**SUPERSET ID - 6364309**

**Name- Aryabrat Mishra**

**REACTJS HOL - 9 (LAB1)**

**THEORIES FOR LAB:**

Here’s a comprehensive explanation for each of your points related to ES6 (ECMAScript 2015) features:

**Features of ES6 (ECMAScript 2015)**

1. let and const keywords for block-scoped variables.
2. Arrow functions for concise syntax.
3. Classes and class inheritance (OOP support).
4. Template literals for string interpolation.
5. Default parameters in functions.
6. Destructuring assignment for arrays and objects.
7. Spread (...) and rest (...) operators.
8. Modules (import/export syntax).
9. Promises for asynchronous programming.
10. Map and Set objects.
11. for...of loop.
12. Symbol type.
13. Enhanced object literals (shorthand syntax, computed property names).
14. Iterators and generators.
15. New built-in methods (e.g., Array.from(), Object.assign()).

**JavaScript let**

1. Introduced in ES6 to declare block-scoped variables.
2. Unlike var, let respects the boundaries of { }.
3. Not hoisted in the same way as var (temporal dead zone applies).
4. Can be reassigned but not redeclared in the same scope.

Example:

let x = 10;

if (true) {

let x = 20;

console.log(x); // 20

}

console.log(x); // 10

**✅ Difference between var and let**

| **Feature** | **var** | **let** |
| --- | --- | --- |
| **Scope** | **Function-scoped** | **Block-scoped ({})** |
| **Redeclaration** | **Allowed** | **Not allowed in same scope** |
| **Hoisting** | **Hoisted (undefined)** | **Hoisted (TDZ error if used before declaration)** |
| **Use case** | **Legacy code** | **Preferred in modern JS** |

**JavaScript const**

1. Used to declare block-scoped constants.
2. Cannot be reassigned, but the content of objects/arrays declared with const can be modified.
3. Must be initialized at the time of declaration.

Example:

const PI = 3.14;

// PI = 3.14159; // Error

const arr = [1, 2];

arr.push(3); // Works

**ES6 Class Fundamentals**

A cleaner and more intuitive syntax for creating objects and inheritance.

Equivalent to constructor functions under the hood.

Example:

class Person {

constructor(name) {

this.name = name;

}

greet() {

console.log(`Hello, I'm ${this.name}`);

}

}

const p = new Person("Alice");

p.greet(); // Hello, I'm Alice

**ES6 Class Inheritance**

Classes can extend other classes using extends keyword.

super() is used to call the parent constructor.

Example:

class Animal {

constructor(name) {

this.name = name;

}

speak() {

console.log(`${this.name} makes a noise`);

}

}

class Dog extends Animal {

speak() {

console.log(`${this.name} barks`);

}

}

const d = new Dog("Rex");

d.speak(); // Rex barks

**ES6 Arrow Functions**

Concise function syntax: () => {}.

Do not bind their own this, making them useful in callbacks.

Cannot be used as constructors.

Example:

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

console.log(add(2, 3)); // 5

With no parameters:

const greet = () => console.log("Hello");

Set and Map in ES6

Set:

Collection of unique values.

Can store any type of value (primitives or object references).

Example:

const mySet = new Set();

mySet.add(1);

mySet.add(2);

mySet.add(1); // ignored

console.log(mySet); // Set { 1, 2 }

🔹 Map:

Collection of key-value pairs.

Keys can be of any type (unlike plain objects where keys are strings or symbols).

Example:

const myMap = new Map();

myMap.set('a', 1);

myMap.set(1, 'one');

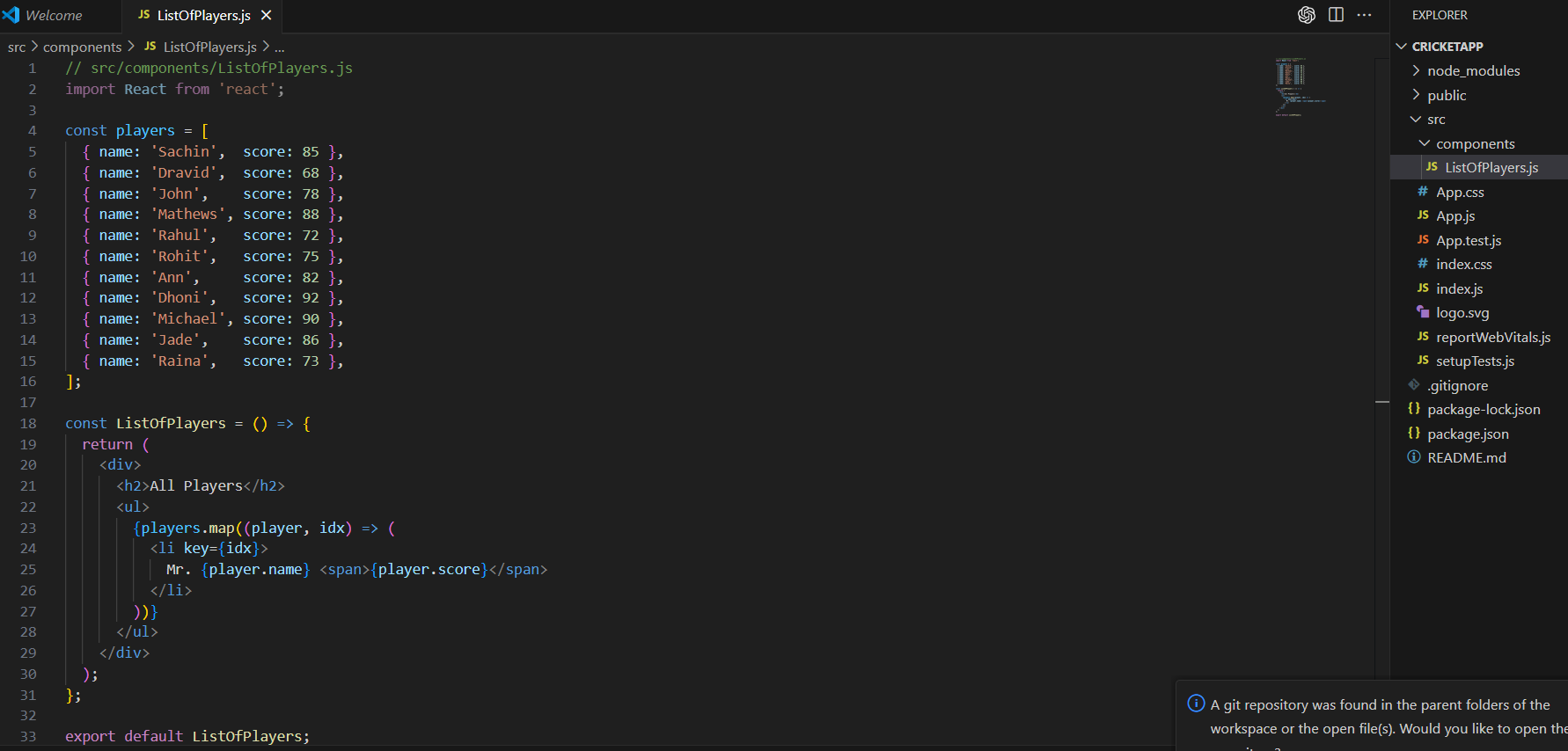
console.log(myMap.get(1)); // "one"

MY WORK LAB1

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

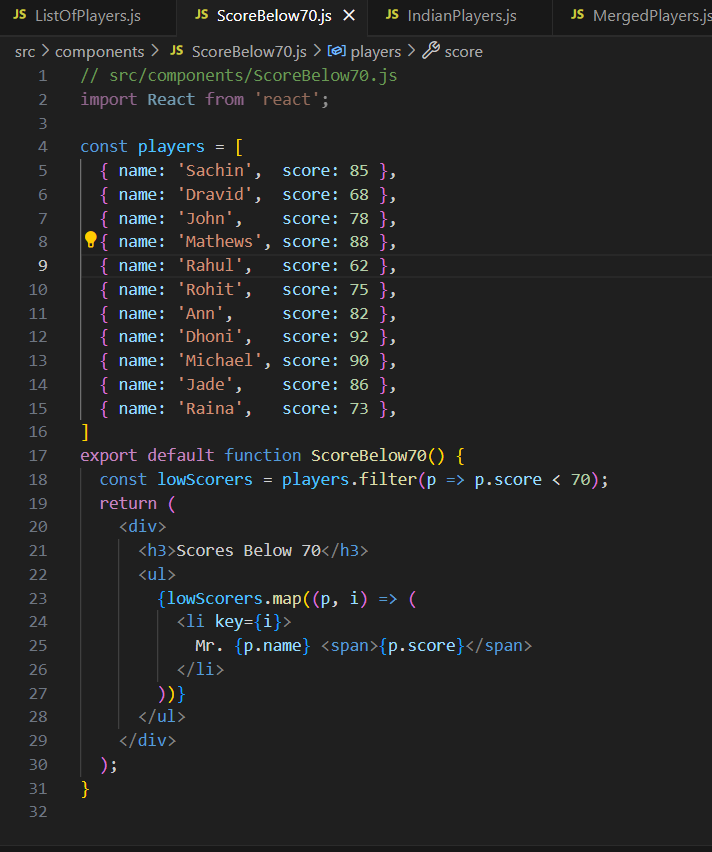
1. ListofPlayers

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



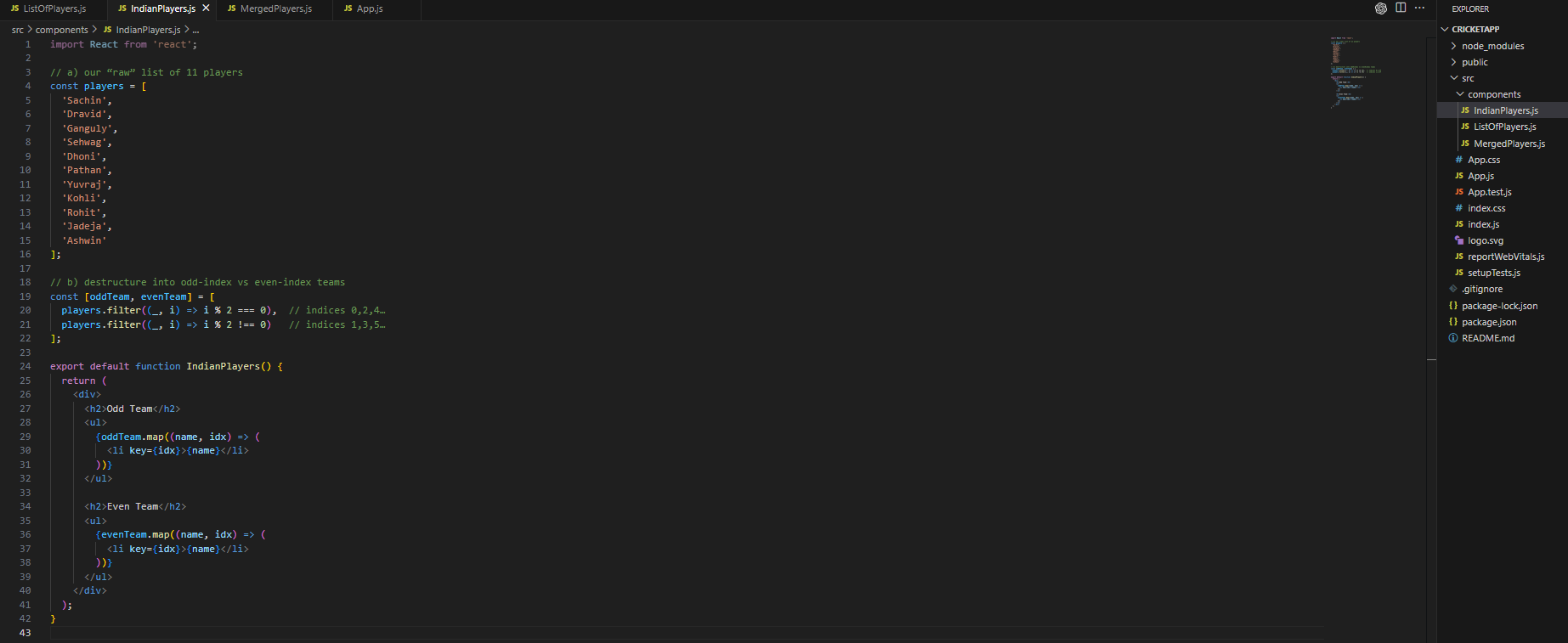
* Filter the players with scores below 70 using arrow functions of ES6.





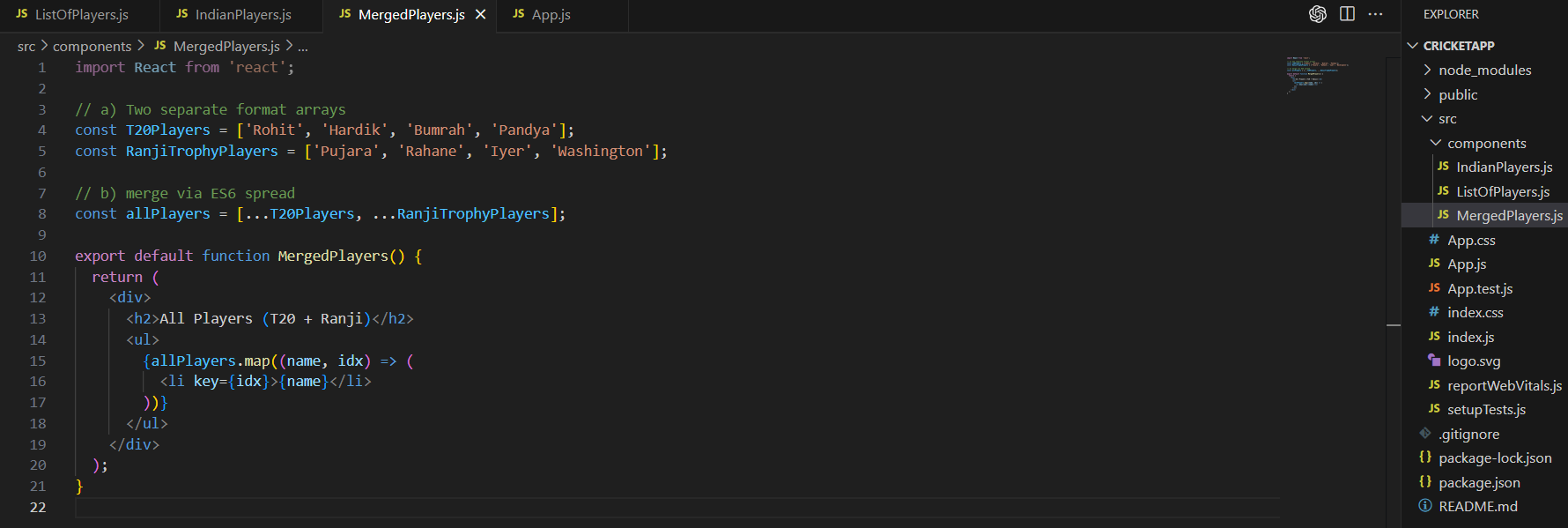
1. **IndianPlayers**
   1. Display the Odd Team Player and Even Team players using the Destructuring features of ES6 as IndianPlayers.js in Components





* 1. Declare two arrays T20players and RanjiTrophy players and merge the two arrays and display them using the Merge feature of ES6 as MergedPlayers.js

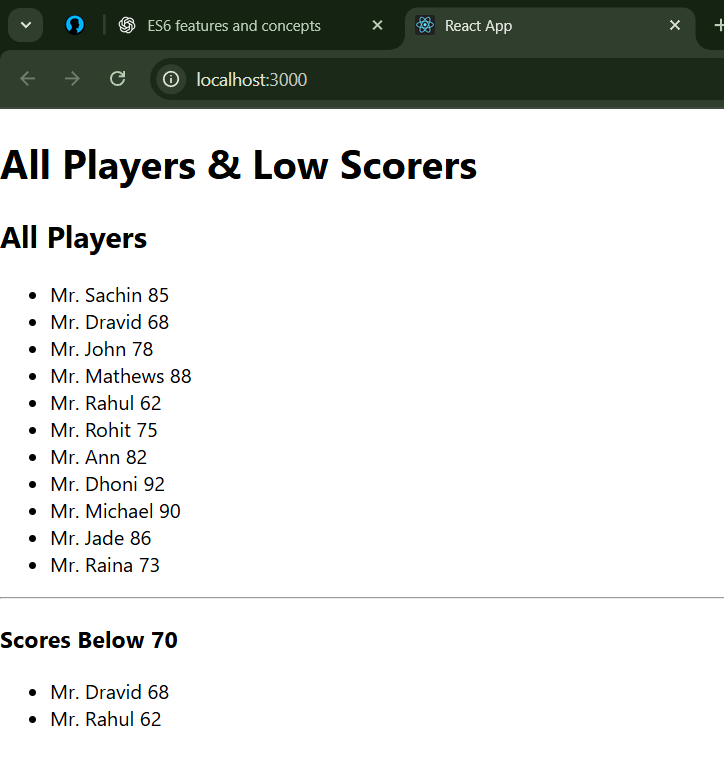




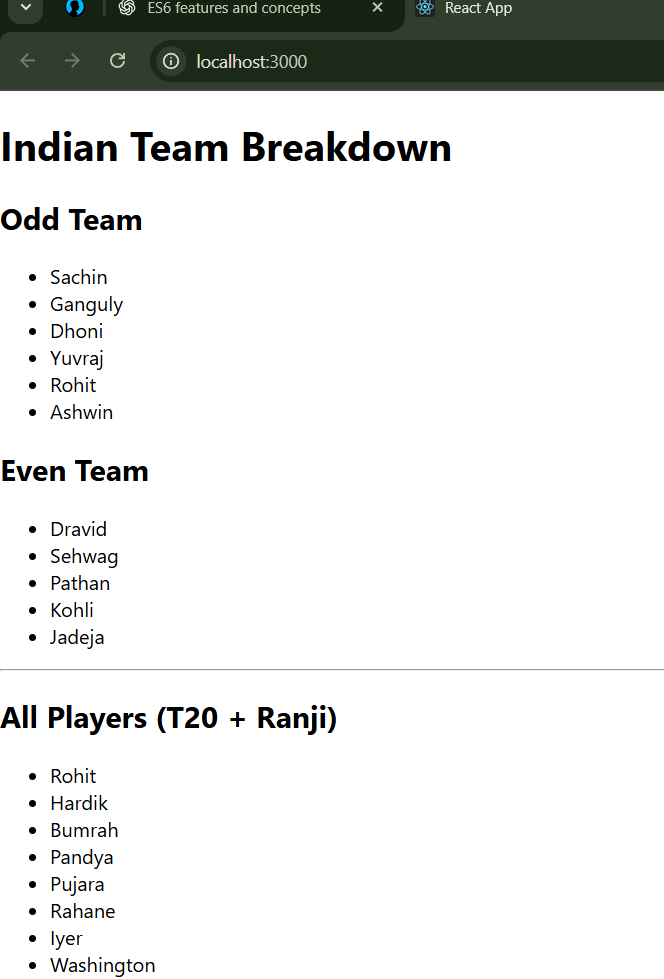
Display these two components in the same home page using a simple if else in the flag variable.

**Output:**

When Flag=true



**When Flag = false**



**REACTJS\_HOL\_10(LAB 2)**

**THEORIES FOR LAB:**

Here’s a complete explanation of your React + ECMAScript + JSX-related questions, in a clear and concise format:

### ✅ ****Define JSX (JavaScript XML)****

**JSX** stands for **JavaScript XML**.

It is a **syntax extension** for JavaScript used with **React**.

JSX allows writing **HTML-like code** inside JavaScript.

Makes it easier to create React elements visually close to actual HTML.

**Example:**

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

Transpiles to:

React.createElement('h1', null, 'Hello, world!');

### ✅ ****Explain ECMA Script****

**ECMAScript (ES)** is the **standardized scripting language specification** upon which **JavaScript** is based.

Developed by **ECMA International**.

ES versions:

**ES5 (2009)**: Standard JavaScript we used pre-2015.

**ES6 (2015)** (also called **ES2015**): Introduced let, const, arrow functions, classes, promises, etc.

New versions released **yearly** (e.g., ES2017, ES2020, ES2023).

React and modern JS development heavily rely on **ES6+ features**.

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

It's the **core function** used by React to **create virtual DOM elements**.

JSX is syntactic sugar for React.createElement.

**Syntax:**

React.createElement(

type, // HTML tag or Component

props, // attributes (null if none)

...children // child elements or text

)

**Example:**

const element = React.createElement('h1', { className: 'greet' }, 'Hello!');

**Equivalent JSX:**

const element = <h1 className="greet">Hello!</h1>;

### ✅ ****How to Create React Nodes with JSX****

A **React Node** is created using JSX by writing HTML-like syntax in JavaScript.

You can use tags like <div>, <h1>, <p>, or even your own component

**Example:**

const node = <p>This is a React node created using JSX.</p>;

**Component example:**

function Greeting() {

return <h2>Welcome to React</h2>;

}

### ✅ ****How to Render JSX to the DOM****

Use the ReactDOM.render() method (for older React versions) or createRoot().render() (React 18+):

**React 18+ Example:**

import React from 'react';

import { createRoot } from 'react-dom/client';

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

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

root.render(element);

**Note**: For React <18, use ReactDOM.render(element, container).

### ✅ ****How to Use JavaScript Expressions in JSX****

You can embed any **JavaScript expression** inside {} within JSX.

**Examples:**

const name = "Arya";

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

const total = <p>Total is {5 + 10}</p>;

You **cannot** use full statements (like if, for) inside {}—only expressions.

✅ **How to Use Inline CSS in JSX**

1. Inline styles are written as a **JavaScript object**.
2. Property names must be in **camelCase** (e.g., backgroundColor).

**Example:**

const divStyle = {

color: 'white',

backgroundColor: 'blue',

padding: '10px'

};

const element = <div style={divStyle}>Styled text</div>;

**Direct inline:**

<div style={{ fontSize: '20px', color: 'red' }}>Inline styled</div>

**MY WORK LAB2**

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

Create an element to display the heading of the page.

Attribute to display the image of the office space

Create an object of office to display the details like Name, Rent and Address.

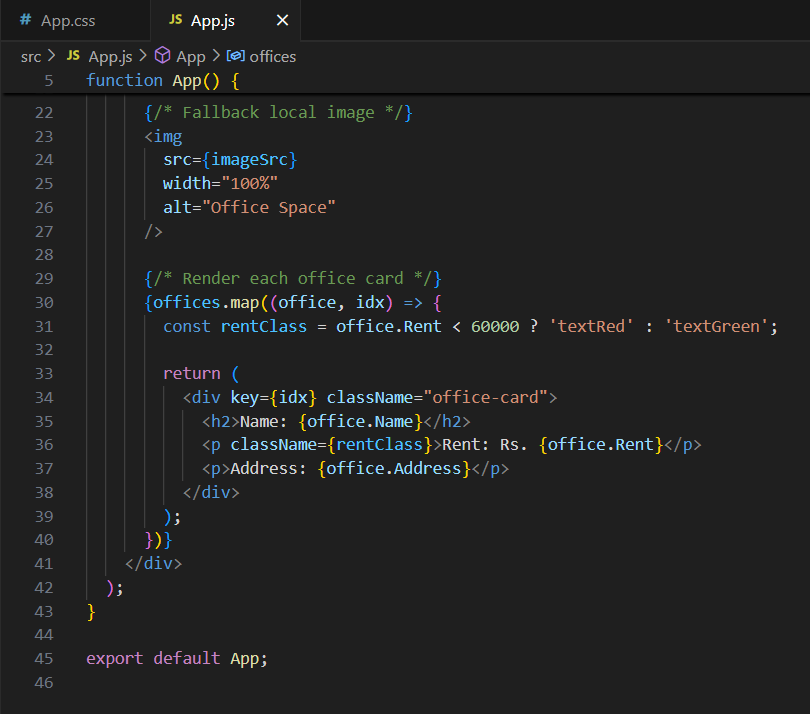
Create a list of Object and loop through the office space item to display more data.

To apply Css, Display the color of the Rent in Red if it’s below 60000 and in Green if it’s above 60000.

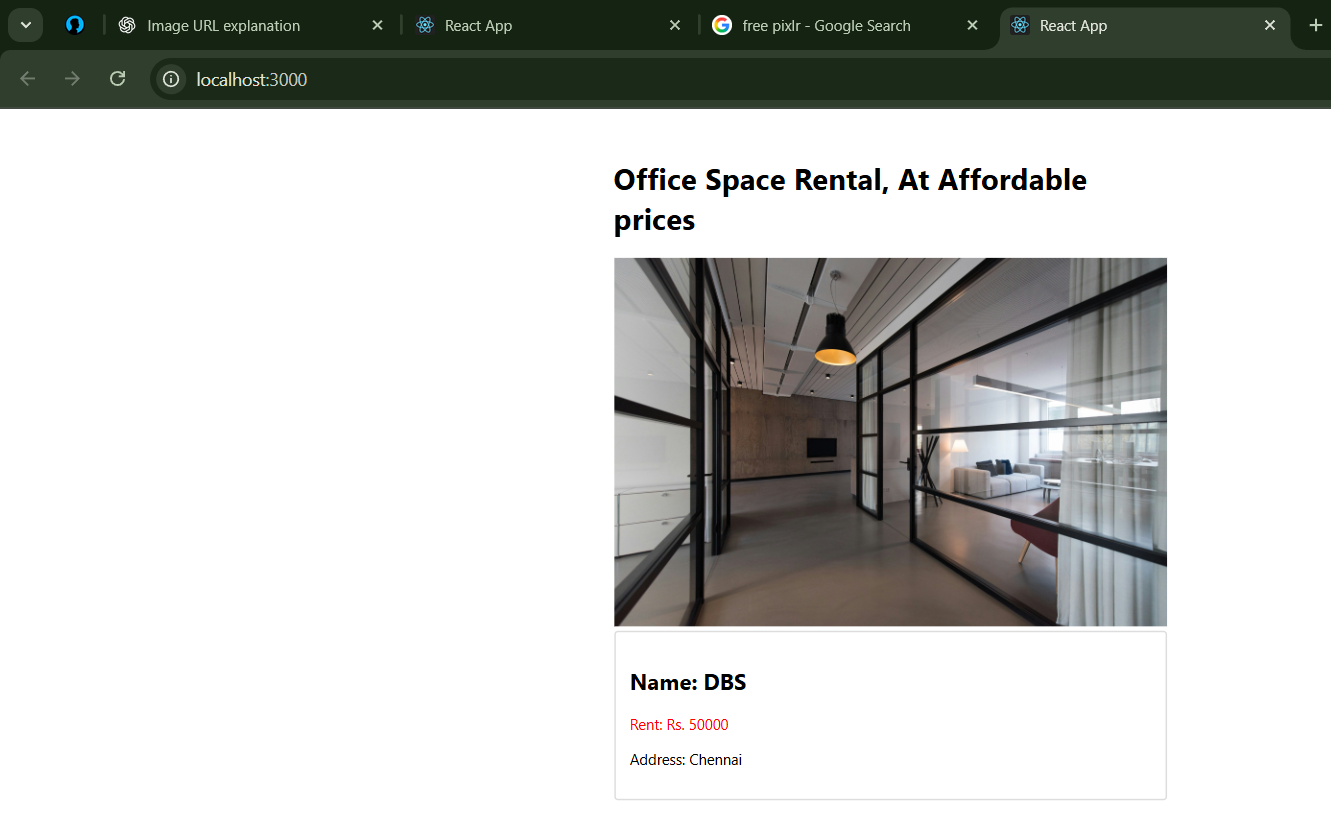
**Src\App.css**



**Src\App.js**



**Output:**



**REACTJS-HOL11(LAB 3)**

**THEORIES FOR LAB:**

Here’s a complete and beginner-friendly breakdown of your React events-related questions:

### ✅ ****Explain React Events****

React **events** are similar to DOM events (like onclick, onchange) but are handled using **React’s event system**.

Events in React are **wrapped in a cross-browser wrapper** called a **SyntheticEvent**.

React supports all the standard DOM events like:

onClick, onChange, onSubmit, onMouseOver, onKeyDown, etc.

**Example:**

function handleClick() {

alert("Button clicked!");

}

function App() {

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

}

### ✅ ****Explain Event Handlers in React****

An **event handler** is a function that is called when an event occurs.

In React, you typically define a function and pass it to an element as a **prop** using **camelCase syntax**.

**Example:**

function App() {

const handleChange = (event) => {

console.log("Input changed to:", event.target.value);

};

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

}

### ✅ ****Define Synthetic Event****

A **SyntheticEvent** is a **cross-browser wrapper** around the native browser event.

It ensures consistent behavior across different browsers.

It contains all the standard properties and methods of native events.

React automatically **pools** SyntheticEvents for performance, meaning the event object might be reused (unless you call event.persist()).

**Example:**

function handleClick(e) {

console.log(e); // SyntheticEvent object

}

### ✅ ****React Event Naming Convention****

| **Feature** | **React Event Convention** | **DOM Convention** |
| --- | --- | --- |
| Naming | **CamelCase** | Lowercase |
| Assignment | Pass a **function**, not string | Pass a string in HTML |
| Binding | Done via arrow functions or binding | N/A in HTML |

**Examples:**

| **DOM (HTML)** | **React (JSX)** |
| --- | --- |
| **<button onclick="doSomething()">** | **<button onClick={doSomething}>** |
| **<form onsubmit="submitForm()">** | **<form onSubmit={submitForm}>** |

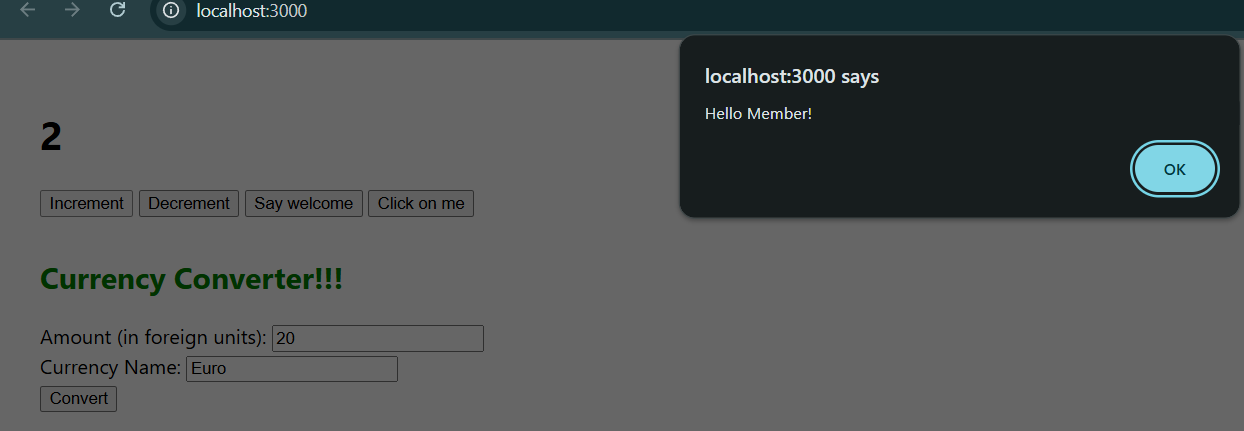
### ✅ Quick Recap:

| **Concept** | **Description** |
| --- | --- |
| ****React Events**** | **React’s way to handle user interactions, similar to DOM events.** |
| ****Event Handler**** | **A function that gets triggered when a specific event occurs.** |
| ****Synthetic Event**** | **React’s wrapper for native events, for consistent cross-browser behavior.** |
| ****Naming**** | **Use **camelCase** like onClick, onChange, and pass functions directly.** |

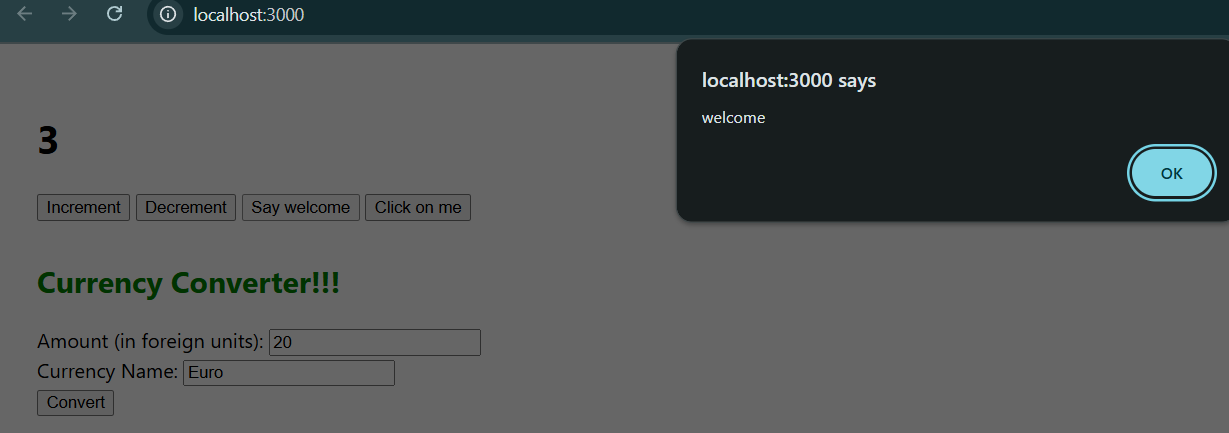
**MY WORK LAB3:**

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

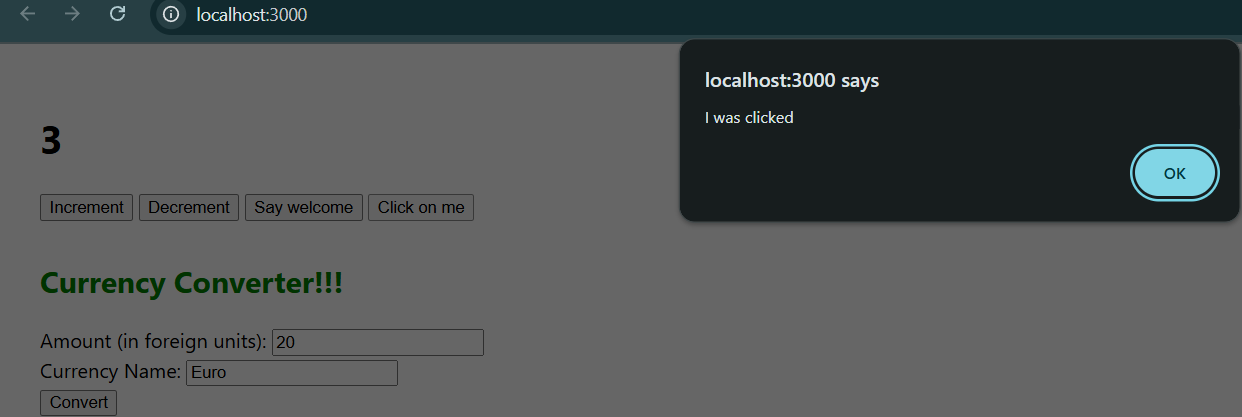
1. Create “Increment” button to increase the value of the counter and “Decrement” button to decrease the value of the counter. The “Increase” button should invoke multiple methods.
   1. To increment the value
   2. Say Hello followed by a static message.



1. Create a button “Say Welcome” which invokes the function which takes “welcome” as an argument.



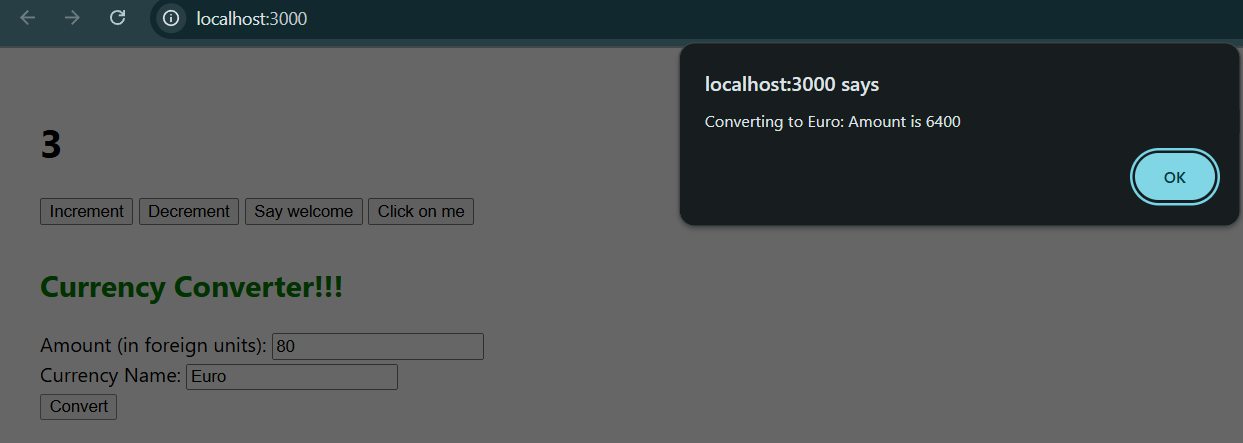
1. Create a button which invokes synthetic event “OnPress” which display “I was clicked”



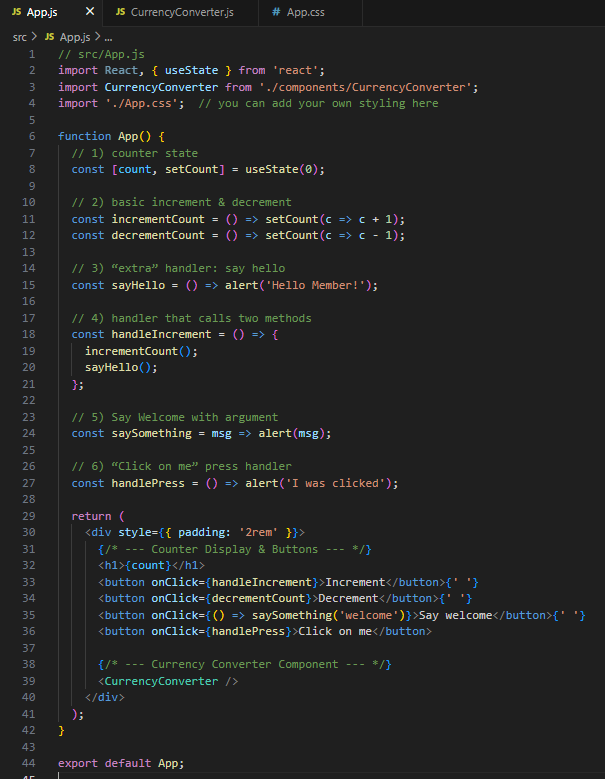
Create a “CurrencyConvertor” component which will convert the Indian Rupees to Euro when the Convert button is clicked.

Handle the Click event of the button to invoke the handleSubmit event and handle the conversion of the euro to rupees.

**Output:**



**Src\App.js**



**Src\CurrencyConverter.js**



**REACTJS - HOL12 (LAB4)**

**THEORIES FOR LAB:**

Here's a clear explanation of the three React concepts you've asked for:

### ✅ ****Conditional Rendering in React****

**Conditional rendering** means showing or hiding components/elements **based on a condition**, like if, ? :, or logical &&.

#### Common Methods:

**Using if/else:**

function Greeting(props) {

if (props.isLoggedIn) {

return <h1>Welcome back!</h1>;

}

return <h1>Please sign in.</h1>;

}

**Using ternary operator:**

<p>{isAdmin ? "Admin Panel" : "User Dashboard"}</p>

**Using logical AND (**&&**):**

{notifications.length > 0 && <p>You have {notifications.length} new messages</p>}

**Using a function that returns JSX conditionally:**

function renderMessage(user) {

return user ? <p>Hello, {user.name}</p> : <p>Welcome, guest</p>;

}

### ✅ ****Define Element Variables****

**Element variables** are used to **store JSX** in a variable, so it can be conditionally rendered later.

This helps keep the JSX cleaner and easier to manage.

**Example:**

function LoginControl(props) {

let button;

if (props.isLoggedIn) {

button = <button onClick={props.handleLogout}>Logout</button>;

} else {

button = <button onClick={props.handleLogin}>Login</button>;

}

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

}

### ✅ ****How to Prevent Components from Rendering****

You can **prevent rendering** in React by:

#### 1. ****Returning**** null from a component:

This means **render nothing**. The component will still run logic/lifecycle but won’t output anything.

function WarningBanner(props) {

if (!props.showWarning) {

return null;

}

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

}

#### 2. ****Using conditional rendering:****

Only render a component when a condition is met.

{isVisible && <MyComponent />}

#### 3. ****Using**** shouldComponentUpdate ****(in class components)****:

You can stop re-rendering based on certain conditions.

shouldComponentUpdate(nextProps, nextState) {

return nextProps.someValue !== this.props.someValue;

}

#### 4. ****React.memo for function components:****

Prevents re-rendering unless props change.

const MyComponent = React.memo(function MyComponent(props) {

return <div>{props.value}</div>;

});

### **✅ Summary:**

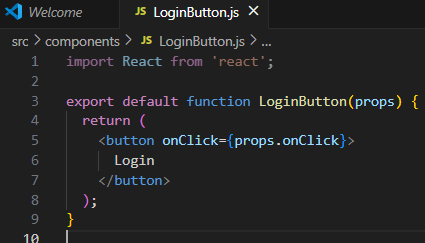
| **Concept** | **Purpose** | **Method** |
| --- | --- | --- |
| **Conditional Rendering** | **Show/hide elements/components** | **if, ? :, &&, functions** |
| **Element Variables** | **Store JSX for reuse or conditional use** | **let button = <JSX />** |
| **Prevent Rendering** | **Skip rendering when not needed** | **Return null, conditionals, memo, shouldComponentUpdate** |

**MY WORK LAB4:**

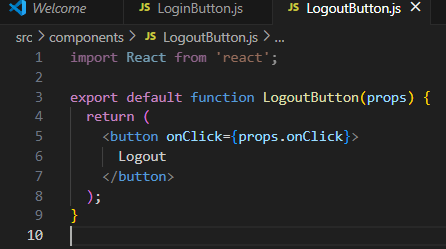
Create a React Application named “ticketbookingapp” where the guest user can browse the page where the flight details are displayed whereas the logged in user only can book tickets.

The Login and Logout buttons should accordingly display different pages. Once the user is logged in the User page should be displayed. When the user clicks on Logout, the Guest page should be displayed.

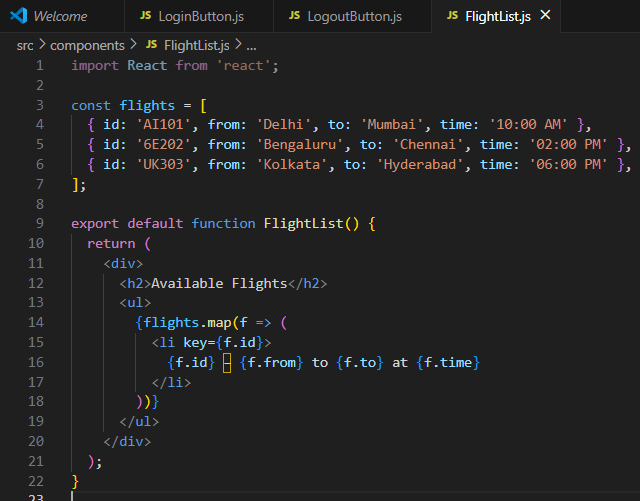
**Src\components\LoginButton.js**



**Src\components\LogoutButton.js**



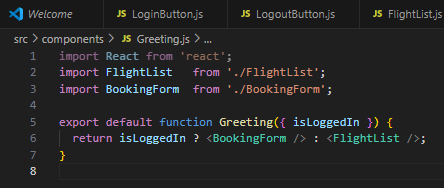
**Src\components\FlightList.js**



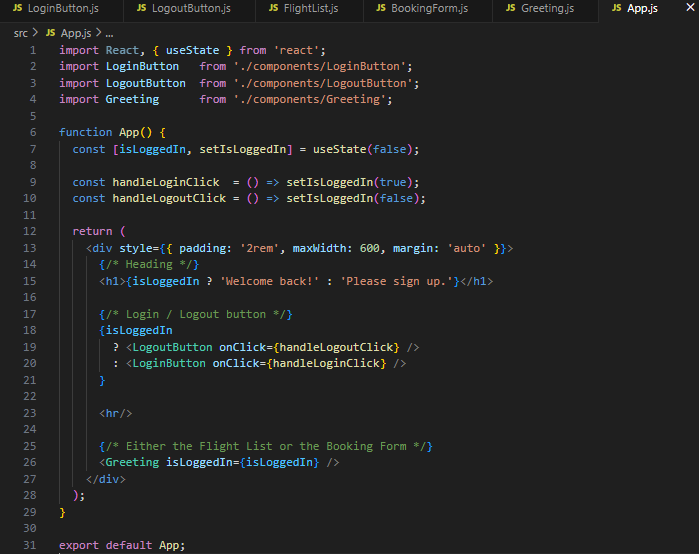
**Src\components\BookingForm.js**



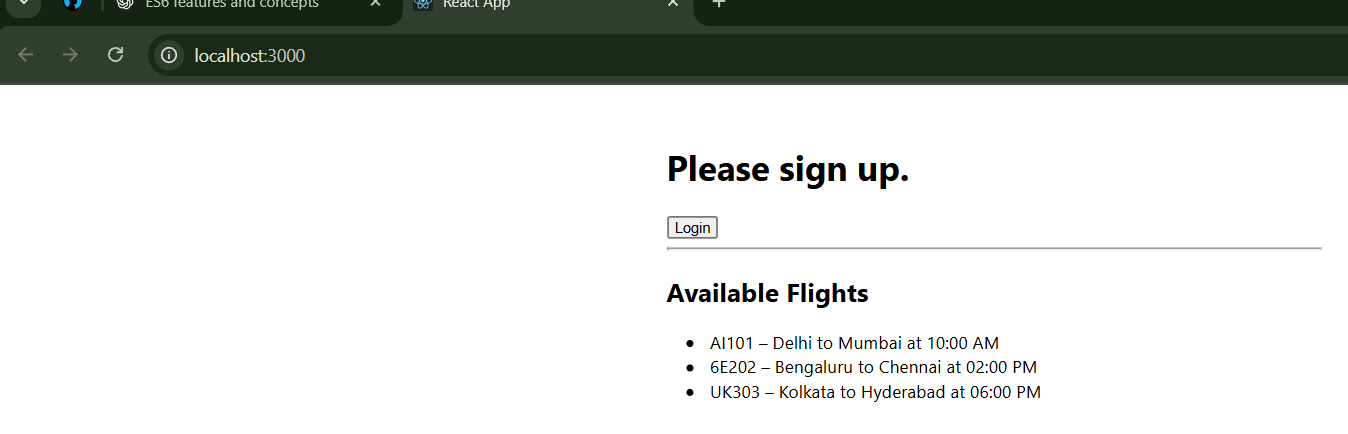
**Src\components\Greeting.js**

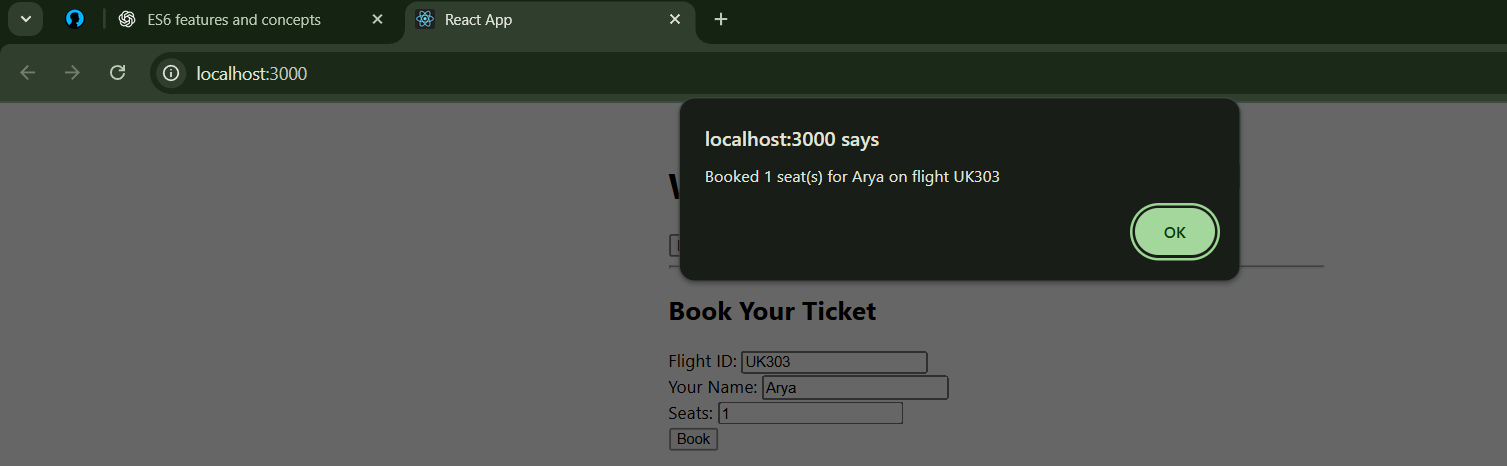


**Src\App.js**



**OUTPUT:**





**REACTJS-HOL-13 (LAB 5)**

**THEORIES FOR LAB :**

Here's a comprehensive guide to your React questions on **conditional rendering**, **lists**, **keys**, and **map()**:

### ✅ ****Various Ways of Conditional Rendering in React****

**Using** if/else **statement**:

if (isLoggedIn) {

return <Dashboard />;

} else {

return <Login />;

}

**Using** element variables:

let button;

if (isLoggedIn) {

button = <LogoutButton />;

} else {

button = <LoginButton />;

}

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

**Using ternary operator**:

return (

<div>

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

</div>

);

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

return (

<div>

{isAdmin && <AdminPanel />}

</div>

);

**Immediately invoked function expressions (IIFE)**:

return (

<div>

{(() => {

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

if (user.role === 'user') return <User />;

return <Guest />;

})()}

</div>

);

### ✅ ****How to Render Multiple Components****

You can render multiple components in a single return using:

**Fragment (**<> </>**)**:

return (

<>

<Header />

<Content />

<Footer />

</>

);

**Array of components**:

return [

<Header key="1" />,

<Content key="2" />,

<Footer key="3" />

];

**Wrapper** <div> **(not recommended always)**:

return (

<div>

<Header />

<Content />

<Footer />

</div>

);

### ✅ ****Define List Component****

A **list component** is a component that displays a collection of items.

It usually takes an array of data and renders each item using map().

**Example:**

function ListComponent({ items }) {

return (

<ul>

{items.map((item, index) => (

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

))}

</ul>

);

}

### ✅ ****Keys in React Applications****

**Keys** help React **identify which items have changed, are added, or removed**.

Must be **unique** among siblings.

Improve rendering **performance** in lists.

Avoid using index as a key unless there’s no unique ID available.

**Good:**

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

**Not ideal (index-based):**

{items.map((item, index) => <li key={index}>{item}</li>)}

### ✅ ****How to Extract Components with Keys****

When mapping over data, you can extract a **child component** and still pass the key to it.

**Example:**

function Item({ value }) {

return <li>{value}</li>;

}

function ItemList({ items }) {

return (

<ul>

{items.map(item => (

<Item key={item.id} value={item.name} />

))}

</ul>

);

}

The key is always passed to the **outermost element** returned in the map — **not** as a prop to the child component.

### ✅ ****React**** map() ****Function****

map() is a JavaScript array method used in React to **transform arrays into lists of elements**.

Used heavily in rendering dynamic lists of components.

**Example:**

const names = ['Arya', 'John', 'Jane'];

const listItems = names.map((name, index) =>

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

);

return <ul>{listItems}</ul>;

### ✅ Summary Table

| **Concept** | **Description** |
| --- | --- |
| **Conditional Rendering** | **Display components conditionally (if, ? :, &&)** |
| **Render Multiple Components** | **Use Fragment, arrays, or divs** |
| **List Component** | **Component rendering a list using map()** |
| **Keys** | **Unique identifier for list elements** |
| **Extract Component with Keys** | **Pass key to JSX tag when using extracted components** |
| **map() Function** | **Transforms arrays into React elements** |

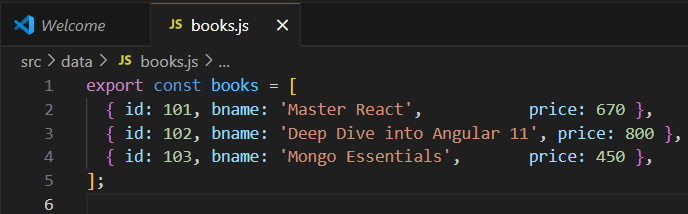
**MY WORK LAB5:**

Create a React App named “bloggerapp” in with 3 components.

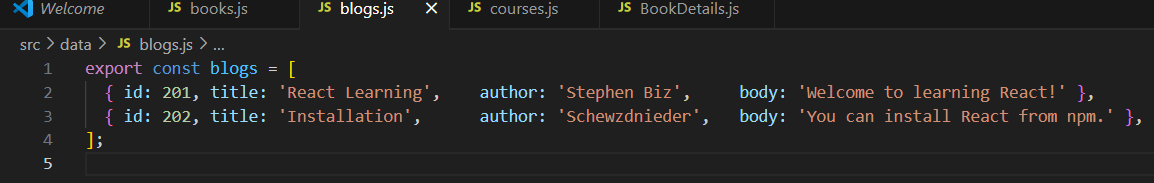
1. Book Details
2. Blog Details
3. Course Details

**A data Folder was created within which books , blogs and courses files would be there**

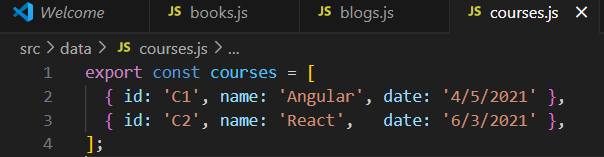
**Src\data\books.js**



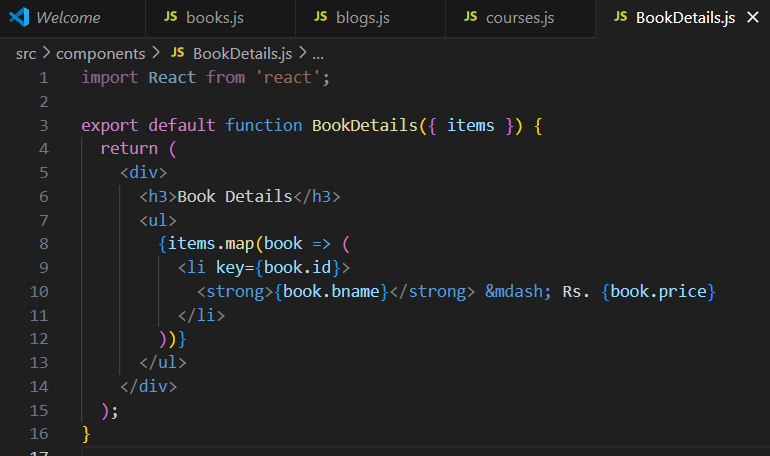
**Src\data\blogs.js**



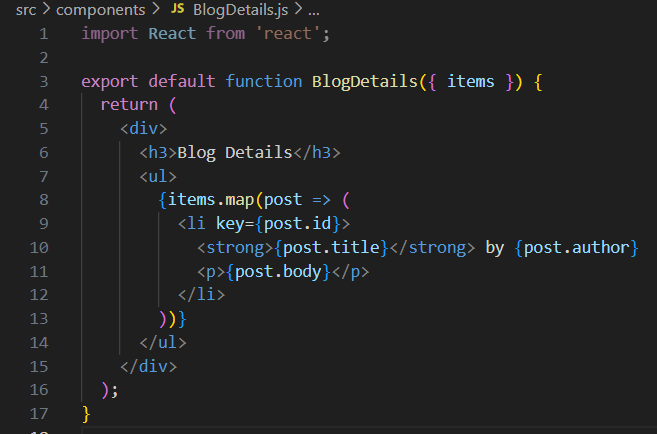
**Src\data\courses.js**



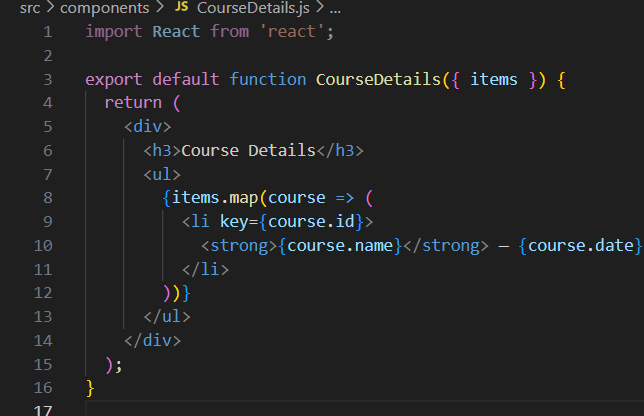
**Src\components\BookDetails.js**



**BlogDetails.js**



**CourseDetails.js**



**App.js**

// src/App.js

import React from 'react';

import './App.css';

import CourseDetails from './components/CourseDetails';

import BookDetails   from './components/BookDetails';

import BlogDetails   from './components/BlogDetails';

import { courses } from './data/courses';

import { books   } from './data/books';

import { blogs   } from './data/blogs';

function App() {

  return (

    <div className="layout-container">

      {/\* Column 1 \*/}

      <div className="layout-column">

        <CourseDetails items={courses} />

      </div>

      {/\* Green separator \*/}

      <div className="layout-separator" />

      {/\* Column 2 \*/}

      <div className="layout-column">

        <BookDetails items={books} />

      </div>

      {/\* Green separator \*/}

      <div className="layout-separator" />

      {/\* Column 3 \*/}

      <div className="layout-column">

        <BlogDetails items={blogs} />

      </div>

    </div>

  );

}

export default App;

**OUTPUT:**

