-: 6363213\_WEEK 7 :-

## **-: ReactJS:-**

## **-: (ReactJs:-9) :-**

## **Objectives :-**

## **Features of ES6.**

• let and const for variable declarations with block scope.  
• Arrow functions for concise function syntax.  
• Template literals for easier string interpolation.  
• Default parameters in functions.  
• Classes for object-oriented programming.  
• Modules for importing and exporting code.  
• Destructuring assignment for arrays and objects.  
• Spread and rest operators for flexible data handling.  
• Promises for handling asynchronous operations.  
• New data structures: Set and Map.  
• Enhanced object literals for cleaner syntax.

## **JavaScript let.**

• 'let' is used to declare variables with block scope.  
• Variables declared with let can be reassigned but not redeclared in the same scope.  
• Unlike var, let does not hoist the variable to the function scope.  
• It prevents accidental overwriting of variables outside the intended block.

## **Differences between var and let.**

• var is function-scoped, while let is block-scoped.  
• var variables can be redeclared in the same scope; let variables cannot.  
• var is hoisted and initialized with undefined; let is hoisted but not initialized, causing a ReferenceError if accessed before declaration.  
• let provides better control over variable usage in modern JavaScript.

## **JavaScript const.**

• const is used to declare variables whose values cannot be reassigned.  
• It is block-scoped like let.  
• For objects and arrays declared with const, their contents can be modified but the reference cannot be reassigned.  
• Ensures immutability for primitive values.

## **ES6 Class Fundamentals.**

• ES6 introduced the class syntax for creating objects and handling inheritance.  
• Classes use a constructor method for initializing properties.  
• Methods can be added directly inside the class body without using the function keyword.  
• Classes provide a cleaner and more structured approach to object-oriented programming in JavaScript.

## **ES6 Class Inheritance.**

• ES6 classes can inherit from other classes using the extends keyword.  
• The super() function is used to call the parent class constructor.  
• Inherited classes can override methods or add new ones.  
• Supports a clear hierarchical structure for complex applications.

## **ES6 Arrow Functions.**

• Arrow functions provide a shorter syntax for writing functions.  
• They do not have their own 'this' context and inherit it from the surrounding scope.  
• Arrow functions cannot be used as constructors.  
• Useful for callbacks and functional programming patterns.

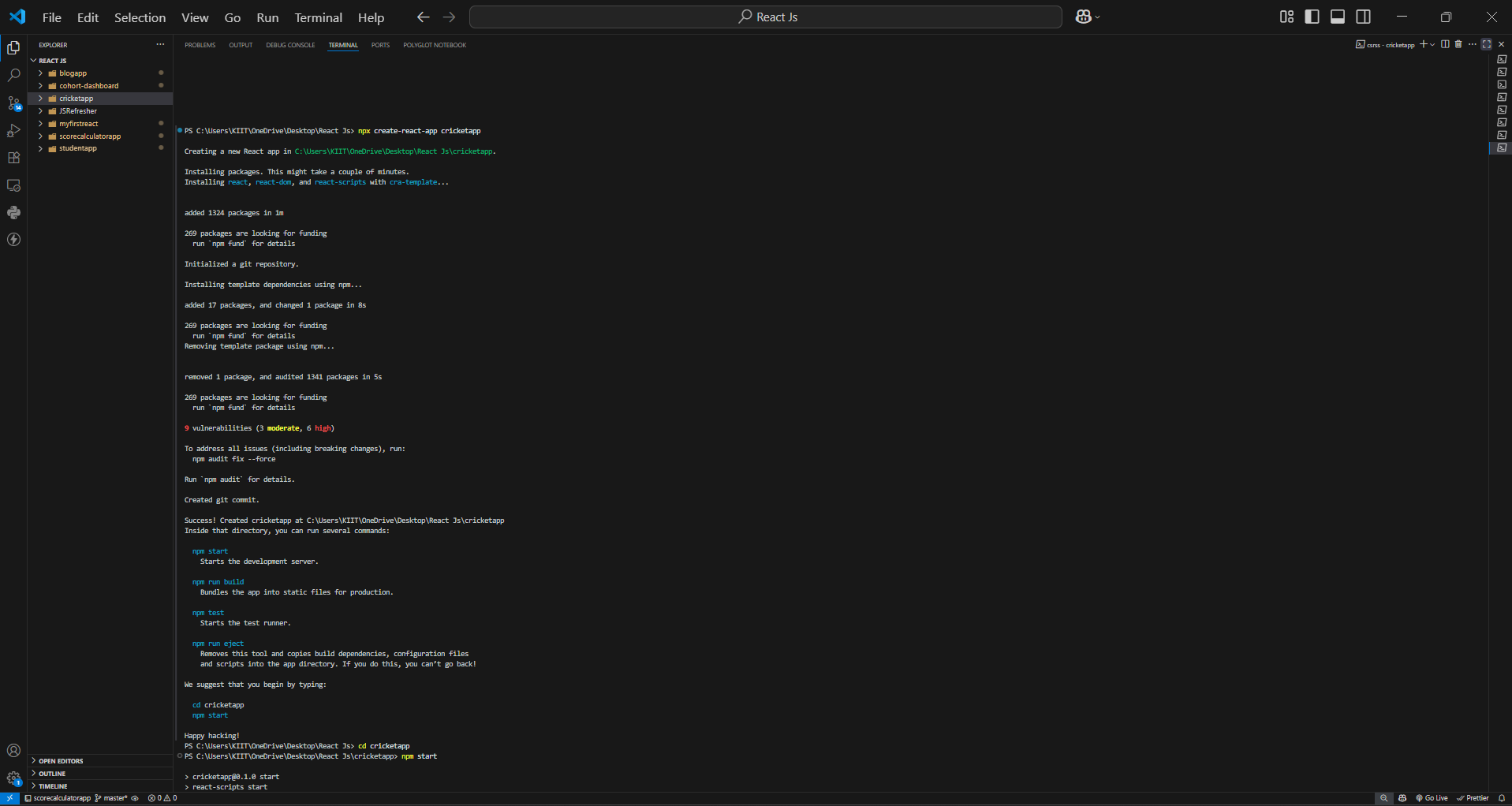
## **set() and map().**

**• Set:**  
 - A collection of unique values.  
 - Does not allow duplicate elements.  
 - Provides methods like add(), delete(), has().  
  
**• Map:**  
 - Stores key-value pairs.  
 - Keys can be of any data type.  
 - Provides methods like set(), get(), delete(), has().

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

## **Step 1: Create the React Application.**

**OUTPUT:-**



1. **ListofPlayers.**

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

**CODE:-**

import React from 'react';

const ListofPlayers = () => {

  const players = [

    { name: 'Virat Kohli', score: 95 },

    { name: 'Rohit Sharma', score: 80 },

    { name: 'KL Rahul', score: 65 },

    { name: 'Shikhar Dhawan', score: 55 },

    { name: 'Hardik Pandya', score: 72 },

    { name: 'Rishabh Pant', score: 60 },

    { name: 'Ravindra Jadeja', score: 85 },

    { name: 'MS Dhoni', score: 90 },

    { name: 'Jasprit Bumrah', score: 45 },

    { name: 'Mohammed Shami', score: 68 },

    { name: 'R Ashwin', score: 77 }

  ];

  return (

    <div>

      <h2>All Players</h2>

      {players.map((item, index) => {

        return (

          <div key={index}>

            <li>

              Mr. {item.name} <span>{item.score}</span>

            </li>

          </div>

        );

      })}

    </div>

  );

};

export default ListofPlayers;

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

**CODE:-**

 <h2>Players with Score ≤ 70</h2>

      {players70.map((item, index) => (

        <div key={index}>

          <li>

            Mr. {item.name} <span>{item.score}</span>

          </li>

        </div>

      ))}

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

**CODE:-**

import React from 'react';

export function OddPlayers([first, , third, , fifth]) {

  return (

    <div>

      <li>First : {first}</li>

      <li>Third : {third}</li>

      <li>Fifth : {fifth}</li>

    </div>

  );

}

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

**CODE:-**

const T20Players = ['First Player', 'Second Player', 'Third Player'];

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

export const IndianPlayers = [...T20Players, ...RanjiTrophyPlayers];

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

**Output:-**

**When Flag=true.**

**CODE :-**

import React from 'react';

import ListofPlayers from './ListofPlayers';

import IndianPlayersComponent from './IndianPlayersComponent';

function App() {

  const flag = true;

  return (

    <div className="App">

      <h1>Cricket App</h1>

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

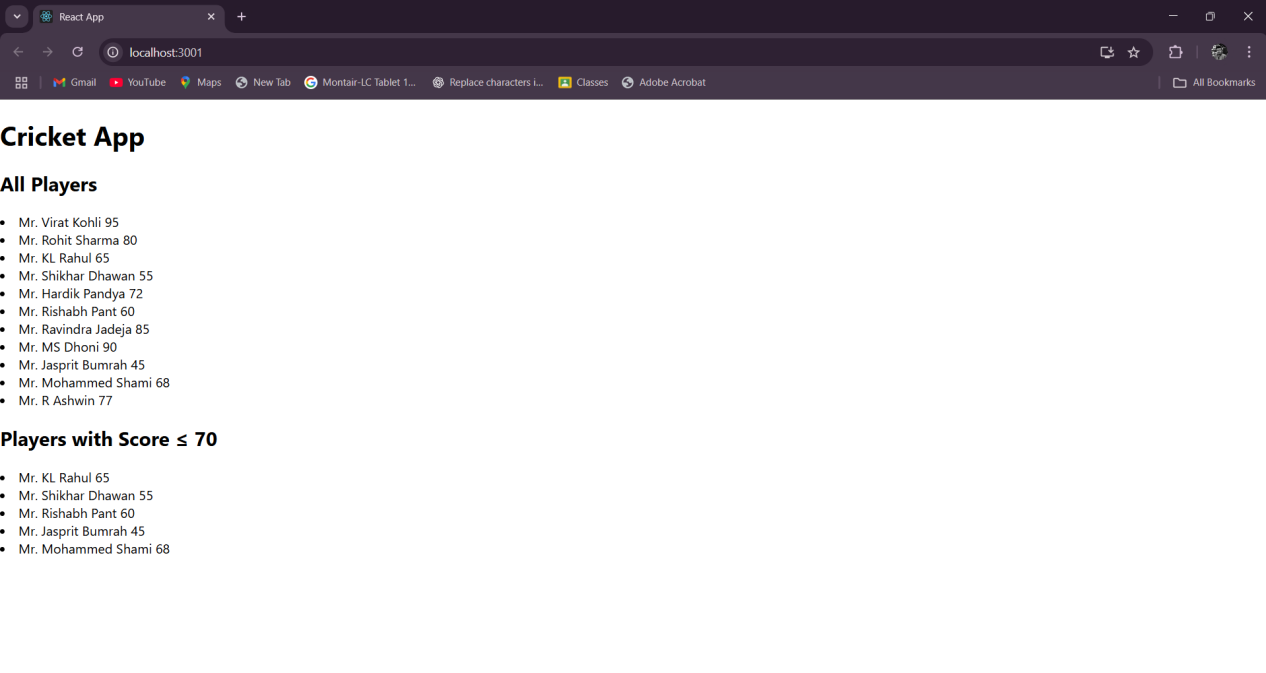
    </div>

  );

}

export default App;

**OUTPUT:-**



**When Flag=false.**

**CODE:-**

import React from 'react';

import ListofPlayers from './ListofPlayers';

import IndianPlayersComponent from './IndianPlayersComponent';

function App() {

  const flag = false;

  return (

    <div className="App">

      <h1>Cricket App</h1>

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

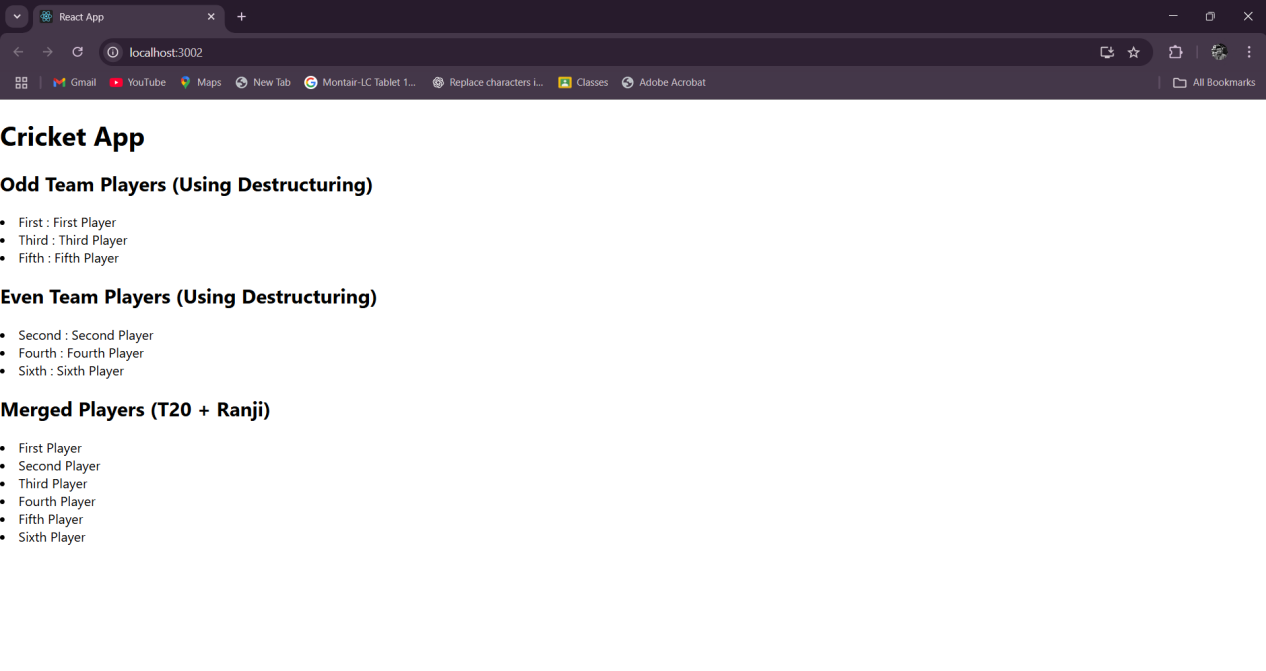
    </div>

  );

}

export default App;

**OUTPUT:-**



**-: ( ReactJs - 10 ) :-**

## **Objectives.**

## **Define JSX.**

JSX (JavaScript XML) is a syntax extension for JavaScript commonly used with React. It allows developers to write HTML-like code inside JavaScript, making it easier to create and visualize UI components. JSX is not directly understood by browsers; it is transpiled into regular JavaScript using tools like Babel.

## **2. Explain about ECMA Script.**

ECMAScript (ES) is a standardized scripting language specification upon which JavaScript is based. It defines the rules, syntax, and features of the language. Different versions of ECMAScript, such as ES5, ES6 (also known as ES2015), and beyond, introduce new functionalities, including classes, modules, arrow functions, and promises, which enhance JavaScript development.

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

React.createElement() is a core method in React used to create React elements without JSX. It takes three main arguments: the element type (string or component), an object containing props, and the children nodes. JSX code is ultimately transpiled into React.createElement() calls during compilation.

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

React nodes can be created using JSX by writing HTML-like tags directly in JavaScript. For example:-

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

Here, the JSX syntax defines a React node that represents an h1 element. Under the hood, this is converted into a React.createElement() call.

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

JSX can be rendered to the DOM using the ReactDOM.render() method. It takes a React element (created via JSX or React.createElement()) and a target DOM node as arguments. For example:

ReactDOM.render(<App />, document.getElementById('root'));

This renders the App component into the DOM element with the id 'root'.

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

JavaScript expressions can be embedded inside JSX using curly braces {}. This allows dynamic values to be rendered in the UI. For example:-

const name = 'John';

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

Here, the variable 'name' is evaluated and displayed inside the h1 tag.

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

Inline CSS in JSX is applied using the 'style' attribute, which accepts a JavaScript object. CSS property names are written in camelCase instead of kebab-case. For example:-

const styleObj = { color: 'blue', fontSize: '20px' };

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

This applies the inline styles to the h1 element.

**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.**

**CODE:-**

**(App.jsx):-**

import './App.css';

function App() {

  const office = {

    name: "Skyline Office",

    rent: 55000,

    address: "MG Road, Bangalore",

    image: "/office.jpg"

  };

  const officeList = [

    { name: "Skyline Office", rent: 55000, address: "MG Road, Bangalore" },

    { name: "Tech Park", rent: 75000, address: "Whitefield, Bangalore" },

    { name: "Business Bay", rent: 45000, address: "HSR Layout, Bangalore" }

  ];

  const getRentStyle = (rent) => ({

    color: rent < 60000 ? 'red' : 'green',

    fontWeight: 'bold'

  });

  return (

    <div className="App">

      <h1>Office Space Rental App</h1>

      <div className="office-card">

        <img src={office.image} alt="Office Space" style={{ width: '300px', borderRadius: '8px' }} />

        <h2>{office.name}</h2>

        <p style={getRentStyle(office.rent)}>Rent: ₹{office.rent}</p>

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

      </div>

      <hr />

      <h2>Available Office Spaces:</h2>

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

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

          <h3>{item.name}</h3>

          <p style={getRentStyle(item.rent)}>Rent: ₹{item.rent}</p>

          <p>Address: {item.address}</p>

        </div>

      ))}

    </div>

  );

}

export default App;

**(App.css) :-**

**CODE:-**

.App {

  text-align: center;

  padding: 20px;

  font-family: Arial, sans-serif;

}

.office-card {

  margin: 15px auto;

  padding: 15px;

  border: 1px solid #ddd;

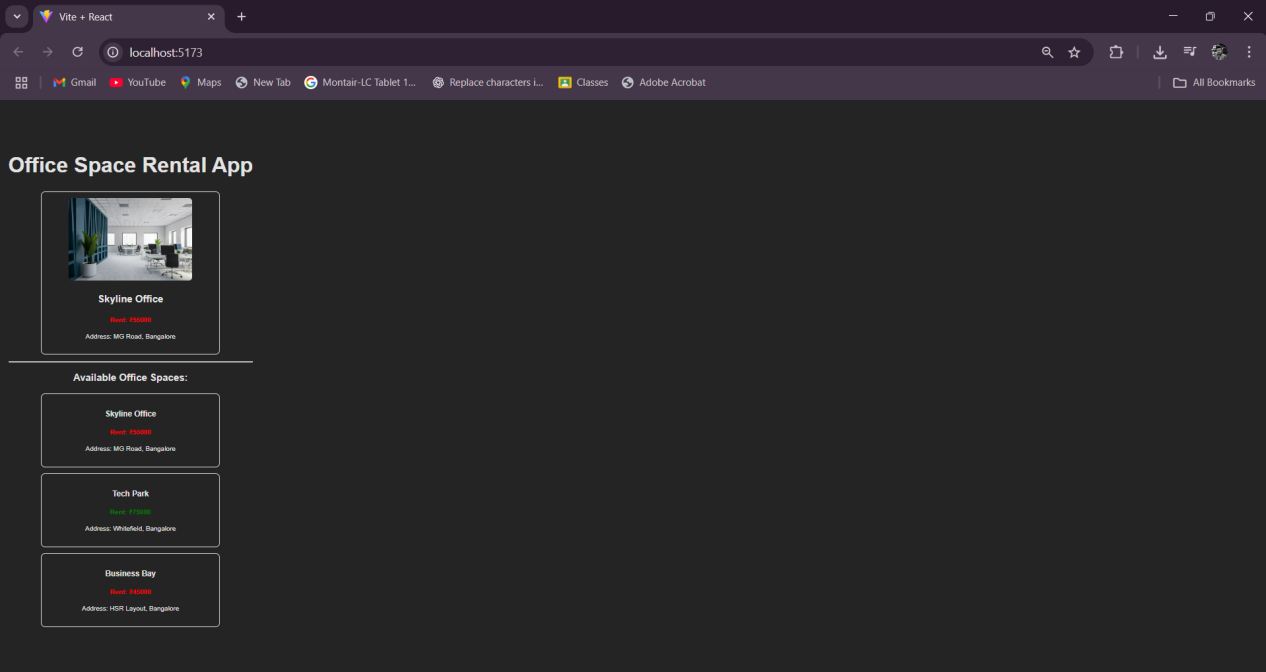
  border-radius: 10px;

  max-width: 400px;

  box-shadow: 0px 2px 5px rgba(0,0,0,0.1);

}

**Output:-**



## **-: (ReactJs - 11) :-**

## **Objectives.**

## **1. Explain React events.**

React events are similar to native DOM events but are wrapped in React’s SyntheticEvent for cross-browser compatibility. They allow you to handle user interactions such as clicks, form submissions, and key presses in a declarative way inside React components.

## **2. Explain about event handlers.**

Event handlers are functions that are triggered when a specific event occurs. In React, event handlers are written in camelCase (e.g., onClick, onChange) and are passed as functions instead of strings. They define the logic to execute when a user interacts with the UI.

## **3. Define Synthetic event.**

A SyntheticEvent is a cross-browser wrapper around the browser’s native event object provided by React. It normalizes the event properties to ensure consistent behavior across different browsers, and pools event objects for better performance.

## **4. Identify React event naming convention.**

React uses camelCase naming convention for event handlers instead of lowercase as in HTML. For example, onClick, onChange, and onSubmit are used in React instead of onclick, onchange, and onsubmit. Also, event handlers are passed as functions, not as strings.

**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.**

**CODE:-**

import React, { useState } from 'react';

function Counter() {

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

  const increment = () => {

    setCount(count + 1);

    sayHello();

  };

  const decrement = () => setCount(count - 1);

  const sayHello = () => {

    alert('Hello! This is a static message.');

  };

  return (

    <div>

      <h2>Counter: {count}</h2>

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

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

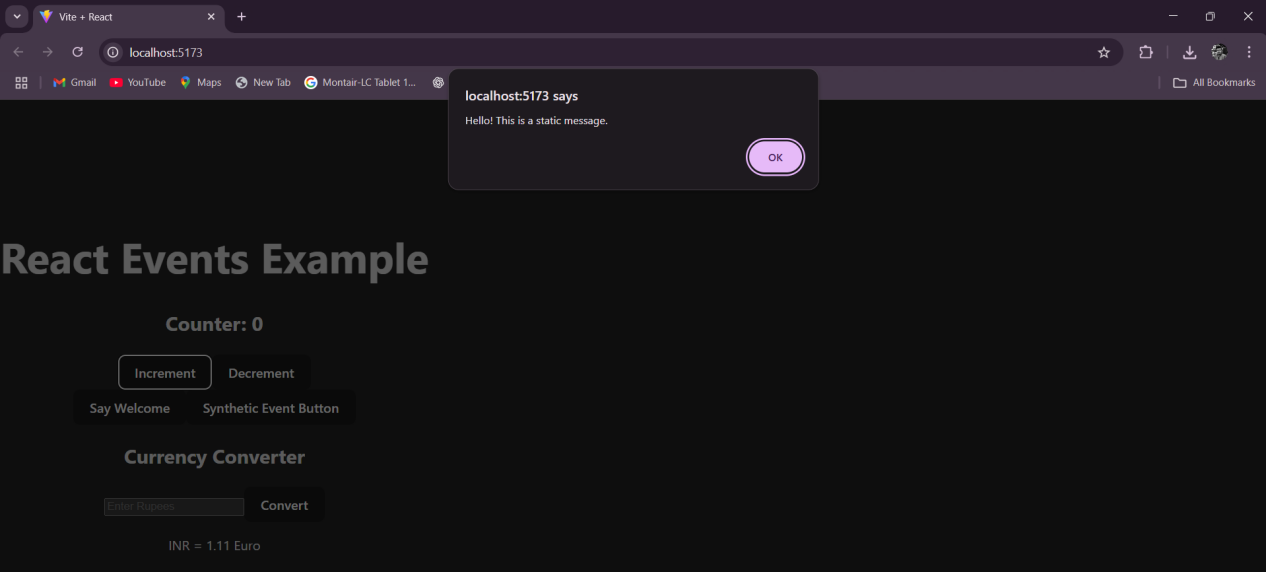
    </div>

  );

}

export default Counter;

**OUTPUT:-**



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

**CODE:-**

import React from 'react';

function WelcomeButton() {

  const sayWelcome = (msg) => {

    alert(`Message: ${msg}`);

  };

  return (

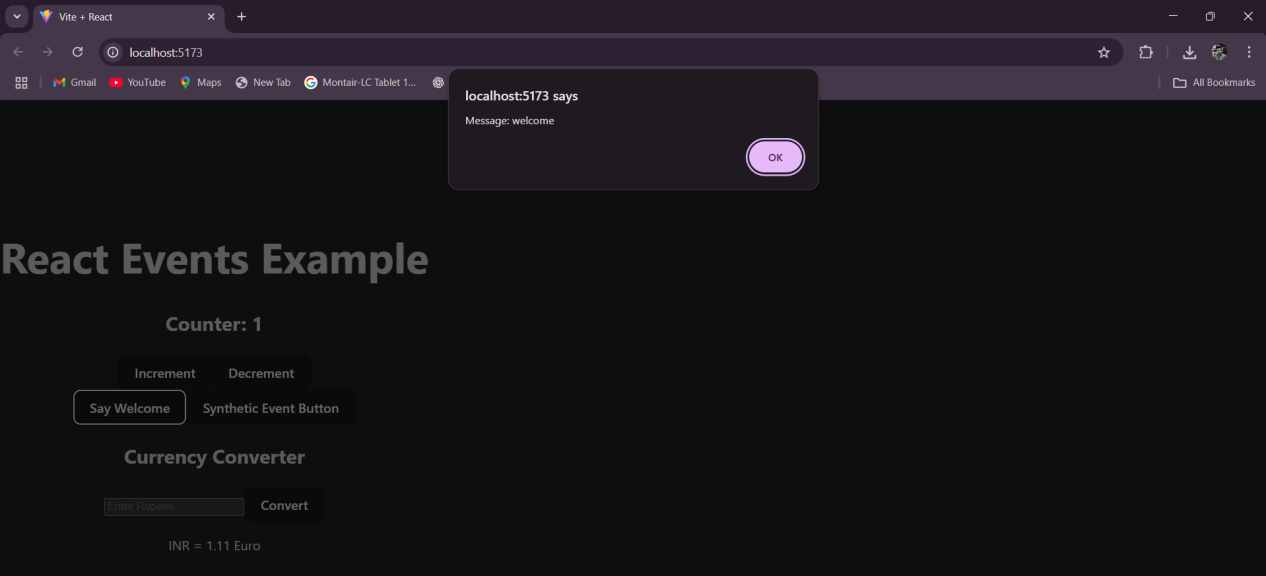
    <button onClick={() => sayWelcome('welcome')}>Say Welcome</button>

  );

}

export default WelcomeButton;

**OUTPUT:-**



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

**CODE:-**

import React from 'react';

function SyntheticEventButton() {

  const handleClick = (event) => {

    alert('I was clicked');

    console.log(event);

  };

  return (

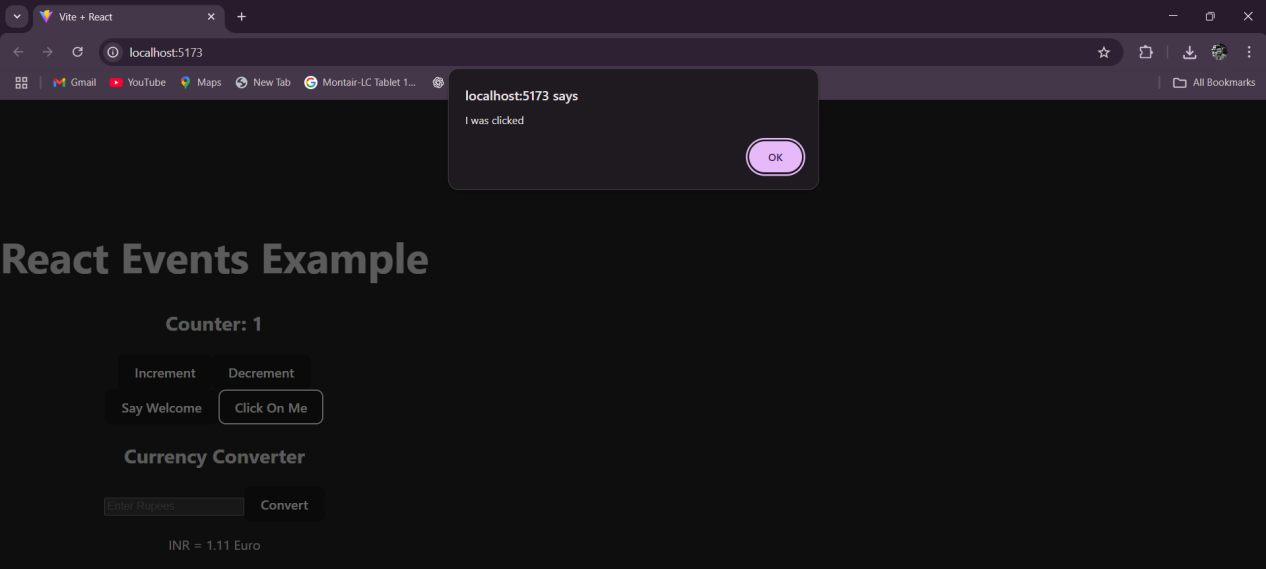
    <button onClick={handleClick}>Click On Me</button>

  );

}

export default SyntheticEventButton;

**OUTPUT:-**



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

**CODE:-**

import React, { useState } from 'react';

function CurrencyConvertor() {

  const [amount, setAmount] = useState('');

  const [currency, setCurrency] = useState('Euro');

  const handleSubmit = (e) => {

    e.preventDefault();

    if (currency === 'Euro') {

      const converted = parseFloat(amount) \* 80; // Example rate: 1 Euro = 80 INR

      alert(`Converting to ${currency}. Amount is ${converted}`);

    } else {

      alert('Please select a valid currency');

    }

  };

  return (

    <div>

      <h2>Currency Convertor!!!</h2>

      <form onSubmit={handleSubmit}>

        <div>

          <label>Amount: </label>

          <input

            type="number"

            value={amount}

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

            required

          />

        </div>

        <div>

          <label>Currency: </label>

          <select

            value={currency}

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

          >

            <option value="Euro">Euro</option>

          </select>

        </div>

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

      </form>

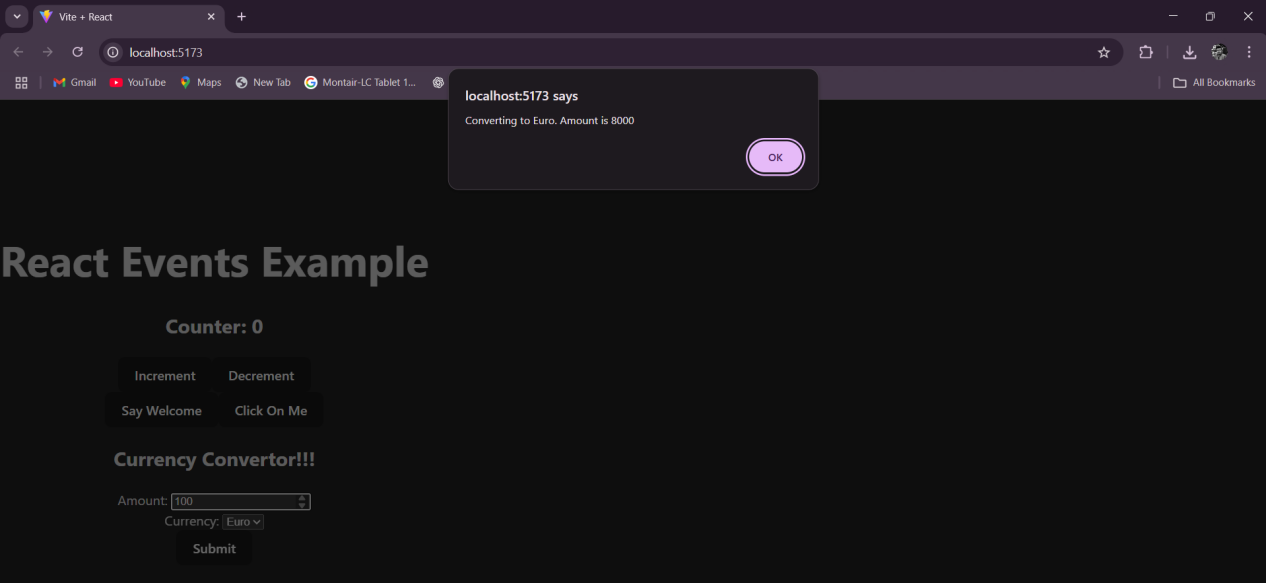
    </div>

  );

}

export default CurrencyConvertor;

**OUTPUT:-**



## **-: (ReactJs - 12) :-**

## **Objectives.**

## **1. Conditional Rendering in React.**

Conditional rendering in React allows components to render different outputs based on certain conditions. It works similarly to conditional statements in JavaScript (if-else, ternary operators). You can use it to dynamically display content, such as showing a login page when the user is not authenticated and a dashboard when they are logged in.

## **Element Variables.**

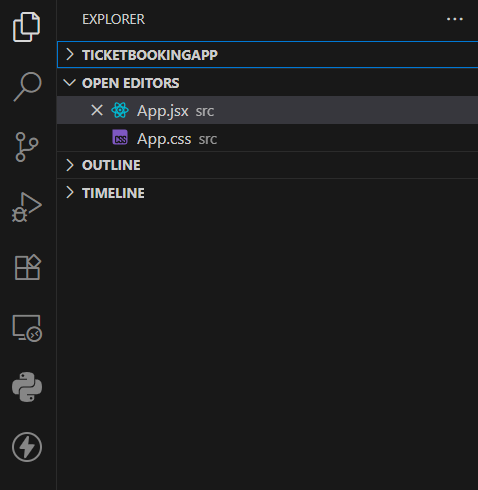
Element variables are used to store elements in variables and render them conditionally. Instead of writing conditional logic directly inside JSX, you can assign elements to variables and then include those variables in the render output. This makes the code more readable and manageable.

## **3. Preventing Components from Rendering.**

Sometimes, you may want to prevent a component from rendering under certain conditions. In React, you can achieve this by returning 'null' from the component's render function. When a component returns 'null', React will not render anything for that component. This technique is often used to hide components or avoid rendering unnecessary UI elements.

**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.**

**OUTPUT:-**



**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.**

**CODE:-**

**(App.jsx):-**

import React, { useState } from 'react';

function App() {

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

  return (

    <div style={{ textAlign: 'center', marginTop: '50px' }}>

      {isLoggedIn ? (

        <div>

          <h1>Welcome back</h1>

          <button onClick={() => setIsLoggedIn(false)}>Logout</button>

        </div>

      ) : (

        <div>

          <h1>Please sign up.</h1>

          <button onClick={() => setIsLoggedIn(true)}>Login</button>

        </div>

      )}

    </div>

  );

}

export default App;

**(App.css):-**

.App {

  text-align: center;

  margin-top: 40px;

  font-family: Arial, sans-serif;

}

button {

  padding: 10px 15px;

  margin: 10px;

  cursor: pointer;

  border: none;

  border-radius: 5px;

  background-color: #007bff;

  color: white;

  font-weight: bold;

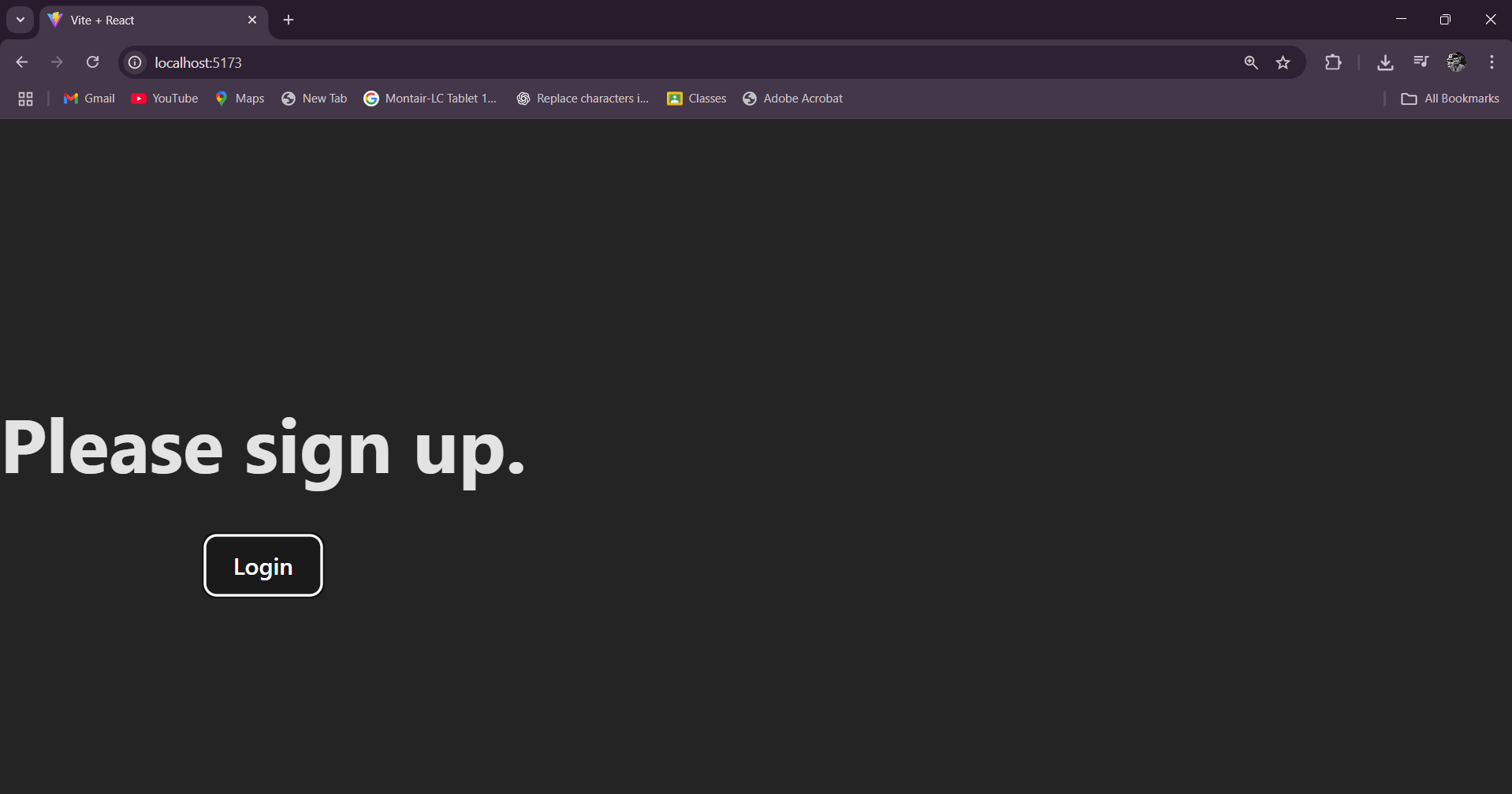
}

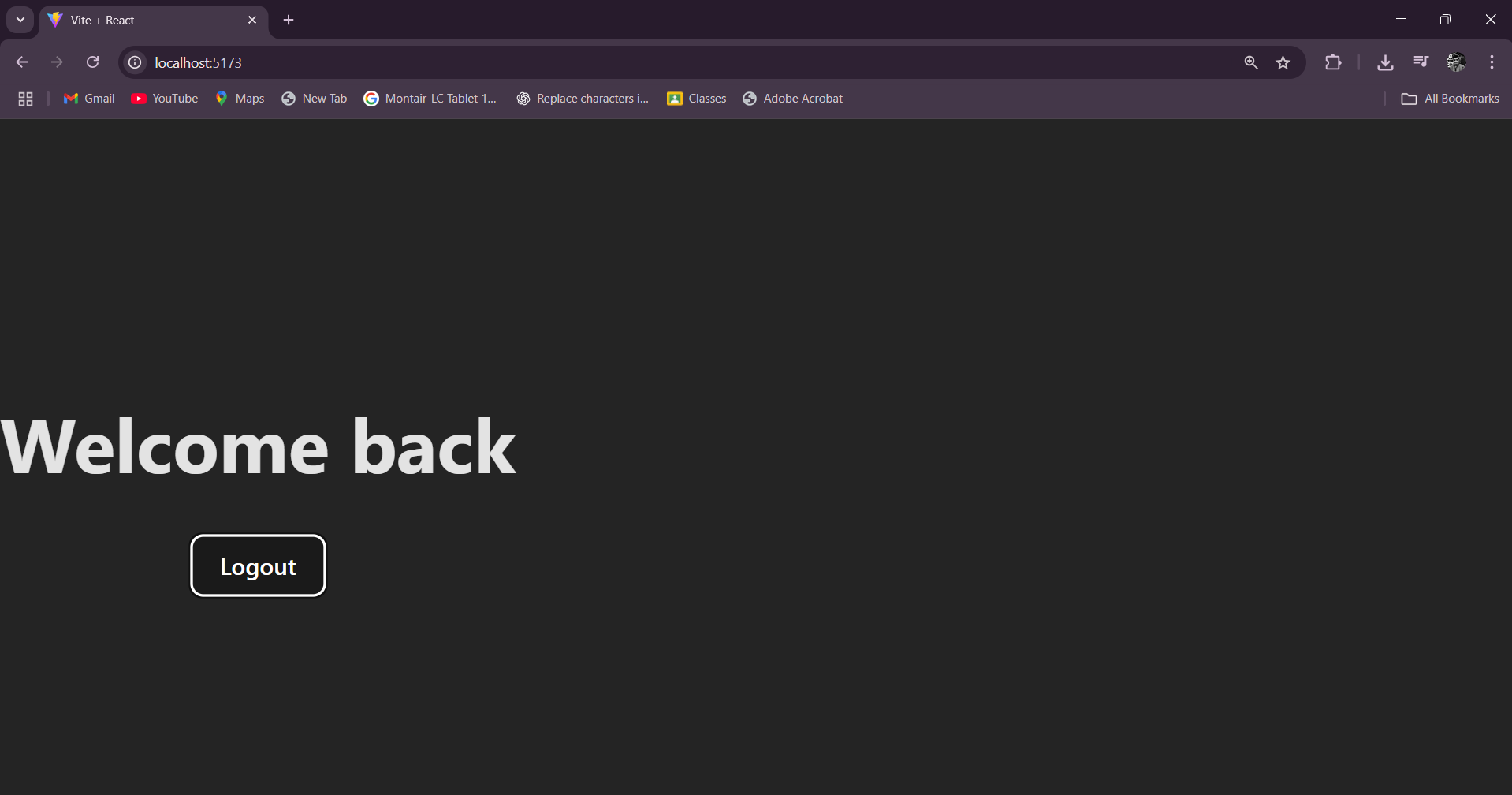
button:hover {

  background-color: #0056b3;

}

**OUTPUT:-**





## **-: (ReactJs - 13) :-**

## **Objectives.**

## **1. Various Ways of Conditional Rendering.**

Conditional rendering in React allows components to render different outputs based on conditions. Several methods can be used to achieve conditional rendering:  
  
• if/else statements: Standard JavaScript if/else blocks can be used to return components based on conditions.  
• Ternary operator: The ternary operator (condition ? true : false) is a concise way to render components conditionally.  
• Logical && operator: Renders a component only if a condition evaluates to true.  
• Switch case statements: Useful when multiple conditions need to be checked for rendering different components.  
• Using Immediately Invoked Function Expressions (IIFE): Enables inline conditional rendering inside JSX.

## **2. Rendering Multiple Components.**

React allows rendering multiple components simultaneously by including them within a parent element. This can be achieved using React fragments (<></>) or a parent <div> element. Fragments are preferred when you want to avoid adding extra nodes to the DOM.

## **3. Definition of a List Component.**

A list component in React is a component designed to render a collection of data items dynamically. It typically uses JavaScript's map() function to iterate over an array and display each element as a React element. This approach allows efficient rendering of data-driven UI components.

## **4. Keys in React Applications.**

Keys are unique identifiers used by React to keep track of elements in a list. They help React identify which items have changed, been added, or removed, optimizing the re-rendering process. Keys should be stable, predictable, and unique for each element in the list.

## **5. Extracting Components with Keys.**

When mapping data to components, keys should be assigned to the outermost element of the component being returned. If a component is extracted from a list, the key should be passed as a prop to ensure React can maintain identity across renders. Proper usage of keys ensures better performance and avoids unnecessary re-renders.

## **6. React Map and map() Function.**

The map() function in React is commonly used to transform arrays into lists of elements. It is a JavaScript array method that takes a callback function and applies it to each element in the array, returning a new array. In React, map() is often used to create lists of components from data arrays. Each element generated using map() should include a unique key prop for efficient rendering.

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

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

**Implement this with as many ways possible of Conditional Rendering.**

**CODES :-**

**( BlogDetails.jsx ) :-**

import React from "react";

function BlogDetails() {

  const blogs = [

    { title: "React Learning", author: "Stephen Biz", content: "Welcome to learning React!" },

    { title: "Installation", author: "Schwezdenier", content: "You can install React from npm." }

  ];

  return (

    <div className="section">

      <h2>Blog Details</h2>

      {blogs.map((blog, index) => (

        <div key={index}>

          <h3>{blog.title}</h3>

          <p><strong>{blog.author}</strong></p>

          <p>{blog.content}</p>

        </div>

      ))}

    </div>

  );

}

export default BlogDetails;

**( BookDetails.jsx ) :-**

import React from "react";

function BookDetails() {

  const books = [

    { title: "Master React", price: 670 },

    { title: "Deep Dive into Angular 11", price: 800 },

    { title: "Mongo Essentials", price: 450 }

  ];

  return (

    <div className="section">

      <h2>Book Details</h2>

      {books.map((book, index) => (

        <div key={index}>

          <h3>{book.title}</h3>

          <p>{book.price}</p>

        </div>

      ))}

    </div>

  );

}

export default BookDetails;

**( CourseDetails.jsx ) :-**

import React from "react";

function CourseDetails() {

  const courses = [

    { name: "Angular", date: "4/5/2021" },

    { name: "React", date: "6/3/2021" }

  ];

  return (

    <div className="section">

      <h2>Course Details</h2>

      {courses.map((course, index) => (

        <div key={index}>

          <h3>{course.name}</h3>

          <p>{course.date}</p>

        </div>

      ))}

    </div>

  );

}

export default CourseDetails;

**( App.jsx ) :-**

import React from "react";

import CourseDetails from "./components/CourseDetails";

import BookDetails from "./components/BookDetails";

import BlogDetails from "./components/BlogDetails";

import "./App.css";

function App() {

  return (

    <div className="container">

      <CourseDetails />

      <BookDetails />

      <BlogDetails />

    </div>

  );

}

export default App;

**( App.css ) :-**

.container {

  display: flex;

  justify-content: space-around;

  padding: 30px;

  font-family: Arial, sans-serif;

}

.section {

  flex: 1;

  margin: 0 10px;

  padding: 20px;

  border-left: 3px solid green;

}

.section h2 {

  font-size: 1.5rem;

  font-weight: bold;

}

.section h3 {

  margin-top: 10px;

  margin-bottom: 5px;

}

.section p {

  margin: 0;

}

**OUTPUT:-**

