**9. ReactJS-HOL**

**1. Features of ES6 (ECMAScript 2015)**

ES6 introduced significant advancements to the JavaScript language, making it more powerful, readable, and developer-friendly. Key features include:

* **Block-scoped declarations**: let and const provide block-level scoping, unlike var which is function-scoped.
* **Arrow functions**: Offer concise syntax for writing functions and lexically bind this.
* **Classes**: Syntactic sugar over prototype-based inheritance, making OOP in JavaScript clearer.
* **Modules**: import and export facilitate modular programming.
* **Template literals**: Use back ticks for easier string interpolation and multi-line strings.
* **Destructuring**: Enables unpacking of arrays or objects into distinct variables.
* **Default parameters**: Allow default values in function parameters.
* **Spread and rest operators**: Simplify array and object handling.
* **Promises**: Provide better support for asynchronous programming.
* **New data structures**: Map, Set, WeakMap, and WeakSet enhance data handling capabilities

**2. Explain JavaScript let**

The let keyword is used to declare variables that are **block-scoped**, meaning the variable is accessible only within the enclosing {} block. Unlike var, let does not allow redeclaration in the same scope and does not get hoisted in the same way — it's subject to the **temporal dead zone (TDZ)**, where accessing the variable before declaration leads to an error.

Example:

**{**

**let x = 5;**

**console.log(x); =>5**

**}**

**console.log(x); =>Reference Error**

**3. Differences between var and let**

| **Feature** | **var** | **let** |
| --- | --- | --- |
| Scope | Function-scoped | Block-scoped |
| Hoisting | Hoisted and initialized with undefined | Hoisted but not initialized |
| Redeclaration | Can be re-declared in the same scope | Cannot be re-declared in the same scope |
| Usage | Can lead to bugs in larger codebases due to scope issues | Safer and more predictable |

**4. Explain JavaScript const**

The const keyword declares **read-only variables** that cannot be reassigned after initialization. Like let, it is block-scoped and does not allow redeclaration. However, if the variable is an object or array, its **contents can still be modified** only the binding to the memory address is immutable.

Example:

**const obj = { a: 1 };**

**obj.a = 2; // Allowed**

**obj = {}; // Error**

**5. ES6 Class Fundamentals**

ES6 classes provide a cleaner, more intuitive syntax for creating objects and handling inheritance compared to prototypes.

Key components:

* **Constructor**: Special method for initializing objects.
* **Methods**: Declared directly within the class.
* **Static methods**: Belong to the class, not instances.

Example:

class Person {

constructor(name) {

this.name = name;

}

greet() {

return `Hello, I'm ${this.name}`;

}

}

**6. ES6 Class Inheritance**

Inheritance in ES6 is implemented using the extends keyword, allowing one class to inherit properties and methods from another. The super() function must be called in the constructor of the subclass to access and invoke the parent class’s constructor.

Example:

class Person {

constructor(name) {

this.name = name;

}

class Student extends Person {

constructor(name, grade) {

super(name);

this.grade = grade;

}

info() {

return `${this.name} is in grade ${this.grade}`;

}

}

**7. Define ES6 Arrow Functions**

Arrow functions provide a concise syntax for writing anonymous functions and automatically bind the **this** context lexically, which is particularly useful in callbacks and asynchronous operations.

Example:

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

Differences from regular functions:

* Do not have their own **this**, **arguments**, or **super**.
* Cannot be used as constructors.
* Always anonymous.

**8. Identify Set() and Map()**

* Set(): A Set is a collection of unique values. It removes duplicates automatically. You can add, delete, and check for elements easily.

let mySet = new Set([1, 2, 3, 2, 4, 3]);

Output: Set {1, 2, 3, 4}

* Map(): A Map stores key-value pairs, where keys can be of any data type. It remembers the order of insertion and offers methods like set(), get(), and has().

let myMap = new Map();

myMap.set('name', 'Josh');

myMap.set(1, 'One');

**Code:**

App.js:

import React from 'react';

import './App.css';

import ListofPlayers from './components/ListofPlayers';

import IndianPlayers from './components/IndianPlayers';

function App() {

  const flag = true;

  return (

    <div className="App">

      <h1>Cricket App</h1>

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

    </div>

  );

}

export default App;

ListofPlayers.js:

import React from 'react';

const ListofPlayers = () => {

  const players = [

    { name: 'Virat', score: 88 },

    { name: 'Rohit', score: 45 },

    { name: 'Dhoni', score: 90 },

    { name: 'Jadeja', score: 35 },

    { name: 'Ashwin', score: 71 },

    { name: 'Shami', score: 65 },

    { name: 'Bumrah', score: 85 },

    { name: 'Surya', score: 25 },

    { name: 'Gill', score: 95 },

    { name: 'Kohli', score: 78 },

    { name: 'KL Rahul', score: 55 },

  ];

  const filteredPlayers = players.filter(player => player.score < 70);

  return (

    <div style={{ padding: '20px', fontFamily: 'Arial, sans-serif' }}>

      <h2>All Players</h2>

      <ul>

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

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

        ))}

      </ul>

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

      <ul>

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

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

        ))}

      </ul>

    </div>

  );

};

export default ListofPlayers;

IndianPlayers.js:

import React from 'react';

const IndianPlayers = () => {

  const team = ['Sachin1', 'Dhoni2', 'Virat3', 'Rohit4', 'Yuvaraj5', 'Raina6'];

  const oddPlayers = team.filter((\_, index) => index % 2 === 0);

  const evenPlayers = team.filter((\_, index) => index % 2 !== 0);

  const oddLabels = ['First', 'Third', 'Fifth'];

  const evenLabels = ['Second', 'Fourth', 'Sixth'];

  const T20players = ['First Player', 'Second Player', 'Third Player'];

  const RanjiTrophy = ['Fourth Player', 'Fifth Player', 'Sixth Player'];

  const mergedPlayers = [...T20players, ...RanjiTrophy];

  return (

    <div style={{ padding: '20px', fontFamily: 'Arial, sans-serif' }}>

      <h2>Odd Players</h2>

      <ul>

        {oddPlayers.map((name, i) => (

          <li key={i}>

            <strong>{oddLabels[i]}:</strong> {name}

          </li>

        ))}

      </ul>

      <h2>Even Players</h2>

      <ul>

        {evenPlayers.map((name, i) => (

          <li key={i}>

            <strong>{evenLabels[i]}:</strong> {name}

          </li>

        ))}

      </ul>

      <hr />

      <h2>List of Indian Players Merged:</h2>

      <ul>

        {mergedPlayers.map((player, i) => (

          <li key={i}>Mr. {player}</li>

        ))}

      </ul>

    </div>

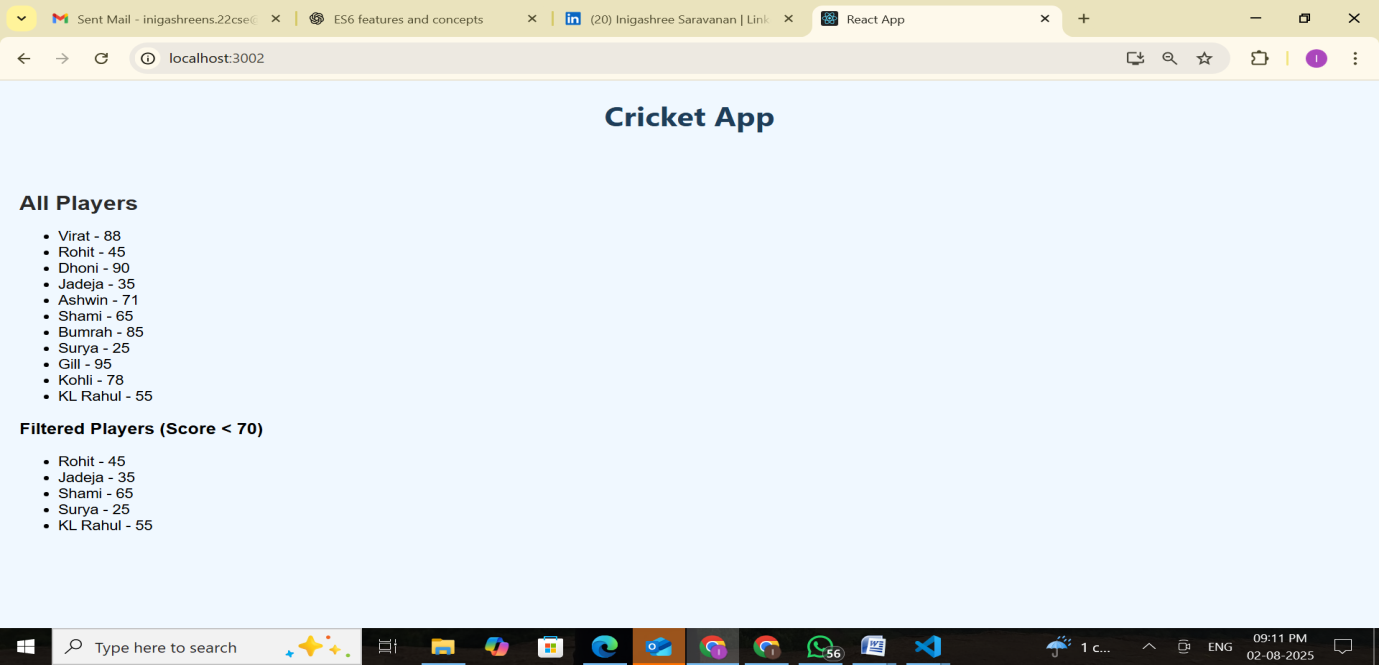
  );

};

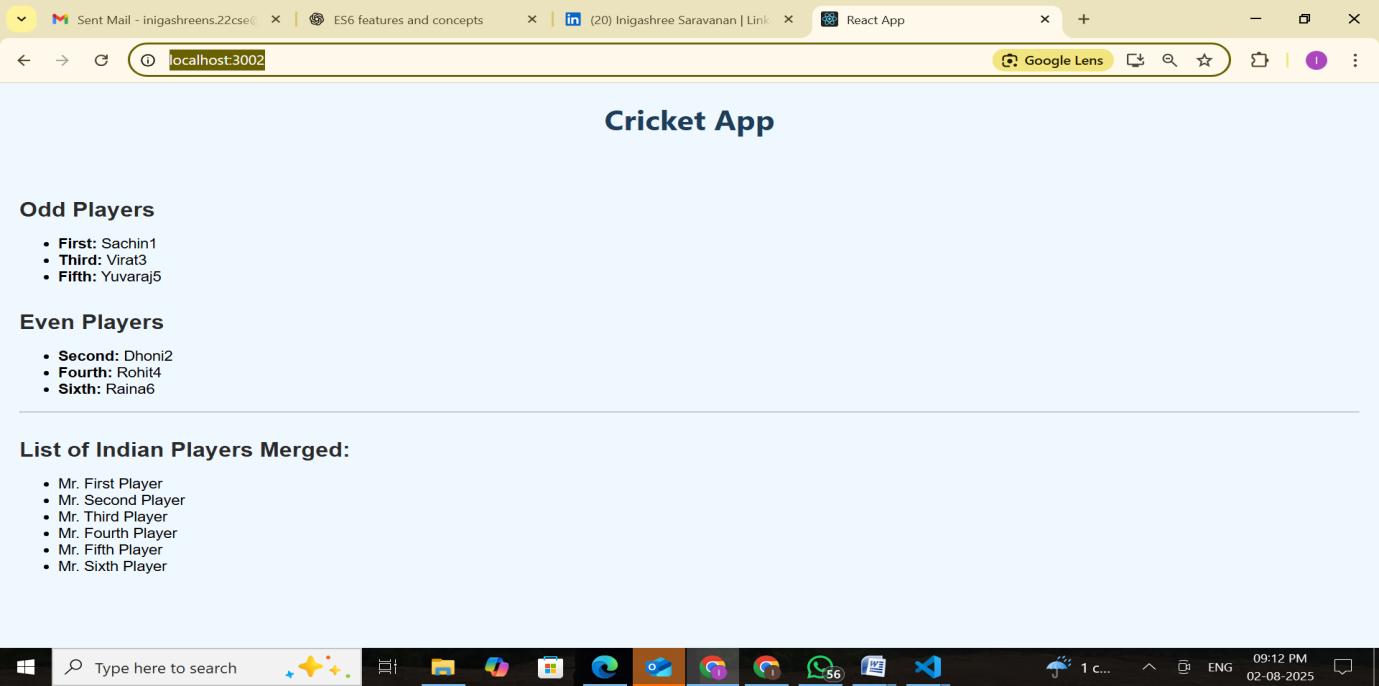
export default IndianPlayers;

**Output:**

When const flag=true, in App.js:



When const flag=false, in App.js:



**10. ReactJS-HOL**

### 1. ****Define JSX****

JSX stands for JavaScript XML.It allows writing HTML-like syntax inside JavaScript code, especially in React. JSX makes it easier to create and visualize UI components. It is not plain HTML, but it gets converted into JavaScript using a compiler like Babel.

### 2. ****Explain about ECMAScript****

ECMAScript is the official standard for scripting languages like JavaScript. It defines how the language should behave and what features it supports. React uses features from ECMAScript, especially ES6 and later, such as let, const, arrow functions, classes, promises, and modules.

### 3. ****Explain React.createElement()****

The React.createElement() method is used by React to create virtual DOM elements.It takes three arguments: the type of element, its properties (if any), and its content. For example, React.createElement('h1', null, 'Hello') creates an element similar to <h1>Hello</h1>.When we write JSX, it is internally converted into React.createElement() calls.

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

React nodes can be created using JSX by writing code that looks like HTML inside JavaScript files. For example:const element = <h2>Welcome</h2>;

This creates a React node which can be rendered to the DOM.  
JSX allows nesting of elements and can also include custom components.

### 5. ****Define how to render JSX to DOM****

To render JSX into the browser, we use a method provided by React.  
In React version 18 and above, we use ReactDOM.createRoot() to connect JSX with a real HTML element in the DOM.

Example:

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

root.render(<App />);

### 6. ****Explain how to use JavaScript expressions in JSX****

JavaScript expressions can be used inside JSX by wrapping them in curly braces {}.This allows dynamic values to be displayed.

Example:

const name = "Inigashree";

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

### 7.****Explain how to use inline CSS in JSX****

Inline CSS in JSX is written as a JavaScript object.  
Property names use camelCase instead of normal CSS syntax.

Example:

const headingStyle = { color: 'blue', fontSize: '24px' };

const element = <h1style={headingStyle}>Welcome</h1>;

We can also write the style object directly inside the style attribute:

<h1 style={{ backgroundColor: 'yellow' }}>Hello</h1>

**Code:**

App.js:

import React from "react";

import "./App.css";

function App() {

  const offices = [

    {

      name: "GreenSpace Hub",

      rent: 45000,

      address: "12, Park Avenue, Mumbai",

      img: "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRLe4GmRON8mAKtUofgE3IVB15ua-unX3upvg&s",

    },

    {

      name: "Skyline WorkBay",

      rent: 72000,

      address: "88, Corporate Road, Delhi",

      img: "data:image/jpeg;base64,/Z",

    },

    {

      name: "OceanView Offices",

      rent: 58000,

      address: "5, Marina Bay, Chennai",

      img: "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTlyJ1JKyNvPaKxjqTkQRLbpDcaZfpLh3HEKw&s",

    },

  ];

  return (

    <div className="container">

      <h1 className="heading">Office Space Rental Portal</h1>

      <div className="office-list">

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

          <div className="office-card" key={index}>

            <img src={office.img} alt="office" className="office-img" />

            <h2>{office.name}</h2>

            <p>

              <strong>Rent:</strong>{" "}

              <span

                style={{

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

                  fontWeight: "bold",

                }}

              >

                ₹{office.rent}

              </span>

            </p>

            <p>

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

            </p>

          </div>

        ))}

      </div>

    </div>

  );

}

export default App;

App.css:

.container {

  text-align: center;

  padding: 40px;

  background: #f8f9fa;

}

.heading {

  color: #2c3e50;

  margin-bottom: 40px;

}

.office-list {

  display: flex;

  justify-content: center;

  gap: 40px;

  flex-wrap: wrap;

}

.office-card {

  background: white;

  padding: 20px;

  border-radius: 12px;

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

  width: 300px;

}

.office-img {

  width: 100%;

  height: 180px;

  border-radius: 10px;

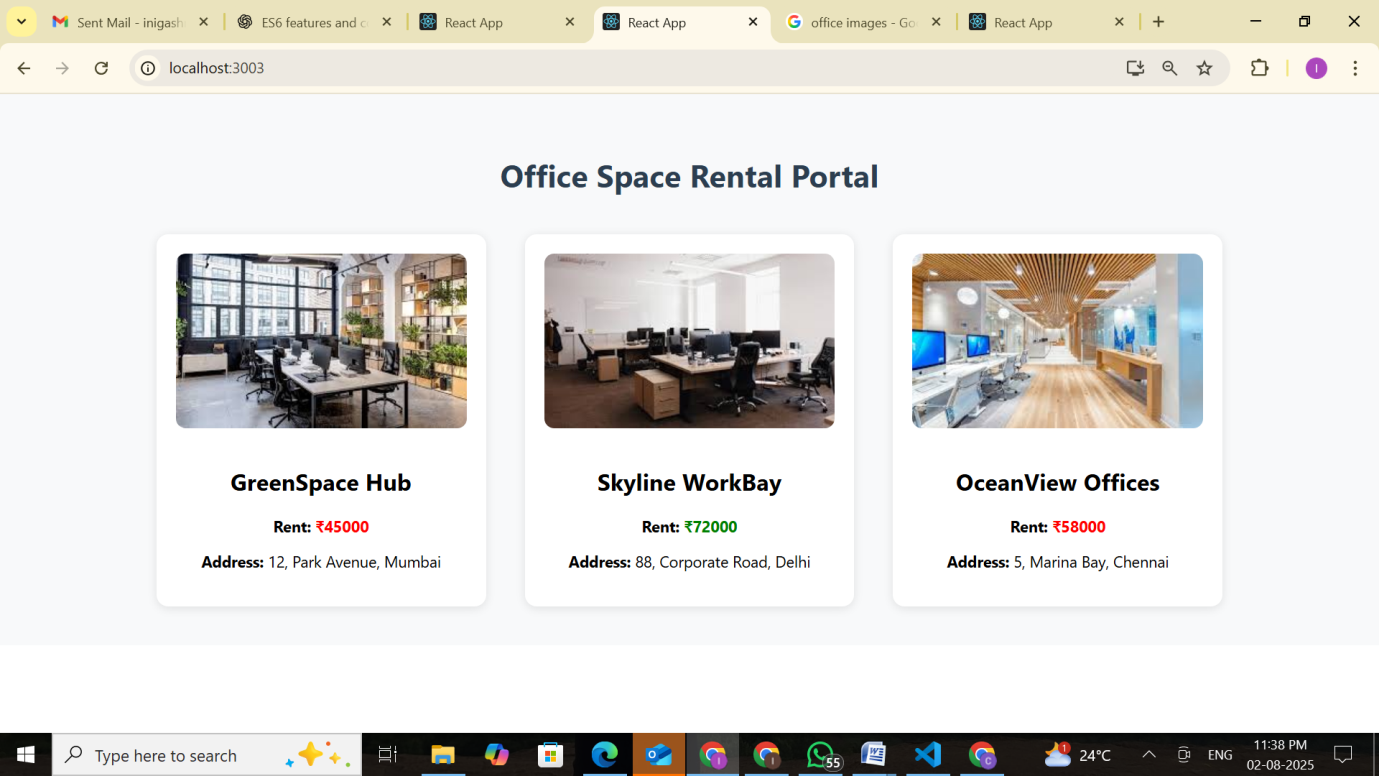
  object-fit: cover;

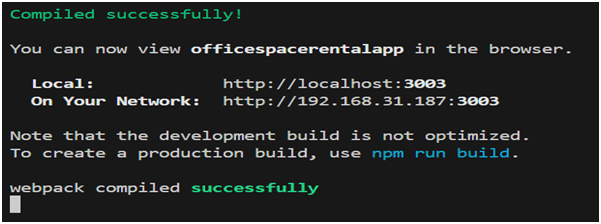
  margin-bottom: 15px;

}

**Output:**

Rent in **green** if ≥ ₹60,000, **red** if below





**11. ReactJS-HOL**

**1. Explain React Events**  
React events are mechanisms used to respond to user interactions like clicks, typing, mouse movement, etc. They are similar to native DOM events but React uses its own event system known as Synthetic Events to ensure consistent behavior across all browsers.

**2. Explain about Event Handlers**  
Event handlers are functions triggered when a specific event happens. In React, they are declared within components and assigned to elements through JSX. When the user interacts with the element, the corresponding function is executed.  
**Example:**

function showAlert() {

alert("You clicked the button!");

}

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

**3. Define Synthetic Event**  
A Synthetic Event is React’s abstraction over the browser’s native event. It ensures event handling works uniformly in all browsers. React optimizes performance by reusing the event object through event pooling.

**4. Identify React Event Naming Convention**  
React uses camelCase for naming events, unlike standard HTML. Also, in React, functions are passed directly, not strings.

* React: onMouseOver={handleHover}
* HTML: onmouseover="handleHover()"  
  This ensures JSX is consistent with JavaScript function handling rules.

**Code:**

App.js:

import React, { useState } from 'react';

import './App.css';

function App() {

  const [count, setCount] = useState(0);

  const [inr, setInr] = useState('');

  const [euro, setEuro] = useState('');

  const increment = () => {

    setCount(prev => prev + 1);

  };

  const decrement = () => {

    setCount(prev => prev - 1);

  };

  const sayHello = () => {

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

  };

  const handleIncrementClick = () => {

    increment();

    sayHello();

  };

  const sayWelcome = (msg) => {

    alert(msg);

  };

  const handleSynthetic = (e) => {

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

    alert("I was clicked");

  };

  const handleSubmit = (e) => {

    e.preventDefault();

    const result = (parseFloat(inr) \* 0.00984).toFixed(2);

    setEuro(result);

  };

  return (

    <div className="container">

      <h1>React Event Handling App</h1>

      <div className="counter-box">

        <p>Counter Value: {count}</p>

        <button onClick={handleIncrementClick}>Increment</button>

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

      </div>

      <div className="section">

        <button onClick={() => sayWelcome("Welcome to React Lab!")}>

          Say Welcome

        </button>

      </div>

      <div className="section">

        <button onClick={handleSynthetic}>Click me</button>

      </div>

      <div className="section">

        <h2>Currency Converter (INR ➡ Euro)</h2>

        <form onSubmit={handleSubmit}>

          <input

            type="number"

            value={inr}

            onChange={(e) => setInr(e.target.value)}

            placeholder="Enter INR"

            required

          />

          <button type="submit">Convert</button>

        </form>

        {euro && <p>Converted Amount: €{euro}</p>}

      </div>

    </div>

  );

}

export default App;

App.css:

body {

  margin: 0;

  padding: 0;

  font-family: 'Segoe UI', sans-serif;

  background-color: #f4f6f8;

}

.container {

  max-width: 600px;

  margin: 40pxauto;

  background: white;

  padding: 30px;

  border-radius: 12px;

  box-shadow: 0015pxrgba(0, 0, 0, 0.1);

}

h1 {

  color: #333;

  text-align: center;

}

.section, .counter-box {

  margin: 20px0;

  text-align: center;

}

button {

  margin: 5px;

  padding: 10px15px;

  background-color: #4caf50;

  color: white;

  border: none;

  border-radius: 5px;

  cursor: pointer;

}

button:hover {

  background-color: #388e3c;

}

input {

  padding: 10px;

  margin-right: 10px;

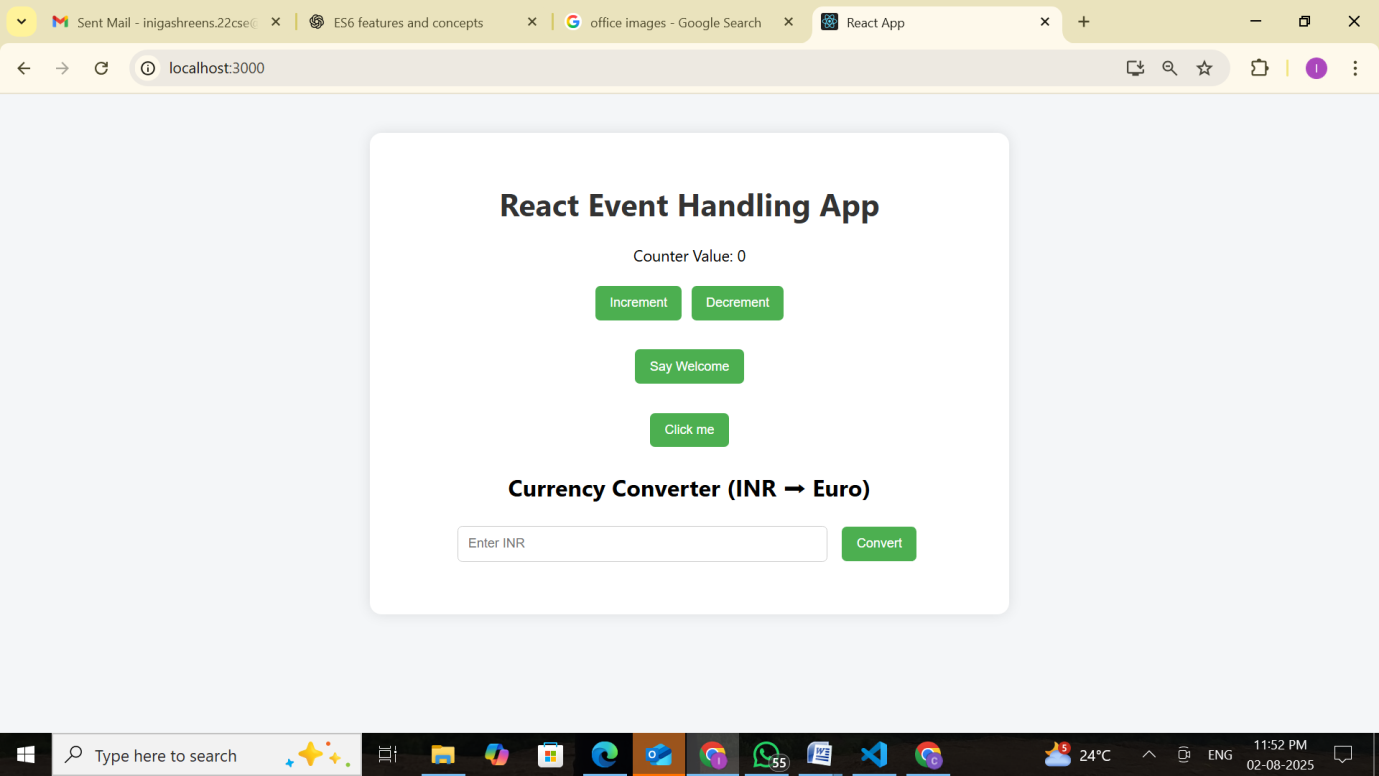
  width: 60%;

  border: 1pxsolid#ccc;

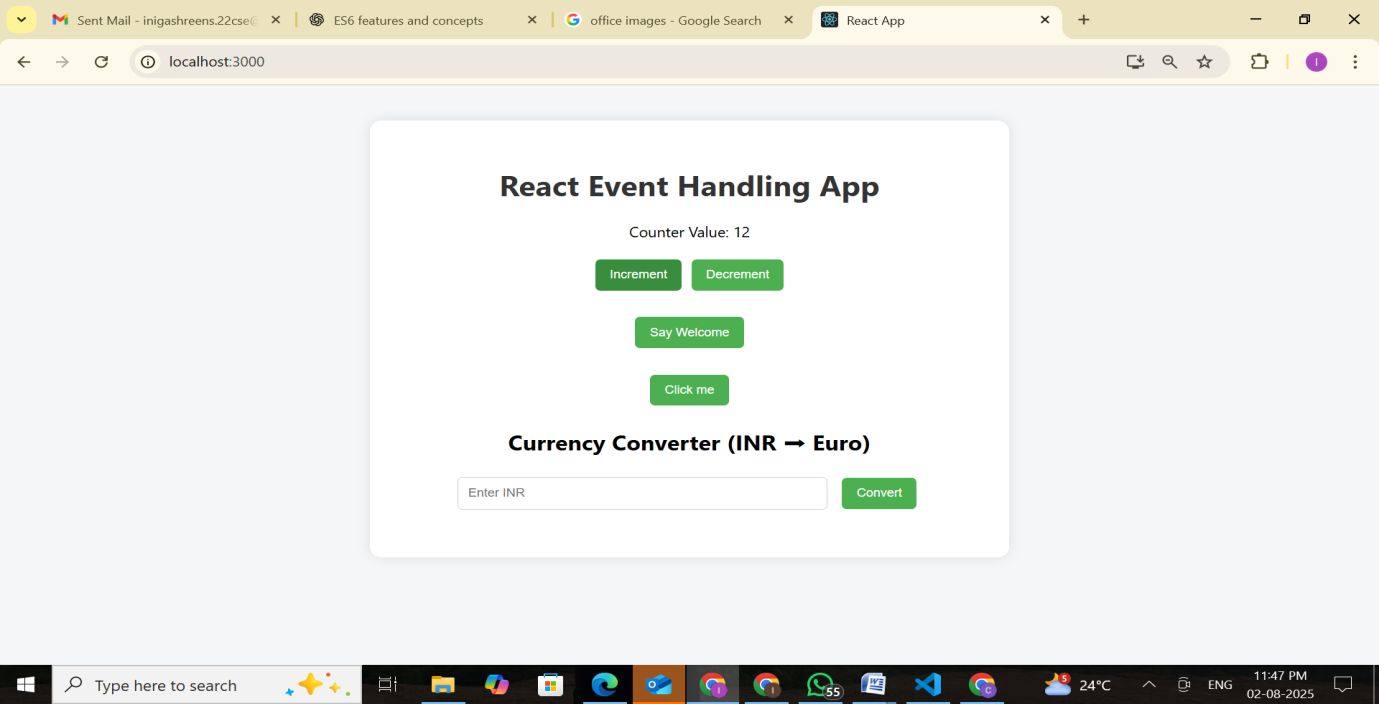
  border-radius: 5px;

