WEEK 7 –

9.REACT-JS-HOL –

1. Features of ES6 (ECMAScript 2015)

ES6 introduced major improvements to JavaScript. Key features include:

- let and const (block-scoped variables)
- Arrow functions (=>)
- Template literals
- Classes and inheritance
- Default and rest parameters
- Destructuring (arrays & objects)
- Modules (import and export)
- Promises (for asynchronous code)
- Map and Set data structures
- Spread (...) and rest (...) operators

2. let in JavaScript

- Declares block-scoped variables.
- Can be updated but not re-declared in the same block.
- Unlike var, it avoids hoisting-related bugs.

```
let x = 10;
x = 20; // valid
let x = 30; //  Error: Already declared
```

3. Difference Between var and let

Feature	var	let
Scope	Function-scoped	Block-scoped ({})
Hoisting	Hoisted and initialized	Hoisted but not initialized
Re-declaration	Allowed	Not allowed in same block
Use in loops	Shared across iterations	Unique per iteration

4. const in JavaScript

- Also block-scoped like let.
- Must be initialized at the time of declaration.
- Cannot be reassigned, but mutable objects like arrays can still be modified.

```
const pi = 3.14;
// pi = 3.15;  Error

const arr = [1, 2];
arr.push(3); //  Valid
```

5. ES6 Class Fundamentals

Introduces object-oriented class syntax.

javascript

CopyEdit

```
class Person {
  constructor(name) {
    this.name = name;
  }

  greet() {
    return `Hello, ${this.name}`;
  }
}

const p = new Person("Harshi");
console.log(p.greet());
```

6. ES6 Class Inheritance

One class can inherit from another using extends.

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  speak() {
    return `${this.name} makes a sound`;
  }
}

class Dog extends Animal {
  speak() {
    return `${this.name} barks`;
  }
}

const dog = new Dog("Bruno");
console.log(dog.speak()); // Bruno barks
```

7. Arrow Functions in ES6

- Compact function syntax.
- Lexically binds this.
- Cannot be used as constructors.

```
// Traditional
function add(a, b) {
  return a + b;
}

// Arrow
const add = (a, b) => a + b;
```

Set() and Map()

Set

- Stores unique values of any type.
- No duplicate entries allowed.

```
const numbers = new Set([1, 2, 2, 3]);

console.log(numbers); // Set { 1, 2, 3 }
```

Map

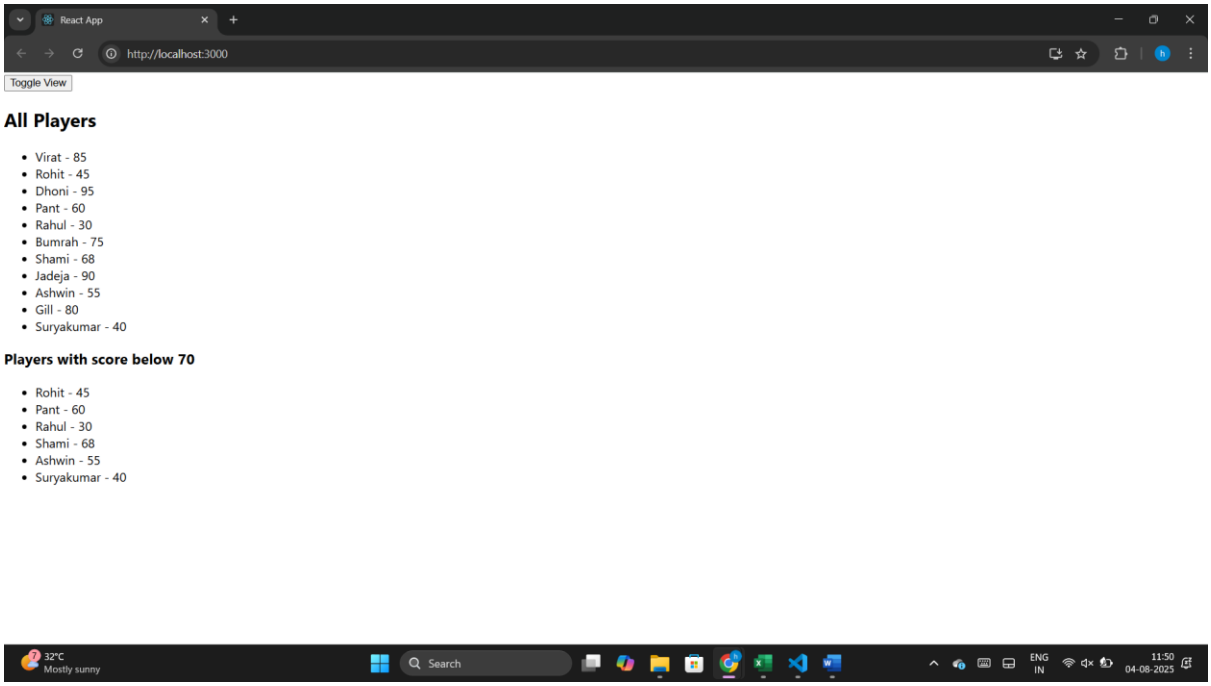
- Key-value pairs.
- Keys can be of any type (object, function, etc.).

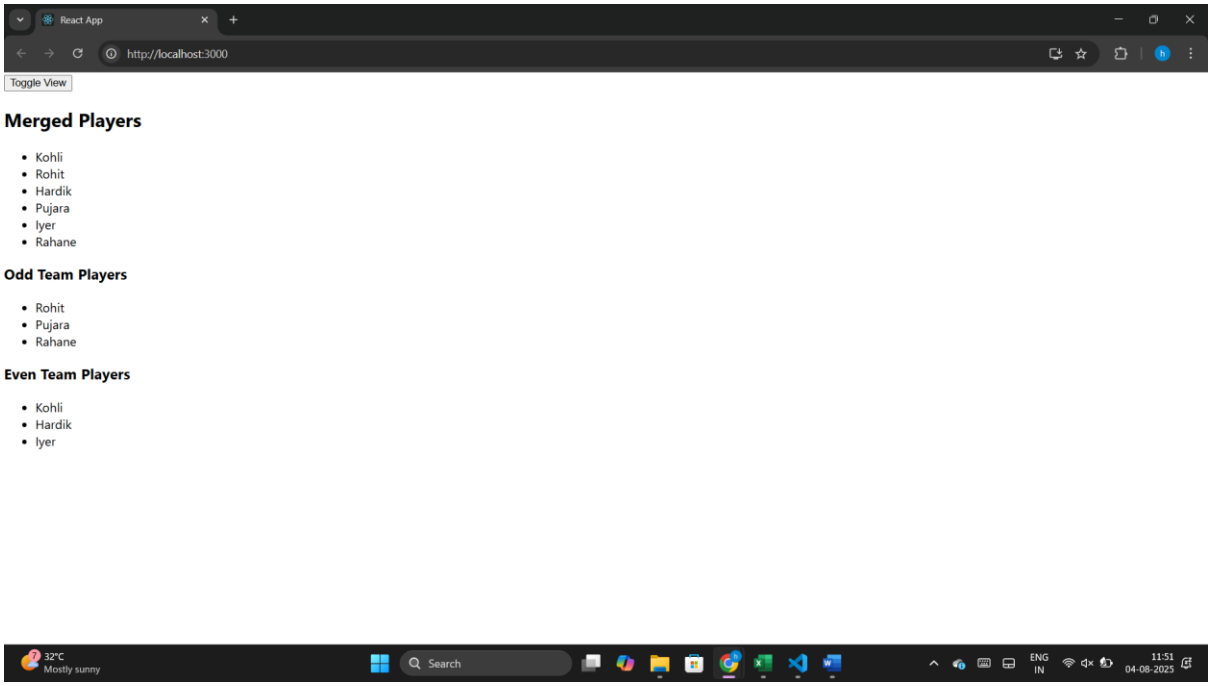
```
const capitals = new Map();

capitals.set('India', 'New Delhi');

capitals.set('Germany', 'Berlin');

