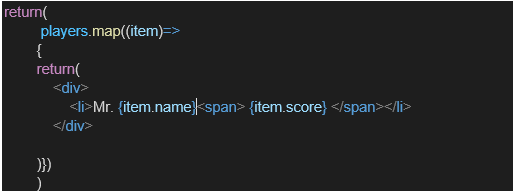
WEEK 7 Hands On solutions:-

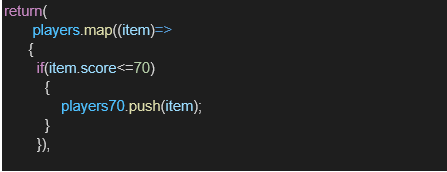
**9. Create a React Application named “cricketapp” with the following components:**

1. ListofPlayers

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

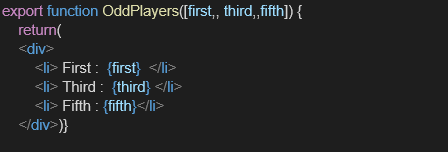


· Filter the players with scores below 70 using arrow functions of ES6.

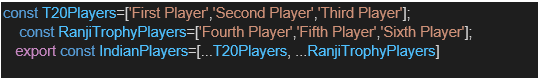


2. IndianPlayers

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



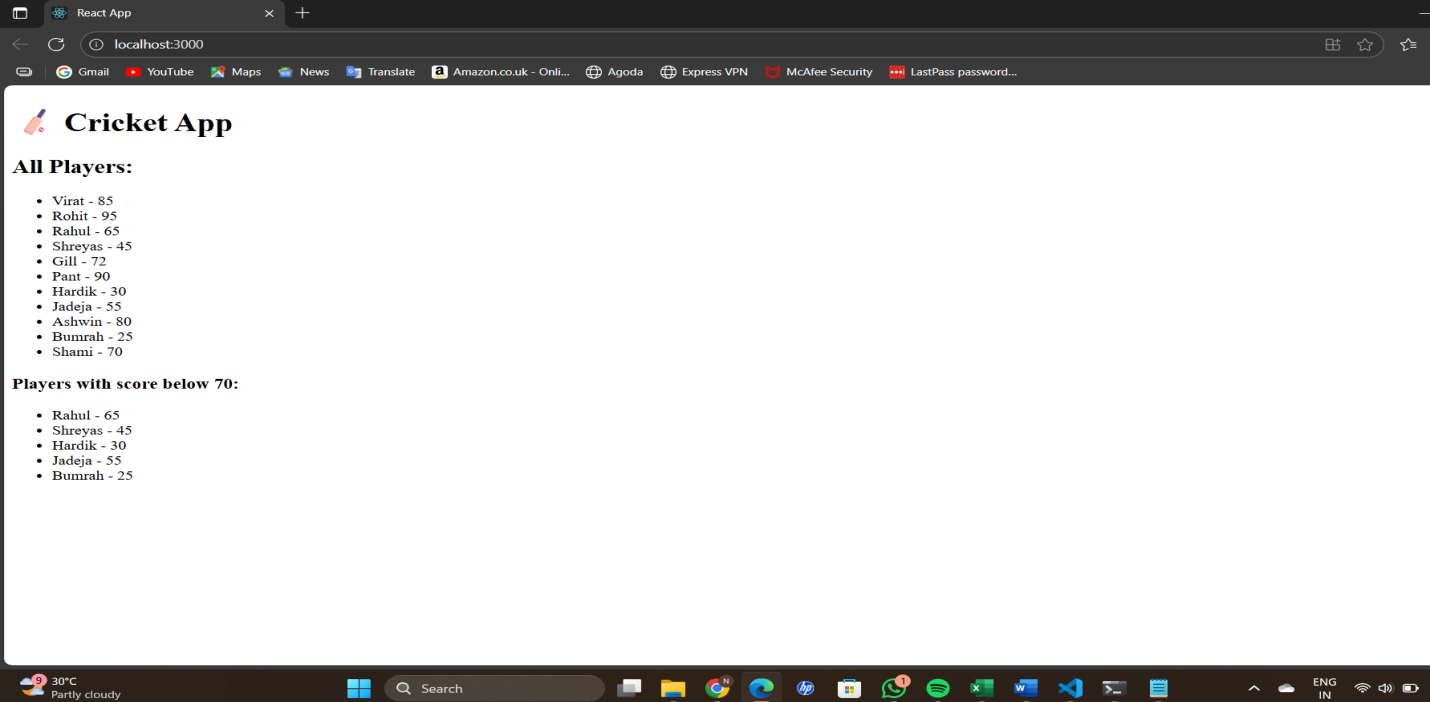
b. Declare two arrays T20players and RanjiTrophy players and merge the two arrays and display them using the Merge feature of ES6



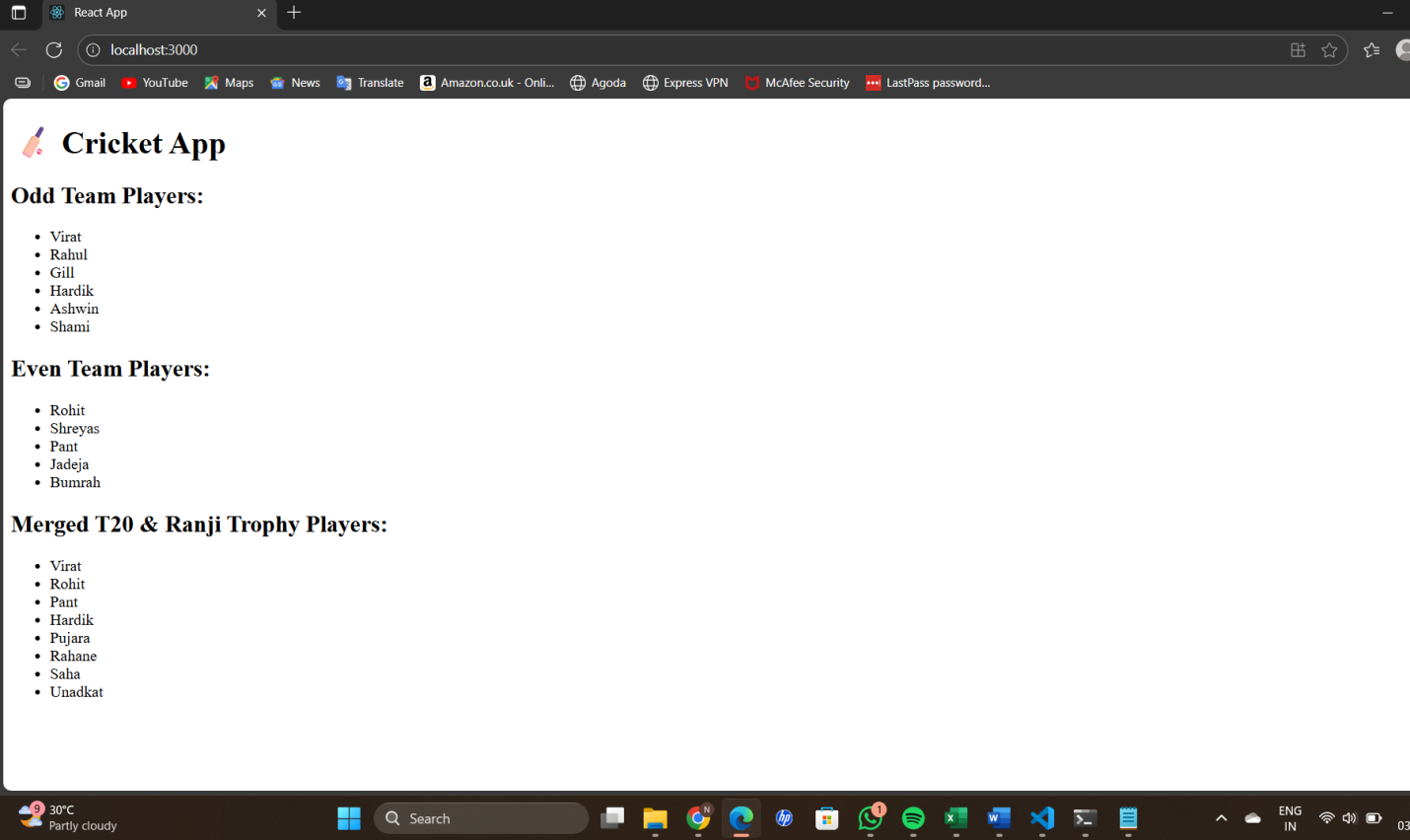
Display these two components in the same home page using a simple if else in the flag variable.

**Output:**

When Flag=true



When Flag=false



**10. React JS-HOL Objectives:-**

**1. Define JSX:-** JSX (JavaScript XML) is a syntax extension for JavaScript used in React. It allows you to write HTML-like code inside JavaScript, which makes it easier to create and visualize UI components.

2. **Explain ECMA Script:-** ECMA Script is the standardized version of JavaScript. The most commonly used version is ES6 (ECMAScript 2015) and newer versions, which introduced features like:

* let and const
* Arrow functions
* Classes
* Modules
* Promises
* Destructuring

React applications often use ES6+ features.

#### **3. Explain React.createElement():-**This is the core method React uses under the hood to create virtual DOM nodes.

**Syntax:-** React.createElement(type, props, ...children)

**Example:-**React.createElement('h1', { className: 'greeting' }, 'Hello, world!');

#### 4. **Explain how to create React nodes with JSX**

React nodes are created by writing JSX elements. JSX gets transpiled into React.createElement() calls.

**Example:-** const element = <p>This is a paragraph node in React.</p>;

#### **5. Define how to render JSX to the DOM:-**To display JSX in the browser, use ReactDOM.render() (or createRoot in React 18+):

**Example:-** import React from 'react';

import ReactDOM from 'react-dom/client';

const element = <h1>Hello React</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(element);

#### **6. Explain how to use JavaScript expressions in JSX:-**You can embed any valid JavaScript expression in JSX using curly braces {}.

