**Week 7 Hands on**

**9. Documentation – ES6 Features and React Hands-on Lab**

**Overview**

In this lab, I created a React application named **“cricketapp”** to implement and demonstrate modern JavaScript (ES6) features in a practical setup. The project consists of two functional components – ListofPlayers and IndianPlayers – which together showcase the use of map(), arrow functions, destructuring, array merging, and conditional rendering in React using ES6 syntax.

**What I Did**

* I started by setting up the React project named cricketapp using create-react-app.
* Inside this project, I created two components: ListofPlayers and IndianPlayers.

**In the ListofPlayers component:**

* I declared an array of 11 cricket players, each with a name and a score.
* I displayed the list of all players using the map() function of ES6.
* Then, I used an **arrow function** to filter and display only those players whose score was **below 70**.

**In the IndianPlayers component:**

* I declared two arrays: T20Players and RanjiTrophyPlayers.
* I used **destructuring** to split players into odd and even teams.
* I merged the two arrays using the ES6 **spread operator**, and displayed the result.
* Finally, I used a simple **if-else condition based on a flag variable** to render one component at a time on the homepage:
  + If flag is true, it shows ListofPlayers.
  + If flag is false, it shows IndianPlayers.

**Objectives and How I Addressed Them**

**• List the features of ES6**

I used the following ES6 features in the lab:

* let and const for variable declarations
* Arrow functions
* map() method
* Object and array destructuring
* Spread operator for merging arrays
* ES6 class structure (though not central to this lab)

**• Explain JavaScript let**

I used let to declare block-scoped variables which may be reassigned later. This avoids the function-scoping issue of var.

**• Identify the differences between var and let**

* var is function-scoped and allows hoisting.
* let is block-scoped and does not hoist in the same way.  
  In the lab, I used let to maintain cleaner and more predictable variable scope.

**• Explain JavaScript const**

I used const to declare arrays and objects that were not meant to be reassigned. Though their internal contents (like array elements) can be modified, the reference remains constant.

**• Explain ES6 class fundamentals**

Though I did not create custom classes in this project, I am familiar with the concept of defining constructors, methods, and inheritance using class and extends.

**• Explain ES6 class inheritance**

Not directly implemented in this lab, but I understand that ES6 allows one class to inherit properties and methods from another using the extends keyword.

**• Define ES6 arrow functions**

Arrow functions were used for filtering players. They provide a concise syntax and preserve the this context compared to traditional functions.

**• Identify set() and map()**

While I used map() extensively to iterate over player arrays, I did not require set() for this use case. I understand that map() is used for transforming arrays and set() is a collection of unique values.

**Hands-On Objectives Completed**

**• Use map() method of ES6**

Yes, used to display the list of 11 players.

**• Apply arrow functions of ES6**

Yes, used to filter out players scoring below 70.

**• Implement destructuring features of ES6**

Yes, used to separate players into Odd and Even teams and to merge two arrays.

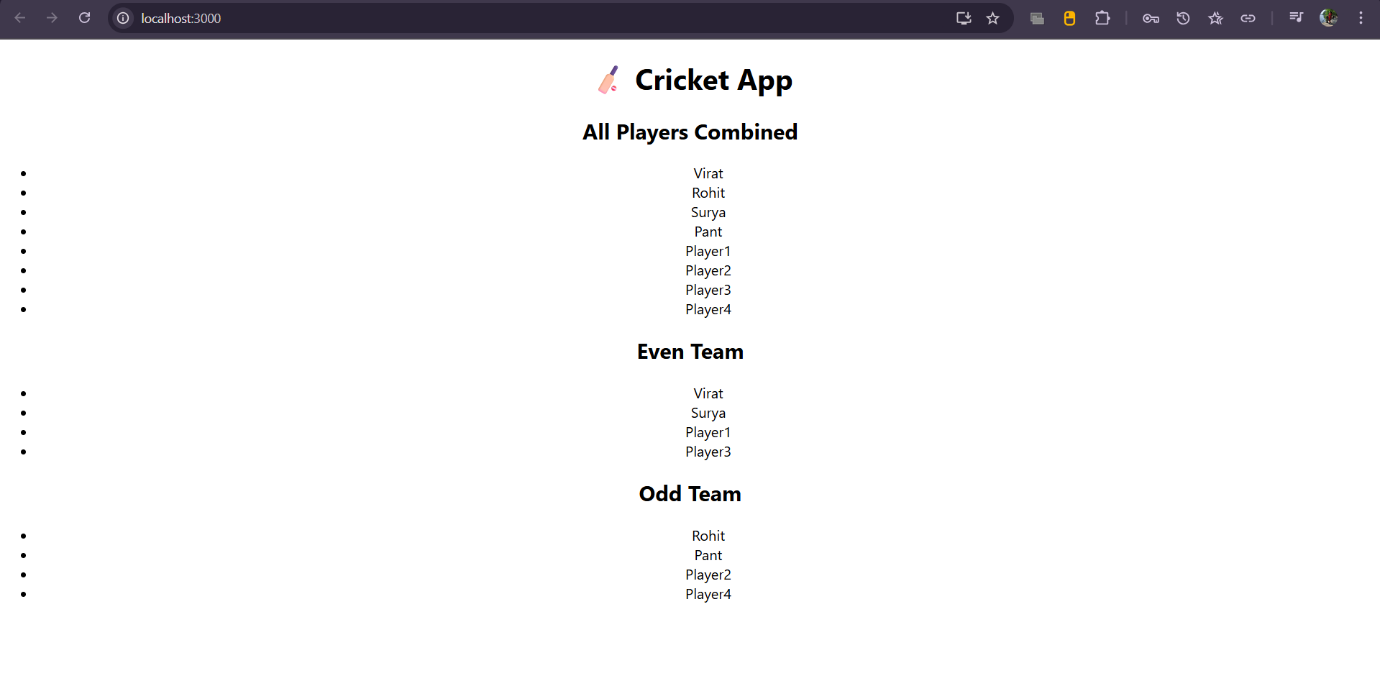
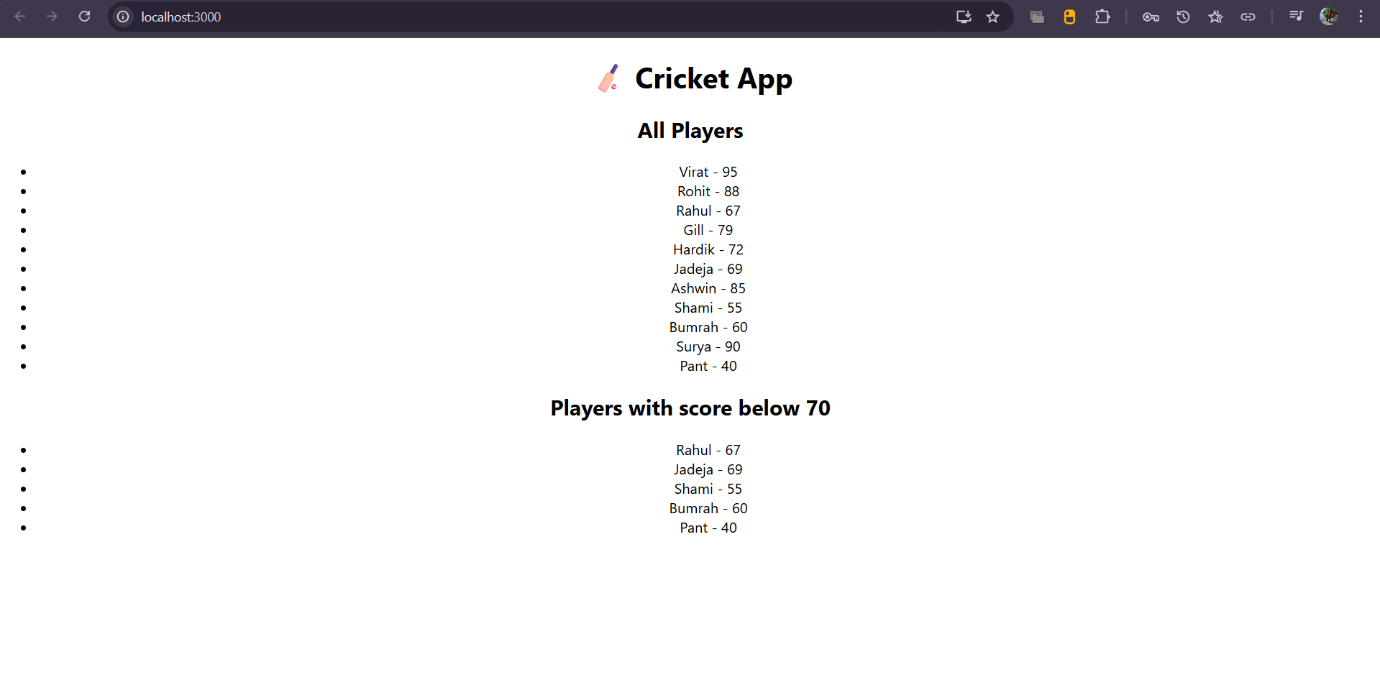
**Prerequisites Used**

* Node.js
* NPM
* Visual Studio Code

All tools were used for project setup, development, and running the application.

**Estimated Time Taken**

The task was estimated for 60 minutes. I completed the lab within the given time frame including development, testing, and documentation.

Output:

**10.** **Documentation – React JSX and Inline CSS Lab**

**Overview**

In this hands-on lab, I created a React application named **“officespacerentalapp”** to practice and implement the core features of **JSX** in React, including rendering elements, using JavaScript expressions inside JSX, applying inline CSS dynamically, and rendering data from objects and arrays. This lab helped me understand how JSX works under the hood and how it integrates seamlessly with JavaScript.

**What I Did**

* I started by setting up the React project using create-react-app and named it officespacerentalapp.
* I created a React component where I:
  + Displayed a **page heading** using JSX.
  + Added an **image element** using JSX attributes to visually represent the office space.
  + Declared an **object named office** that holds details like Name, Rent, and Address, and rendered those properties dynamically using JSX expressions.
  + Created a **list of office objects**, and used the map() function to loop through each office and display its data in a structured format.
* For styling:
  + I applied **inline CSS** in JSX to dynamically change the color of the rent amount.
  + If the rent was **below ₹60,000**, the text appeared in **red**; otherwise, it appeared in **green**.
  + This was done using a conditional (ternary) operator inside the JSX style attribute.

**Objectives and How I Addressed Them**

**• Define JSX**

JSX stands for **JavaScript XML**. It is a syntax extension for JavaScript used with React to describe what the UI should look like. I used JSX to create and render UI elements in the app.

**• Explain about ECMA Script**

ECMAScript is the scripting language standard upon which JavaScript is based. ES6 (ECMAScript 2015) introduced several modern features like arrow functions, let/const, destructuring, and modules—all of which are supported in React. I used ES6 features alongside JSX in this lab.

**• Explain React.createElement()**

JSX is syntactic sugar for React.createElement(). Under the hood, JSX expressions are transpiled into React.createElement() calls. While I didn’t manually write React.createElement() in this lab, every JSX line in my code is ultimately converted to this method during compilation.

**• Explain how to create React nodes with JSX**

I created React nodes (like <h1>, <img>, and custom JSX blocks) using standard JSX syntax. JSX allows me to write HTML-like code within JavaScript files, making UI development more intuitive.

**• Define how to render JSX to DOM**

JSX is rendered to the DOM by React’s rendering engine. I returned JSX from my component's return() function, and React mounted it to the DOM using the ReactDOM.render() method internally.

**• Explain how to use JavaScript expressions in JSX**

I embedded JavaScript expressions inside curly braces {} within JSX. For example, I displayed office.name, office.rent, and dynamically applied style conditions using expressions inside JSX.

**• Explain how to use inline CSS in JSX**

I used the style attribute in JSX, which takes a JavaScript object instead of a traditional string. I wrote conditional inline styles to color the rent text:

style={{ color: office.rent < 60000 ? "red" : "green" }}

This allowed dynamic styling based on the rent amount.

**Hands-On Objectives Completed**

**• Use JSX syntax in React applications**

Yes, I used JSX to create heading elements, image attributes, object rendering, and list rendering.

**• Use inline CSS in JSX**

Yes, I implemented inline conditional styling to color the rent value based on conditions.

**Prerequisites Used**

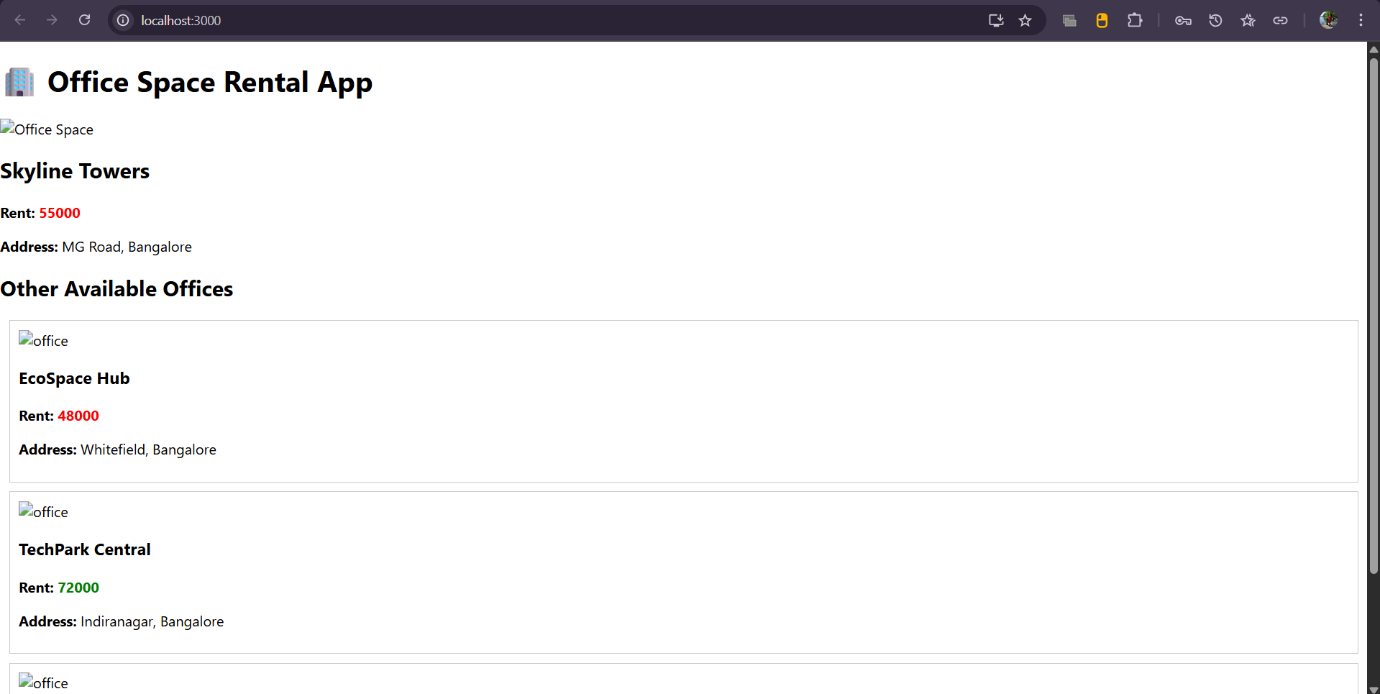
* Node.js
* NPM
* Visual Studio Code

These tools were used for creating the React app, writing the code, and running the development server.

**Estimated Time Taken**

The estimated completion time was 60 minutes. I was able to complete the setup, development, styling, and testing within the given timeframe.

Output:



**11. Documentation – React Event Handling and Synthetic Events Lab**

**Overview**

In this lab, I created a React application named **“eventexamplesapp”** to implement and understand various **event handling techniques** in React. The tasks involved using event listeners on form elements and buttons, working with the this keyword, invoking synthetic events, and passing arguments to event handler functions. The lab also included building a CurrencyConvertor component to apply event handling in a practical scenario.

**What I Did**

* I initialized a React project called eventexamplesapp using create-react-app.

**1. Counter with Increment and Decrement Buttons:**

* I created a simple counter using state.
* The **"Increment"** button was wired to:
  + A method that **increments the counter**
  + Another method that prints **"Hello, welcome to the counter app"** (demonstrating multiple method calls on one event)
* The **"Decrement"** button reduced the counter value by one using a separate event handler function.

**2. Say Welcome Button:**

* I created a **"Say Welcome"** button.
* It was linked to a function that accepts a string argument "Welcome" and displays it when the button is clicked. This demonstrated **how to pass arguments to event handlers** in React.

**3. Synthetic Event – OnPress:**

* I implemented a button that simulates a **synthetic event**.
* When clicked, it displays the message **"I was clicked"**, showing how synthetic events work in React.

**4. CurrencyConvertor Component:**

* I developed a separate component named CurrencyConvertor.
* This component included:
  + An input field for Indian Rupees
  + A “Convert” button
* On button click, it handled the handleSubmit event which:
  + Retrieved the input value
  + Converted the amount into **Euros** using a fixed exchange rate (e.g., 1 Euro = ₹88)
  + Displayed the converted result

**Objectives and How I Addressed Them**

**• Explain React events**

React events are event handlers like onClick, onChange, etc., which allow us to respond to user actions. I used these events to detect clicks on buttons and changes in input fields.

**• Explain about event handlers**

Event handlers are functions that respond to an event in React. In this lab, I wrote multiple event handler functions for incrementing counters, displaying welcome messages, and converting currency.

**• Define Synthetic event**

Synthetic events are React’s wrapper around native browser events. They provide a consistent and cross-browser interface. In this lab, I used a synthetic event in the “I was clicked” button to demonstrate how React handles such events internally.

**• Identify React event naming convention**

In React, event names follow **camelCase** notation, unlike HTML which uses lowercase. For example, onClick, onChange, and onSubmit are all camelCase. I followed this convention throughout the lab.

**Hands-On Objectives Completed**

**• Implement Event handling concept in React applications**

Yes, I implemented click event handlers, argument-passing handlers, and form submission handlers using React.

**• Use this keyword**

Yes, I used the this keyword within class-based components to refer to component methods and state. I ensured proper binding to avoid undefined context issues.

**• Use Synthetic event**

Yes, I created a button that handled a synthetic event (onClick) to display a message, showcasing React’s abstraction over native DOM events.

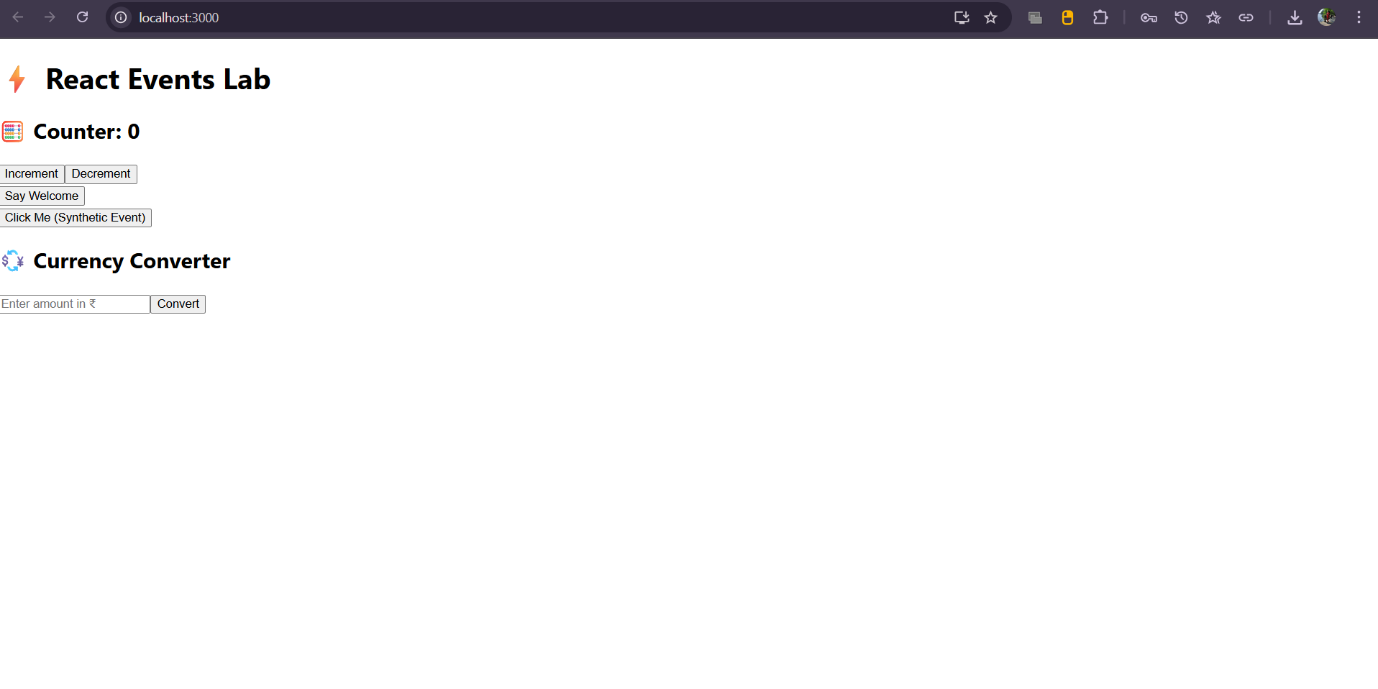
**Prerequisites Used**

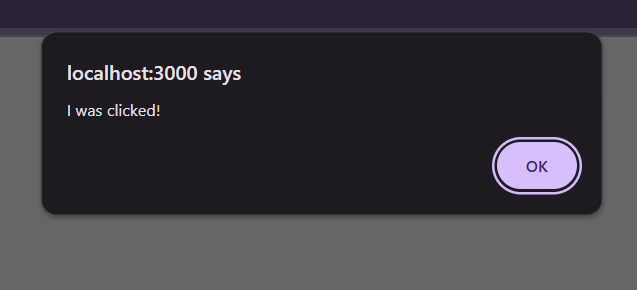
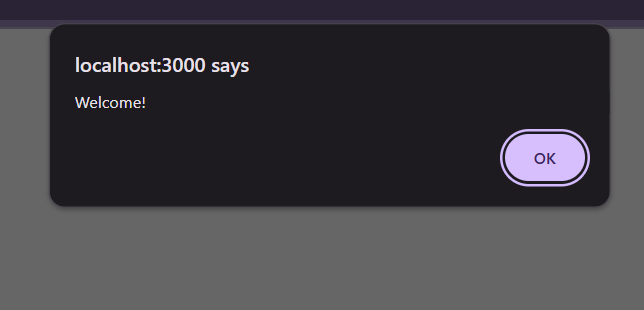
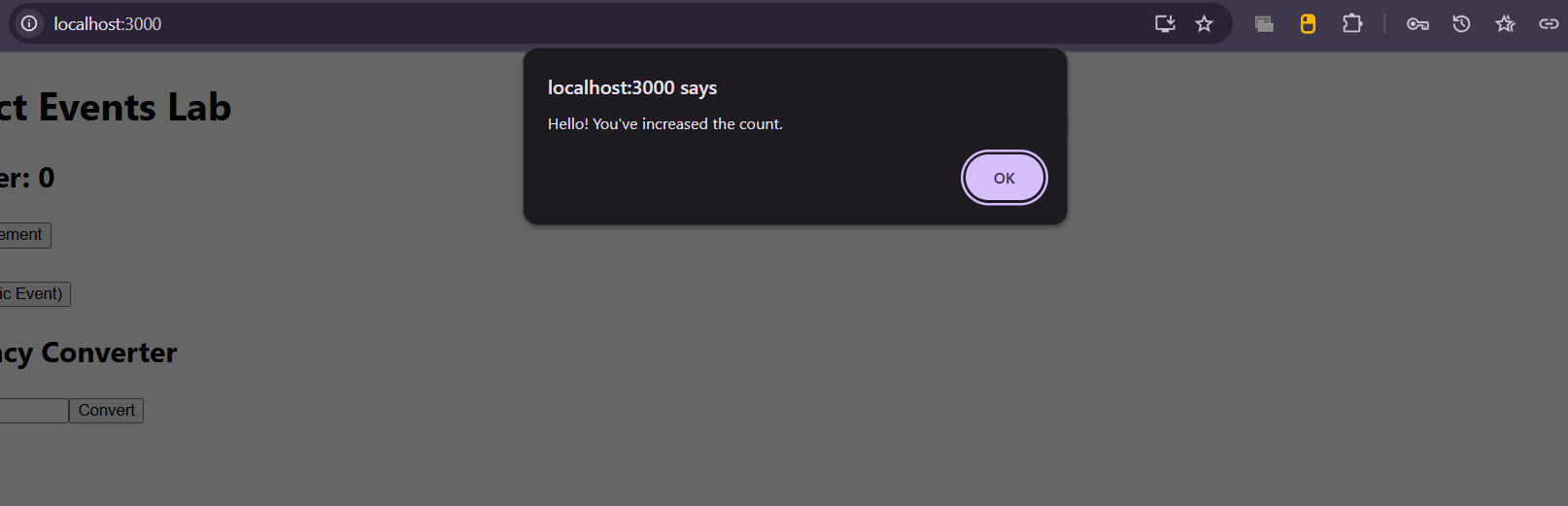
* Node.js
* NPM
* Visual Studio Code

All tools were used for setting up, writing, running, and testing the application.

**Estimated Time Taken**

The lab was estimated to take 90 minutes. I completed the setup, component creation, event handling implementation, and testing within the allotted time.

Output:





12. **✅ Documentation for “ticketbookingapp” React Lab**

In this lab, I created a React application named **“ticketbookingapp”** to demonstrate the concept of **conditional rendering** based on user authentication status.

**💻 What I Did (My POV Documentation):**

1. **Set up the React App** using npx create-react-app ticketbookingapp.
2. Created two simple components:
   * **GuestPage** → displays flight details only (read-only).
   * **UserPage** → allows logged-in users to **book tickets**.
3. Created a **Login/Logout toggle button**:
   * On click of **Login**, state changes to isLoggedIn: true → shows UserPage.
   * On click of **Logout**, state resets to isLoggedIn: false → shows GuestPage.
4. Used:
   * **Element variables** to hold components conditionally rendered.
   * A central component (App.js) to handle logic.
   * **JSX short-circuiting** and ternary operator for rendering.
5. Verified that:
   * On initial load → **GuestPage** is shown.
   * On clicking **Login**, it switches to **UserPage**.
   * On clicking **Logout**, it goes back to **GuestPage**.

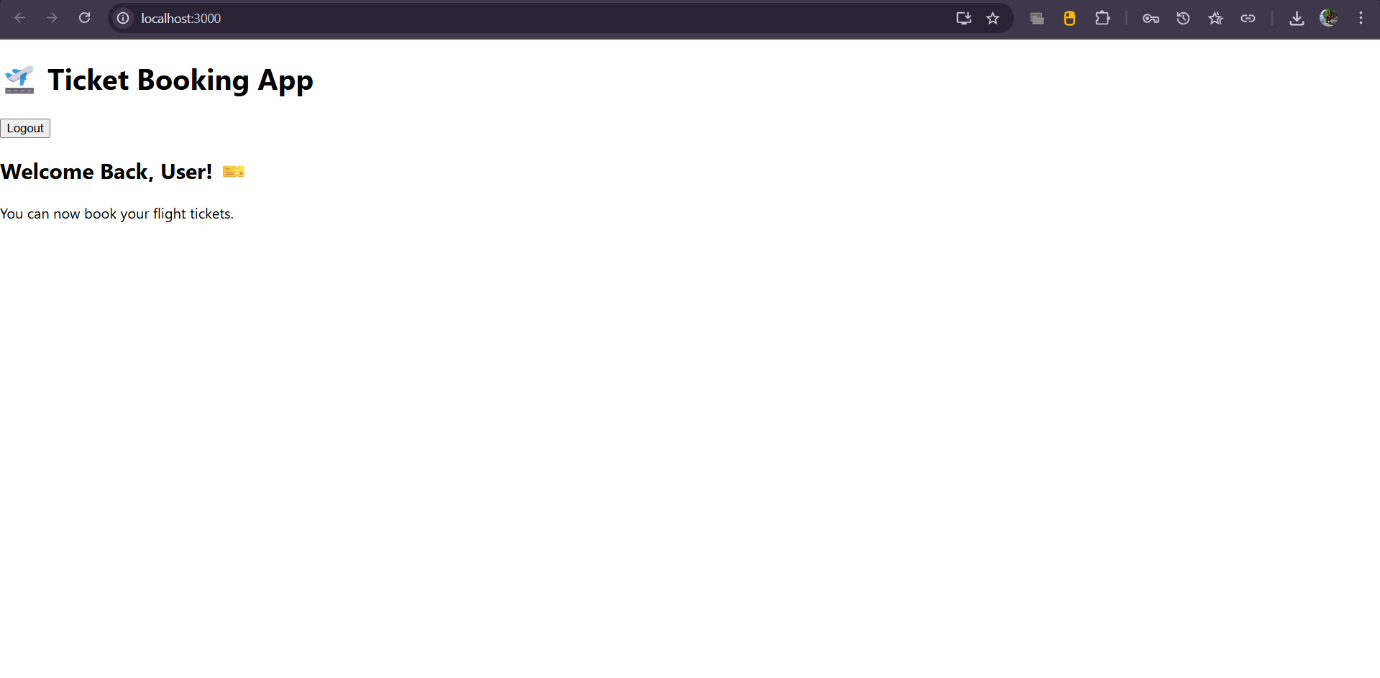
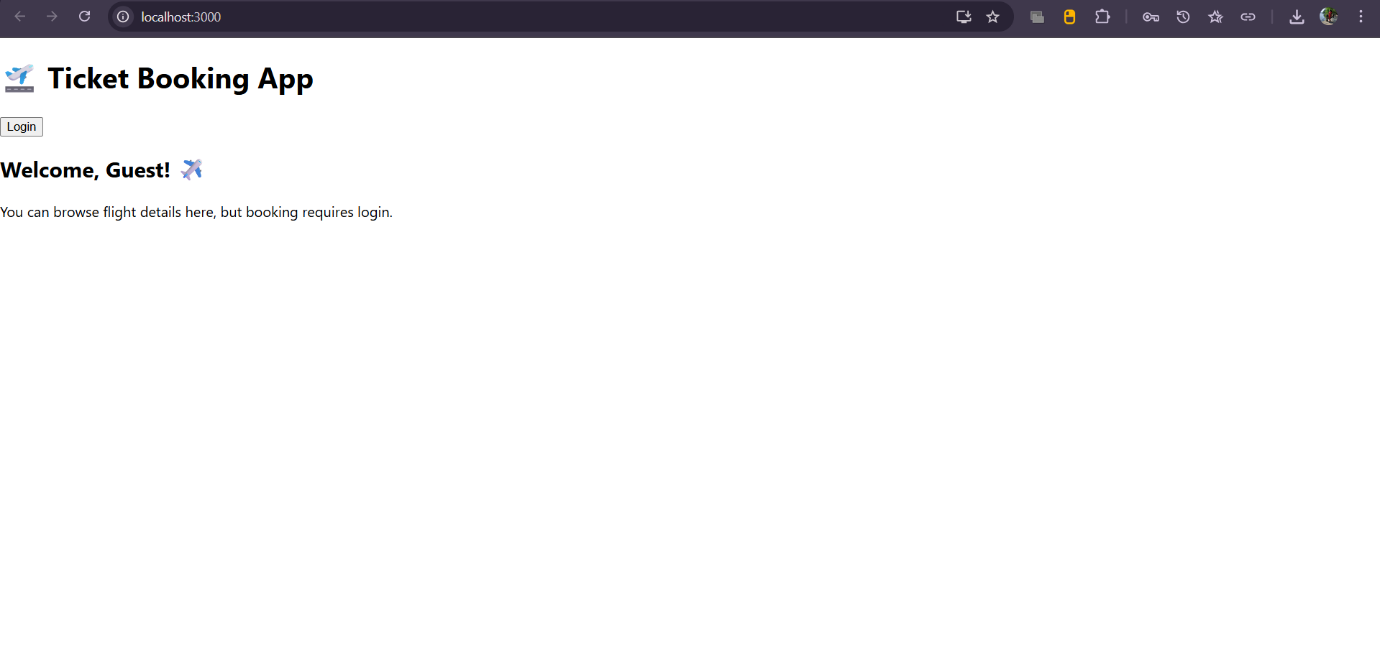
**Objectives Covered:**

**✅ Explain about conditional rendering in React**  
→ Used if conditions and ternary operators to switch between views based on login status.

**✅ Define element variables**  
→ Stored JSX in variables to conditionally assign them inside render.

**✅ Explain how to prevent components from rendering**  
→ Used conditional checks like isLoggedIn ? <UserPage /> : <GuestPage /> so only the relevant component is shown at any time.

Output:



13. **Documentation – React Conditional Rendering and Lists Lab**

**Overview**

In this lab, I created a React application named **“bloggerapp”** to explore and apply various techniques of **conditional rendering** and **list rendering** in React. The tasks involved rendering multiple components based on conditions, creating lists with keys, extracting components for reuse, and using the map() function to dynamically render data. The goal was to get hands-on experience with rendering strategies and best practices in React UI design.

**What I Did**

• I initialized a new React project called **bloggerapp** using create-react-app.

**1. Created Three Core Components:**

• BookDetails  
• BlogDetails  
• CourseDetails

Each component was designed to display specific content using props and dummy data lists.

**2. Implemented Conditional Rendering in Multiple Ways:**

• **If-else logic** using classic JavaScript conditions  
• **Ternary operators** to switch between components dynamically  
• **Logical && operator** for short-circuit conditional rendering  
• **Element variables** to store which component to render based on state

I used a dropdown or buttons to simulate user input and change which section is shown.

**3. Rendered Multiple Components Conditionally:**

• Based on a selected category, the app displayed the relevant component (Books, Blogs, or Courses).  
• I used a currentView state to track user selection and display components accordingly.

**4. Rendered Lists Using map():**

• Inside each component (e.g., BookDetails), I used a dummy array of objects with details like title, author, or price.  
• These were rendered using map() with proper **keys** to ensure React could track list items efficiently.

**5. Explained and Used Keys:**

• Assigned unique keys to each list item to prevent rendering issues and warnings.  
• Used index or ID from the object data as the key while mapping.

**6. Extracted Components with Keys:**

• Instead of writing list elements inline, I created reusable subcomponents (e.g., BookItem, BlogCard) and passed data via props.  
• This demonstrated how to separate logic and maintain clean code while handling lists.

**Objectives and How I Addressed Them**

• **Explain various ways of conditional rendering**  
→ I implemented if-else, ternary operators, && short-circuit, and element variables to control component visibility.

• **Explain how to render multiple components**  
→ I displayed Book, Blog, and Course components dynamically based on the selected state. Used multiple rendering strategies inside a single file.

• **Define list component**  
→ I created lists using map() inside each detail component and rendered content from an array of objects.

• **Explain about keys in React applications**  
→ I used key props inside map() and explained their role in performance and preventing re-rendering bugs.

• **Explain how to extract components with keys**  
→ Created small reusable components for list items and passed unique keys while rendering via map().

• **Explain React Map, map() function**  
→ Used map() to loop over data arrays like books and courses. Each iteration returned JSX which was displayed in the UI.

**Hands-On Objectives Completed**

• **Implement conditional rendering in React applications**  
✅ Yes, done using four different approaches, integrated across three components.

**Prerequisites Used**

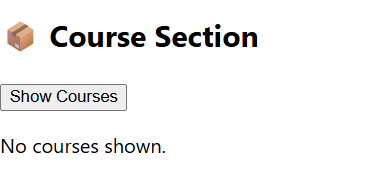
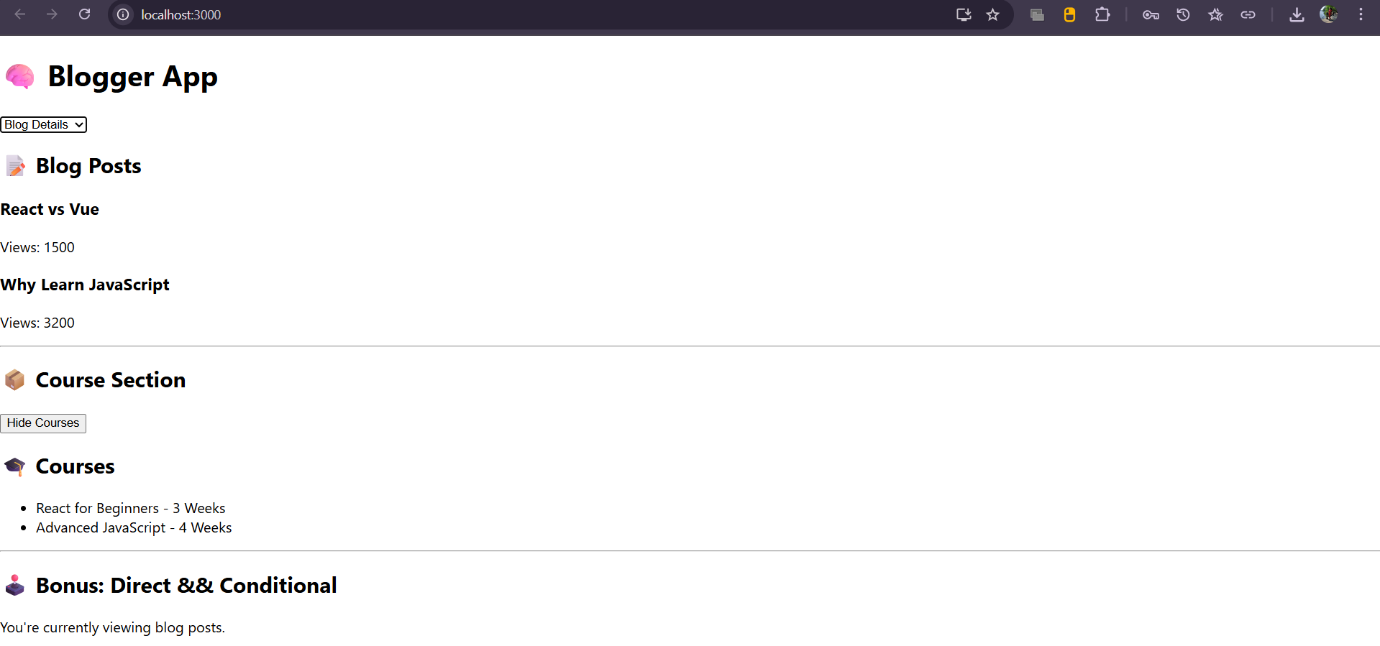
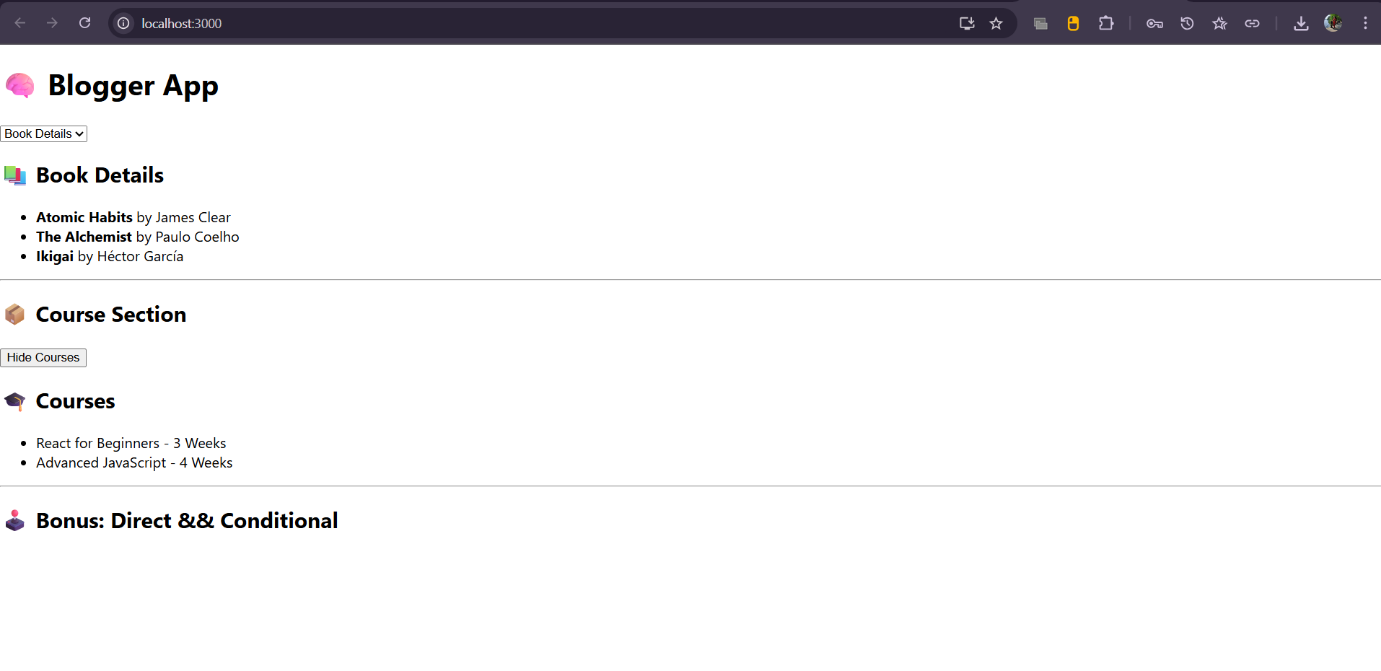
• Node.js  
• NPM  
• Visual Studio Code

All three tools were essential for setup, coding, running the dev server, and debugging output.

**Estimated Time Taken**

The estimated time was **60 minutes**. I was able to finish all tasks, implement conditional rendering, set up list views, and extract reusable components within the timeframe.

Output:



**14.Documentation**

I started by unzipping the given application and opening it in **Visual Studio Code**. Then I restored all required node modules by running npm install in the terminal. After installation, I launched the app using npm start to confirm the app was working properly.

Upon exploring the files App.js, EmployeesList.js, and EmployeeCard.js, I observed that theme data (light/dark) was being passed down as props through multiple levels, which led to prop-drilling.

To eliminate this, I created a new file called ThemeContext.js where I used createContext() to define and export a new context called ThemeContext with a default value of 'light'.

Next, I updated App.js to:

* Import the ThemeContext.
* Wrap its JSX with ThemeContext.Provider.
* Provide the theme value dynamically from the state.
* Remove the prop-based theme passing to EmployeesList.

Then I updated EmployeesList.js to stop passing the theme as a prop to EmployeeCard.

Finally, I went to EmployeeCard.js and:

* Imported ThemeContext.
* Retrieved the context value using useContext(ThemeContext).
* Used the theme value to set the className for the button dynamically.

This improved the structure and maintainability of the application by removing unnecessary prop chains and following best practices with **React Context API**.

**✅ Answers to Objectives**

1. **Explain the need and Benefits of React Context API**
   * It helps **avoid prop drilling**, especially when data needs to be accessed by deeply nested components.
   * Encourages cleaner and more maintainable code.
   * Improves component reusability and reduces coupling.
2. **Working with createContext()**
   * createContext() is used to **create a new Context object** in React.
   * Syntax:

js

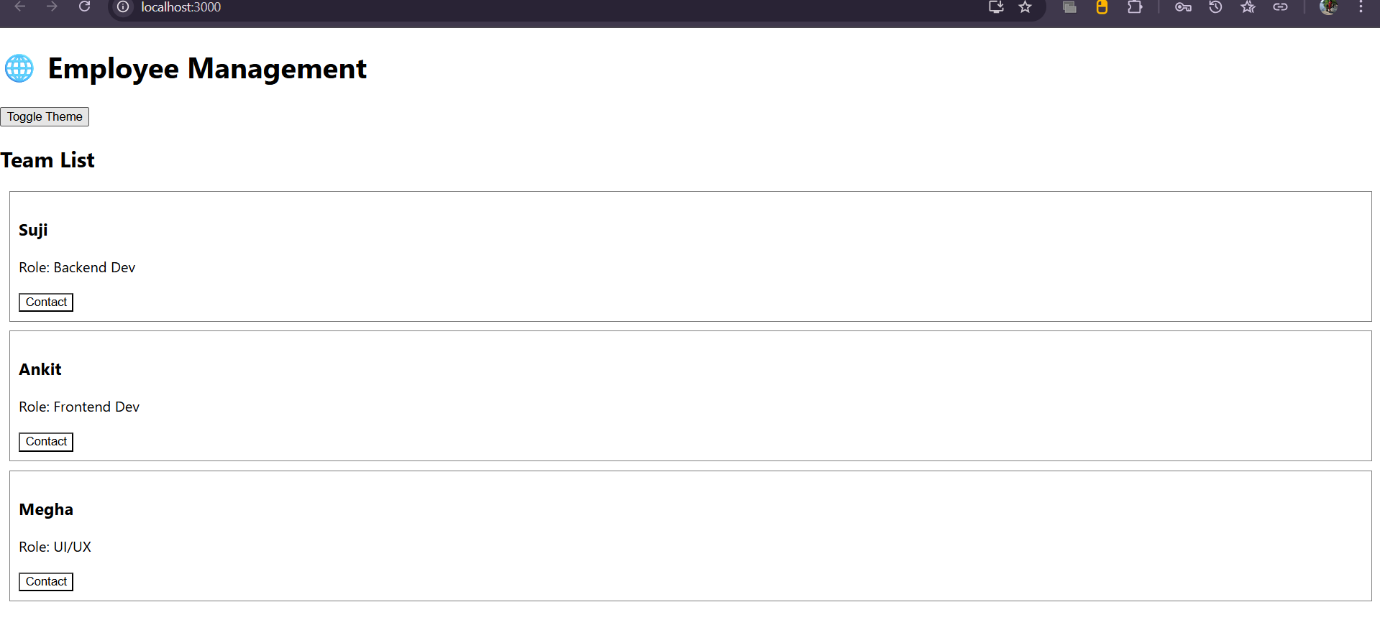
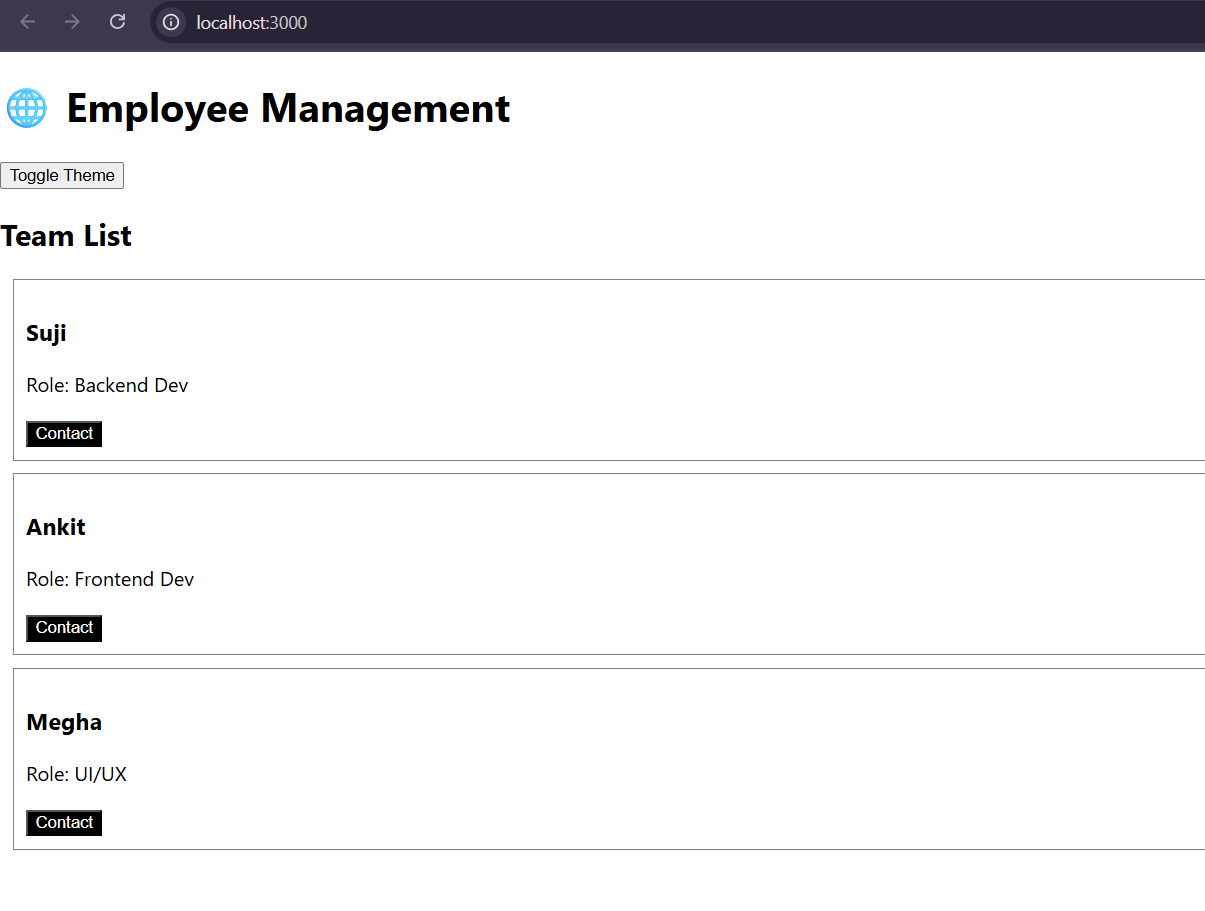
CopyEdit

const ThemeContext = createContext('light');

* + It returns an object with two components: **Provider** and **Consumer**.
  + The value passed to Provider becomes available to any child component using useContext().

1. **List the types of Router Components**
   * BrowserRouter: Uses the HTML5 history API (for clean URLs).
   * HashRouter: Uses the hash portion of the URL (/#/) for routing (e.g., for static file hosting).
   * MemoryRouter: Keeps the history in memory (used mostly for testing).
   * Routes: Wrapper for Route components.
   * Route: Renders a component based on the path.
   * Link and NavLink: Used for navigation without full page reloads.
   * Outlet: Used for nested routing.

Output:



**15. Documentation – “TicketRaisingApp”**

In this hands-on lab, I created a simple complaint registration form using **React forms**. The project is named **“ticketraisingapp”** and it contains a single component called ComplaintRegister.

**Objectives Covered**

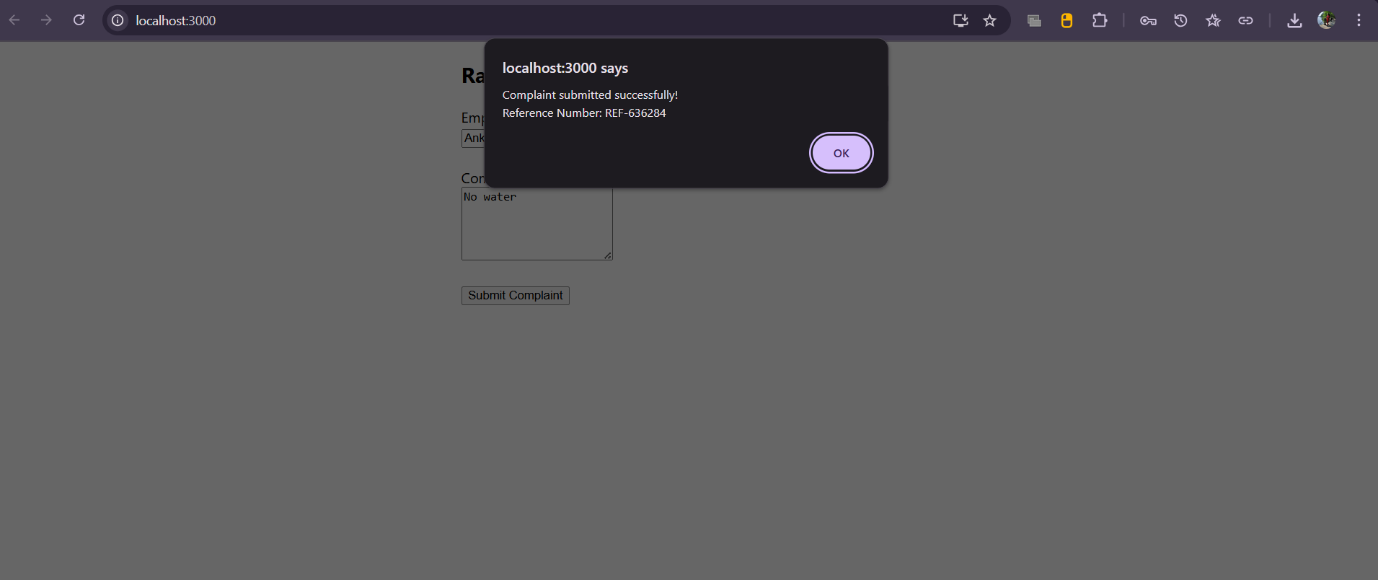
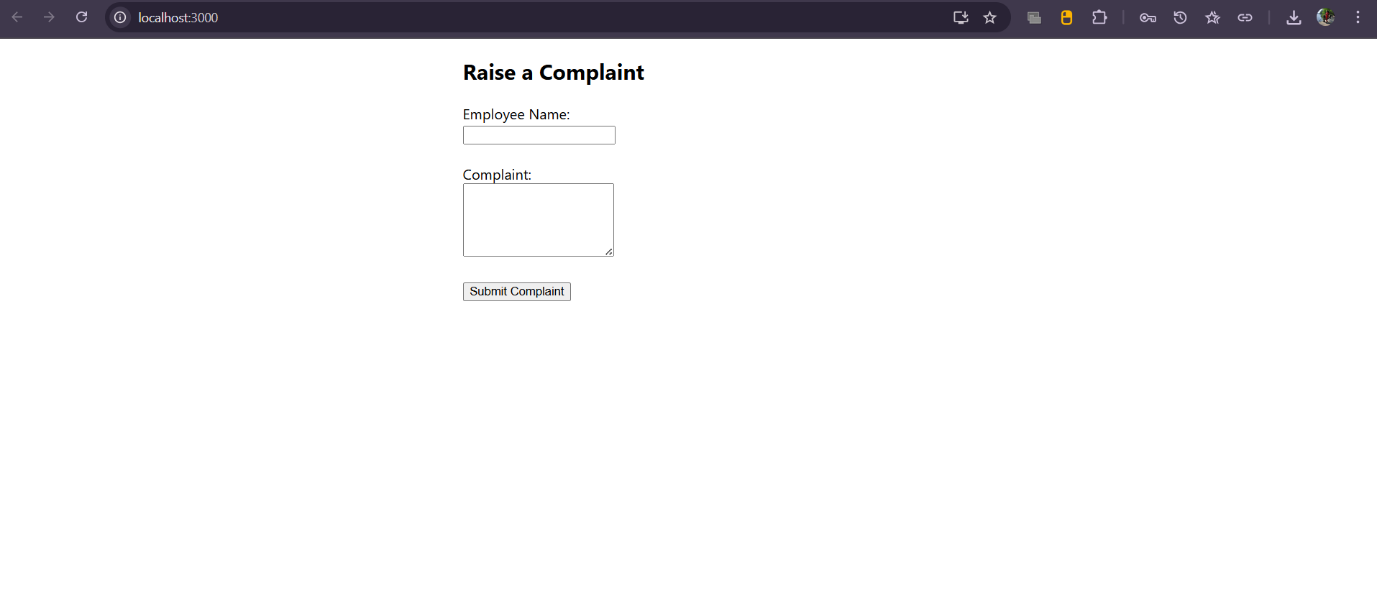
1. **Explain about React Forms**  
   I implemented a form using the standard <form> element and managed its behavior using state hooks (useState). The form captured and handled user input dynamically.
2. **Define Controlled Components**  
   I created **controlled components** by binding form inputs (textbox and textarea) to React state variables. This allowed the component to control the values of the form fields.
3. **Explain Various Input Controls**  
   The app uses two types of input controls:
   * A **textbox** for entering the employee name
   * A **textarea** for writing the complaint description
4. **Explain Handling Forms**  
   I used the onChange event to update the respective state variables as the user typed in the fields. The form’s behavior was fully managed using React’s event handlers.
5. **Explain Submitting Forms**  
   I used the handleSubmit function, which is triggered on clicking the **Submit** button. On submission, the form:
   * Prevents default page reload using e.preventDefault()
   * Generates a random **Reference Number** using Math.random()
   * Displays the entered data and reference number in an alert() box

**Tools Used (Prerequisites)**

* **Node.js**
* **NPM**
* **Visual Studio Code**

**App Details**

* **Project Name:** ticketraisingapp
* **Component Created:** ComplaintRegister.js
* **Features:**
  + Controlled form with employee name and complaint fields
  + On submission, shows alert with generated complaint reference number



**16. Documentation: React Form Validation – mailregisterapp**

**Objectives Covered**

1. **Explain React Forms validation**  
   → In React, form validation ensures that input fields are filled correctly before the form can be submitted. It’s handled either in real-time or during form submission using event handlers like onChange and onSubmit.
2. **Identify the differences between React Form and HTML Form**  
   → HTML forms rely on browser defaults and manual DOM manipulation. In React, forms are built with **controlled components** that are synced with state, giving full control over inputs and validation.
3. **Explain about controlled components**  
   → In this lab, each input (name, email, password) is tied to a state variable. As the user types, the state updates, making it a controlled component.
4. **Identify various React Form input controls**  
   → I’ve used:
   * input for **name**, **email**, and **password**
   * button to **submit** the form
5. **Explain how to handle React Forms**  
   → Form is handled using:
   * onChange event to track user input
   * onSubmit event to validate fields and either allow submission or show errors
6. **Explain about submitting forms in React**  
   → Submission is managed through the handleSubmit function which checks:
   * Name is ≥ 5 characters
   * Email includes @ and .
   * Password is ≥ 8 characters  
     If any validation fails, error messages are shown and form is not submitted.

**🛠 What I Did (Steps)**

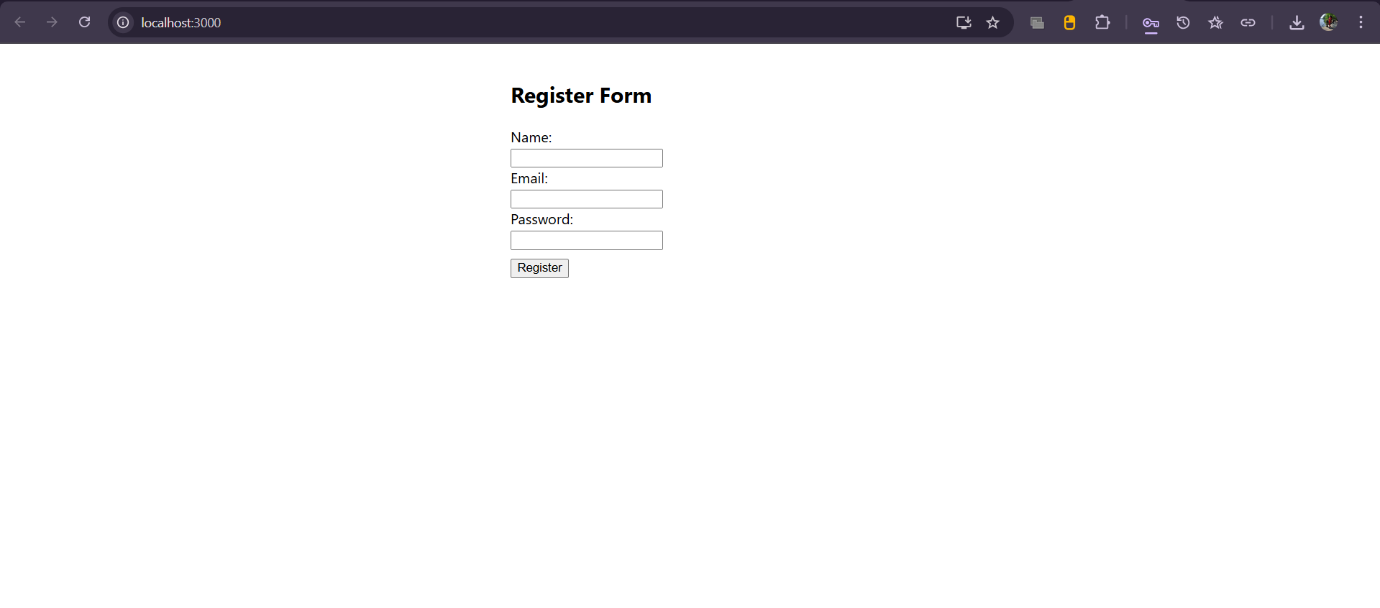
1. Created a React app:

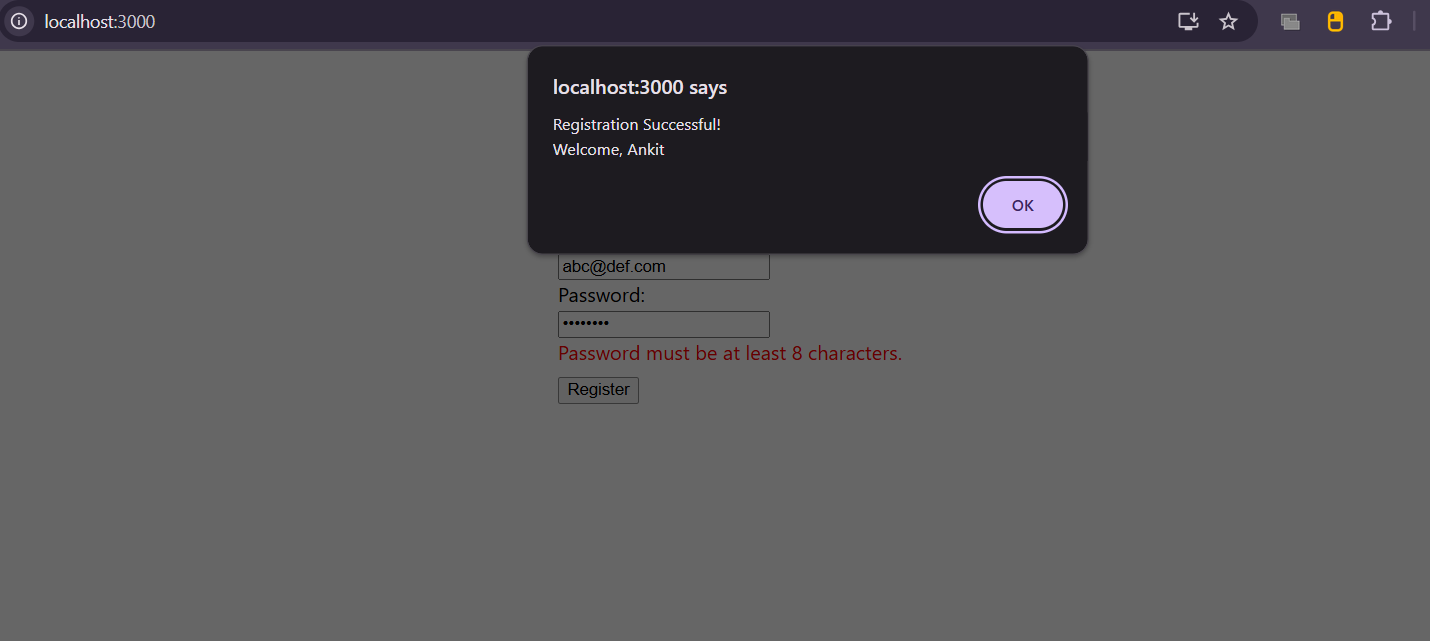
npx create-react-app mailregisterapp

cd mailregisterapp

npm start

1. Inside src/, created a file register.js.
2. Built a form component with three fields:
   * Name
   * Email
   * Password
3. Added validation logic using state and functions.
4. Connected the onChange event for real-time tracking and onSubmit for validation before submission.
5. Displayed alert for success or error messages near each field.
6. Imported and used <Register /> component inside App.js.

Output: 



17. **Documentation for React Hands-on Lab – fetchuserapp**

**✅ Objectives**

* Explain how to consume REST APIs from React applications.

**What I Did**

In this hands-on lab, I built a React application named **fetchuserapp**.  
The goal was to **fetch user data from a public API** and display specific fields.

Here's the breakdown of what I did:

1. Created a new React app using npx create-react-app fetchuserapp.
2. Inside src, created a component named **Getuser.js**.
3. Used componentDidMount() (in a class component) to:
   * **Make a GET request** to https://api.randomuser.me/ using fetch().
   * Parsed the JSON response.
   * Retrieved **title**, **first name**, and **user image**.
4. Stored this data in the **component's state** and used render() to display it.
5. Displayed the fetched user details in a clean format (name + image).
6. Ensured everything ran smoothly without any external libraries other than React.

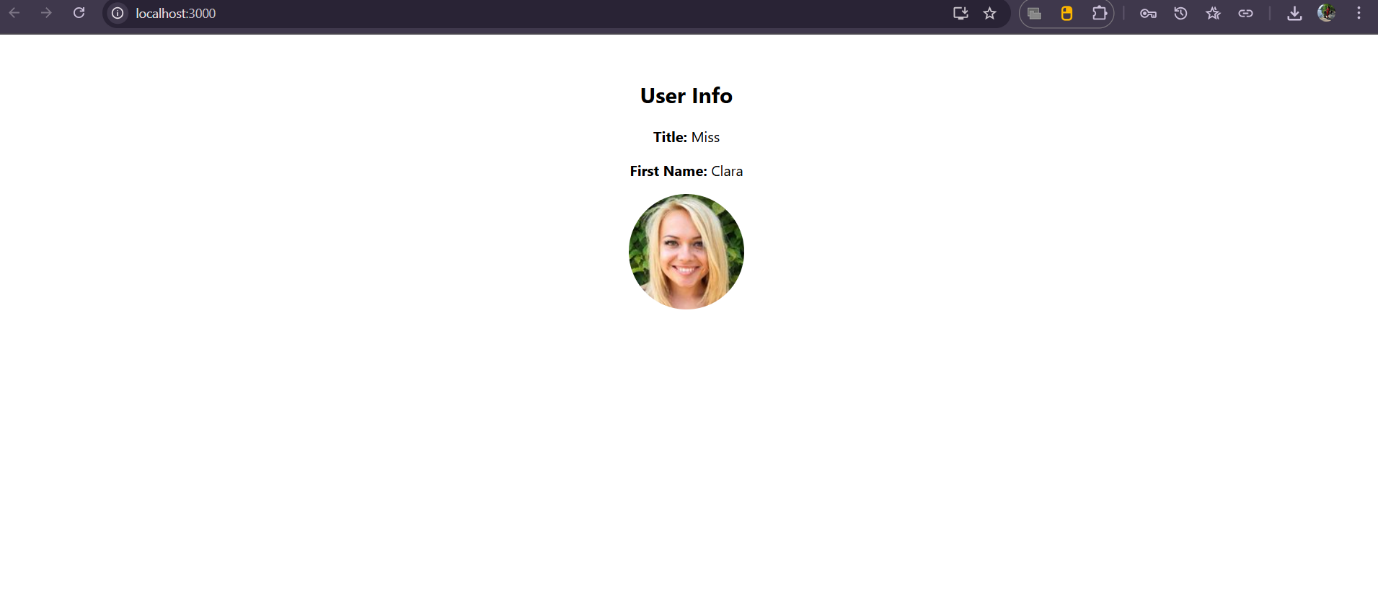
**📌 What I Learned**

* How to **fetch and consume REST APIs** in React.
* How to use the **componentDidMount()** lifecycle method to make API calls.
* How to **update state based on fetched data**.
* Render dynamic content fetched from an external API in a React component.

**⚙️ Prerequisites**

To complete this lab, I had installed:

* Node.js
* NPM
* Visual Studio Code

**Output:**

