**ADDITIONAL HANDSON EXERCISES SOLUTIONS-WEEK-06**

***-AAISHA SULTANA GUDURU-6384619***

**EXERCISE-01 06-REACTJS-HOL**

**IMPLEMENTATION:**

The objective of this hands-on exercise was to develop a Single Page Application (SPA) using ReactJS and React Router DOM to manage and display trainer information. The application enables navigation between different views, including a home page, a trainers list page, and individual trainer detail pages. This project demonstrates React’s component-based architecture and client-side routing capabilities.

**Prerequisites**

To complete this lab, the following software and tools were required:  
- Node.js (v14 or above) and npm for package management.  
- Visual Studio Code as the development environment.  
- Basic knowledge of JavaScript, React components, and routing.

**Step 1: Project Setup**

The development began by creating a new React project using the create-react-app tool. In the terminal, the command 'npx create-react-app TrainersApp' was executed, followed by navigating into the project directory with 'cd TrainersApp'. The React Router DOM library was then installed using 'npm install react-router-dom@6'. Finally, the project folder was opened in Visual Studio Code for implementation.

**Step 2: Data Model and Mock Data**

A file named trainer.js was created inside the src folder to define the Trainer class. This class contained the properties: TrainerId, Name, Email, Phone, Technology, and Skills. Another file named TrainersMock.js was created to store an array of mock trainer objects. Each object included sample trainer details matching the defined model. This mock data served as the source for displaying information in the application.

**Step 3: Component Creation**

Three core components were developed:  
**1. Home.js** – Displays a welcome message and serves as the landing page for the application.  
2**. TrainersList.js** – Accepts the mock trainer data as props and renders a list of trainer names. Each trainer name was made clickable using the Link component from React Router DOM to navigate to the respective trainer’s detail page.  
**3. TrainerDetail.js** – Retrieves the id parameter from the URL using the useParams hook, searches for the matching trainer in the mock data, and displays their full details including email, phone, technology, and skills.

**Step 4: Routing Configuration**

The App.js file was modified to set up navigation and routing. The BrowserRouter, Routes, and Route components from React Router DOM were used to define routes:  
- / – Renders the Home component.  
- /trainers – Renders the TrainersList component.  
- /trainer/:id – Renders the TrainerDetail component based on the selected trainer ID.  
Navigation links were added at the top using the Link component for smooth transitions between pages without refreshing the browser.

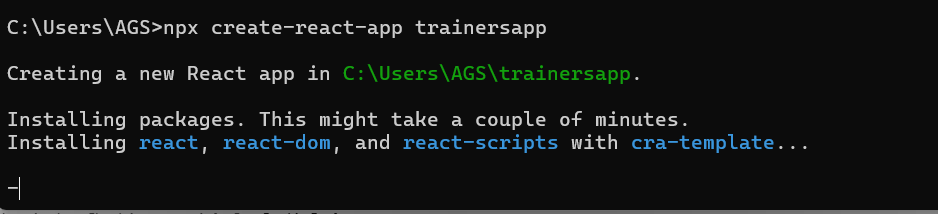
**Step 5: Styling**

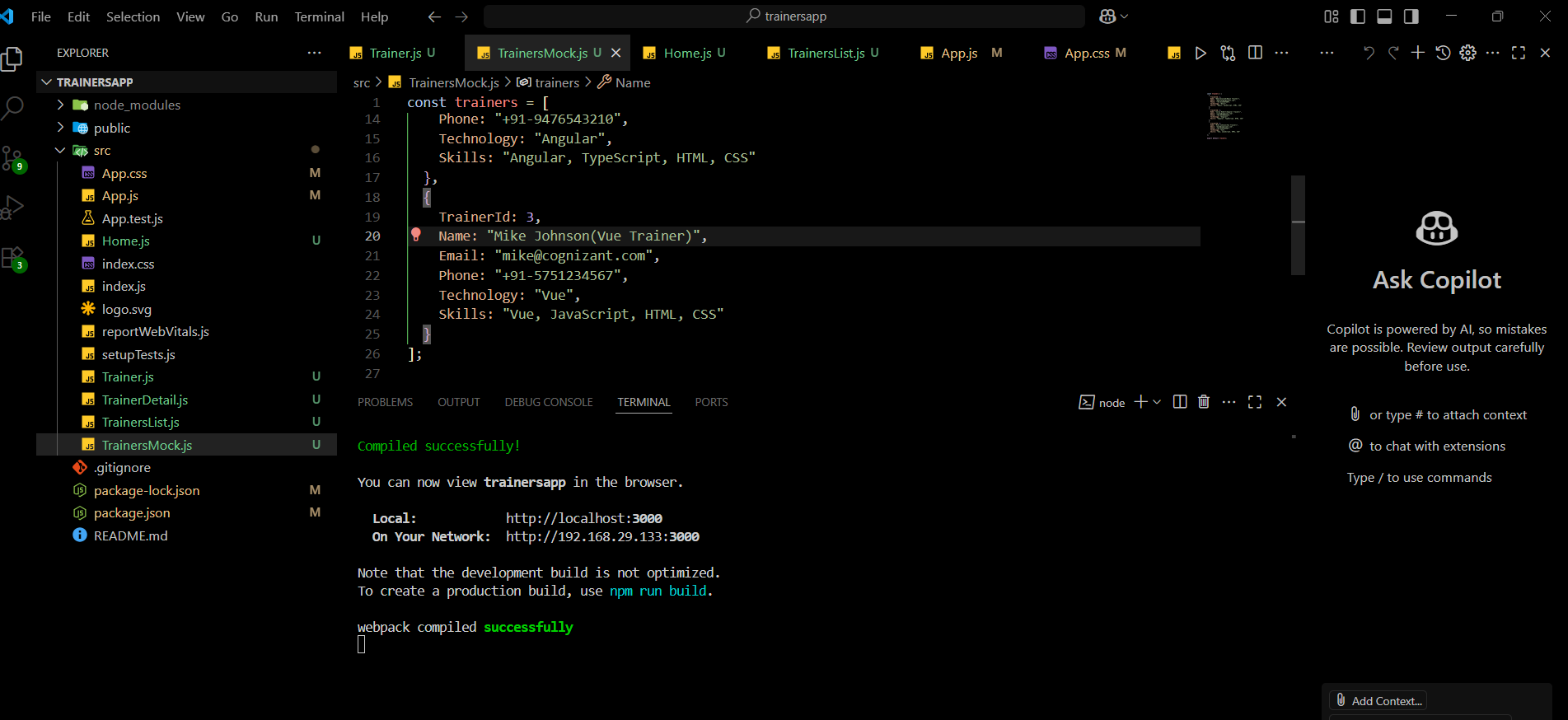
A separate App.css file was created to enhance the visual presentation. The styles included a dark background for the navigation bar with white text links, hover effects, spacing between list items, and consistent margins for the page layout. The CSS file was imported into App.js to apply the styles globally.

**Step 6: Testing the Application**

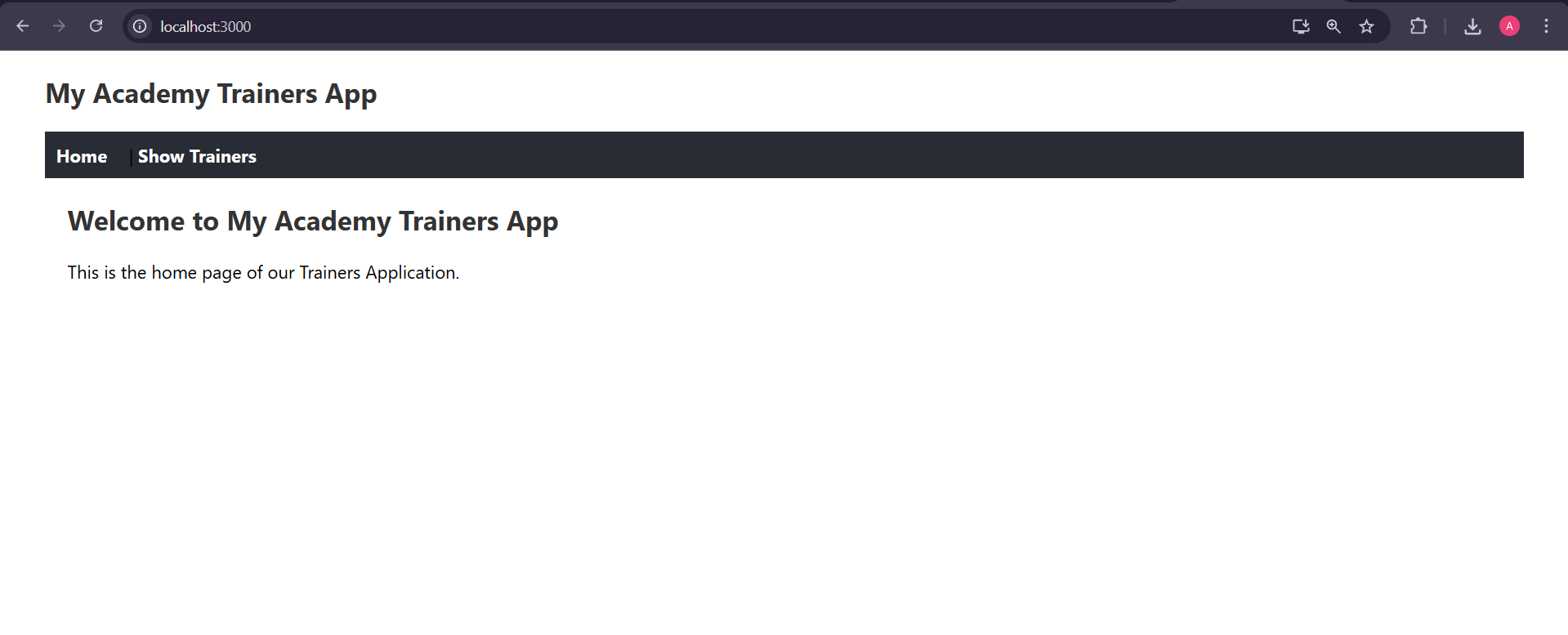
The application was launched using the command 'npm start'. The browser opened automatically at http://localhost:3000. Testing was conducted as follows:  
- On accessing '/', the Home component displayed the welcome message.  
- On navigating to '/trainers', a clickable list of trainers appeared. **-** On clicking any trainer’s name, the '/trainer/:id' route displayed that trainer’s full profile details.  
All navigation was smooth without page reloads, confirming the correct configuration of client-side routing.

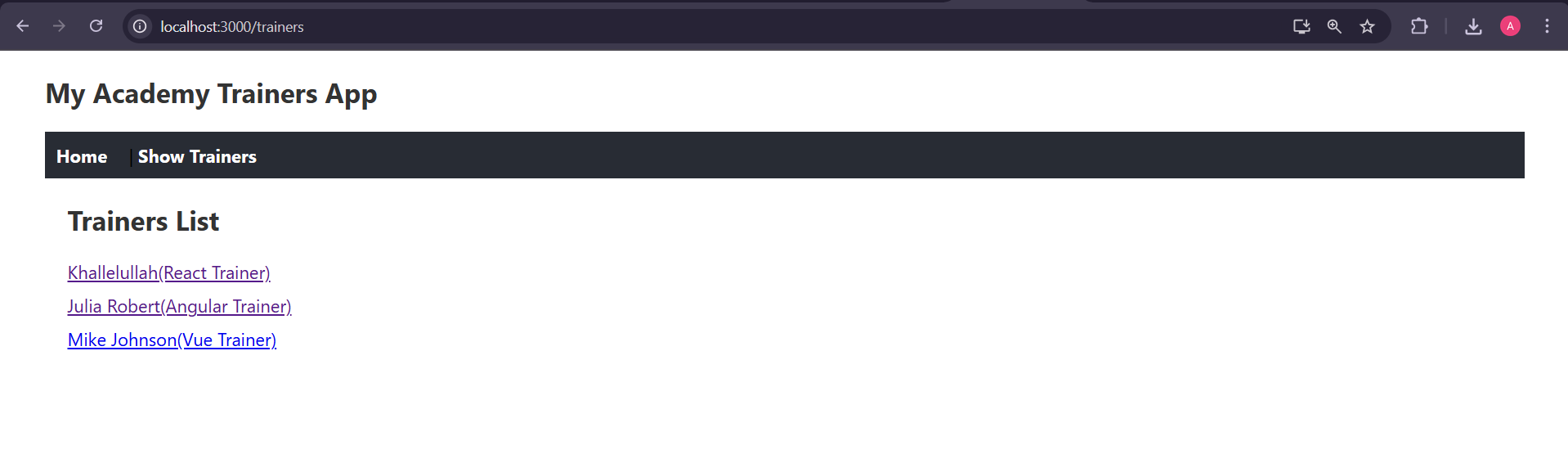
Overall,This hands on lab successfully demonstrated the implementation of a React-based Single Page Application with routing using React Router DOM. The application provides a functional navigation system, dynamic route parameters, and expected UI for displaying structured trainer data.

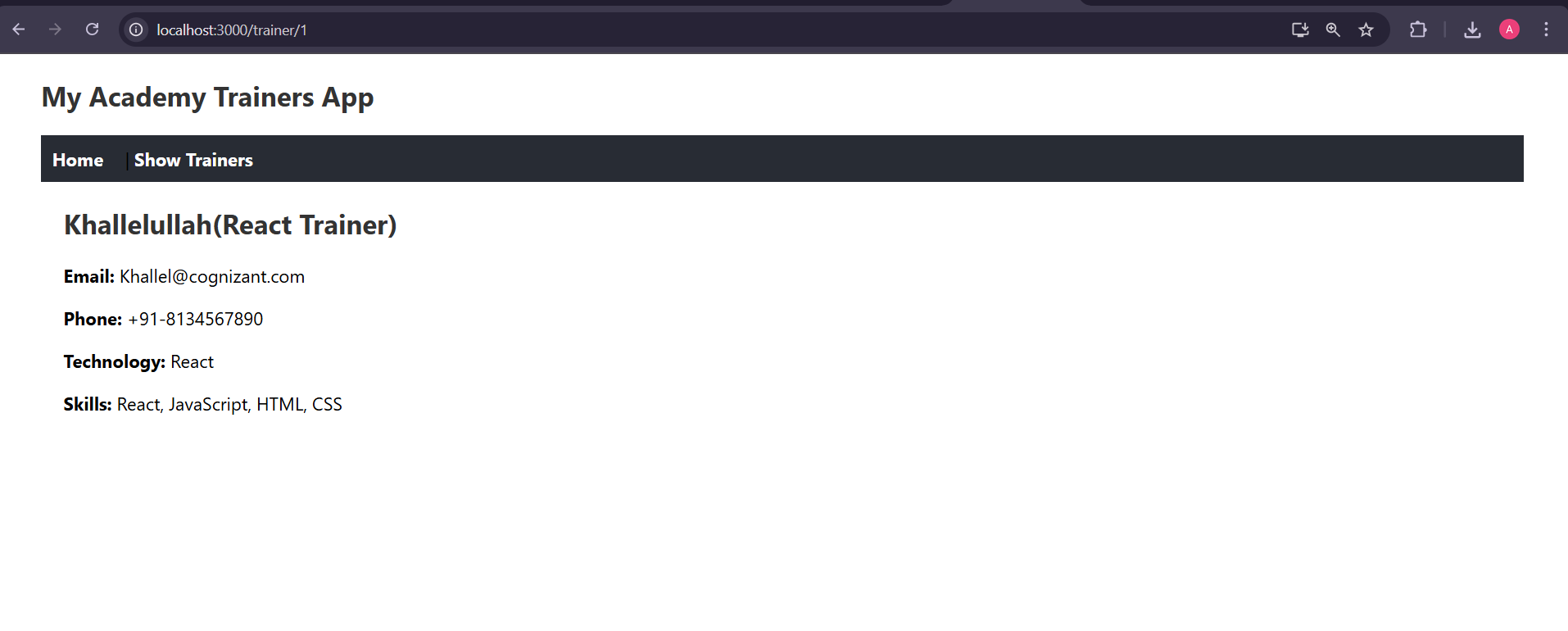




**OUTPUT:**







**EXERCISE-02 07-REACTJS-HOL**

**IMPLEMENTATION:**

The objective of this hands-on exercise was to develop a ReactJS application named ShoppingApp that demonstrates the use of props, class components, and rendering data dynamically in a table format.

The application displays a list of shopping items along with their prices using two class components, OnlineShopping and Cart.

The final output is a neatly formatted table with item names and prices aligned under their respective headings.

To complete this lab, Node.js (v14 or above) and npm were used for package management, along with Visual Studio Code as the development environment.

A basic understanding of JavaScript, React class components, and props was necessary.

The project was created using the create-react-app tool with the command npx create-react-app shoppingapp.

After navigating into the project directory using cd shoppingapp, the project was opened in Visual Studio Code and all development work was carried out inside the src folder.

A file named Cart.js was created to define a class component named Cart. This component accepts two props, Itemname and Price, and renders them as a single row in a table.

Each table cell was styled with a light gray border, padding, and proper text alignment, with item names left-aligned and prices right-aligned.

Another file named OnlineShopping.js was created to define the OnlineShopping class component.

In its constructor, an array of items with their names and prices was initialized in the component state.

The render method created a table structure with a header row for Name and Price, and the items array was looped through using the map() function to pass each item’s data to the Cart component.

Table-level styling was applied to ensure border collapse, fixed table width for consistent column alignment, and centered positioning to match the required design.

The App.js file was updated to import and display the OnlineShopping component within the root application component so that it would render automatically when the application started.

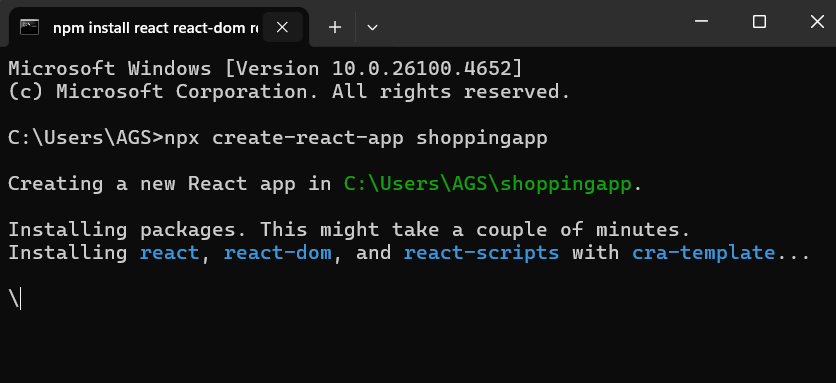
The application was executed using the npm start command, and the browser opened at <http://localhost:3000> displaying the green heading “Items Ordered :” followed by a centered table containing the list of items and their corresponding prices.

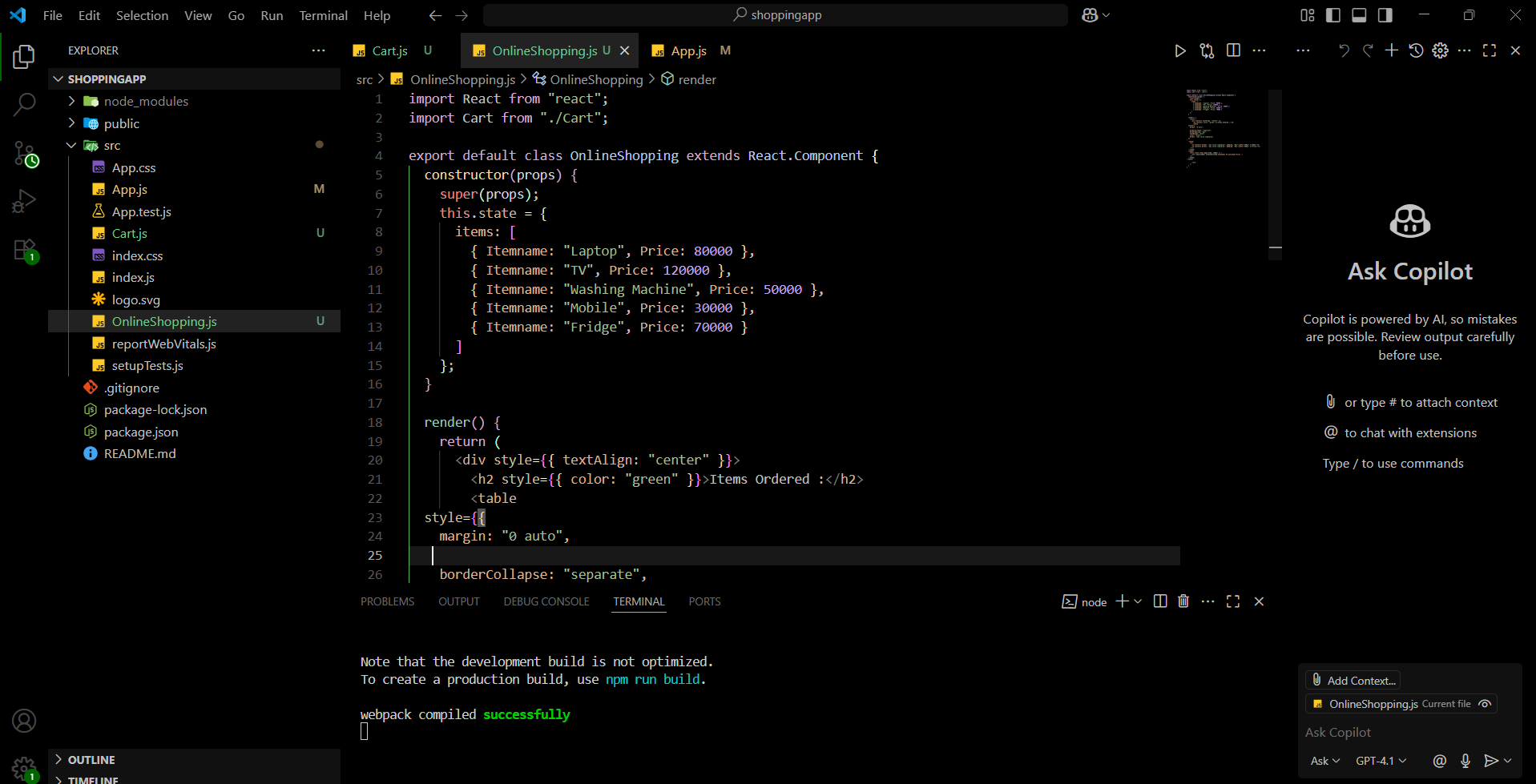
The prices were perfectly aligned under the Price column without empty spaces, and each table cell was enclosed in a thin light gray border, exactly as per the expected output.

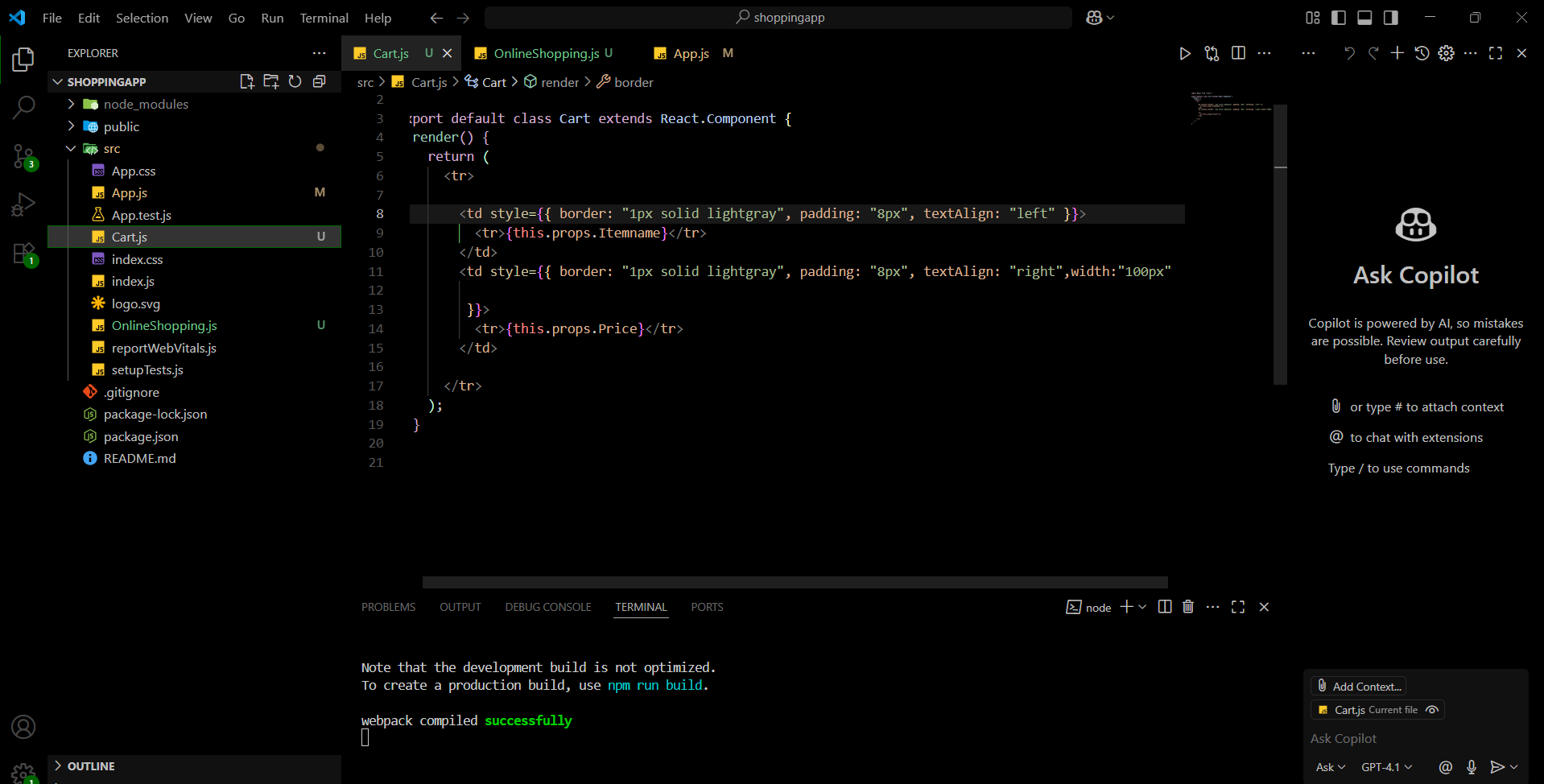
This hands-on lab successfully demonstrated the process of building a simple ReactJS application using class components and props, along with the application of clean and consistent table formatting.

The final result matched the required design precisely and can serve as a foundation for further enhancements such as dynamic product addition or integration with APIs for fetching live data.

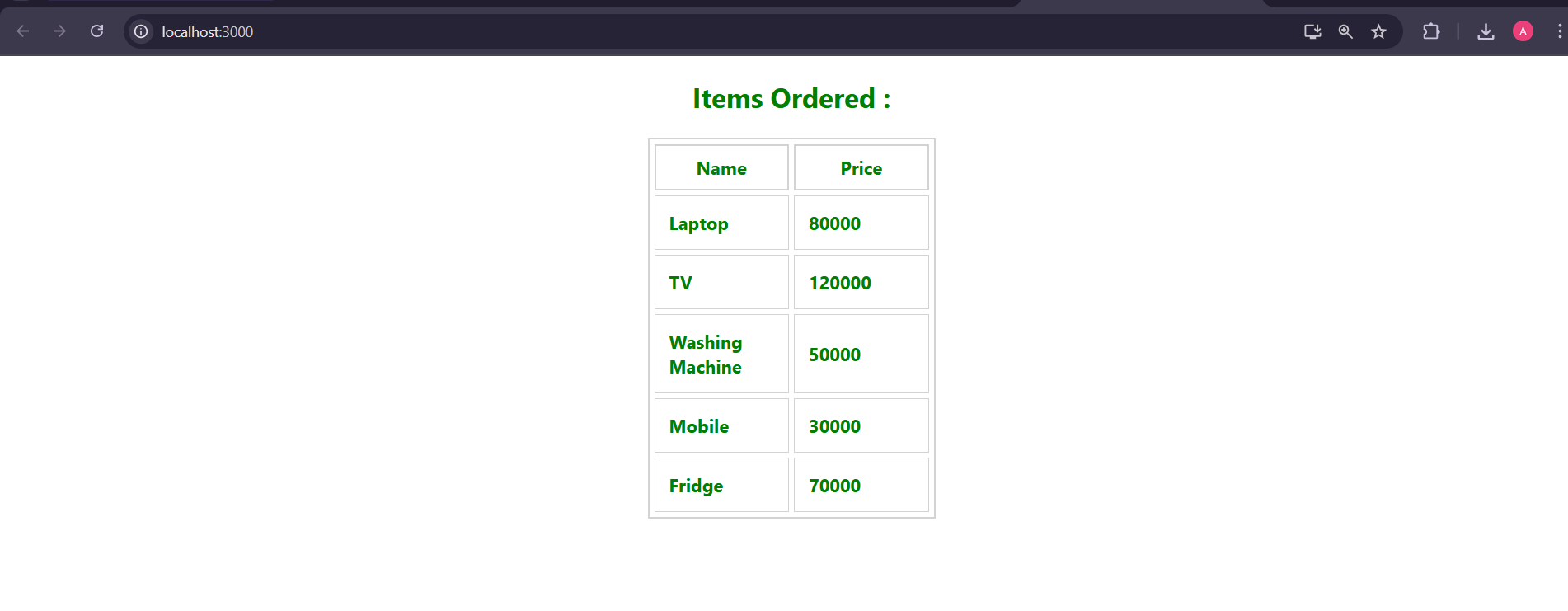
Bottom of Form

****

****

****

**OUTPUT:**

****

**EXERCISE-03 08REACTJS-HOL**

**IMPLEMENTATION:**

To begin with, a new React application was created using create-react-app by running npx create-react-app counterapp and navigating into the project directory. The development server was started using npm start, which launched the app in the browser on http://localhost:3000.

Next, a new component named CountPeople.jsx was created inside the src folder. This component was implemented as a class component since the requirement specified using a constructor and state. Inside the constructor, two state variables, entryCount and exitCount, were initialized to zero.

Two methods, updateEntry and updateExit, were defined and bound to this in the constructor.

These methods use setState to increment the respective count values when the corresponding button is clicked.

The component’s render method was structured to display a heading, two count boxes for "Entered" and "Exited" values, and two buttons labeled Login and Exit.

The count values were displayed dynamically by accessing the component’s state.

The buttons were wired to the updateEntry and updateExit methods via the onClick event so that clicking on them would update the displayed counts immediately.

The CountPeople component was then imported into App.js and rendered within the main application layout.

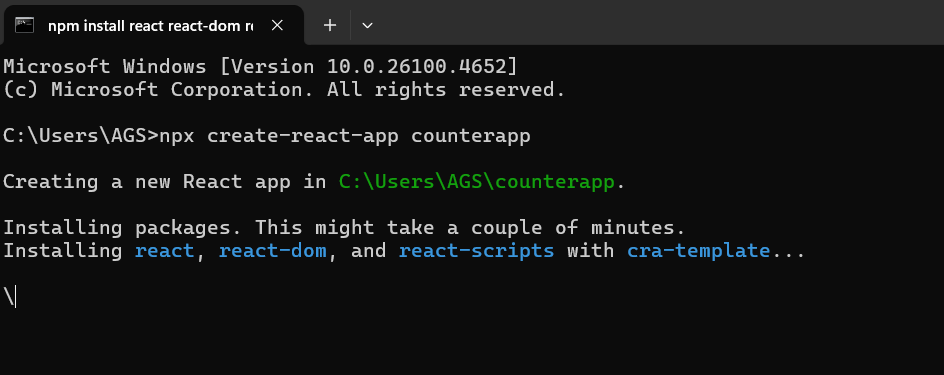
Basic CSS styling was added in a separate styles.css file to make the counter visually appealing.

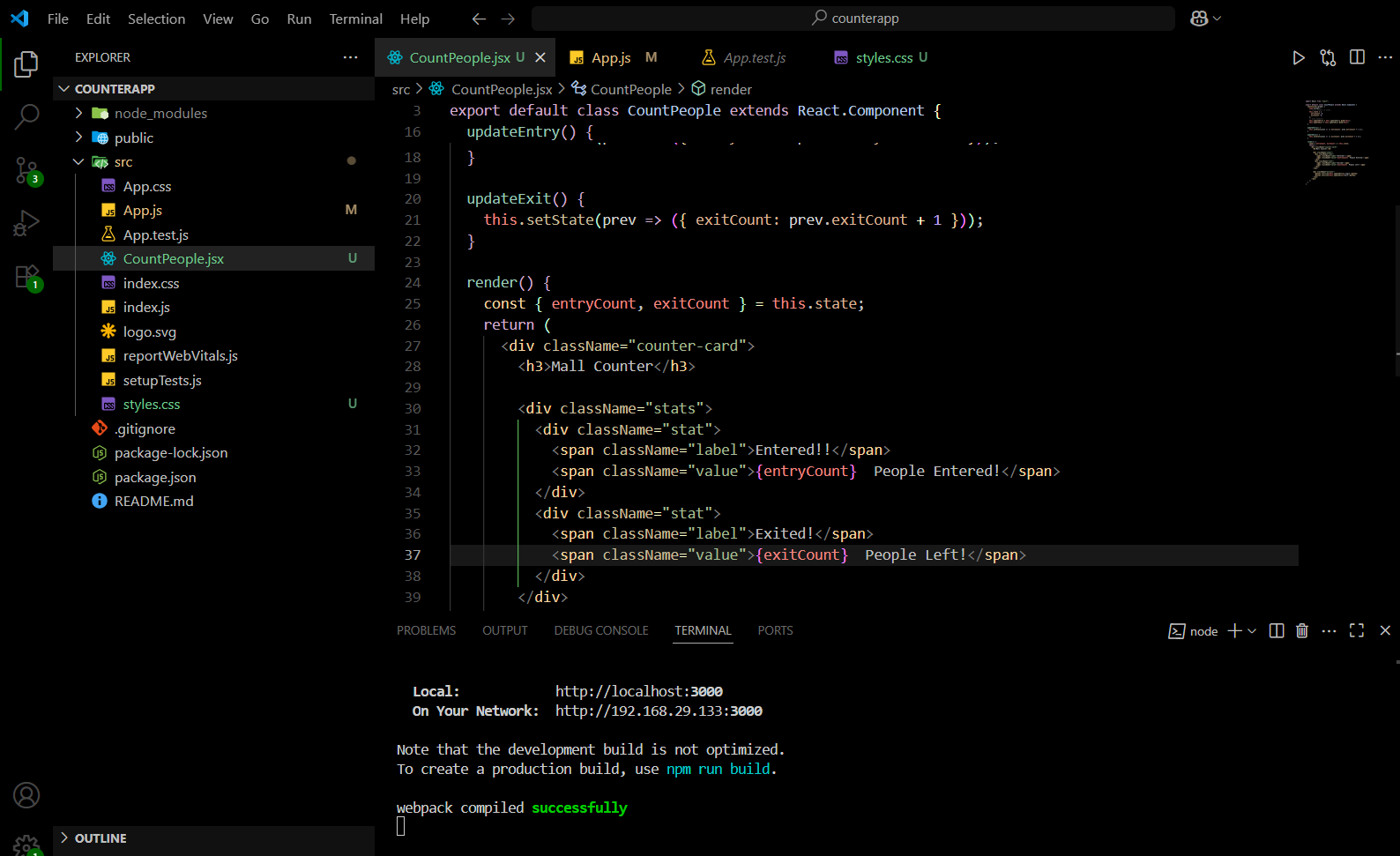
This included styling for a bordered and rounded container, boxed count displays with green text, and neatly spaced buttons.

The design ensures each count appears in its own bordered box, creating a clean, user-friendly interface.

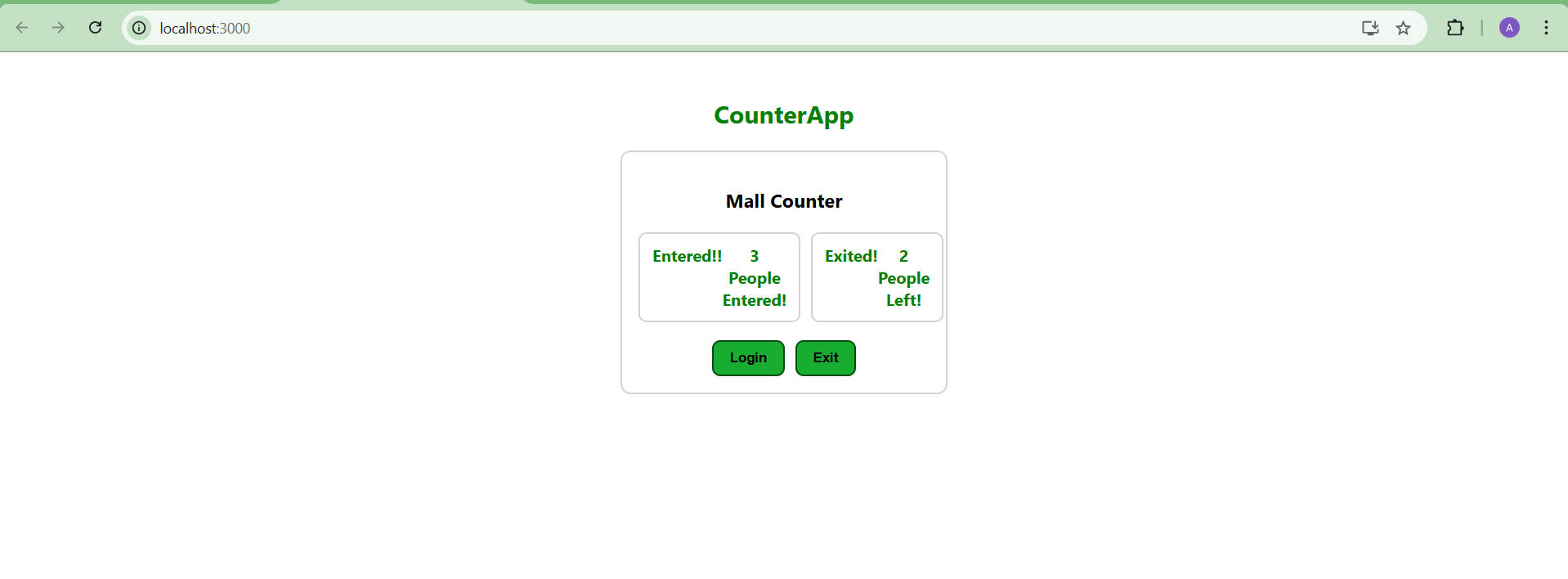
Finally, the application was tested by clicking the Login and Exit buttons to confirm that the counts incremented as expected.

This completed the hands-on exercise, fulfilling the requirement of using a class component, constructor-based state, and button-driven updates in React with added UI as expected by the handson.

****

****

**OUTPUT:**

****