**Example:-** const name = 'Spandana';

const element = <h1>Hello, {name}!</h1>;

#### **7. Explain how to use inline CSS in JSX:-**You can apply styles directly using the style attribute as an object.

**Example:-c**onst styleObj = {

color: 'blue',

fontSize: '20px',

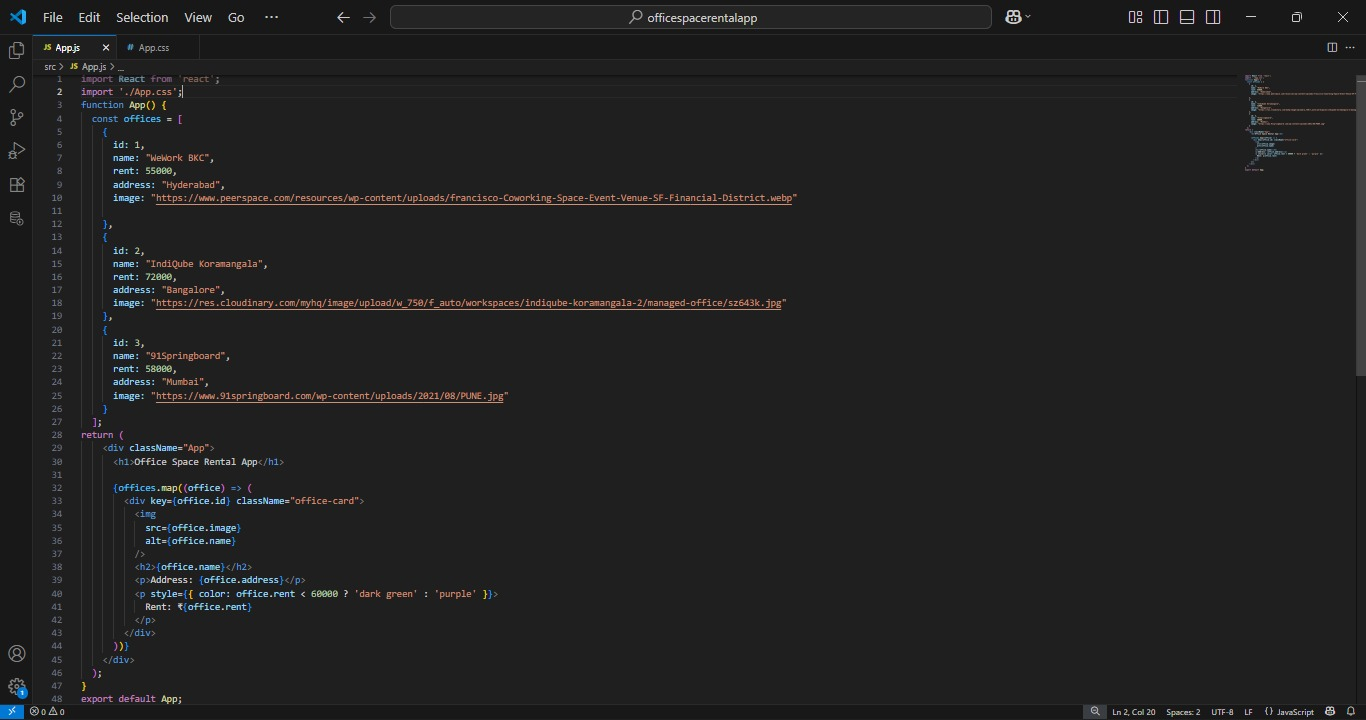
backgroundColor: 'lightgray',

};

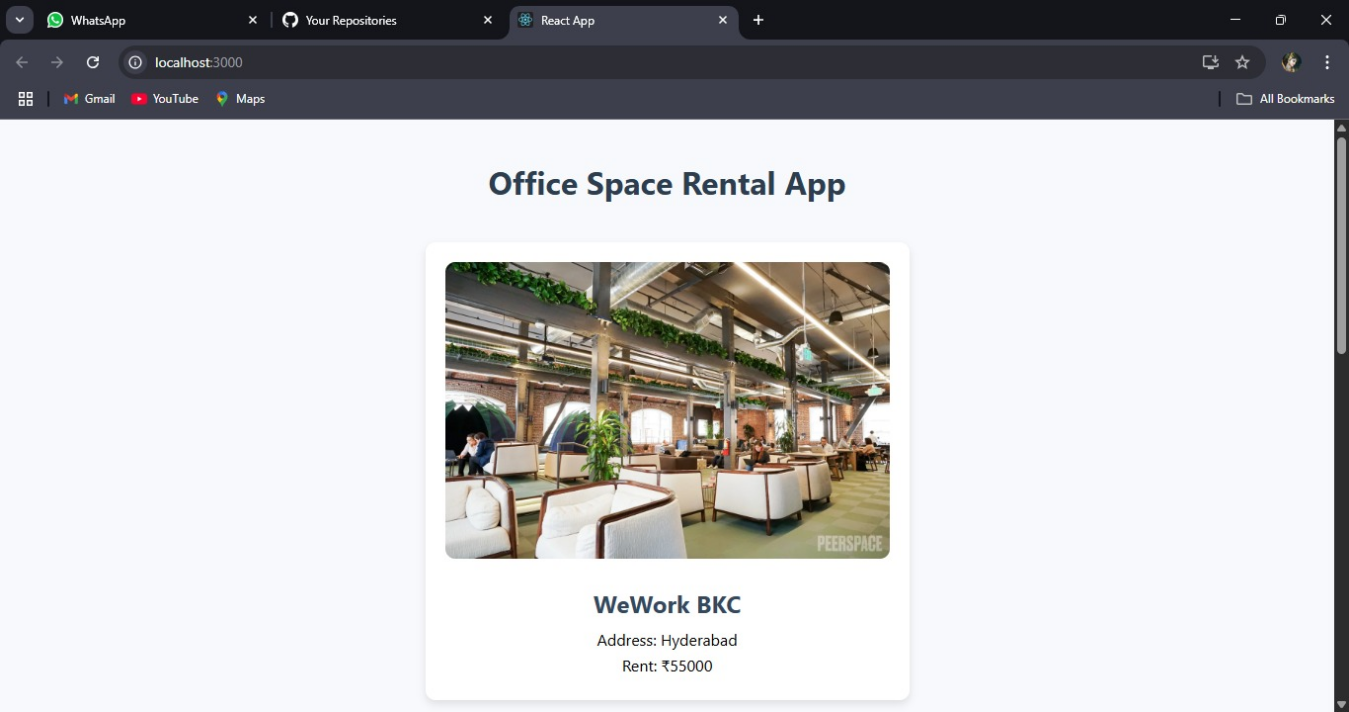
const element = <h2 style={styleObj}>Styled Text</h2>;

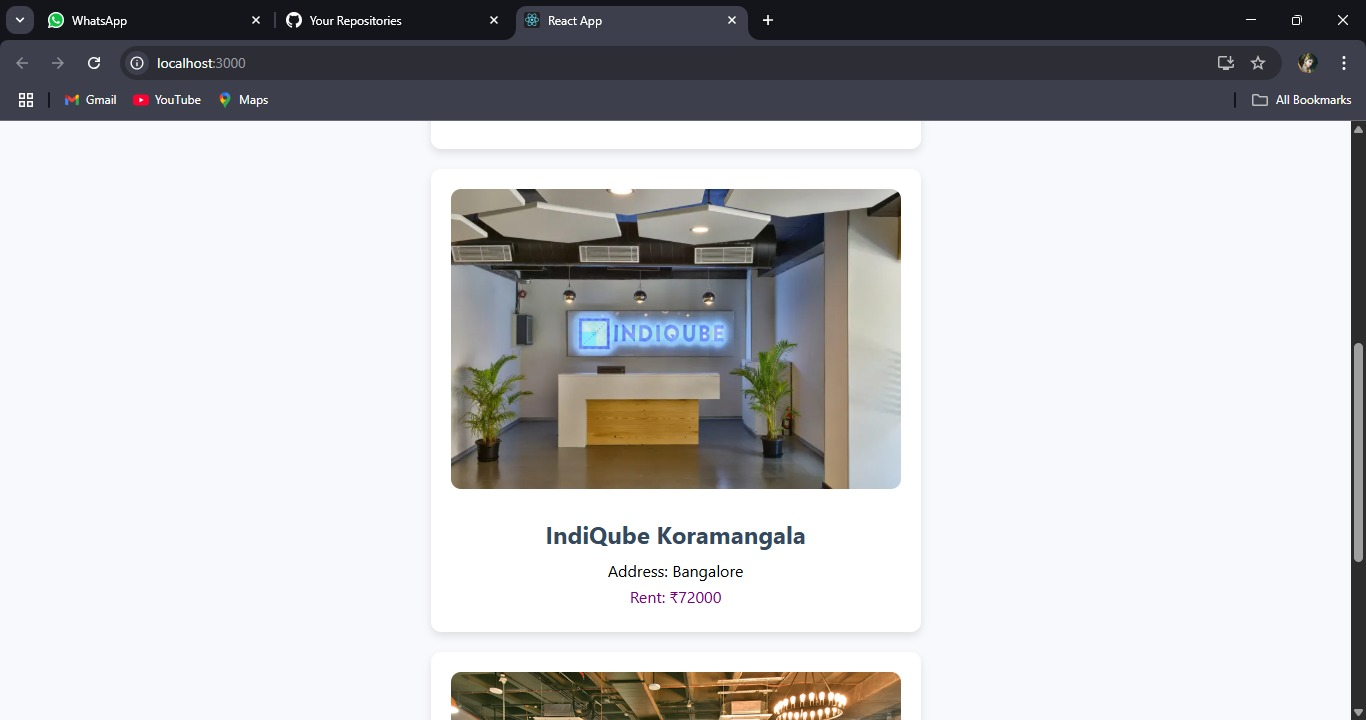
* **Create a React Application named “officespacerentalapp” which uses React JSX to create elements, attributes and renders DOM to display the page.**