console.log(capitals.get('India')); // New Delhi
```





10-REACT-JS-HOL

1. Define JSX

JSX (JavaScript XML) is a syntax extension for JavaScript used with React to describe what the UI should look like.

- JSX looks like HTML but is actually syntactic sugar for React.createElement() calls.
- JSX allows embedding HTML-like code directly in JavaScript.

Example:

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

Under the hood, it becomes:

javascript

CopyEdit

const element = React.createElement('h1', null, 'Hello, World!');

2. Explain ECMAScript (ES)

ECMAScript (often abbreviated as ES) is the standardized version of JavaScript defined by ECMA International.

- It ensures JavaScript works consistently across browsers and platforms.

- Major versions: ES5 (2009), ES6/ES2015 (biggest update), followed by annual updates like ES7, ES8, etc.

ES6 introduced features like:
let, const, arrow functions, classes, template literals, destructuring, spread/rest operators, Map, Set, promises, and modules.

3. Explain React.createElement()

This is the core function used by React to create a virtual DOM node.

Syntax:

React.createElement(type, props, ...children)

- type: HTML tag or React component (e.g., 'div' or MyComponent)
- props: an object with properties (e.g., { className: "box" })
- children: nested elements or text

Example:

React.createElement('h1', { style: { color: 'blue' } }, 'Welcome!');

4. How to Create React Nodes with JSX

JSX allows you to create React nodes in a readable, HTML-like format:

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

You can use:

- Self-closing tags: <img />
- Nested elements: Text

Each JSX element is compiled to React.createElement().

5.How to Render JSX to DOM

You use ReactDOM.render() (for older React versions) or createRoot().render() in React 18+.

jsx

CopyEdit

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);
```

Make sure the HTML has a div with id root:

```
<div id="root"></div>
```

6. Using JavaScript Expressions in JSX

You can use any JavaScript expression inside {} in JSX:

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

You can also use:

```
<p>{2 + 3}</p>
<p>{name.toUpperCase()}</p>
<p>{isLoggedIn ? "Logout" : "Login"}</p>
```

7. Using Inline CSS in JSX

In JSX, inline CSS is written using an object (camelCase for properties):