}

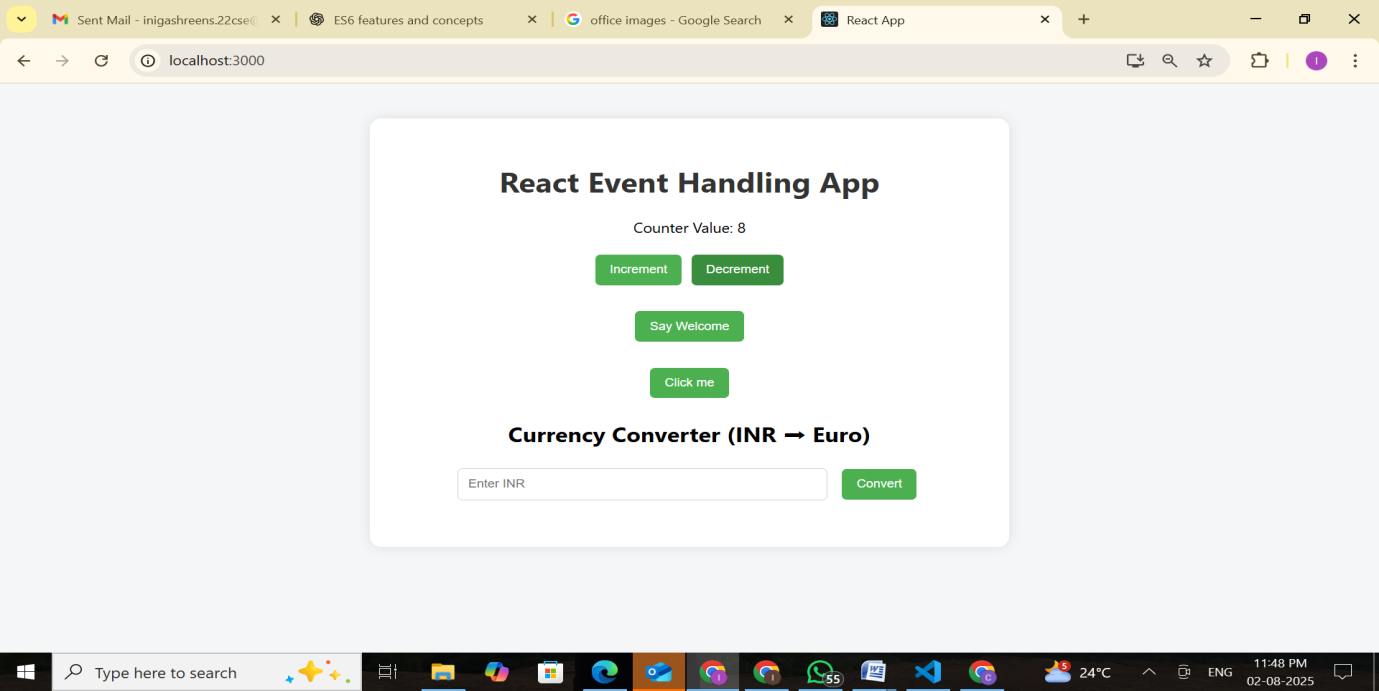
**Output:**



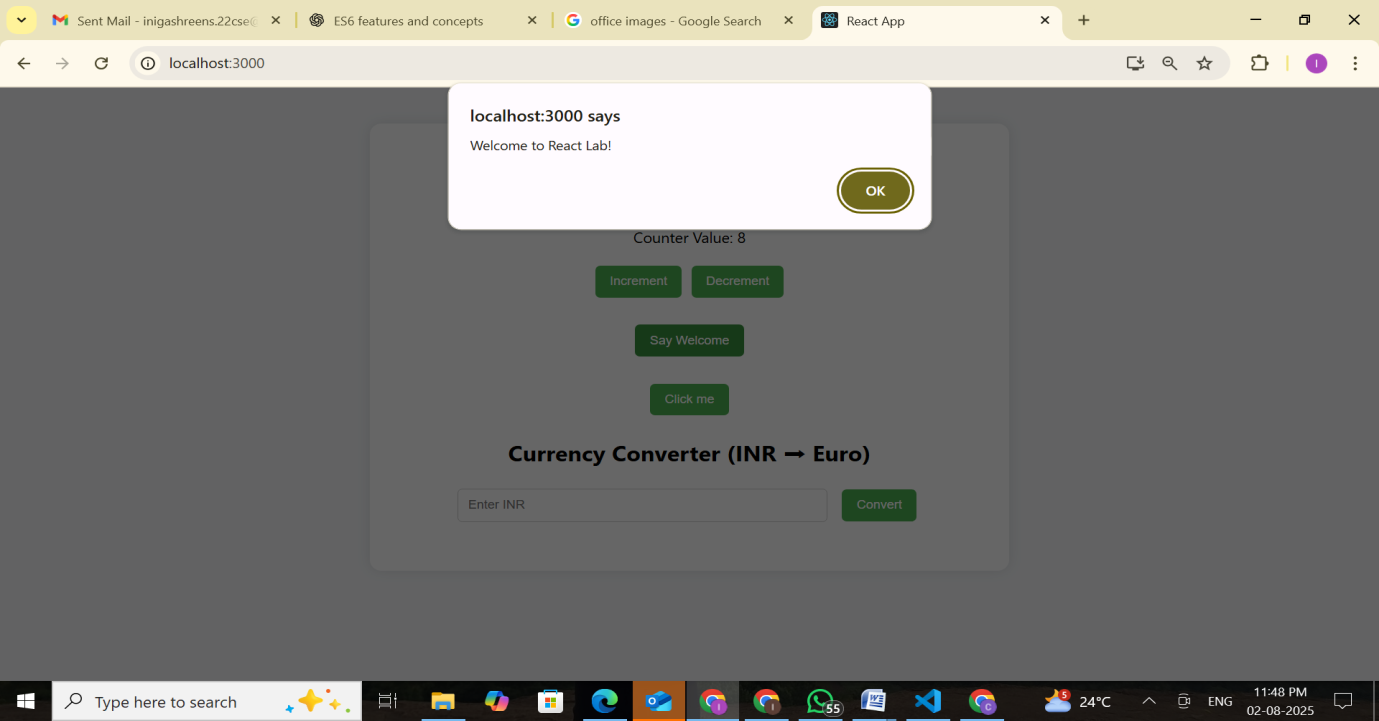
Increment:



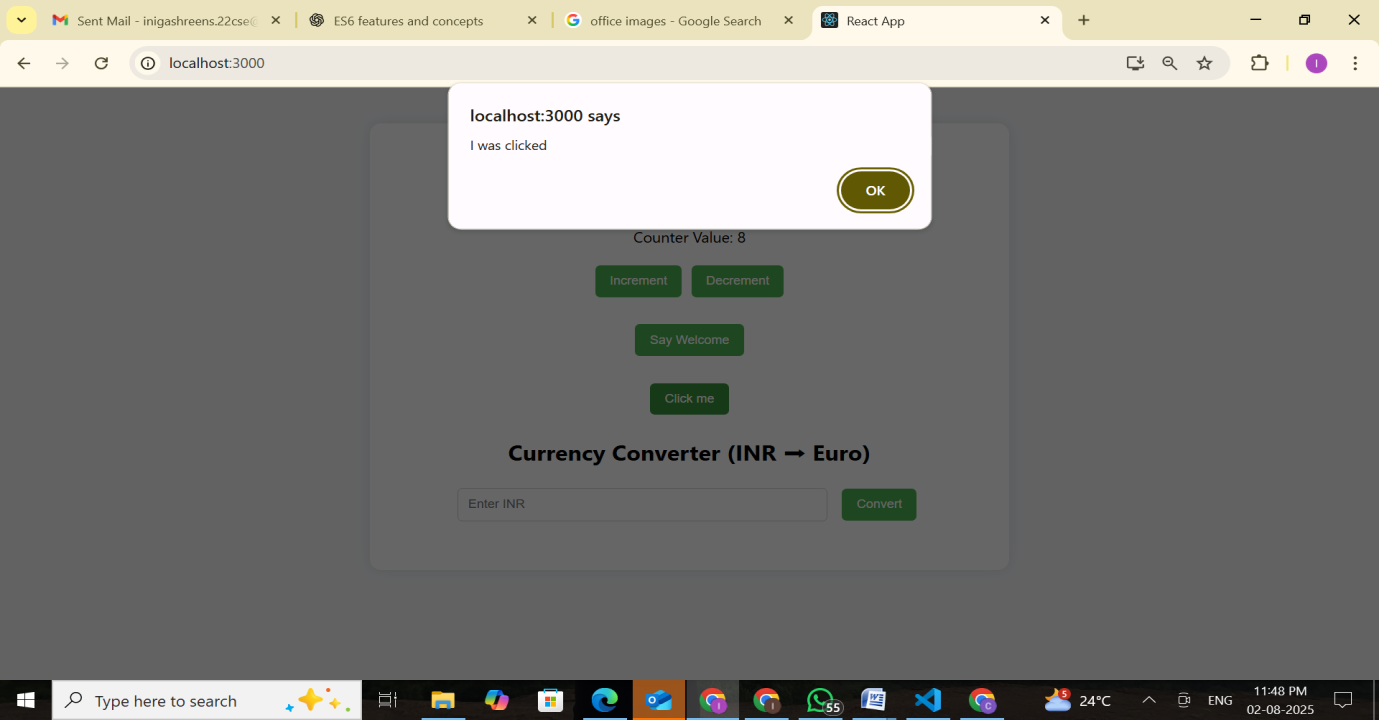
Decrement:



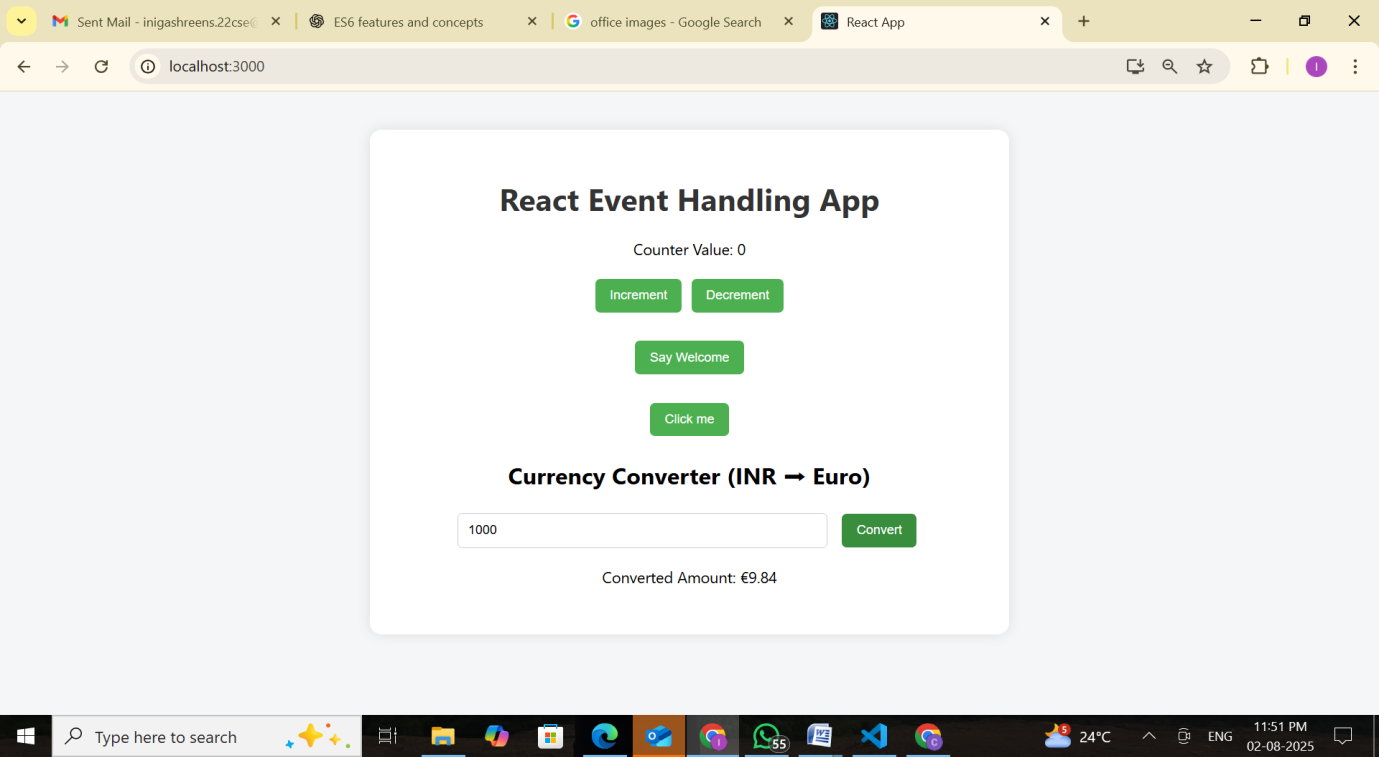
Say Welcome:



Click me:



Currency converter (Indian to Euro):



**12. ReactJS-HOL**

1. **Explain about conditional rendering in React**

Conditional rendering in React is the practice of showing different components or elements depending on specific conditions. Similar to JavaScript conditionals like if, else, ternary (? :), and logical &&, React uses these to dynamically control what appears in the UI.  
For example, based on user authentication, loading status, or errors, you can show or hide elements.  
**Example:**

{isLoggedIn ? <Dashboard /> : <Login />}

Here, if isLoggedIn is true, the Dashboard is shown; otherwise, the Login screen is displayed.

1. **Define element variables**

Element variables are variables used to store JSX code before it's returned in the component’s render method. This makes conditional rendering cleaner and more manageable.  
Instead of writing conditions directly inside JSX, you assign the JSX to a variable and then render that variable.  
**Example:**

let content;

if (isOnline) {

content = <p>User is online</p>;

} else {

content = <p>User is offline</p>;

}

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

This approach improves code readability and keeps logic separated from markup.

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

There are a few strategies to stop a React component from rendering when it’s not needed:

* **Return null** from the component itself:

if (!props.show) {

return null;

}

* **Use logical AND (&&) operator**:

{isVisible && <MyComponent />}

* **Avoid rendering it in the parent JSX**:  
  Skip including the component altogether based on conditions in the parent.

These methods help reduce unnecessary processing and enhance performance, particularly for components with heavy logic or rendering tasks.

**Code:**

App.js:

import React, { useState } from 'react';

import GuestPage from './components/GuestPage';

import UserPage from './components/UserPage';

import './App.css';

function App() {

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

  const handleLogin = () => setIsLoggedIn(true);

  const handleLogout = () => setIsLoggedIn(false);

  return (

    <div className="App">

      <header>

        <h1>✈️ Ticket Booking App</h1>

        {isLoggedIn ? (

          <button onClick={handleLogout} className="logout-btn">Logout</button>

        ) : (

          <button onClick={handleLogin} className="login-btn">Login</button>

        )}

      </header>

      {isLoggedIn ? <UserPage /> : <GuestPage />}

    </div>

  );

}

export default App;

FlightDetails.js:

import React from 'react';

const flights = [

  { id: 1, name: 'IndiGo 6E-203', from: 'Chennai', to: 'Delhi', price: 4800, time: '06:30 AM' },

  { id: 2, name: 'Air India AI-431', from: 'Mumbai', to: 'Bangalore', price: 5200, time: '09:15 AM' },

  { id: 3, name: 'SpiceJet SG-325', from: 'Kolkata', to: 'Hyderabad', price: 4500, time: '02:45 PM' }

];

function FlightDetails({ showBooking }) {

  return (

    <div>

      <h3>Available Flights</h3>

      <ul>

        {flights.map(flight => (

          <li key={flight.id} className="flight">

            <p><strong>{flight.name}</strong></p>

            <p>{flight.from} ➡️ {flight.to}</p>

            <p>Time: {flight.time}</p>

            <p>Price: ₹{flight.price}</p>

            {showBooking && <button className="book-btn">Book Ticket</button>}

          </li>

        ))}

      </ul>

    </div>

  );

}

export default FlightDetails;

GuestPage.js:

import React from 'react';

import FlightDetails from './FlightDetails';

function GuestPage() {

  return (

    <div>

      <h2>Welcome Guest!</h2>

      <p>Please log in to book tickets.</p>

      <FlightDetails />

    </div>

  );

}

export default GuestPage;

UserPage.js:

import React from 'react';

import FlightDetails from './FlightDetails';

function UserPage() {

  return (

    <div>

      <h2>Welcome, User!</h2>

      <FlightDetails showBooking={true} />

    </div>

  );

}

export default UserPage;

App.css:

.App {

  text-align: center;

  font-family: Arial, sans-serif;

}

header {

  background-color: #282c34;

  padding: 1rem;

  color: white;

  display: flex;

  justify-content: space-between;

  align-items: center;

}

button {

  padding: 8px 16px;

  margin-left: 1rem;

  border: none;

  border-radius: 4px;

  cursor: pointer;

}

.login-btn {

  background-color: #4CAF50;

  color: white;

}

.logout-btn {

  background-color: #f44336;

  color: white;

}

ul {

  list-style-type: none;

  padding: 0;

  display: flex;

  gap: 1rem;

  justify-content: center;

  flex-wrap: wrap;

}

.flight {

  border: 1px solid #ccc;

  border-radius: 10px;

  padding: 1rem;

  width: 220px;

  background-color: #f9f9f9;

}

.book-btn {

  margin-top: 10px;

  padding: 6px 12px;

  background-color: #007bff;

  color: white;

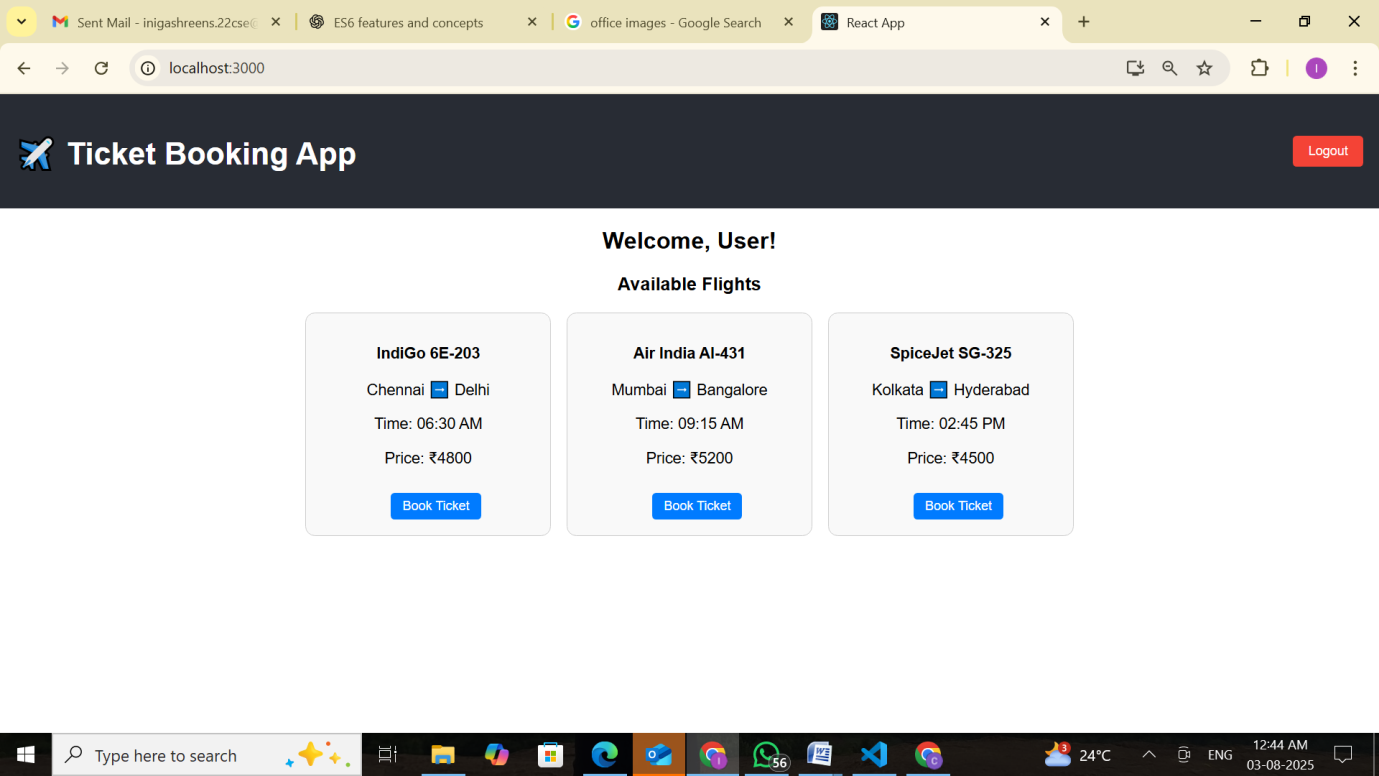
  border: none;

  border-radius: 4px;

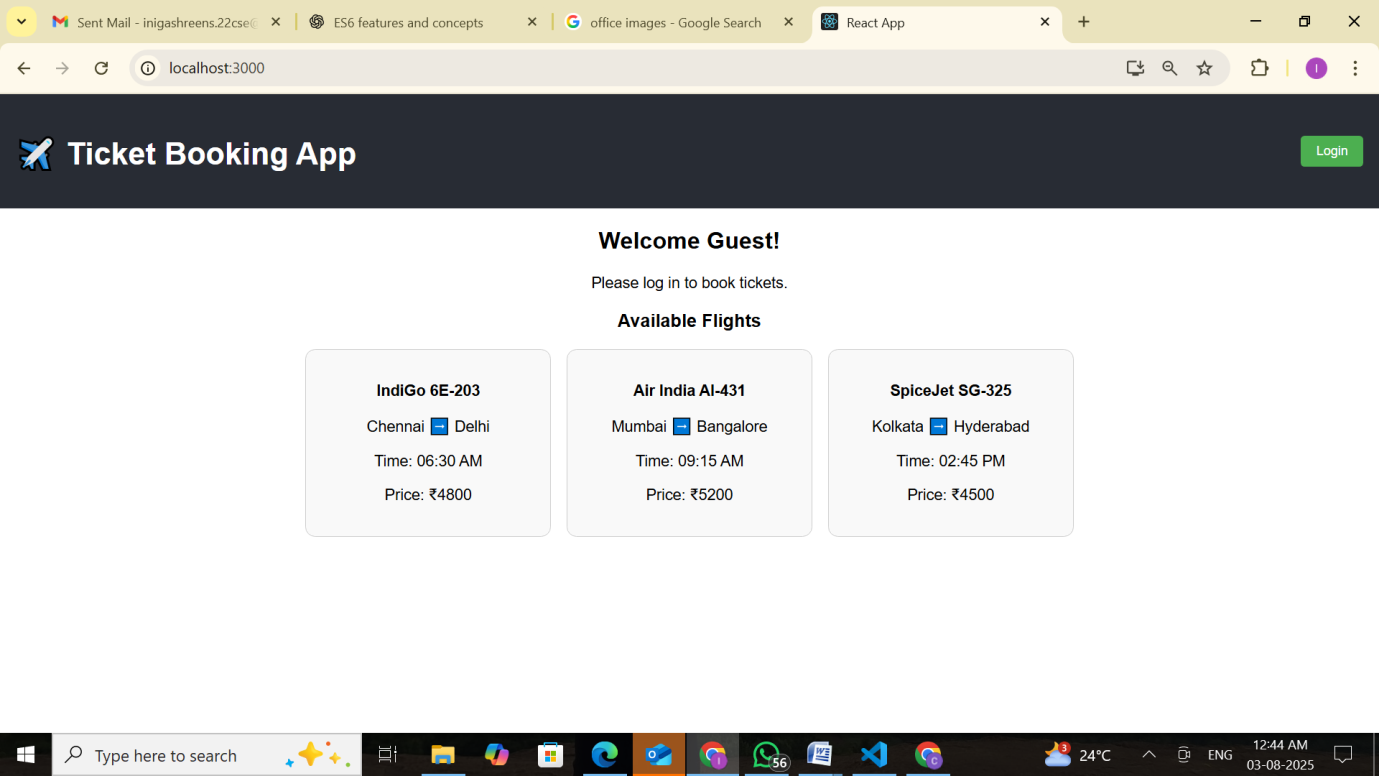
}

**Output:**

After login:



After logout:



**13. ReactJS-HOL**

**1.Explain various ways of conditional rendering**  
Conditional rendering in React means displaying content based on conditions (like if a user is logged in, data is available, etc.). It uses JavaScript logic to determine what to show in the UI.

**Different ways:**

1. **If-Else Statement:**  
   Use when you want to decide what to render before the return block.

if (isLoggedIn) return <Dashboard />;

else return <Login />;

1. **Ternary Operator (condition ? true : false)**  
   Short syntax for rendering one of two elements.

{isAdmin ? <AdminPanel /> : <UserPanel />}

1. **Logical AND (&&)**  
   Renders only if the condition is true; nothing otherwise.

{showBanner && <Banner />}

1. **IIFE (Immediately Invoked Function Expression)**  
   Useful when the condition is complex and needs logic inside JSX.

{(() => {

if (role === 'admin') return <Admin />;

if (role === 'editor') return <Editor />;

return <Viewer />;

})()}

1. **Switch Statement:**  
   Cleaner for multiple condition types.

switch (status) {

case 'loading': return <Loading />;

case 'success': return <Success />;

case 'error': return <Error />;

default: return null;

}

**2.Explain how to render multiple components**

React allows you to render more than one component at once. But you must wrap them inside a single parent — either a div, section, or React.Fragment (<>...</>). This keeps JSX valid and organized.

**Example:**

<>

<Navbar />

<Content />

<Footer />

</>

**3.Define list component**  
A list component dynamically displays multiple items using map(). Instead of writing multiple elements manually, React lets you generate them from an array. This promotes reusability and scalability.

**Example:**

function NameList({ names }) {

return (

<ul>

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

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

))}

</ul>

);

}

**4.Explain about keys in React applications**

Keys are unique identifiers assigned to elements when rendering lists. They help React track which items are added, changed, or removed. Without keys, React may re-render inefficiently or incorrectly.

**Best Practices:**

* Use **unique IDs** from the data (like user.id).
* Avoid using array index unless data has no unique ID.

**Example:**

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

**5.Explain how to extract components with keys**When rendering lists, you often break out each item into a separate component (called *component extraction*). The key must always be applied **in the parent component** where the .map() is used — not inside the child.

**Example:**

function ProductList({ products }) {

return products.map(p => <ProductCard key={p.id} data={p} />);

}

**6.Explain React Map, map() function**  
The map() function is a standard JavaScript method used in React to render lists of elements. It loops through an array and returns a new array of JSX elements. It's core to dynamic UI rendering.

**Usage in React:**

* Used for lists, tables, grids, etc.
* Requires a unique key for each element.

