**Week-7 Hands-On(React)**

9.ReactJS-HOL:

Objective:

**List the features of ES6**

ES6, also known as ECMAScript 2015, introduced many powerful features to modern JavaScript. These include the block-scoped let and const keywords, arrow functions for concise function expressions, classes and inheritance for object-oriented programming, template literals for easier string interpolation, default function parameters, destructuring assignment, rest and spread operators, modules (import and export) for modular code, promises for better asynchronous programming, and new data structures like Map and Set. Together, these features help developers write cleaner, more maintainable, and expressive code.

**Explain JavaScript let**

The let keyword in JavaScript is used to declare variables that are limited in scope to the block, statement, or expression where they are defined. Unlike var, which is function-scoped, let is block-scoped, meaning variables declared with let only exist within the nearest set of curly braces { }. This helps avoid common bugs caused by variable redeclaration or unintentional overwriting, and it prevents variables from being hoisted and initialized with undefined before their declaration.

**Identify the differences between var and let**

The main differences between var and let relate to their scope, hoisting behavior, and redeclaration rules. var is function-scoped, meaning it’s accessible anywhere within the function where it is declared, and it is hoisted to the top of its scope and initialized with undefined. This can lead to unexpected behavior if you try to use the variable before declaration. On the other hand, let is block-scoped, existing only within the nearest set of curly braces, and while it is also hoisted, it is not initialized until its declaration is evaluated, leading to a "temporal dead zone" if accessed early. Additionally, variables declared with let cannot be redeclared within the same scope, which further helps reduce bugs.

**Explain JavaScript const**

The const keyword is similar to let in that it creates block-scoped variables, but with the added constraint that the variable must be assigned a value at the time of declaration and cannot be reassigned later. However, it’s important to note that if the value is an object or array, the contents of that object or array can still be modified (such as adding or changing properties or elements). const is typically used when you want to declare constants whose bindings should remain fixed throughout the code.

**Explain ES6 class fundamentals**

ES6 introduced the class syntax, providing a clearer and more structured way to create objects and handle inheritance. A class in JavaScript is essentially a blueprint for creating objects and can include a constructor method to initialize object properties, along with other methods defined within the class body. This syntax is syntactic sugar over the traditional prototype-based inheritance but makes the code easier to read and understand. For example, a Person class can define properties like name and methods like greet to make objects based on this blueprint.

**Explain ES6 class inheritance**

ES6 makes inheritance straightforward through the use of the extends keyword. A subclass can extend a parent class and inherit its properties and methods. Within the subclass constructor, the super() function must be called to execute the parent class's constructor before accessing this. This allows the subclass to build on and customize the behavior of its parent while still leveraging shared functionality. This model supports building more complex and reusable object-oriented designs.

**Define ES6 arrow functions**

Arrow functions, introduced in ES6, provide a shorter syntax to write function expressions using the => syntax. They are especially useful for concise callbacks or simple computations. Importantly, arrow functions do not have their own this binding; instead, they inherit this from their surrounding context, which makes them very handy when working inside class methods or callbacks. Arrow functions also do not have their own arguments object.

**Identify Set() and Map()**

ES6 introduced two useful data structures: Set and Map. A Set is a collection of unique values, meaning no duplicate elements are allowed, which makes it ideal for storing lists where uniqueness matters. A Map is a collection of key-value pairs where keys can be of any data type, unlike plain JavaScript objects that only allow strings or symbols as keys. Map preserves the insertion order of items and offers methods like set, get, and has for managing its entries.

**ListOfPlayers.js**

import React from 'react';

function ListOfPlayers() {

const players = [

{ name: "Virat", score: 90 },

{ name: "Rohit", score: 85 },

{ name: "Dhoni", score: 60 },

{ name: "Pant", score: 40 },

{ name: "Kohli", score: 75 },

{ name: "Hardik", score: 55 },

{ name: "Rahul", score: 70 },

{ name: "Shami", score: 30 },

{ name: "Bumrah", score: 80 },

{ name: "Gill", score: 50 },

{ name: "Siraj", score: 95 }

];

const filtered = players.filter(p => p.score < 70); // ES6 arrow function

return (

<div>

<h2>All Players</h2>

{players.map((player, index) => (

<p key={index}>{player.name} - {player.score}</p>

))}

<h3>Filtered (Score &lt; 70)</h3>

{filtered.map((player, index) => (

<p key={index}>{player.name} - {player.score}</p>

))}

</div>

);

}

export default ListOfPlayers;

**IndianPlayers.js**

import React from 'react';

function IndianPlayers() {

const allPlayers = ["Virat", "Rohit", "Dhoni", "Pant", "Kohli", "Hardik"];

const oddTeam = allPlayers.filter((\_, i) => i % 2 !== 0);

const evenTeam = allPlayers.filter((\_, i) => i % 2 === 0);

const T20 = ["Ishan", "Surya"];

const Ranji = ["Pujara", "Rahane"];

const merged = [...T20, ...Ranji]; // Spread operator

return (

<div>

<h2>Odd Team Players</h2>

{oddTeam.map((player, index) => (

<p key={index}>{player}</p>

))}

<h2>Even Team Players</h2>

{evenTeam.map((player, index) => (

<p key={index}>{player}</p>

))}

<h2>Merged Team (T20 + Ranji)</h2>

{merged.map((player, index) => (

<p key={index}>{player}</p>

))}

</div>

);

