9. ReactJS-HOL

1. Features of ES6 (ECMAScript 2015)

ES6 introduced several new features that made JavaScript more powerful and easier to write. Some key features include:

- let and const for block-scoped variable declarations.
- Arrow functions for shorter syntax in writing functions.
- Classes to write object-oriented code in a cleaner way.
- Template literals to allow embedded expressions and multi-line strings.
- Destructuring to extract values from arrays or objects easily.
- Default parameters for setting function parameters with default values.
- Spread and Rest operators to simplify array and object handling.
- Modules to import and export code between files.
- Promises for easier asynchronous programming.
- Map and Set data structures for storing unique values and key-value pairs.

2. Explain JavaScript let

The let keyword is used to declare variables in JavaScript. A variable declared with let is block scoped, which means it is only accessible within the block in which it is defined (like inside a loop or a function). It does not get hoisted in the same way as var, so using it before declaration will cause an error. It also allows re-assigning values but not re-declaring the same variable within the same scope.

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 is used to declare constants, which means once a value is assigned, it cannot be changed. Just like let, const is also block scoped. However, while the variable reference cannot change, objects and arrays declared with const can still be modified internally (their contents can be changed, but the reference can't be reassigned).

5. ES6 Class Fundamentals

In ES6, classes provide a more clear and structured way to create objects and handle inheritance. A class is created using the class keyword. It can have a constructor() method to initialize properties, and additional methods can be defined inside the class. Classes are not hoisted, and the syntax is more readable compared to older function-based prototypes.

Example:

```
class Car {

  constructor(name) {
    this.name = name;
  }

  drive() {
    return `${this.name} is driving`;
  }
}
```

6. ES6 Class Inheritance

ES6 allows one class to inherit from another using the extends keyword. The super() function is used inside the child class constructor to call the parent class constructor. This helps in code reusability and better organization.

Example:

```
class Vehicle {

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

```
class Bike extends Vehicle {
  constructor(name, brand) {
    super(name);
    this.brand = brand;
  }
}
```

7. Define ES6 Arrow Functions

Arrow functions offer a shorter syntax to write functions. They are written using => (fat arrow) and do not have their own this context, which makes them useful in callbacks and methods where the value of this needs to remain consistent.

Example:

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

Arrow functions are cleaner and avoid common issues with this.

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.

  ```
  js
  CopyEdit
  let mySet = new Set([1, 2, 3, 2]);
  // Output: Set {1, 2, 3}
  ```

- 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().

  ```
  js
  CopyEdit
  let myMap = new Map();
  myMap.set('name', 'John');
  myMap.set(1, 'One');
  ```

Code:

App.js:

```
import React from 'react';
import './App.css';
import ListofPlayers from './ListofPlayers';
import IndianPlayers from './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;
```

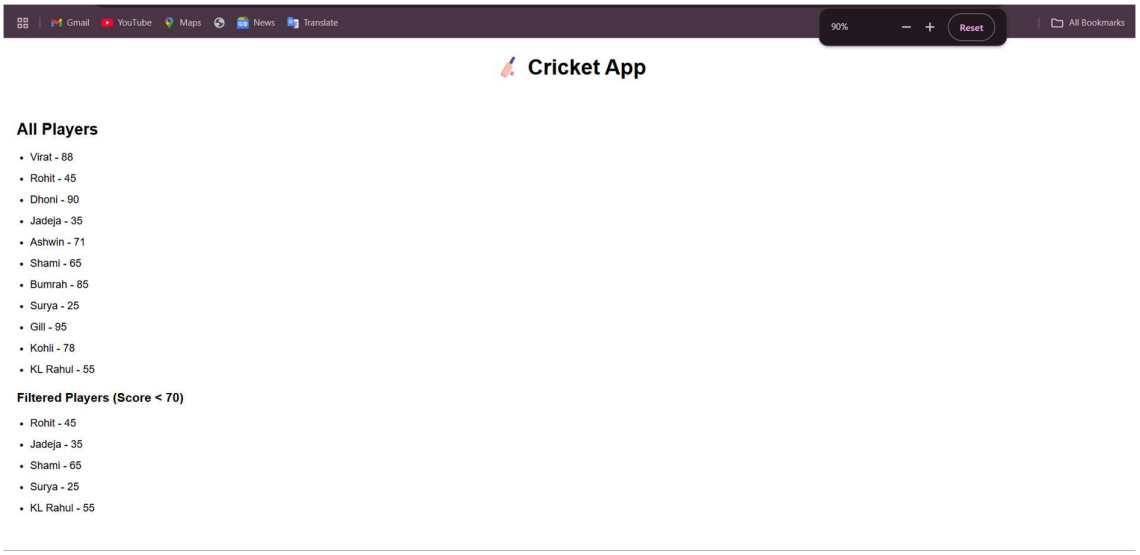
IndianPlayers.js:

```
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>
```

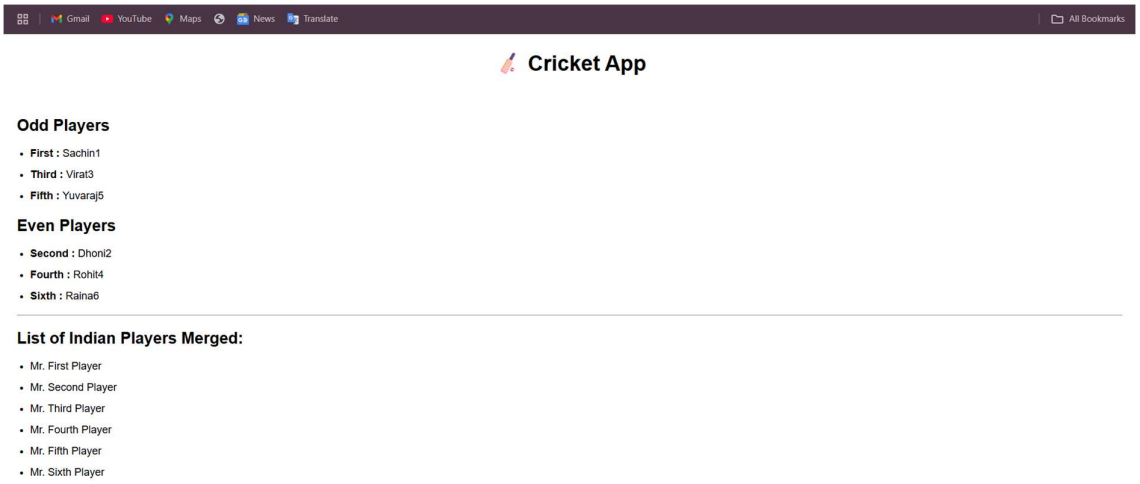
```
    <ul>
      {oddPlayers.map((name, i) => (
        <li key={i}>
          <strong>{oddLabels[i]} :</strong> {name}
        </li>
      ))}
    </ul>

    <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 = "Shivendra";

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 = <h1 style={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 officeList = [
    {
      Name: 'DBS',
      Rent: 50000,
      Address: 'Chennai',
      Image: 'https://officesnapshots.com/wp-content/uploads/2020/07/dbs-bank-
offices-hyderabad-12.jpg'
    },
    {
      Name: 'Regus',
      Rent: 75000,
      Address: 'Bangalore',
      Image:
'https://assets.iwgplc.com/image/upload/c_fill,f_auto,q_auto,h_211,w_auto,ar_1
78:100/CentreImagery/Fallbacks/P1_Marketing_Selects/Regus_standard_SINGAPORE_G
uoco_Tower_4058_Singapore_CoworkingSpace.jpg'
    },
    {
      Name: 'WeWork',
      Rent: 58000,
      Address: 'Hyderabad',
      Image:
'https://ctfassets.imgix.net/vh7r69kgcki3/1yD4Tmm83DWGhp6UzGId5z/baa2c10600153
343d3b72c24762ba571/Web_150DPI-20201217_WeWork_Km_5_Av_Las_Palmas_-
_Medellin_007.jpg'
    }
  ];

  return (
    <div className="App">
      <h2>Office Space , at Affordable Range</h2>

      <div className="card-container">
        {officeList.map((item, index) => {
          const rentStyle = {
            color: item.Rent < 60000 ? 'red' : 'green'
          };

          return (
            <div className="office-card" key={index}>
              <img src={item.Image} alt="Office Space" />
```

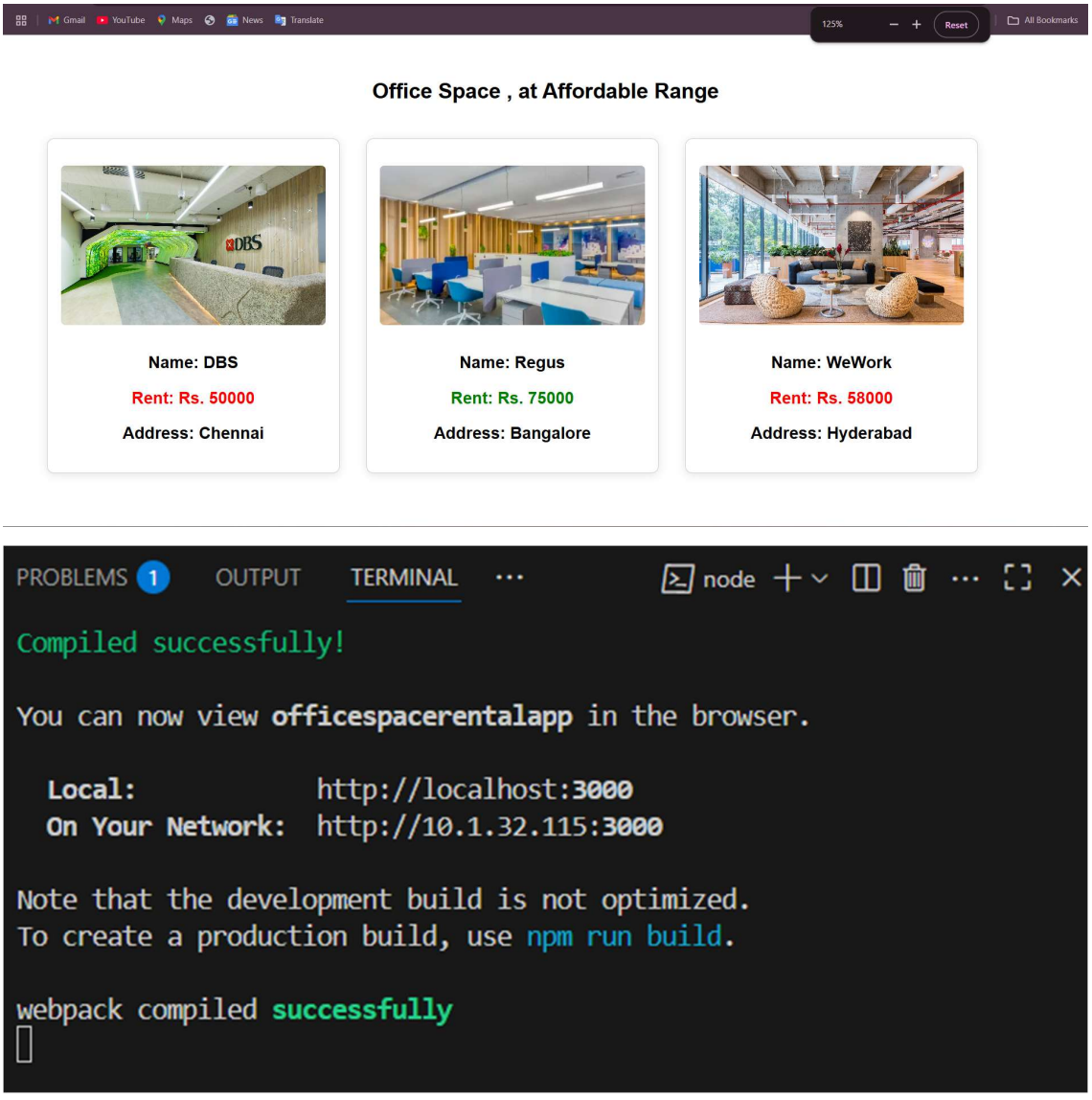
```
              <h3>Name: {item.Name}</h3>
              <h3 style={rentStyle}>Rent: Rs. {item.Rent}</h3>
              <h3>Address: {item.Address}</h3>
            </div>
          );
        })}
      </div>
    </div>
  );
}

export default App;
```

App.css:

```
.App {
  text-align: center;
  font-family: Arial, sans-serif;
  padding: 30px;
}

img {
  margin: 15px 0;
  border-radius: 8px;
}

.office-card img {
  width: 100%;
  height: 180px;
  object-fit: cover;
  border-radius: 5px;
  margin-bottom: 10px;
}

.card-container {
  display: flex;
  flex-direction: row;
  justify-content: flex-start;
  gap: 30px;
  overflow-x: auto;
  white-space: nowrap;
  padding: 20px;
}

.office-card {
  flex: 0 0 auto
  width: 300px;
  border: 1px solid #ccc;
```

```
  padding: 15px;
  border-radius: 8px;
  text-align: center;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
}
```

Output:



---

11. ReactJS-HOL

1. Explain React Events
React events are used to handle user interactions in a React application, such as clicks, form submissions, key presses, etc. These events are similar to DOM events in HTML but use React's own event system. React wraps the native events into a SyntheticEvent to ensure they behave consistently across different browsers.

2. Explain about Event Handlers
Event handlers are functions that define what should happen when a specific event occurs. In React, these handlers are usually defined in the component and passed to the JSX element as props. When the event (like a click or input change) is triggered, the associated function is executed.

Example:

```
function handleClick() {
  console.log("Clicked");
}
<button onClick={handleClick}>Click Me</button>
```

3. Define Synthetic Event
A Synthetic Event in React is a wrapper around the browser's native event. It is part of React's event delegation system and provides a consistent API regardless of browser differences. SyntheticEvent normalizes events to work the same way across all browsers and improves performance by pooling and reusing event objects.

4. Identify React Event Naming Convention
React uses camelCase for event names, unlike HTML which uses lowercase. Additionally, in React, event handlers are passed as functions instead of strings.

- **React syntax:** onClick={handleClick}
- **HTML syntax:** onclick="handleClick()" (not used in React)

This convention ensures better integration with JavaScript and JSX syntax.

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">
```

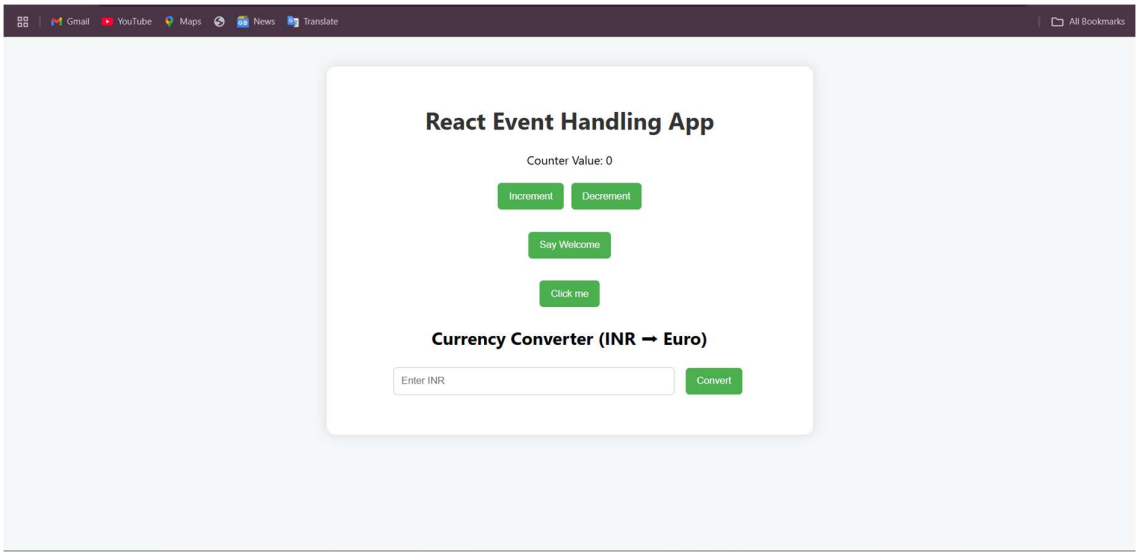
```
        <p>Counter Value: {count}</p>
        <button onClick={handleIncrementClick}>Increment</button>
        <button onClick={decrement}>Decrement</button>
      </div>

      <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:

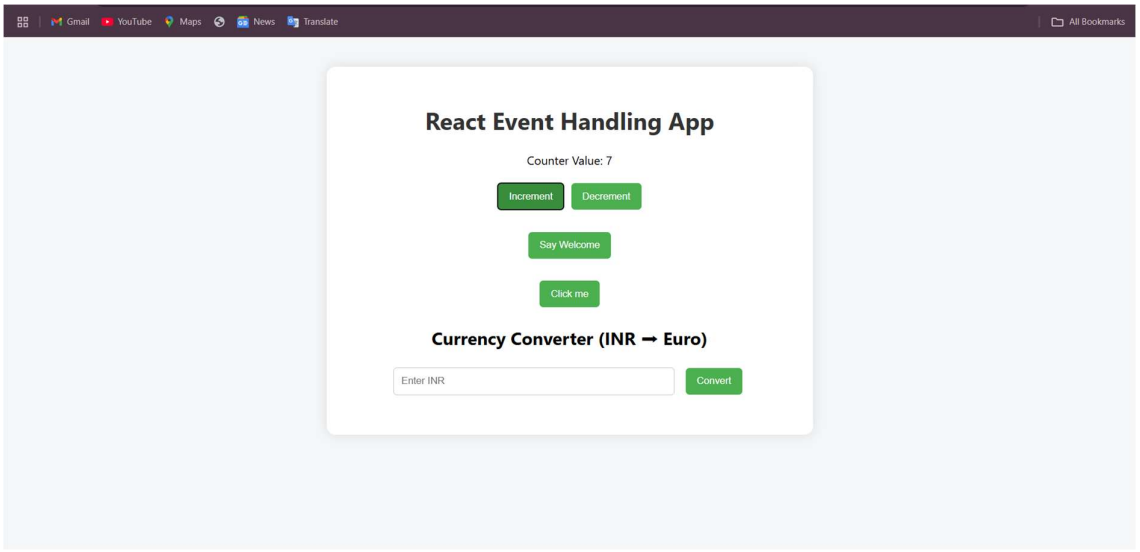
```
body {
  margin: 0;
  padding: 0;
  font-family: 'Segoe UI', sans-serif;
  background-color: #f4f6f8;
}

.container {
  max-width: 600px;
  margin: 40px auto;
  background: white;
```

```css
  padding: 30px;
  border-radius: 12px;
  box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
}

h1 {
  color: #333;
  text-align: center;
}

.section, .counter-box {
  margin: 20px 0;
  text-align: center;
}

button {
  margin: 5px;
  padding: 10px 15px;
  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: 1px solid #ccc;
  border-radius: 5px;
}
```

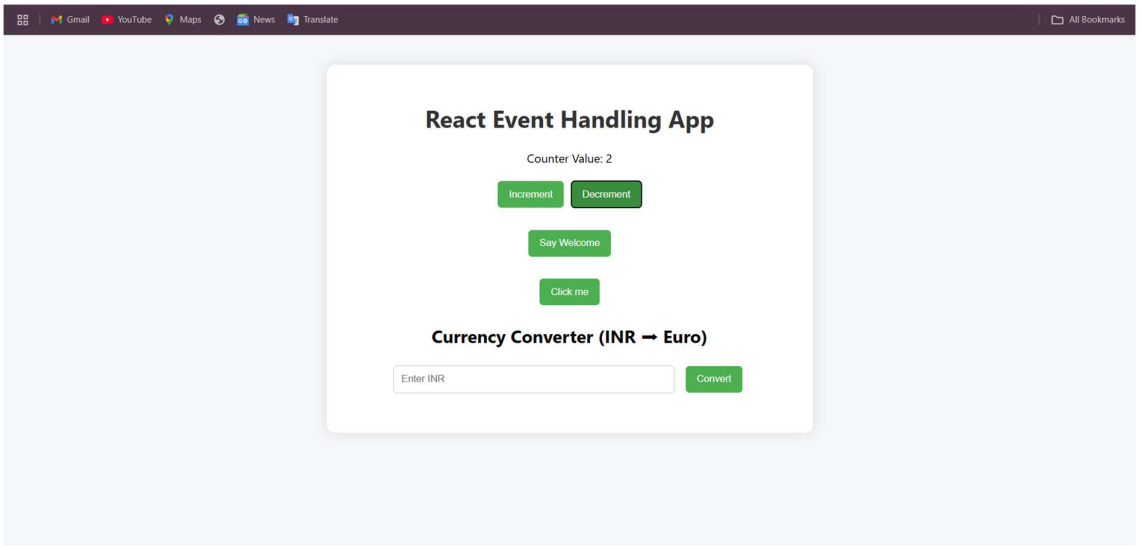
Output:



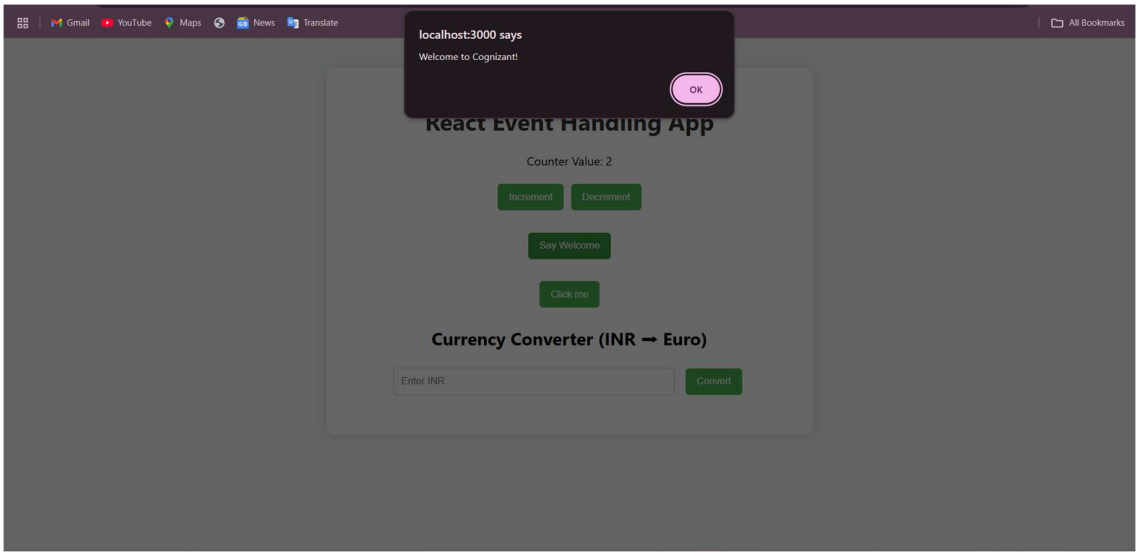
Increment:



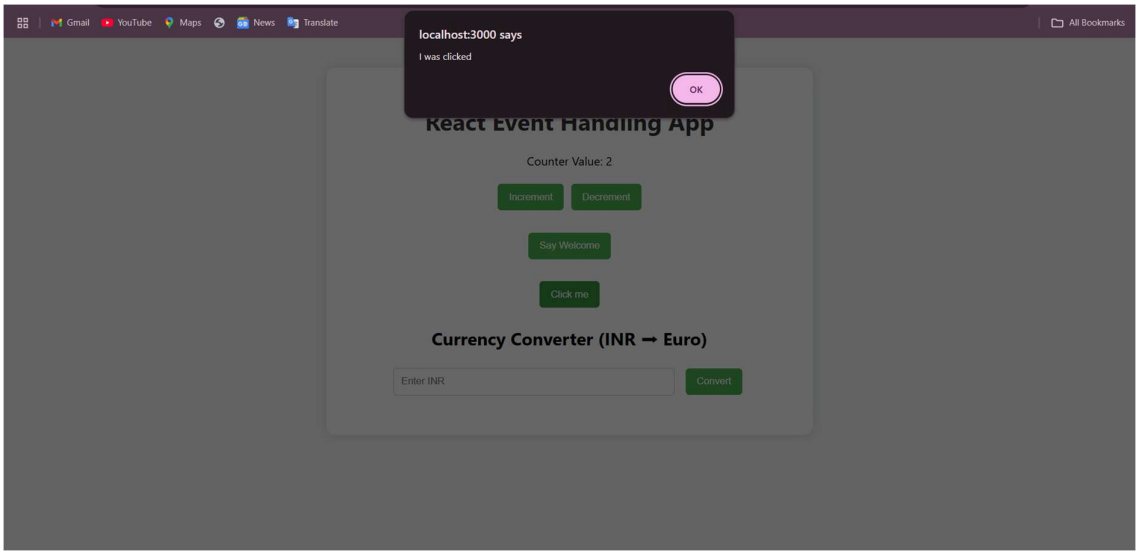
Decrement:



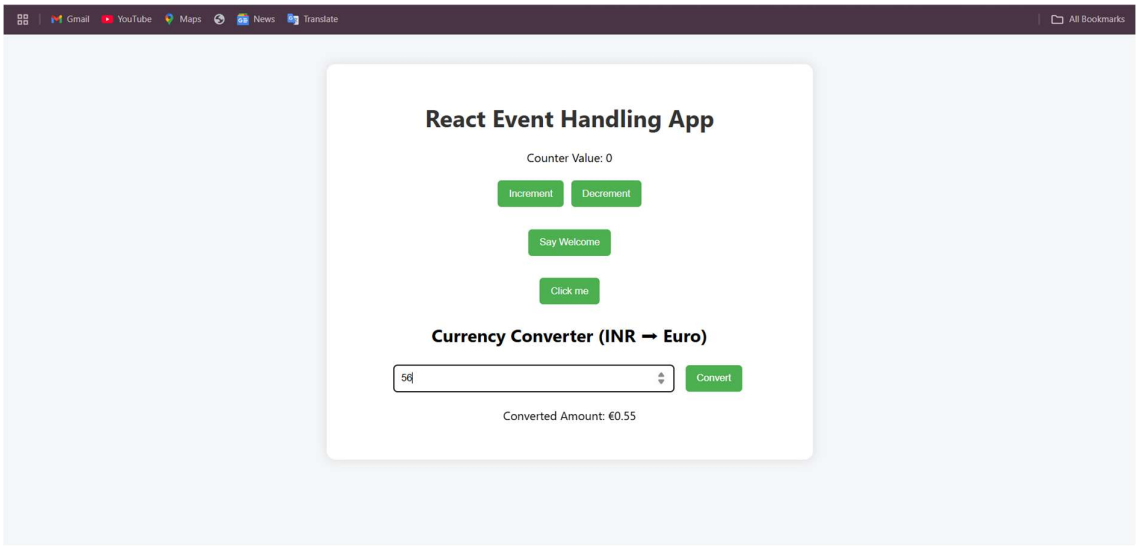
Say Welcome:



Click me:



Currency converter (Indian to Euro):



12. ReactJS-HOL

1. Explain about conditional rendering in React

In React, conditional rendering refers to the process of displaying different UI elements or components based on certain conditions. It works similarly to conditions in JavaScript using if, else, ternary operators, or logical &&. Instead of rendering everything at once, React allows developers to control what should be shown based on the application state or props. This is useful for showing loading indicators, error messages, user authentication views, or toggling content visibility.

Example:

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

Here, the Dashboard component will be rendered only if isLoggedIn is true, otherwise the Login component is shown.

2. Define element variables

Element variables in React are variables that are used to store JSX elements. This allows for cleaner and more readable code, especially when deciding what to render based on some condition. Instead of writing complex logic directly inside JSX, we can store the element in a variable and use it in the return() method.

Example:

```
let content;
if (isOnline) {
  content = <p>User is online</p>;
} else {
  content = <p>User is offline</p>;
}
return <div>{content}</div>;
```

This helps in breaking down complex UI logic into manageable parts and improves the structure of the render method.

3. Explain how to prevent components from rendering

To prevent components from rendering in React, we can use conditional statements that return null or avoid calling the component at all. When null is returned from a component, React does not render anything for that component.

Ways to prevent rendering:

1. Return null from the component

   ```
   if (!props.isVisible) {
     return null;
   }
   ```

2. Use short-circuit rendering

   ```
   {showComponent && <MyComponent />}
   ```

3. Avoid including the component in the JSX
   Control the flow in the parent and do not render the child component unless necessary.

Preventing unnecessary renders is useful for optimizing performance, especially when dealing with large or complex components.

Code:

App.js:

```
import React, { useState } from 'react';
import Navbar from './components/Navbar';
import GuestView from './components/GuestView';
import UserView from './components/UserView';
import FlightList from './components/FlightList';

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

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

  return (
    <div className="app-container">
      <Navbar isLoggedIn={isLoggedIn} onLogin={handleLogin}
onLogout={handleLogout} />
      <FlightList />
      {isLoggedIn ? <UserView /> : <GuestView />}
    </div>
  );
}

export default App;
```

FlightList.js:

```
import React from 'react';

function FlightList() {
  const flights = [
    { id: 1, from: "Chennai", to: "Delhi", time: "10:00 AM" },
    { id: 2, from: "Mumbai", to: "Bangalore", time: "12:00 PM" },
    { id: 3, from: "Kolkata", to: "Hyderabad", time: "03:30 PM" }
  ];

  return (
    <div className="flight-list">
      <h3>Available Flights</h3>
      <ul>
        {flights.map(flight => (
          <li key={flight.id}>
            ✈ {flight.from} → {flight.to} at {flight.time}
          </li>
```

```
        ))}
      </ul>
    </div>
  );
}

export default FlightList;
```

GuestView.js:

```
import React from 'react';

function GuestView() {
  return (
    <div className="guest-view">
      <h3>Welcome, Guest!</h3>
      <p>Please log in to book your tickets.</p>
    </div>
  );
}

export default GuestView;
```

Navbar.js:

```
import React from 'react';

function Navbar({ isLoggedIn, onLogin, onLogout }) {
  return (
    <div className="navbar">
      <h2>Ticket Booking App</h2>
      <div>
        {isLoggedIn ? (
          <button className="nav-btn" onClick={onLogout}>Logout</button>
        ) : (
          <button className="nav-btn" onClick={onLogin}>Login</button>
        )}
      </div>
    </div>
  );
}

export default Navbar;
```

UserView.js:

```
import React from 'react';

function UserView() {
  return (
    <div className="user-view">
      <h3>Welcome, User!</h3>
      <p>You can now book tickets.</p>
    </div>
  );
}

export default UserView;
```

App.css:

```
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background: #f4f4f9;
  margin: 0;
  padding: 0;
}

.app-container {
  padding: 20px;
  max-width: 800px;
  margin: auto;
  background: #fff;
  border-radius: 8px;
  box-shadow: 0px 4px 12px rgba(0, 0, 0, 0.1);
}

.navbar {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: #222831;
  color: white;
  padding: 15px 20px;
  border-radius: 8px;
}

.nav-btn {
  background-color: #00adb5;
  color: white;
  border: none;
```

```
  padding: 8px 16px;
  font-weight: bold;
  cursor: pointer;
  border-radius: 5px;
  transition: 0.3s ease;
}

.nav-btn:hover {
  background-color: #007f86;
}

.flight-list {
  margin-top: 20px;
}

.flight-list ul {
  list-style-type: none;
  padding: 0;
}

.flight-list li {
  background: #eeeeee;
  margin: 10px 0;
  padding: 10px 15px;
  border-left: 4px solid #00adb5;
  border-radius: 5px;
}

.guest-view, .user-view {
  margin-top: 20px;
  padding: 15px;
  background-color: #e4f9f5;
  border: 1px solid #00adb5;
  border-radius: 6px;
}
```

Output:

After login:

Ticket Booking App

Logout

Available Flights

- ✈️ Chennai → Delhi at 10:00 AM
- ✈️ Mumbai → Bangalore at 12:00 PM
- ✈️ Kolkata → Hyderabad at 03:30 PM

Welcome, User!

You can now book tickets.

After logout:

Ticket Booking App

Login

Available Flights

- ✈️ Chennai → Delhi at 10:00 AM
- ✈️ Mumbai → Bangalore at 12:00 PM
- ✈️ Kolkata → Hyderabad at 03:30 PM

Welcome, Guest!

Please log in to book your tickets.

13. ReactJS-HOL

Explain various ways of conditional rendering
In React, conditional rendering allows us to render components or elements based on specific conditions. There are several ways to implement this:

1. If-else statements: You can use traditional JavaScript if-else to render components conditionally before the return() statement.
2. Ternary operator: Commonly used inside JSX, it allows rendering one of two elements based on a condition: condition ? <ComponentA /> : <ComponentB />
3. Logical AND (&&) operator: Useful when you want to render something only if the condition is true: condition && <Component />
4. Immediately Invoked Function Expressions (IIFE): Useful for more complex conditions directly inside JSX.
5. Switch-case: Ideal when there are multiple conditions to check. It can be written before the JSX return block.

Explain how to render multiple components
Rendering multiple components in React means displaying more than one component together within a single parent. This can be done in the following ways:

- Wrap the components in a single enclosing element like a div, section, or React.Fragment (<>...</>).
- Example:

```
function App() {
  return (
    <>
      <Header />
      <MainContent />
      <Footer />
    </>
  );
}
```

This helps maintain a clean structure and keeps the component tree readable.

Define list component

A list component in React is a component used to render a list of items dynamically using the map() function. The component iterates over an array and displays UI elements like <li> or custom child components.

Example:

```
function ListComponent(props) {
  return (
    <ul>
      {props.items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  );
}
```

Explain about keys in React applications

Keys are special string attributes used in lists to help React identify which items have changed, are added, or are removed. They provide stability to component identity across renders.

- Keys should be unique among siblings.
- Preferably, use a unique id from the data instead of array index.
- Example:

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

Without keys, React might re-render incorrectly or inefficiently, leading to performance issues or unexpected behavior.

Explain how to extract components with keys

When working with lists, each item can be passed to a separate child component. This is known as extracting components. When doing this, make sure the key prop is assigned to the component being returned from the map() call not inside the child component.

Example:

```
function UserCard({ user }) {
  return <div>{user.name}</div>;
}

function UserList({ users }) {
```

```
  return (
    <div>
      {users.map((user) => (
        <UserCard key={user.id} user={user} />
      ))}
    </div>
  );
}
```

The key helps React manage each UserCard component efficiently during updates.

Explain React Map, map() function

The map() function in React is used to loop through arrays and transform them into React elements. It's a core part of dynamic rendering.

- Syntax:

  ```
  array.map((item, index) => {
    return <Component key={index} data={item} />;
  });
  ```

- It helps in creating UI elements like lists, tables, or cards dynamically.
- Always remember to assign a unique key when using map() in JSX.

Example:

```
const fruits = ['Apple', 'Banana', 'Cherry'];
const fruitList = fruits.map((fruit, index) => <li key={index}>{fruit}</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 [view, setView] = useState('book');

  const renderComponent = () => {
    switch (view) {
      case 'book':
        return <BookDetails />;
      case 'blog':
        return <BlogDetails />;
      case 'course':
        return <CourseDetails />;
      default:
        return <p>Please select a valid view</p>;
    }
  };

  return (
    <div className="container">
      <h1>BloggerApp </h1>
      <div className="button-group">
        <button onClick={() => setView('book')}>Book Details</button>
        <button onClick={() => setView('blog')}>Blog Details</button>
        <button onClick={() => setView('course')}>Course Details</button>
      </div>
      <div className="card">{renderComponent()}</div>
    </div>
  );
}

export default App;
```

BookDetails.js:

```
import React from 'react';

function BookDetails() {
  const books = [
    { id: 1, title: 'Dark Verse: The Reaper', author: 'RuNyx' },
    { id: 2, title: 'Haunting Adeline', author: 'H.D. Carlton' },
```

```
    { id: 3, title: 'Twisted Lies', author: 'Ana Huang' },
  ];

  return (
    <div>
      <h2>Book Details</h2>
      <ul>
        {books.map(book => (
          <li key={book.id}>
            <strong>{book.title}</strong> - {book.author}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default BookDetails;
```

CourseDetails.js:

```
import React from 'react';

function CourseDetails() {
  const courses = [
    { id: 'C1', name: 'React.js', duration: '4 Weeks' },
    { id: 'C2', name: 'Node.js', duration: '3 Weeks' },
    { id: 'C3', name: 'MongoDB', duration: '2 Weeks' },
  ];

  return (
    <div>
      <h2>Course Details</h2>
      <ul>
        {courses.map(course => (
          <li key={course.id}>
            <strong>{course.name}</strong> - {course.duration}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default CourseDetails;
```

App.css:

```css
body {
  margin: 0;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background: linear-gradient(to right, #f7f8fc, #e0eafc);
  color: #333;
}

.container {
  padding: 40px;
  text-align: center;
}

h1 {
  font-size: 2.5rem;
  color: #444;
  margin-bottom: 30px;
}

.button-group {
  margin-bottom: 20px;
}

button {
  background-color: #5a67d8;
  color: white;
  border: none;
  padding: 10px 20px;
  margin: 5px;
  border-radius: 8px;
  cursor: pointer;
  font-size: 16px;
  transition: background-color 0.3s;
}

button:hover {
  background-color: #434190;
}

.card {
  max-width: 600px;
  margin: auto;
  background: white;
  border-radius: 10px;
  padding: 20px;
  box-shadow: 0px 8px 20px rgba(0, 0, 0, 0.1);
  text-align: left;
}
```

```css
ul {
  list-style-type: none;
  padding: 0;
}

li {
  padding: 10px 0;
  border-bottom: 1px solid #eee;
}
```

Output:

Book details:

BloggerApp

Book Details | Blog Details | Course Details

Book Details

- **Dark Verse: The Reaper** - RuNyx
- **Haunting Adeline** - H.D. Carlton
- **Twisted Lies** - Ana Huang

Course details:

BloggerApp

Book Details | Blog Details | Course Details

Course Details

- **React.js** - 4 Weeks
- **Node.js** - 3 Weeks
- **MongoDB** - 2 Weeks