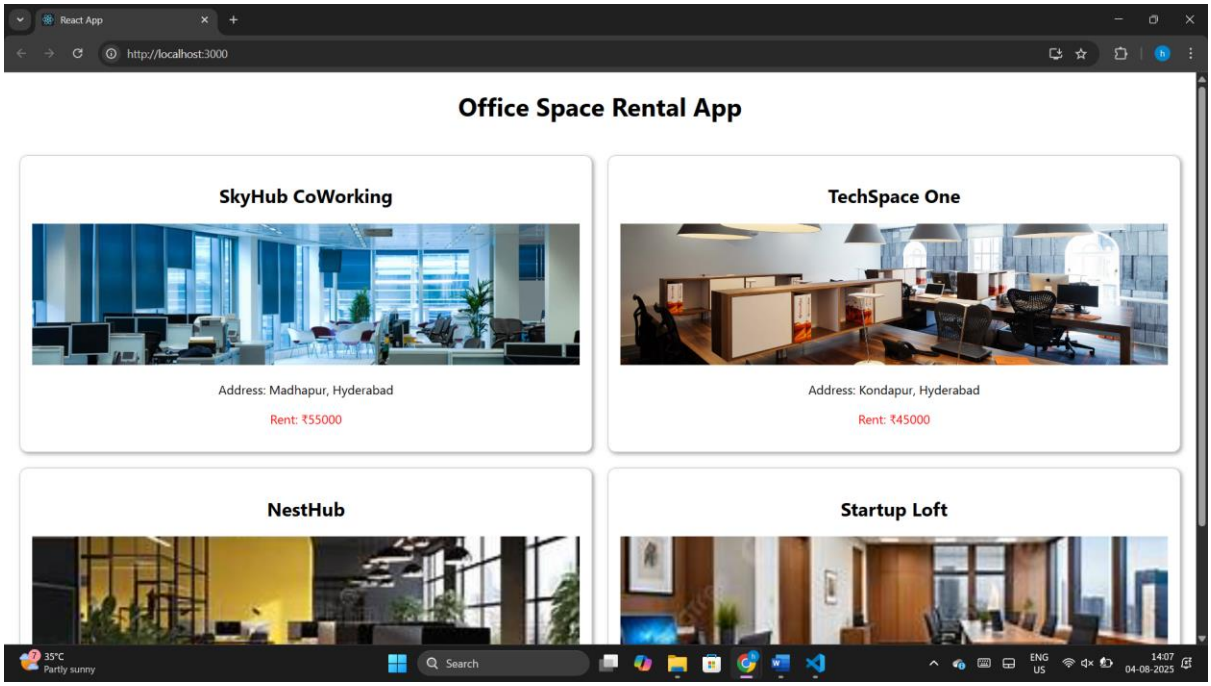
jsx

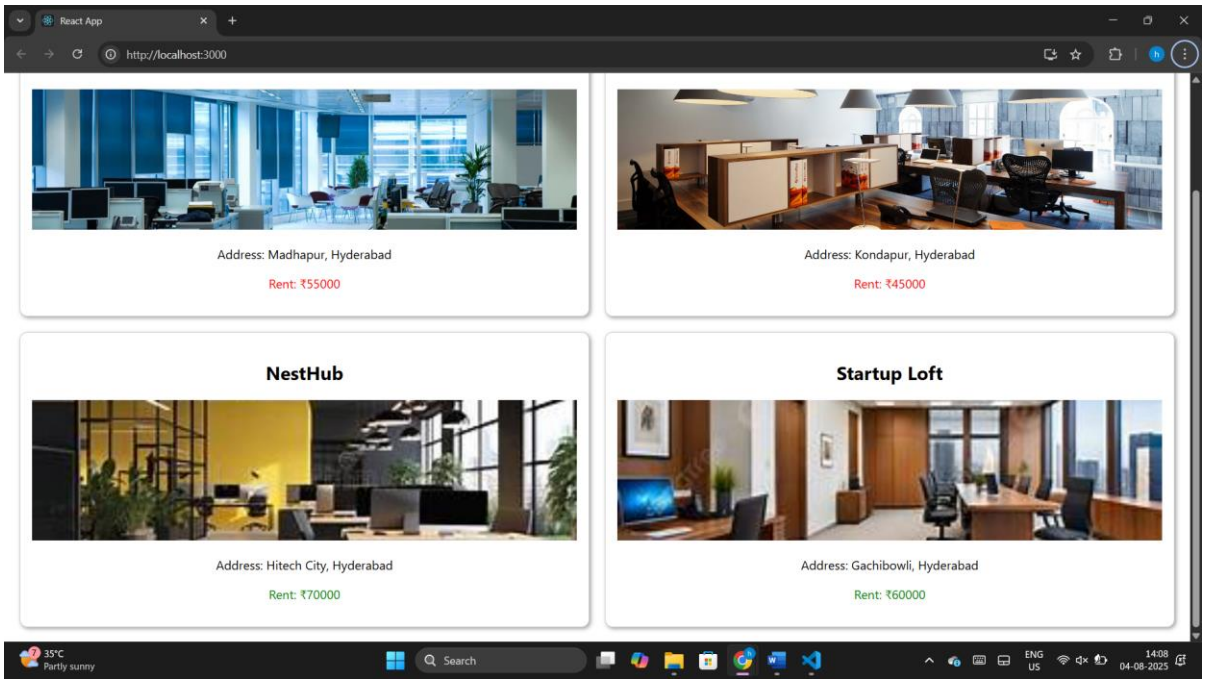
CopyEdit

```
const styleObj = {
  color: "white",
  backgroundColor: "black",
  padding: "10px"
};

const element = <h1 style={styleObj}>Styled Text</h1>;
```

Output -





11. REACT-JS-HOL – 11

1. Explain React Events

In React, events are actions that occur due to user interaction such as clicking, typing, submitting, hovering, etc.

React events are very similar to DOM events in plain JavaScript, but with a few differences:

- React uses a cross-browser wrapper around the browser's native events called SyntheticEvent.
- React events are written in camelCase (e.g., onClick instead of onclick).

Example:

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

2. Explain Event Handlers

An event handler is a function that gets called when an event is triggered.

In React:

- You assign the handler function directly inside JSX.
- You do not use parentheses unless you're passing arguments.

Example:

```
function handleClick() {
  alert('Button was clicked!');
}
```

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

With parameters:<button onClick={() => handleClick("Hi")}>Click</button>

3.Define Synthetic Event

A SyntheticEvent is React's cross-browser wrapper around the native browser event.
It normalizes events so that they behave the same in every browser.

- It wraps the native DOM event and provides the same interface.
- It's part of React's event system for better performance and consistency.

Example:

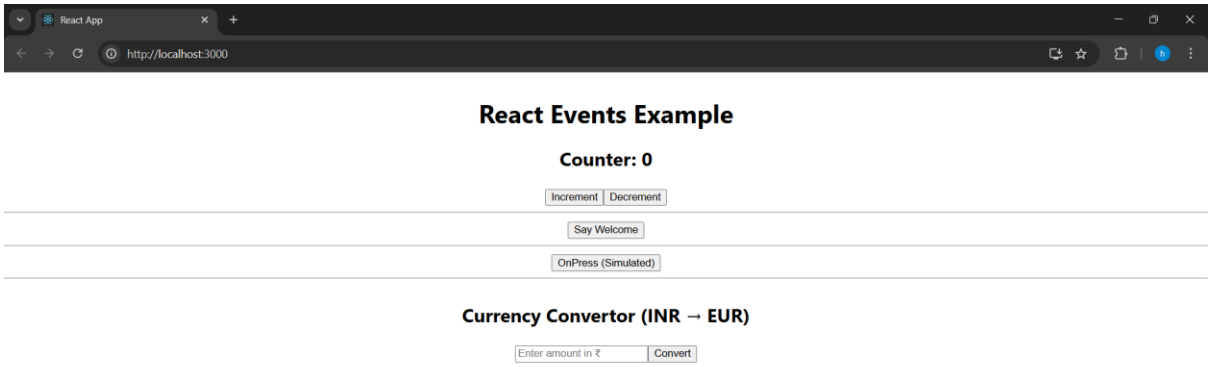
```
function handleChange(e) {
  console.log(e); // SyntheticEvent
  console.log(e.target.value); // Access input value
}
```

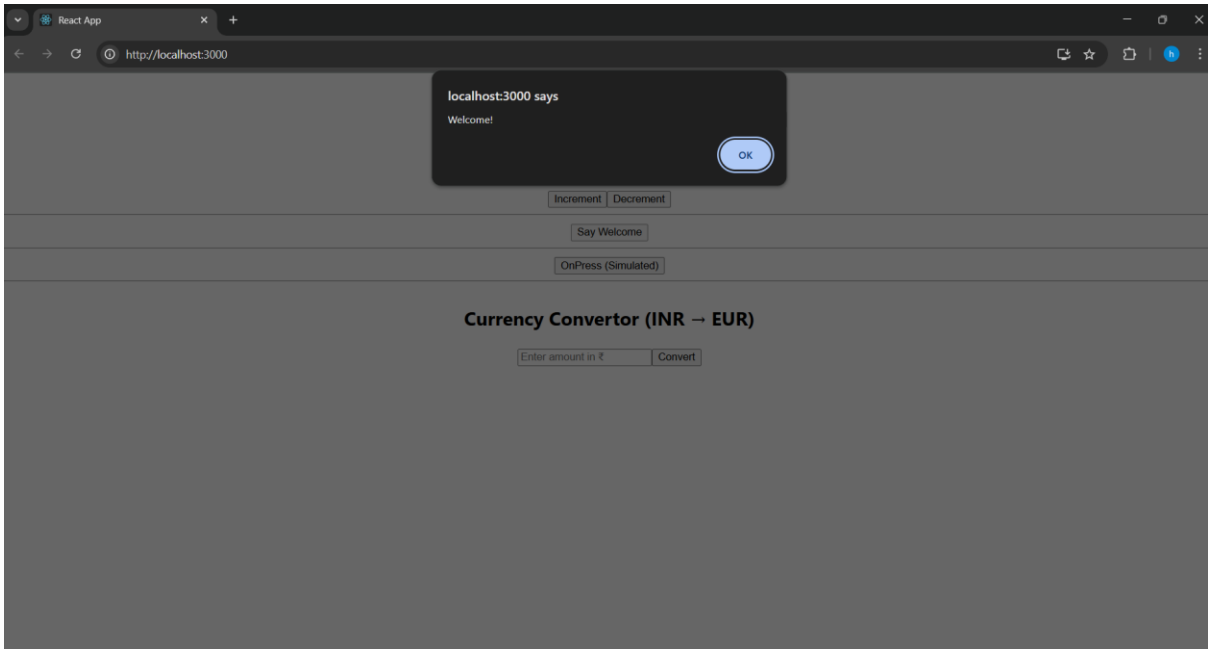
<input type="text" onChange={handleChange} />

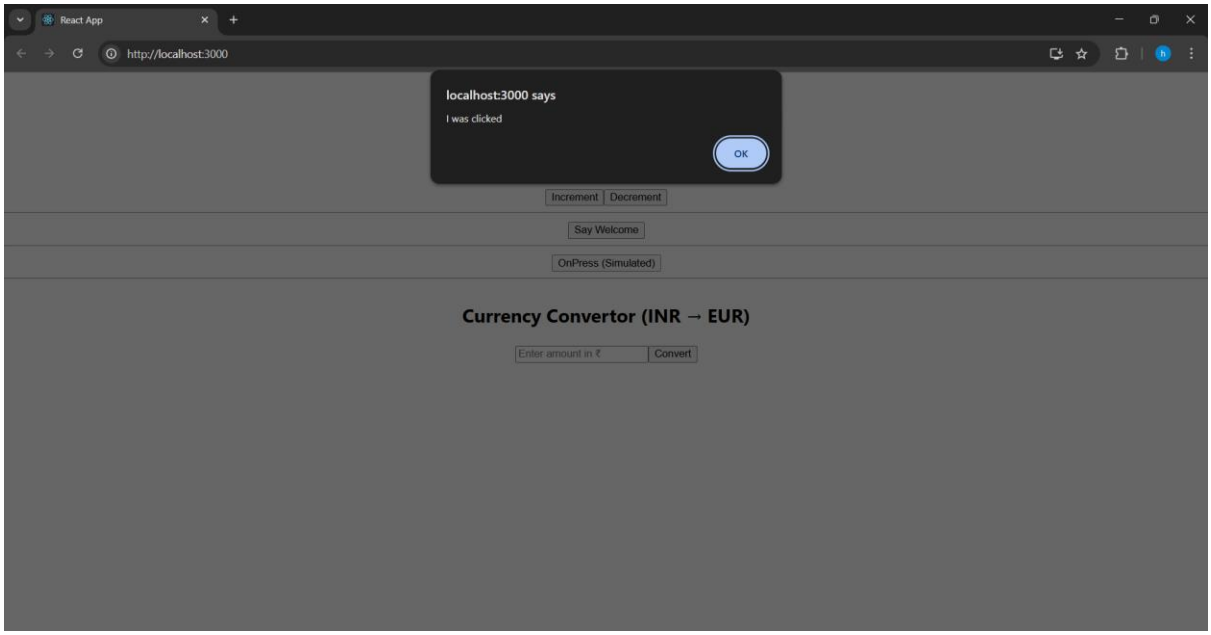
4. React Event Naming Convention

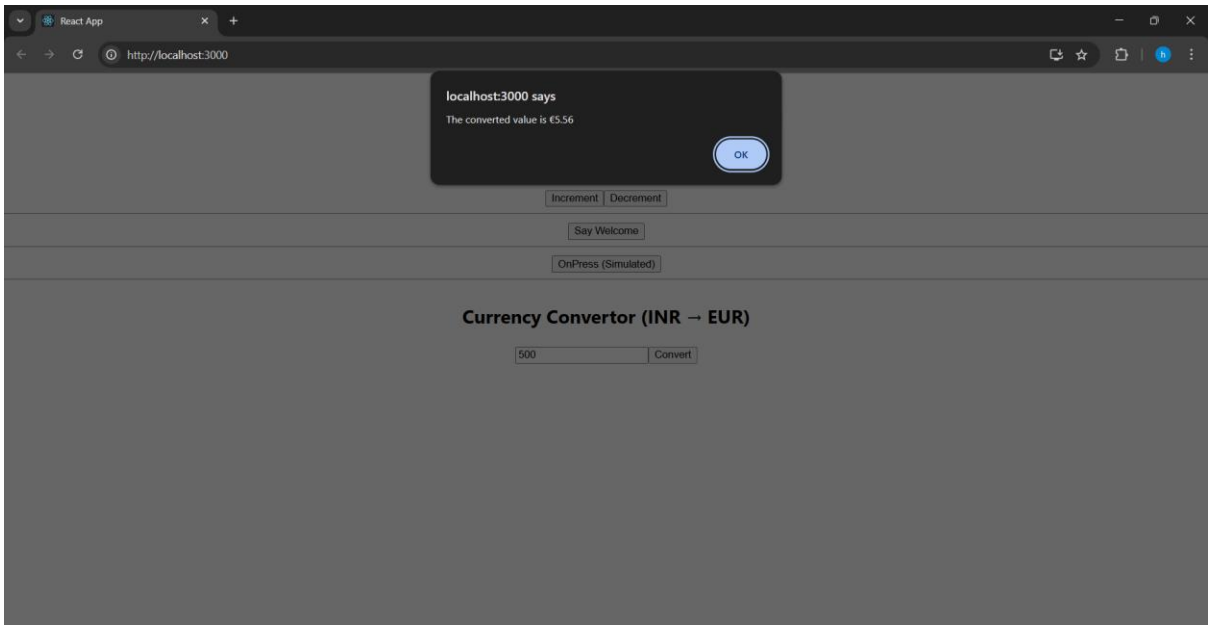
Feature	React	HTML
Syntax	camelCase	lowercase
Value	JSX expression	String

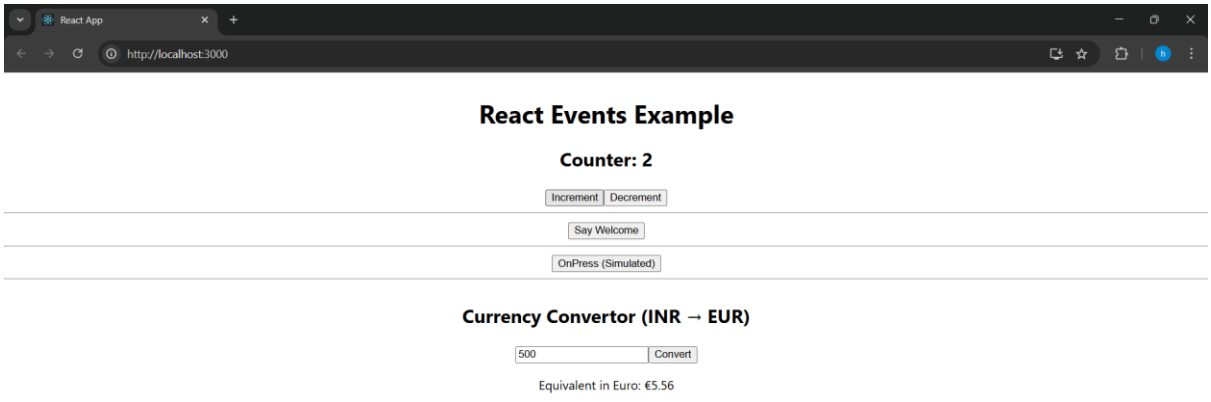
Output:






React App
http://localhost:3000
localhost:3000 says
I was clicked
OK
Increment
Decrement
Say Welcome
OnPress (Simulated)
Currency Convertor (INR → EUR)
Enter amount in ₹
Convert


React App
http://localhost:3000
localhost:3000 says
The converted value is €5.56
OK
Increment
Decrement
Say Welcome
OnPress (Simulated)
Currency Convertor (INR → EUR)
500
Convert


React App
http://localhost:3000
React Events Example
Counter: 2
Increment
Decrement
Say Welcome
OnPress (Simulated)
Currency Convertor (INR → EUR)
500
Convert
Equivalent in Euro: €5.56

12.REACT-JS-HOL

1. Conditional Rendering in React

Conditional Rendering means displaying different UI elements depending on certain conditions (like a flag or value in state).