}

export default IndianPlayers;

**App.js**

import React from 'react';

import ListOfPlayers from './ListOfPlayers';

import IndianPlayers from './IndianPlayers';

function App() {

const flag = true; // Change to false to test both components

return (

<div className="App">

<h1>Cricket App Dashboard</h1>

{flag ? <ListOfPlayers /> : <IndianPlayers />}

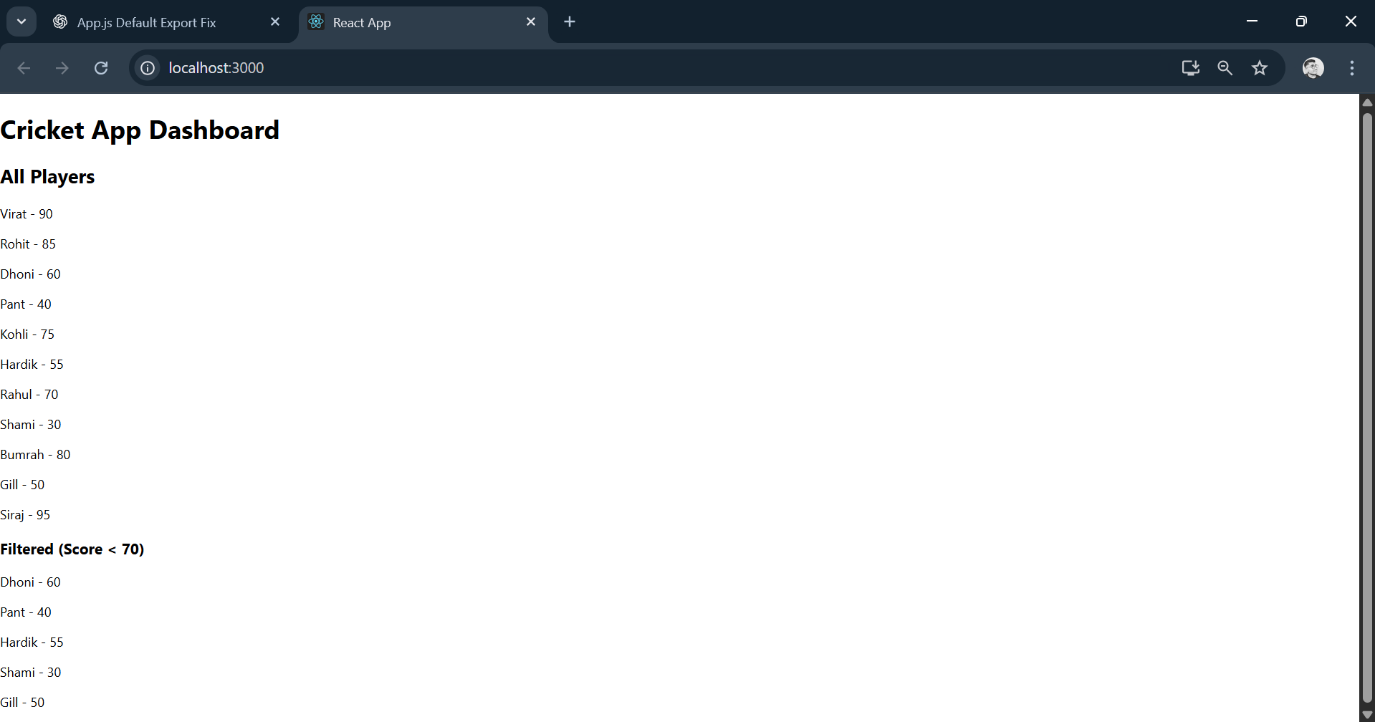
</div>

);

}

export default App;

Output:



10.ReactJS-HOL:

Objective:

**Define JSX**

JSX stands for **JavaScript XML**. It is a syntax extension for JavaScript used in React to describe what the UI should look like. JSX allows developers to write HTML-like code directly within JavaScript, which makes it easier to visualize the component structure. Under the hood, JSX code is transformed into calls to React.createElement(), which then creates JavaScript objects representing DOM elements. This syntax helps make React components easier to read and maintain by closely resembling the final HTML structure.

**Explain about ECMA Script**

ECMAScript (often abbreviated as ES) is the standardized specification that JavaScript is based on. It defines the core language features, syntax, and rules that JavaScript engines implement. Over the years, several versions have been released, such as ES5, ES6 (also known as ES2015), and beyond, each adding new features like classes, modules, arrow functions, async/await, and more. By following ECMAScript standards, JavaScript developers can write code that works consistently across different browsers and environments.