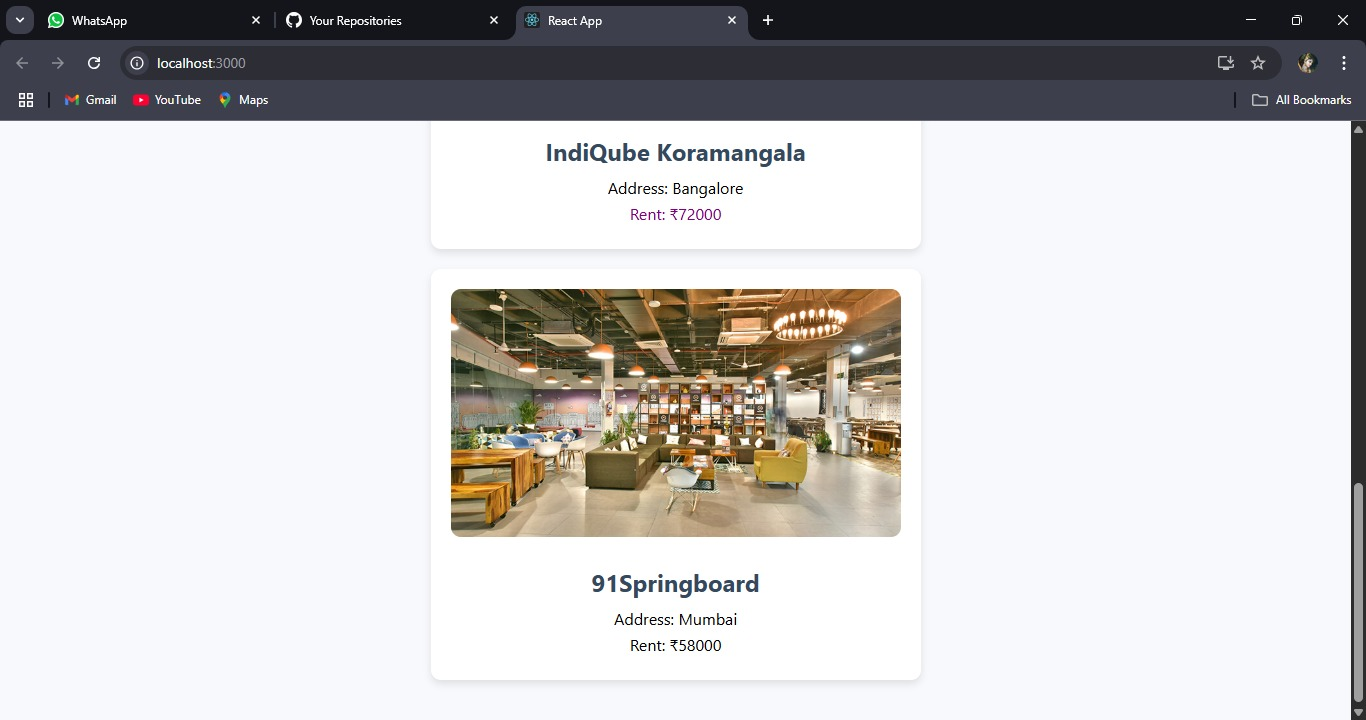
**Code:-**

****

**OUTPUT:**-







**11. ReactJs-HQL :-**

**1. Explain React Events:-** React events are how you handle user interactions like clicks, form submissions, typing, etc., in React applications. These events are very similar to DOM events but are wrapped by React’s Synthetic Event system to ensure consistent behavior across all browsers.

#### **2. Explain Event Handlers:-**Event handlers are functions that are triggered in response to an event, such as onClick, onChange, onSubmit, etc.

Example:-function handleClick() {

alert('Button was clicked!');

}

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

#### **3. Define Synthetic Event:-**A **Synthetic Event** is a **cross-browser wrapper** around the native browser event. It has the same interface as the browser’s native event but works identically across all browsers.React uses SyntheticEvent to improve performance and consistency.

#### **4. Identify React Event Naming Convention:-** React follows camelCase naming for event handlers, unlike HTML where it's lowercase.

|  |  |
| --- | --- |
| **In HTML** | **In React** |

|  |  |
| --- | --- |
| onclick | onClick |

|  |  |
| --- | --- |
| onchange | onChange |

|  |  |
| --- | --- |
| onsubmit | onSubmit |

|  |  |
| --- | --- |
| onmouseover | onMouseOver |

<!-- HTML -->

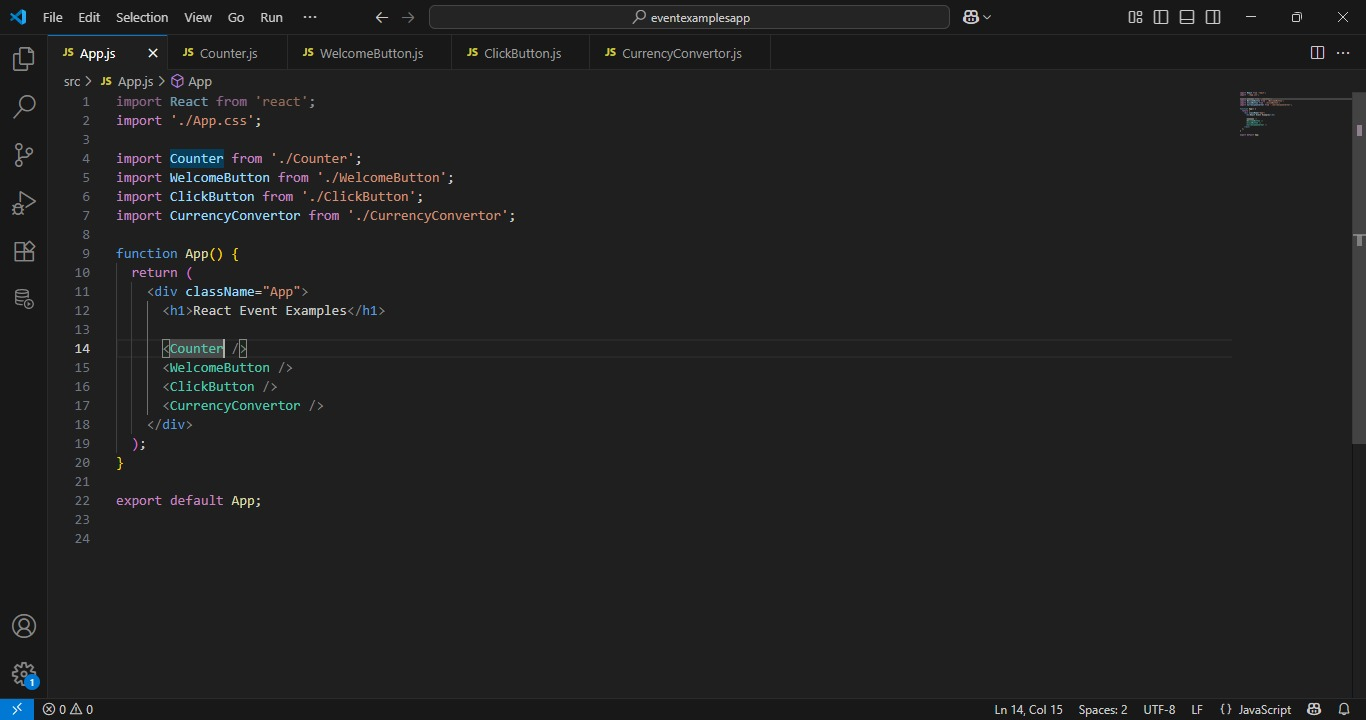
<button onclick="doSomething()">Click</button>

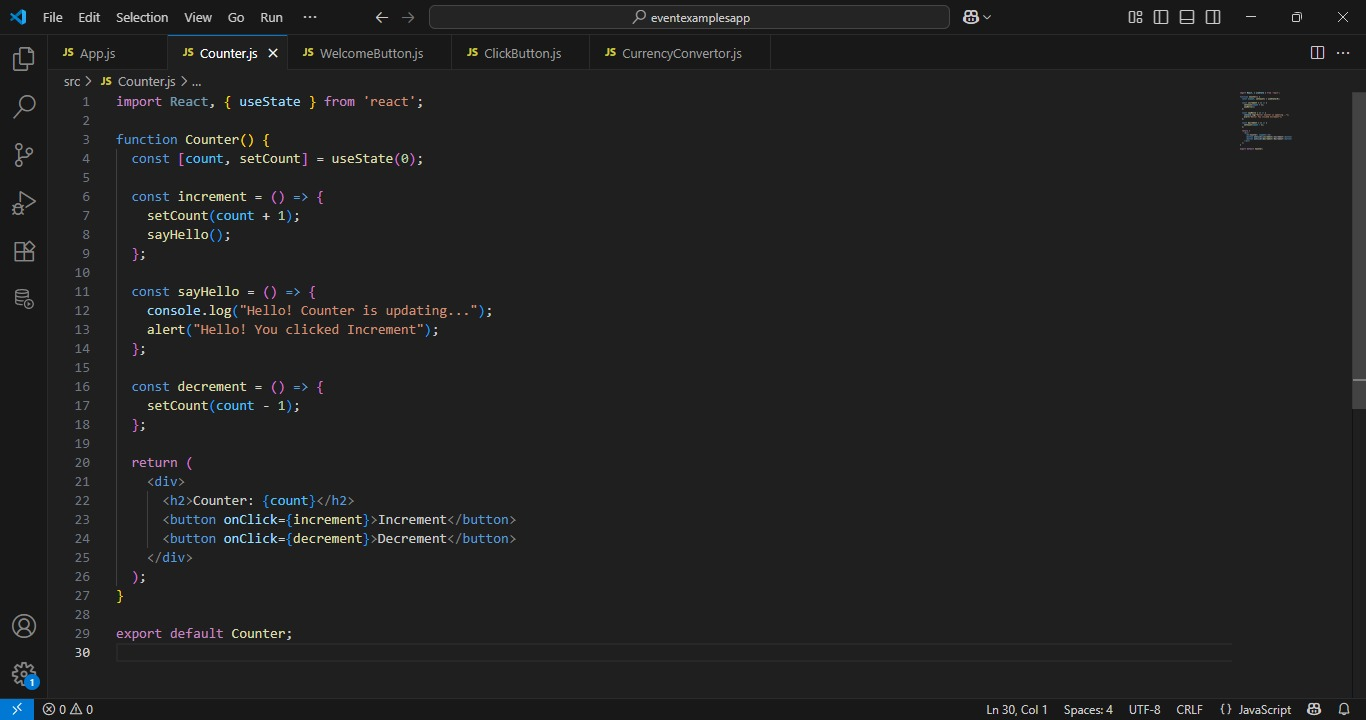
// React

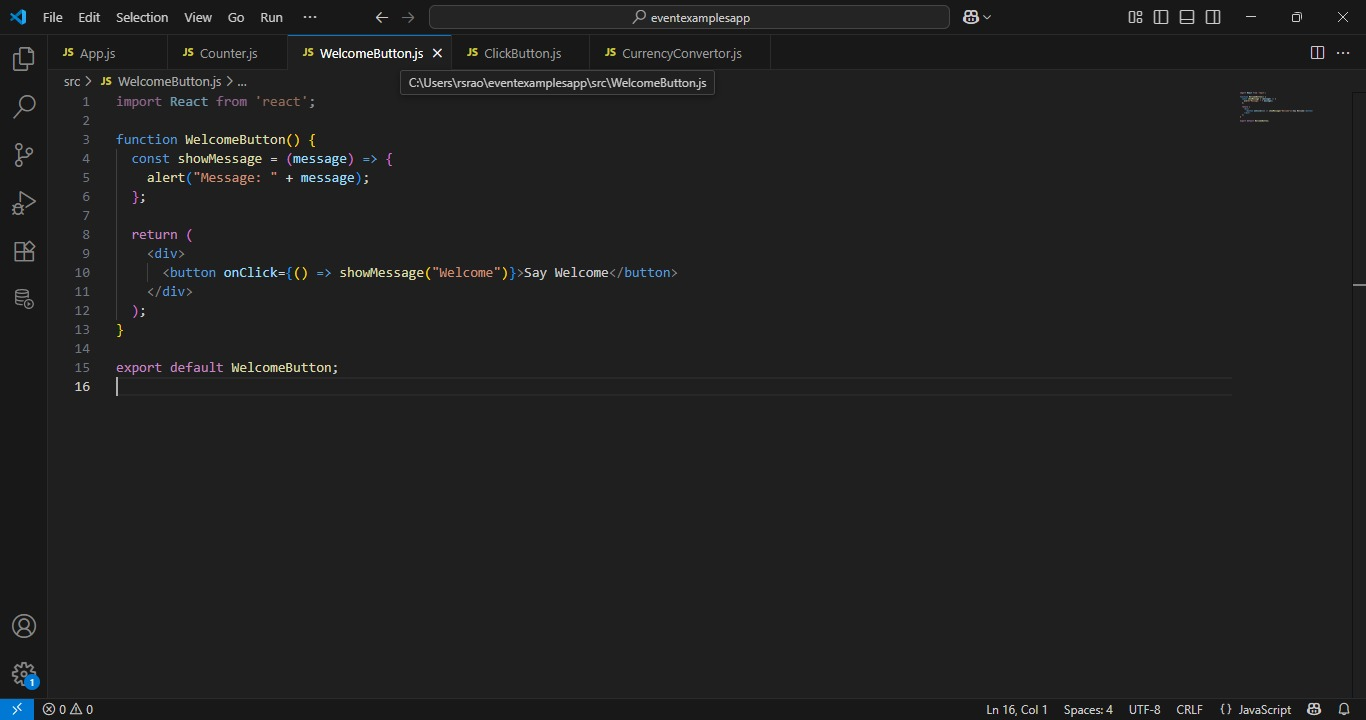
<button onClick={doSomething}>Click</button>

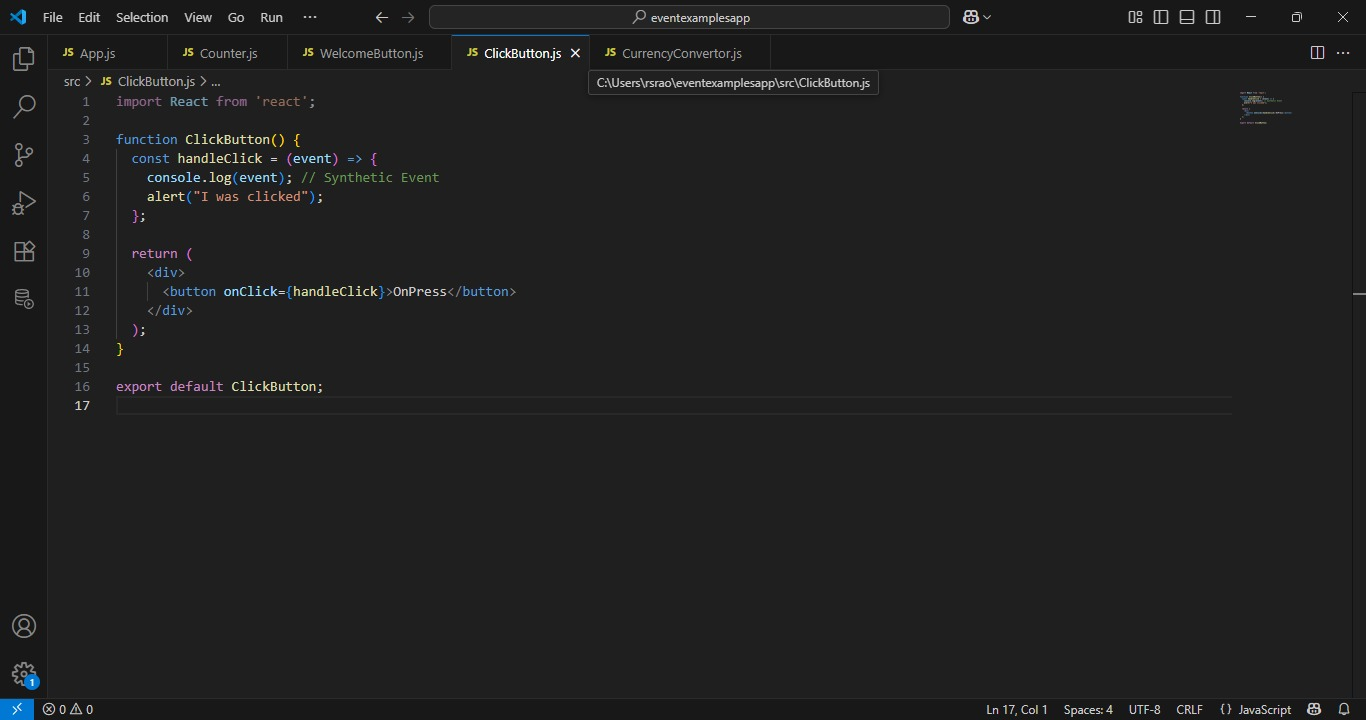
* **Create a React Application “eventexamplesapp” to handle various events of the form elements in HTML.**

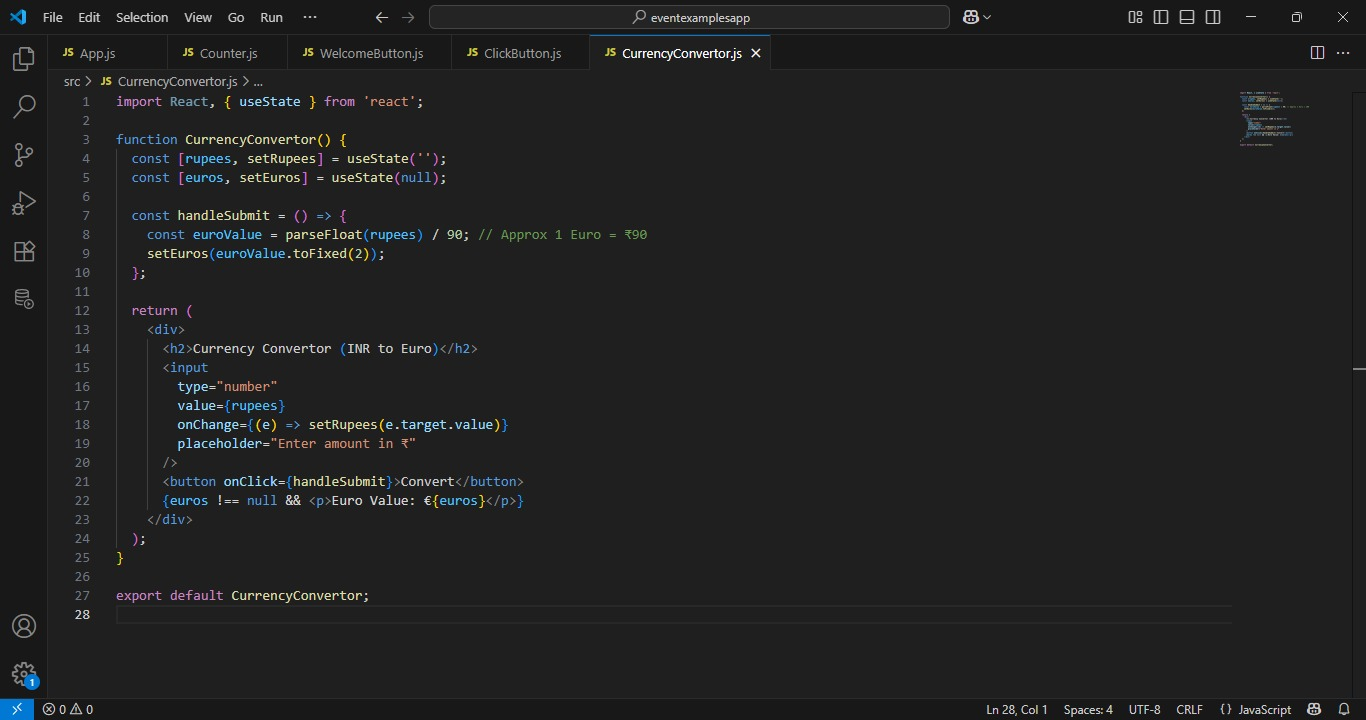
**CODE:-**

****

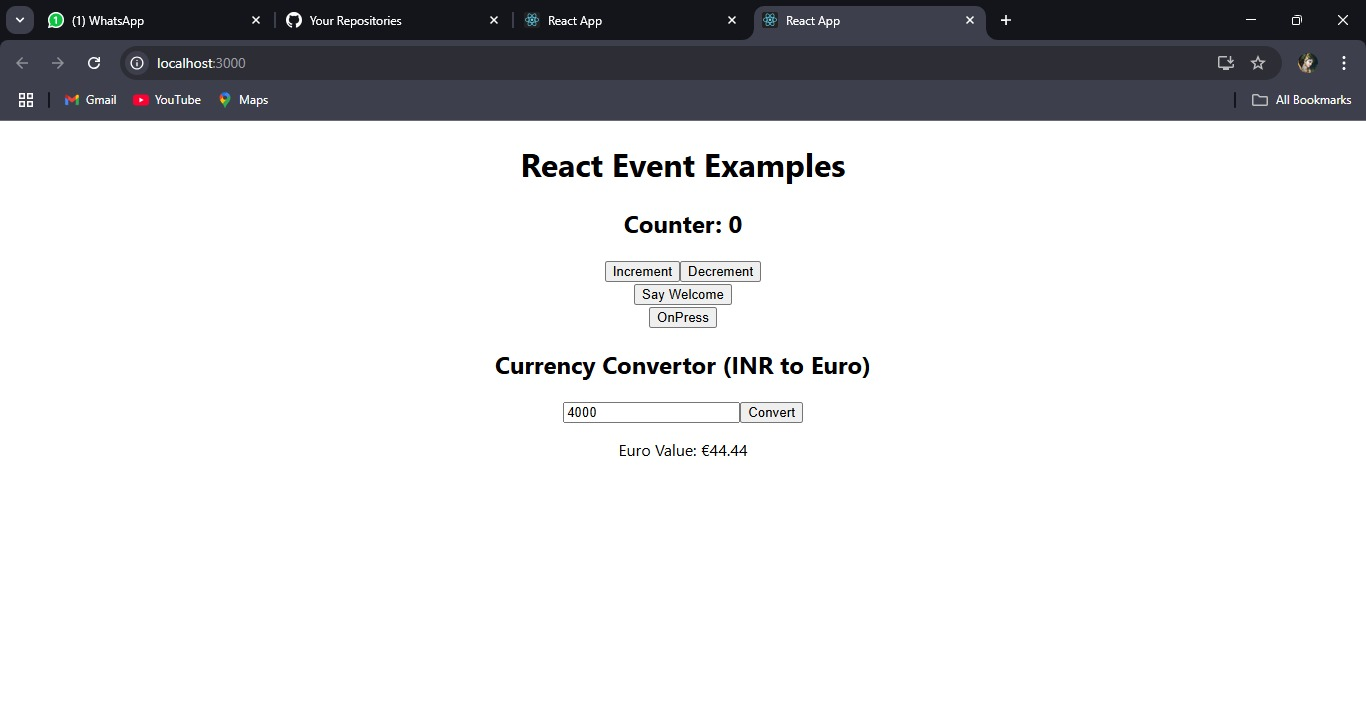
****

****

****

****

**OUTPUT:-**

****

**12. ReactJs-HQL:-**

#### **1. Explain about Conditional Rendering in React**

Conditional rendering in React allows you to show or hide parts of the UI based on conditions (like user actions or state). It's just like using conditions in JavaScript.

**Example:-** if (isLoggedIn) {

return <p>Welcome back!</p>;

} else {

return <p>Please log in</p>;

}

#### **2. Define Element Variables**

Element variables are used to store JSX elements in variables, allowing flexible conditional rendering.

**Example:-** let message;

if (isLoggedIn) {

message = <p>Welcome, Spandana!</p>;

} else {

message = <p>Please log in.</p>;

}

return <div>{message}</div>;

3. **Explain How to Prevent Components from Rendering**

* **Return null from the component:**

function WarningBanner({ show }) {

if (!show) {

return null; // Component will not render anything

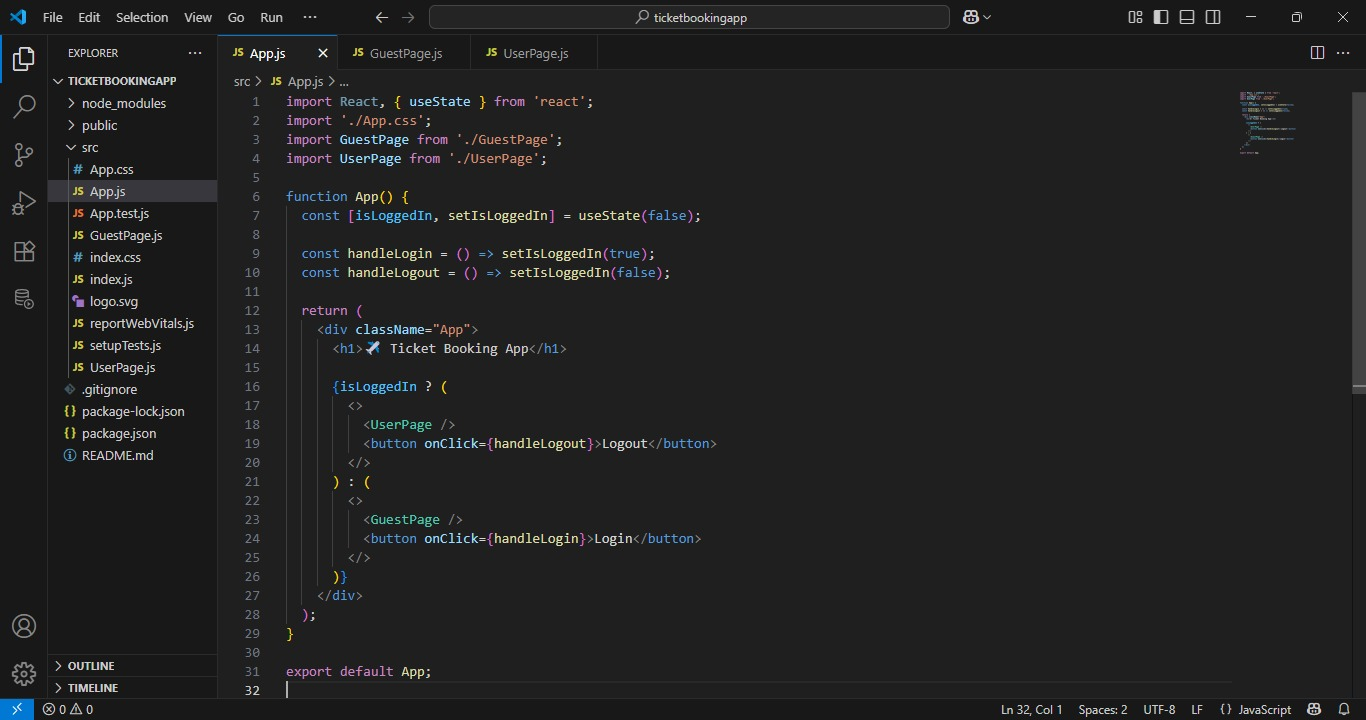
}

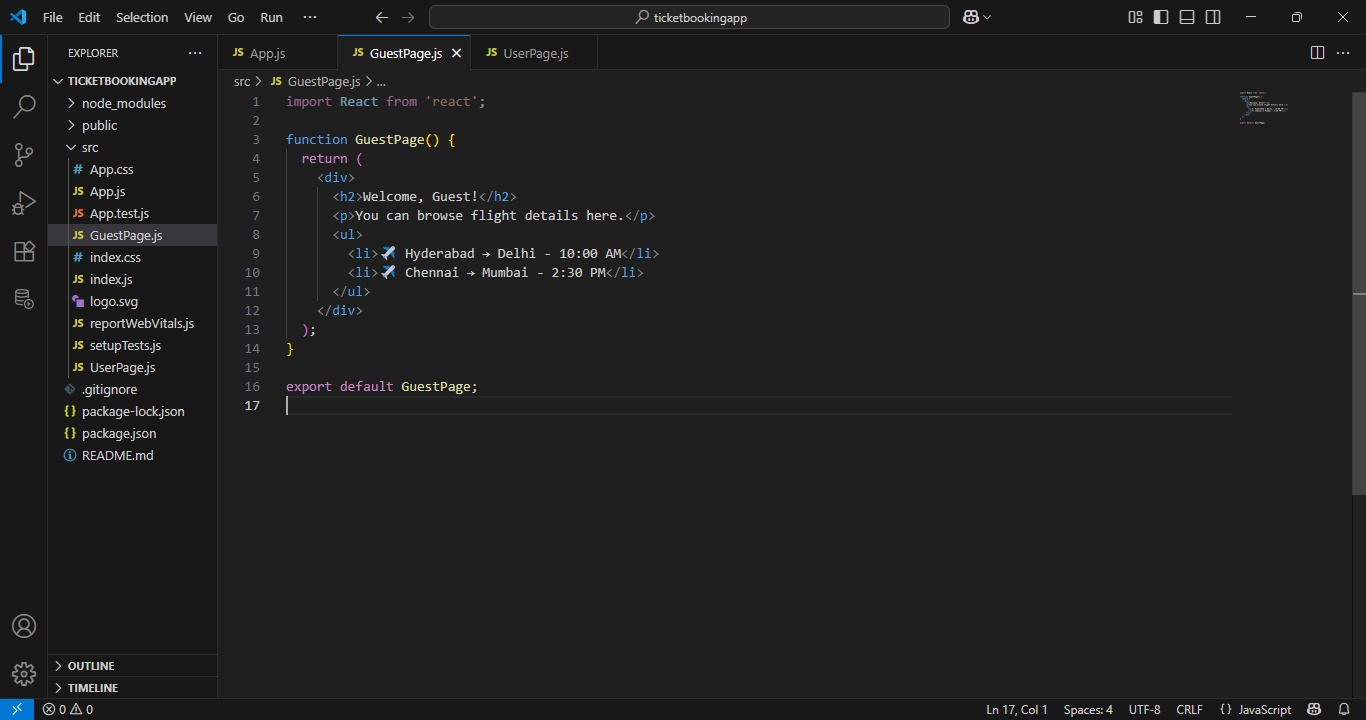
return <div className="warning">Warning!</div>;

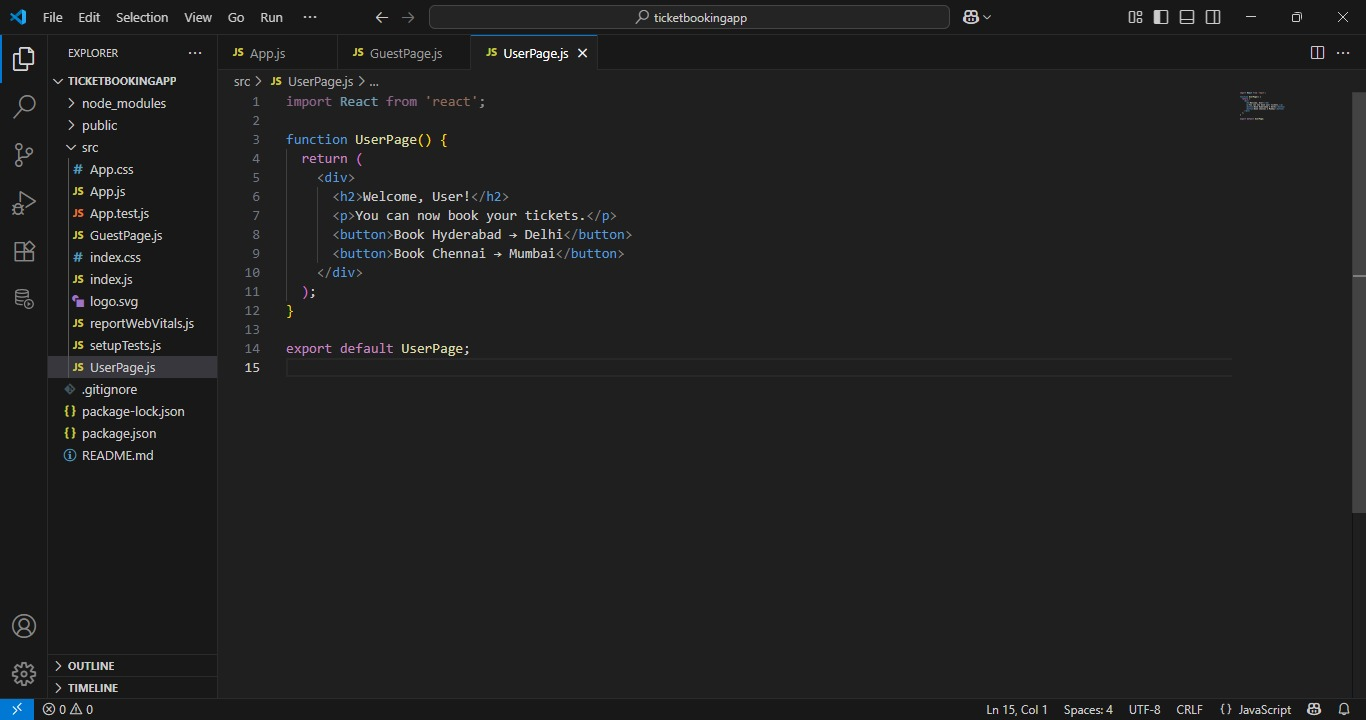
}

* **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.**

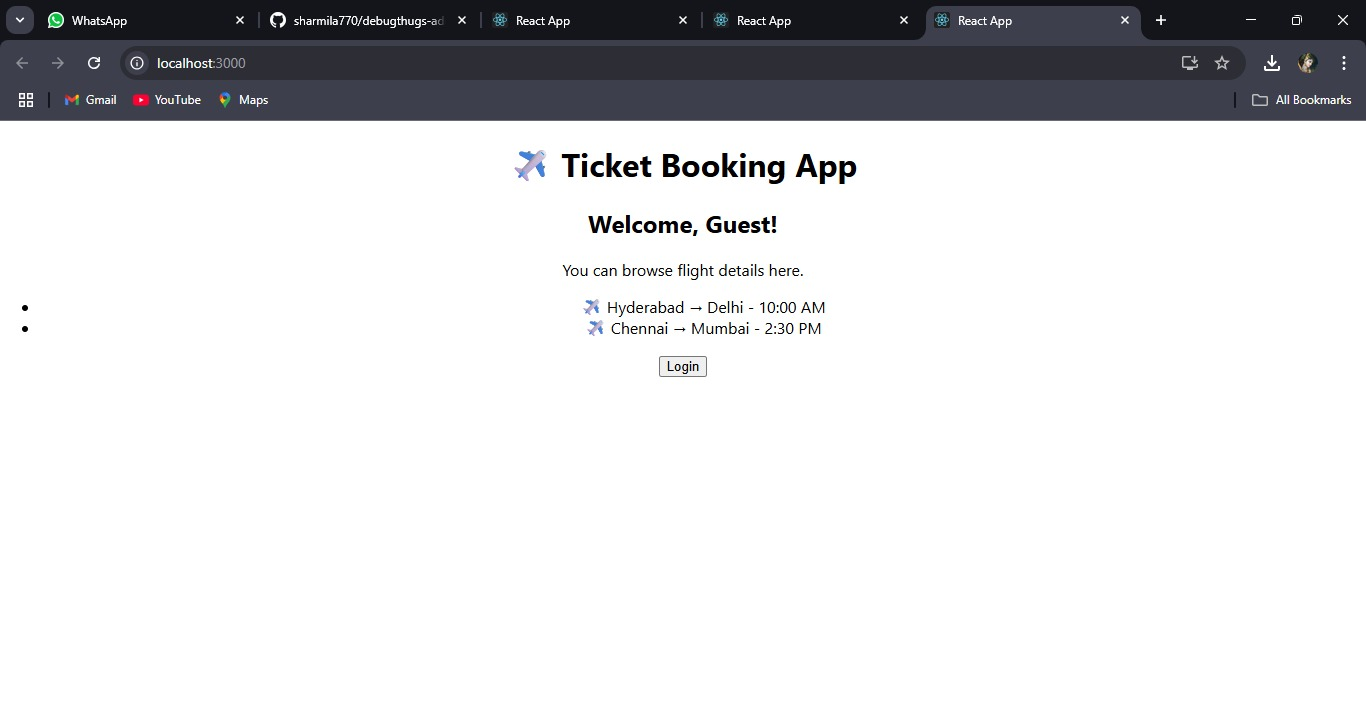
**CODE:-**

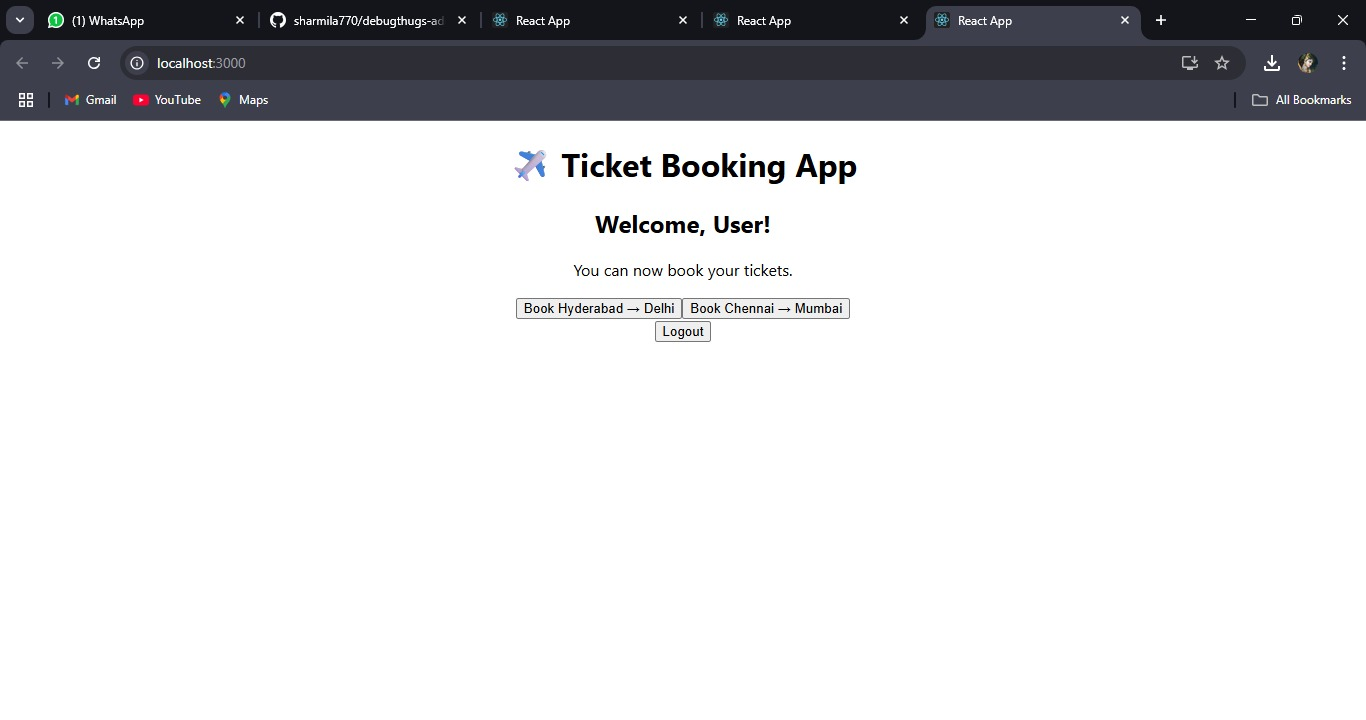
****

****

****

**OUTPUT:-**

****

****

**13.ReactJs-HQL:-**

#### **1. Explain Various Ways of Conditional Rendering**

In React, there are several ways to render content conditionally:

* Using if-else:

if (isLoggedIn) {

return <Welcome />;

} else {

return <Login />;

}

* Using Logical AND (&&):

{hasAccess && <AdminPanel />}

* Using Switch-Case (inside functions):

switch (role) {

case 'admin':

return <Admin />;

case 'user':

return <User />;

default:

return <Guest />;

}

#### **2. Explain How to Render Multiple Components**

You can render multiple components in one return statement using:

* Fragments:

<>

<Header />

<Sidebar />

<Footer />

</>

* React.Fragment:

<React.Fragment>

<Header />

<Sidebar />

</React.Fragment>

* **Divs or wrappers:**

<div>

<Comp1 />

<Comp2 />

</div>

#### **3. Define List Component**

A **List component** is a React component that renders a list of items (like users, messages, or products), usually by mapping over an array.

#### **4. Explain About Keys in React Applications**

**Keys** help React identify which items in a list have changed, are added, or are removed. They should be **unique** and **stable** (not change across renders).

**Why are keys important?**

* Help React optimize rendering.
* Avoid re-rendering all items unnecessarily.

#### **5. Explain How to Extract Components with Keys**

When mapping over data, it's good practice to extract individual components and assign key when rendering them in a list.

#### **6. Explain React map() Function**

In React, map() is used to loop over arrays and return JSX elements.

Example:-

const fruits = ['Apple', 'Banana', 'Mango'];

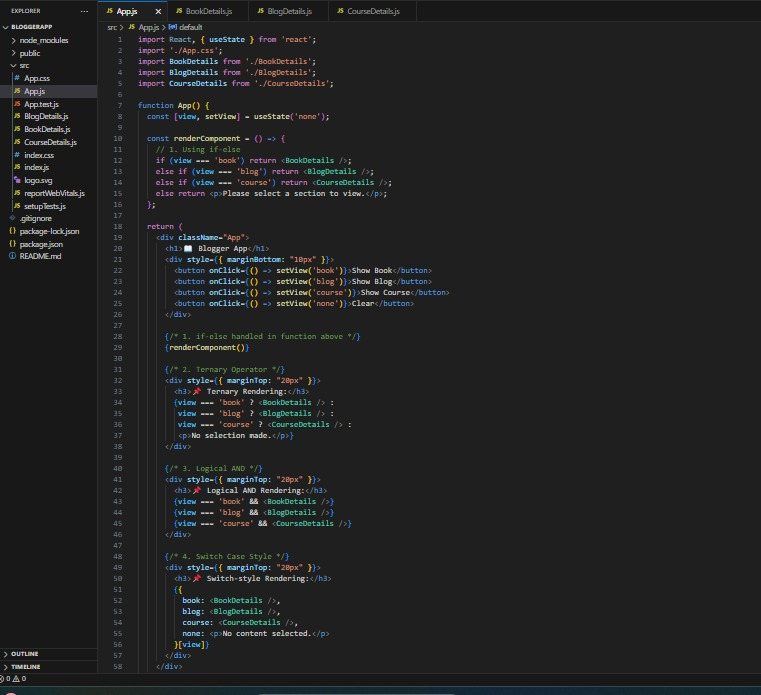
const fruitList = fruits.map((fruit, index) => (

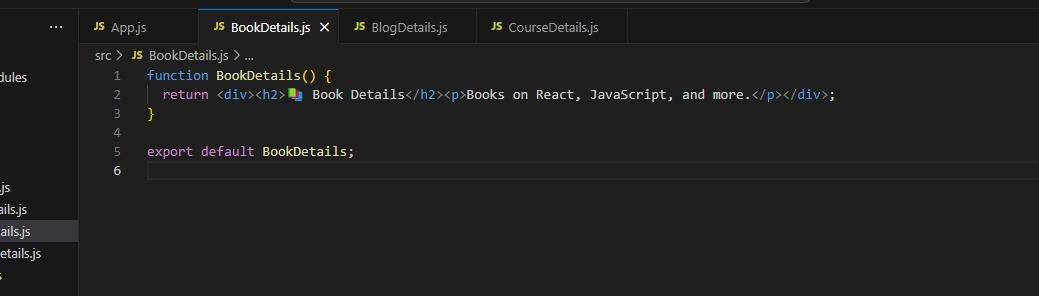
<li key={index}>{fruit}</li>

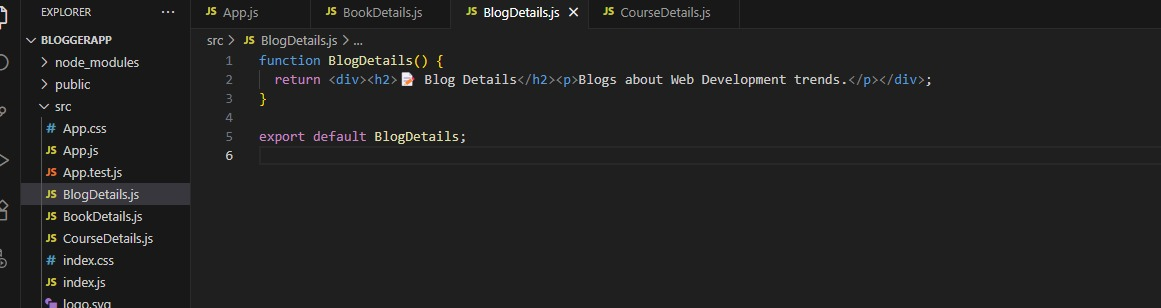
));

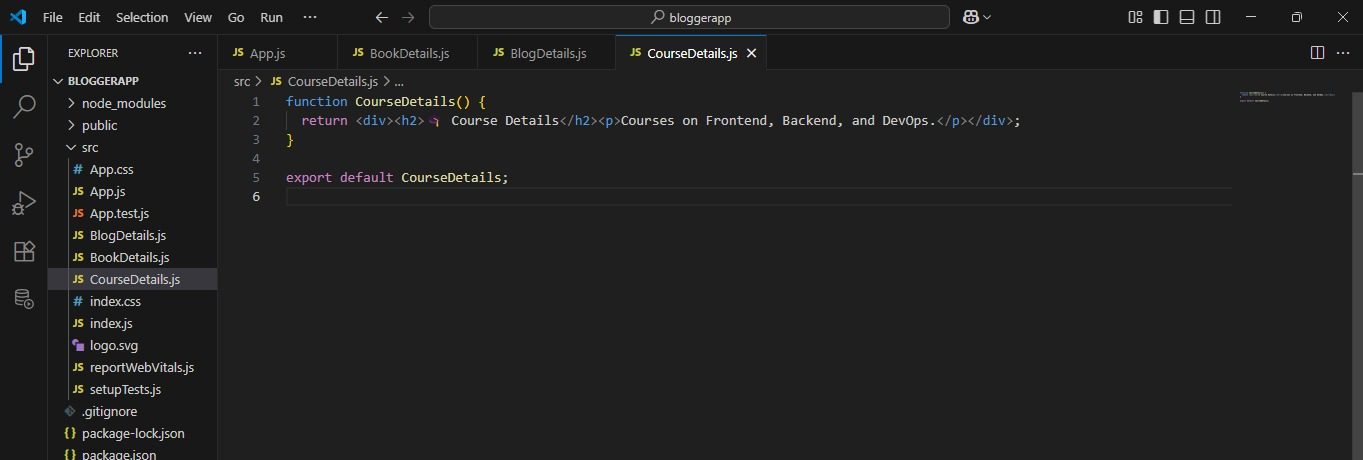
* **Create a React App named “bloggerapp” with 3 components.** 
  + **1. Book Details**
  + **2. Blog Details**
  + **3. Course Details**

**CODE:-**

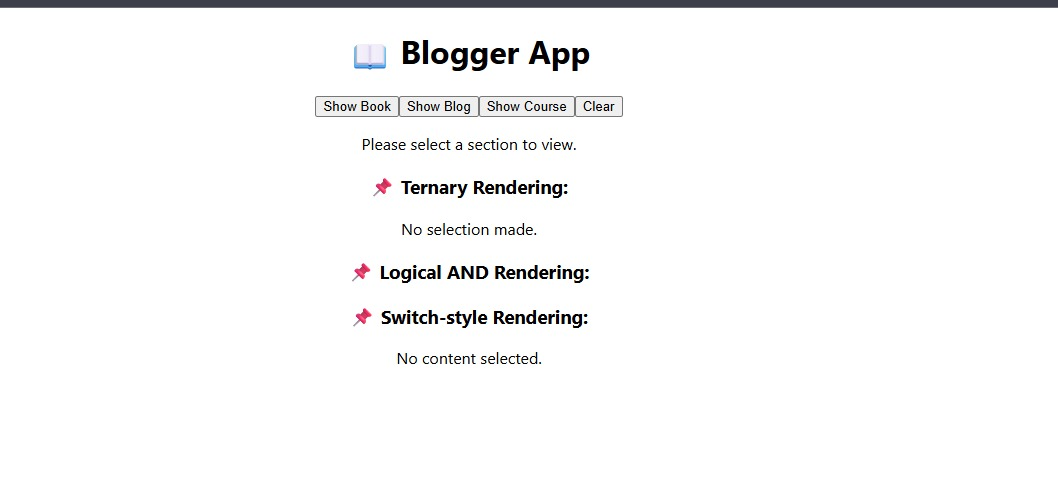
****

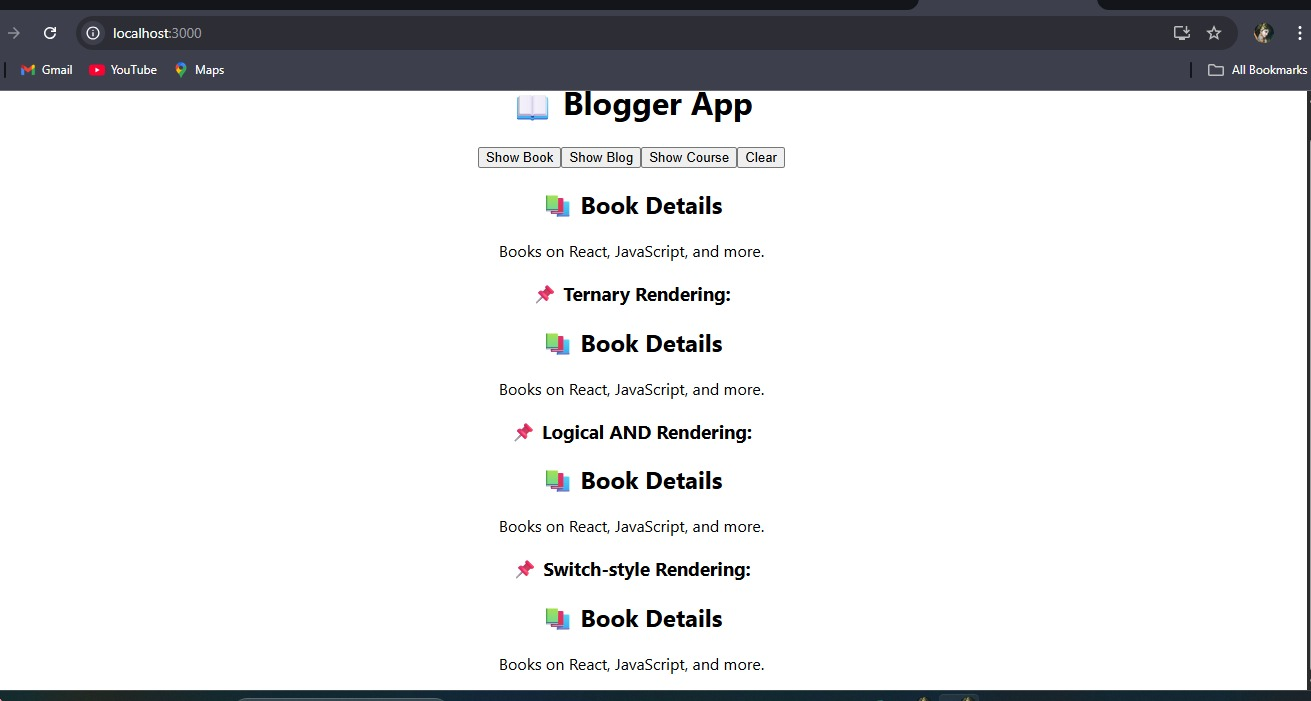
****

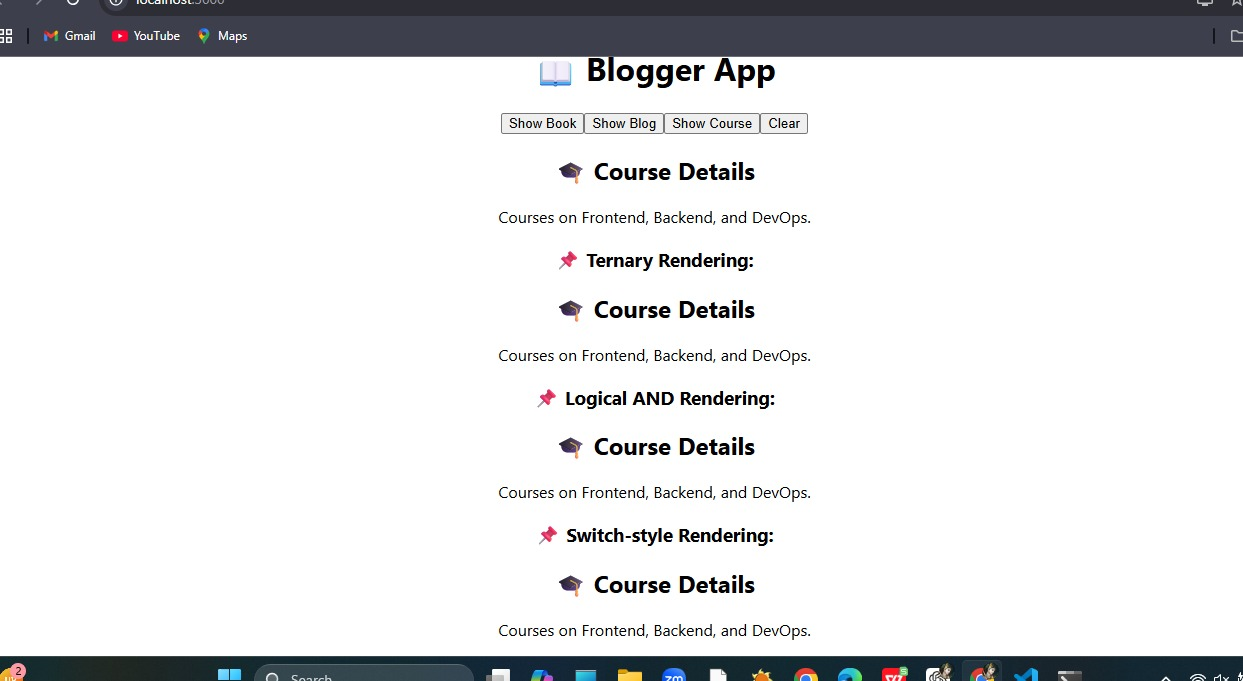
****

****

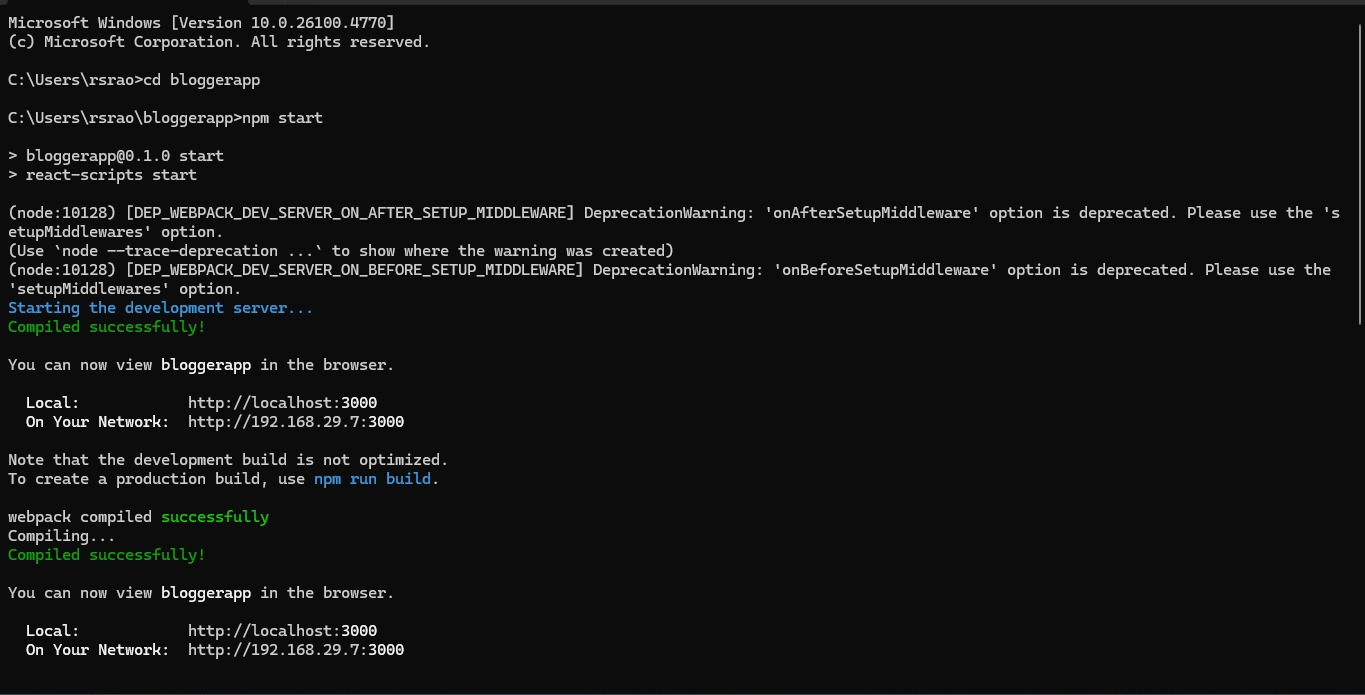
**OUTPUT:-**

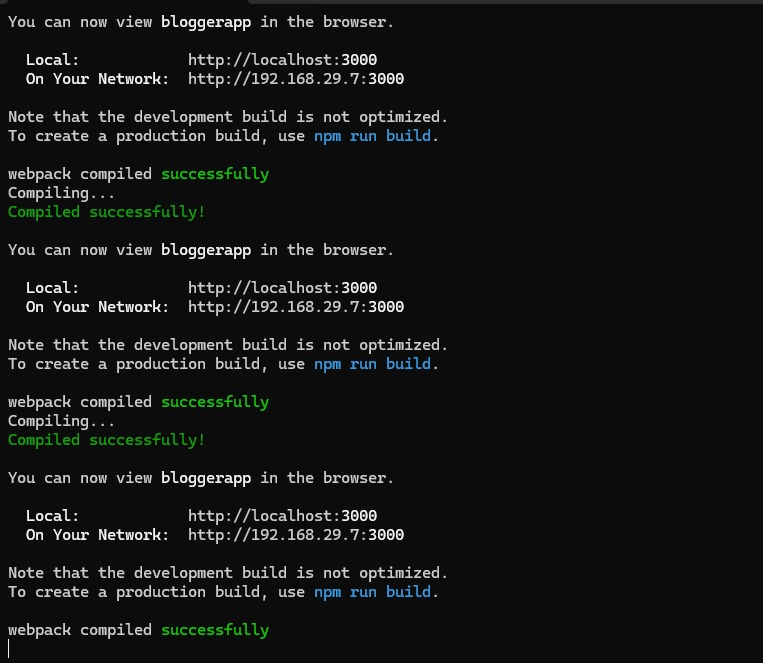
****

****

****

**CMD:-**

****

****