Example using if-else:

```
function Greeting(props) {
  if (props.isLoggedIn) {
    return <h1>Welcome back!</h1>;
  } else {
    return <h1>Please sign in.</h1>;
  }
}
```

Example using ternary operator:

```
const isLoggedIn = true;
return <h1>{isLoggedIn ? "Welcome!" : "Please log in"}</h1>;
```

Example using logical AND (&&):

```
{isLoggedIn && <button>Logout</button>}
```

2. Define Element Variables in React

You can use element variables to conditionally assign and render JSX elements.

Example:

```
let button;
if (isLoggedIn) {
  button = <button>Logout</button>;
} else {
  button = <button>Login</button>;
}

return <div>{button}</div>;
```

Here, button is a JSX element stored in a variable, and it's later rendered based on a condition.

3. Prevent Components from Rendering

To prevent a component from rendering, you can:

Return null from a component:

```
function WarningBanner(props) {
  if (!props.show) {
    return null; // Prevents rendering
  }
  return <div className="warning">Warning!</div>;
}
```

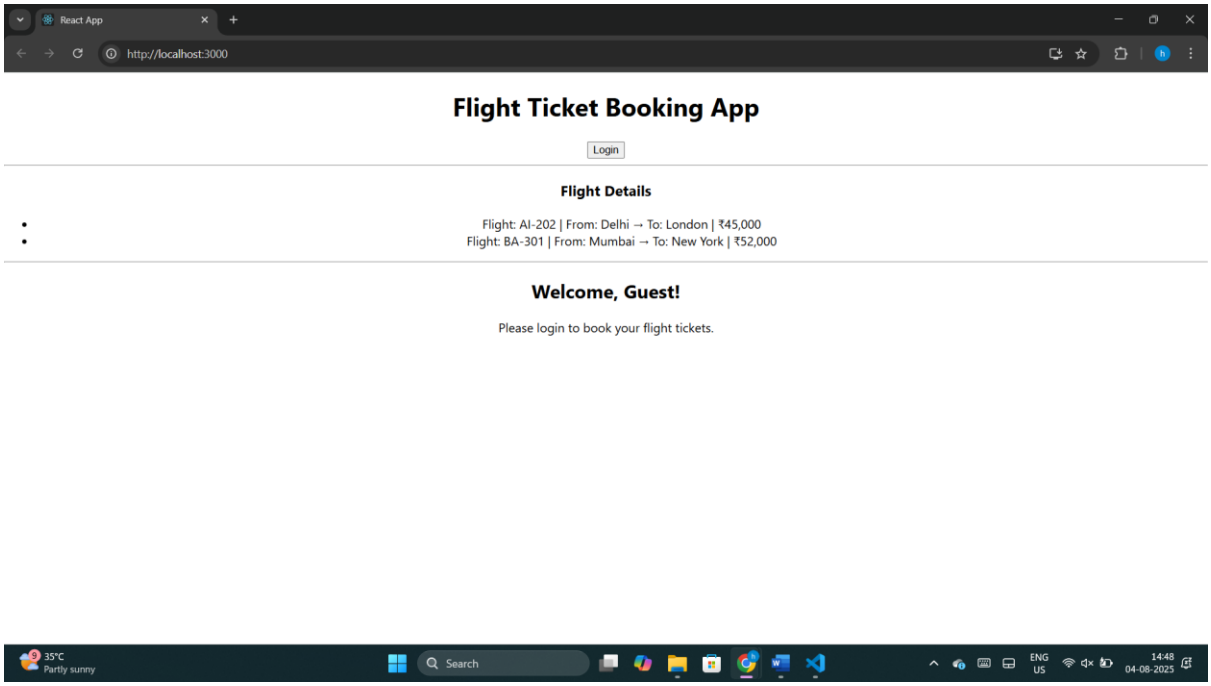
Use conditions in parent:

```
{showBanner && <WarningBanner />}
```

Use conditional logic before rendering:

```
function MyComponent({ shouldDisplay }) {
  if (!shouldDisplay) return null;
  return <div>This will only show if allowed</div>;
}
```

Output:



Flight Ticket Booking App

Logout

Flight Details

- Flight: AI-202 | From: Delhi → To: London | ₹45,000
- Flight: BA-301 | From: Mumbai → To: New York | ₹52,000

Welcome, User!

You can now book your flight tickets.