**Explain React.createElement()**

React.createElement() is a core function provided by React that creates and returns a React element (which is a plain JavaScript object representing a DOM node). It takes three main arguments: the type of the element (such as 'div'), optional props (such as attributes or event handlers), and children (which can be text, numbers, or other React elements). For example, React.createElement('h1', { className: 'title' }, 'Hello World') creates an <h1> element with the text content “Hello World.” JSX is syntactic sugar that compiles into these React.createElement() calls.

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

With JSX, React nodes can be created simply by writing HTML-like tags within JavaScript code. For example, instead of manually calling React.createElement('h1', null, 'Welcome'), you can write <h1>Welcome</h1>. JSX supports nesting, dynamic data insertion using curly braces {}, and conditional rendering. These JSX nodes ultimately represent the structure of the UI that React will render, and they get compiled to React elements by tools like Babel during the build process.

**Define how to render JSX to DOM**

To render JSX to the DOM in a React application, you typically use the ReactDOM.render() method (for React versions before React 18) or createRoot().render() (from React 18 onwards). This method takes a React element (usually created with JSX) and attaches it to a specified DOM node in the HTML, like document.getElementById('root'). For example:

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

This tells React to render the <App /> component and display it inside the element with the ID “root” in the HTML file.

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

In JSX, you can embed JavaScript expressions inside curly braces {} to display dynamic content or evaluate values. For example, you can write <h1>{user.name}</h1> to show the value of user.name, or <p>{2 + 3}</p> to render the result of the expression 2 + 3. These expressions can include variables, function calls, or any valid JavaScript expression, but not full statements like if or for. This makes it easy to mix logic and UI together when building components.

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

In JSX, inline CSS styles are applied by passing an object to the style attribute, where CSS property names are written in camelCase instead of kebab-case. For example, instead of 'font-size', you would use fontSize. The value of each style property is usually a string or number. For example:

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

This applies a blue color and font size of 24 pixels to the heading. Inline styles are useful for dynamic styling based on component state or props.

**App.js**

import React from 'react';

function App() {

const heading = <h1>Office Space Rental Portal</h1>;

const officeImage = "https://via.placeholder.com/300x150?text=Office+Space";

const officeList = [

{ name: "Elite Tower", rent: 75000, address: "Bandra, Mumbai" },

{ name: "WorkHub", rent: 50000, address: "Kondapur, Hyderabad" },

{ name: "Startup Square", rent: 62000, address: "Indiranagar, Bangalore" },

{ name: "BizNest", rent: 45000, address: "Salt Lake, Kolkata" }

];

return (

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

{heading}

<img src={officeImage} alt="Office Space" style={{ width: "300px", marginBottom: "20px" }} />

<h2>Available Office Spaces</h2>

{officeList.map((office, index) => {

const rentStyle = {

color: office.rent < 60000 ? "red" : "green",

fontWeight: "bold"

};

return (

<div key={index} style={{ borderBottom: "1px solid #ccc", marginBottom: "10px", paddingBottom: "10px" }}>

<p><strong>Name:</strong> {office.name}</p>

<p><strong>Address:</strong> {office.address}</p>

<p><strong>Rent:</strong> <span style={rentStyle}>₹{office.rent}</span></p>

</div>

);

})}

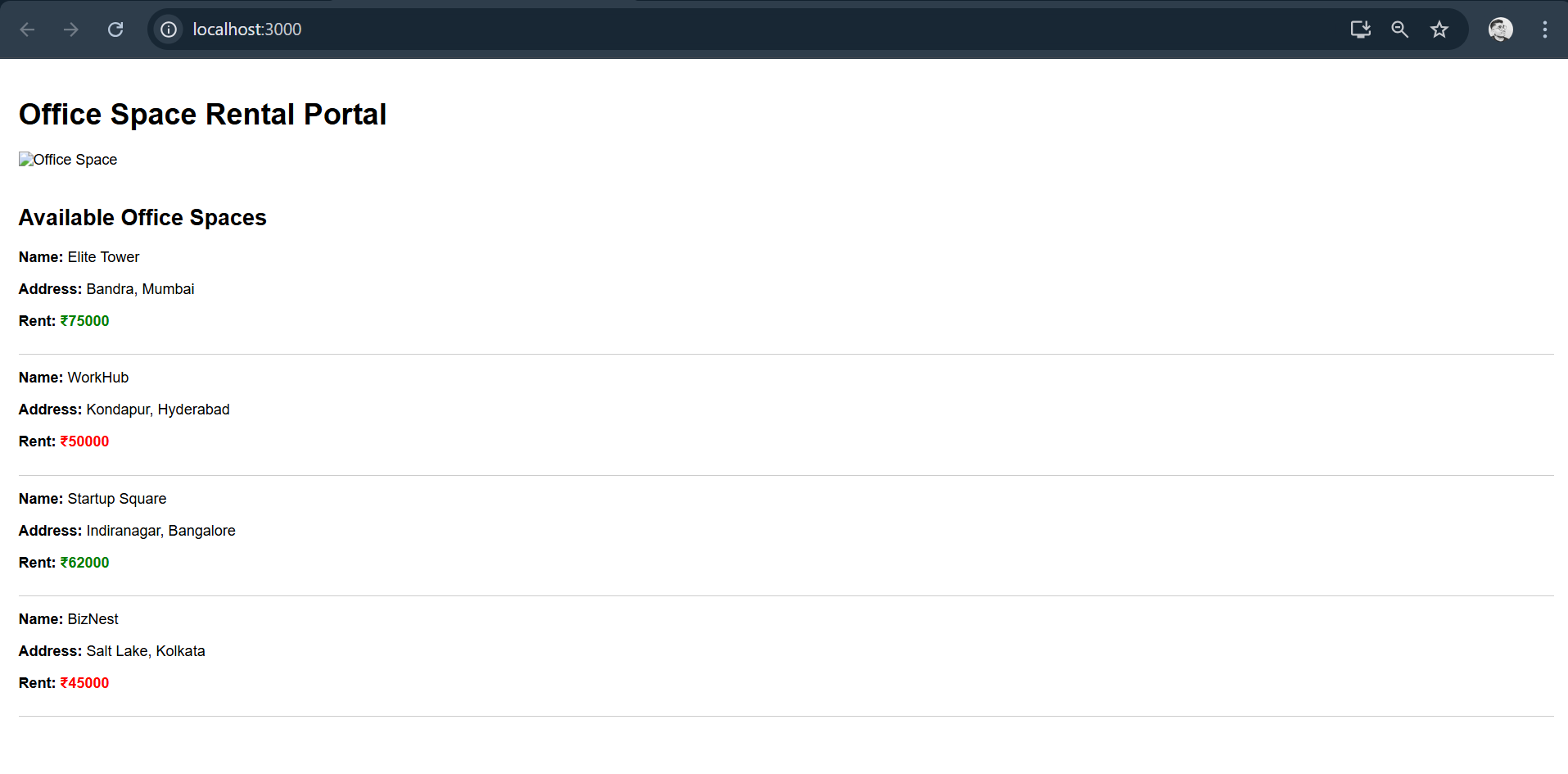
</div>

);

}

export default App;

Output:



11.ReactJS-HOL:

Objectives:

**Explain React events**

React events are React’s way of handling user interactions like clicks, form submissions, keyboard presses, and more within components. While they are similar to native DOM events, React events are implemented in a cross-browser compatible way and follow React’s declarative programming style. Instead of adding event listeners manually in JavaScript, developers specify events directly on JSX elements using props like onClick, onChange, and others. React efficiently manages these events in its virtual DOM, updating the actual DOM only when necessary, leading to better performance and cleaner code.

**Explain about event handlers**

In React, event handlers are functions that are triggered in response to user interactions with UI elements. For example, clicking a button can trigger an onClick event handler function. Event handlers are typically defined as functions inside a component and are passed as props to JSX elements. For instance, you might write <button onClick={handleClick}>Click me</button>, where handleClick is a function defined in the component. Event handlers can also use arrow functions or bind methods to maintain the correct this context, especially in class components.

**Define Synthetic event**

React uses something called a **SyntheticEvent**, which is a cross-browser wrapper around the browser’s native event. Synthetic events normalize event properties and behavior so they work the same way across different browsers. This means developers don’t need to worry about browser inconsistencies when handling events. Synthetic events in React also provide a consistent interface and improve performance by reusing event objects through React’s event pooling system.

**Identify React event naming convention**

React follows a specific naming convention for events: event names use camelCase rather than the lowercase style used in traditional HTML. For example, instead of writing onclick or onchange, you write onClick and onChange in JSX. Additionally, the event handler value in JSX must always be a reference to a function, not a string of JavaScript code, which is different from how inline event handling works in HTML.

**EventDemo.js**

import React, { Component } from 'react';

import CurrencyConvertor from './CurrencyConvertor';

class EventDemo extends Component {

constructor(props) {

super(props);

this.state = {

counter: 0

};

}

increment = () => {

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

this.sayHello();

};

decrement = () => {

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

};

sayHello = () => {

alert("Hello! Counter was incremented.");

};

sayWelcome = (message) => {

alert("Welcome: " + message);

};

handleClick = (e) => {

alert("I was clicked (Synthetic Event)");

console.log("Synthetic event object:", e);

};

render() {

return (

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

<h2>Event Handling in React</h2>

<p><strong>Counter:</strong> {this.state.counter}</p>

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

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

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

<button onClick={this.handleClick}>Synthetic Event (OnPress)</button>

<hr />

<CurrencyConvertor />

</div>

);

}

}

export default EventDemo;

**CurrencyConvertor.js**

import React, { Component } from 'react';

class CurrencyConvertor extends Component {

constructor(props) {

super(props);

this.state = {

inr: '',

euro: ''

};

}

handleChange = (e) => {

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

};

handleSubmit = () => {

const euroRate = 90; // Assume 1 Euro = ₹90

const euroValue = (this.state.inr / euroRate).toFixed(2);

this.setState({ euro: euroValue });

};

render() {

return (

<div style={{ marginTop: '30px' }}>

<h3>Currency Convertor (INR ➝ Euro)</h3>

<input

type="number"

placeholder="Enter amount in INR"

value={this.state.inr}

onChange={this.handleChange}

/>

<button onClick={this.handleSubmit} style={{ marginLeft: '10px' }}>

Convert

</button>

<p><strong>In Euros:</strong> {this.state.euro}</p>

</div>

);

}

}

export default CurrencyConvertor;

**App.js**

import React from 'react';

import EventDemo from './EventDemo';

function App() {

return (

<div className="App">

<EventDemo />

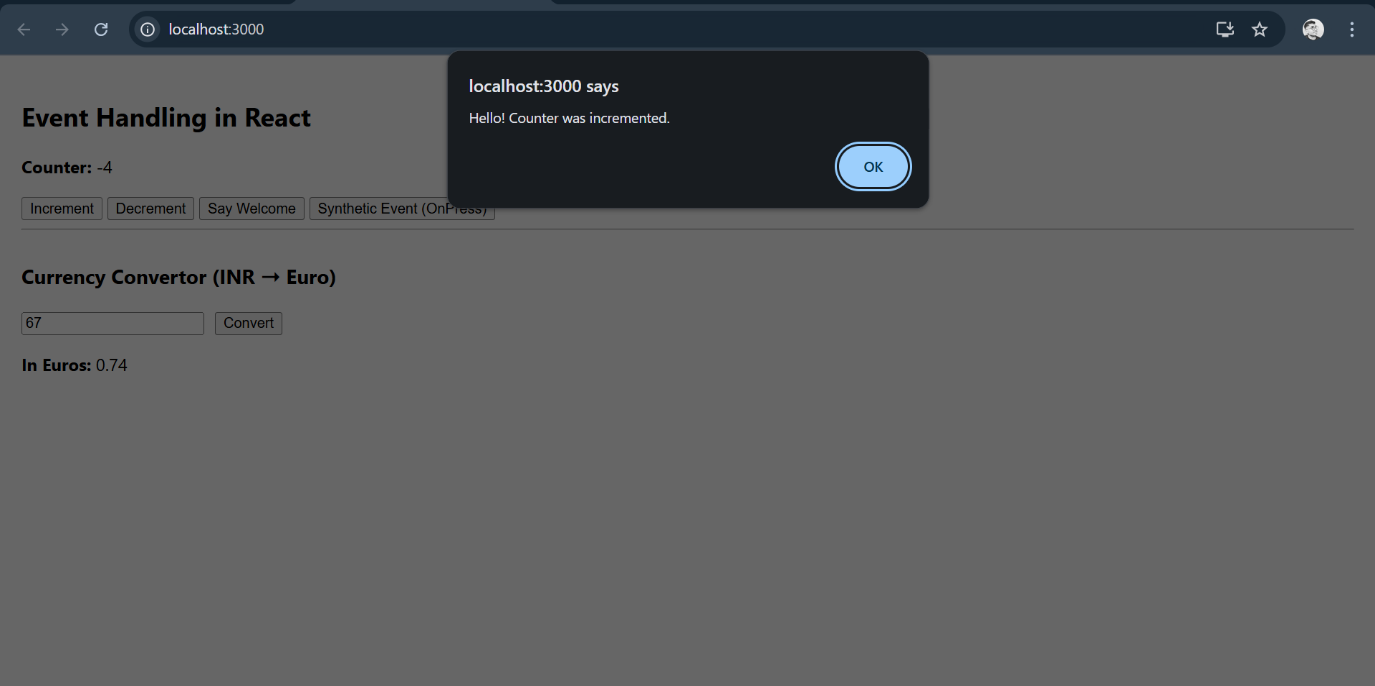
</div>

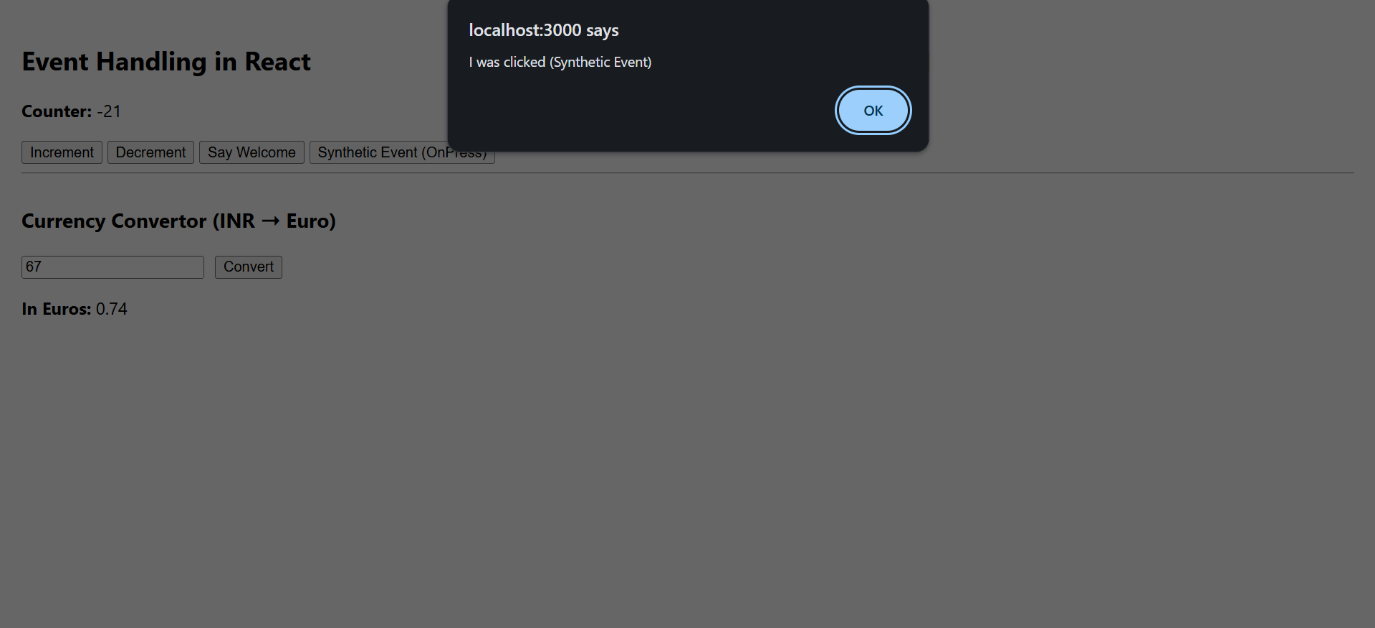
);

}

export default App;

Output:





12.ReactJS-HOL:

Objectives:

**Explain about conditional rendering in React**

Conditional rendering in React is the practice of displaying different UI elements or components based on certain conditions, such as user input, state, or props. Instead of creating separate templates, React lets you use JavaScript logic inside JSX to decide what to render. This can be done using if statements, the ternary operator (condition ? trueCase : falseCase), logical && (to render something only if a condition is true), or by calling functions that return different JSX elements. This flexible approach makes it easy to build dynamic user interfaces that respond to changing data.

