**Week-7 REACT**

**Objectives**

* **List the features of ES6**

ES6 (also called ECMAScript 2015) introduced many new features to make JavaScript more powerful and cleaner.

**Key features:**

* let and const for variable declaration
* Arrow functions (=>)
* Classes and inheritance
* Template literals (Hello ${name})
* Destructuring
* Spread and Rest operators (...)
* Promises
* map(), set(), forEach() etc.
* **Explain JavaScript let**

 let is used to declare a **block-scoped** variable.

 Unlike var, it **doesn’t get hoisted** to the top of the function.

 You **can change** the value of a let variable.

* **Identify the differences between var and let**

| **Feature** | **var** | **let** |
| --- | --- | --- |
| Scope | Function scoped | Block scoped {} |
| Hoisting | Yes (initialized as undefined) | Yes (but not initialized) |
| Redeclaration | Allowed | Not allowed in same scope |

* **Explain JavaScript const**

const is used to declare constants (values that cannot be changed).

Like let, it's block-scoped.

Once assigned, you cannot reassign a const variable.

* **Explain ES6 class fundamentals**

ES6 allows you to write object-oriented code using the class keyword.

A class contains:

constructor() method (called when object is created)

* **Explain ES6 class inheritance**

Inheritance means one class can **extend another class**.

The child class gets all properties and methods of the parent.

* **Define ES6 arrow functions**

Arrow functions are a shorter way to write functions.

**Normal function:**

function add(a, b) {

return a + b;

}

**Arrow function:**

const add = (a, b) => a + b;

**Benefits:**

Cleaner syntax

No need to write function

this behaves differently (it keeps the outer context)

* **Identify set(), map()**

Set – Unique values only

A Set stores unique values (no duplicates)

const mySet = new Set();

mySet.add(1);

mySet.add(2);

mySet.add(1);

console.log(mySet);

**Map – Key-value pairs**

A Map holds key-value pairs

Keys can be any type (string, number, object)

const myMap = new Map();

myMap.set("name", "Sohana");

myMap.set("age", 22);

console.log(myMap.get("name"));

**Create a React Application named “cricketapp” with the following components:**

1. ListofPlayers

* Declare an array with 11 players and store details of their names and scores using the map feature of ES6



* Filter the players with scores below 70 using arrow functions of ES6.



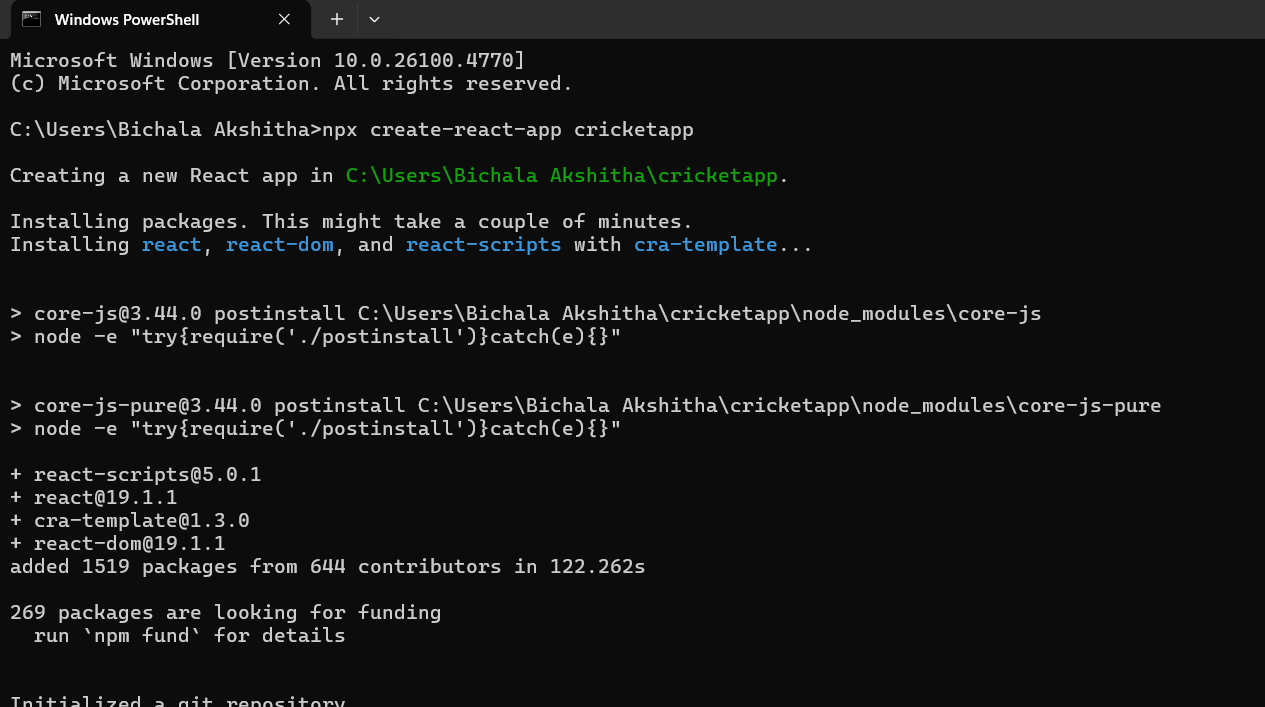
1. IndianPlayers
   1. Display the Odd Team Player and Even Team players using the Destructuring features of ES6



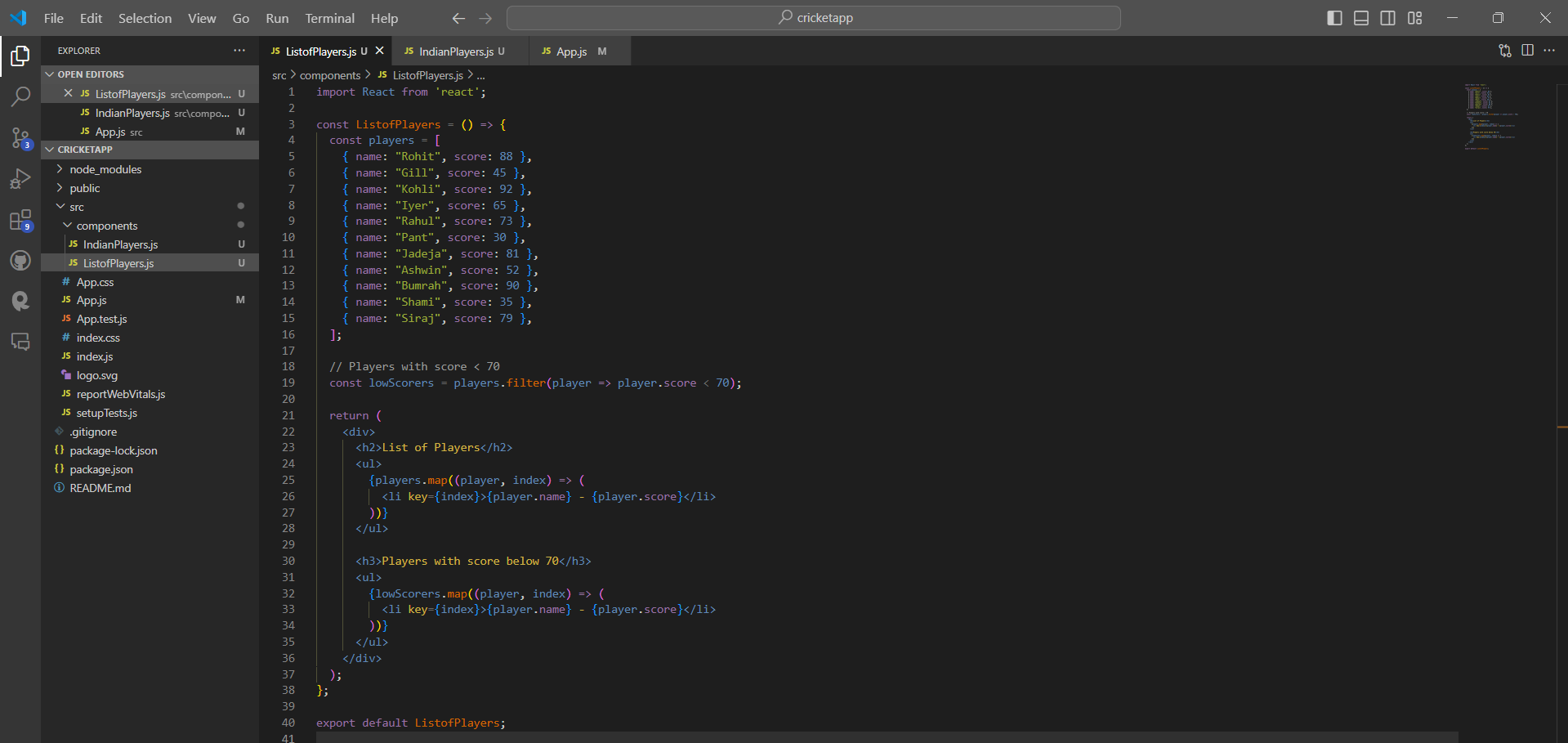
* 1. Declare two arrays T20players and RanjiTrophy players and merge the two arrays and display them using the Merge feature of ES6



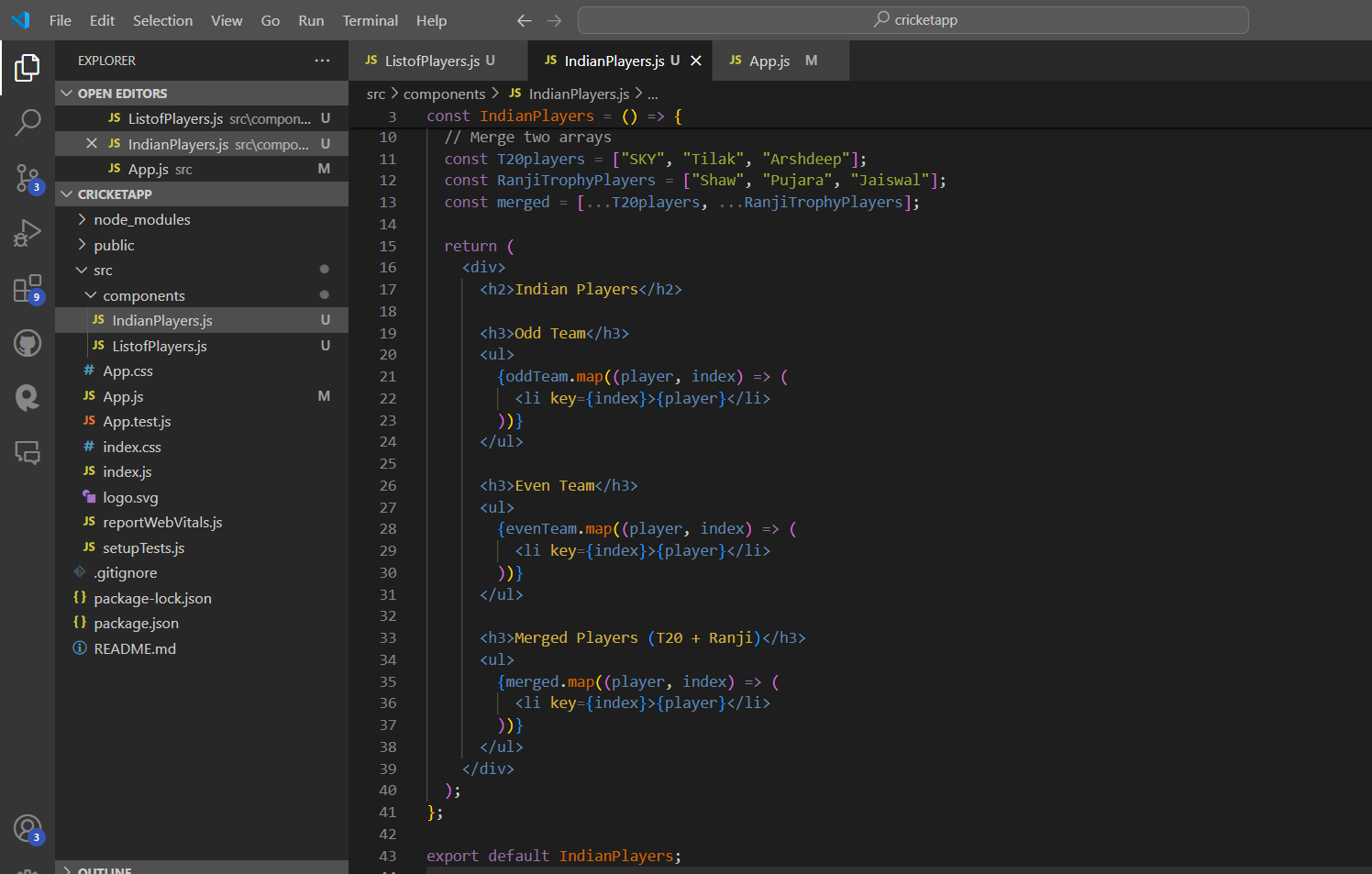
Display these two components in the same home page using a simple if else in the flag variable.



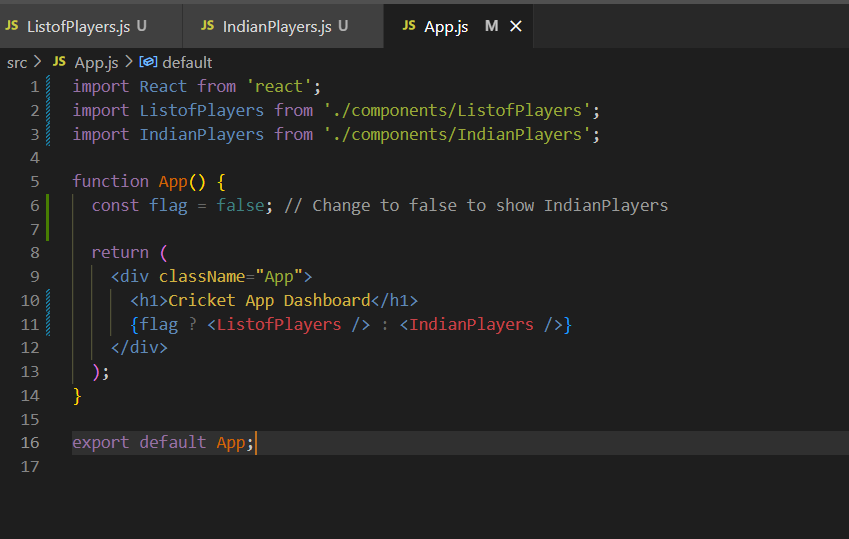
ListofPlayers.js:

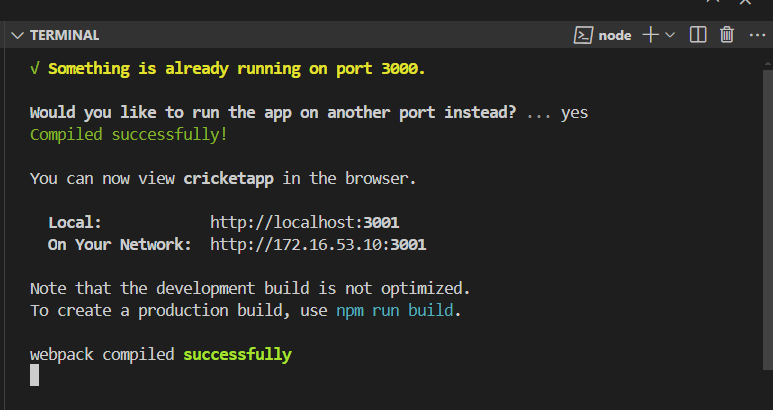


IndianPlayers.js:



App.js:



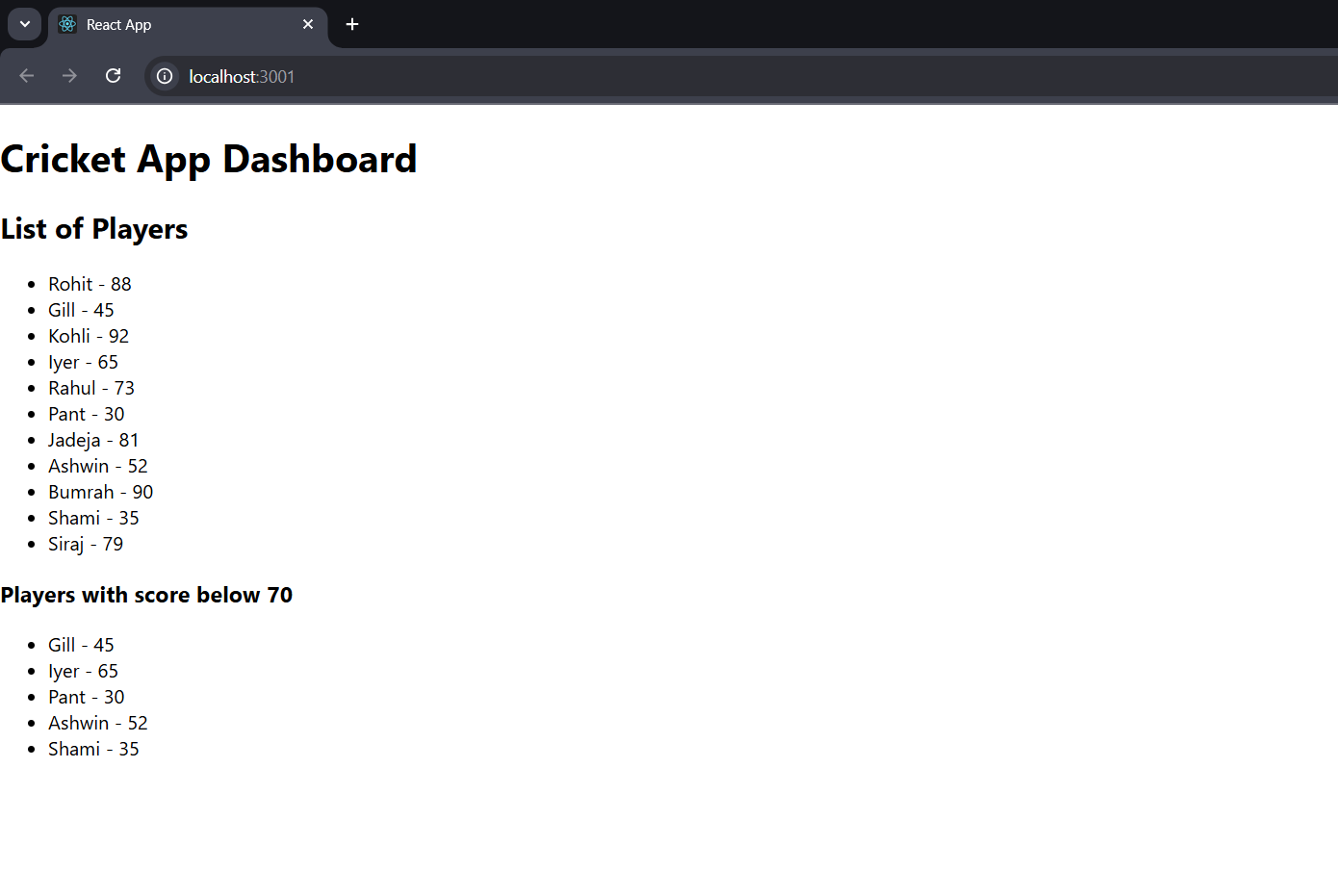


Output:

Flag=false



Flag=true



**Objectives**

* **Define JSX**

**JSX (JavaScript XML)** is a syntax extension for JavaScript used in React.

It lets you write HTML-like code inside JavaScript.

**Example:**

const element = <h1>Hello, Sohana!</h1>;

JSX is easier to write and read than using React.createElement() directly.

* **Explain about ECMA Script**

ECMAScript (ES) is the standard that defines how JavaScript should work.  
React apps use modern JavaScript (ES6 and beyond).

ES6 introduced:

let, const

Arrow functions (=>)

Classes

Template literals (Hello ${name})

Destructuring

Spread and Rest operators

* **Explain React.createElement()**

React doesn’t use real HTML inside JS. Behind the scenes, JSX becomes:

React.createElement(type, props, children)

**Example:**

// This JSX:

<h1>Hello!</h1>

// Becomes this:

React.createElement("h1", null, "Hello!")

JSX is just a **shortcut** for React.createElement().

* **Explain how to create React nodes with JSX**

A **React Node** is a piece of UI (element) created using JSX.

const title = <h2>Welcome to My Page</h2>;

function Welcome() {

return <h1>Hello React</h1>;

}

Every JSX tag creates a **React element (node)** behind the scenes.

* **Define how to render JSX to DOM**

You use ReactDOM.render() (in older React) or <App /> in index.js (modern way).

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));

* **Explain how to use JavaScript expressions in JSX**

You can put any valid JavaScript expression inside {} in JSX.

const name = "Soha";

const element = <h2>Hello, {name}</h2>;

Math: <p>{5 + 10}</p>

Function calls: <p>{greet()}</p>

Ternary condition: <p>{score > 50 ? "Pass" : "Fail"}</p>

* **Explain how to use inline CSS in JSX**

You can apply CSS directly to elements using the style attribute as an object.

Syntax: style={{ key: 'value' }}  
(Use **camelCase** for CSS properties)

<h1 style={{ color: 'blue', fontSize: '24px' }}>

Styled Heading

</h1>

Double braces {{ }} → first for JSX, second for object

**Create a React Application named “officespacerentalapp” which uses React JSX to create elements, attributes and renders DOM to display the page.**

**Create an element to display the heading of the page.**

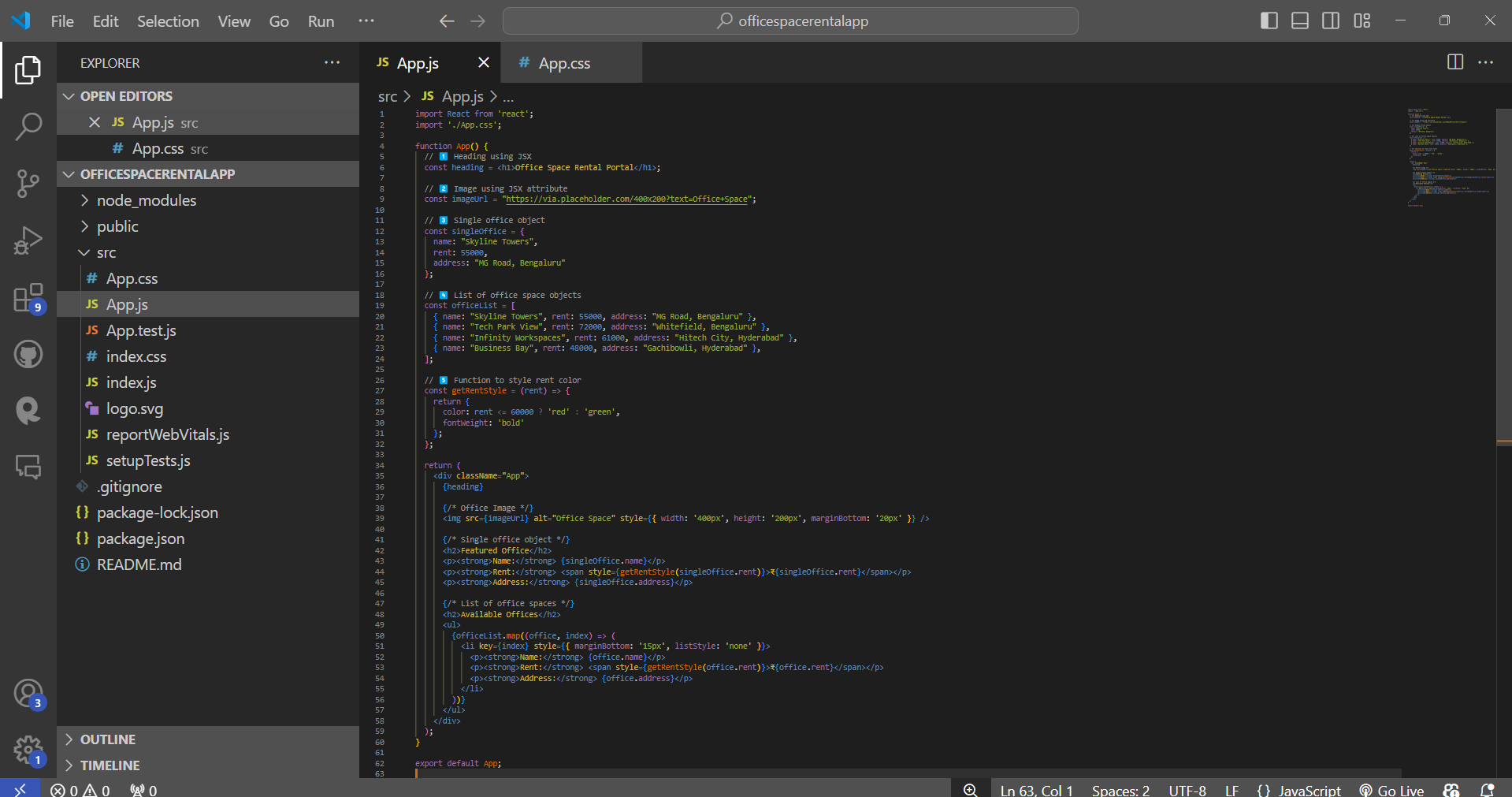
**Attribute to display the image of the office space**

**Create an object of office to display the details like Name, Rent and Address.**

**Create a list of Object and loop through the office space item to display more data.**

**To apply Css, Display the color of the Rent in Red if it’s below 60000 and in Green if it’s above 60000.**

**App.js:**



App.css:

.App {

  padding: 20px;

  font-family: Arial, sans-serif;

}

h1 {

  color: #2c3e50;

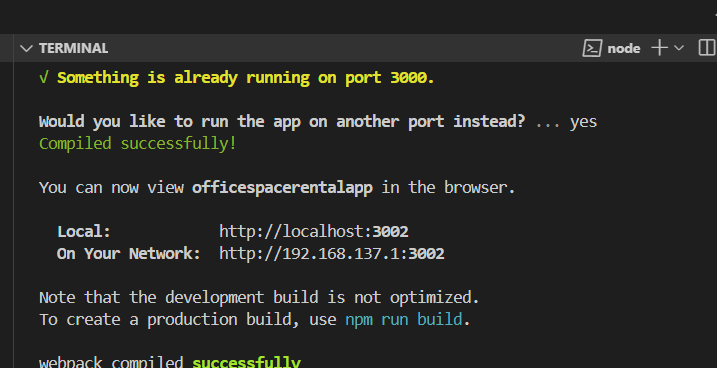
}

h2 {

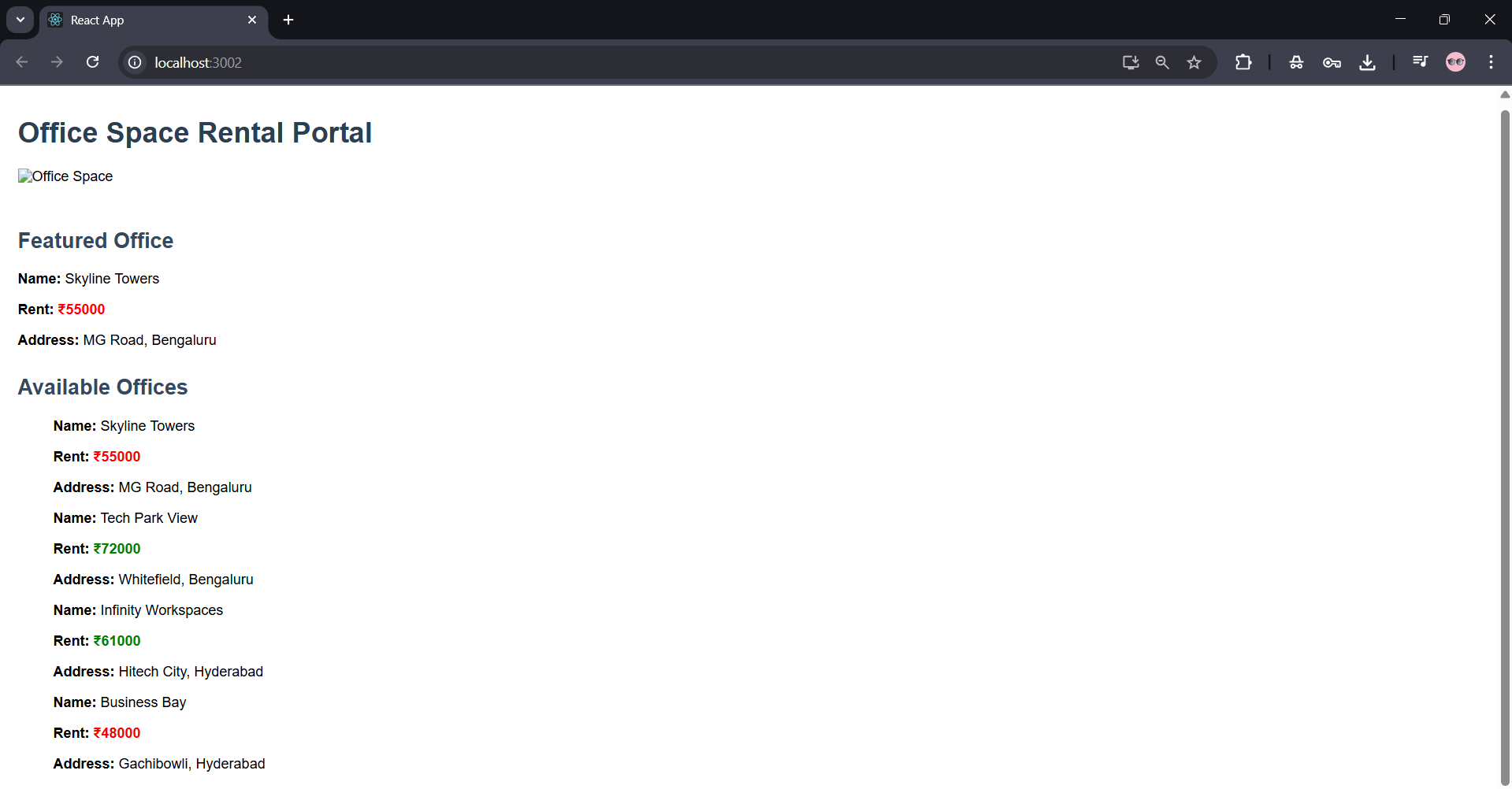
  color: #34495e;

  margin-top: 30px;

}



Output:



**Objectives**

* **Explain React events**

React events are just like regular DOM events (like click, submit, hover),  
but they use React’s own system called Synthetic Events to work across all browsers.

Common examples:

* onClick
* onChange
* onSubmit
* onMouseOver
* **Explain about event handlers**

An **event handler** is just a **function** that runs **when an event occurs** (like click, input, etc.)

function handleClick() {

alert("Button clicked!");

}

<button onClick={handleClick}>Click</button>

In React, event handlers are usually passed as props like onClick={functionName}.

* **Define Synthetic event**

React uses something called a **SyntheticEvent** – a wrapper around the browser's native events.

It gives the **same behavior across all browsers** (cross-browser compatibility).

function handleClick(e) {

console.log(e); // This 'e' is a synthetic event

}

It acts just like the real event object but works the same in Chrome, Firefox, Safari, etc.

* **Identify React event naming convention**

| **Feature** | **React Naming** | **DOM Naming** |
| --- | --- | --- |
| React uses camelCase | onClick, onChange, onMouseOver | HTML uses lowercase |
| React uses JSX function call | {handleClick} | HTML uses string like "handleClick()" |

**Create a React Application “eventexamplesapp” to handle various events of the form elements in HTML.**

1. **Create “Increment” button to increase the value of the counter and “Decrement” button to decrease the value of the counter. The “Increase” button should invoke multiple methods.** 
   1. **To increment the value**
   2. **Say Hello followed by a static message.**

**EventExamples.js:**

import React, { Component } from 'react';

class EventExamples extends Component {

  constructor(props) {

    super(props);

    this.state = {

      counter: 0,

      result: null,

      rupees: ''

    };

    this.handleConvert = this.handleConvert.bind(this);

  }

  increment = () => {

    this.setState({ counter: this.state.counter + 1 });

    this.sayHello();

  };

  decrement = () => {

    this.setState({ counter: this.state.counter - 1 });

  };

  sayHello = () => {

    alert("Hello! This is a static message.");

  };

  sayWelcome = (message) => {

    alert(`Message: ${message}`);

  };

  handleSyntheticEvent = (e) => {

    alert("I was clicked");

  };

  handleConvert(e) {

    e.preventDefault();

    const euro = (this.state.rupees / 90).toFixed(2);

    this.setState({ result: `€${euro}` });

  }

  handleInputChange = (e) => {

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

  };

  render() {

    return (

      <div style={{ padding: "20px" }}>

        <h2>React Event Examples</h2>

        {/\* Counter \*/}

        <h3>Counter: {this.state.counter}</h3>

        <button onClick={this.increment}>Increment</button>

        <button onClick={this.decrement}>Decrement</button>

        {/\* Welcome Button \*/}

        <div style={{ marginTop: '20px' }}>

          <button onClick={() => this.sayWelcome("Welcome to React Events")}>Say Welcome</button>

        </div>

        {/\* Synthetic Event \*/}

        <div style={{ marginTop: '20px' }}>

          <button onClick={this.handleSyntheticEvent}>OnPress</button>

        </div>

        {/\* Currency Converter \*/}

        <div style={{ marginTop: '40px' }}>

          <h3>Currency Converter (INR to Euro)</h3>

          <input

            type="number"

            placeholder="Enter INR"

            value={this.state.rupees}

            onChange={this.handleInputChange}

          />

          <button onClick={this.handleConvert}>Convert</button>

          {this.state.result && <p>Converted Amount: {this.state.result}</p>}

        </div>

      </div>

    );

  }

}

export default EventExamples;

**App.js:**

import React from 'react';

import './App.css';

import EventExamples from './components/EventExamples';

function App() {

  return (

    <div className="App">

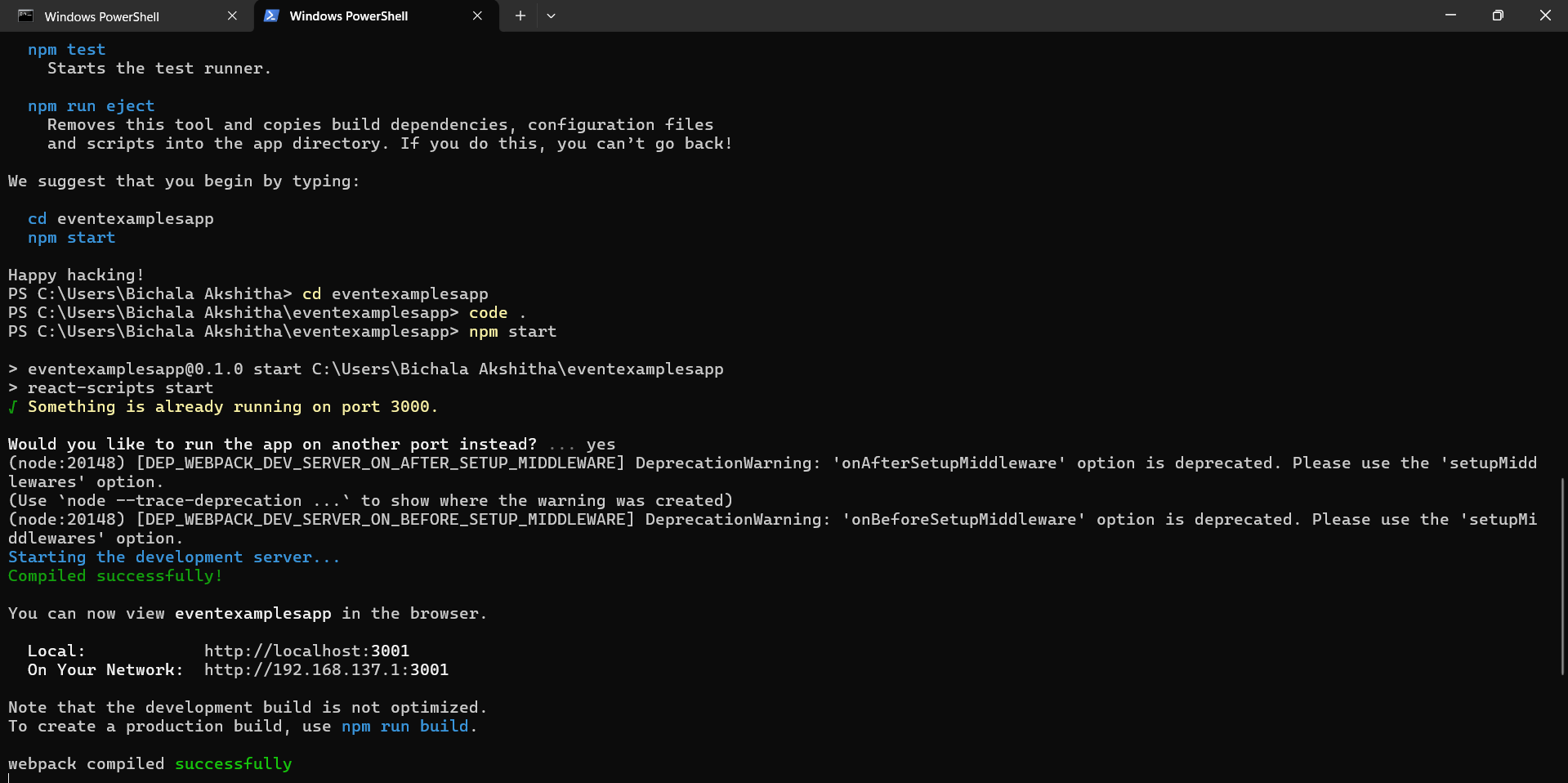
      <EventExamples />

    </div>

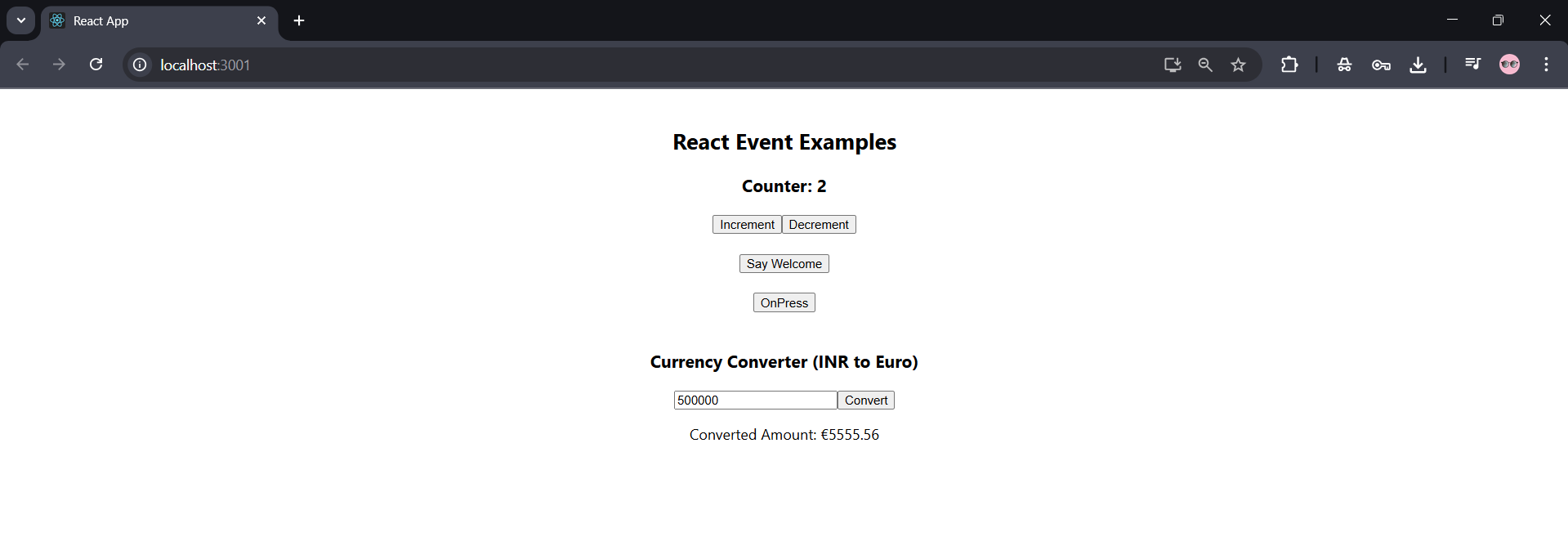
  );

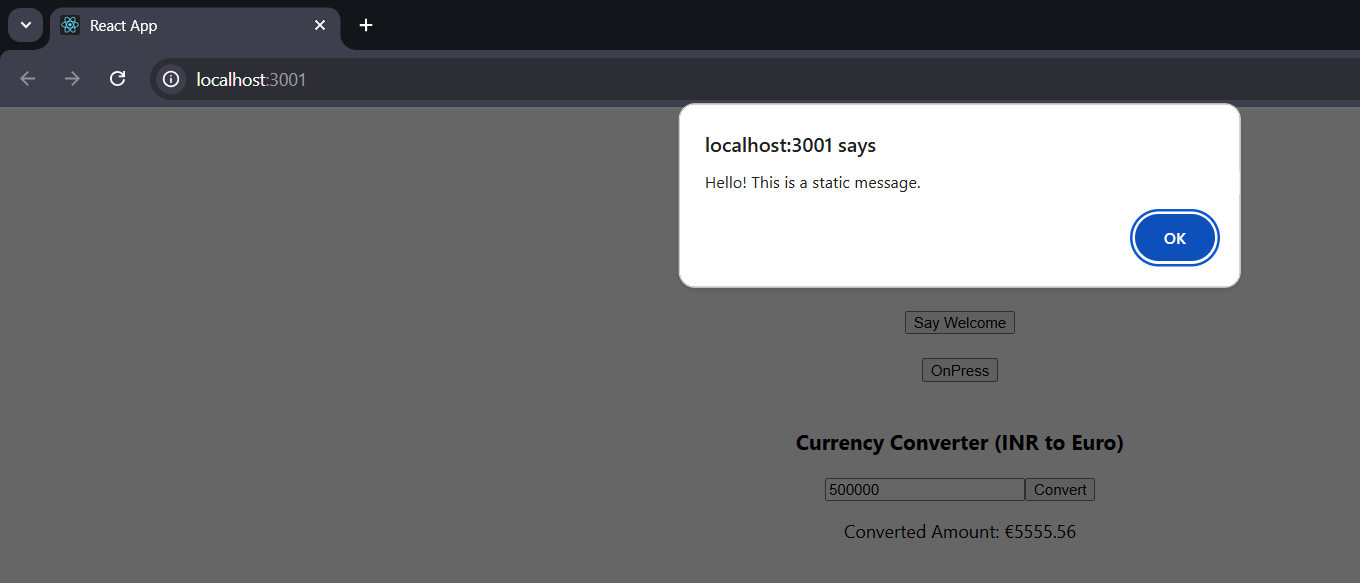
}

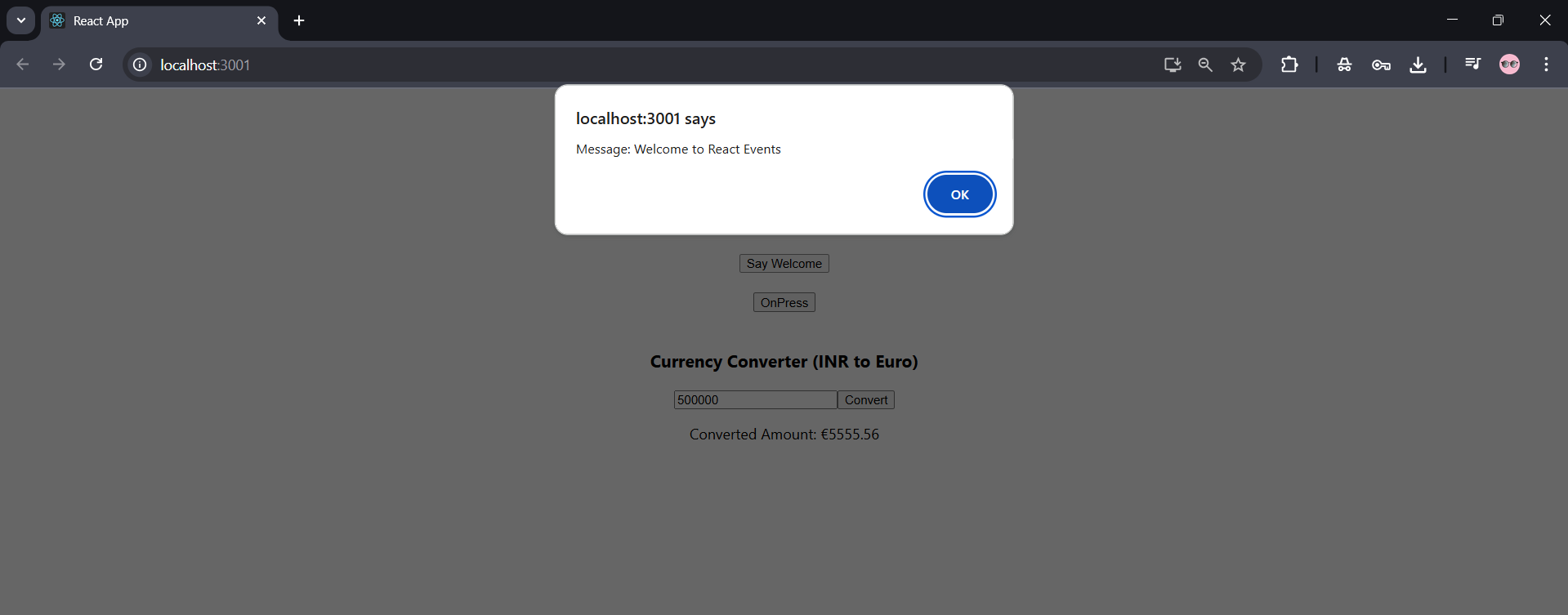
export default App;

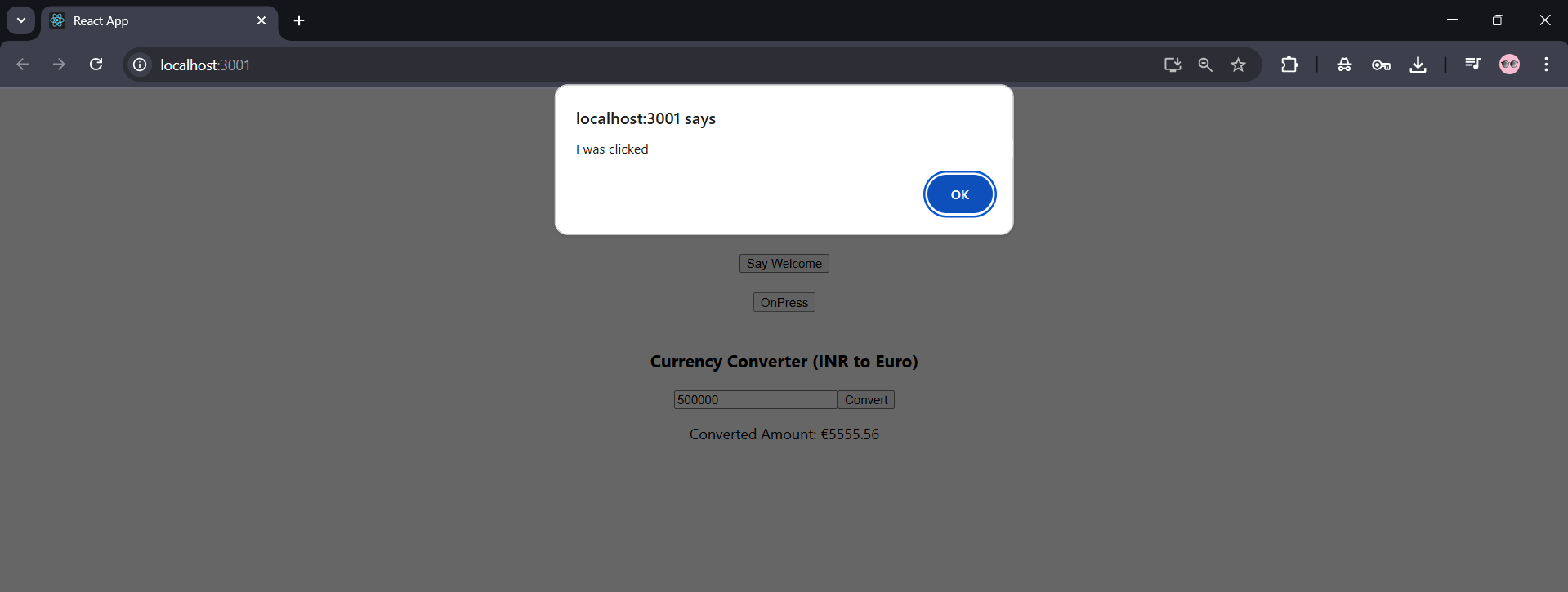


**Output:**



****



****

**Objectives**

* **Explain about conditional rendering in React**

Conditional rendering means showing different UI elements based on a condition (like user login, data availability, etc.).

Just like if-else in JavaScript, React allows you to show or hide components dynamically.

**Common use cases:**

Show **Login** button if user is not logged in

Show **User Dashboard** if logged in

Show **“No Data Found”** if array is empty

* **Define element variables**

You can store **JSX elements inside a variable**, and then use it in return() to control what gets displayed.

Makes your render() cleaner

Helps you use if-else (not just ternary ? :)

Useful when components to render are big or complex

**Example:**

let content;

if (isLoggedIn) {

content = <UserPage />;

} else {

content = <GuestPage />;

}

return (

<div>

{content} {/\* rendered element stored in variable \*/}

</div>

);

* **Explain how to prevent components from rendering**

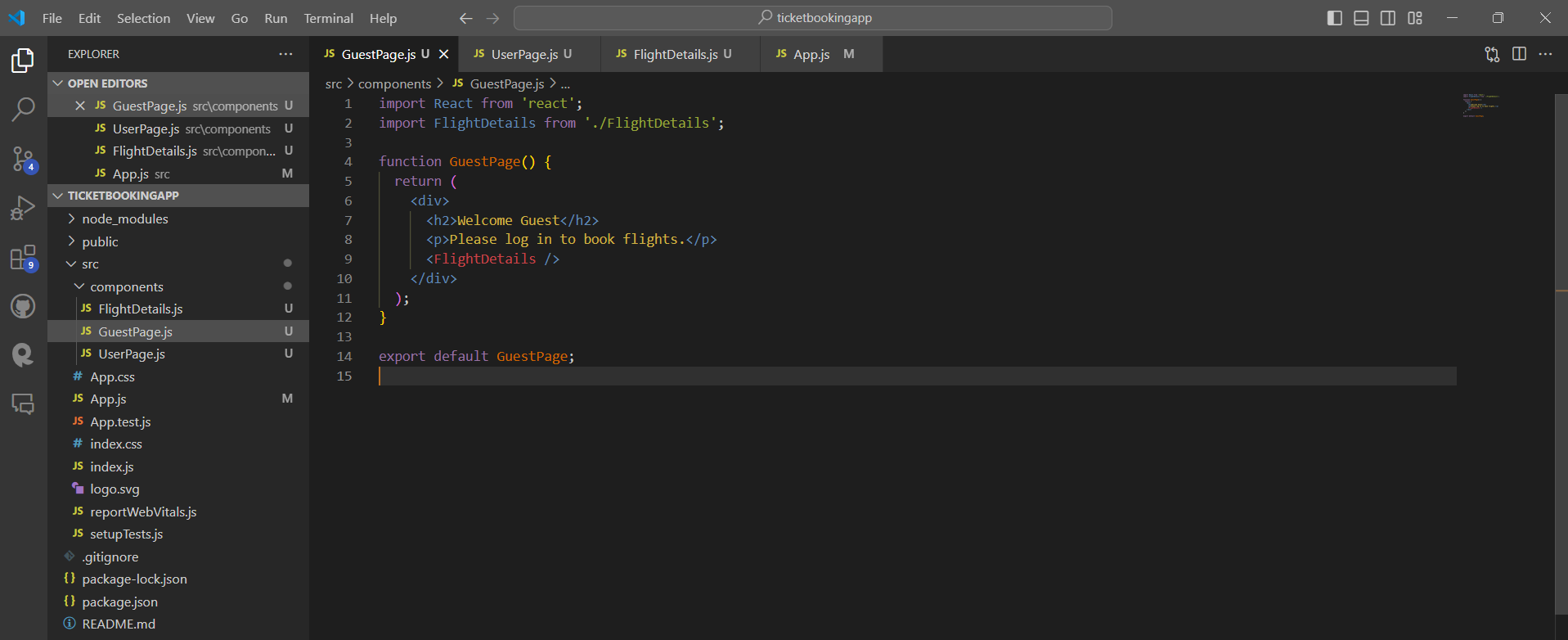
Sometimes you may want to **completely skip rendering** a component based on a condition.

You can prevent rendering by returning **null** or using short-circuit logic (&&).

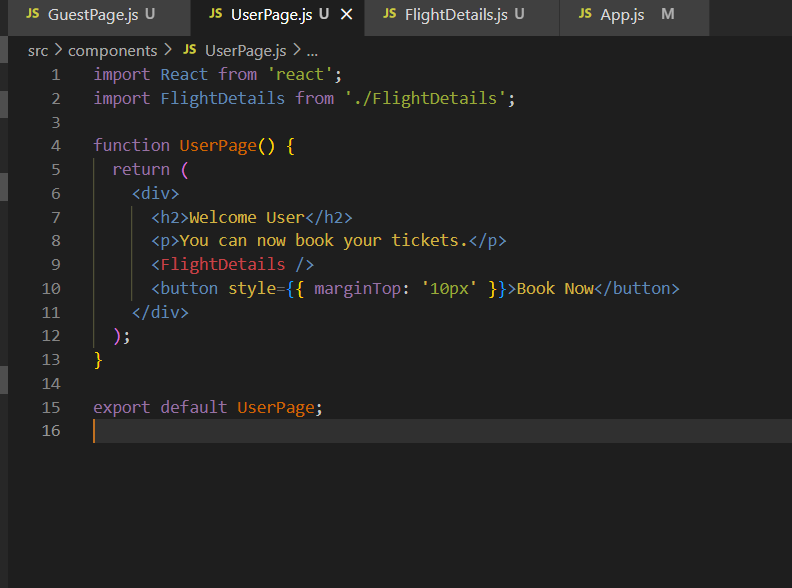
**Create a React Application named “ticketbookingapp” where the guest user can browse the page where the flight details are displayed whereas the logged in user only can book tickets.**

**The Login and Logout buttons should accordingly display different pages. Once the user is logged in the User page should be displayed. When the user clicks on Logout, the Guest page should be displayed.**

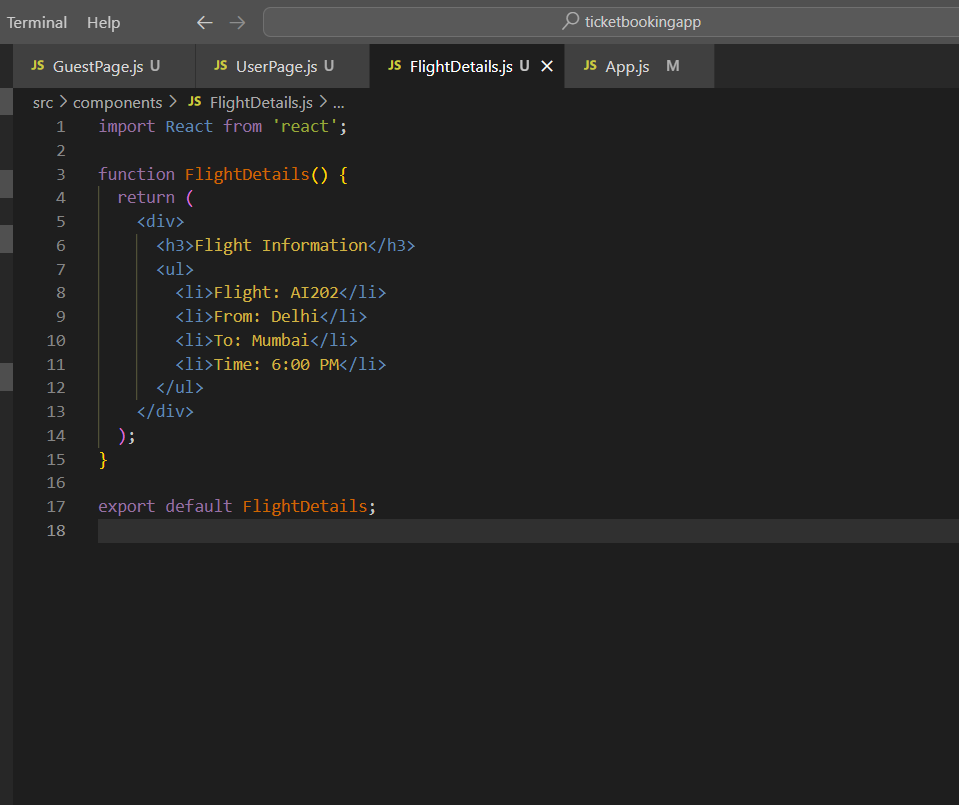
**GuestPage.js:**

****

UserPage.js:



FlightDetails.js:



App.js:

import React, { useState } from 'react';

import './App.css';

import GuestPage from './components/GuestPage';

import UserPage from './components/UserPage';

function App() {

  const [isLoggedIn, setIsLoggedIn] = useState(false);

  const loginHandler = () => {

    setIsLoggedIn(true);

  };

  const logoutHandler = () => {

    setIsLoggedIn(false);

  };

  // Element variables

  let content;

  if (isLoggedIn) {

    content = <UserPage />;

  } else {

    content = <GuestPage />;

  }

  return (

    <div className="App">

      <h1>Ticket Booking App</h1>

      {isLoggedIn ? (

        <button onClick={logoutHandler}>Logout</button>

      ) : (

        <button onClick={loginHandler}>Login</button>

      )}

      {/\* Conditional content \*/}

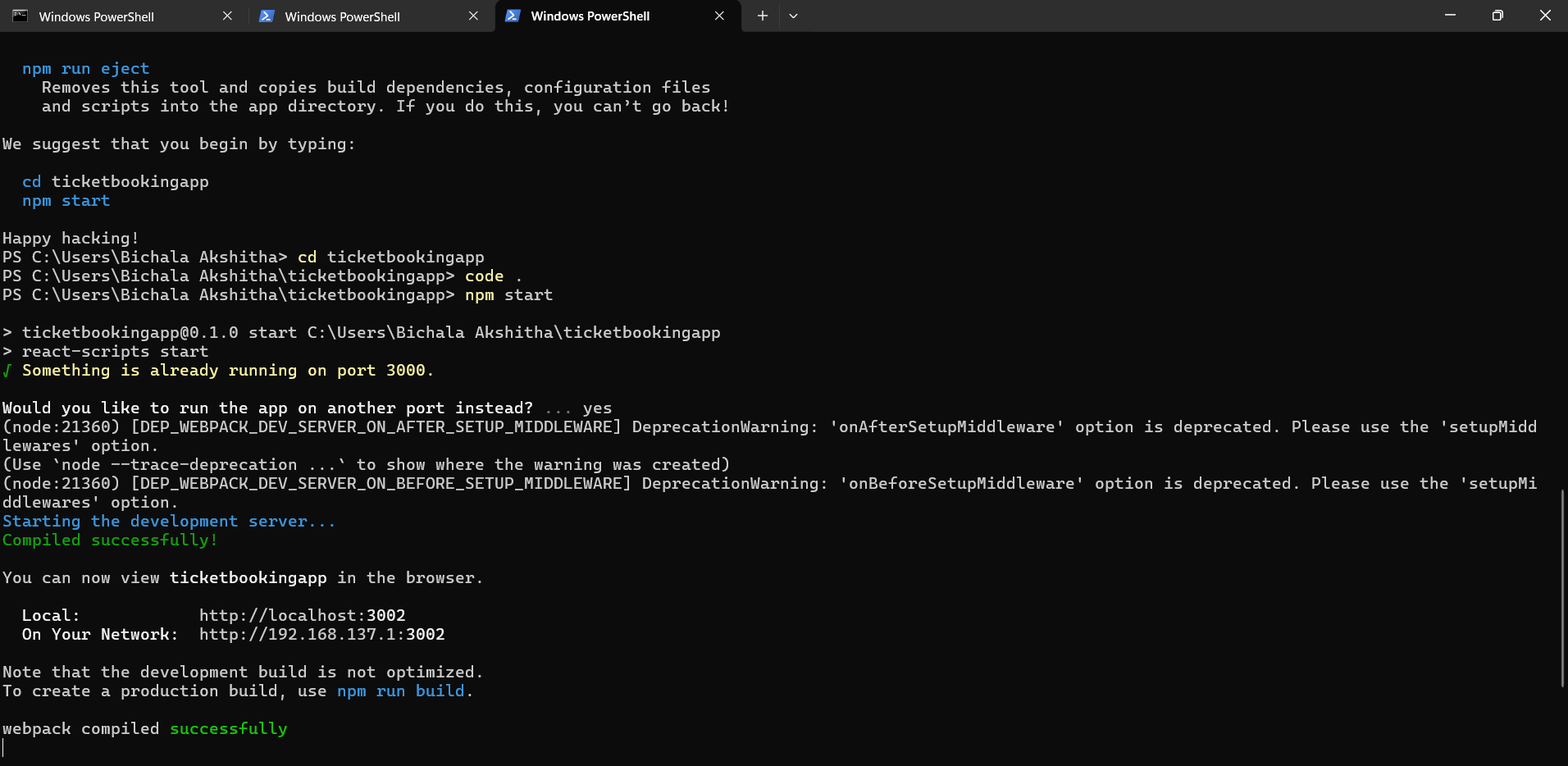
      {content}

    </div>

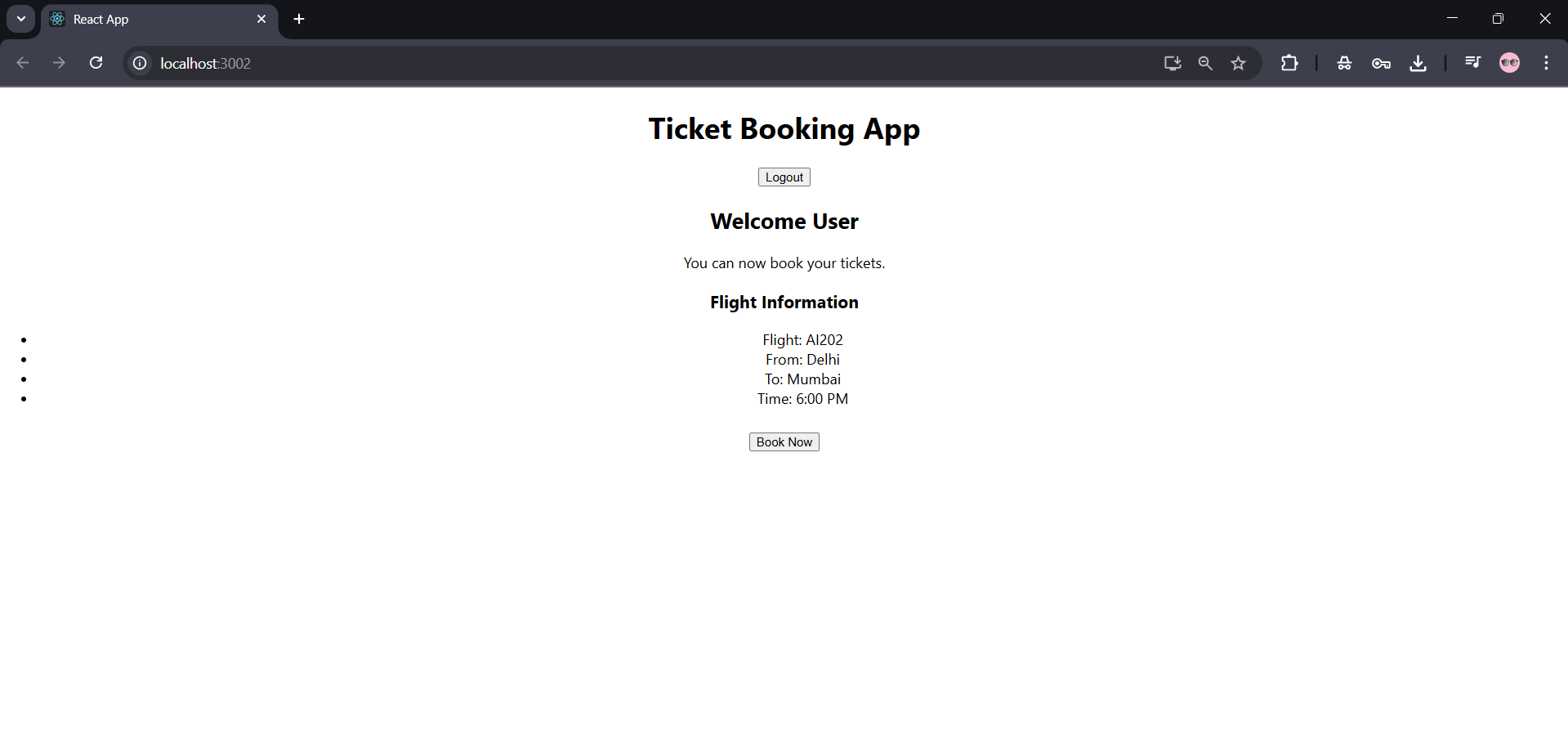
  );

}

export default App;



**Output:**

****

**Objectives**

* **Explain various ways of conditional rendering**

Conditional rendering means showing different UI elements based on conditions like user login, selection, etc.

Common ways:

| **Method** | **Example** | **Use Case** |
| --- | --- | --- |
| **if-else** | if (loggedIn) return <Dashboard />; | Long blocks |
| **element variables** | let content = <Home /> | Clean return() |
| **ternary (? :)** | {isAdmin ? <AdminPanel /> : <UserPanel />} | Simple two-option logic |
| **short-circuit (&&)** | {hasPermission && <SecretPage />} | Show only if true |

* **Explain how to render multiple components**

You can render multiple components inside a single parent element using:

return (

<div>

<Header />

<Content />

<Footer />

</div>

);

Or with React Fragments (no extra <div>):

<>

<Navbar />

<Sidebar />

</>

This is useful when you want to show different parts of the UI together.

* **Define list component**

A **List component** in React is used to display multiple items (like products, users, blog posts) using map().

function UserList({ users }) {

return (

<ul>

{users.map(user => <li key={user.id}>{user.name}</li>)}

</ul>

);

}

It loops through an array and returns a JSX list.

* **Explain about keys in React applications**

key is a **unique identifier** for each element in a list.

It helps React **track which item changed, added, or removed** during re-rendering.

It must be **stable and unique** (e.g., id, not index if items change dynamically).

key is a **unique identifier** for each element in a list.

It helps React **track which item changed, added, or removed** during re-rendering.

It must be **stable and unique** (e.g., id, not index if items change dynamically).

{products.map(p => <Product key={p.id} name={p.name} />)}

Without key, React may show incorrect or buggy behavior.

* **Explain how to extract components with keys**

You can **split a list into smaller components** and still assign keys when rendering:

function ProductItem({ product }) {

return <li>{product.name}</li>;

}

function ProductList({ products }) {

return (

<ul>

{products.map(prod => (

<ProductItem key={prod.id} product={prod} />

))}

</ul>

);

}

The key must always go to the outermost element in the loop (<ProductItem /> here).

* **Explain React Map, map() function**

React uses JavaScript’s Array.map() to create components from arrays.

array.map(item => { return <Component key={...} /> })

it’s commonly used to:

* Render lists of elements
* Generate dynamic content

const names = ["Sohana", "Akshitha", "Kiran"];

return (

<ul>

{names.map((name, index) => (

<li key={index}>{name}</li>

))}

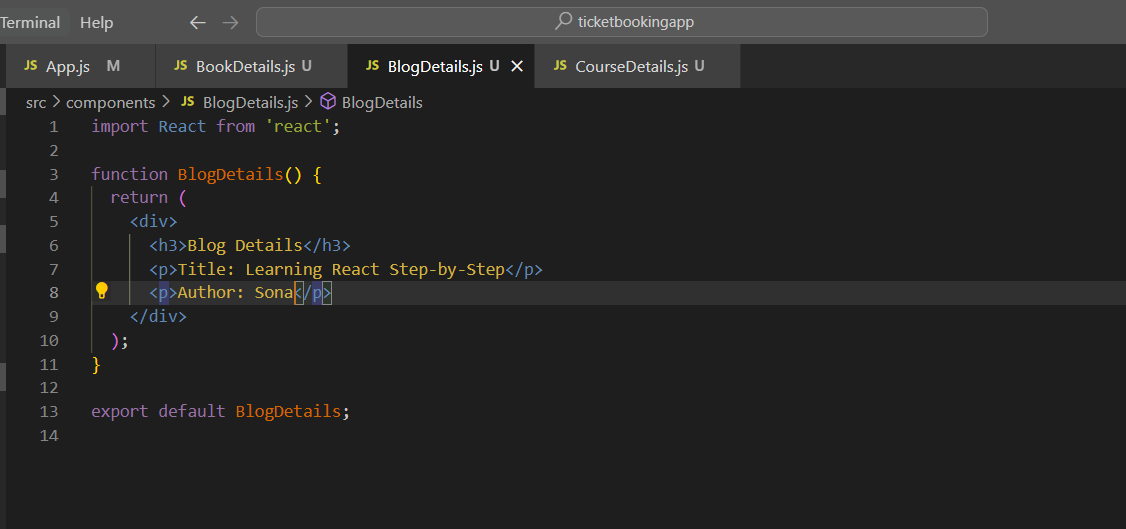
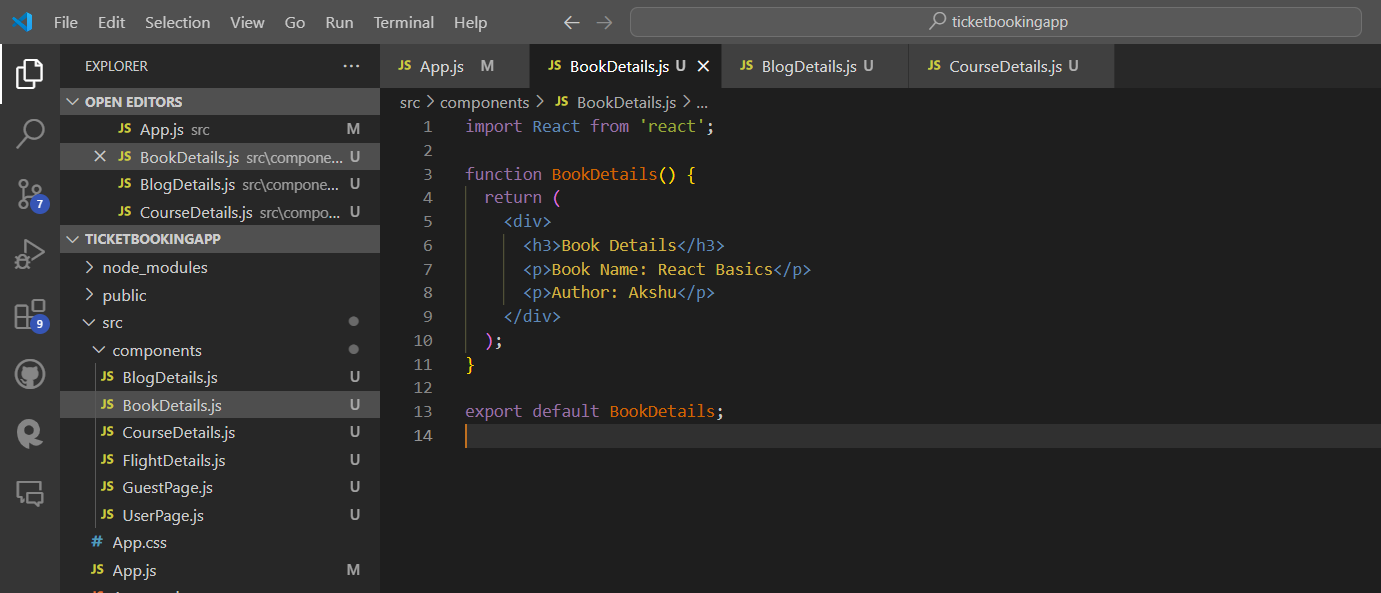
</ul>

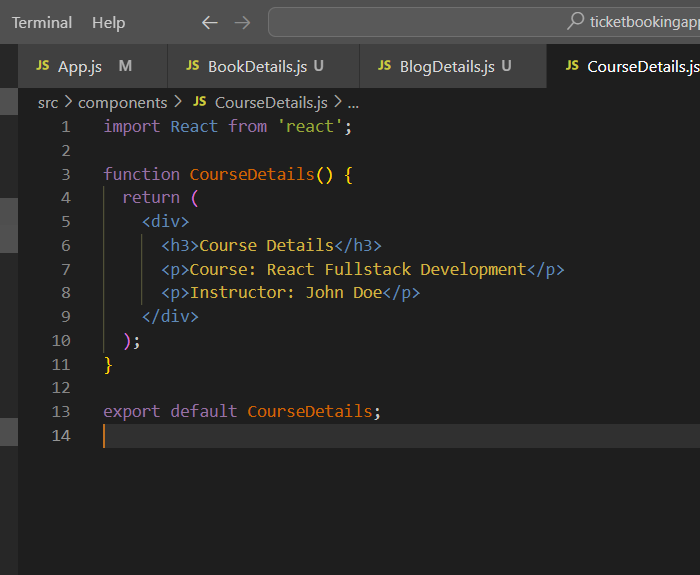
);

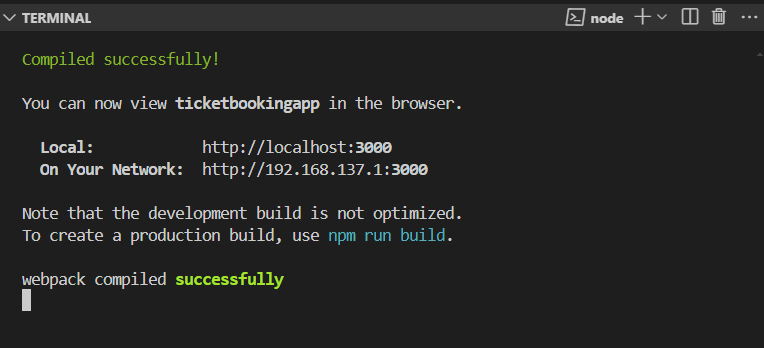
**Implement this with as many ways possible of Conditional Rendering.** **Create a React App named “bloggerapp” in with 3 components.**

1. **Book Details**
2. **Blog Details**
3. **Course Details**

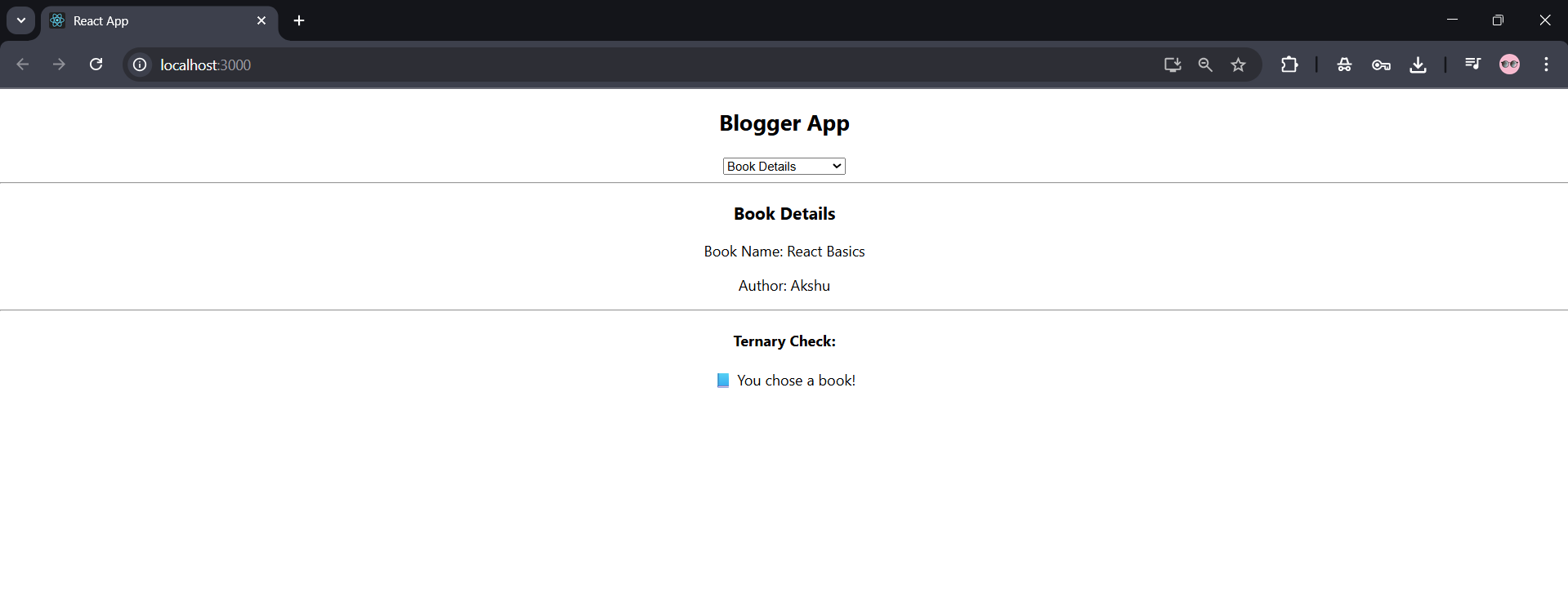
**Implement this with as many ways possible of Conditional Rendering.**

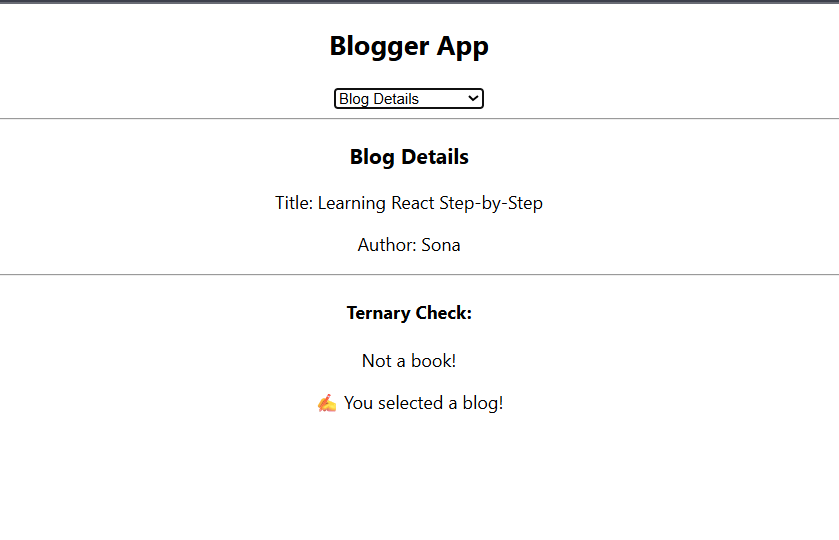
****

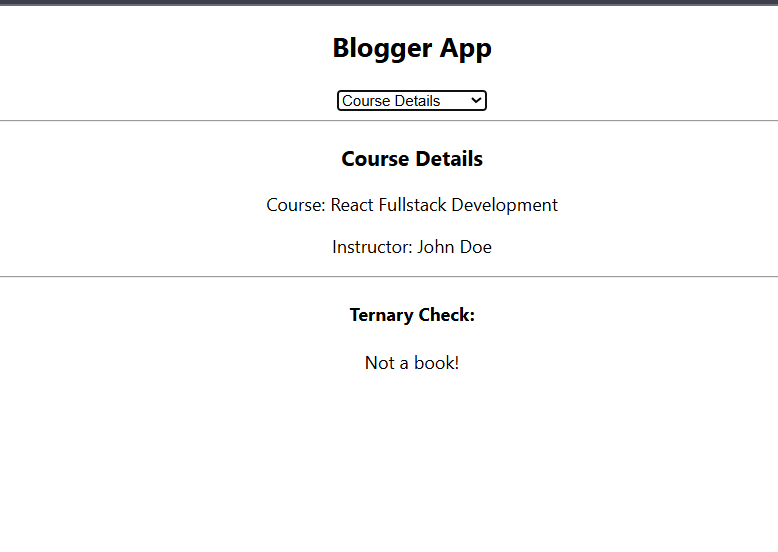




**Output:**

****

****

****