

5

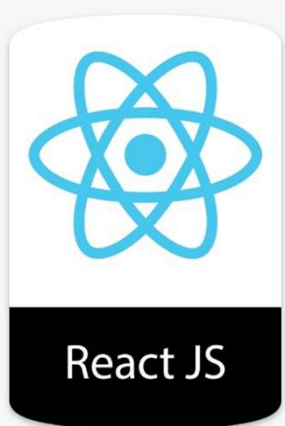
Component Types

Objectives

After completing this lesson, you should be able to do the following:

- Creating Components
- Basic Component
- Nesting Components
- Props





Introduction to Stateless & Statefull Components

- A **stateless** component as it does not contain state (in the React sense of the word).
- In such a case, some people find it preferable to use Stateless Functional Components, which are based on ES6
- arrow functions.

EXPLORER

REACT

components-demo

node_modules

public

src

App.css

App.js

App.test.js

FirstComponent.js

index.css

index.js

logo.svg

reportWebVitals.js

setupTests.js

.gitignore

package-lock.json

package.json

README.md

JS FirstComponent.js U X

JS App.js M

JS index.js

components-demo > src > JS FirstComponent.js > ...

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

PROBLEMS

DEBUG CONSOLE

OUTPUT

TERMINAL

node + ^ x

Note that the development build is not optimized.
To create a production build, use `npm run build`.

webpack compiled **successfully**

> OUTLINE

> TIMELINE

The image shows a development environment with VS Code and a web browser. In VS Code, the Explorer sidebar on the left shows a project structure under 'REACT' with a folder 'components-demo'. Inside 'components-demo', there is a 'src' folder containing 'App.css', 'App.js' (selected), 'App.test.js', 'FirstComponent.js', 'index.css', 'index.js', 'logo.svg', 'reportWebVitals.js', and 'setupTests.js'. The main editor displays 'App.js' with the following code:

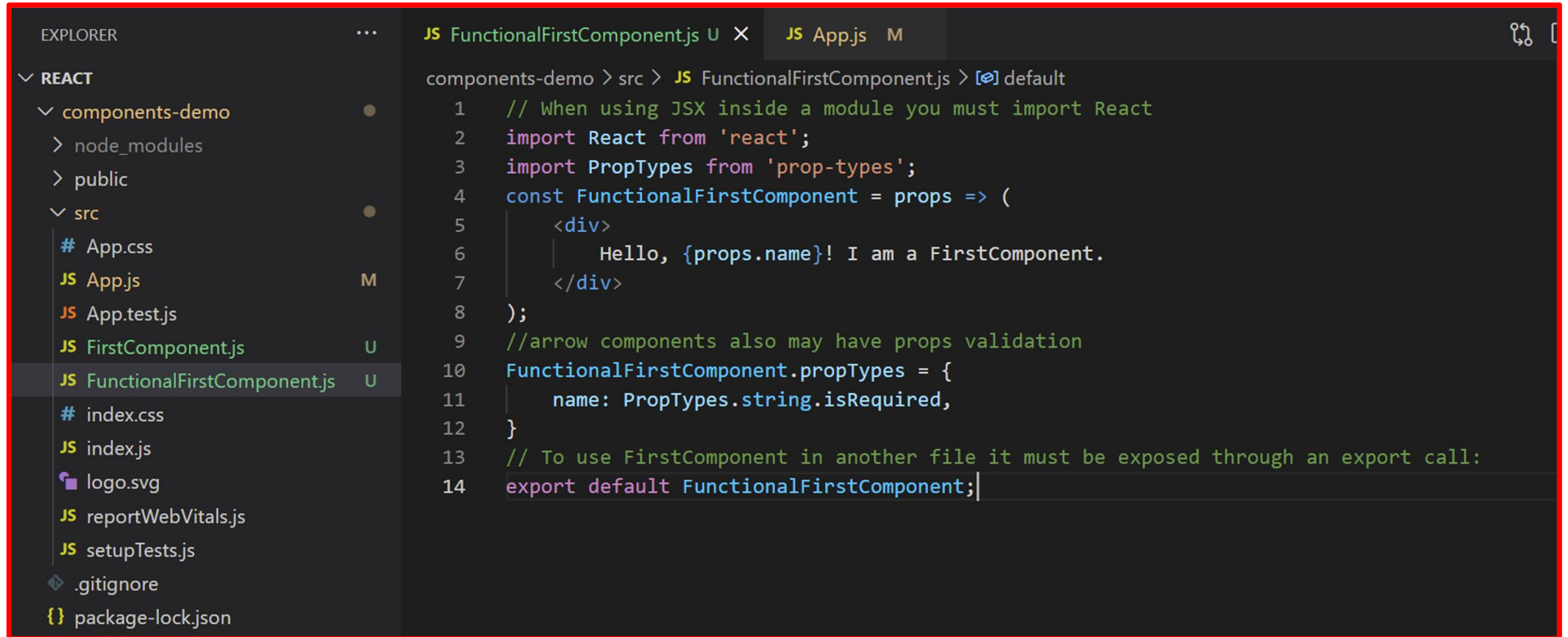
```
components-demo > src > JS App.js > ...
1  import './App.css';
2  import FirstComponent from './FirstComponent';
3
4  function App() {
5    return (
6      <div className="App" id="content">
7        |
8        |   <FirstComponent name={'User'} />
9        |
10     </div>
11   );
12 }
13
14 export default App;
```

Below the code editor, a web browser window titled 'React App' is open at 'localhost:3000'. The browser's address bar and tabs are visible. The page content displays the text: 'Hello, User! I am a FirstComponent.'

Stateless Functional Components

- In many applications there are smart components that hold state but render dumb components that simply receive
- props and return HTML as JSX. Stateless functional components are much more reusable and have a positive
- performance impact on your application.
- They have 2 main characteristics:
 - 1. When rendered they receive an object with all the props that were passed down
 - 2. They must return the JSX to be rendered

Stateless Functional Components



The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar showing a project structure for 'components-demo'. The 'src' folder is expanded, listing files like App.css, App.js, App.test.js, FirstComponent.js, FunctionalFirstComponent.js, index.css, index.js, logo.svg, reportWebVitals.js, setupTests.js, .gitignore, and package-lock.json. The file 'FunctionalFirstComponent.js' is selected and highlighted. The main editor area shows the code for 'FunctionalFirstComponent.js'. The code is as follows:

```
components-demo > src > JS FunctionalFirstComponent.js > [🔍] default
1  // When using JSX inside a module you must import React
2  import React from 'react';
3  import PropTypes from 'prop-types';
4  const FunctionalFirstComponent = props => (
5    <div>
6      Hello, {props.name}! I am a FirstComponent.
7    </div>
8  );
9  //arrow components also may have props validation
10 FunctionalFirstComponent.propTypes = {
11   name: PropTypes.string.isRequired,
12 }
13 // To use FirstComponent in another file it must be exposed through an export call:
14 export default FunctionalFirstComponent;
```


EXPLORER

▼ REACT

▼ components-demo

> node_modules

> public

▼ src

App.css

JS App.js M

JS App.test.js

JS FirstComponent.js U

JS FunctionalFirstComponent.js U

index.css

JS index.js

logo.svg

JS reportWebVitals.js

JS setupTests.js

.gitignore

{ } package-lock.json

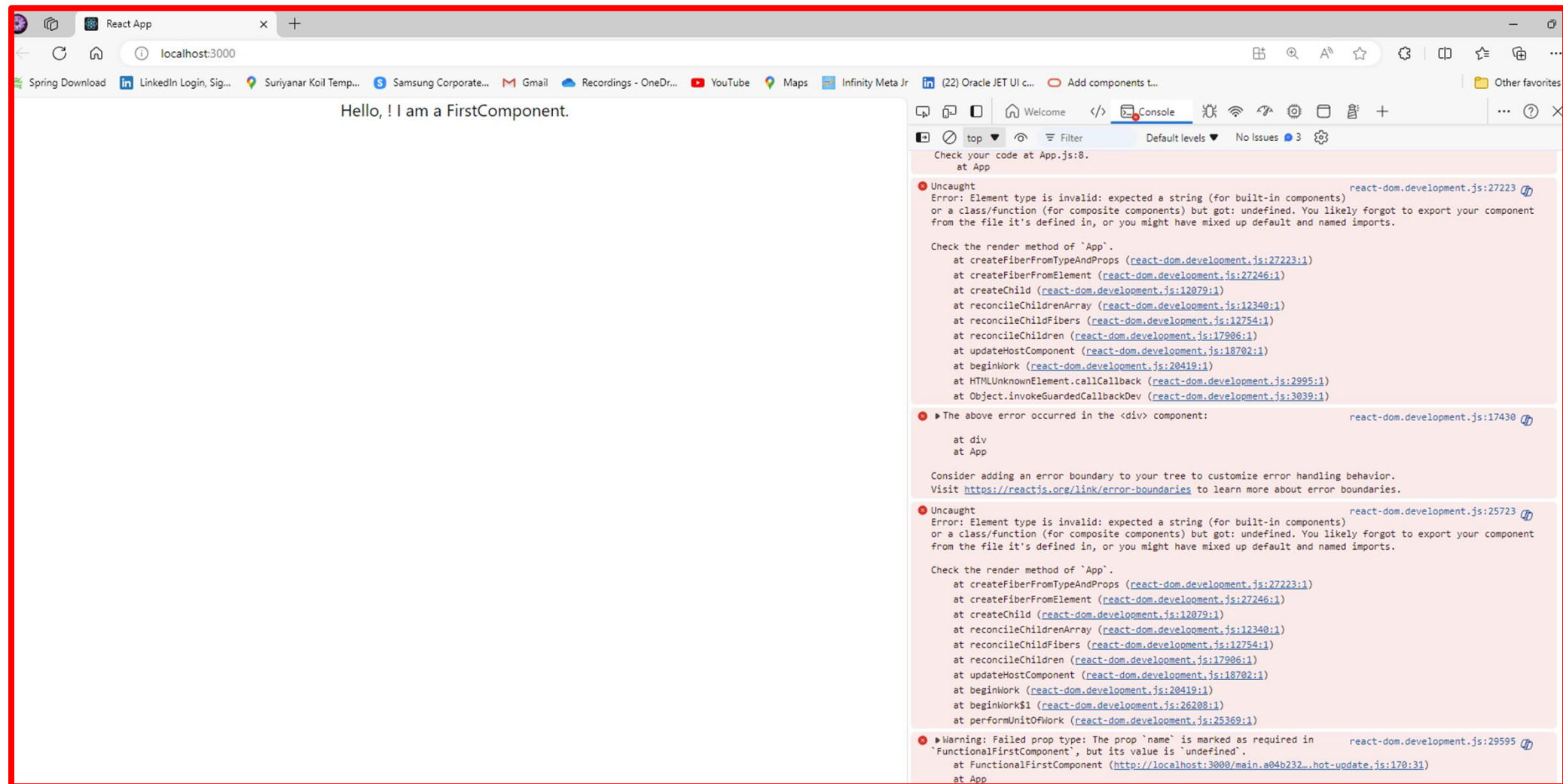
JS FunctionalFirstComponent.js U

JS App.js M X

components-demo > src > JS App.js > App

```
1 import './App.css';
2 import FirstComponent from './FirstComponent';
3 import FunctionalFirstComponent from './FunctionalFirstComponent';
4
5 function App() {
6   return (
7     <div className="App" id="content">
8
9       { /* <FirstComponent name={'User'} /> */ }
10
11     <FunctionalFirstComponent />
12   </div>
13 );
14 }
15
16 export default App;
17
```

Output With Validation Error



Stateful Components

- In contrast to the 'stateless' components shown above, 'stateful' components have a state object that can be
- updated with the `setState` method. The state must be initialized in the constructor before it can be set:

EXPLORER

REACT

components-demo

- node_modules
- public
- src
 - components
 - JS FirstComponent.js U
 - JS FunctionalFirstComponent.js U
 - JS SecondComponent.js U

App.css

JS App.js M

JS App.test.js

index.css

JS index.js

logo.svg

JS reportWebVitals.js

JS setupTests.js

.gitignore

{ } package-lock.json

{ } package.json

README.md

OUTLINE

TIMELINE

JS App.js M

JS SecondComponent.js U X

components-demo > src > components > JS SecondComponent.js > SecondComponent > onClick

```
1 import React from 'react';
2 class SecondComponent extends React.Component {
3   constructor(props) {
4     super(props);
5     this.state = {
6       toggle: true
7     };
8     // This is to bind context when passing onClick as a callback
9     this.onClick = this.onClick.bind(this);
10  }
11  onClick() {
12    this.setState((prevState, props) => ({
13      toggle: !prevState.toggle
14    }));
15  }
16  render() {
17    return (
18      <div onClick={this.onClick}>
19        Hello, {this.props.name}! I am a SecondComponent.
20        <br />
21        Toggle is: {this.state.toggle}
22      </div>
23    );
24  }
25 }
26 export default SecondComponent;
```

12 of 8

Topic: Component Types

MENTORLABSSM

Higher Order Components

- Higher order components (HOC) allow to share component functionality.
- Higher order components are used when you want to share logic across several components regardless of how different they render.

EXPLORER

... JS App.js M JS HighOrderComponents.js U X

▼ REACT

▼ components-demo

> node_modules

> public

▼ src

▼ components

JS FirstComponent.js U

JS FunctionalFirstComponent.js U

JS HighOrderComponents.js U

JS SecondComponent.js U

App.css

JS App.js M

JS App.test.js

index.css

JS index.js

logo.svg

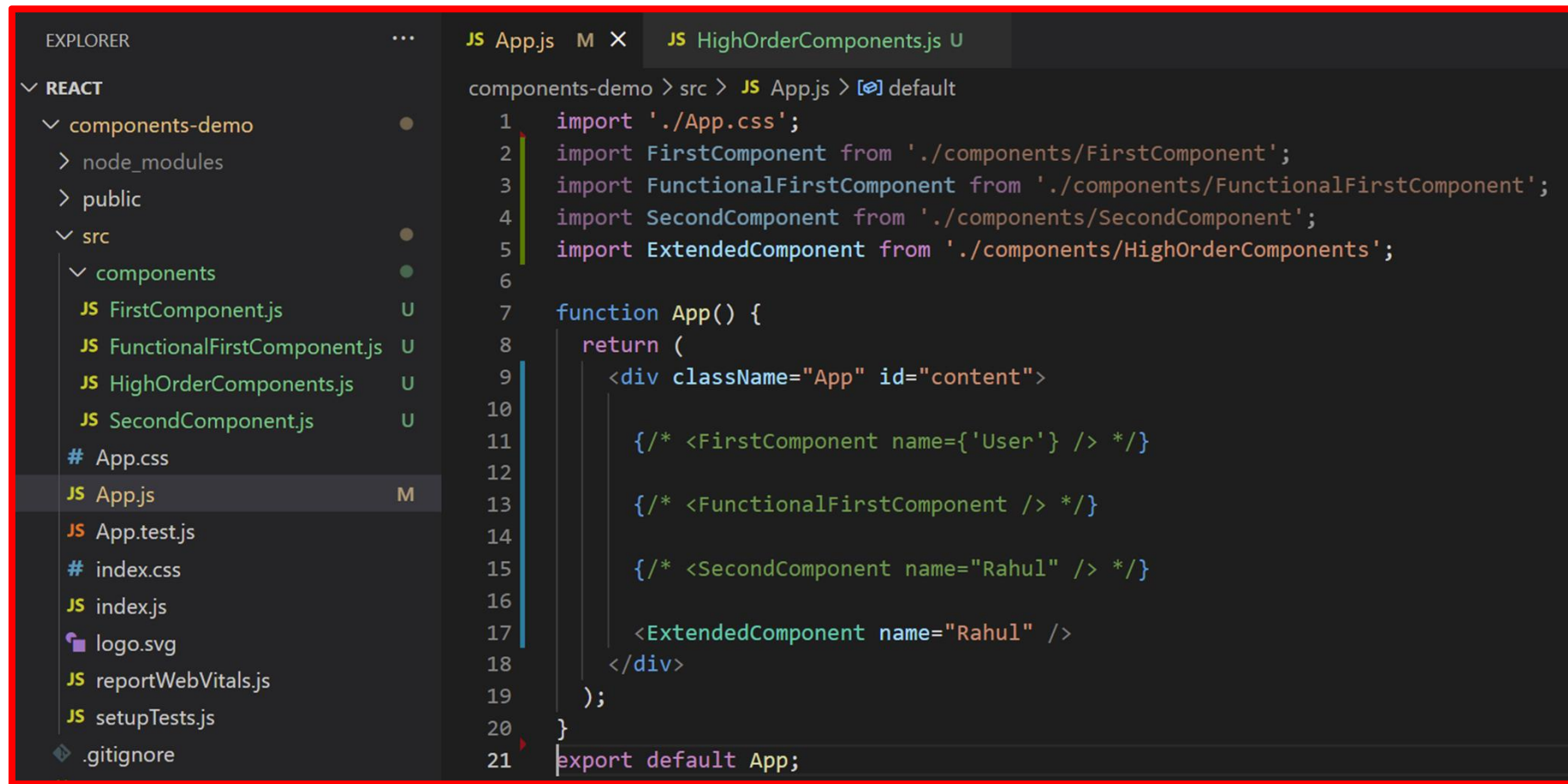
JS reportWebVitals.js

JS setupTests.js

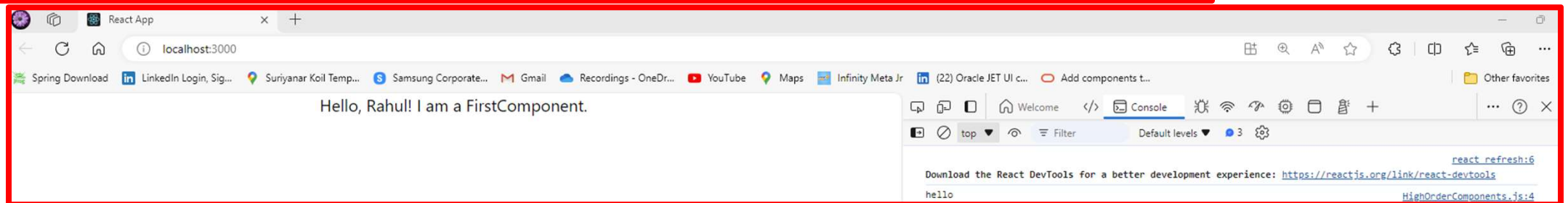
.gitignore

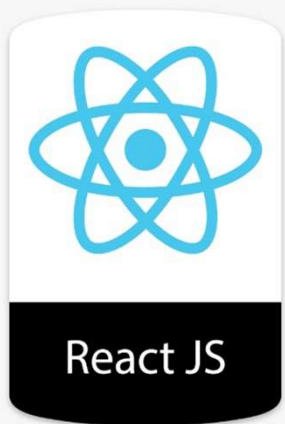
components-demo > src > components > JS HighOrderComponents.js > [🔗] default

```
1 import React, { Component } from 'react';
2 const PrintHello = ComposedComponent => class extends Component {
3   onClick() {
4     console.log('hello');
5   }
6   /* The higher order component takes another component as a parameter
7    and then renders it with additional props */
8   render() {
9     return <ComposedComponent {...this.props} onClick={this.onClick} />
10  }
11 }
12
13 const FirstComponent = props => (
14   <div onClick={props.onClick}>
15     Hello, {props.name}! I am a FirstComponent.
16   </div>
17 );
18 const ExtendedComponent = PrintHello(FirstComponent);
19
20 export default ExtendedComponent;
```

```
1 import './App.css';
2 import FirstComponent from './components/FirstComponent';
3 import FunctionalFirstComponent from './components/FunctionalFirstComponent';
4 import SecondComponent from './components/SecondComponent';
5 import ExtendedComponent from './components/HighOrderComponents';
6
7 function App() {
8   return (
9     <div className="App" id="content">
10
11       {/* <FirstComponent name={ 'User' } /> */}
12
13       {/* <FunctionalFirstComponent /> */}
14
15       {/* <SecondComponent name="Rahul" /> */}
16
17       <ExtendedComponent name="Rahul" />
18     </div>
19   );
20 }
21 export default App;
```





Nesting Components

Nesting Components

- A lot of the power of ReactJS is its ability to allow nesting of components.

```
F:\Training\IQ Gateway\2024\Workspaces\React\components-demo>npm i create-react-class
```

EXPLORER

REACT

components-demo

node_modules

public

src

components

nesting-components

NestingCompsDemo01.js

App.css

App.js

App.test.js

index.css

index.js

logo.svg

reportWebVitals.js

setupTests.js

.gitignore

package-lock.json

package.json

JS App.js M

JS NestingCompsDemo01.js U X

components-demo > src > nesting-components > JS NestingCompsDemo01.js > ...

```
1  var React = require('react');
2  var createReactClass = require('create-react-class');
3  var CommentList = createReactClass({
4    render: function () {
5      return (
6        <div className="commentList">
7          Hello, world! I am a CommentList.
8        </div>
9      );
10   }
11 });
12 var CommentForm = createReactClass({
13   render: function () {
14     return (
15       <div className="commentForm">
16         Hello, world! I am a CommentForm.
17       </div>
18     );
19   }
20 });
21
```

18 of 8

Topic: Component Types

MENTORLABSSM

EXPLORER

▼ REACT

▼ components-demo

> node_modules

> public

▼ src

> components

▼ nesting-components

JS NestingCompsDemo01.js U

App.css

JS App.js M

JS App.test.js

index.css

JS index.js

logo.svg

JS reportWebVitals.js

JS setupTests.js

JS App.js M

JS NestingCompsDemo01.js U X

components-demo > src > nesting-components > JS NestingCompsDemo01.js > ...

```
21
22 var React = require('react');
23 var createReactClass = require('create-react-class');
24 var CommentBox = createReactClass({
25   render: function () {
26     return (
27       <div className="commentBox">
28         <h1>Comments</h1>
29         <CommentList />
30         <CommentForm />
31       </div>
32     );
33   }
34 });
35
36 export default CommentBox;
```

19 of 8

Topic: Component Types

MENTORLABSSM

EXPLORER

...

JS App.js M X JS NestingCompsDemo01.js U

components-demo > src > JS App.js > App

6 import CommentBox from './nesting-components/NestingCompsDemo01';

7

8 function App() {

9 return (

10 <div className="App" id="content">

11

12 /* <FirstComponent name={'User'} /> */

13

14 /* <FunctionalFirstComponent /> */

15

16 /* <SecondComponent name="Rahul" /> */

17

18 /* <ExtendedComponent name="Rahul" /> */

19 <CommentBox />

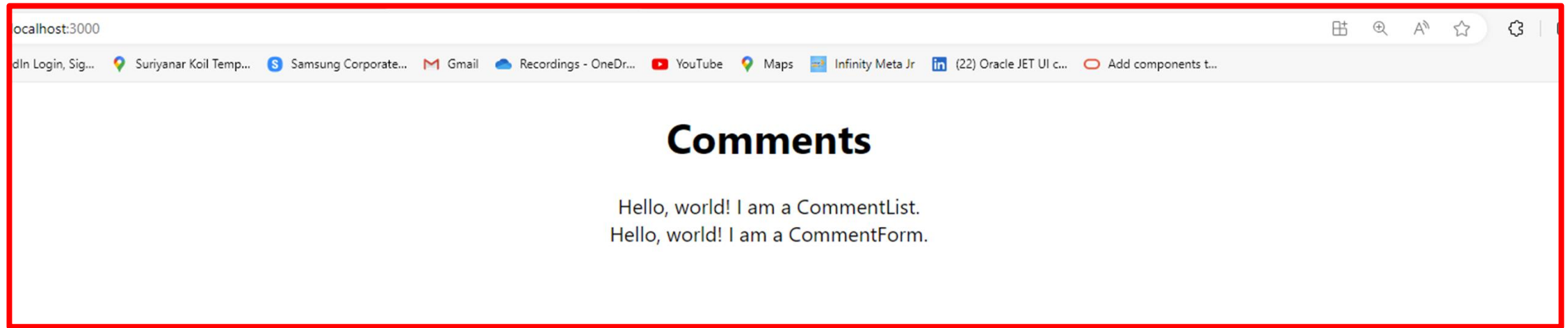
20 </div>

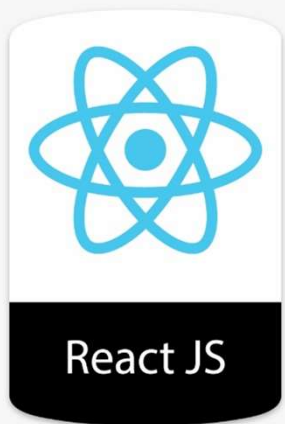
21);

22 }

23 export default App;

24





Props

Props

- Props are a way to pass information into a React component, they can have any type including functions -
- sometimes referred to as callbacks.
- In JSX props are passed with the attribute syntax

```
<MyComponent userID={123} />
```

Inside the definition for MyComponent userID will now be accessible from the props object

```
// The render function inside MyComponent  
render() {  
  return (  
    <span>The user's ID is {this.props.userID}</span>  
  )  
}
```


- It's important to define all props, their types, and where applicable, their default value:

```
// defined at the bottom of MyComponent  
MyComponent.propTypes = {  
  someObject: React.PropTypes.object,  
  userID: React.PropTypes.number.isRequired,  
  title: React.PropTypes.string  
};  
  
MyComponent.defaultProps = {  
  someObject: {},  
  title: 'My Default Title'  
}
```

In this example the prop `someObject` is optional, but the prop `userID` is required. If you fail to provide `userID` to `MyComponent`, at runtime the React engine will show a console warning you that the required prop was not provided. Beware though, this warning is only shown in the development version of the React library, the production version will not log any warnings.

Summary

In this lesson, you should have learned how to:

- Creating Components
- Basic Component
- Nesting Components
- Props

