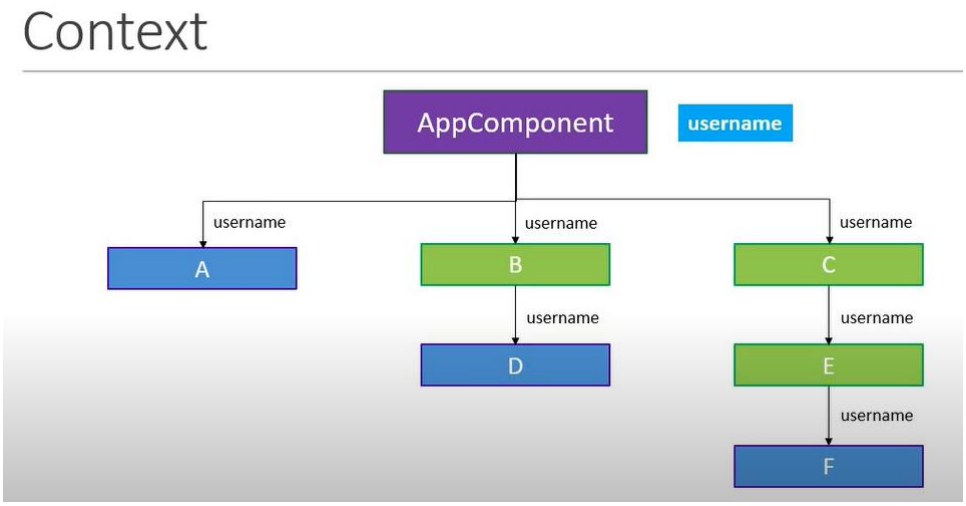
```
1   import React, { Component } from "react";
2
3   class ClickCounterTwo extends Component {
4     render() {
5       return (
6         <div>
7           <button onClick={this.props.incrementCount}>
8             button clicked {this.props.count}
9           </button>
10        </div>
11      );
12    }
13  }
14
15  export default ClickCounterTwo;
16
```

App.js M | HowerCounterTwo.js U | ClickCounterTwo.js U | NewCounter.js U

hello-world > src > components > HowerCounterTwo.js > HowerCounterTwo > render

```
1   import React, { Component } from "react";
2
3   class HowerCounterTwo extends Component {
4     render() {
5       const { count, incrementCount } = this.props;
6       return (
7         <div>
8           <h1 onMouseDown={incrementCount}>Clicked {count}</h1>
9         </div>
10      );
11    }
12  }
13
14  export default HowerCounterTwo;
15
```

Context (Part 1)



Let say if we want to use the username down in the A, D, F components which are present in the app component so what we need to do is passing the values from top to bottom by tree list but what if we want to pass towards the end of the component without traversing through the nodes. So, here is the solution we can use it by **context** concept in the react.

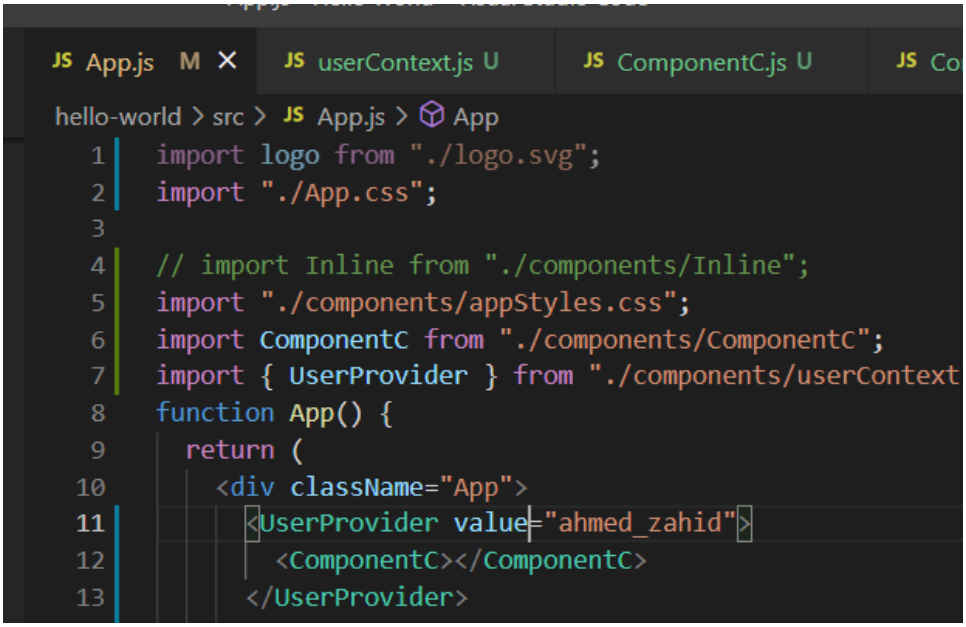
Context

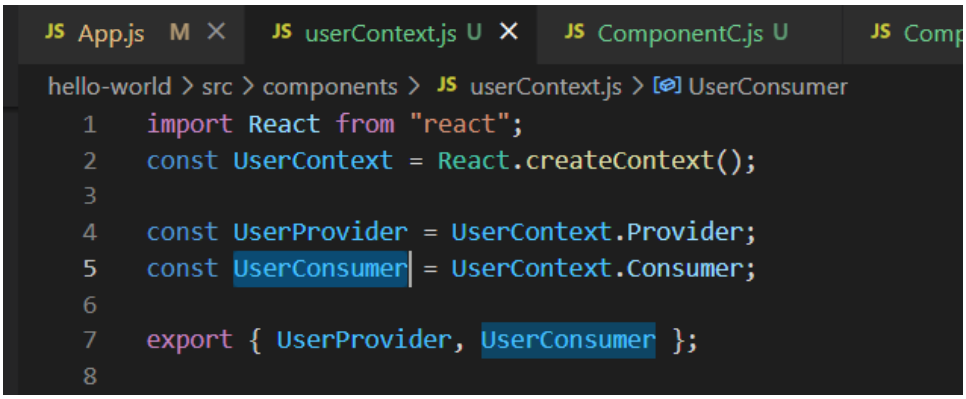
Context provides a way to pass data through the component tree without having to pass props down manually at every level.

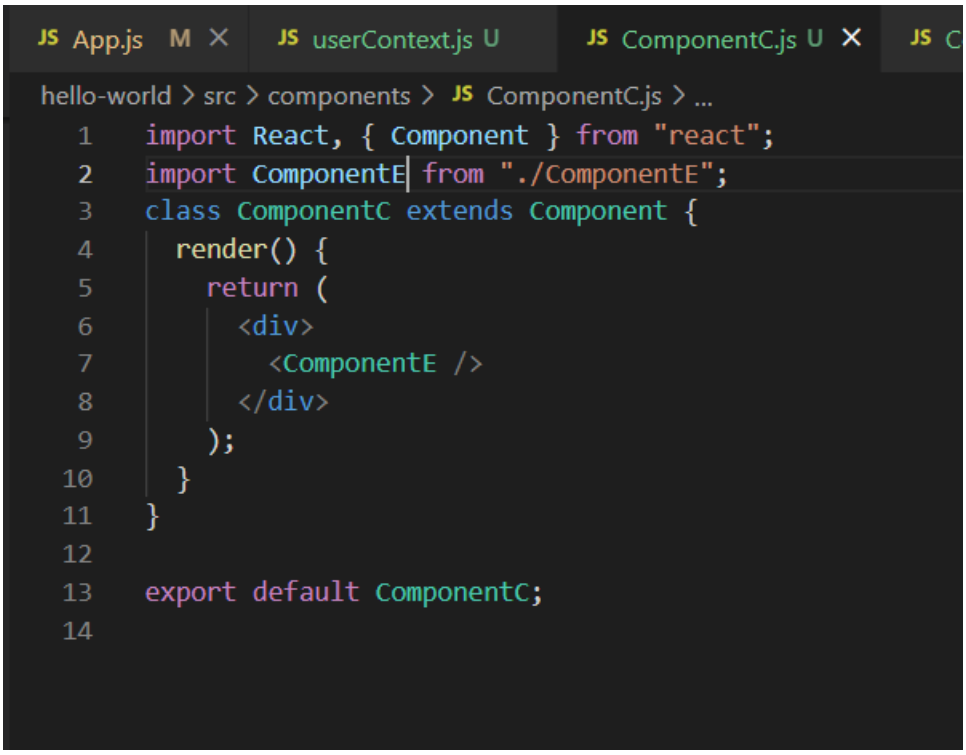
Context

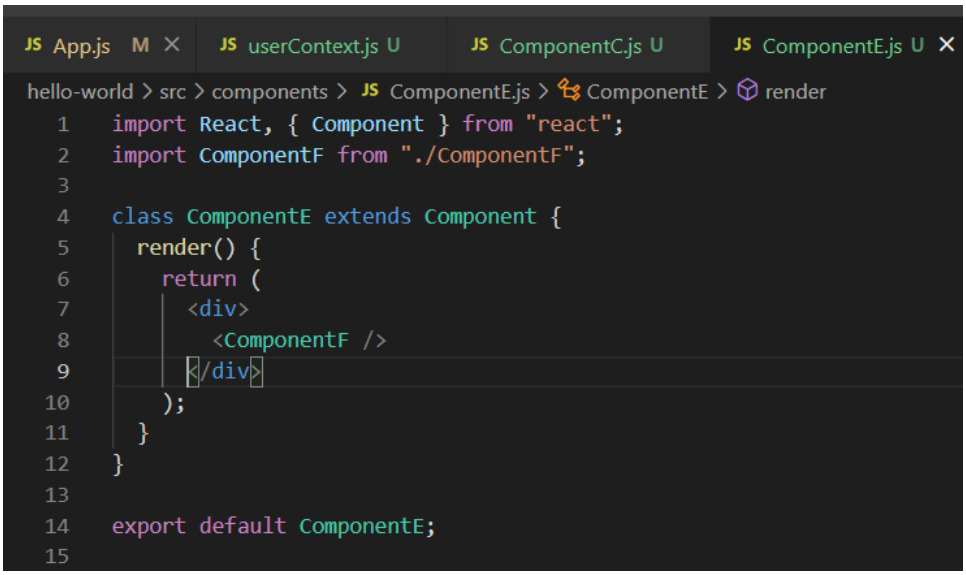
1. Create the context
2. Provide a context value
3. Consume the context value

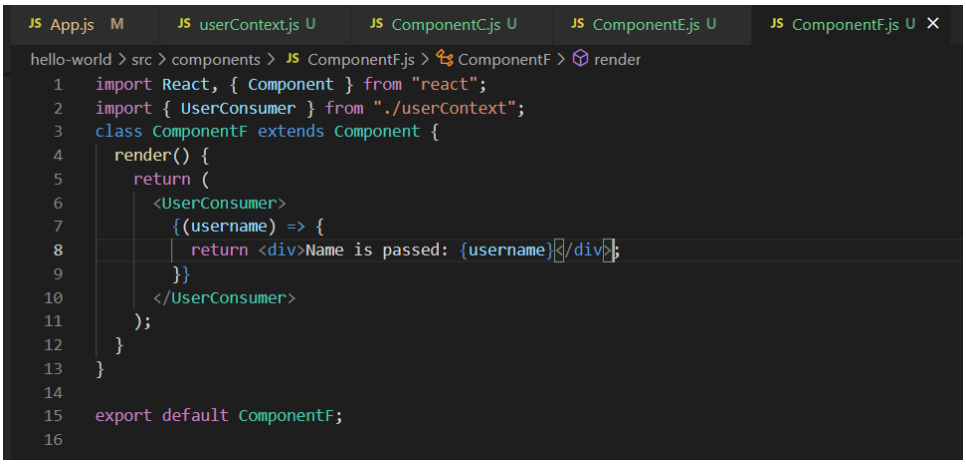
Below are the code for reference




App.js M
userContext.js U
ComponentC.js U
hello-world > src > components > userContext.js > UserConsumer
1 import React from "react";
2 const UserContext = React.createContext();
3
4 const UserProvider = UserContext.Provider;
5 const UserConsumer = UserContext.Consumer;
6
7 export { UserProvider, UserConsumer };
8


App.js M
userContext.js U
ComponentC.js U
hello-world > src > components > ComponentC.js > ...
1 import React, { Component } from "react";
2 import ComponentE from "./ComponentE";
3 class ComponentC extends Component {
4 render() {
5 return (
6
7 <ComponentE />
8
9);
10 }
11 }
12
13 export default ComponentC;
14





Context (Part 3)

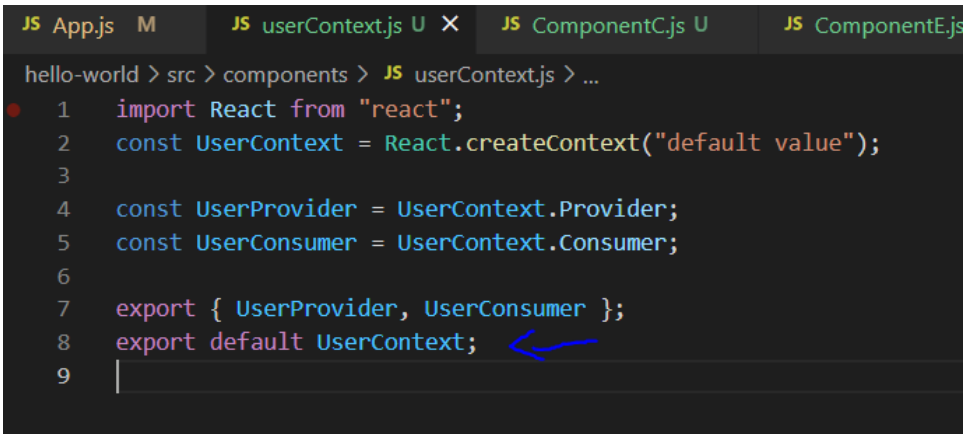
You can set the default value of the context

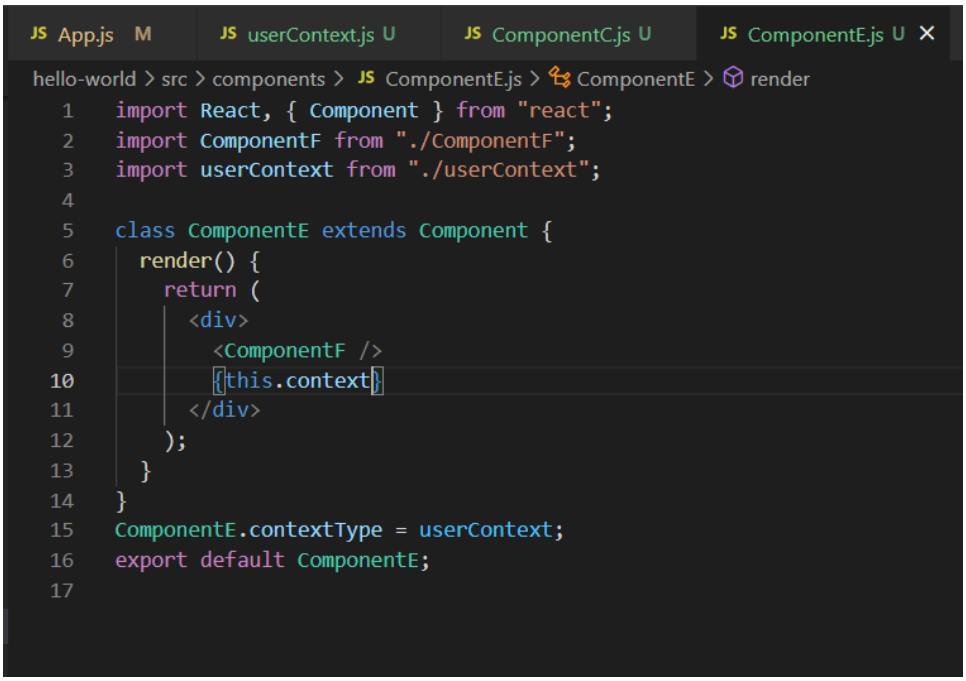
2nd is the context type property

It only works with class component

It works with single one component not multiple values but a number of times in your application you want to use them.

Now this is another way of accessing the value in the component

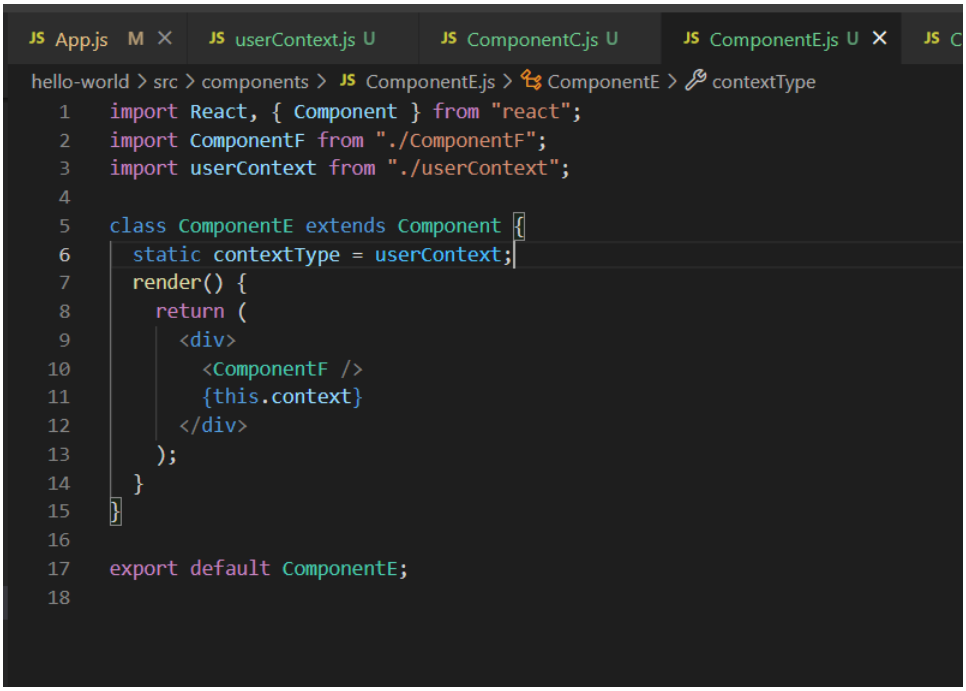



App.js M
userContext.js U
ComponentC.js U
ComponentE.js U
hello-world > src > components > ComponentE.js > ComponentE > render
import React, { Component } from "react";
import ComponentF from "./ComponentF";
import userContext from "./userContext";
class ComponentE extends Component {
render() {
return (

<ComponentF />
{this.context}

);
}
}
ComponentE.contextType = userContext;
export default ComponentE;

Or

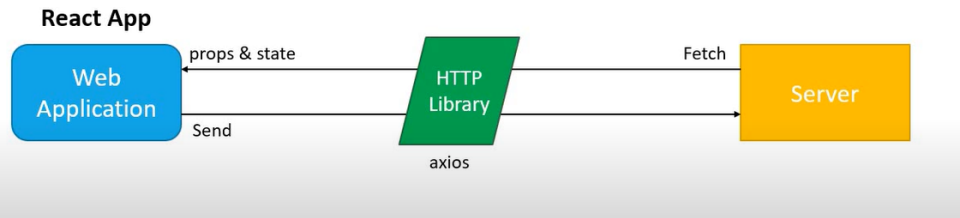

App.js M
userContext.js U
ComponentC.js U
ComponentE.js U
hello-world > src > components > ComponentE.js > ComponentE > contextType
import React, { Component } from "react";
import ComponentF from "./ComponentF";
import userContext from "./userContext";
class ComponentE extends Component {
static contextType = userContext;
render() {
return (

<ComponentF />
{this.context}

);
}
}
export default ComponentE;

HTTP and React

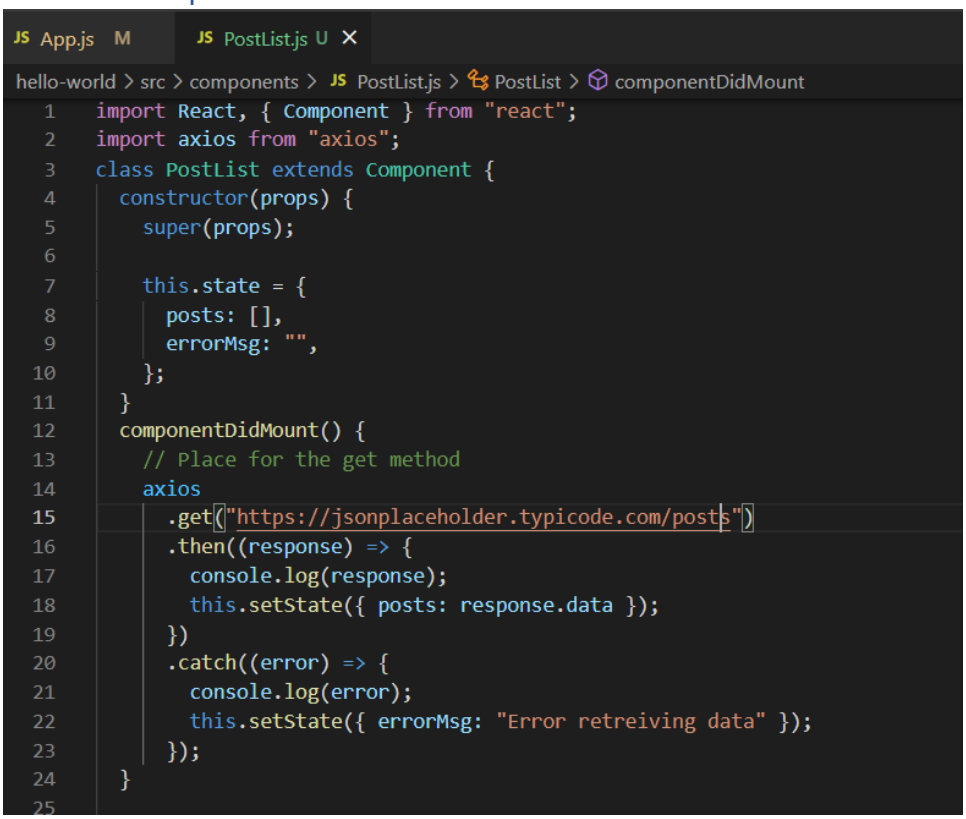
React and HTTP




```
D:\react\Hello World\hello-world>npm install axios
```

Installing the package and then you can see that package inside project folder.

HTTP GET Request



HTTP Post Request

App.js M | PostForm.js U

hello-world > src > components > PostForm.js > PostForm > submitHandler > catch() callback

```
1   import React, { Component } from "react";
2   import axios from "axios";
3   class PostForm extends Component {
4     constructor(props) {
5       super(props);
6
7       this.state = {
8         userId: "",
9         title: "",
10        body: "",
11      };
12    }
13    submitHandler = (e) => {
14      e.preventDefault();
15      console.log("submit Handler is clicked");
16      console.log(this.state);
17      axios
18        .post("https://jsonplaceholder.typicode.com/posts", this.state)
19        .then((response) => console.log(response))
20        .catch((error) => console.log(error));
21    };
22    changeHandler = (e) => {
23      this.setState({ [e.target.name]: e.target.value });
24    };
```

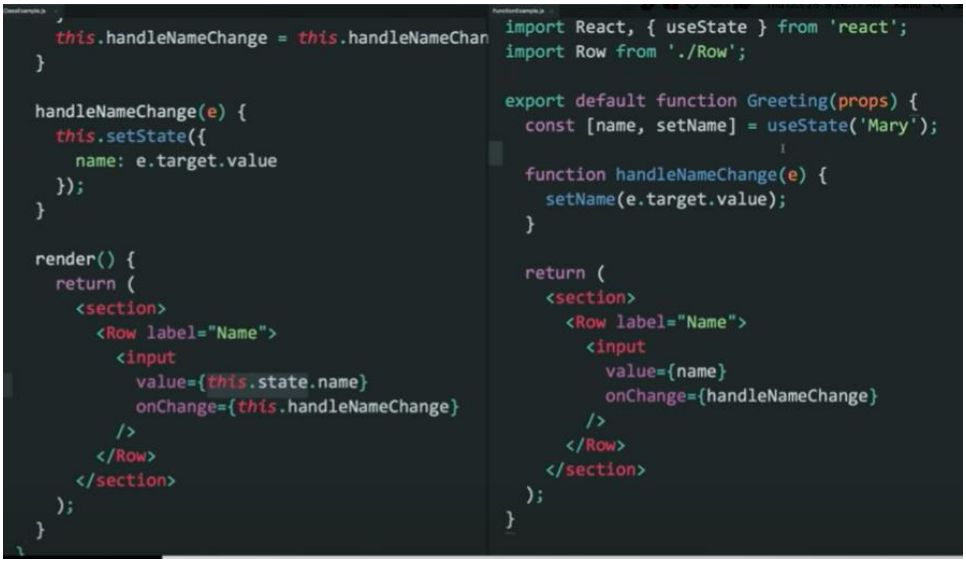
App.js M | PostForm.js U

hello-world > src > components > PostForm.js > PostForm > submitHandler > catch() callback

```
25    render() {
26      const { userId, title, body } = this.state;
27      return (
28        <div>
29          <form onSubmit={this.submitHandler}>
30            <input
31              type="text"
32              onChange={this.changeHandler}
33              value={userId}
34              name="userId"
35            />
36            <br />
37            <input
38              type="text"
39              onChange={this.changeHandler}
40              value={title}
41              name="title"
42            />
43            <br />
44            <input
45              type="text"
46              onChange={this.changeHandler}
47              value={body}
48              name="body"
49            />
50            <br />
51            <button type="submit">Submit</button>
52          </form>
53        </div>
54      );
55    }
56  }
57
58  export default PostForm;
59
```

React Hooks

It is an important feature you can use basically you can use the stateful components or variable inside the react without using the **class.**



```
    this.handleNameChange = this.handleNameChan
  }

  handleNameChange(e) {
    this.setState({
      name: e.target.value
    });
  }

  render() {
    return (
      <section>
        <Row label="Name">
          <input
            value={this.state.name}
            onChange={this.handleNameChange}
          />
        </Row>
      </section>
    );
  }
```

```
import React, { useState } from 'react';
import Row from './Row';

export default function Greeting(props) {
  const [name, setName] = useState('Mary');

  function handleNameChange(e) {
    setName(e.target.value);
  }

  return (
    <section>
      <Row label="Name">
        <input
          value={name}
          onChange={handleNameChange}
        />
      </Row>
    </section>
  );
}
```

If you want to use two different variables.

```
export default function Greeting(props) {
  const [name, setName] = useState('Mary');
  const [surname, setSurname] = useState('Poppins');

  function handleNameChange(e) {
    setName(e.target.value);
  }

  function handleSurnameChange(e) {
    setSurname(e.target.value);
  }

  return (
    <section>
      <Row label="Name">
        <input
          value={name}
          onChange={handleNameChange}
        />
```

You can use the context inside the Hooks

```
import React, { useState, useContext } from 'react
import Row from './Row';
import { ThemeContext, LocaleContext } from './co

export default function Greeting(props) {
  const [name, setName] = useState('Mary');
  const [surname, setSurname] = useState('Poppins
  const theme = useContext(ThemeContext);
  const locale = useContext(LocaleContext);

  function handleNameChange(e) {
    setName(e.target.value);
  }

  function handleSurnameChange(e) {
    setSurname(e.target.value);
  }

  return (
```

You cannot use **if** condition when working with the hooks it should be used in the upper level.

```
useEffect(() => {
  document.title = name + ' ' + surname;
});
```

You can use **useEffect()** method if you don't want to use the component did mount and component did update it will do both the functionalities for you.

```
componentDidMount() {
  document.title = this.state.name + ' ' + this.state.surname;
  window.addEventListener('resize', this.handleResize);
}

componentDidUpdate() {
  document.title = this.state.name + ' ' + this.state.surname;
}

componentWillUnmount() {
  window.removeEventListener('resize', this.handleResize);
}

handleResize() {
  this.setState({
    width: window.innerWidth
  });
}

handleNameChange(e) {
  this.setState({
```

Let say if you want to update the value of the window which is changed over the period of time this will do

```
useEffect(() => {
  document.title = name + ' ' + surname;
});

const [width, setWidth] = useState(window.innerWidth);
useEffect(() => {
  const handleResize = () => setWidth(window.innerWidth);
  window.addEventListener('resize', handleResize);
  return () => {
    window.removeEventListener('resize', handleResize);
  };
});

function handleNameChange(e) {
  setName(e.target.value);
}

function handleSurnameChange(e) {
  setSurname(e.target.value);
}
```

It will do the subscribe and unsubscribe both methods.

Hooks: it does not what is the life cycle of the class instead it is what your code is actually doing.

If you want to pass data from one function to another function then you can do that.

These are called the custom hooks which are used **with** the following keyword.

Custom hooks are JavaScript functions.

Routers

Hooks

Introduction

Why Hooks?

Reason Set 1

Understand how ***this*** keyword works in JavaScript

Remember to bind event handlers in class components

Classes don't minify very well and make hot reloading very unreliable

Why Hooks?

Reason Set 2

There is no particular way to reuse stateful component logic

HOC and render props patterns do address this problem

Makes the code harder to follow

There is need a to share stateful logic in a better way

Reason Set 3

Create components for complex scenarios such as data fetching and subscribing to events

Related code is not organized in one place

Ex: Data fetching - In componentDidMount and componentDidUpdate

Ex: Event listeners – In componentDidMount and componentWillUnmount

Because of stateful logic – Cannot break components into smaller ones

React version 16.8 or higher

Completely opt in

Hooks don’t contain any breaking changes and the release is 100% backwards-compatible.

Classes won’t be removed from React

Can’t use Hooks inside of a class component

Hooks don’t replace your existing knowledge of React concepts

Instead, Hooks provide a more direct API to the React concepts you already know

Hooks are a new feature addition in React version 16.8

They allow you to use React features without having to write a class

Avoid the whole confusion with 'this' keyword

Allow you to reuse stateful logic

Organize the logic inside a component into reusable isolated units

React Hooks Tutorial - 2 - useState Hook

Rules of Hooks

"Only Call Hooks at the Top Level“

Don’t call Hooks inside loops, conditions, or nested functions

"Only Call Hooks from React Functions“

Call them from within React functional components and not just any regular JavaScript function

App.js M | PostForm.js U | HookCounter.js U

hello-world > src > components > HookCounter.js > HookCounter

```
1   import React, { useState } from "react";
2
3   function HookCounter() {
4     const [count, setCount] = useState(0);
5     return (
6       <div>
7         <button onClick={() => setCount(count + 1)}>Click {count}</button>
8       </div>
9     );
10  }
11
12  export default HookCounter;
13
```

useState with previous state

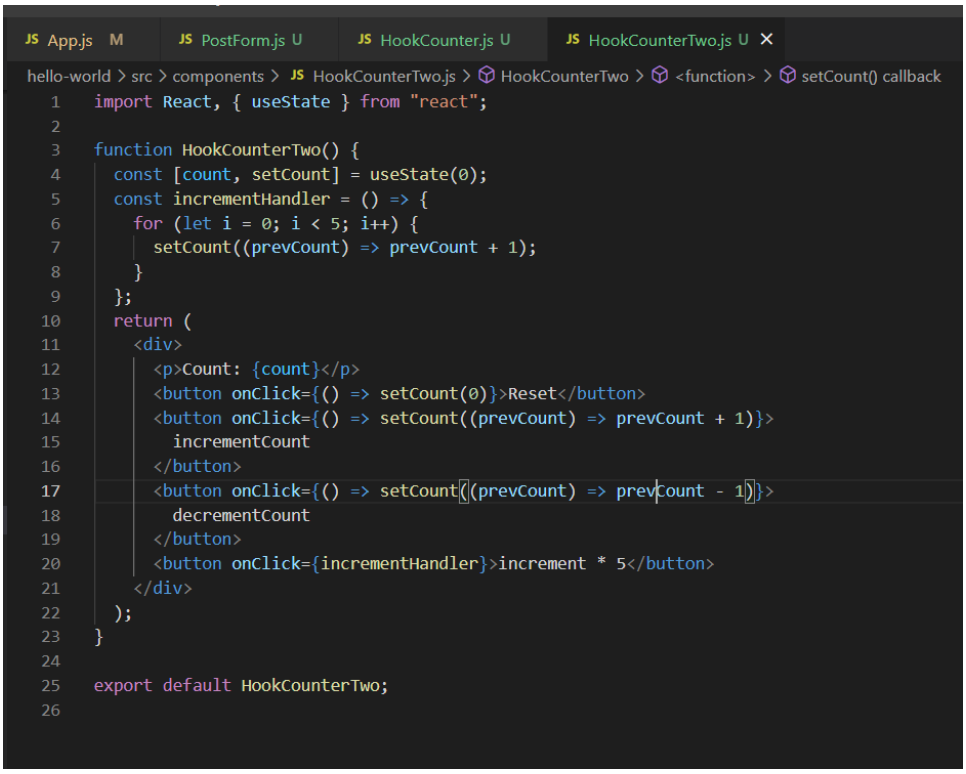
If you want to use the value of a variable then you need to pass that value as a function.

App.js M | PostForm.js U | HookCounter.js U | HookCounterTwo.js U

hello-world > src > components > HookCounterTwo.js > HookCounterTwo

```
1   import React, { useState } from "react";
2
3   function HookCounterTwo() {
4     const [count, setCount] = useState(0);
5     const incrementHandler = () => {
6       for (let i = 0; i < 5; i++) {
7         setCount(count + 1);
8       }
9     };
10    return (
11      <div>
12        <p>Count: {count}</p>
13        <button onClick={() => setCount(0)}>Reset</button>
14        <button onClick={() => setCount(count + 1)}>incrementCount</button>
15        <button onClick={() => setCount(count - 1)}>decrementCount</button>
16        <button onClick={incrementHandler}>increment * 5</button>
17      </div>
18    );
19  }
20
21  export default HookCounterTwo;
22
```

If you can see when you are doing the 5x then it does not do the five times instead it is doing one time so it is good practice if we are doing something using the previous state then we use the arrow function in order to update the value.



useState with object

Use state object can be a number, array, Boolean or object.

The object is not get updated when one entry inside object is updated while the other not because use state does not automatically merge and update the object.

Because the set state will merge the state in case of component while in case of use state it does not perform the both steps.

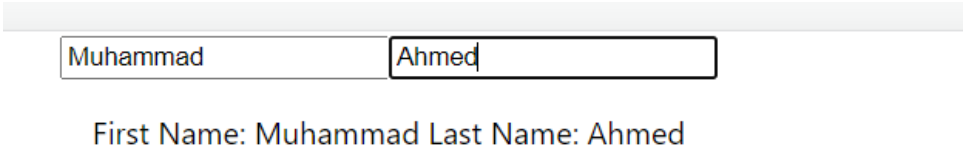
You can do it using the … it will tell that copy the whole object structure and when you copy the whole structure then you can just update the particular value.

App.js M HookCounterThree.js U

hello-world > src > components > HookCounterThree.js > HookCounterThree > <function> > lastname

```
import React, { useState } from "react";

function HookCounterThree() {
  const [name, setname] = useState({ firstname: "", lastname: "" });
  return (
    <div>
      <input
        type="text"
        value={name.firstname}
        onChange={(e) => setname({ firstname: e.target.value })}
      />
      <input
        type="text"
        value={name.lastname}
        onChange={(e) => setname({ lastname: e.target.value })}
      />
      <p>
        First Name: {name.firstname} Last Name: {name.lastname}
      </p>
    </div>
  );
}

export default HookCounterThree;
```

If you can see from the above example, it is clearly showed that the whole object is not updated and cannot merged in one so what is the possible solution for that it is you can use the object … decompression.



App.js M | HookCounterThree.js U

hello-world > src > components > HookCounterThree.js > HookCounterThree > <function>

```
1   import React, { useState } from "react";
2
3   function HookCounterThree() {
4     const [name, setname] = useState({ firstname: "", lastname: "" });
5     return (
6       <div>
7         <input
8           type="text"
9           value={name.firstname}
10          onChange={(e) => setname({ ...name, firstname: e.target.value })}
11        />
12        <input
13          type="text"
14          value={name.lastname}
15          onChange={(e) => setname({ ...name, lastname: e.target.value })}
16        />
17        <p>
18          First Name: {name.firstname} Last Name: {name.lastname}
19        </p>
20      </div>
21    );
22  }
23
24  export default HookCounterThree;
25
```

useState with array

Summary - useState

The useState hook lets you add state to functional components

In classes, the state is always an object.

With the useState hook, the state doesn't have to be an object.

The useState hook returns an array with 2 elements.

The first element is the current value of the state, and the second element is a state setter function.

New state value depends on the previous state value? You can pass a function to the setter function.

When dealing with objects or arrays, always make sure to spread your state variable and then call the setter function

App.js M HookCounterFour.js U

hello-world > src > components > HookCounterFour.js > HookCounterFo

```jsx
import React, { useState } from "react";

function HookCounterFour() {
  const [items, setItems] = useState([]);
  const addItem = () => {
    setItems([
      ...items,
      {
        id: items.length,
        value: Math.floor(Math.random() * 10) + 1,
      },
    ]);
  };
  return (
    <div>
      <button onClick={addItem}>Add item</button>

      <ul>
        {items.map((item) => (
          <li key={item.id}>
            {item.value} {item.id}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default HookCounterFour;
```

useEffect Hook

useEffect

Timer

```
componentDidMount() {
  this.interval = setInterval(this.tick, 1000)
}
componentWillUnmount() {
  clearInterval(this.interval)
}
```

useEffect

Combine the two side effects

```
componentDidMount() {
  document.title = `You clicked ${this.state.count} times`;
  this.interval = setInterval(this.tick, 1000)
}
componentDidUpdate() {
  document.title = `You clicked ${this.state.count} times`;
}
componentWillUnmount() {
  clearInterval(this.interval)
}
```

If you can see that the code is repeated two times for facilitate the same functionality what we can use if you want not to repeat the code.

useEffect

The Effect Hook lets you perform **side effects** in **functional components**

It is a close replacement for ***componentDidMount***, ***componentDidUpdate*** and ***componentWillUnmount***

useEffect after render

Use effect method is called after every render method. Just like the component did update.

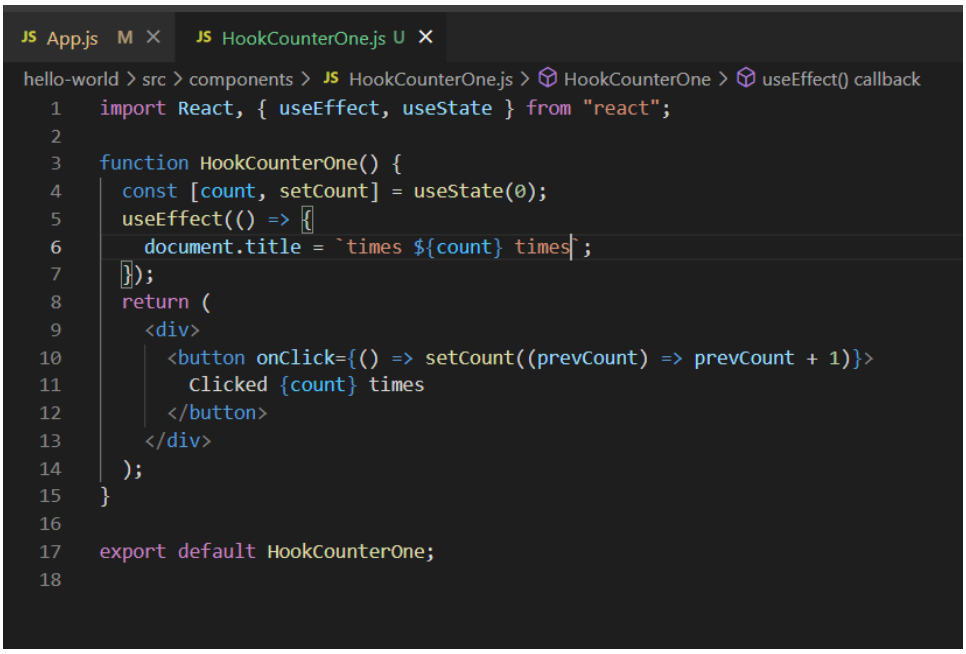
Using class

App.js M ClassCounterOne.js U

hello-world > src > components > ClassCounterOne.js > ClassCounterOne > clickHa

```
1   import React, { Component } from "react";
2
3   class ClassCounterOne extends Component {
4     constructor(props) {
5       super(props);
6
7       this.state = {
8         count: 0,
9       };
10    }
11    clickHandler = () => {
12      this.setState((prevState) => {
13        return { count: prevState.count + 1 };
14      });
15    };
16    componentDidMount() {
17      document.title = `Clicked ${this.state.count} times`;
18    }
19    componentDidUpdate() {
20      document.title = `Clicked ${this.state.count} times`;
21    }
22
23    render() {
24      return (
25        <div>
26          <button onClick={this.clickHandler}>
27            Click {this.state.count} times
28          </button>
29        </div>
30      );
31    }
32  }
33
34  export default ClassCounterOne;
35
```

Using hooks



Conditionally run effects

We want to do something like that after render method the **use effect** is automatically called but you want to put a **condition** that the **use effect** is not called. We are passing an array [] it continuously checking the value whatever you are putting inside that.

Class Example

If you can see whenever the render method is called the component did mount is called while the code inside that function is only responsible for updating the value in the browser. So, we put a condition to that the render method not going to render each time.

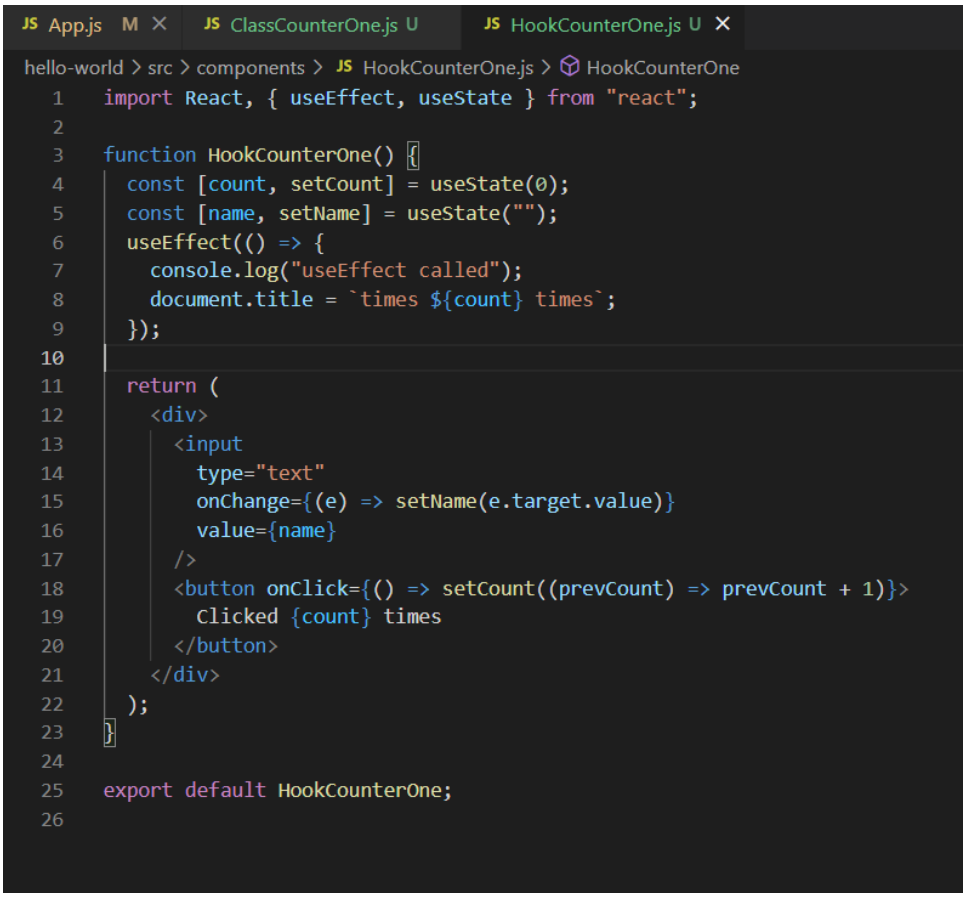
App.js M | ClassCounterOne.js U | HookCounterOne.js U

hello-world > src > components > ClassCounterOne.js > ClassCounterOne > componentDidUpdate

```
import React, { Component } from "react";
class ClassCounterOne extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0,
      name: "",
    };
  }
  clickHandler = () => {
    this.setState((prevState) => {
      return { count: prevState.count + 1 };
    });
  };
  componentDidMount() {
    document.title = `Clicked ${this.state.count} times`;
  }
  componentDidUpdate(prevProps, prevState) {
    if (prevState.count != this.state.count) {
      console.log("Clicked updated");
      document.title = `Clicked ${this.state.count} times`;
    }
  }
  inputHandler = (e) => {
    this.setState(() => {
      return { name: e.target.value };
    });
  };
  render() {
    return (
      <div>
        <input type="text" onChange={this.inputHandler} value={this.name} />
        <button onClick={this.clickHandler}>
          Click {this.state.count} times
        </button>
      </div>
    );
```

Now the above code having the condition reduce the effort on the application.

Hook Example



Every time the render method is called it automatically called the use effect method; we can avoid them using the condition which is need to be put inside the use effect method and it should be in array format and it continuously checking the variables if it gets changed then it called the method otherwise not.

App.js M ClassCounterOne.js U HookCounterOne.js U

llo-world > src > components > HookCounterOne.js > HookCounterOne

```jsx
import React, { useEffect, useState } from "react";

function HookCounterOne() {
  const [count, setCount] = useState(0);
  const [name, setName] = useState("");
  useEffect(() => {
    console.log("useEffect called");
    document.title = `times ${count} times`;
  }, [count]);

  return (
    <div>
      <input
        type="text"
        onChange={(e) => setName(e.target.value)}
        value={name}
      />
      <button onClick={() => setCount((prevCount) => prevCount + 1)}>
        Clicked {count} times
      </button>
    </div>
  );
}

export default HookCounterOne;

```

Run effects only once

Basically, how we mimic the **component did mount** inside the functional component hooks. We can mimic by just using the [] square bracket and it will only bind it ones.

X - 201 Y- 182

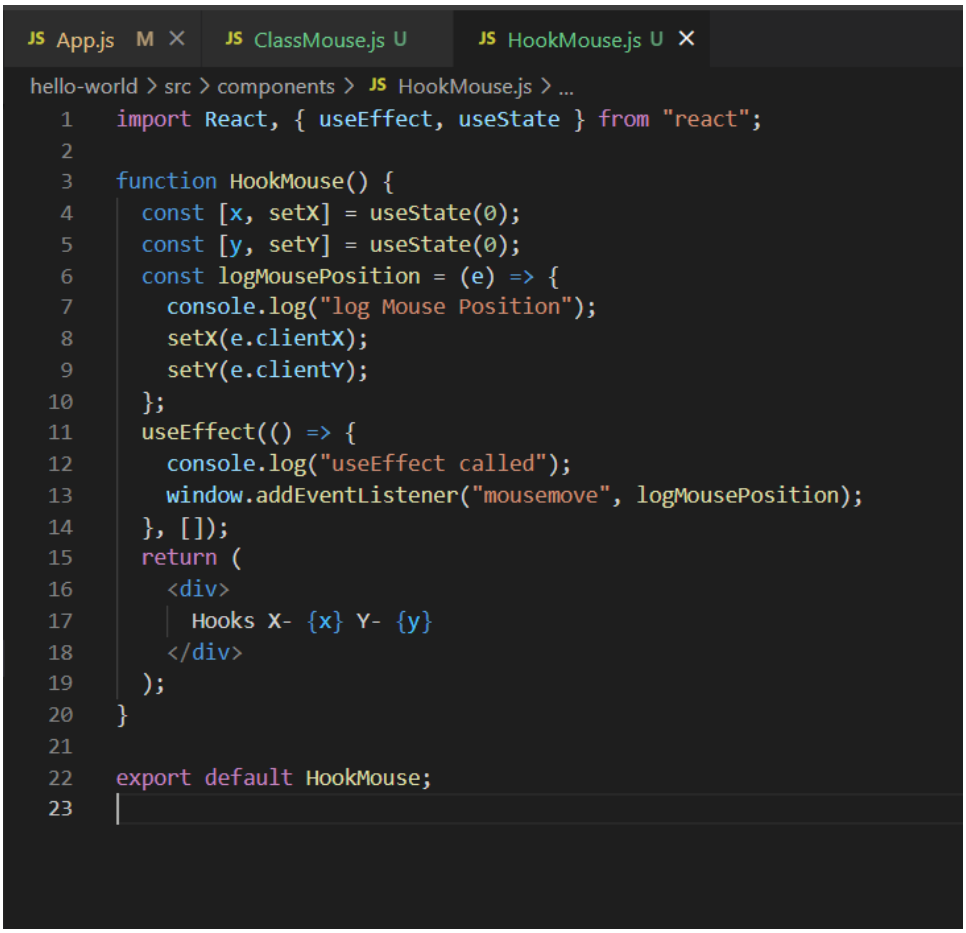
```
Component did mount called
X:  492 Y:  227
X:  797 Y:  171
X:  396 Y:  166
X:  201 Y:  182
>
```

App.js M ClassMouse.js U

hello-world > src > components > ClassMouse.js > ClassMouse > componentDidMount

```
1   import React, { Component } from "react";
2
3   class ClassMouse extends Component {
4     constructor(props) {
5       super(props);
6
7       this.state = {
8         x: 0,
9         y: 0,
10      };
11    }
12    logMousePosition = (e) => {
13      this.setState({ x: e.clientX, y: e.clientY });
14      console.log("X: ", this.state.x, "Y: ", this.state.y);
15    };
16    componentDidMount() {
17      console.log("Component did mount called");
18      window.addEventListener("mousemove", this.logMousePosition);
19    }
20
21    render() {
22      return (
23        <div>
24          X - {this.state.x} Y- {this.state.y}
25        </div>
26      );
27    }
28  }
29
30  export default ClassMouse;
31
```

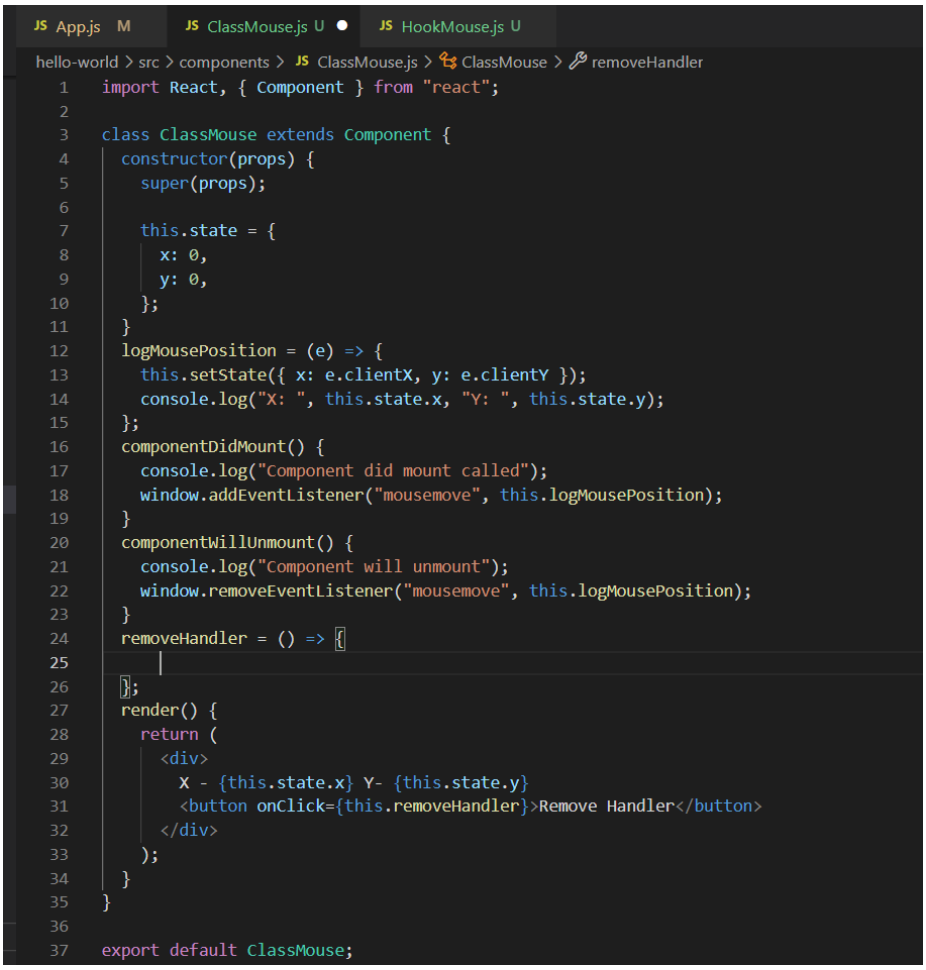
You can see the component did mount only called by ones. Let’s achieve the same functionality using the use effect.



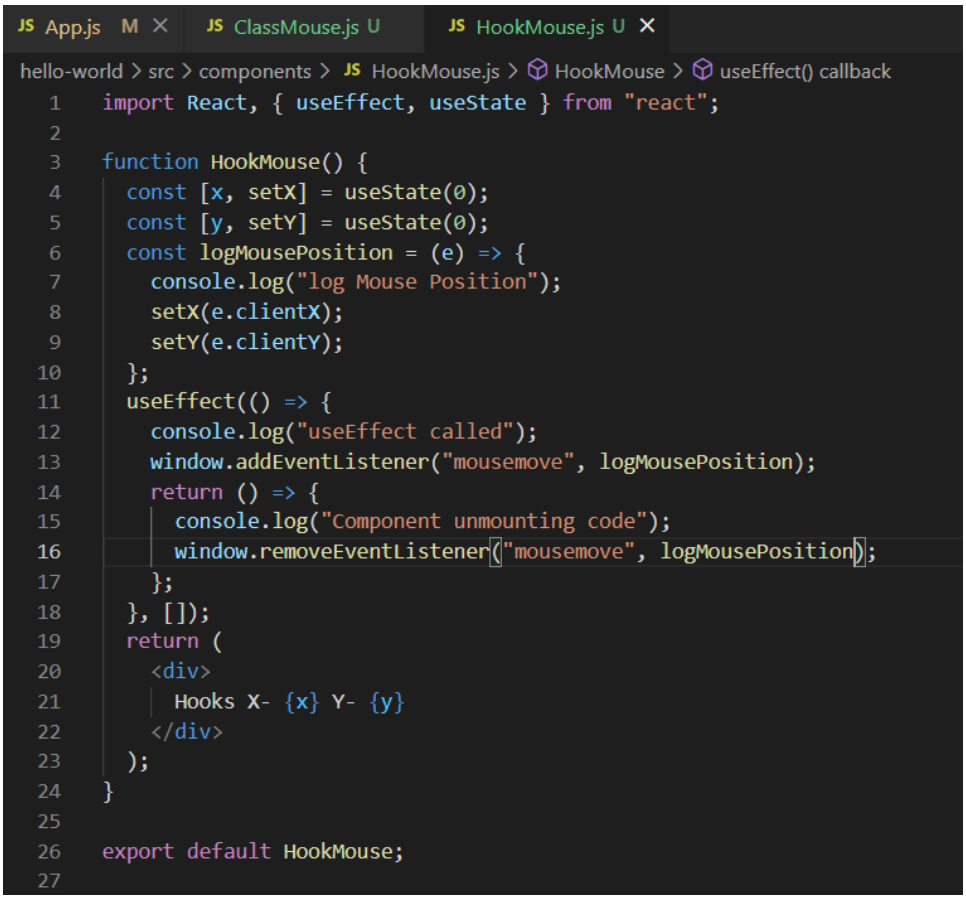
useEffect with cleanup

How to use component will unmount inside use effect. You use just return from the use effect it will serve a component did unmount.

It means when the component not an individual instead if the component is fully removed from the DOM, then the remove component method is automatically called.



And in case of hooks, we put the code in the use effect and we put inside the return method.



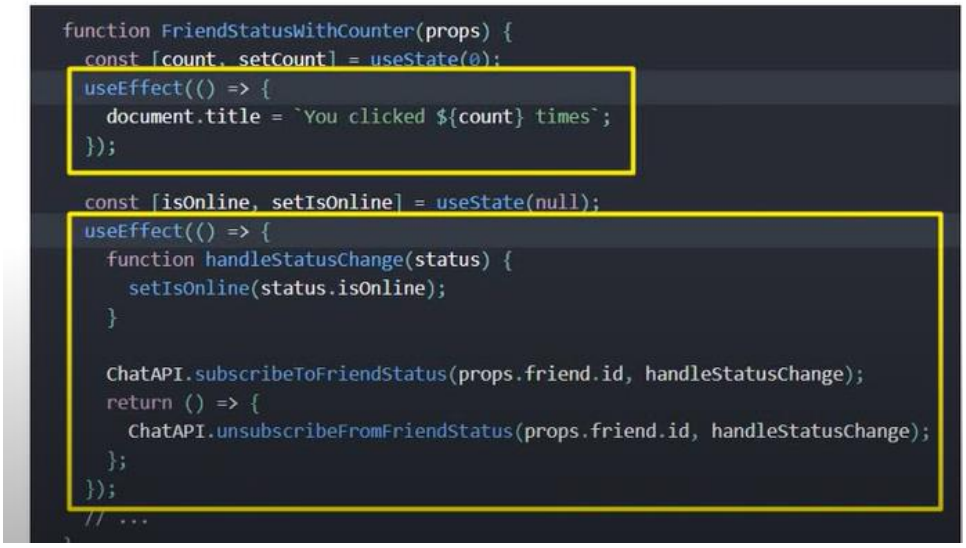
useEffect with incorrect dependency

Increment auto value by one with class component

Hook Component:

You can use the arrow function inside the tick method as it increments using the previous properties or you can put the value inside the [] where it continuously looking into the value update.

Multiple useEffects



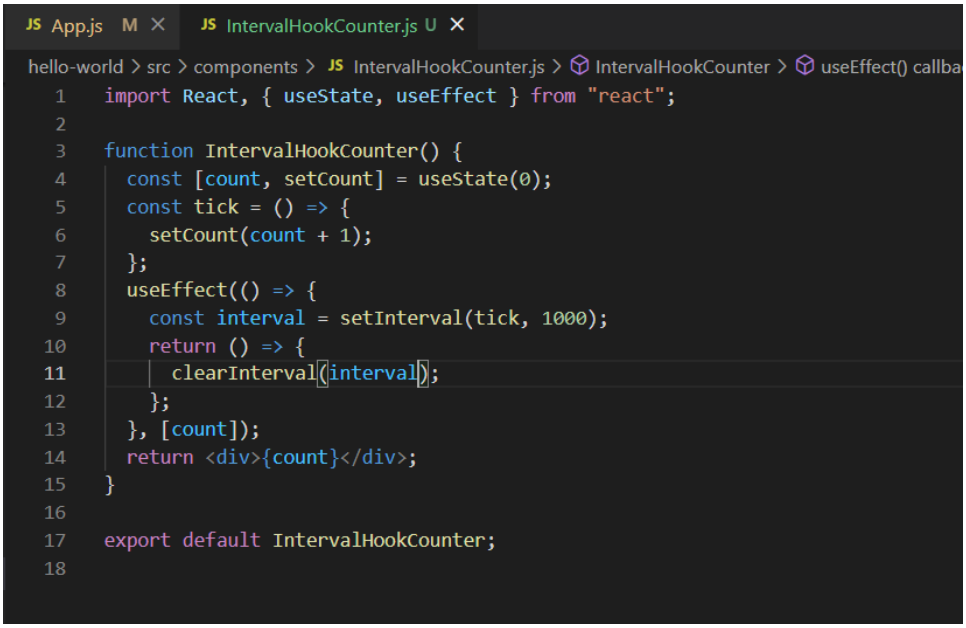
```
function FriendStatusWithCounter(props) {
  const [count, setCount] = useState(0);
  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  const [isOnline, setIsOnline] = useState(null);
  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }

    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });
  // ...
```

App.js M | IntervalClassCounter.js U

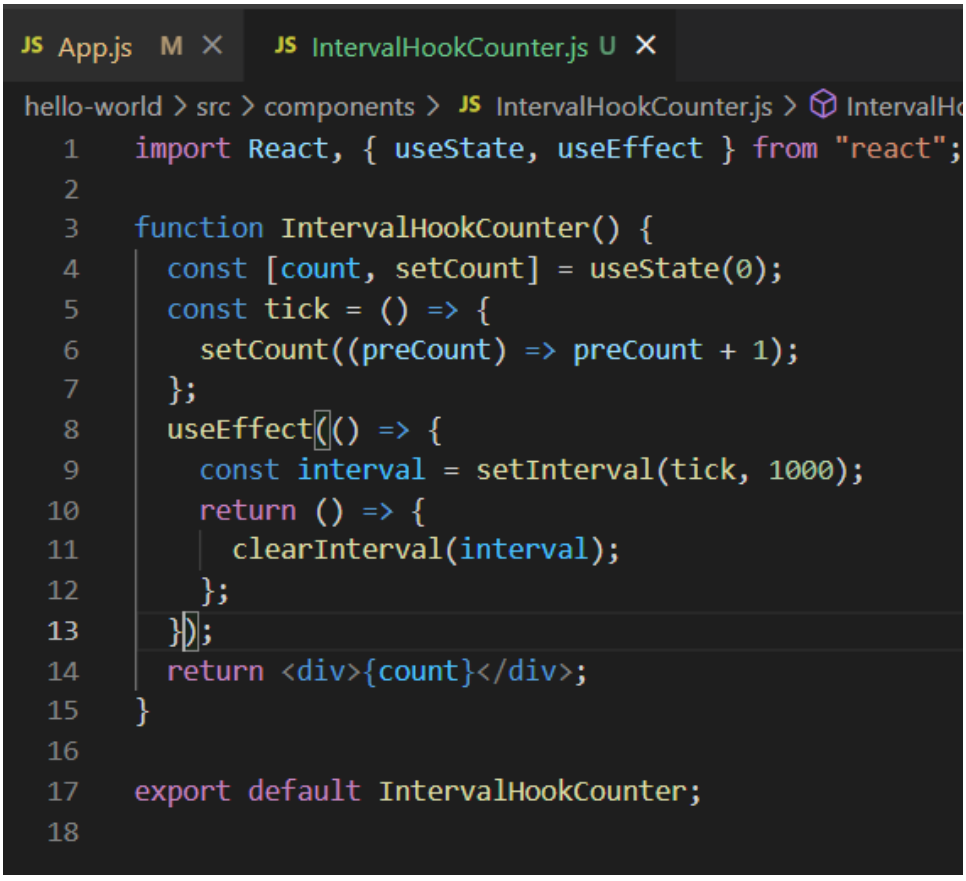
hello-world > src > components > IntervalClassCounter.js > default

```
1   import React, { Component } from "react";
2
3   export class IntervalClassCounter extends Component {
4     constructor(props) {
5       super(props);
6
7       this.state = {
8         count: 0,
9       };
10    }
11    componentDidMount() {
12      this.interval = setInterval(this.tick, 1000);
13    }
14    componentWillUnmount() {
15      clearInterval(this.interval);
16    }
17    tick = () => {
18      this.setState({ count: this.state.count + 1 });
19    };
20    render() {
21      return <div>{this.state.count}</div>;
22    }
23  }
24
25  export default IntervalClassCounter;
26
```

Now, let's replicate the above code using the Hooks



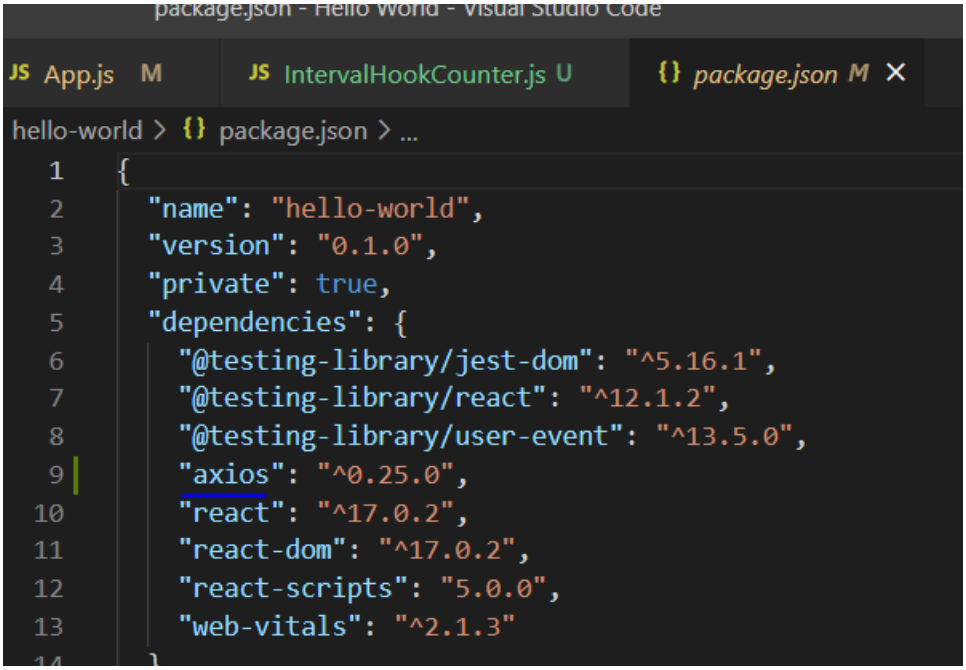
Or you can use previous counter and then increment.



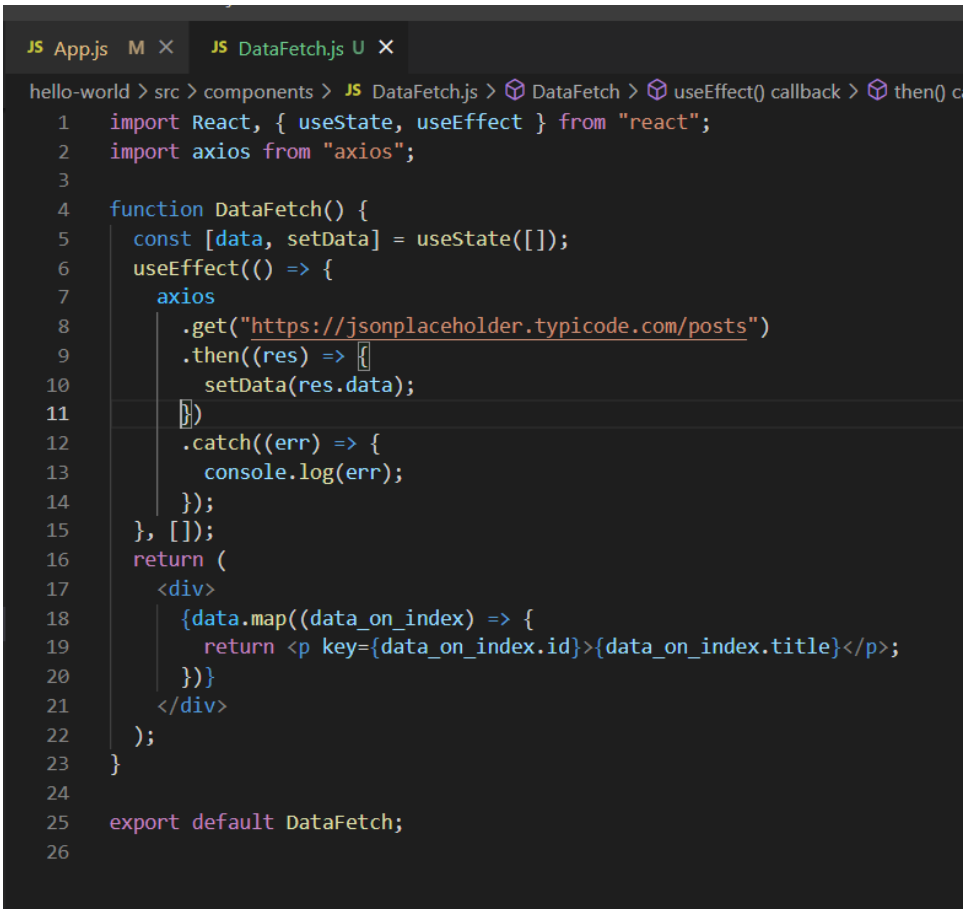
Fetching data with useEffect Part 1

When you enter

npm install axioms then it installed the package.



Empty dependency list **[]** specify that it will works as a document did mount.



Fetching data with useEffect Part 2

App.js M DataFetch.js U

hello-world > src > components > DataFetch.js > DataFetch

```
import React, { useState, useEffect } from "react";
import axios from "axios";

function DataFetch() {
  const [data, setData] = useState({});
  const [data_id, setDataId] = useState(2);
  useEffect(() => {
    axios
      .get(`https://jsonplaceholder.typicode.com/posts/${data_id}`)
      .then((res) => {
        setData(res.data);
      })
      .catch((err) => {
        console.log(err);
      });
  }, [data_id]);
  const changeHandler = (e) => {
    setDataId(e.target.value);
  };
  return (
    <div>
      <input onChange={changeHandler} value={data_id} />
      <br />
      {data.title}
    </div>
  );
}

export default DataFetch;
```

Fetching data with useEffect Part 3

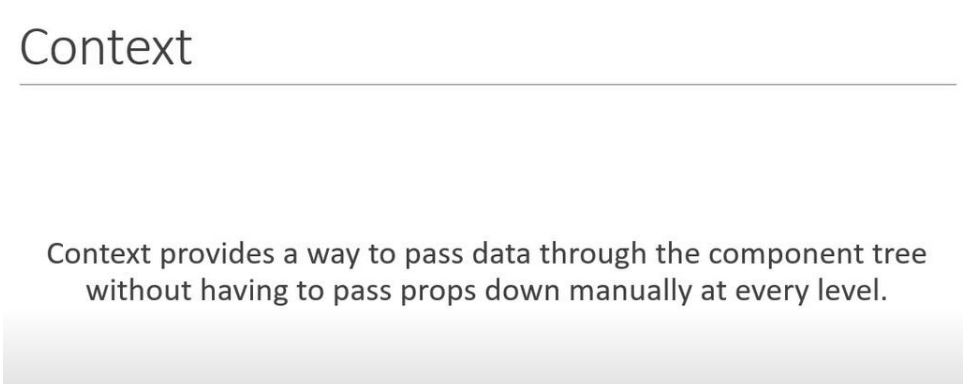
App.js M DataFetch.js U

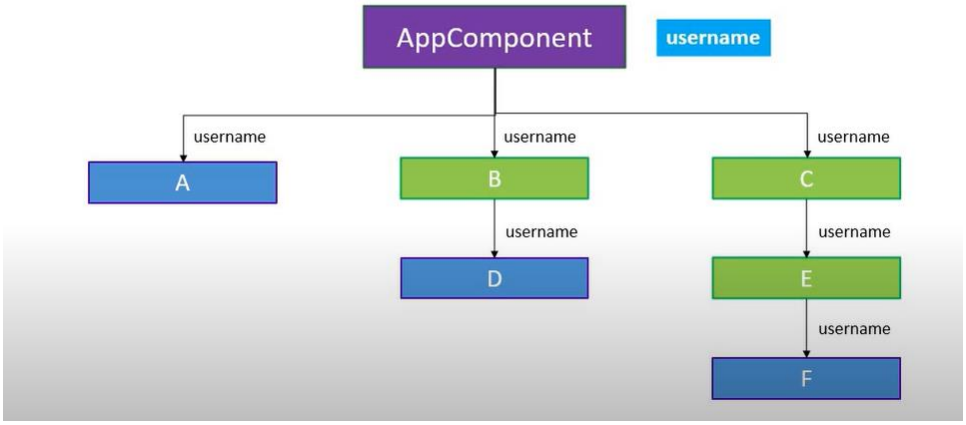
hello-world > src > components > DataFetch.js > DataFetch

```
import React, { useState, useEffect } from "react";
import axios from "axios";

function DataFetch() {
  const [data, setData] = useState({});
  const [data_id, setDataId] = useState(1);
  const [idFromButtonClick, setidFromButtonClick] = useState(1);
  useEffect(() => {
    axios
      .get(`https://jsonplaceholder.typicode.com/posts/${idFromButtonClick}`)
      .then((res) => {
        setData(res.data);
      })
      .catch((err) => {
        console.log(err);
      });
  }, [idFromButtonClick]);
  const changeHandler = (e) => {
    setDataId(e.target.value);
  };
  const ClickHandler = (e) => {
    setidFromButtonClick(data_id);
  };
  return (
    <div>
      <input onChange={changeHandler} value={data_id} />
      <button type="button" onClick={ClickHandler}>
        Fetch Post
      </button>
      <br />
      {data.title}
    </div>
  );
}

export default DataFetch;
```

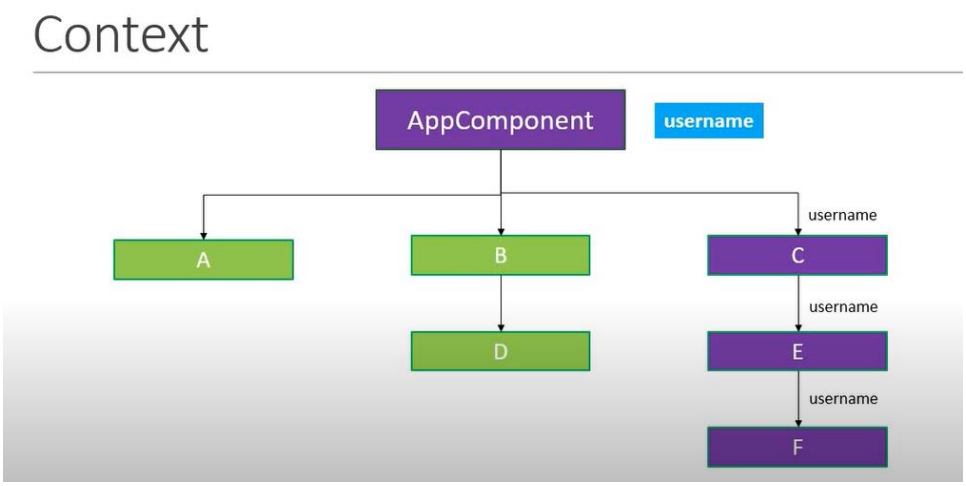
useContext Hook Part 1

If you want to pass the prop by not passing using the tree travel instead directly then you need to use context.





useContext Hook Part 2



First, we save the particular value inside a constant and then we wrap them inside the higher tree and then finally we include that component and then use that.

If you want to pass more than one value then you need to create another one and the process repeats.

There must be a better way to consume those.

App.js M | ComponentAA.js U | ComponentBB.js U | ComponentCC.js U

hello-world > src > App.js > ...

```
import logo from "./logo.svg";
import "./App.css";
import React from "react";
// import Inline from "./components/Inline";
import "./components/appStyles.css";
import ComponentAA from "./components/ComponentAA";
export const UserContext = React.createContext();

function App() {
  return (
    <div className="App">
      <UserContext.Provider value={"Muhammad"}>
        <ComponentAA />
      </UserContext.Provider>
```

App.js M | ComponentAA.js U | ComponentBB.js U | ComponentCC.js U

hello-world > src > components > ComponentAA.js > ...

```
import React from "react";
import ComponentBB from "./ComponentBB";
function ComponentAA() {
  return (
    <div>
      <ComponentBB />
    </div>
  );
}

export default ComponentAA;
```

App.js M | ComponentAA.js U | ComponentBB.js U | ComponentCC.js U

hello-world > src > components > ComponentBB.js > ...

```
import React from "react";
import ComponentCC from "./ComponentCC";
function ComponentBB() {
  return (
    <div>
      <ComponentCC />
    </div>
  );
}

export default ComponentBB;
```

App.js M | ComponentAA.js U | ComponentBB.js U | ComponentCC.js U

hello-world > src > components > ComponentCC.js > ...

```
1   import React from "react";
2   // import { UserContext } from "../App";
3   import { UserContext } from "../App.js";
4   function ComponentCC() {
5     return (
6       <div>
7         {
8           <UserContext.Consumer>
9             {(value) => {
10              return <div>{value}</div>;
11            }}
12          </UserContext.Consumer>
13        }
14      </div>
15    );
16  }
17
18  export default ComponentCC;
19
```

Let's pass 2nd value and see what happens.

App.js M | ComponentAA.js U | ComponentBB.js U | ComponentCC.js U

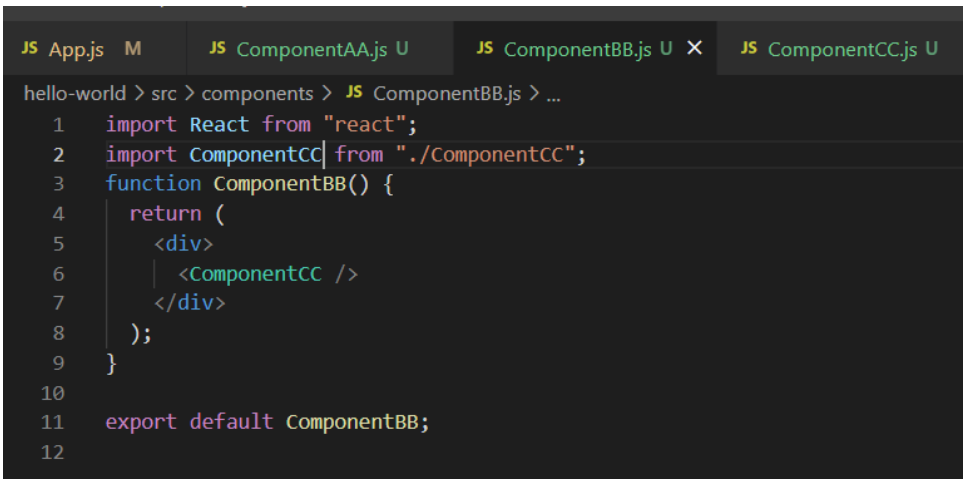
hello-world > src > App.js > App

```
1   import logo from "./logo.svg";
2   import "./App.css";
3   import React from "react";
4   // import Inline from "./components/Inline";
5   import "./components/appStyles.css";
6   import ComponentAA from "./components/ComponentAA";
7   export const UserContext = React.createContext();
8   export const ChannelContext = React.createContext();
9   function App() {
10    return (
11      <div className="App">
12        <UserContext.Provider value={"Muhammad"}>
13          <ChannelContext.Provider value={"Ahmed"}>
14            <ComponentAA />
15          </ChannelContext.Provider>
16        </UserContext.Provider>
```


App.js M
ComponentAA.js U
ComponentBB.js U
ComponentCC.js U
hello-world > src > components > ComponentAA.js > ...
import React from "react";
import ComponentBB from "./ComponentBB";
function ComponentAA() {
return (

<ComponentBB />

);
}
export default ComponentAA;


App.js M
ComponentAA.js U
ComponentBB.js U
ComponentCC.js U
hello-world > src > components > ComponentBB.js > ...
import React from "react";
import ComponentCC from "./ComponentCC";
function ComponentBB() {
return (

<ComponentCC />

);
}
export default ComponentBB;

App.js M | ComponentAA.js U | ComponentBB.js U | ComponentCC.js U

hello-world > src > components > ComponentCC.js > ComponentCC

```
import React from "react";
// import { UserContext } from "../App";
import { UserContext, ChannelContext } from "../App.js";
function ComponentCC() {
  return (
    <div>
      {
        <UserContext.Consumer>
          {(value) => {
            return (
              <ChannelContext.Consumer>
                {(value_2) => {
                  return (
                    <div>
                      Value: {value} Value_2: {value_2}
                    </div>
                  );
                }}
              </ChannelContext.Consumer>
            );
          }}
        </UserContext.Consumer>
      }
    </div>
  );
}

export default ComponentCC;

```

useContext Hook Part 3

Let's make our code cleaner.

You can use them by calling the **use Context** from the react library and then rap them inside this.

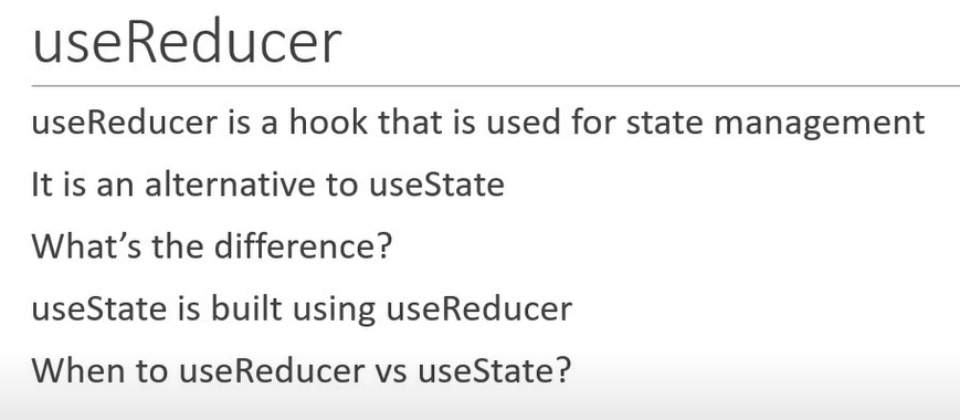
App.js M | ComponentAA.js U | ComponentBB.js U | ComponentCC.js U

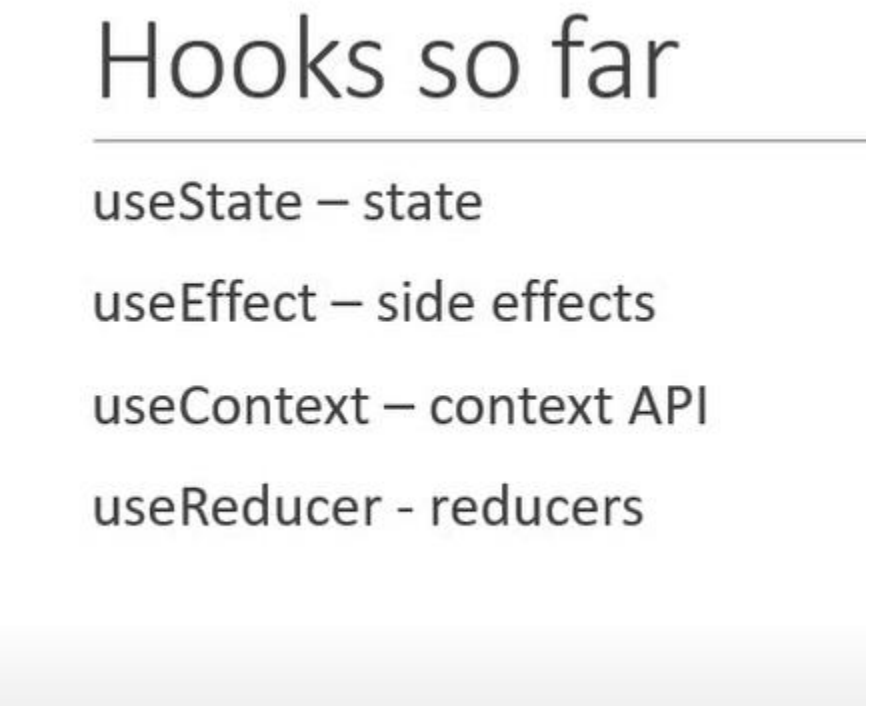
hello-world > src > components > ComponentBB.js > ComponentBB

```
import React, { useContext } from "react";
import ComponentCC from "./ComponentCC";
import { UserContext, ChannelContext } from "../App";
function ComponentBB() {
  const value_1 = useContext(UserContext);
  const value_2 = useContext(ChannelContext);
  return (
    <div>
      <ComponentCC />
      {value_1}
      {value_2}
    </div>
  );
}

export default ComponentBB;

```

This is the more generic and good way of consuming the use Context variables.

useReducer Hook





JavaScript Demo: Array.reduce()

```
1  const array1 = [1, 2, 3, 4];
2  const reducer = (accumulator, currentValue) => accumulator + currentValue;
3
4  // 1 + 2 + 3 + 4
5  console.log(array1.reduce(reducer));
6  // expected output: 10
7
8  // 5 + 1 + 2 + 3 + 4
9  console.log(array1.reduce(reducer, 5));
10 // expected output: 15
11
```

It takes two parameters inside the function; the first parameter is the function and the second parameter is for the initial value.

reduce vs useReducer

reduce in JavaScript	useReducer in React
array.**reduce**(***reducer***, initialValue)	**useReducer**(***reducer***, initialState)
singleValue = ***reducer***(accumulator, itemValue)	newState = ***reducer***(currentState , action)
reduce method returns a single value	**useReducer**returns a pair of values. [newState , dispatch]

useReducer Summary

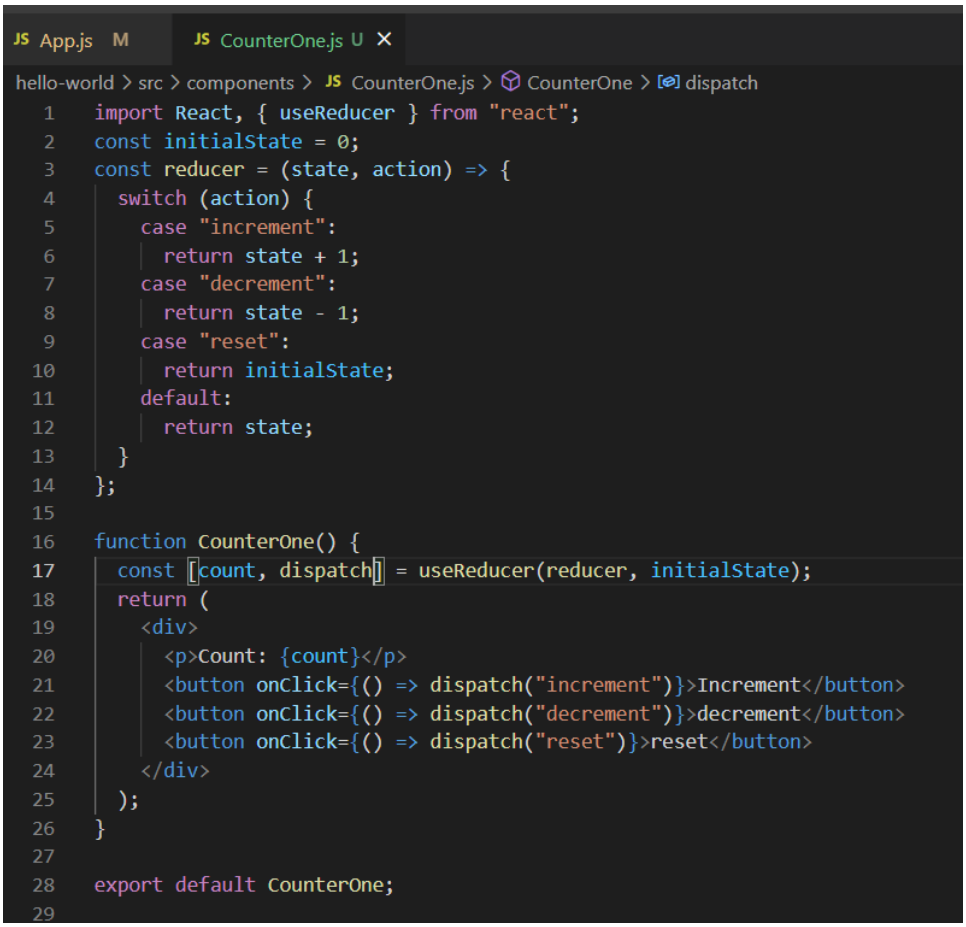
useReducer is a hook that is used for state management in React

useReducer is related to reducer functions

useReducer(reducer, initialState)

reducer(currentState, action)

useReducer (simple state & action)



Let me explain to the above code what's going on there. **Use Reducer** function takes two inputs: the first one is the **function** and the second one is the **initial state**. Initial state is because it needs to start a starting value and the function needs what action is to be performed and also the variable used inside the function tracks the output because the value is saved in the return value so that's all for the use reducer according to this example.

useReducer (complex state & action)