**Define element variables**

In React, element variables are regular JavaScript variables that store JSX elements. These variables can then be used within the render method or inside a component’s return statement. Element variables are useful for handling conditional rendering, as you can assign different JSX to a variable based on certain conditions and include it directly in your JSX

**Explain how to prevent components from rendering**

In React, you can prevent a component from rendering by returning null from its render method or functional component. Returning null tells React not to render anything, and nothing will appear in the DOM for that component. This is useful when you want to hide parts of the UI based on certain conditions, like hiding a loading spinner once data is loaded

**GuestPage.js**

import React from 'react';

function GuestPage() {

return (

<div>

<h2>Welcome Guest</h2>

<p>You can browse flight details here.</p>

<ul>

<li>Flight: AI-202 | Mumbai ➝ Delhi | 10:00 AM</li>

<li>Flight: IN-101 | Bangalore ➝ Hyderabad | 1:30 PM</li>

<li>Flight: SP-999 | Chennai ➝ Goa | 4:45 PM</li>

</ul>

</div>

);

}

export default GuestPage;

**UserPage.js**

import React from 'react';

function UserPage() {

return (

<div>

<h2>Welcome User</h2>

<p>You can book your tickets here.</p>

<button>Book Ticket</button>

</div>

);

}

export default UserPage;

**App.js**

import React, { Component } from 'react';

import GuestPage from './GuestPage';

import UserPage from './UserPage';

class App extends Component {

constructor(props) {

super(props);

this.state = {

isLoggedIn: false

};

}

handleLogin = () => {

this.setState({ isLoggedIn: true });

};

handleLogout = () => {

this.setState({ isLoggedIn: false });

};

render() {

const { isLoggedIn } = this.state;

let content;

if (isLoggedIn) {

content = <UserPage />;

} else {

content = <GuestPage />;

}

return (

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

<h1>Ticket Booking Portal</h1>

{isLoggedIn ? (

<button onClick={this.handleLogout}>Logout</button>

) : (

<button onClick={this.handleLogin}>Login</button>

)}

<hr />

{content}

</div>

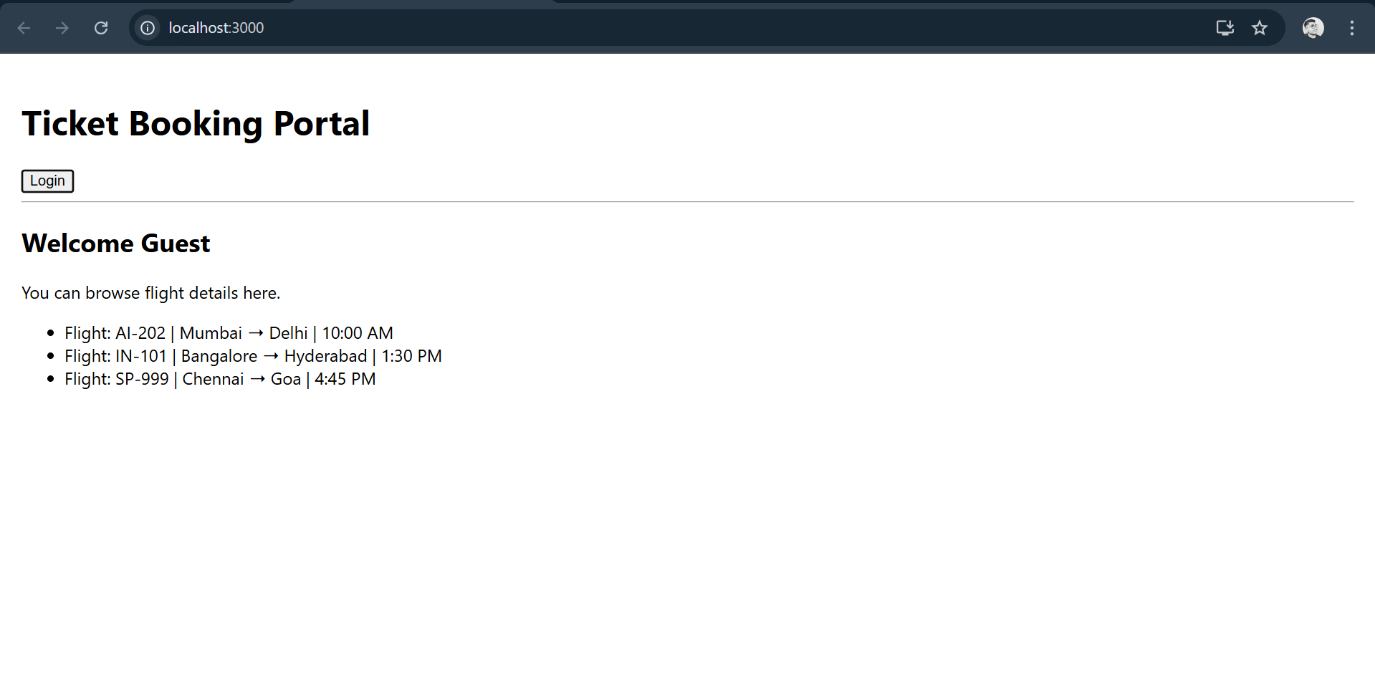
);

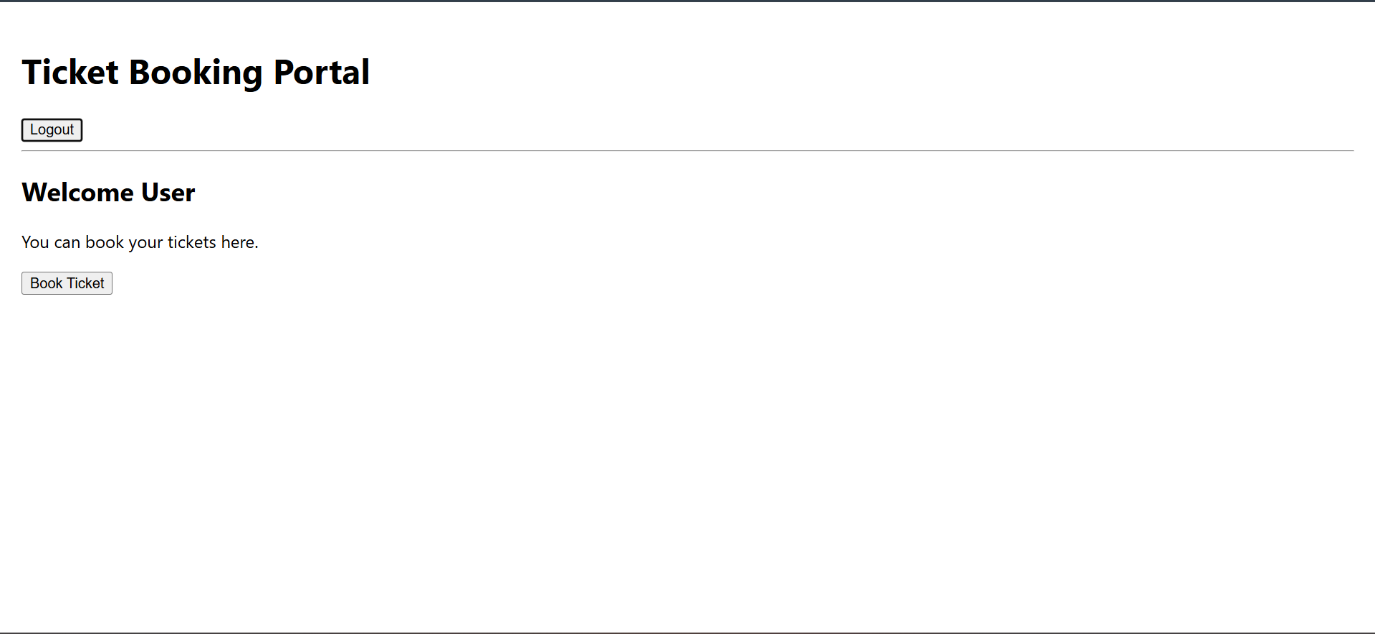
}

}

export default App;