**Example:**

const tasks = ['Read', 'Write', 'Code'];

const taskList = tasks.map((task, i) => <li key={i}>{task}</li>);

**Code:**

App.js:

import React, { useState } from 'react';

import BookDetails from './components/BookDetails';

import BlogDetails from './components/BlogDetails';

import CourseDetails from './components/CourseDetails';

function App() {

  const [selected, setSelected] = useState('');

  return (

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

      <h1>Welcome to Blogger App</h1>

      <div>

        <button onClick={() => setSelected('book')}>Book</button>

        <button onClick={() => setSelected('blog')}>Blog</button>

        <button onClick={() => setSelected('course')}>Course</button>

      </div>

      {/\* Conditional Rendering Methods \*/}

      {/\* 1. Ternary Operator \*/}

      {selected === 'book' ? <BookDetails /> :

       selected === 'blog' ? <BlogDetails /> :

       selected === 'course' ? <CourseDetails /> :

       <p>Please select an option above.</p>}

    </div>

  );

}

export default App;

BookDetails.js:

import React from 'react';

function BookDetails() {

  return (

    <div>

      <h2>📚 Book Details</h2>

      <p>Title: React Made Easy</p>

      <p>Author: Jane Doe</p>

    </div>

  );

}

export default BookDetails;

BlogDetails.js:

import React from 'react';

function BlogDetails() {

  return (

    <div>

      <h2>✍️ Blog Details</h2>

      <p>Title: Understanding JSX</p>

      <p>Published: August 2025</p>

    </div>

  );

}

export default BlogDetails;

CourseDetails.js:

import React from 'react';

function CourseDetails() {

  return (

    <div>

      <h2>🎓 Course Details</h2>

      <p>Course: Frontend Development</p>

      <p>Duration: 4 weeks</p>

    </div>

  );

}

export default CourseDetails;

App.css:

body {

  font-family: Arial, sans-serif;

  background-color: #fff;

  padding: 20px;

  margin: 0;

}

.container {

  display: flex;

  justify-content: space-around;

  align-items: flex-start;

  text-align: left;

}

.box {

  border-left: 3px solid green;

  padding-left: 20px;

  width: 30%;

}

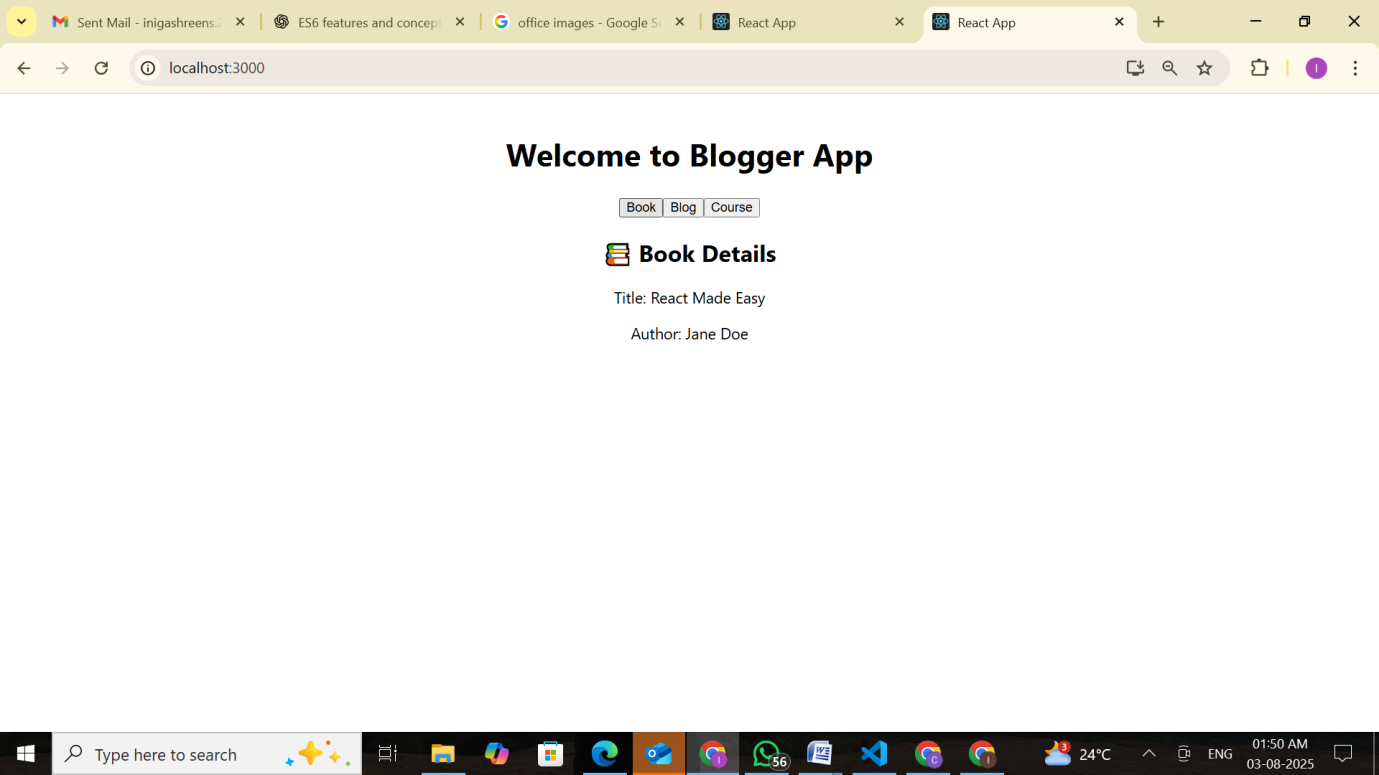
h2 {

  text-align: center;

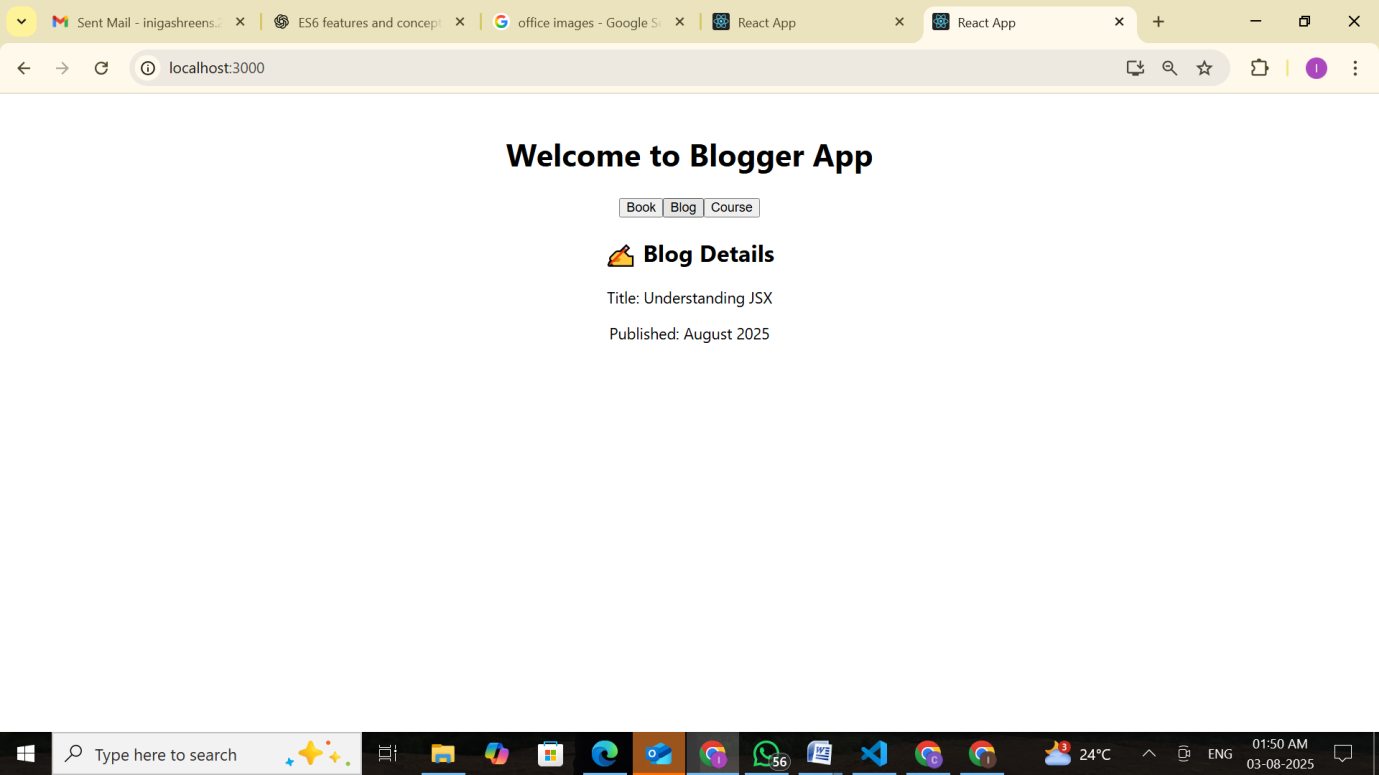
}

**Output:**

Book Details:



Blog Details:



Course Details:

