**Module : ReactJS Hands-On**

**Hands On 9 –cricketapp using map() method and arrow functions**

**Learning Objectives:**

* Understand key features of ES6
* Learn how let and const work
* Compare var vs let
* Understand ES6 classes and inheritance
* Use arrow functions
* Explore map() and set()
* Apply ES6 features in React hands-on

**Theory:**

### **Features of ES6**

* Block-scoped variables using let and const
* Arrow functions (=>)
* Template literals (`Hello ${name}`)
* Destructuring (array/object unpacking)
* Default and rest parameters
* Classes and inheritance
* Modules (import / export)
* Spread and rest operators (...)
* Promises for async handling
* New data structures: Set, Map

Javascript Let

* Declares block-scoped variables.
* Cannot be redeclared in the same block.
* Supports temporal dead zone (not hoisted like var).

Var vs let

| **Feature** | **var** | **let** |
| --- | --- | --- |
| Scope | Function-scoped | Block-scoped |
| Redeclaration | Allowed | Not allowed |
| Hoisting | Yes (initialized to undefined) | Yes (not initialized) |
|  |  |  |

Javascript const

* Declares block-scoped, read-only named constants.
* Value cannot be reassigned.
* Must be initialized at declaration.

ES6 Class Fundamentals

* Introduces class-based OOP syntax in JavaScript.
* Uses constructor() to initialize properties.
* Methods are defined without function keyword.

js

CopyEdit

class Player {

constructor(name, score) {

this.name = name;

this.score = score;

}

display() {

return `${this.name}: ${this.score}`;

}

}

ES6 class Inheritence

* Achieved using extends and super() keywords.
* Allows one class to inherit from another.

js

CopyEdit

class Cricketer extends Player {

constructor(name, score, team) {

super(name, score);

this.team = team;

}

ES6 Arrow functions

* Shorter syntax for function expressions.
* No own this, arguments, or super binding.

js

CopyEdit

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

* Set
  + Stores unique values.
  + Automatically removes duplicates.

js

CopyEdit

const numbers = new Set([1, 2, 2, 3]); // Set(3) {1, 2, 3}

* Map
  + Stores key-value pairs.
  + Maintains insertion order.

js

CopyEdit

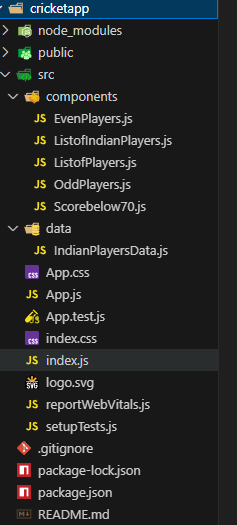
const playerScores = new Map();

playerScores.set('Sachin', 100);

**Hands-On Lab:**

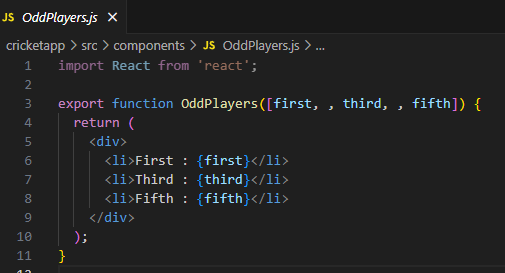
* To create a React Application with the name of “cricketapp”, type the following command:

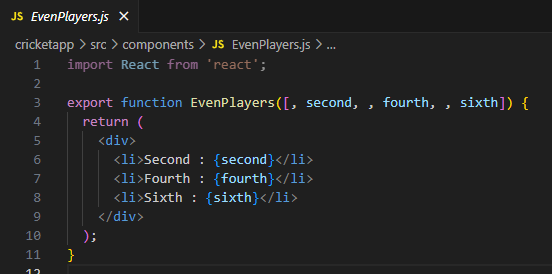
npx create-react-app cricketapp

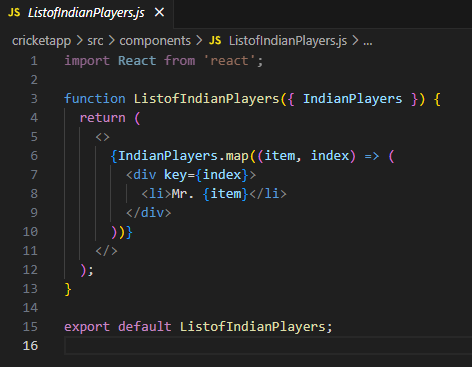


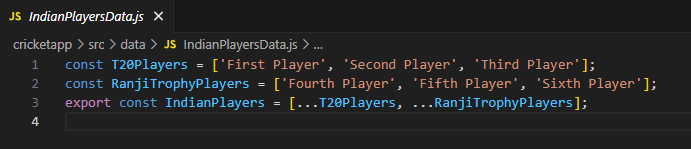




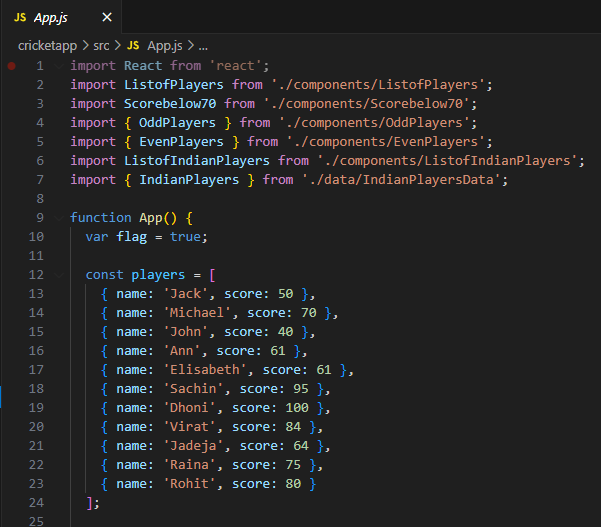


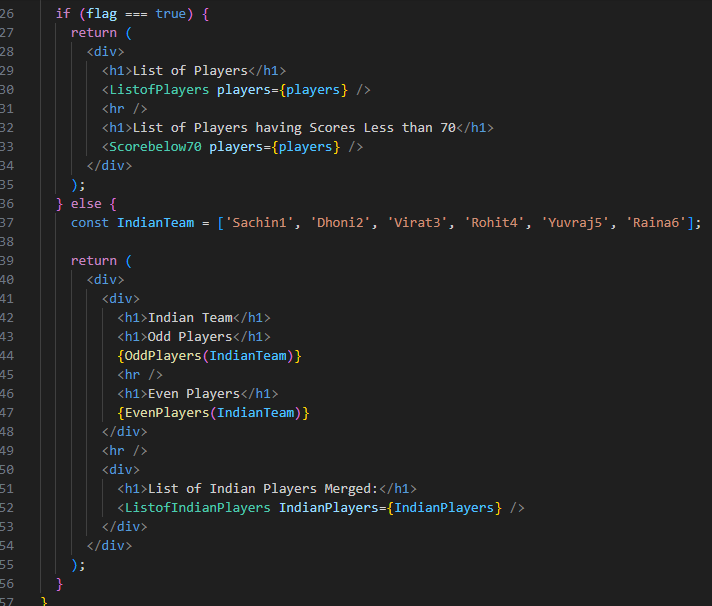




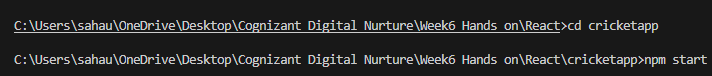


* Content of “App.js”

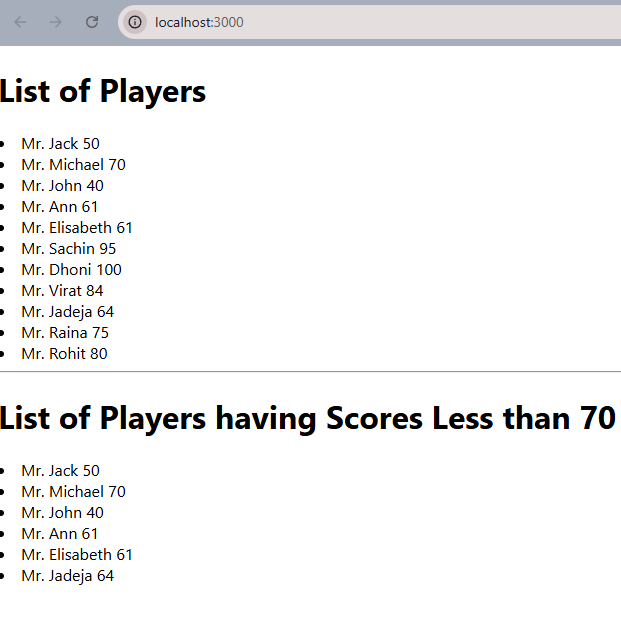


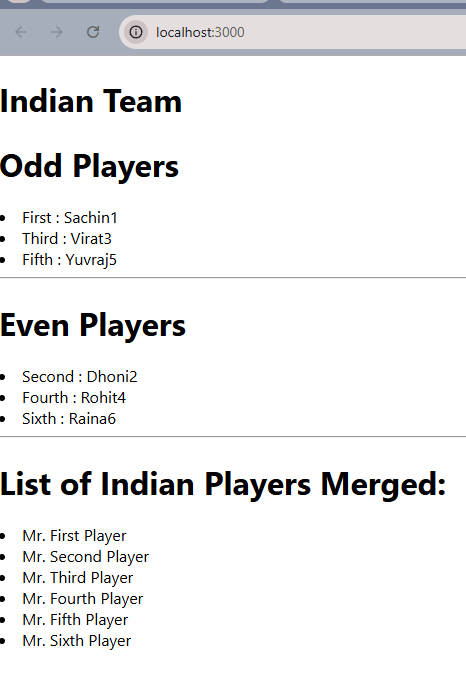


* Run the following command to execute the React application:



* Open a new browser window and type “localhost:3000” in the address bar





**Hands On 10- React App for understanding JSX**

**Learning Objectives:**

* Understand JSX syntax and usage in React
* Learn about ECMAScript and its relevance to React
* Use React.createElement() for element creation
* Create and render React nodes using JSX
* Embed JavaScript expressions in JSX
* Apply inline CSS styles in JSX

**Theory:**

* **JSX**: JSX (JavaScript XML) is a syntax extension that lets you write HTML-like code inside JavaScript, used in React to describe UI structure.
* **ECMAScript**: It’s the scripting language standard (like ES6, ES7) that modern JavaScript (used in React) is based on.
* **React.createElement()**: A core method in React that creates and returns a React element (used behind the scenes when you use JSX).
* **Creating React nodes with JSX**: React nodes can be created using <tagName>content</tagName> syntax directly inside render functions.
* **Rendering JSX to DOM**: Use ReactDOM.render(<JSXElement />, document.getElementById('root')) to mount JSX to the DOM.
* **JavaScript expressions in JSX**: Use {} to embed any valid JS expression inside JSX, e.g., {name}, {count + 1}.
* **Inline CSS in JSX**: Define styles as objects and use the style attribute, e.g., <div style={{ color: 'blue', fontSize: '16px' }}></div>.

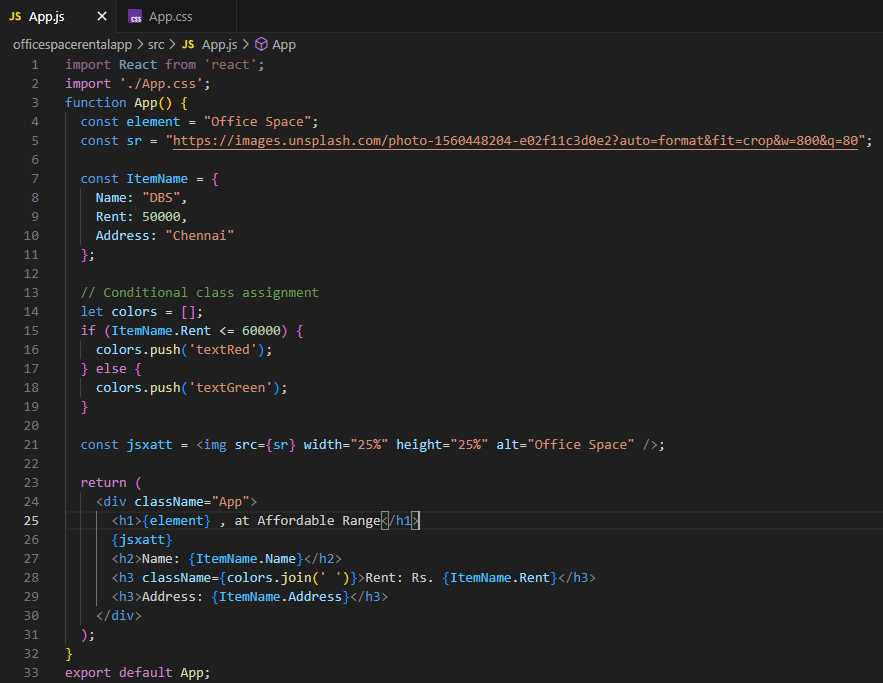
**Hands-On Lab:**

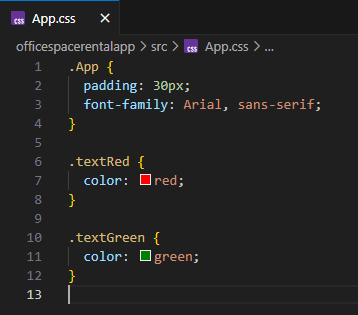
* Create a React project named “officespacetentalapp” type the following command in terminal of Visual studio:

npx create-react-app officespacerentalapp

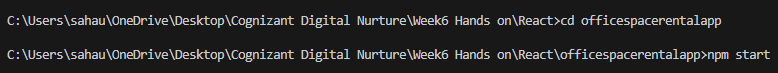


* Edit the App.js and App.css files

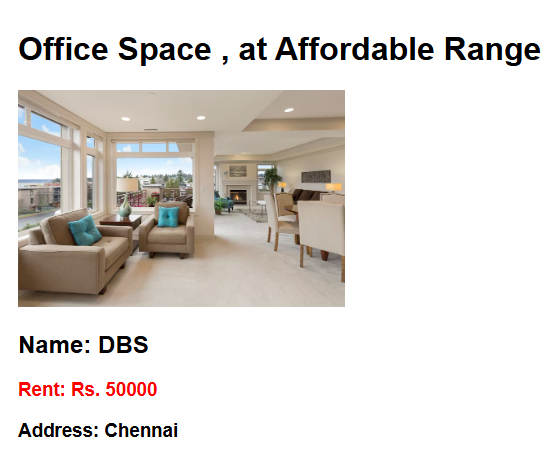


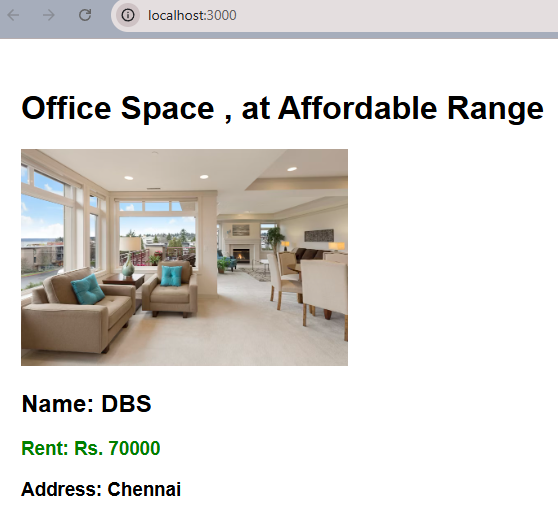


* In command Prompt execute the code by typing the following command:



* Open browser and type “localhost:3000” in the address bar:





**Hands On 11: React App for Event Handlers**

**Learning Objectives:**

* Understand how React handles events.
* Learn to implement event handlers in React.
* Use this keyword properly in class components.
* Work with synthetic events in React.

**Theory:**

**React Events**  
React events are similar to DOM events but follow JSX syntax and React's event system. Examples: onClick, onChange.

**Event Handlers**  
Functions triggered in response to events. In React, handlers like handleClick are passed as props to JSX elements (e.g., <button onClick={handleClick}>).

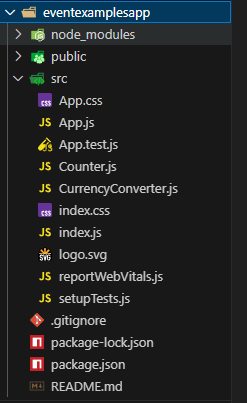
**Synthetic Event**  
A wrapper around the native browser event, provided by React to ensure consistent behavior across different browsers.

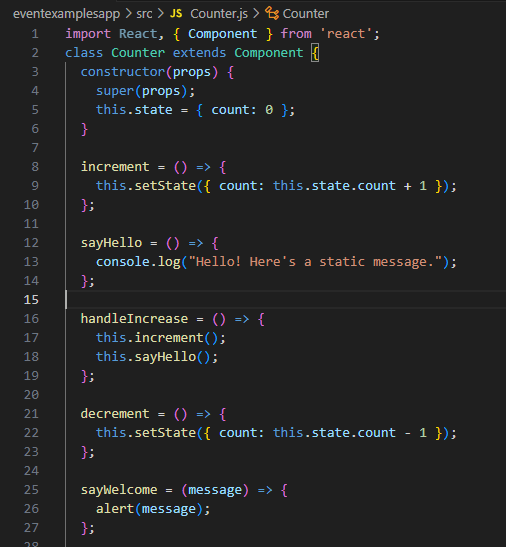
**React Event Naming Convention**  
React uses camelCase naming for events (e.g., onClick, onMouseOver) instead of lowercase (onclick) like in HTML.

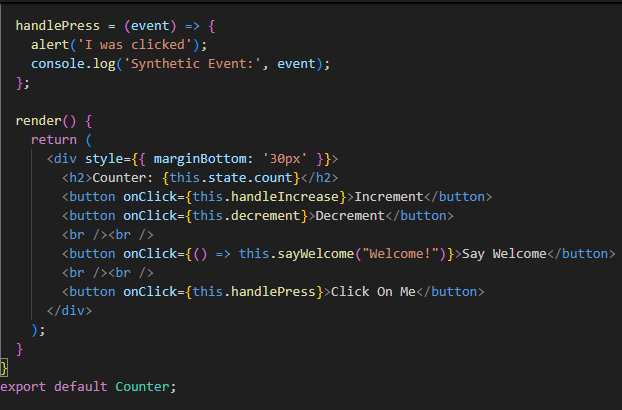
**Hands-On Lab:**

* Create a React project named “eventexamplesapp” type the following command in terminal of Visual studio:

npx create-react-app eventexamplesapp

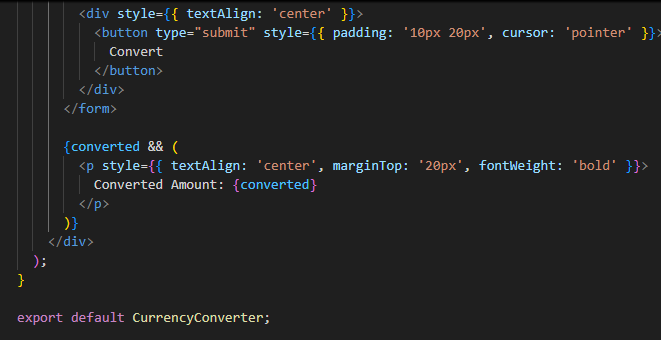


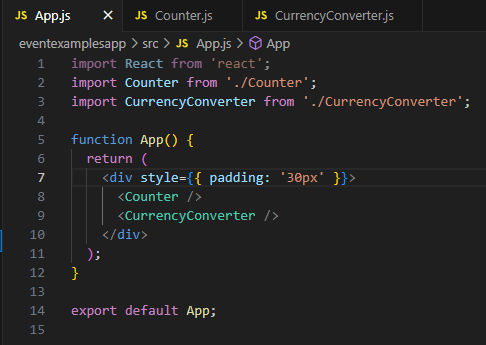
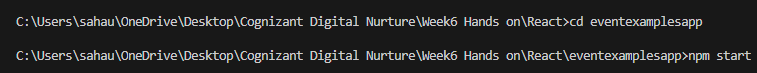


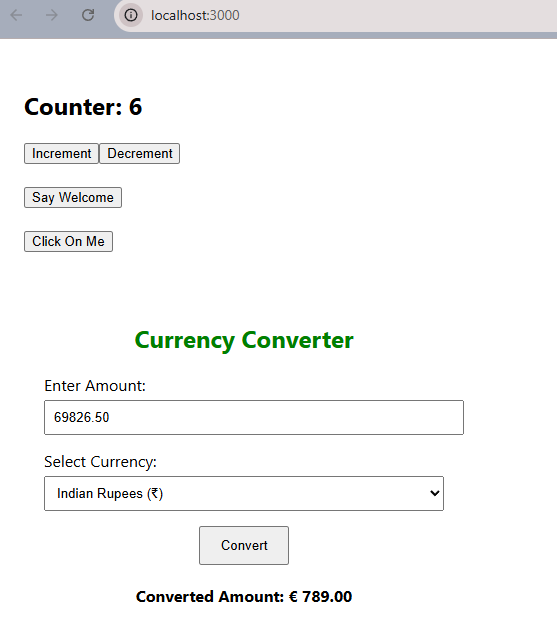


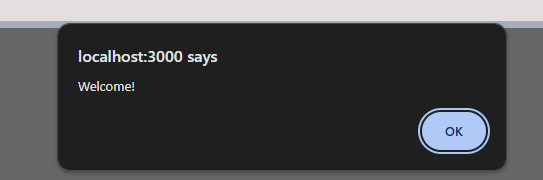


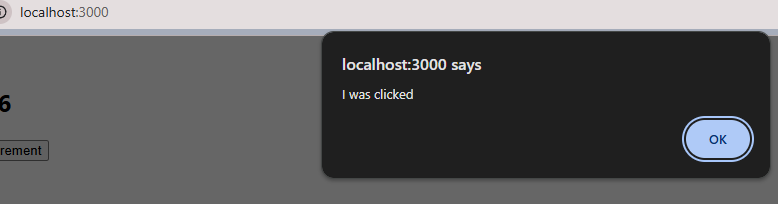


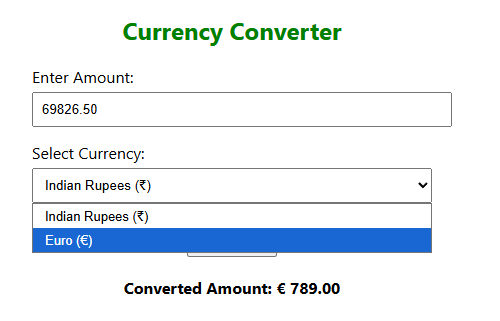


* Edit the App.js
* 
* In command Prompt to execute the code by typing the following command:
* 
* Open browser and type “localhost:3000” in the address bar:









**Hands On 12 :React App for Conditional Rendering**

**Learning Objectives:**

* Implement **conditional rendering** in React applications.
* Define and use **element variables** for flexible rendering logic.
* Understand how to **prevent components from rendering** based on conditions.

**Theory:**

**Conditional Rendering**

React allows components to render different outputs based on conditions using JavaScript logic like if, ? :, or &&.

**Example:**

jsx

CopyEdit

{isLoggedIn ? <LogoutButton /> : <LoginButton />}

**Element Variable**

Element variables can store components to render later based on conditions, helping organize code.

**Example:**

jsx

CopyEdit

let button;

if (isLoggedIn) {

button = <LogoutButton />;

} else {

button = <LoginButton />;

}

**Prevent Component Rendering**

To skip rendering a component, return null inside it.

**Example:**

jsx

CopyEdit

function Banner(props) {

if (!props.show) return null;

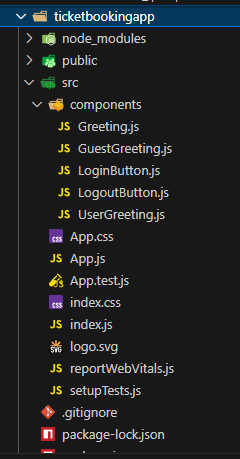
return <div>Alert!</div>;

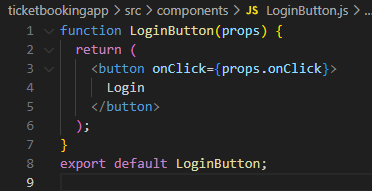
}

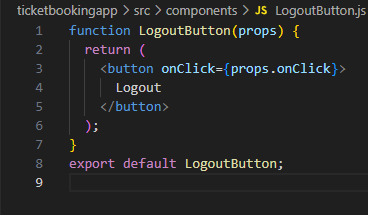
**Hands-On Lab:**

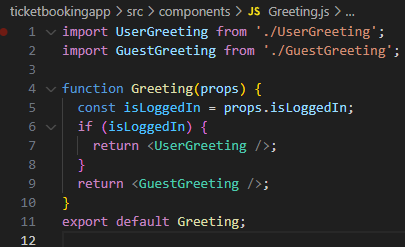
* Create a React project named “ticketbookingapp” type the following command in terminal of Visual studio:

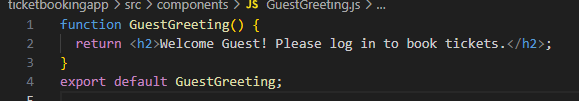
npx create-react-app ticketbookingapp

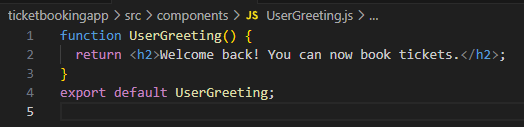




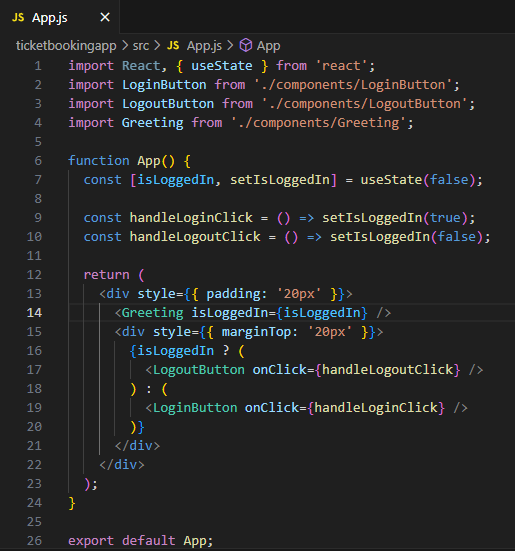




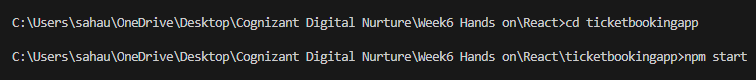




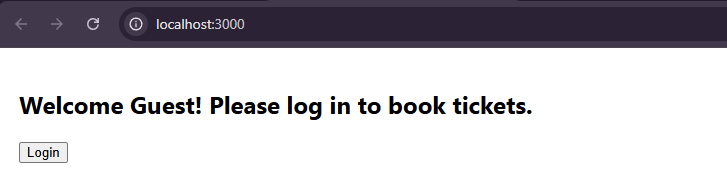
* Edit the App.js

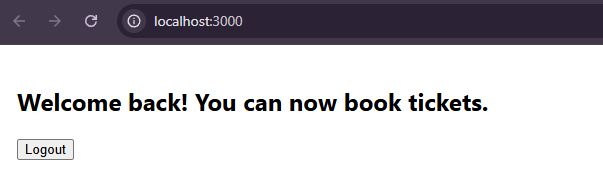


* In command Prompt execute the code by typing the following command:



* Open browser and type “localhost:3000” in the address bar:





**Hands On 13-React App for Implementation of Conditional Rendering**

**Learning Objectives:**

* Implement **conditional rendering** in React applications.
* Render **multiple components** efficiently.
* Define and use a **list component**.
* Understand and apply **keys** in React lists.
* Extract components using **keys** for reusability.
* Use the **map()** function to dynamically render lists of components.

**Theory:**

**Conditional Rendering :** Conditional rendering refers to showing components or UI elements based on specific conditions (like state or props). It can be implemented using:

* **if-else statements**
* **Ternary operator** (condition ? trueComponent : falseComponent)
* **Logical AND (&&) operator**
* **Switch statements**
* **IIFE (Immediately Invoked Function Expressions)** for more complex logic

**Rendering Multiple Components**

You can render multiple components by:

* Wrapping them in a single parent component (like a <div>, <>, or <React.Fragment>)
* Returning an array of components
* Using map() to loop through data and return a component for each item

**List Components**

A list component in React renders a collection of items. It often involves:

* Iterating over an array using map()
* Rendering a component for each item
* Using **keys** to uniquely identify each element

**Example:**

jsx

CopyEdit

const List = ({ items }) => (

<ul>

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

</ul>

);

**Keys in React Application**

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

* Must be unique among siblings
* Improve performance by minimizing re-rendering
* Are commonly derived from item IDs or indexes

**Extracting Components with keys**

When mapping over data, you can extract each item into a **child component** and pass the **key** prop. This keeps code modular and readable.

**Example:**

jsx

CopyEdit

const Item = ({ value }) => <li>{value}</li>;

const List = ({ items }) => (

<ul>

{items.map((item) => <Item key={item.id} value={item.name} />)}

</ul>

);

**React map function**

The map() function is used to iterate over arrays and return JSX elements for each item. It is fundamental to rendering dynamic lists.

**Example:**

jsx

CopyEdit

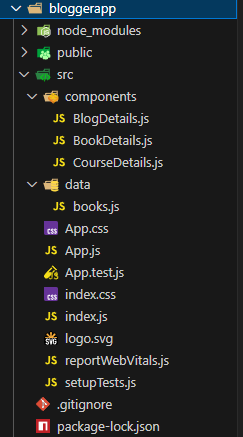
const names = ['Alice', 'Bob', 'Charlie'];

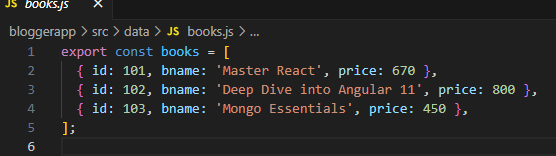
const nameList = names.map(name => <p key={name}>{name}</p>);

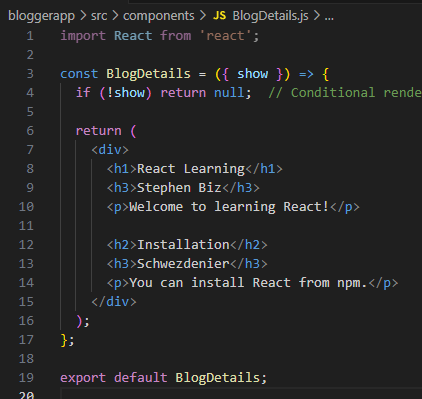
**Hands-On Lab:**

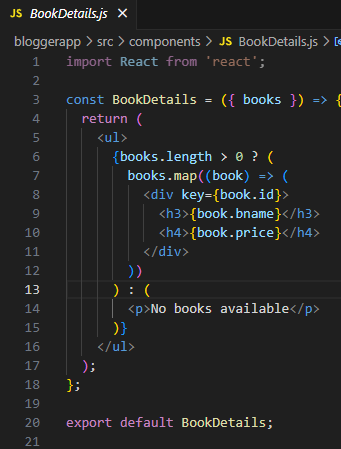
* Create a React project named “bloggerapp” type the following command in terminal of Visual studio:

npx create-react-app bloggerapp



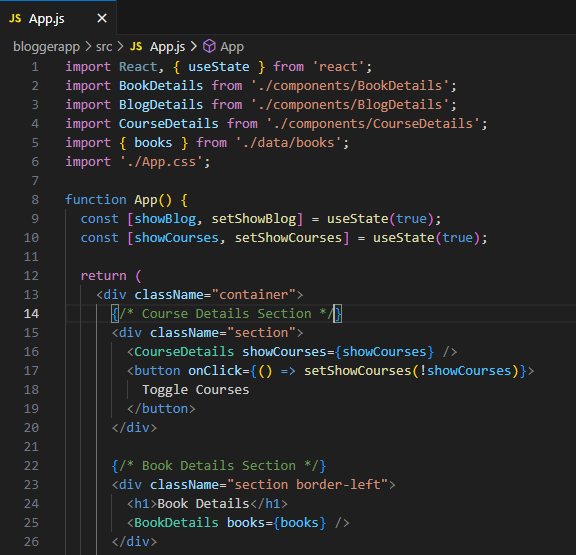


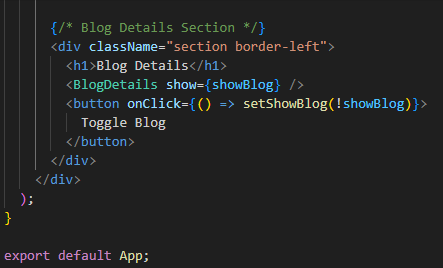


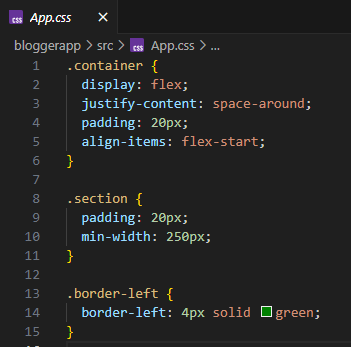




* Edit the App.js and App.js







* In command Prompt execute the code by typing the following command:



* Open browser and type “localhost:3000” in the address bar:

