

# 5

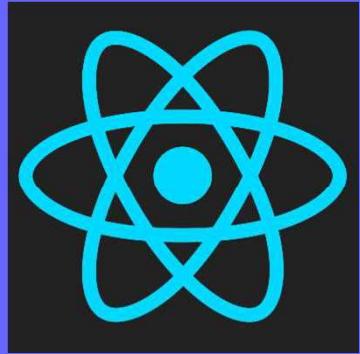
## Component Types

# Objectives

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

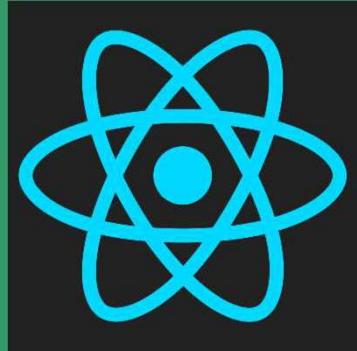
- Creating Components
- Basic Component
- Nesting Components
- Props





# Introduction to Stateless & Statefull Components

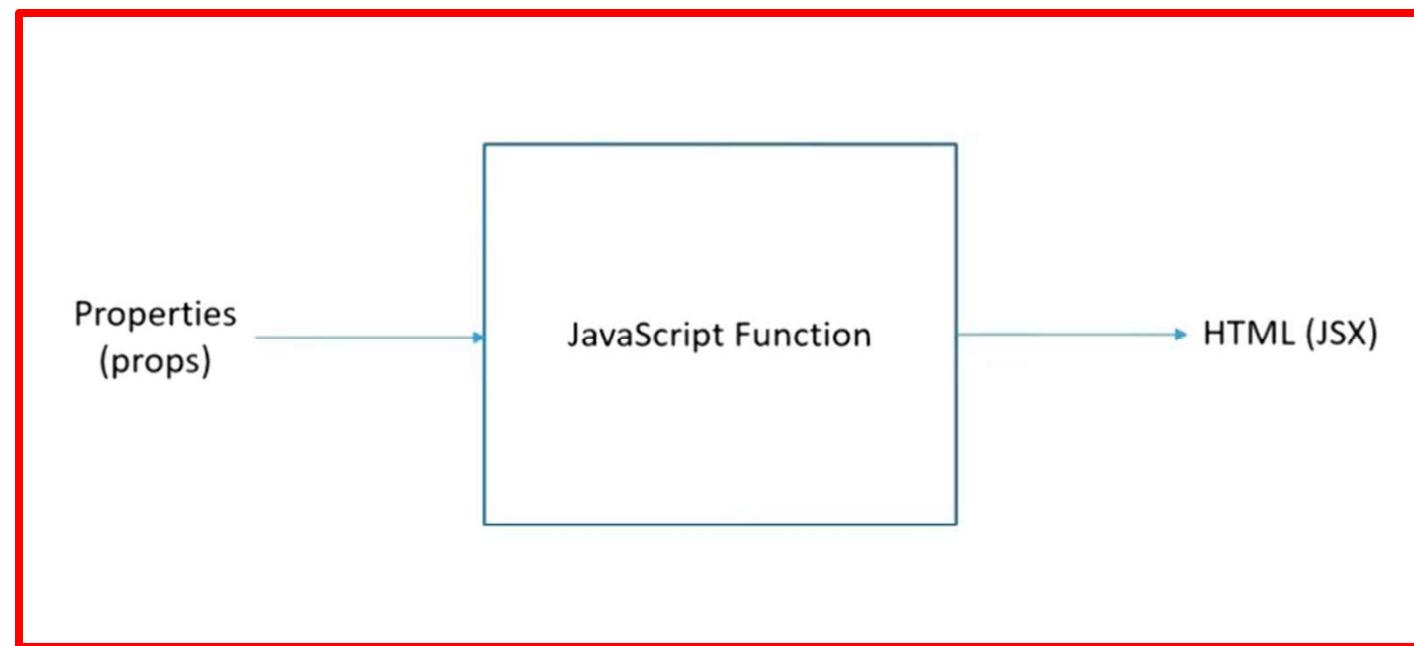
- A **stateless** component as it does not contain state (in the React sense of the word).
- In such a case, some people find it preferable to use Stateless Functional Components, which are based on ES6
- Arrow functions.



# The Functional Components

# Functional Components

- Functional Components are Just JS Functions
- They can Optionally receive object of Properties Which is referred to as **props**
- **Returns HTML as the UI**
- The Functional component is a JavaScript function that accepts an input of properties and returns HTML that describes the UI



The screenshot shows a development environment with two code editors and a browser window.

**Code Editors:**

- Left Editor (JS Greet.js):**

```
my-app > src > components > JS Greet.js > [o] default
1 import React from 'react';
2
3 function Greet() {
4     return <h1>Hello React !</h1>
5 }
6
7 export default Greet;
```
- Right Editor (JS App.js):**

```
my-app > src > JS App.js > ...
1 import logo from './logo.svg';
2 import './App.css';
3 import Greet from './components/Greet';
4
5 function App() {
6     return (
7         <div className="App">
8             <Greet />
9         </div>
10    );
11 }
12
13 export default App;
```

**Browser Output:**

localhost:3000

Hello React !

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

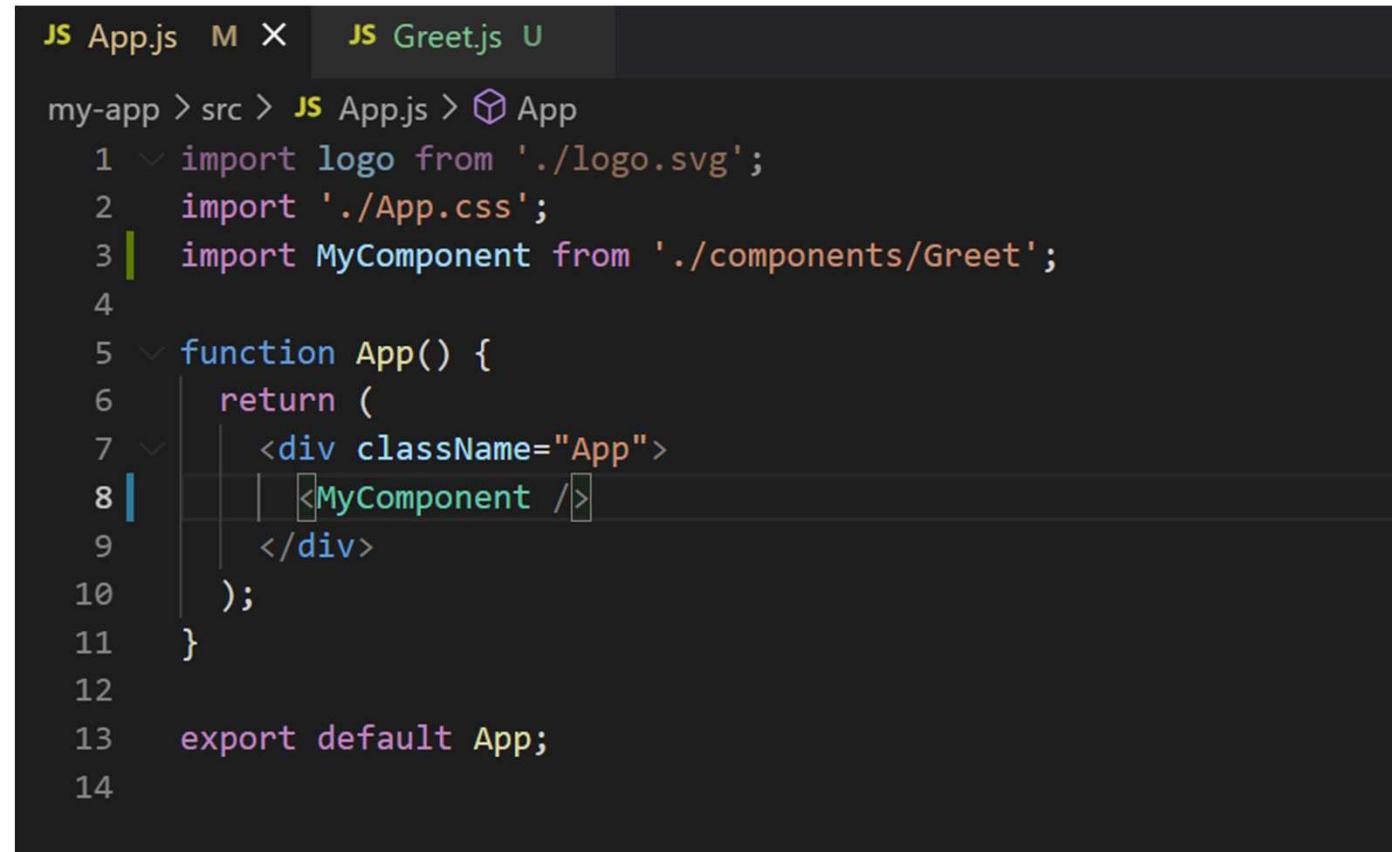
```
my-app > src > JS App.js > ...
1 import logo from './logo.svg';
2 import './App.css';
3 import Greet from './components/Greet';
4
5 function App() {
6     return (
7         <div className="App">
8             <Greet />
9         </div>
10    );
11 }
12
13 export default App;
```

## But Convention is to Use Arrow Functions

```
JS App.js M JS Greet.js U X  
my-app > src > components > JS Greet.js > [o] Greet  
1 import React from 'react';  
2  
3 // function Greet() {  
4 //     return <h1>Hello React !</h1>  
5 // }  
6  
7 const Greet = () => <h1> Hello React !</h1>  
8  
9 export default Greet
```

## Export default Greet

- This allows us to import the component with any name I can change greet to my component and change the tag to my component and you can see that it still works as expected



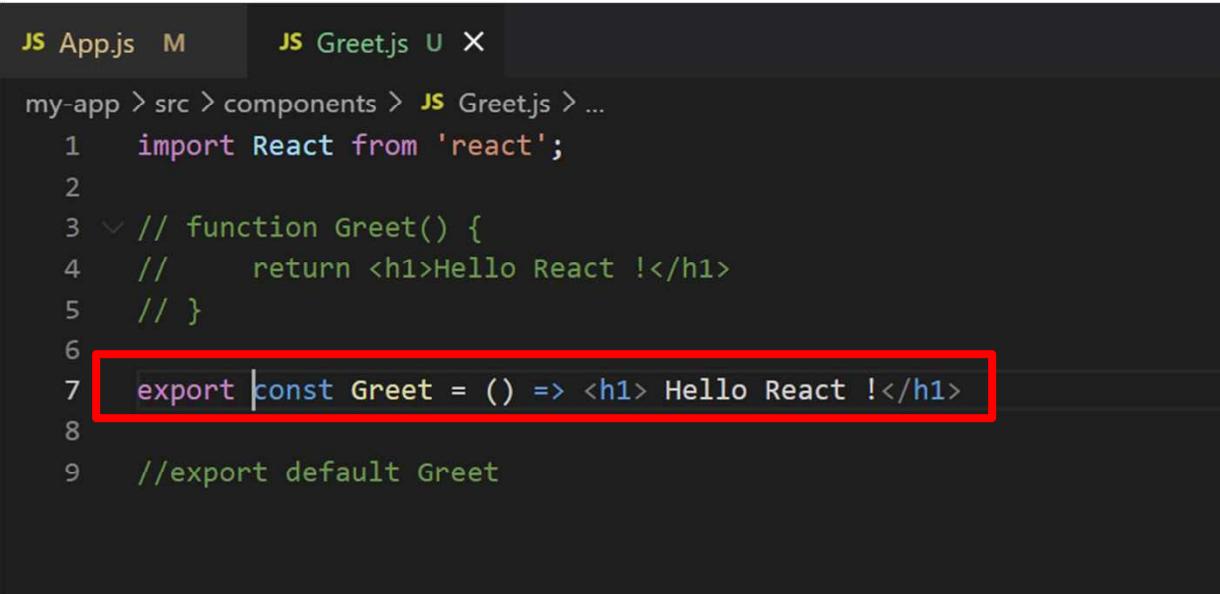
The image shows a code editor with two tabs: "JS App.js" and "JS Greet.js". The "App.js" tab is active, displaying the following code:

```
my-app > src > JS App.js > App
1  import logo from './logo.svg';
2  import './App.css';
3  import MyComponent from './components/Greet';
4
5  function App() {
6    return (
7      <div className="App">
8        <MyComponent />
9      </div>
10 );
11}
12
13 export default App;
14
```

The "Greet.js" tab is visible in the background. The code editor interface includes a file tree on the left showing "my-app", "src", and "App". The code is syntax-highlighted with colors for different language constructs.

## Named exports

- If we use Named export, then we have to import the component with the exact same name.



```
JS App.js M      JS Greet.js U X
my-app > src > components > JS Greet.js > ...
1  import React from 'react';
2
3  // function Greet() {
4  //   return <h1>Hello React !</h1>
5  //}
6
7  export const Greet = () => <h1> Hello React !</h1>
8
9  //export default Greet
```

Compiled with problems:

Hello React !

ERROR in ./src/App.js 11:35-46

```
export 'default' (imported as 'MyComponent') was not found in './components/Greet' (possible exports: Greet)
```

```
JS App.js M X JS Greet.js U
my-app > src > JS App.js > ⚭ App
  1  import logo from './logo.svg';
● 2  import './App.css';
  3  import { Greet } from './components/Greet';
  4
  5  function App() {
  6    return (
  7      <div className="App">
  8        <Greet />
  9      </div>
 10    );
 11  }
 12
 13  export default App;
 14
```

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with a red border around the main content area. On the left is the Explorer sidebar, which displays a file tree for a 'components-demo' directory under a 'REACT' folder. The 'FirstComponent.js' file is selected in the tree. The main central area is the code editor, showing the following code:

```
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;
```

Below the code editor is a status bar with tabs for PROBLEMS, DEBUG CONSOLE, OUTPUT, and TERMINAL. The TERMINAL tab is active, displaying the message: "Note that the development build is not optimized. To create a production build, use npm run build." At the bottom of the status bar, it says "webpack compiled successfully".

The screenshot shows a development environment with a red border around the code editor and browser preview area.

**VS Code Explorer:**

- REACT** folder:
  - components-demo
  - node\_modules
  - public
  - src
    - # App.css
    - JS App.js** (selected)
    - JS App.test.js
    - JS FirstComponent.js
    - # index.css
    - JS index.js
    - logo.svg
    - JS reportWebVitals.js
    - JS setupTests.js
  - .gitignore
  - { package-lock.json

**Code Editor:**

```
JS FirstComponent.js U JS App.js M X  
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       <FirstComponent name={'User'} />  
8     </div>  
9   );  
10 }  
11  
12 export default App;
```

**Browser Preview:**

localhost:3000

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 a code editor interface with a red border. On the left is the Explorer sidebar under the REACT category, 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 main area shows the content of FunctionalFirstComponent.js:

```
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;
```

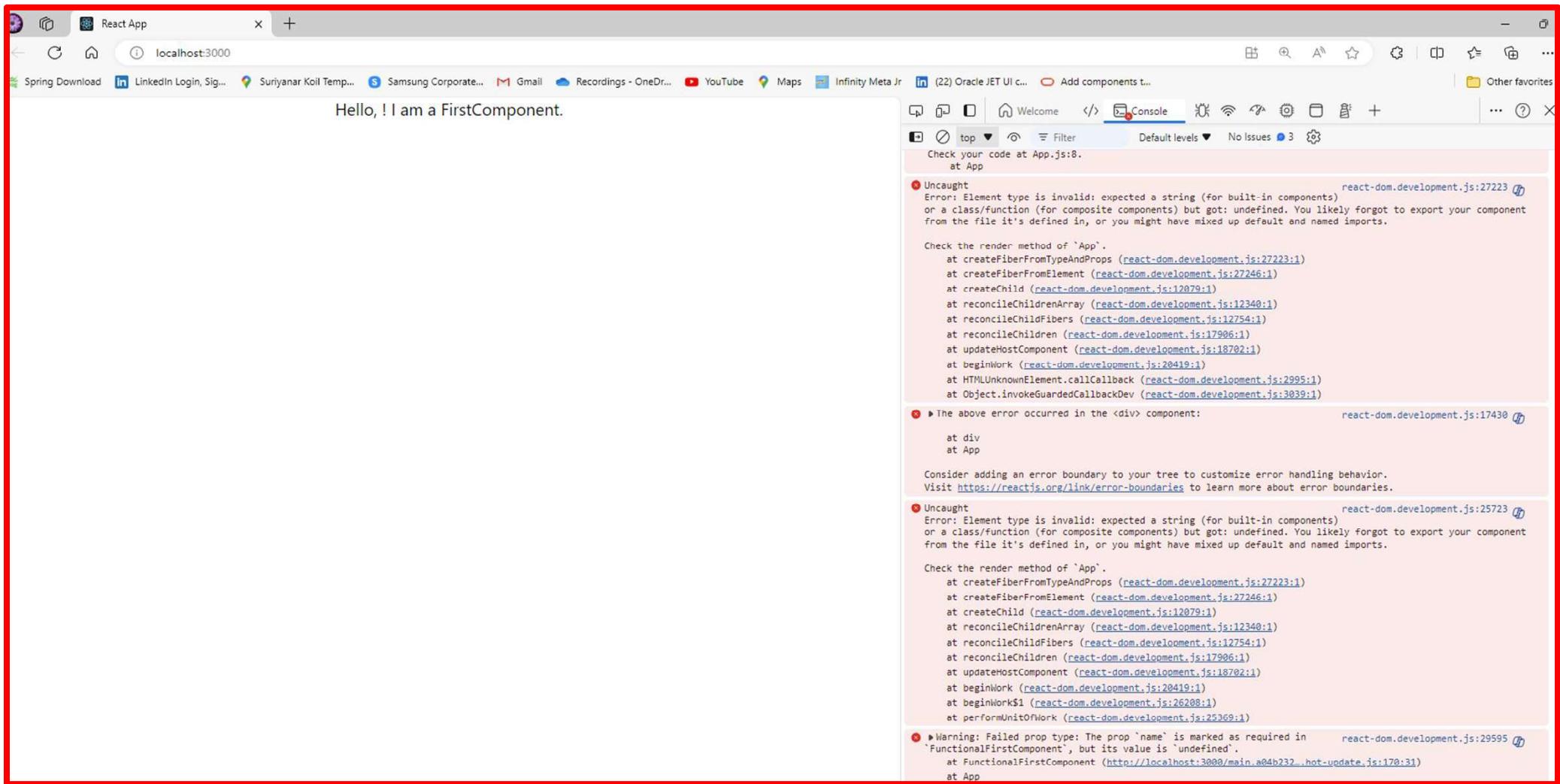
EXPLORER

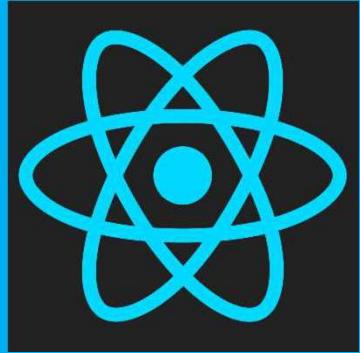
... JS FunctionalFirstComponent.js U JS App.js M X

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       /* <FirstComponent name={'User'} /> */
9       <FunctionalFirstComponent/>
10      </div>
11    );
12  }
13
14 }
15
16 export default App;
17
```

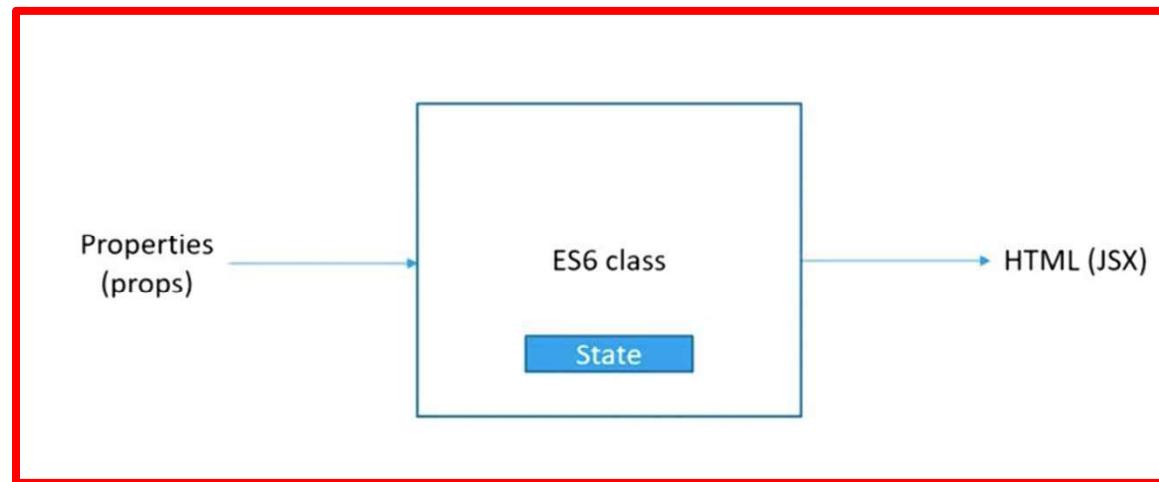
# Output With Validation Error





# The Class Based Components

- Class components are basically ES6 classes similar to a functional component
- A class component also can optionally receive props as input and return HTML apart from the props a class component can also maintain a private internal state.
- In simpler words it can maintain some information which is private to that component and use that information to describe the user interface



## 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

LESSON03-HELLOWORLD

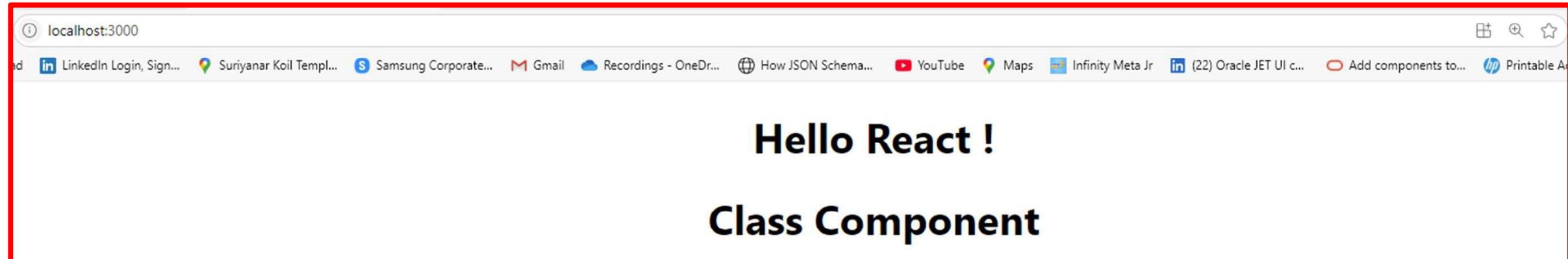
- my-app
- node\_modules
- public
- src
  - components
    - Greet.js
    - Welcome.js
  - App.css
  - App.js

JS App.js M JS Welcome.js U

```
my-app > src > components > JS Welcome.js > [o] default
1 import React, { Component } from 'react';
2
3 class Welcome extends Component {
4     render() {
5         return <h1>Class Component</h1>
6     }
7 }
8
9 export default Welcome;
```

JS App.js M X JS Welcome.js U

```
my-app > src > JS App.js > [o] App
1 import logo from './logo.svg';
2 import './App.css';
3 import { Greet } from './components/Greet';
4 import Welcome from './components/Welcome';
5 function App() {
6     return (
7         <div className="App">
8             <Greet />
9             <Welcome />
10        </div>
11    );
12
13
14 export default App;
15
```



## Functional

Simple functions

Use Func components as much as possible

Absence of 'this' keyword

Solution without using state

Mainly responsible for the UI

Stateless/ Dumb/ Presentational

## Class

More feature rich

Maintain their own private data - state

Complex UI logic

Provide lifecycle hooks

Stateful/ Smart/ Container

```
component-types > src > components > JS GreetClassCompWithState.js > GreetClassCompWithState
1 import React from 'react';
2
3 class GreetClassCompWithState extends React.Component {
4     constructor(props) {
5         super(props);
6         this.state = {
7             toggle: true
8         };
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                <h2>Hello, {this.props.name}, Welcome to React ! </h2>
20                <br/>
21                Toggle is : {this.state.toggle.toString()}
22            </div>
23        )
24    }
25 }
26 export default GreetClassCompWithState
```

The code editor interface includes a navigation bar with File, Edit, Selection, View, Go, and other options. A search bar at the top right contains the text "Lesson05-ComponentTypes". Below the tabs, there are several small icons for file operations like copy, paste, and search. The status bar at the bottom shows file statistics (Ln 16, Col 1), encoding (UTF-8), and file type (JavaScript). The bottom navigation bar indicates "24 of 8" and the topic "Topic: Component Types". The MENTORLABS logo is visible in the bottom right corner.

## 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

✓ 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

## JS App.js M

## JS HighOrderComponents.js U X

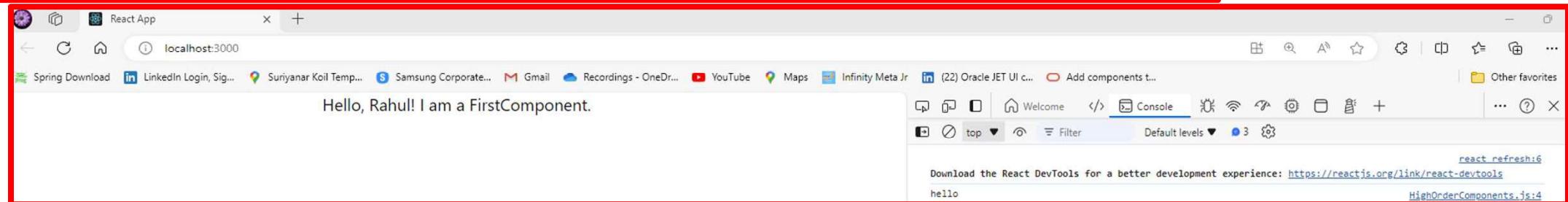
components-demo > src > components > JS HighOrderComponents.js > [?] default

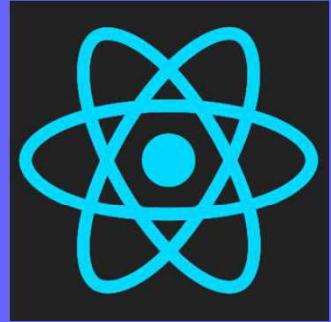
```
1 import React, { Component } from 'react';
2 const PrintHello = ComposedComponent => class extends Component {
3   onClick() {
4     console.log('hello');
5   }
6   /* The higher order component takes another component as a parameter
7   and then renders it with additional props */
8   render() {
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;
```

The screenshot shows the VS Code interface with a red border around the code editor area. The Explorer sidebar on the left shows a project structure under 'REACT' with files like 'App.css', 'App.js', 'App.test.js', etc. The code editor tab bar has 'JS App.js' and 'JS HighOrderComponents.js'. The current file is 'App.js', which contains the following code:

```
components-demo > src > JS App.js > [?] default
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            /* <FirstComponent name={'User'} /> */
11            /* <FunctionalFirstComponent /> */
12            /* <SecondComponent name="Rahul" /> */
13            <ExtendedComponent name="Rahul" />
14        </div>
15    );
16}
17
18
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
```

The screenshot shows the VS Code interface with a red border around the main content area. On the left is the Explorer sidebar showing a project structure under 'REACT'. The 'src' folder contains 'components' and 'nesting-components'. Inside 'nesting-components', 'NestingCompsDemo01.js' is selected and highlighted with a green background. The main editor tab is also titled 'NestingCompsDemo01.js'. The code itself defines two React components: 'CommentList' and 'CommentForm', which are nested within each other.

```
components-demo > src > nesting-components > 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 );
```

The screenshot shows a dark-themed interface of the Visual Studio Code code editor. On the left, the Explorer sidebar displays a project structure under the 'REACT' folder, including files like App.css, App.js, App.test.js, index.css, index.js, logo.svg, reportWebVitals.js, and setupTests.js. The 'NestingCompsDemo01.js' file is currently selected and shown in the main editor area. The code itself is as follows:

```
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;
```

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       /* <FirstComponent name='User' /> */
12       /* <FunctionalFirstComponent /> */
13       /* <SecondComponent name="Rahul" /> */
14       /* <ExtendedComponent name="Rahul" /> */
15       <CommentBox />
16     </div>
17   );
18 }
19
20 export default App;
```

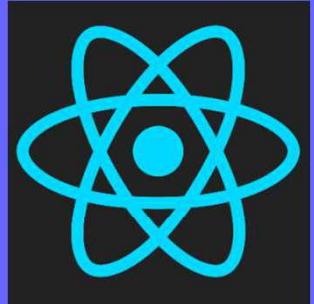
localhost:3000



[dIn Login, Sig...](#) [Suriyanar Koil Temp...](#) [Samsung Corporate...](#) [Gmail](#) [Recordings - OneDr...](#) [YouTube](#) [Maps](#) [Infinity Meta Jr](#) [\(22\) Oracle JET UI c...](#) [Add components t...](#)

## Comments

Hello, world! I am a CommentList.  
Hello, world! I am a CommentForm.



# 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