Output:





13.ReactJS-HOL:

Objective:

**Explain various ways of conditional rendering**

React supports multiple ways to implement conditional rendering, allowing you to decide what to display based on application state or props. The simplest method is using JavaScript if or if-else statements inside the component before returning JSX. Another common way is the **ternary operator** (condition ? elementIfTrue : elementIfFalse) directly inside JSX for inline conditions. You can also use the **logical AND operator** (&&) to render an element only when a condition is true, which is helpful for shorter conditions: isLoggedIn && <LogoutButton />. Additionally, you can store JSX in **element variables** and choose which element to render based on conditions, keeping the JSX clean and readable.

**Explain how to render multiple components**

In React, you can render multiple components by including them inside a single parent element, like a <div>, a React fragment (<>...</>), or using <React.Fragment>. This way, React returns a tree of elements instead of just one.

**Define list component**

A **list component** in React is a component that displays a collection of data items by iterating over them, usually with the map() function. It takes an array (such as an array of users or products) and returns an array of JSX elements, typically rendering each item using another sub-component. This pattern helps handle repetitive structures in the UI efficiently. For example, a UserList component could loop through an array of user objects and render a UserCard component for each.

**Explain about keys in React applications**

Keys are special string attributes used by React to identify which items in a list have changed, been added, or removed. They help React optimize rendering by making the process of diffing virtual DOM trees faster and more predictable. Typically, a unique and stable value like an ID or a unique string from your data should be used as the key.

**Explain how to extract components with keys**

When building lists, it’s common to extract the repeated JSX into its own component for clarity and reuse. Each item should still have a key prop when rendering the list, and the key is usually passed when mapping over the list, **not inside** the child component itself.

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

In React, the map() function is commonly used to transform an array of data into an array of JSX elements. It’s part of JavaScript’s standard Array.prototype.map() method, which takes a callback function and applies it to each element, returning a new array.

**BookDetails.js**

import React from 'react';

function BookDetails() {

return (

<div>

<h2>📘 Book Details</h2>

<ul>

<li>Title: React Basics</li>

<li>Author: Dan Abramov</li>

<li>Price: ₹499</li>

</ul>

</div>

);

}

export default BookDetails;

**BlogDetails.js**

import React from 'react';

function BlogDetails() {

return (

<div>

<h2>📝 Blog Details</h2>

<ul>

<li>Title: Understanding Hooks</li>

<li>Author: Sophie Alpert</li>

<li>Date: Aug 2025</li>

</ul>

</div>

);

}

export default BlogDetails;

**CourseDetails.js**

import React from 'react';

function CourseDetails() {

return (

<div>

<h2>🎓 Course Details</h2>

<ul>

<li>Title: Full Stack React</li>

<li>Instructor: Ryan Florence</li>

<li>Duration: 6 weeks</li>

</ul>

</div>

);

}

export default CourseDetails;

**App.js**

import React, { Component } from 'react';

import BookDetails from './BookDetails';

import BlogDetails from './BlogDetails';

import CourseDetails from './CourseDetails';

class App extends Component {

constructor(props) {

super(props);

this.state = {

selected: "book", // Try: book, blog, course

showCourses: true

};

}

renderComponentByIfElse() {

if (this.state.selected === "book") {

return <BookDetails />;

} else if (this.state.selected === "blog") {

return <BlogDetails />;

} else {

return <CourseDetails />;

}

}

render() {

const { selected, showCourses } = this.state;

const dataList = [

{ id: 1, type: "book", component: <BookDetails /> },

{ id: 2, type: "blog", component: <BlogDetails /> },

{ id: 3, type: "course", component: <CourseDetails /> }

];

return (

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

<h1>📚 Blogger Portal</h1>

<h3>1. If-Else Rendering</h3>

{this.renderComponentByIfElse()}

<h3>2. Ternary Operator Rendering</h3>

{selected === "blog" ? <BlogDetails /> : <p>No blog selected</p>}

<h3>3. Logical AND Rendering</h3>

{showCourses && <CourseDetails />}

<h3>4. Conditional Map Rendering with Keys</h3>

{dataList.map(item =>

item.type === selected ? (

<div key={item.id}>{item.component}</div>

) : null

)}

</div>

);

}

}

export default App;

Output:

