**WEEK-07 HANDSON SOLUTIONS**

**EXERCISE-01**

**9. ReactJS-HOL**

**IMPLEMENTATION:**

This hands-on lab focused on the practical application and demonstration of key ECMAScript 2015 (ES6) features within a modular ReactJS application named "cricketapp".

The primary objective was to leverage modern JavaScript constructs such as array map() and filter() methods combined with arrow functions, array destructuring, and the spread syntax for array manipulation.

Additionally, the lab emphasized fundamental React concepts like component composition and conditional rendering to dynamically display different application states.

The lab successfully addressed several technical and conceptual objectives related to ES6 and React development.

This included demonstrating the practical use of ES6 let and const for variable declaration, applying arrow functions for concise function syntax, and employing the array map() method for transforming array elements into React components for rendering lists.

The array filter() method was utilized to create new arrays based on a specified condition, showcasing data subsetting capabilities.

ES6 destructuring was applied for extracting specific elements from arrays in a structured manner.

Furthermore, the ES6 spread syntax was used to concatenate arrays efficiently, addressing the requirement for array merging.

Finally, the lab involved component composition and conditional rendering, achieved through the strategic rendering of different components based on a boolean flag, to present a dynamic UI.

The "cricketapp" utilized a standard React project structure.

The ListofPlayers component was designed to render a comprehensive list of cricket players. An array containing 11 player objects, each with name and score properties, was initialized in the parent App.js component and passed as a players prop.

Within ListofPlayers.js, the map() method was invoked on the players prop to iterate over each player object, returning a <li> element for each, displaying Mr. {item.name} and <span>{item.score}</span>.

A dedicated Scorebelow70 component was created to handle the filtering requirement.

This component received the players array prop and applied the filter() method (players.filter(item => item.score <= 70)) to generate a new array containing only players whose scores were 70 or less.

The predicate function for filter was defined using an ES6 arrow function. The IndianPlayers section of the lab involved a set of components demonstrating advanced array manipulation.

The OddPlayers and EvenPlayers components were created to display selected players from a team array using ES6 array destructuring.

Both accepted a team array prop, with OddPlayers leveraging const [first, , third, , fifth] = team; to extract specific elements and EvenPlayers similarly using const [, second, , fourth, , sixth] = team;.

To illustrate array merging, two arrays, T20Players and RanjiTrophyPlayers, were declared in App.js.

The ES6 spread syntax (...) was then employed to non-mutatively merge these into a single mergedIndianPlayers array, which was subsequently displayed by the ListofIndianPlayers component.

The App.js component served as the main application entry point and controlled the display logic.

A boolean flag variable was used to govern which set of components was rendered.

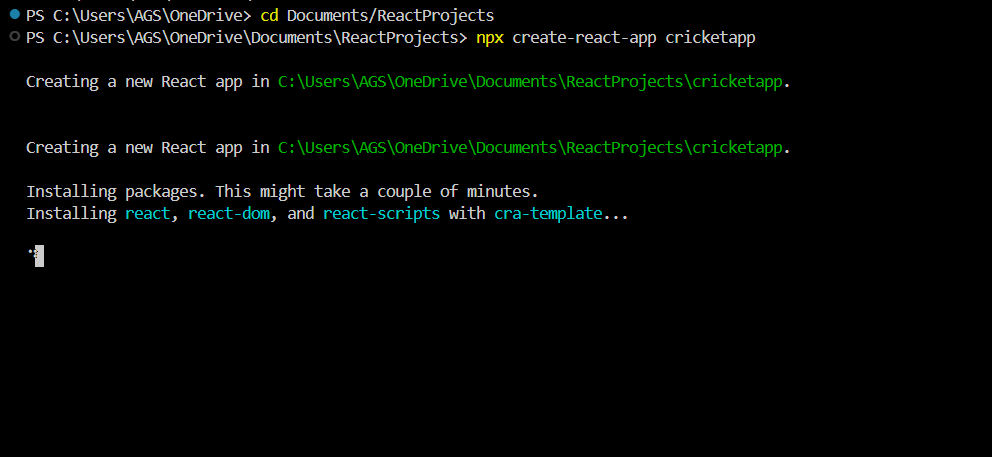
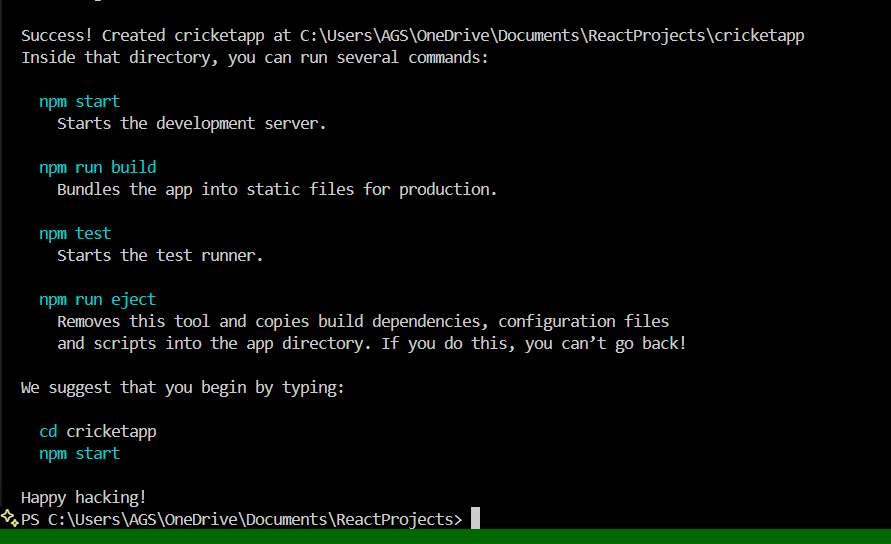
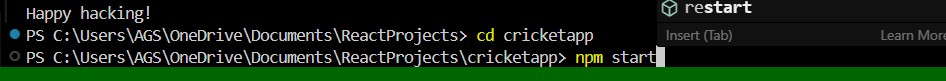
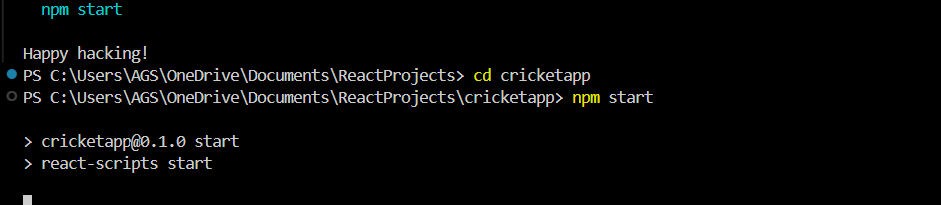
A ternary operator flag === true ? ( ... ) : ( ... ) provided the conditional logic:

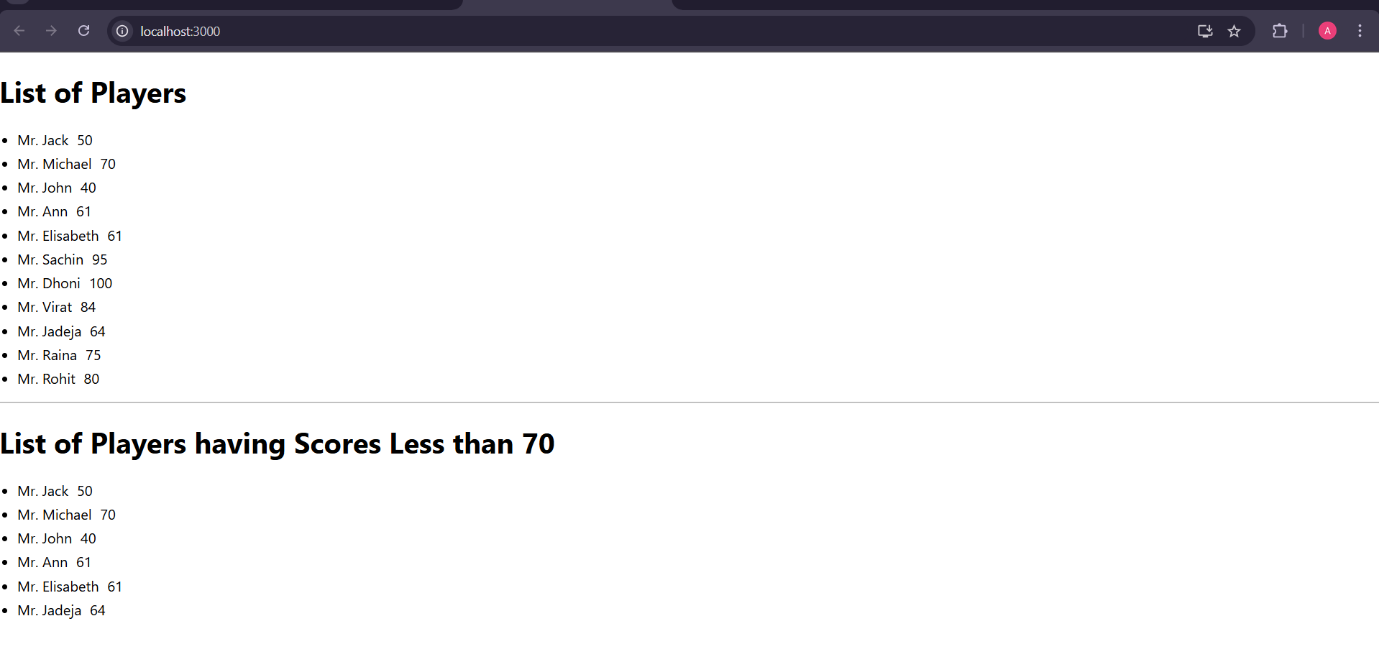
When flag was true, ListofPlayers and Scorebelow70 components were rendered;

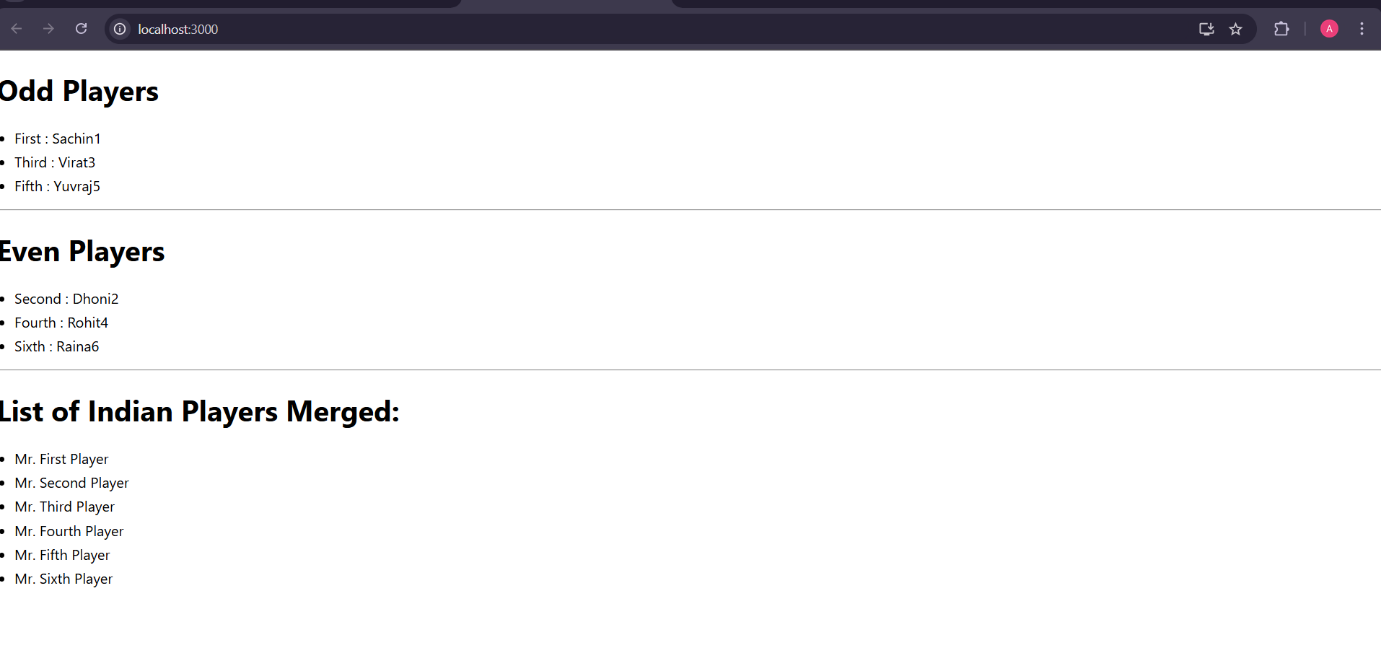
When flag was False, OddPlayers, EvenPlayers, and ListofIndianPlayers components were rendered.

This comprehensive conditional rendering aligned perfectly with the lab's requirements and the provided hint structure.

The implementation successfully generated the exact outputs as depicted below.

****

**OUTPUT:** ****

****

**EXERCISE-02**

**10-REACTJS-HOL**

This hands-on lab focused on mastering the fundamental aspects of React JSX within the context of an "officespacerentalapp" application.

The core objective was to demonstrate how JSX facilitates the creation of UI elements, manages attributes, incorporates JavaScript expressions, applies inline CSS, and renders dynamic lists to the Document Object Model (DOM).

The lab also outlined key objectives such as defining JSX , explaining ECMA Script , React.createElement() , how to create React nodes with JSX , how to render JSX to DOM , how to use JavaScript expressions in JSX , and how to use inline CSS in JSX.

The implementation began by setting up the basic page structure within the App.js functional component. This involved creating an

<h1> element to display the page heading, which dynamically incorporated a JavaScript variable using JSX expressions.

For visual appeal, an <img> tag was used to display an office space image, with its src attribute populated by a JavaScript variable holding the image URL. The width and height attributes were set to "25%" as specified in the hint , demonstrating attribute management within JSX. Further inline CSS was applied to the image for improved responsiveness and centering, showcasing direct inline styling in JSX.

Subsequently, the lab moved to displaying detailed information for a single office, including conditional styling for the rent. A JavaScript object (

officeDetails) was created to store properties like Name, Rent, and Address. These properties were then accessed and displayed within

<h1> and <h3> tags using JavaScript expressions in JSX. Critically, to fulfill the requirement of dynamic rent color, an inline style object (

rentColorStyle) was conditionally generated. A ternary operator was used within this style object:

color: officeDetails.Rent <= 60000 ? 'red' : 'green', which was directly applied to the <h3> tag displaying the rent, ensuring the text appeared red if the rent was below or equal to Rs. 60,000, and green otherwise.

The final phase extended the application to display multiple office spaces by iterating through a list of objects. The single

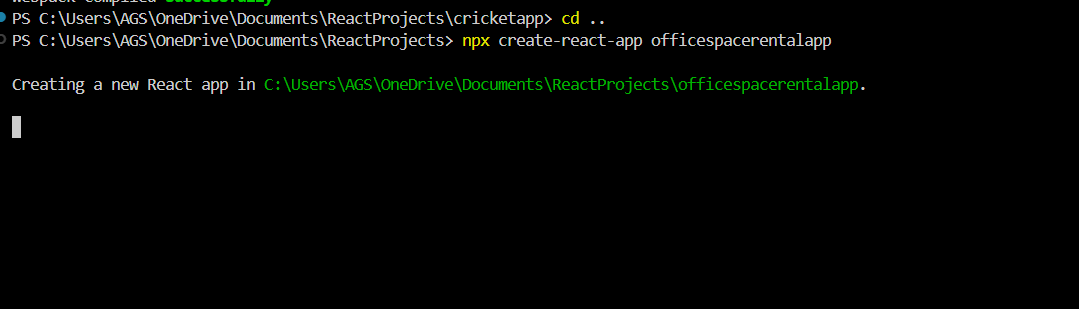
officeDetails object was replaced by an array named officeSpaces, containing several office objects, each assigned a unique id for React's key prop optimization during list rendering. The

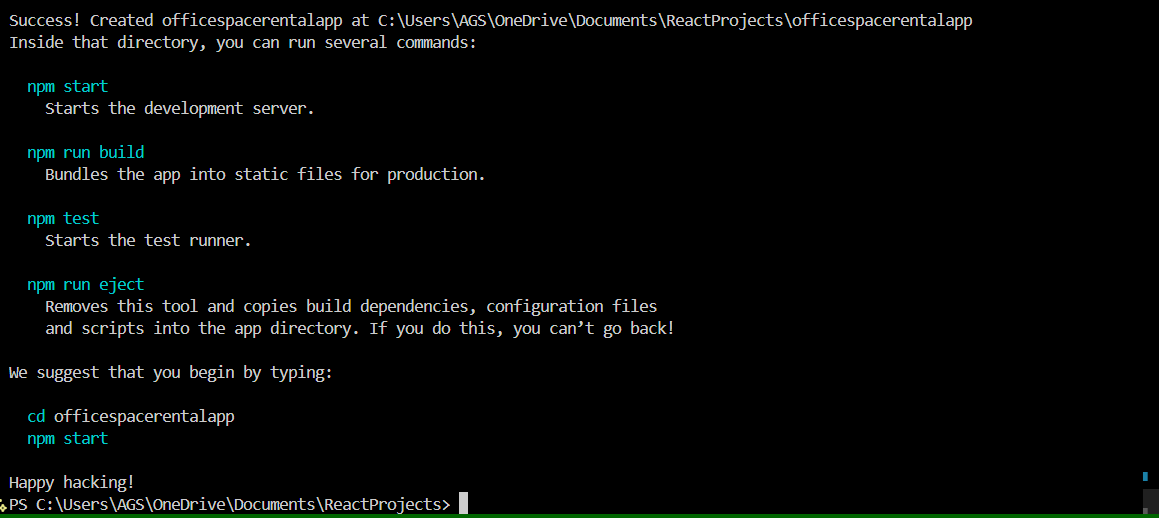
map() method was then used on the officeSpaces array to iterate over each office entry. For every

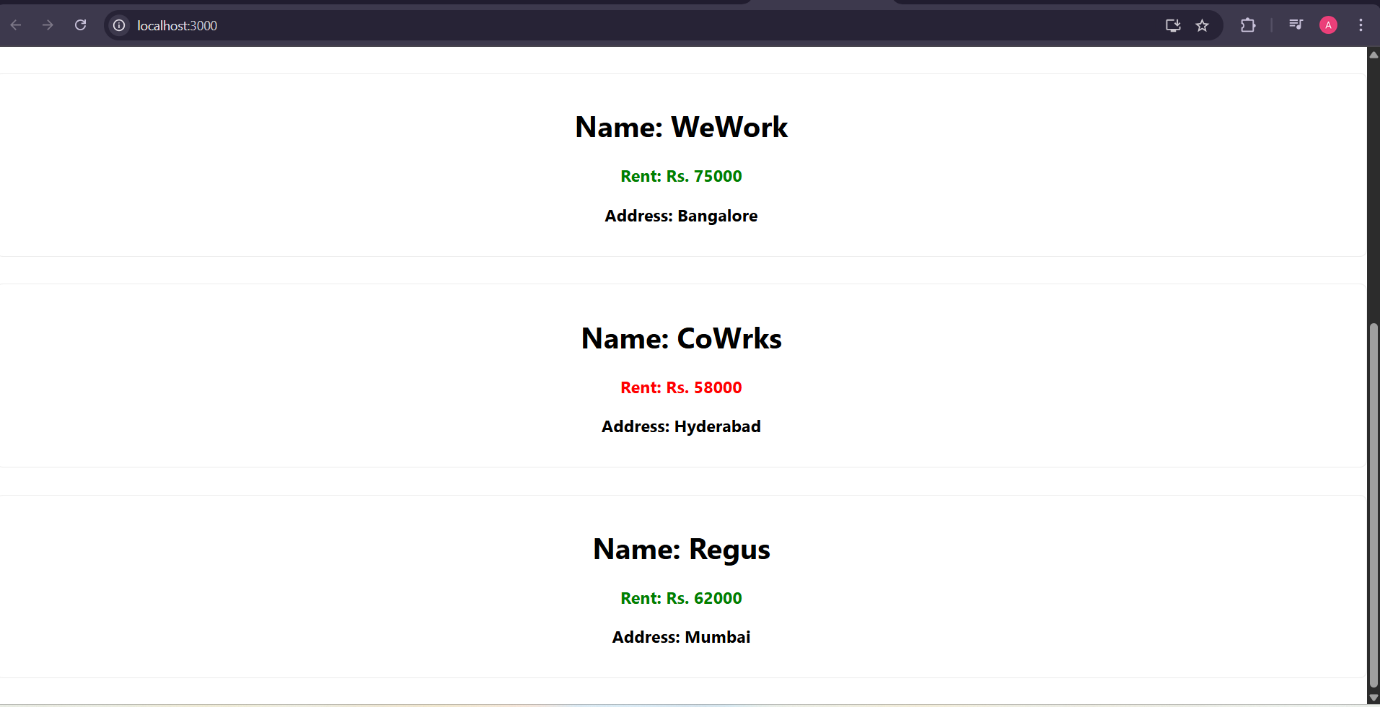
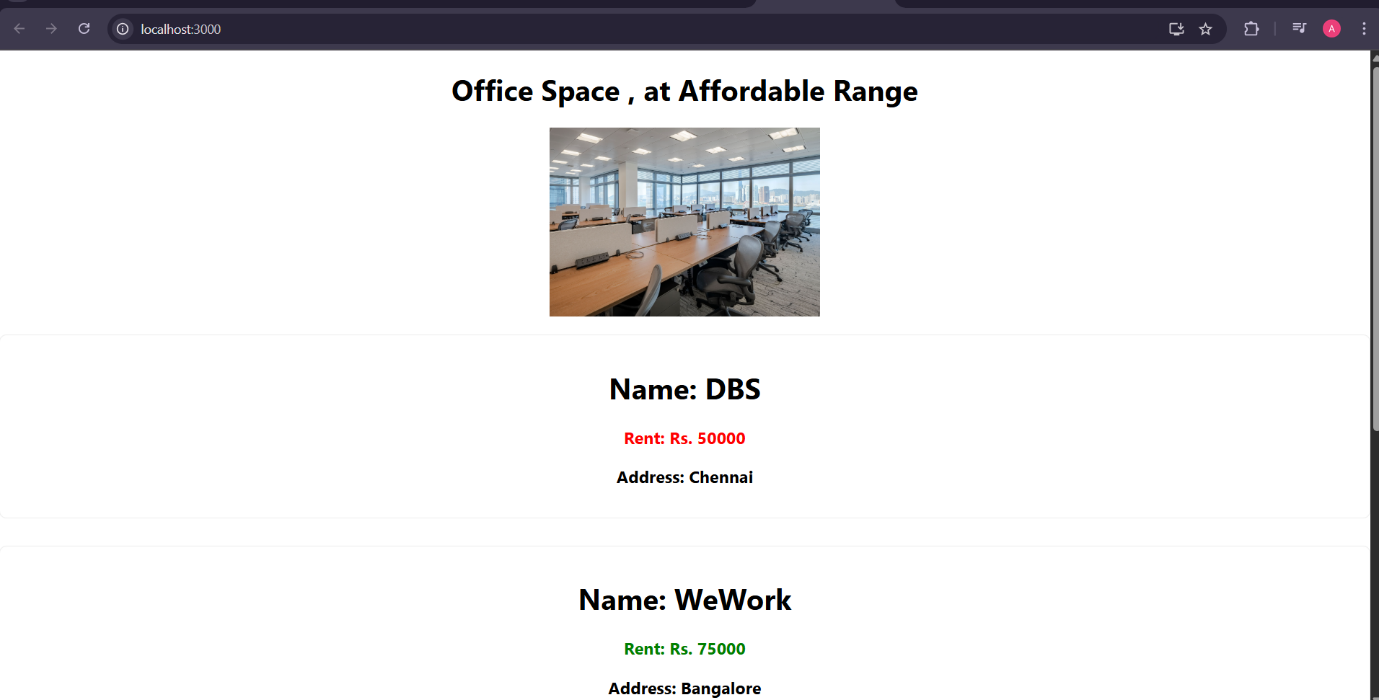
office object, a div element was returned, encapsulating the <h1> for Name, <h3> for Rent, and <h3> for Address. The conditional rentColorStyle logic was implemented *inside* this map() callback function, ensuring that the rent color was evaluated and applied individually for each office space based on its specific rent value.

The implementation successfully rendered the application's UI, precisely matching the specified output. This included the main heading "Office Space , at Affordable Range," the presence of the office image with its attributes, and a dynamically generated list of office details. Each office entry correctly displayed its Name, Rent, and Address, with the rent values accurately colored red for rents equal to or below Rs. 60,000, and green for rents above Rs. 60,000.

The Output is achieved successfully as below.







**EXERCISE-03**

**11-REACTJS-HOL**

**IMPLEMENTATION:**

In this hands-on lab, a ReactJS application named **eventexamplesapp** was developed to explore the implementation of event handling using functional components and hooks.

The lab focused on various event-driven functionalities such as button click handling, function invocation with arguments, synthetic events, and form submissions.

The goal was to gain practical experience with React’s declarative event system and state management through the use of the useState hook.

The central component, EventExamples, was designed to encapsulate all the required features.

A simple counter was implemented with **Increment** and **Decrement** buttons to manage stateful count values.

The increment button was configured to trigger multiple methods simultaneously: increasing the counter and displaying a static alert message saying, “Hello! Member.” This illustrated how multiple functions can be chained in a single event handler in React.

To demonstrate argument passing to event handlers, a button labeled **“Say Welcome”** was introduced. This button triggered a function that accepted a custom string argument (“Welcome”) and displayed it via an alert.

This helped reinforce how React allows flexible event handler binding using arrow functions.

Additionally, the lab included a button to showcase React’s **SyntheticEvent** system.

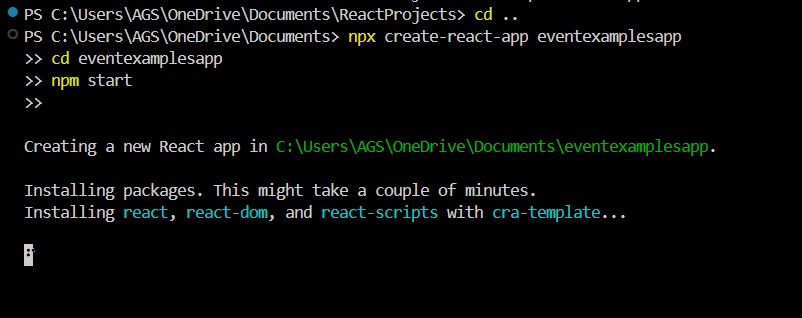
This **“OnPress”** button displayed an alert saying “I was clicked” and utilized e.preventDefault() to demonstrate how event objects are normalized across browsers in React, ensuring consistency and compatibility.

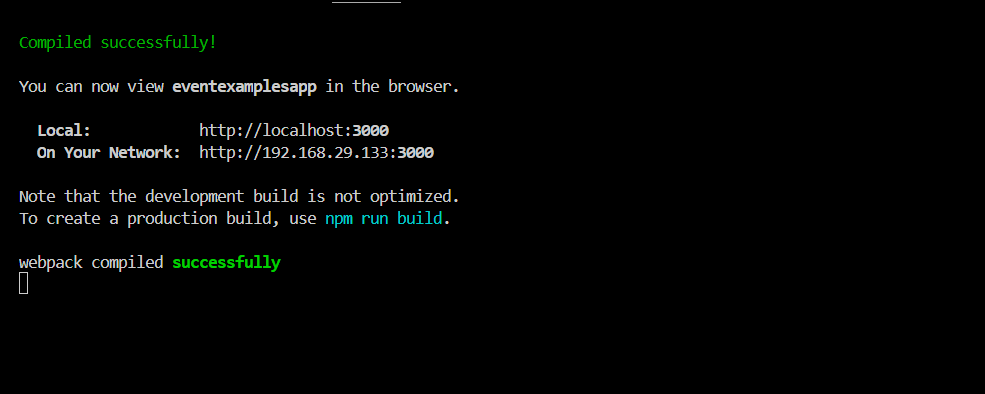
A functional **Currency Converter** component was also created. This feature enabled users to input a value in Indian Rupees and convert it to Euros upon clicking the **Convert** button. The conversion was handled through form submission, with appropriate state updates and dynamic result display using a basic conversion logic. This exercise highlighted the use of controlled components and form event handling in React.

Styling for the application was handled through an external CSS file EventExamples.css, with a container class applied to left-align the layout and add spacing between elements. This provided a clean and readable user interface. Care was taken to ensure that buttons and inputs did not overlap by applying basic spacing rules using margin and padding.

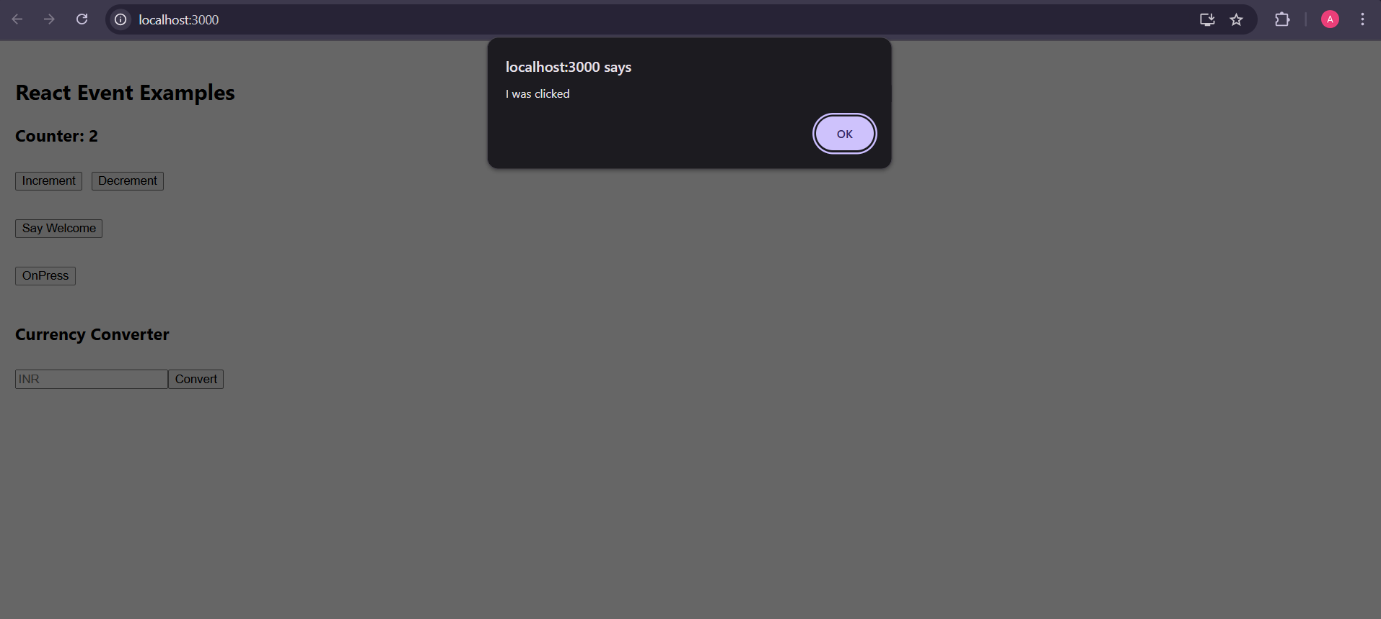
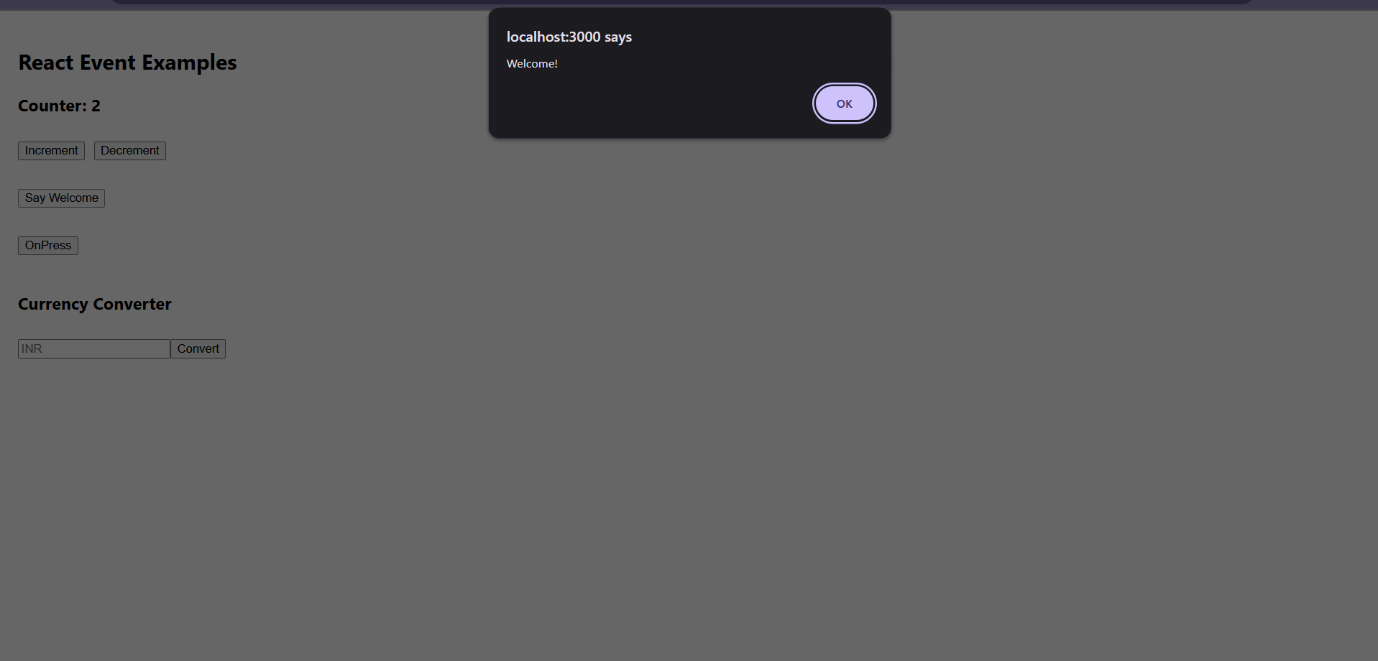
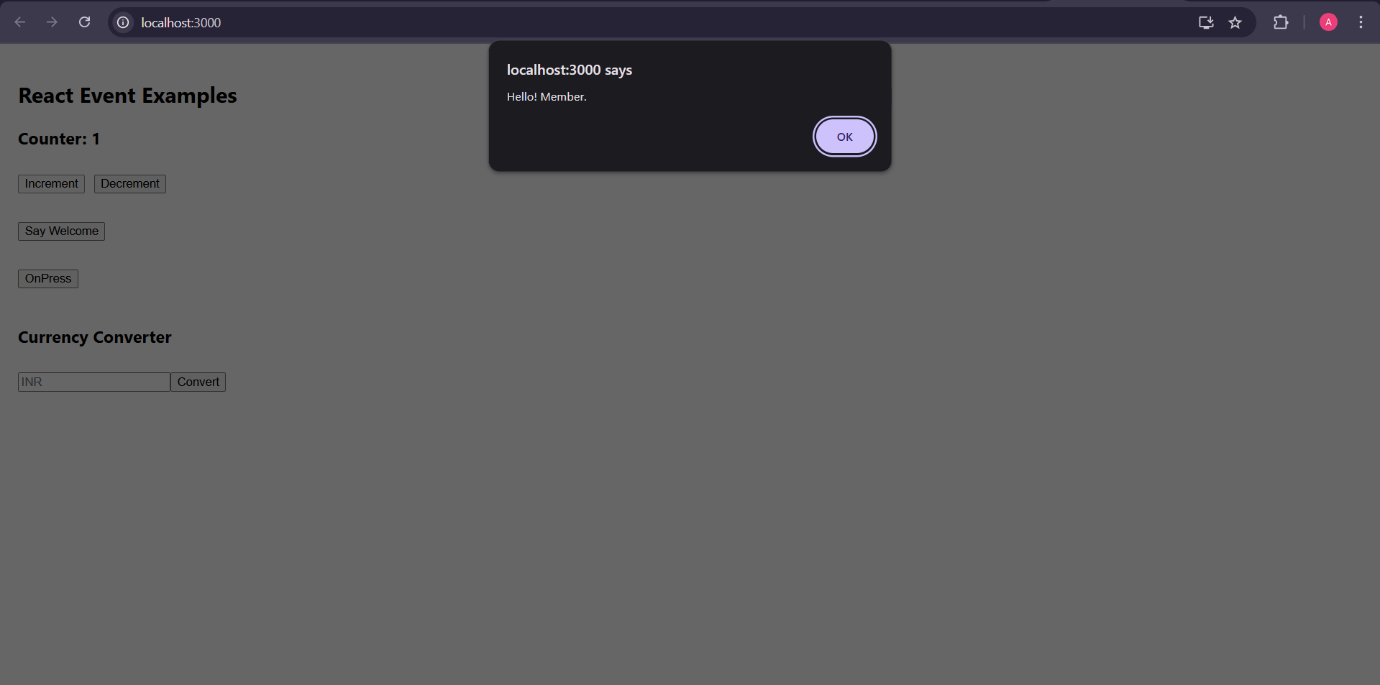
In conclusion, the hands-on lab successfully demonstrated practical usage of React events, including custom function binding, synthetic event handling, and form interaction. It provided a strong foundation in how React abstracts DOM events and manages state-driven UI updates in a declarative manner. The exercise not only fulfilled the learning objectives but also reinforced best practices for building interactive components in React applications.

The expected Output was achieved successfully.



****

**OUTPUT:**



**EXERCISE-04**

**12-REACTJS-HOL**

**IMPLEMENTATION:**

In this hands-on lab, the objective was to understand and implement **conditional rendering** in ReactJS.

The task was to build a basic ticket booking application named ticketbookingapp that conditionally displays content to users based on their login status.

Through this exercise, the concepts of element variables, conditional rendering logic, and preventing unnecessary component rendering were effectively demonstrated.

The application was created using create-react-app, and a central component App.js was used to manage login state using React's useState hook. Two separate components were developed: GuestPage and UserPage.

The GuestPage component displays a simple list of available flights, which any unauthenticated user (guest) can browse.

In contrast, the UserPage component presents a logged-in user interface that allows users to book flight tickets. This clearly separates the user experience based on login state.

The conditional rendering was handled using an **element variable**, pageContent, which was assigned different component JSX (<GuestPage /> or <UserPage />) depending on the login state (isLoggedIn).

This approach ensures that only the relevant component is rendered at any time, which also prevents unnecessary rendering of components that are not needed.

Additionally, a login/logout toggle was implemented using buttons that dynamically switch their label and function based on the same login state.

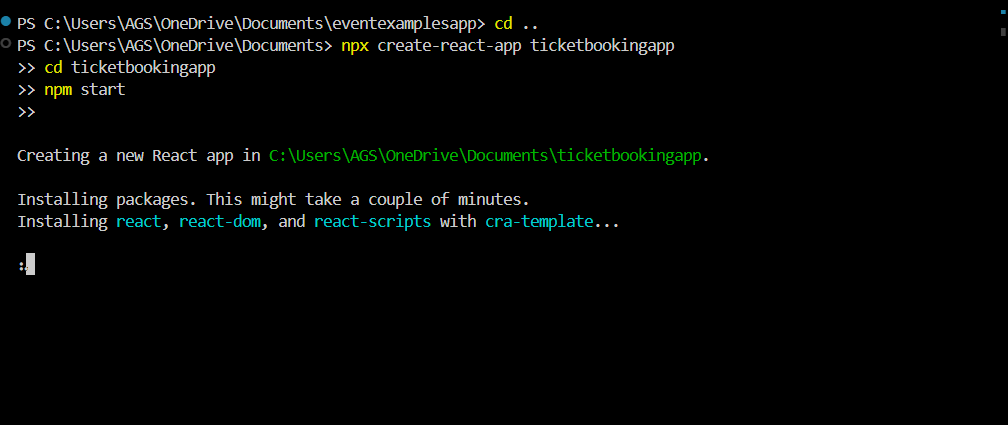
The use of a conditional if block to assign values to an element variable exemplifies one of the recommended patterns for rendering elements conditionally in React.

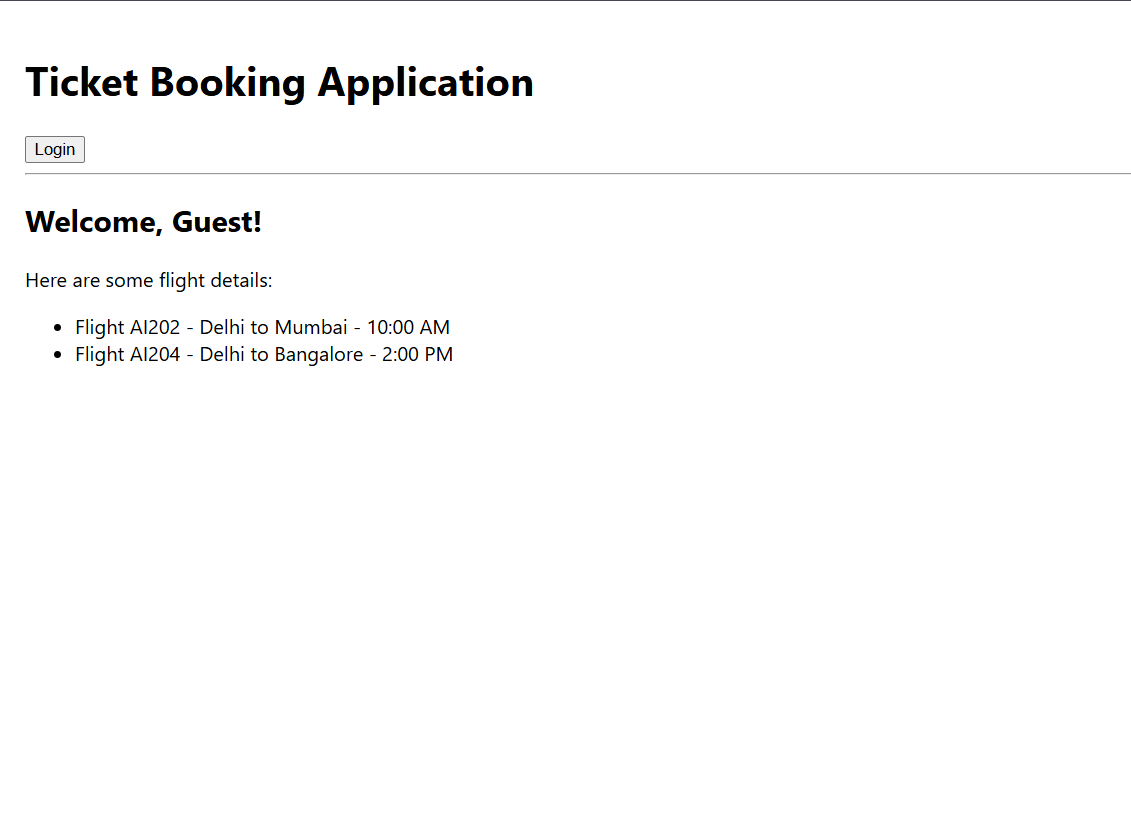
By placing the login logic and component decision-making in the App.js file, the design maintains simplicity while demonstrating practical usage of conditional rendering techniques.

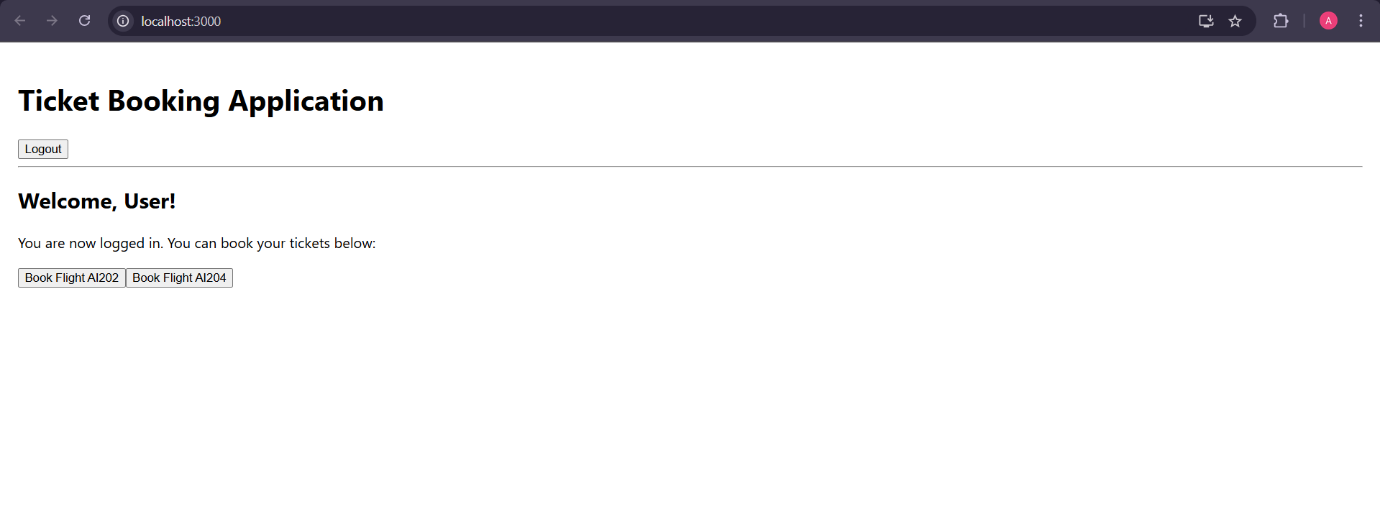
In conclusion, this hands-on lab successfully demonstrated how conditional rendering works in ReactJS, how element variables can be used to control what gets displayed, and how to prevent non-required components from being rendered.

The lab reinforced core React principles such as state management, declarative UI design, and clean separation of component responsibilities.

**OUTPUT:**

****

****

****

**EXERCISE-05**

**13-REACTJS-HOL**

**IMPLEMENTATION:**

In this hands-on lab, the objective was to explore and implement various techniques of **conditional rendering** in ReactJS.

The task involved building a React application named bloggerapp, which comprises three main components:

**Book Details**, **Blog Details**, and **Course Details**.

The application demonstrates the use of different approaches for conditional rendering, mapping lists using the map() function, assigning unique keys, and extracting reusable components — all arranged in a structured side-by-side layout for enhanced user experience.

The application was developed using create-react-app, and each of the three sections was implemented as a standalone React component.

Conditional rendering was handled using several approaches, including the **ternary (? :) operator**, **logical AND (&&) operator**, and **null return pattern**, which are commonly recommended in React for rendering logic-based views.

These patterns were embedded within both the main component (App.js) and individual subcomponents, showcasing their versatility and effectiveness.

The BookDetails component was used to display a list of book titles passed as props.

It utilized the map() function for dynamic list generation and included unique key values for each rendered item, as recommended by React for list rendering.

This demonstrated not only conditional rendering based on list availability but also the proper use of keys to maintain component identity.

The BlogDetails component was conditionally rendered based on a showBlogs state variable.

By using a simple boolean check and returning null when the flag was false, this component illustrated a minimal and clean approach to prevent unnecessary rendering of inactive content.

This also emphasized how performance can be optimized in React applications by avoiding DOM rendering when not required.

The CourseDetails component demonstrated conditional rendering using the logical AND operator.

It only displayed course-related information when the user was logged in, determined by the isLoggedIn state variable.

The course data was represented as an array of objects and rendered dynamically using the map() function, further reinforcing the concept of list rendering with keys in React.

Visually, the three components were organized side by side using Flexbox, and each section was visually separated by a **green vertical line** using simple CSS.

This layout aligned with the desired output structure indicated in the lab reference and created a clean and intuitive user interface.

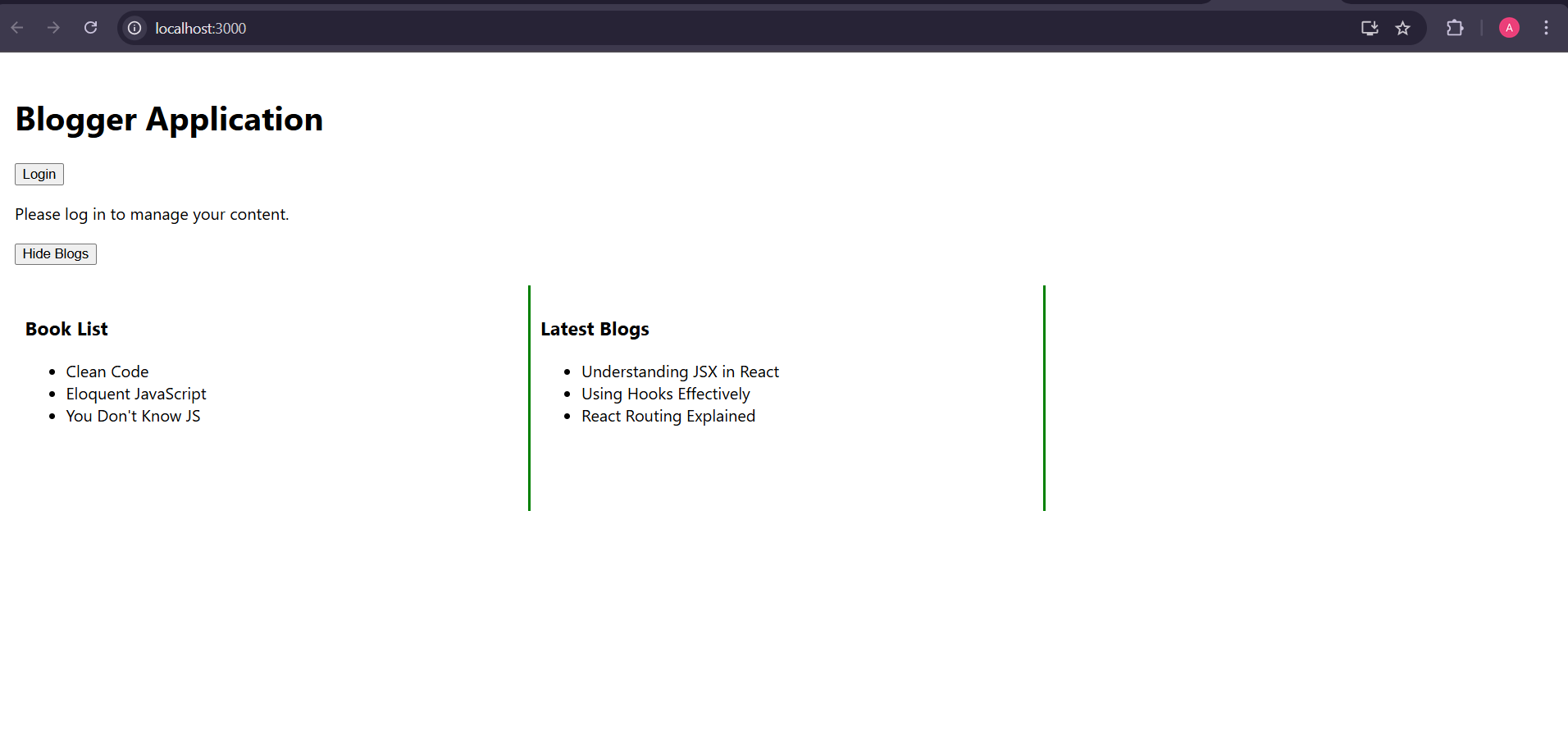
The layout was responsive and readable, and each section remained independently interactive based on its state-controlled visibility.

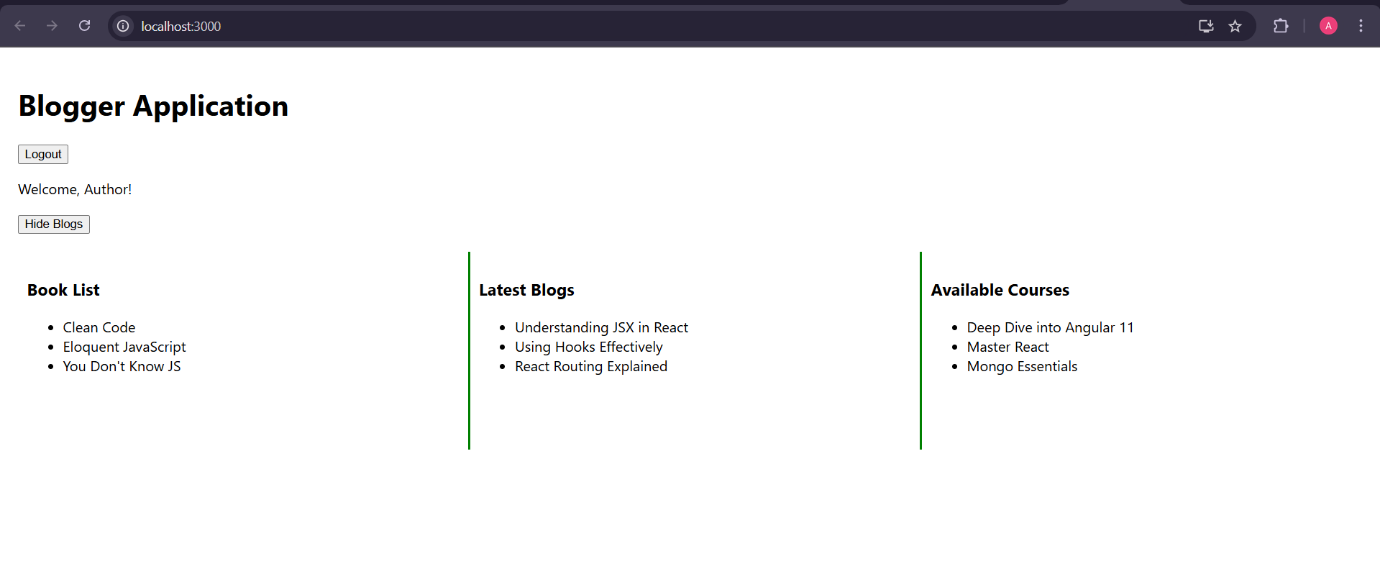
In conclusion, this hands-on lab successfully demonstrated me multiple approaches to conditional rendering in React, the use of dynamic list rendering with map() and keys, and the extraction of logical UI components.

It also reinforced the importance of designing modular components with controlled visibility and state-driven behavior.

The final output not only met the functional requirements but also aligned with the visual design expectations successfully.

**OUTPUT:**

****

****