**What is session management in react?**

Session management in React refers to the process of maintaining and managing the state of a user's interaction with an application across different components, routes, and sessions.

This is especially important in web applications that require user authentication, where you need to ensure that a user remains logged in as they navigate through the app, or across different visits.

**Key Aspects of Session Management**

1. **Authentication State:**
   * This involves checking whether a user is logged in or not.
   * The authentication state determines whether the user can access certain parts of the application (like private routes) or whether they need to be redirected to a login page.
2. **Session Persistence:**
   * The session must persist across page reloads and even browser sessions.
   * This is often achieved by storing session data (like tokens or user information) in the browser's local storage, session storage, or cookies.
3. **Session Expiration:**
   * For security reasons, sessions might need to expire after a certain period of inactivity.
   * When a session expires, the user might be logged out automatically, requiring them to log in again.
4. **Session Storage:**
   * The session data is typically stored in the browser using one of the following:
     + **Local Storage:** Persistent storage that lasts even after the browser is closed.
     + **Session Storage:** Temporary storage that is cleared when the session ends (i.e., when the browser or tab is closed).
     + **Cookies:** Small data files stored on the user's device, often used to store session tokens securely.
5. **State Management:**
   * React provides tools like Context API, and third-party libraries like Redux, to manage the global state of the application.
   * These can be used to store and manage session information across the entire app, ensuring that the user state is consistent regardless of where they are in the application.
6. **Secure Communication:**
   * Session management also involves ensuring that communication between the client (React app) and the server is secure.
   * This often involves using HTTPS, secure tokens (like JWT), and ensuring that sensitive data is not exposed.

**Typical Workflow of Session Management**

1. **User Login:**
   * When a user logs in, the application sends their credentials to the server.
   * If the login is successful, the server returns a session token (e.g., JWT) that is stored in the client’s local storage, session storage, or a cookie.
2. **Maintaining Session State:**
   * The token or session information is used to authenticate the user for subsequent requests to protected resources.
   * The React app checks for the presence of the token to determine if the user is logged in.
3. **Session Expiry and Logout:**
   * The session may expire based on time or when the user logs out.
   * The session data is then cleared from storage, and the user is redirected to the login page.
4. **Refreshing Tokens:**
   * To maintain a session without requiring frequent logins, some applications implement token refresh mechanisms.
   * A refresh token can be used to obtain a new session token without requiring the user to log in again.

**Example**

* **Login:** User logs in, and a **JWT token** is stored in local storage.
* **Protected Routes:** The React app checks for the token on every route change. If the token is absent or invalid, the user is redirected to the login page.
* **Logout:** When the user logs out, the token is removed from storage, and the session is effectively terminated.

**Conclusion**

Session management in React is about keeping track of the user's state within an application, ensuring that they stay authenticated across different routes and sessions, and securely handling session data.

It's a crucial aspect of building secure and user-friendly web applications.

**What is SessionContext in react?**

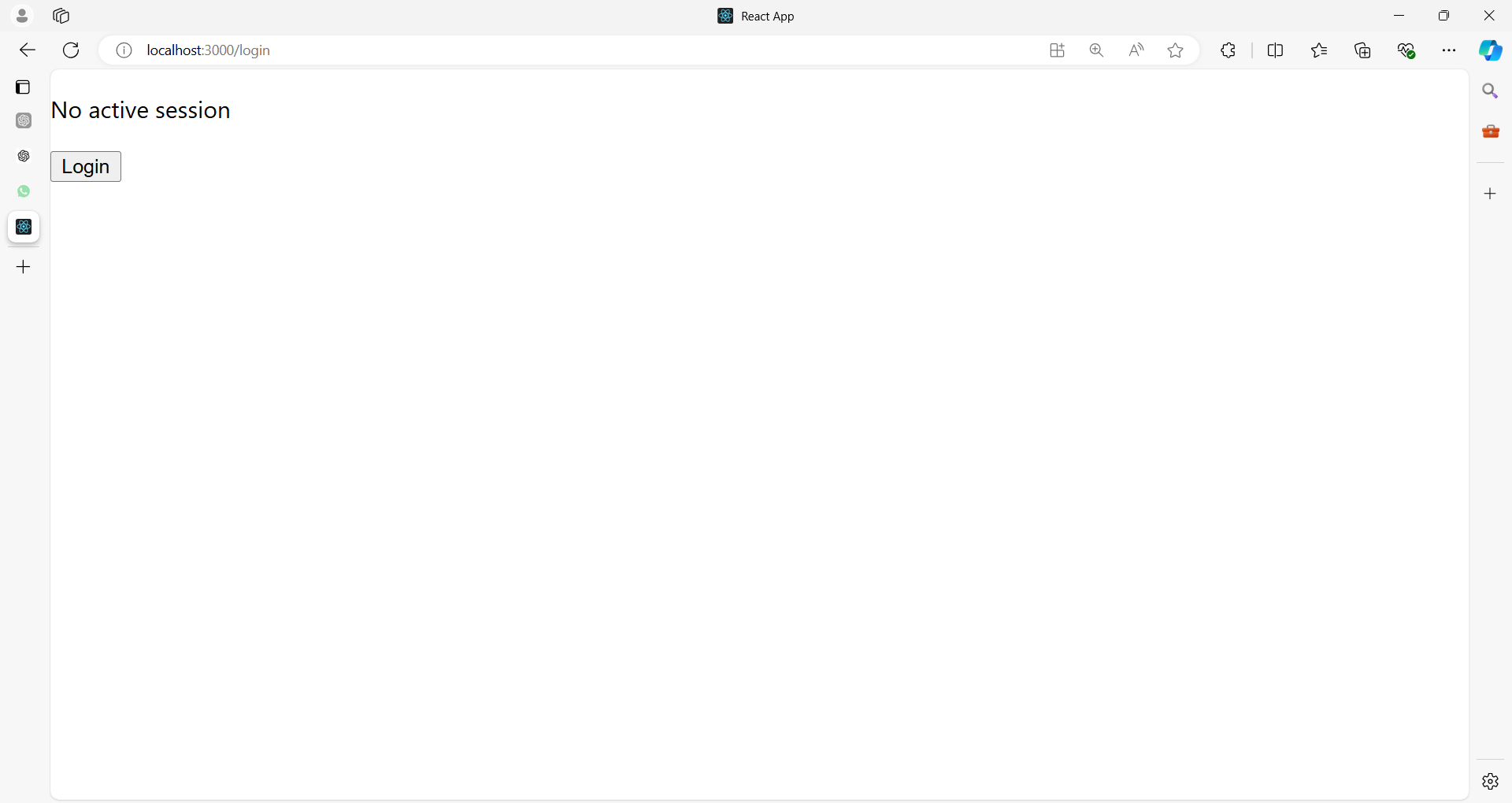
In React, SessionContext refers to a pattern where the context API is used to manage session-related data, such as user authentication, session tokens, or any other state that needs to persist across different components of the application.

This is part of React's Context API, which is useful for managing global state without the need for prop drilling.

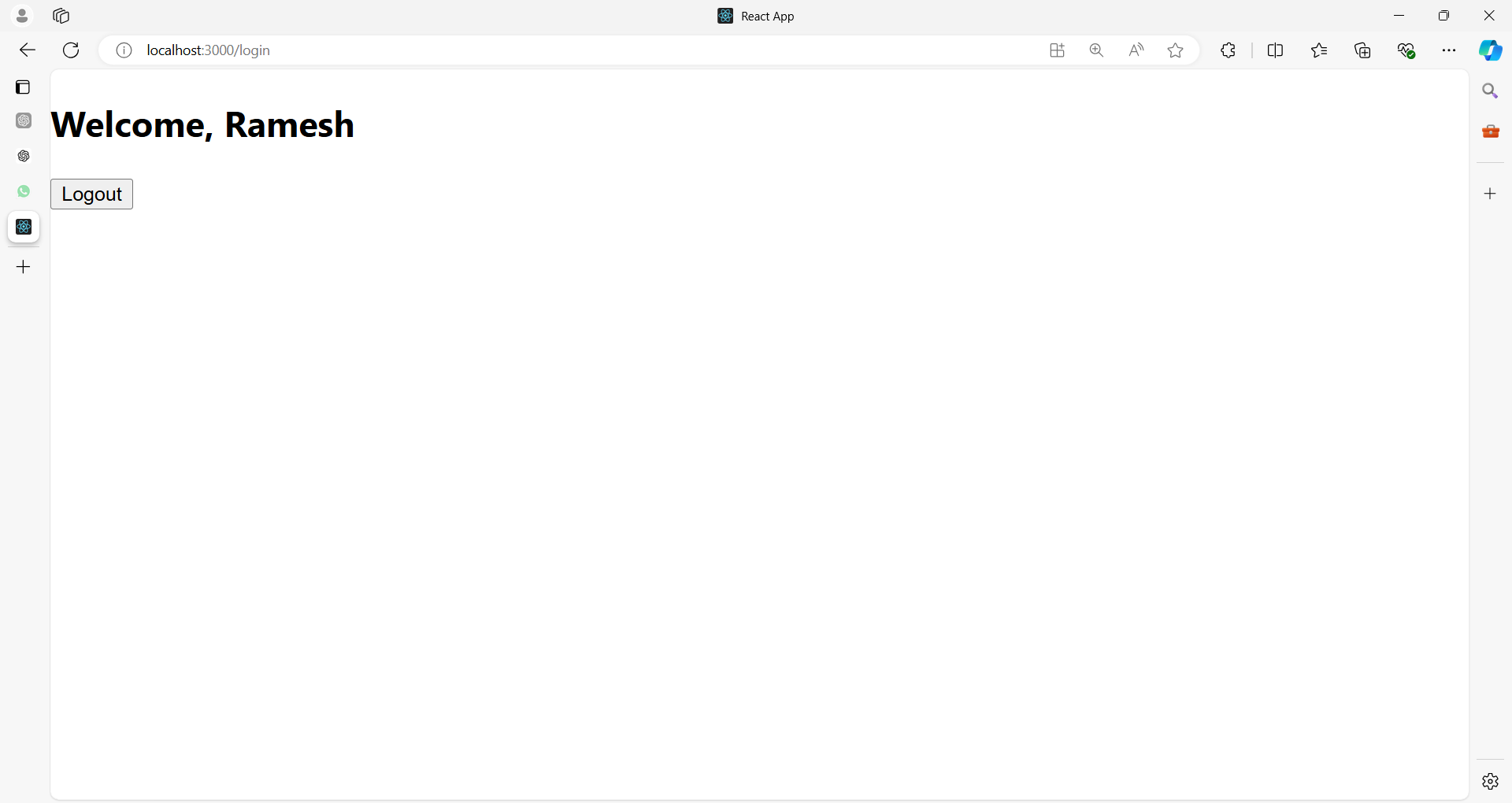
Here’s how SessionContext works:

1. **Creation of Context:** A SessionContext is created using React.createContext(). This will act as a container to store session-related data like user authentication status, tokens, and user information.
2. **Provider:** The SessionContext.Provider component is used to wrap the parts of your application that need access to the session data. It provides the session state and any methods to modify that state to all the child components.
3. **Consumer:** Components can then consume this session data using either useContext hook or the SessionContext.Consumer component.

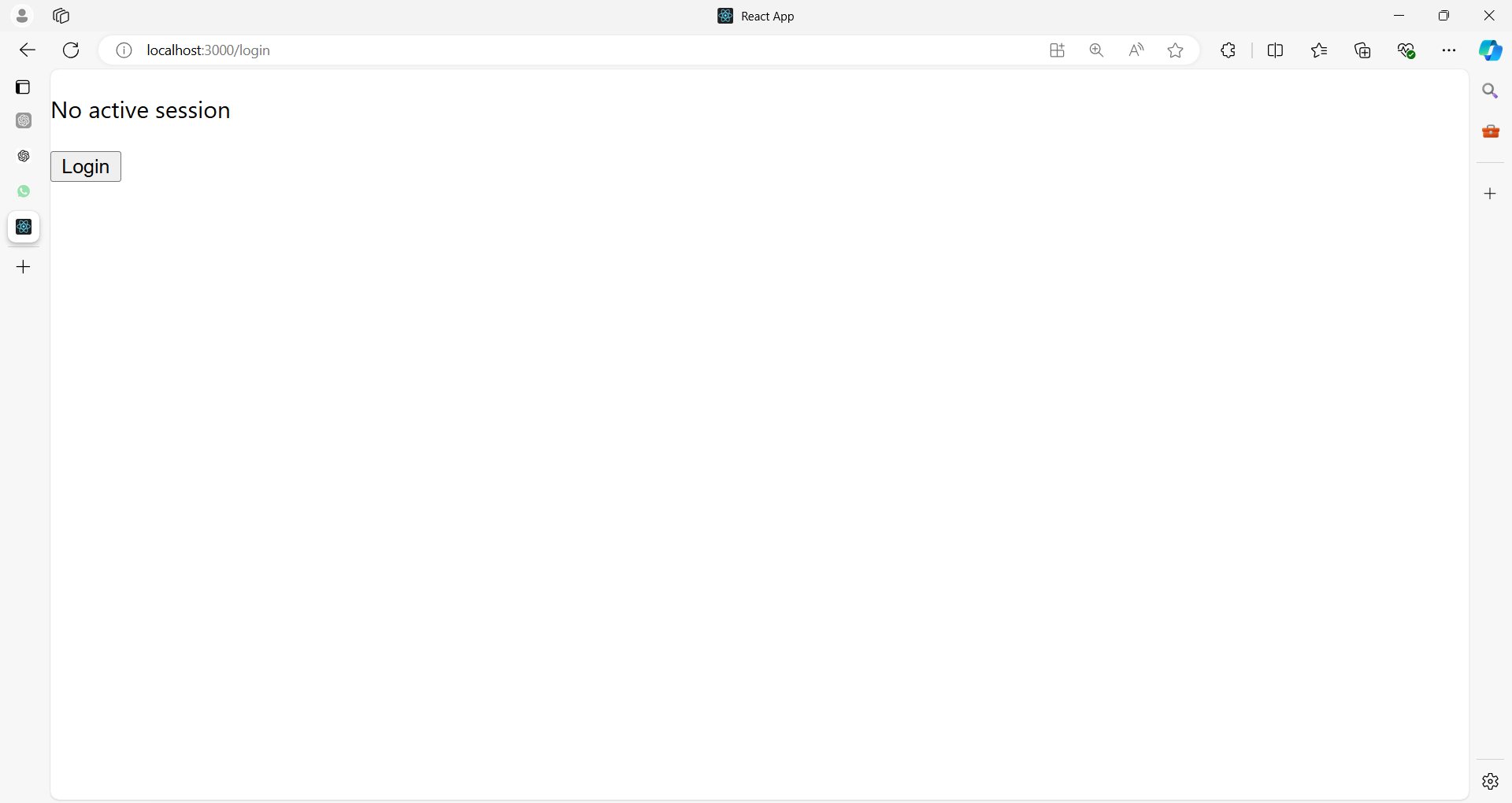
**Example of SessionContext:**



**After clicking on Login button**



**After clicking on Logout button**



**UserProfile.js**

import React, { createContext, useState, useContext } from 'react';

// Create the SessionContext

const SessionContext = createContext();

// SessionProvider component to wrap the app

export const SessionProvider = ({ children }) => {

  const [session, setSession] = useState(null); // null implies no active session

  const login = (userData) => {

    setSession(userData); // Set session with user data

  };

  const logout = () => {

    setSession(null); // Clear session data on logout

  };

  return (

    <SessionContext.Provider value={{ session, login, logout }}>

      {children}

    </SessionContext.Provider>

  );

};

// Custom hook to access the session data

export const useSession = () => useContext(SessionContext);

// Example Component that consumes session data

const UserProfile = () => {

  const { session, login, logout } = useSession();

  if (!session) {

    // Simulate a login with hardcoded user data

    return (

      <div>

        <p>No active session</p>

        <button onClick={() => login({ username: 'Ramesh' })}>Login</button>

      </div>

    );

  }

  return (

    <div>

      <h2>Welcome, {session.username}</h2>

      <button onClick={logout}>Logout</button>

    </div>

  );

};

// Usage in App

const App = () => (

  <SessionProvider>

    <UserProfile />

  </SessionProvider>

);

export default App;

**Breakdown:**

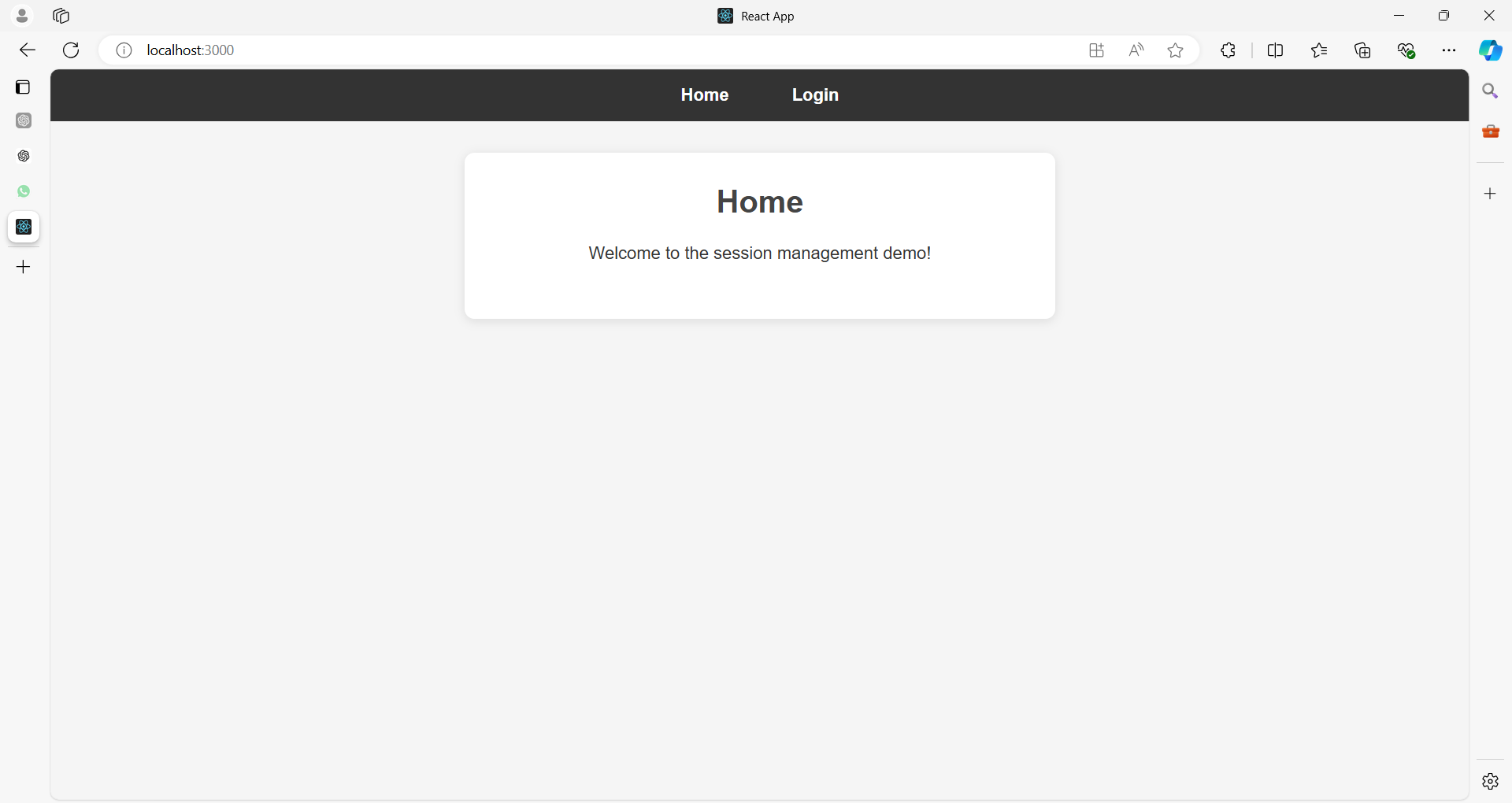
* **SessionContext:** Holds the session state and provides login and logout functions.
* **SessionProvider:** Wraps the component tree to allow access to session state throughout the app.
* **useSession Hook:** Allows easy access to session data and functions like login and logout in any component.

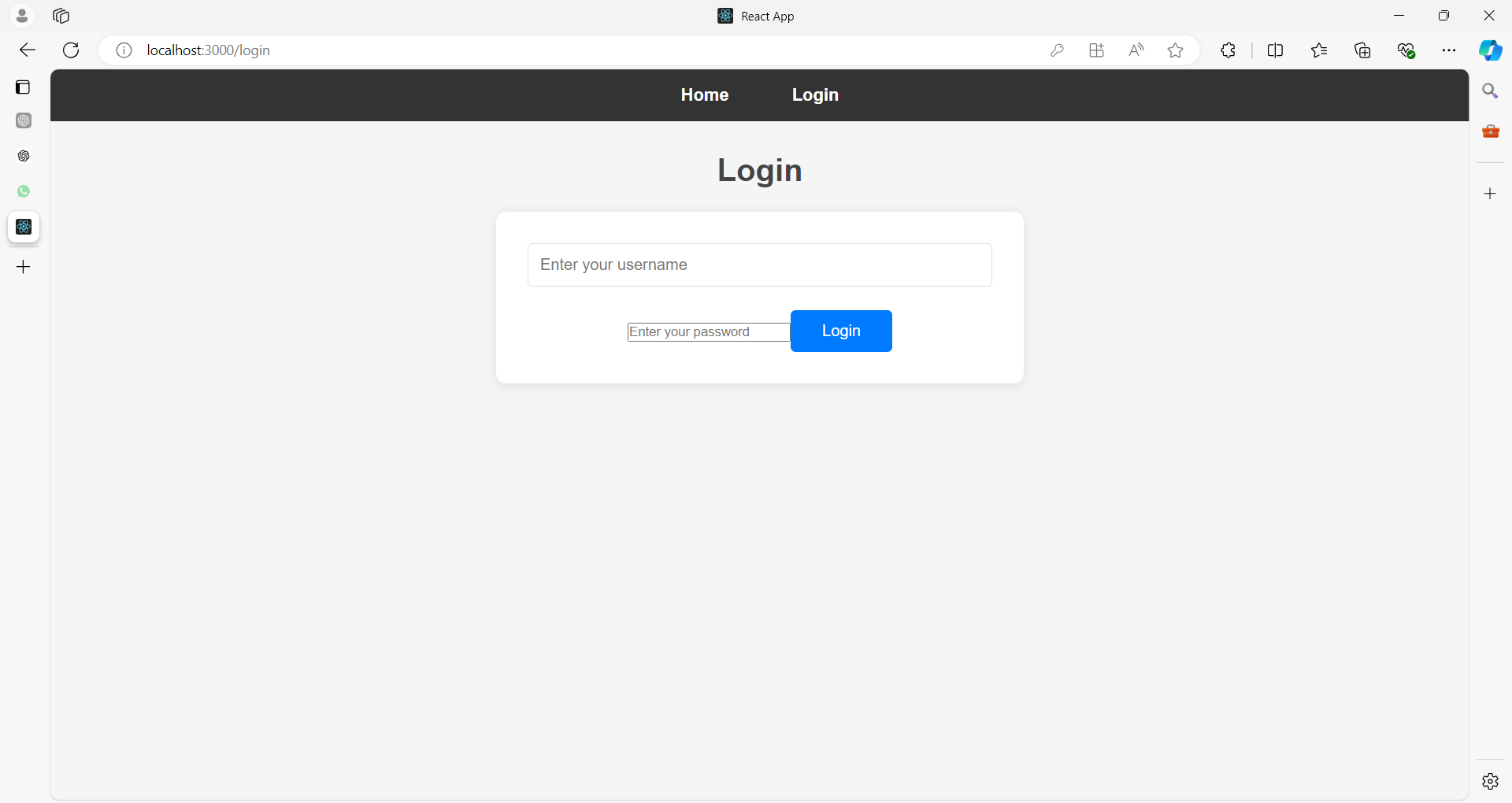
This pattern is especially useful for managing global session data like user login status and maintaining state across multiple routes or components in a React application.

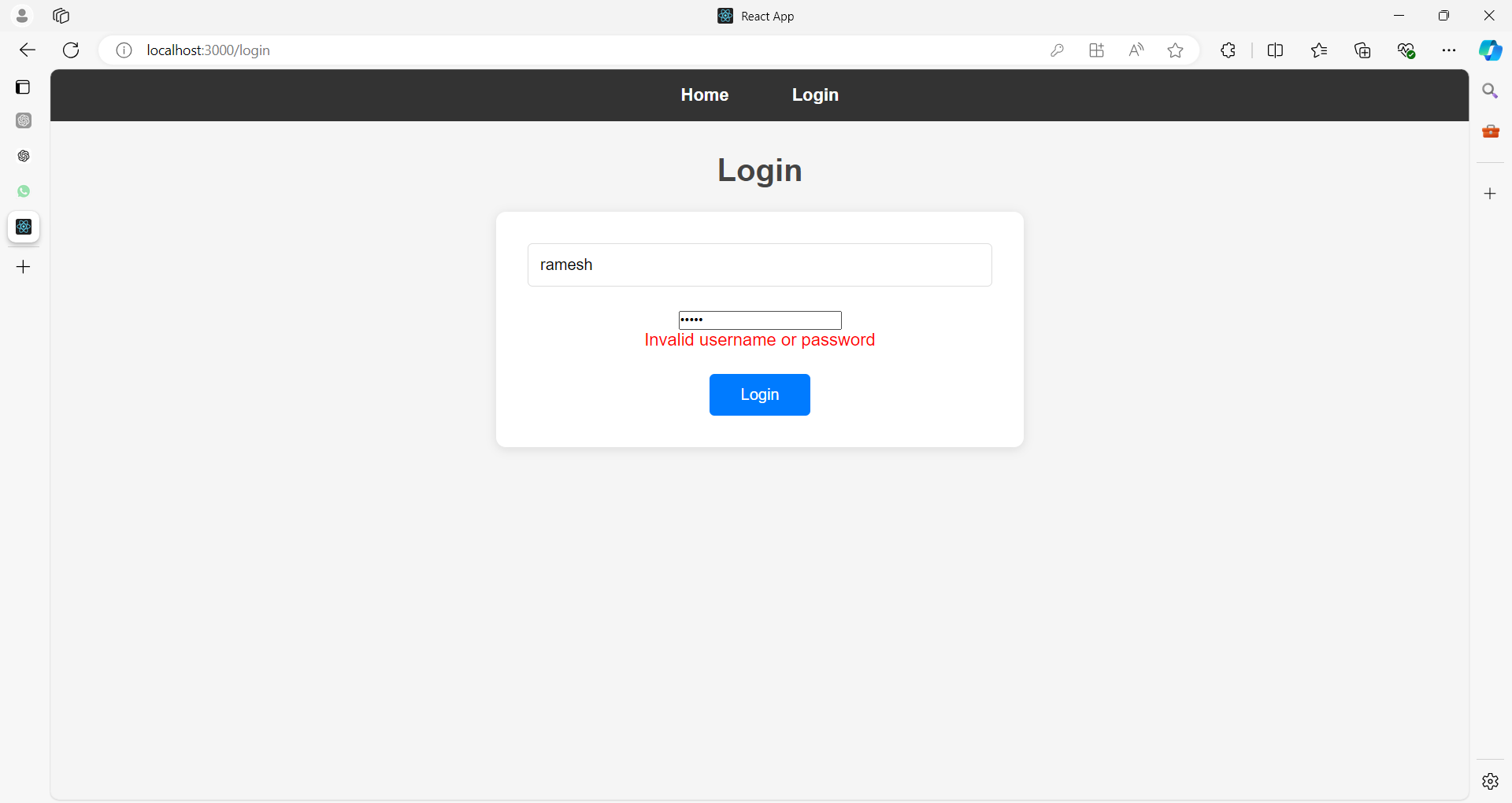
**Example 01:**

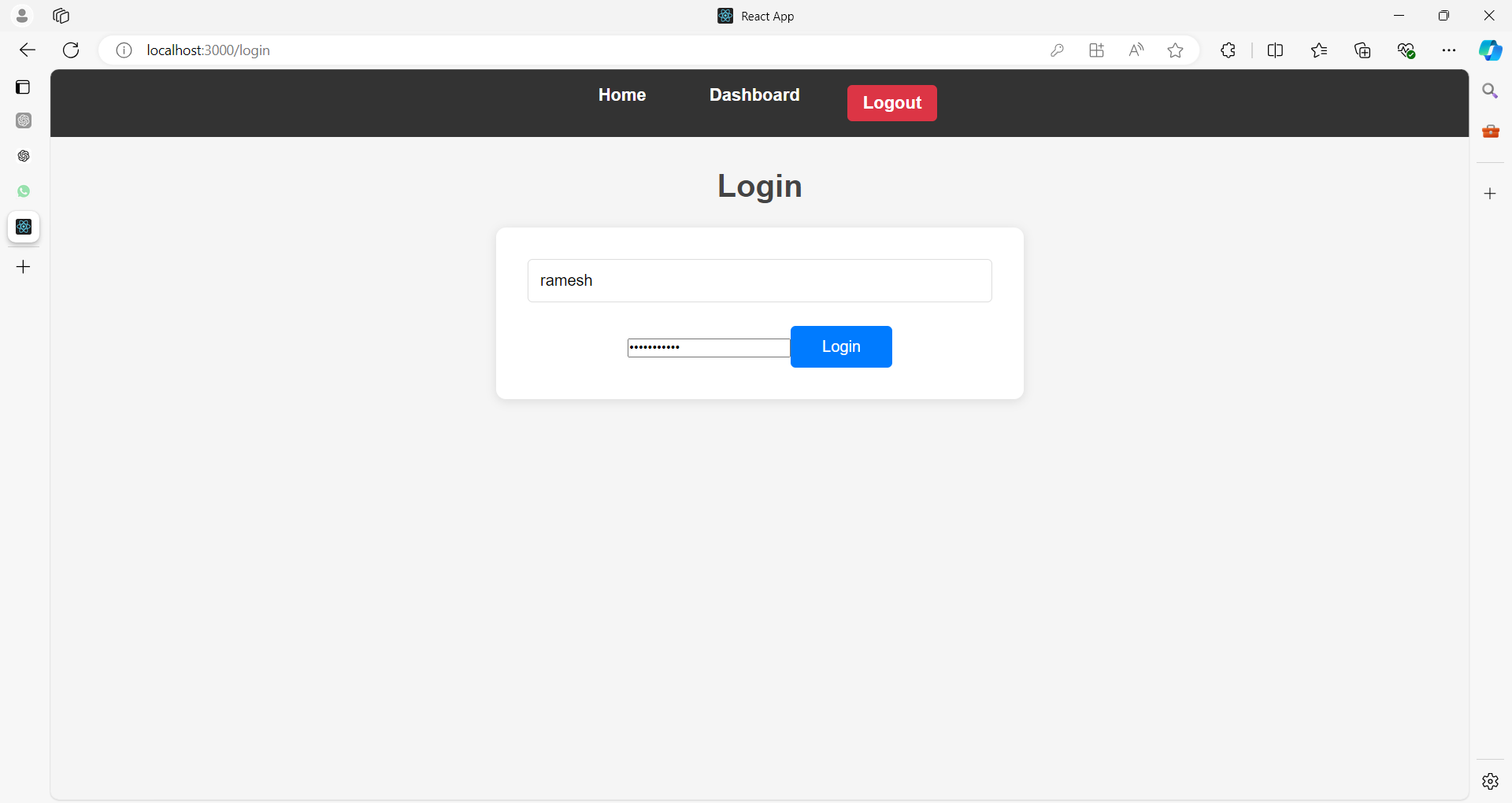
Let's create a simple React application that demonstrates basic session management using the Context API and local storage.

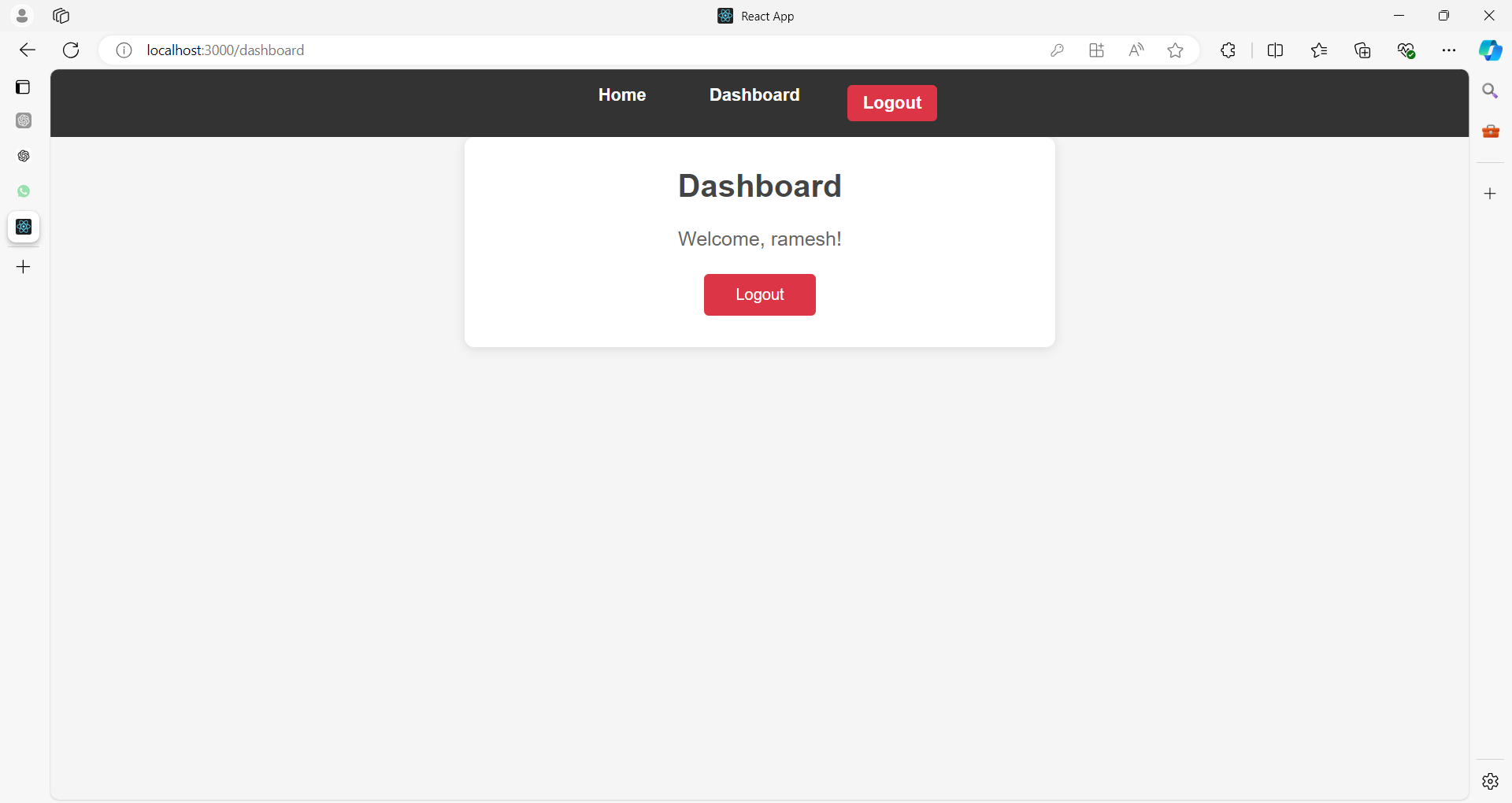
This example will include a login/logout system where the session is maintained across page reloads.



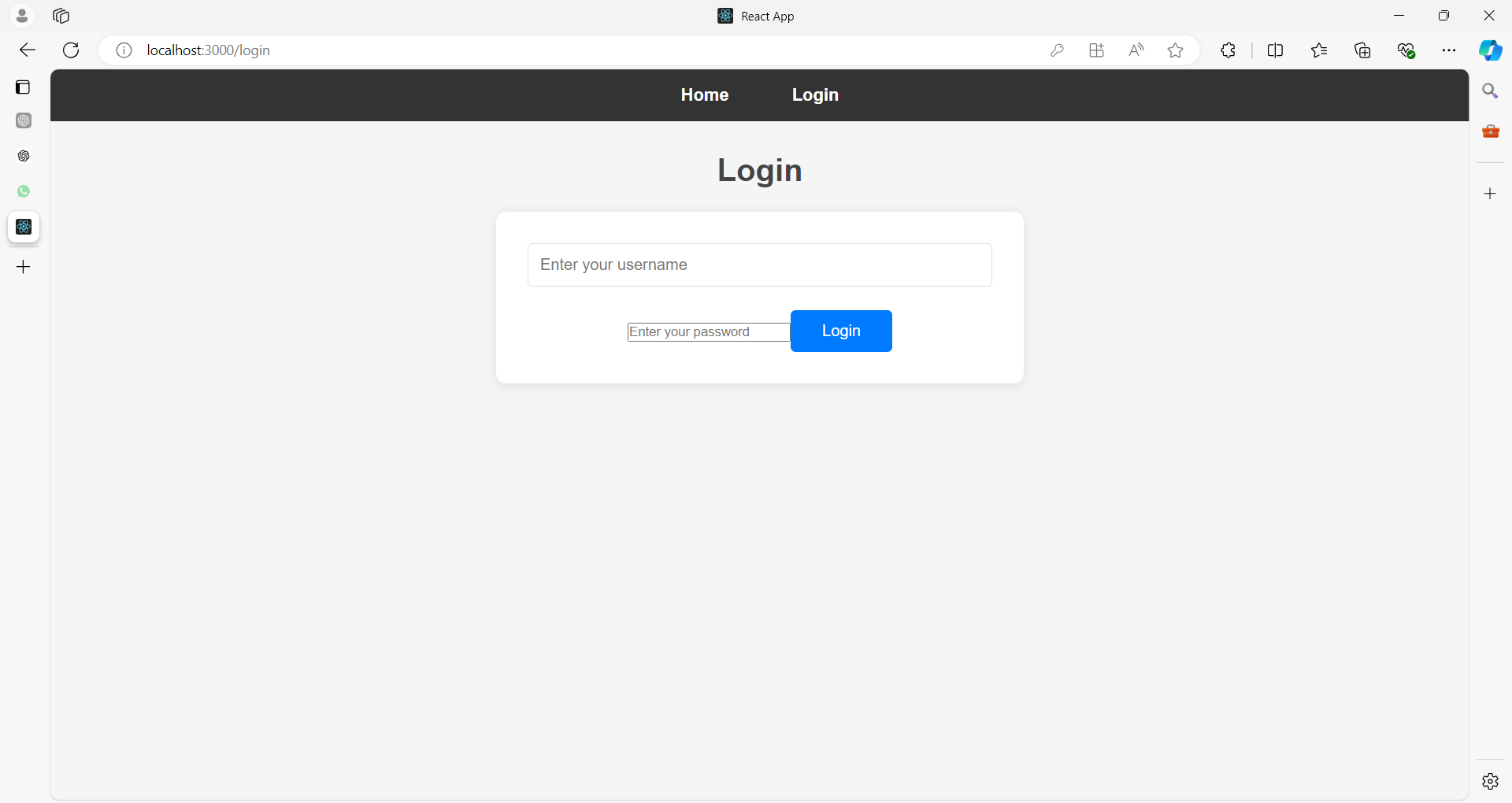








**After clicking on Logout**



**Step 1: Set Up the Project**

Start by creating a new React application if you haven't already.

npx create-react-app react-session-management

cd react-session-management

public/users.json

[

    {

        "username": "ramesh",

        "password": "password123"

    },

    {

        "username": "anish\_naik",

        "password": "naik123"

    },

    {

      "username": "john\_doe",

      "password": "password123"

    },

    {

      "username": "jane\_doe",

      "password": "password456"

    }

  ]

**App.css**

Here’s a simple and clean CSS design that will give your React app a modern look while maintaining good user experience.

Here is the styles for your login form, dashboard, and the overall layout, using a minimalist design with centered content.

You can add the CSS in a App.css file (or another relevant CSS file) and link it to your React components.

**src/App.css**

/\* Global Styles \*/

\* {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

  font-family: 'Arial', sans-serif;

}

body {

  background-color: #f5f5f5;

  color: #333;

}

.container {

  max-width: 600px;

  margin: 0 auto;

  padding: 2rem;

  text-align: center;

}

h2 {

  color: #444;

  font-size: 2rem;

  margin-bottom: 1.5rem;

}

p {

  margin-bottom: 1.5rem;

  font-size: 1.1rem;

}

/\* Header \*/

header {

  background-color: #333;

  color: #fff;

  padding: 1rem;

}

header a {

  color: #fff;

  text-decoration: none;

  margin: 0 1rem;

  font-weight: bold;

}

header a:hover {

  text-decoration: underline;

}

/\* Form Styles \*/

form {

  background-color: #fff;

  padding: 2rem;

  border-radius: 10px;

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

}

input[type="text"] {

  width: 100%;

  padding: 0.75rem;

  margin-bottom: 1.5rem;

  border: 1px solid #ddd;

  border-radius: 5px;

  font-size: 1rem;

}

button {

  padding: 0.75rem 2rem;

  background-color: #007bff;

  color: white;

  border: none;

  border-radius: 5px;

  font-size: 1rem;

  cursor: pointer;

}

button:hover {

  background-color: #0056b3;

}

/\* Home Page \*/

.home {

  background-color: #fff;

  padding: 2rem;

  border-radius: 10px;

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

  margin-top: 2rem;

}

/\* Dashboard \*/

.dashboard {

  background-color: #fff;

  padding: 2rem;

  border-radius: 10px;

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

}

.dashboard p {

  font-size: 1.25rem;

  color: #666;

}

button.logout-btn {

  background-color: #dc3545;

}

button.logout-btn:hover {

  background-color: #c82333;

}

nav ul {

  list-style-type: none;

  display: flex;

  justify-content: center;

}

nav ul li {

  margin: 0 1rem;

}

nav ul li a, nav ul li button {

  color: white;

  text-decoration: none;

  font-size: 1.1rem;

  font-weight: bold;

}

nav ul li button {

  background-color: #dc3545;

  border: none;

  padding: 0.5rem 1rem;

  cursor: pointer;

  color: white;

  border-radius: 5px;

}

nav ul li a:hover, nav ul li button:hover {

  text-decoration: underline;

}

**Explanation:**

* **Global Styles:** Basic styles such as resetting margins and padding, setting a global font, and adjusting the overall body color for a clean and minimal design.
* **Container:** Centers the content horizontally with a maximum width and padding. This class can be applied to your main wrapper components like Login, Dashboard, and Home.
* **Form Styles:** Provides styling for the login form, including the input fields and submit button. The button has hover effects to enhance interactivity.
* **Header:** Defines styles for the header, with links having hover effects.
* **Home & Dashboard:** Adds basic styling for both the home and dashboard components, giving them a card-like appearance with shadows and rounded corners for a clean, polished look.
* **Buttons:** The buttons are styled to be modern, with different colors for login and logout buttons.

**Step 2: Create the Session Context**

Create a SessionContext.js file to manage the session state.

// src/SessionContext.js

import React, { createContext, useState, useEffect, useContext } from 'react';

// Create a context

const SessionContext = createContext();

// Session provider component

export const SessionProvider = ({ children }) => {

  const [session, setSession] = useState(null);

  // Load session from local storage if available

  useEffect(() => {

    const storedSession = localStorage.getItem('session');

    if (storedSession) {

      setSession(JSON.parse(storedSession));

    }

  }, []);

  // Save session to local storage

  useEffect(() => {

    if (session) {

      localStorage.setItem('session', JSON.stringify(session));

    } else {

      localStorage.removeItem('session');

    }

  }, [session]);

  const login = (username) => setSession({ username });

  const logout = () => setSession(null);

  return (

    <SessionContext.Provider value={{ session, login, logout }}>

      {children}

    </SessionContext.Provider>

  );

};

// Custom hook to use session context

export const useSession = () => useContext(SessionContext);

Here's a detailed explanation of the SessionContext.js code, breaking it down line by line:

**1. Imports**

**import React, { createContext, useState, useEffect, useContext } from 'react';**

* React: The core library for building user interfaces in React.
* createContext: Used to create a context that can be shared across components without passing props.
* useState: A React hook that allows you to add state to functional components.
* useEffect: A React hook that performs side effects in functional components (e.g., fetching data, saving to local storage).
* useContext: A React hook that allows you to consume context values in any child component.

**2. Creating a Context**

**const SessionContext = createContext();**

* SessionContext: A context object created using createContext(). This object allows us to manage and share the session state across multiple components in the app. Initially, it doesn't have any value.

**3. SessionProvider Component**

**export const SessionProvider = ({ children }) => {**

* SessionProvider: A React component that acts as a wrapper around the entire application or parts of it, providing session data and functions (like login and logout) to its children components.
* It takes children as a prop, which represents the child components inside the SessionProvider.

**const [session, setSession] = useState(null);**

* session: The state variable that holds the session information. Initially, it's set to null (no session).
* setSession: A function that updates the session state.
* useState(null): The useState hook initializes the session state with null to represent that no user is logged in when the app starts.

**4. Load Session from Local Storage (on Component Mount)**

**useEffect(() => {**

**const storedSession = localStorage.getItem('session');**

**if (storedSession) {**

**setSession(JSON.parse(storedSession));**

**}**

**}, []);**

* useEffect: This hook runs after the component mounts (i.e., when the component is first rendered). The [] (empty dependency array) ensures it only runs once.
* localStorage.getItem('session'): Retrieves the stored session data from the browser’s localStorage (if it exists). localStorage is a browser feature that allows data to persist across page reloads.
* if (storedSession): Checks if a session exists in localStorage.
* setSession(JSON.parse(storedSession)): If session data is found, it is parsed from JSON format (because localStorage stores everything as strings) and set as the new session state using setSession.

This ensures that if the user has previously logged in and their session was saved in localStorage, the app restores their session when it is reloaded.

**5. Save Session to Local Storage (when session changes)**

**useEffect(() => {**

**if (session) {**

**localStorage.setItem('session', JSON.stringify(session));**

**} else {**

**localStorage.removeItem('session');**

**}**

**}, [session]);**

* This useEffect watches for changes to the session state, and whenever it changes:
  + If session is **not null** (user has logged in), it stores the session in localStorage using localStorage.setItem(). Since localStorage only supports strings, it converts the session object to a string using JSON.stringify.
  + If session is **null** (user has logged out), it removes the session from localStorage using localStorage.removeItem('session').

This keeps the session state synchronized between the application state and the browser's localStorage.

**6. Login and Logout Functions**

**const login = (username) => setSession({ username });**

* login: This function accepts a username and updates the session state with an object that contains the username.
* setSession({ username }): The session state is updated to include the provided username.

**const logout = () => setSession(null);**

* logout: This function sets the session state to null, effectively logging the user out.
* When the session is set to null, the useEffect that monitors session will remove the session from localStorage.

**7. Providing the Context to Children**

**return (**

**<SessionContext.Provider value={{ session, login, logout }}>**

**{children}**

**</SessionContext.Provider>**

**);**

* SessionContext.Provider: This is a special React component that allows us to share the session state and the login and logout functions with any component that needs it.
* value={{ session, login, logout }}: The value prop provides the data that any component inside the SessionProvider can access. Here, it provides:
  + session: The current session state.
  + login: The function to log in a user.
  + logout: The function to log out a user.
* {children}: Represents the child components wrapped inside SessionProvider. These child components can now access the session data and functions.

**8. Custom Hook for Consuming the Context**

**export const useSession = () => useContext(SessionContext);**

* useSession: A custom hook that uses useContext(SessionContext) to access the SessionContext. This simplifies how components access the session data.
* useContext(SessionContext): This hook allows any component to consume the SessionContext, giving it access to the session state, login, and logout functions.

By using the useSession hook, components can easily get or modify the session without manually dealing with useContext every time.

**Summary:**

1. **SessionContext** provides a global state for managing user session data.
2. **SessionProvider**:
   * Loads session data from localStorage on mount.
   * Saves or removes session data from localStorage when the session state changes.
   * Exposes the session, login, and logout functions to child components via SessionContext.Provider.
3. **useSession Hook** simplifies access to the session data and methods from any component.

**Step 3: Create Login, Dashboard, and Home Components**

Create simple components to demonstrate session management.

**Login Component**

The Login component allows users to "log in" by entering a username.

// src/components/Login.js

import React, { useState, useEffect } from 'react';

import { useSession } from '../SessionContext';

import '../App.css'; // Import the CSS file

const Login = () => {

  const [username, setUsername] = useState('');

  const [password, setPassword] = useState('');

  const [users, setUsers] = useState([]);

  const [error, setError] = useState('');

  const { login } = useSession();

  // Fetch users from the users.json file on component mount

  useEffect(() => {

    fetch('/users.json')

      .then((response) => response.json())

      .then((data) => setUsers(data))

      .catch((err) => console.error('Error fetching users:', err));

  }, []);

  const handleSubmit = (e) => {

    e.preventDefault();

    // Find the user from the users.json file

    const user = users.find((u) => u.username === username && u.password === password);

    if (user) {

      login(username); // If the user is found, log them in

      setError(''); // Clear any previous errors

    } else {

      setError('Invalid username or password'); // Display error if user is not found

    }

  };

  return (

    <div className="container">

      <h2>Login</h2>

      <form onSubmit={handleSubmit}>

        <input

          type="text"

          placeholder="Enter your username"

          value={username}

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

        />

        <input

          type="password"

          placeholder="Enter your password"

          value={password}

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

        />

        {error && <p style={{ color: 'red' }}>{error}</p>}

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

      </form>

    </div>

  );

};

export default Login;

**Dashboard Component**

The Dashboard component displays the logged-in user's information and provides a logout option.

// src/components/Dashboard.js

import React from 'react';

import { useSession } from '../SessionContext';

import '../App.css'; // Import the CSS file

const Dashboard = () => {

  const { session, logout } = useSession();

  if (!session) {

    return <div className="container">Please log in to access the dashboard.</div>;

  }

  return (

    <div className="container dashboard">

      <h2>Dashboard</h2>

      <p>Welcome, {session.username}!</p>

      <button className="logout-btn" onClick={logout}>Logout</button>

    </div>

  );

};

export default Dashboard;

Here’s a breakdown of the Dashboard.js component:

**1. Imports**

**import React from 'react';**

**import { useSession } from '../SessionContext';**

**import './App.css'; // Import the CSS file**

* **React**: The core library for building user interfaces in React.
* **useSession**: The custom hook from the SessionContext that provides access to the current session and logout functionality.
* **'./App.css'**: Imports the CSS file where styles for the component will be defined.

**2. Component Setup**

**const Dashboard = () => {**

**const { session, logout } = useSession();**

* **session**: Contains the current session data (e.g., username). It is extracted from the SessionContext using the useSession hook.
* **logout**: This function is also provided by SessionContext. It allows the user to log out, which clears the session.

**3. Conditional Rendering**

**if (!session) {**

**return <div className="container">Please log in to access the dashboard.</div>;**

**}**

* **if (!session)**: Checks if there is no active session. If no user is logged in (i.e., session is null or undefined), a message is displayed asking the user to log in.
* **container**: A CSS class for styling the container.

**4. Rendering the Dashboard**

**return (**

**<div className="container dashboard">**

**<h2>Dashboard</h2>**

**<p>Welcome, {session.username}!</p>**

**<button className="logout-btn" onClick={logout}>Logout</button>**

**</div>**

**);**

**};**

* **container**: A general class applied to style the overall container of the dashboard.
* **dashboard**: An additional class that will apply dashboard-specific styles, such as layout, spacing, or typography.
* **{session.username}**: Displays the username of the currently logged-in user, which is stored in the session state.
* **logout-btn**: A CSS class for styling the logout button.
* **onClick={logout}**: Attaches the logout function to the button. When clicked, the user will be logged out and the session cleared.

**5. Exporting the Component**

**export default Dashboard;**

* **Dashboard**: The Dashboard component is exported for use in other parts of the app.

**Summary:**

* The Dashboard component checks if the user is logged in by checking the session state.
  + If no session is found, a message is displayed prompting the user to log in.
  + If a session exists, the username is displayed, and a logout button is provided.
* The logout function allows the user to clear the session and log out.
* The component is styled with the CSS classes container, dashboard, and logout-btn, which are assumed to be defined in the App.css file.

**Home Component**

The Home component will serve as the landing page.

// src/components/Home.js

import React from 'react';

import '../App.css'; // Assuming you're importing your global CSS

const Home = () => (

  <div className='container home'>

    <h2>Home</h2>

    <p>Welcome to the session management demo!</p>

  </div>

);

export default Home;

**Step 4: Set Up Routing**

Update the App.js to handle routing between the Home, Login, and Dashboard components.

// src/App.js

import React from 'react';

import { BrowserRouter as Router, Routes, Route, Link, Navigate } from 'react-router-dom';

import { SessionProvider, useSession } from './SessionContext';

import Home from './components/Home';

import Login from './components/Login';

import Dashboard from './components/Dashboard';

// Private Route component

const PrivateRoute = ({ element, ...rest }) => {

  const { session } = useSession();

  return session ? element : <Navigate to="/login" />;

};

// Header Component for navigation

const Header = () => {

  const { session, logout } = useSession();

  return (

    <header>

      <nav>

        <ul>

          <li>

            <Link to="/">Home</Link>

          </li>

          {!session && (

            <li>

              <Link to="/login">Login</Link>

            </li>

          )}

          {session && (

            <li>

              <Link to="/dashboard">Dashboard</Link>

            </li>

          )}

          {session && (

            <li>

              <button onClick={logout}>Logout</button>

            </li>

          )}

        </ul>

      </nav>

    </header>

  );

};

function App() {

  return (

    <SessionProvider>

      <Router>

        <Header />

        <Routes>

          <Route path="/" element={<Home />} />

          <Route path="/login" element={<Login />} />

          <Route path="/dashboard" element={<PrivateRoute element={<Dashboard />} />} />

        </Routes>

      </Router>

    </SessionProvider>

  );

}

export default App;

Let's go through the App.js file:

**import React from 'react';**

**import { BrowserRouter as Router, Routes, Route, Link, Navigate } from 'react-router-dom';**

**import { SessionProvider, useSession } from './SessionContext';**

**import Home from './components/Home';**

**import Login from './components/Login';**

**import Dashboard from './components/Dashboard';**

* **Imports**:
  + React: Importing the React library to use JSX.
  + Router, Routes, Route, Link, Navigate: Components from react-router-dom for routing and navigation.
  + SessionProvider, useSession: Custom context and hook for session management.
  + Home, Login, Dashboard: Importing the components for different pages in the app.

**const PrivateRoute = ({ element, ...rest }) => {**

**const { session } = useSession();**

**return session ? element : <Navigate to="/login" />;**

**};**

* **PrivateRoute Component**:
  + This component is used to protect routes that require authentication.
  + element: The component to render if the user is authenticated.
  + useSession(): A custom hook to get the current session state.
  + If session exists (i.e., the user is authenticated), it renders the element.
  + If not, it redirects the user to the /login page using the Navigate component.

**const Header = () => {**

**const { session, logout } = useSession();**

**return (**

**<header>**

**<nav>**

**<ul>**

**<li>**

**<Link to="/">Home</Link>**

**</li>**

**{!session && (**

**<li>**

**<Link to="/login">Login</Link>**

**</li>**

**)}**

**{session && (**

**<li>**

**<Link to="/dashboard">Dashboard</Link>**

**</li>**

**)}**

**{session && (**

**<li>**

**<button onClick={logout}>Logout</button>**

**</li>**

**)}**

**</ul>**

**</nav>**

**</header>**

**);**

**};**

* **Header Component**:
  + This component contains navigation links and a logout button.
  + useSession(): Used to get session state and the logout function.
  + nav element includes links to the Home, Login, and Dashboard pages, depending on whether a session exists.
  + If the user is logged in (session exists), the Dashboard link and Logout button are shown.
  + If not logged in, only the Home and Login links are shown.

**function App() {**

**return (**

**<SessionProvider>**

**<Router>**

**<Header />**

**<Routes>**

**<Route path="/" element={<Home />} />**

**<Route path="/login" element={<Login />} />**

**<Route path="/dashboard" element={<PrivateRoute element={<Dashboard />} />} />**

**</Routes>**

**</Router>**

**</SessionProvider>**

**);**

**}**

* **App Component**:
  + **SessionProvider**: Wraps the app to provide session context to its children.
  + **Router**: Manages routing for the application.
  + **Header**: Renders the navigation bar at the top of the page.
  + **Routes**: Defines the routing for different paths:
    - path="/": Renders the Home component for the home page.
    - path="/login": Renders the Login component for the login page.
    - path="/dashboard": Uses PrivateRoute to render Dashboard if the user is authenticated, otherwise redirects to /login.

**export default App;**

* **Export**: Exports the App component as the default export so it can be used in other parts of the application.

This setup ensures that navigation is properly handled and that protected routes are only accessible to authenticated users.

**Step 5: Run the Application**

Start the development server to see the application in action.

npm start

**Summary of Functionality**

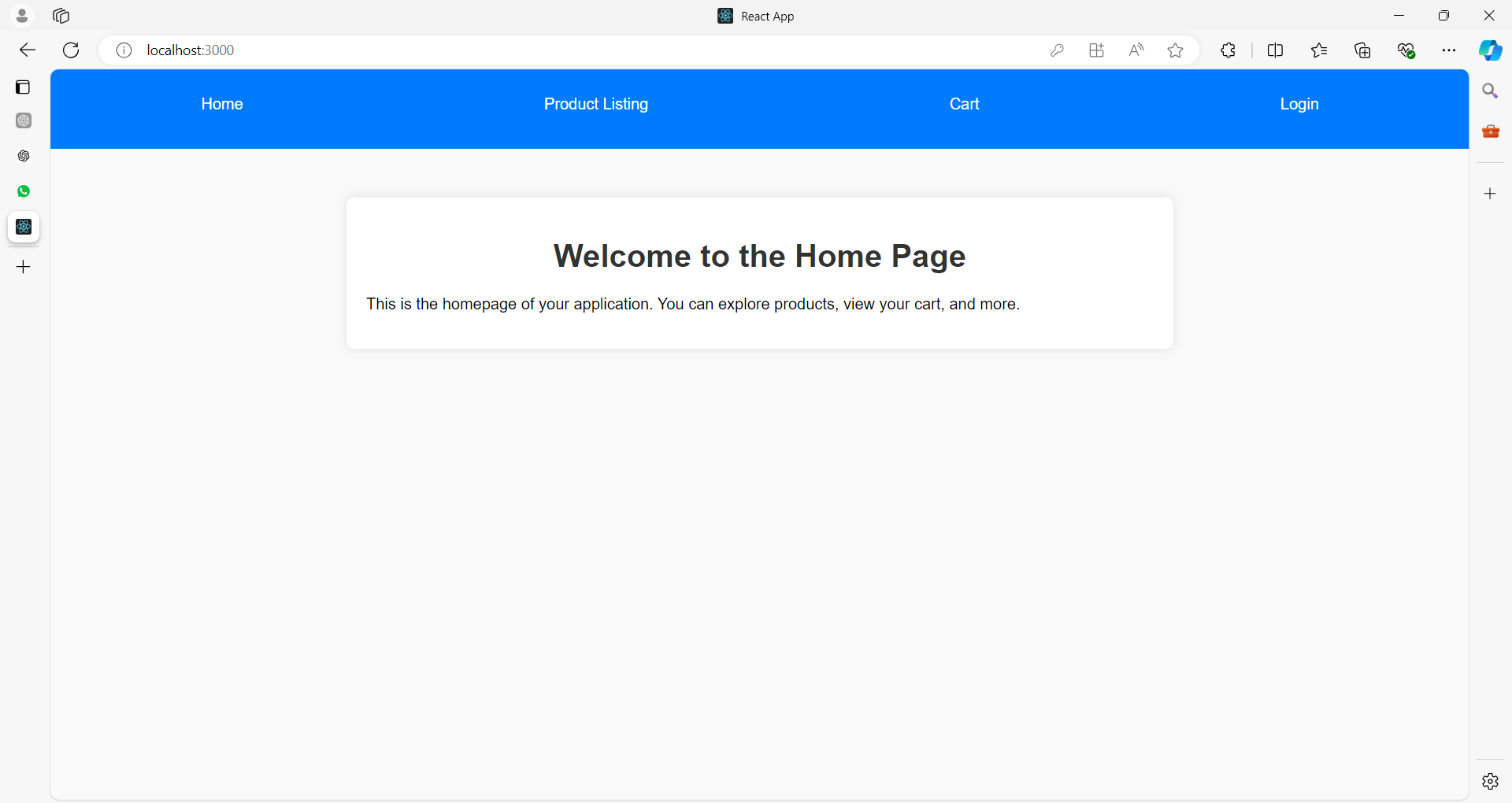
* **Home Page:** The landing page of the application.
* **Login Page:** Allows the user to log in by entering a username. After login, the session data is stored in the context and local storage.
* **Dashboard Page:** Displays a welcome message with the username and provides a logout option. Access to the dashboard is restricted to logged-in users (protected route).
* **Session Persistence:** The user remains logged in even after refreshing the page, thanks to session data being stored in local storage.

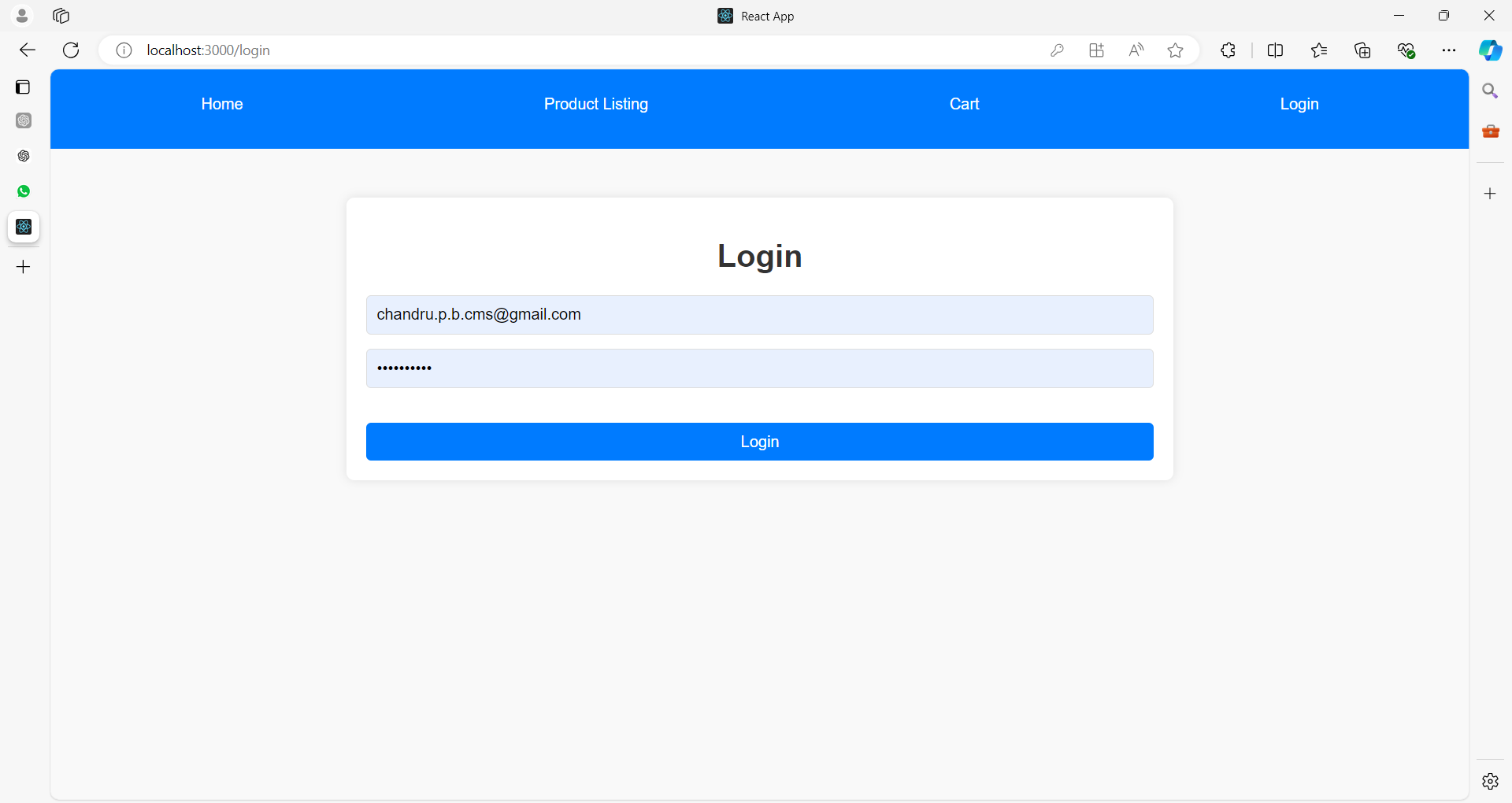
This is a basic implementation of session management in React using the Context API and local storage. You can expand this by adding more features like session expiration, secure token handling, and integration with a backend authentication service.

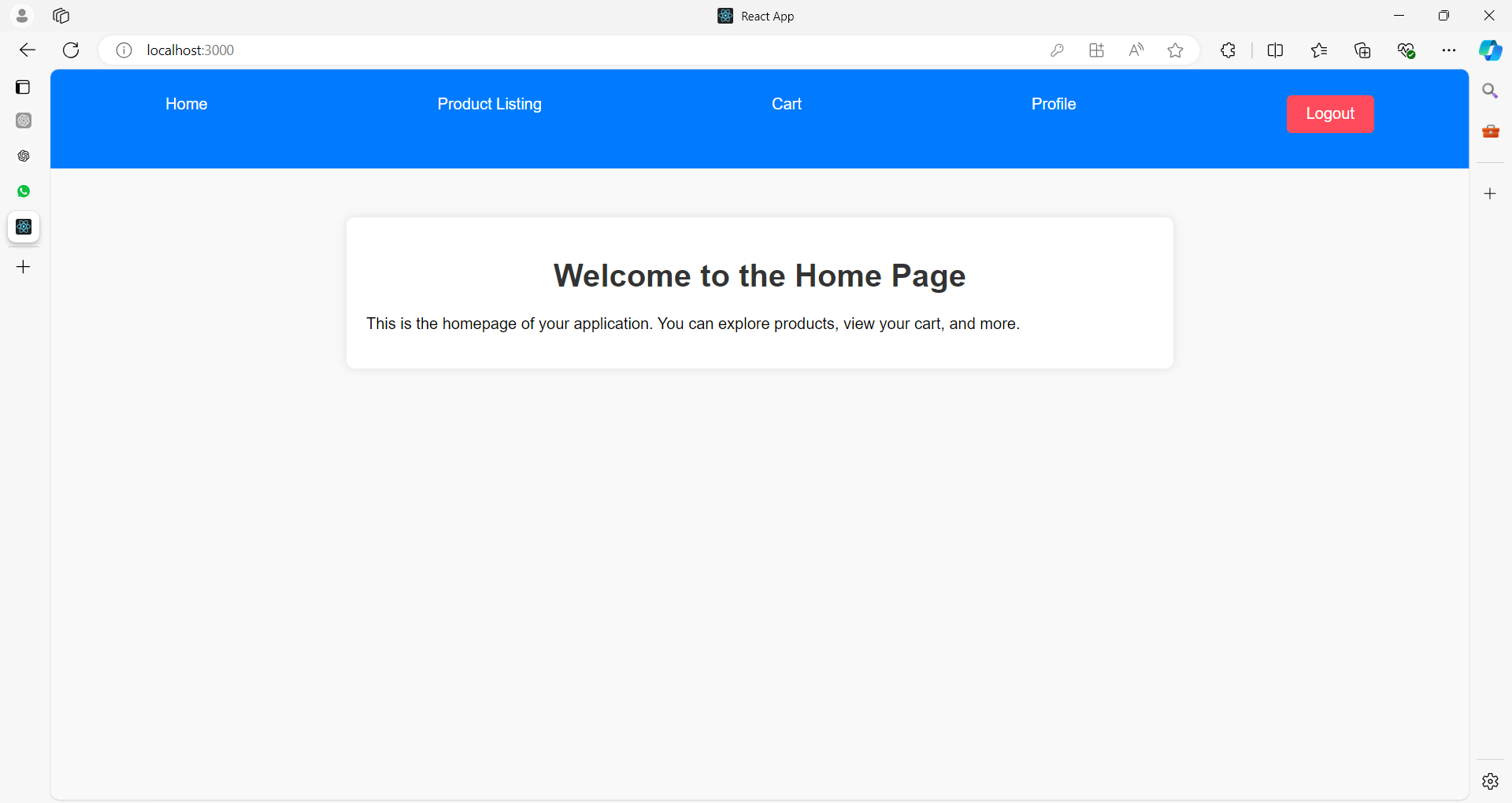
**Example 02**

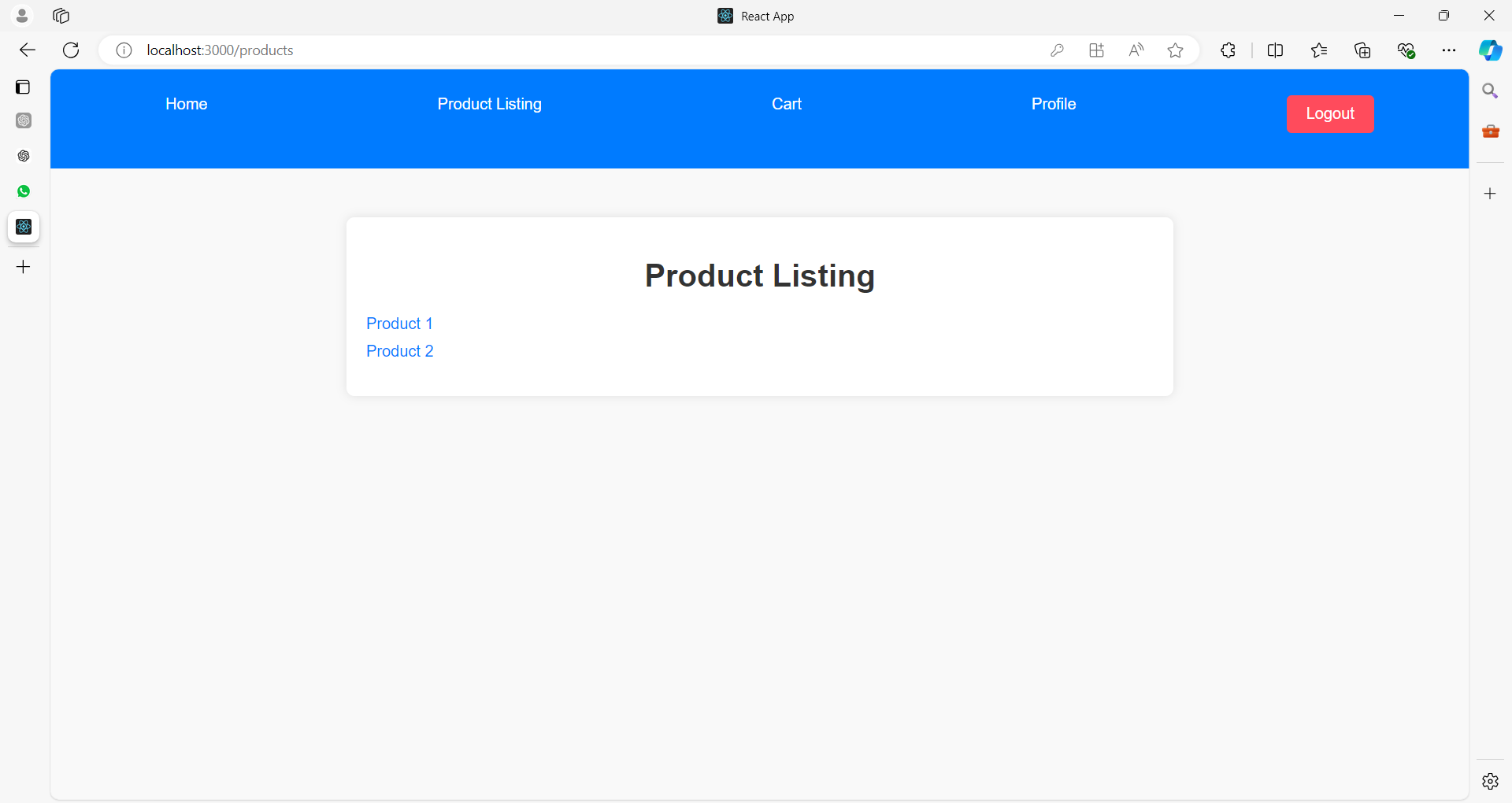
Let's create a more comprehensive eCommerce application using React that includes:

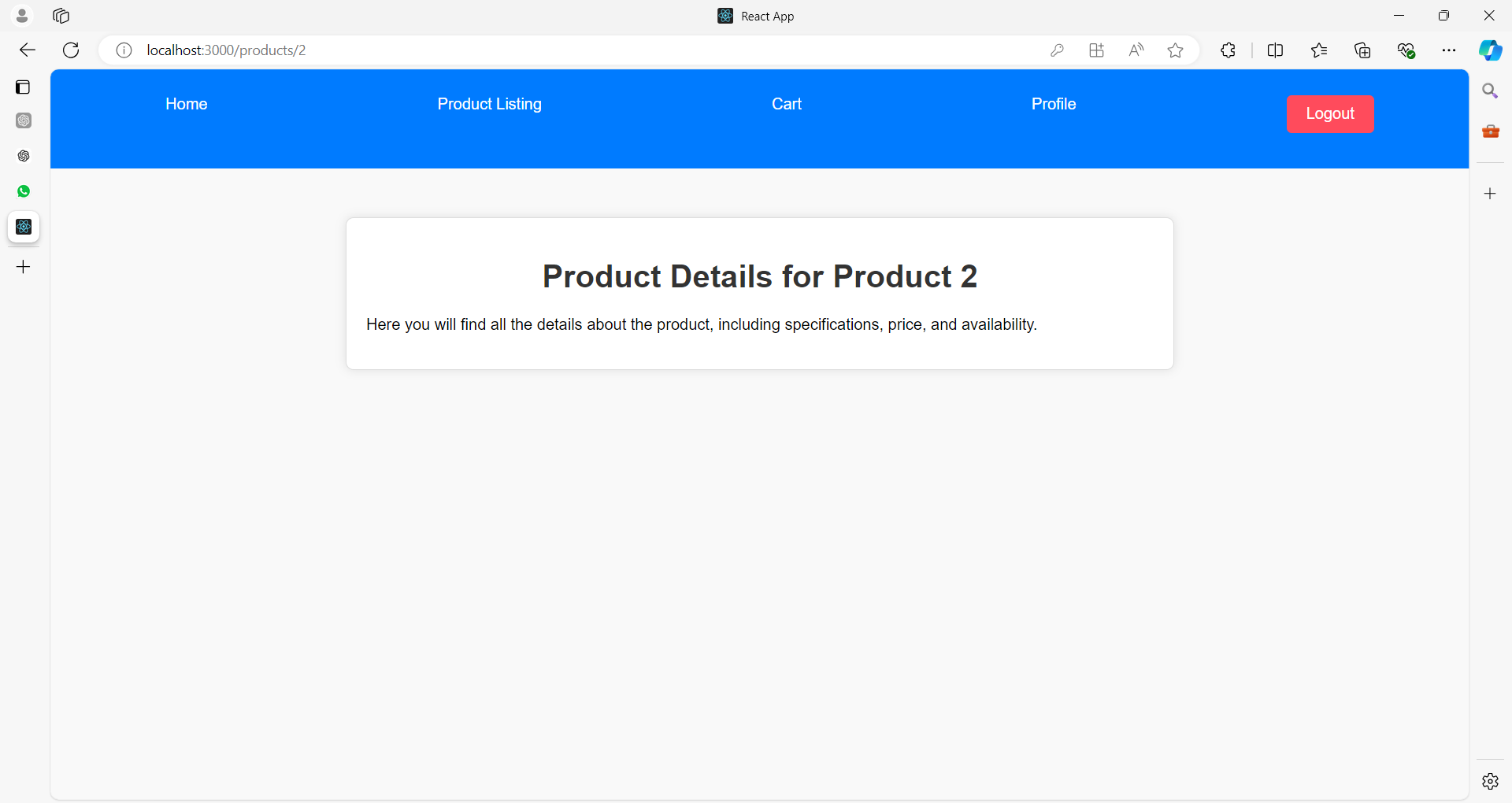
* JSON files to store user data.
* Routing for different pages (e.g., Home, Product Listing, Cart, Profile).
* Session management for login and logout.
* Basic eCommerce functionalities.

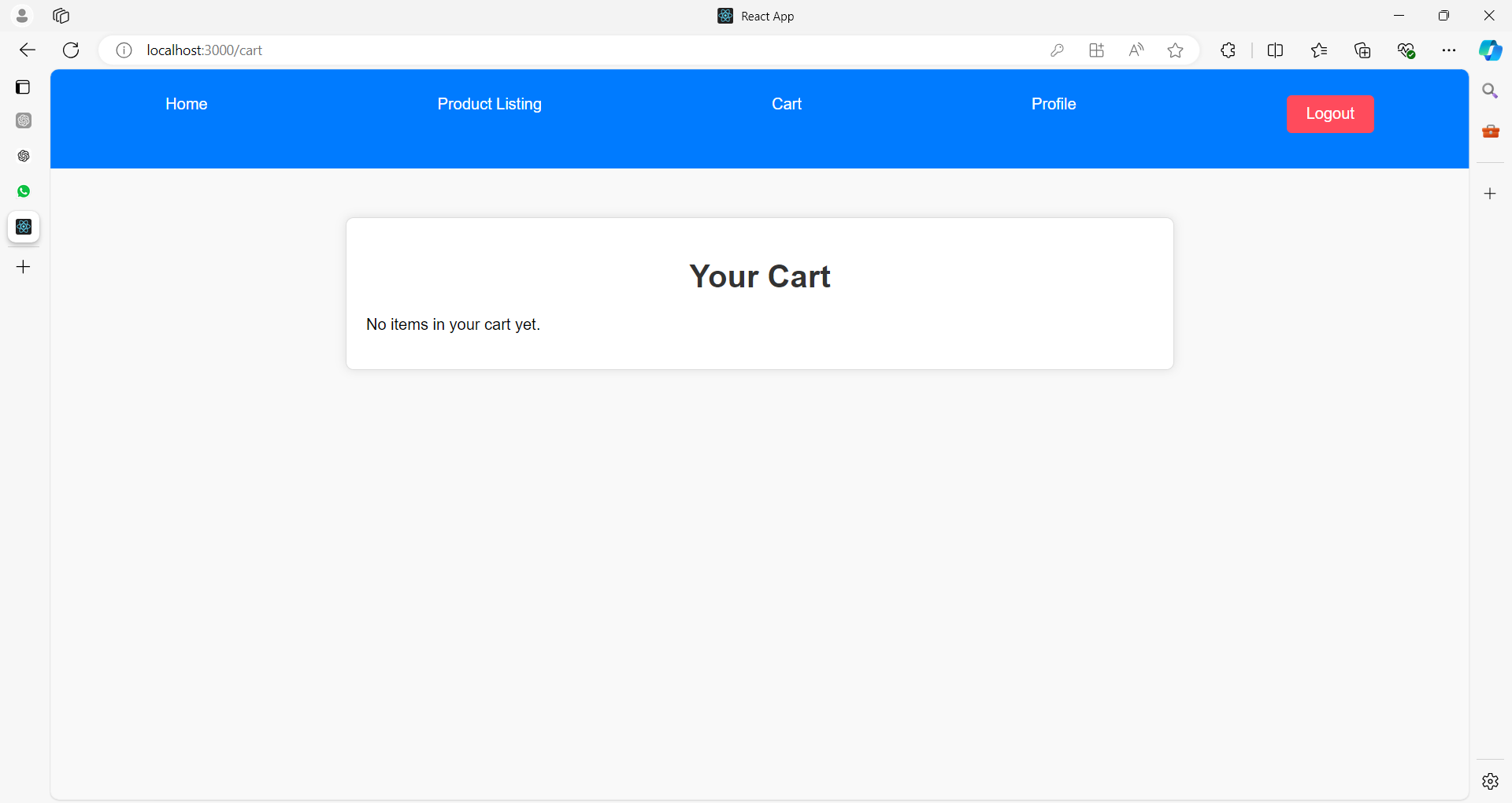


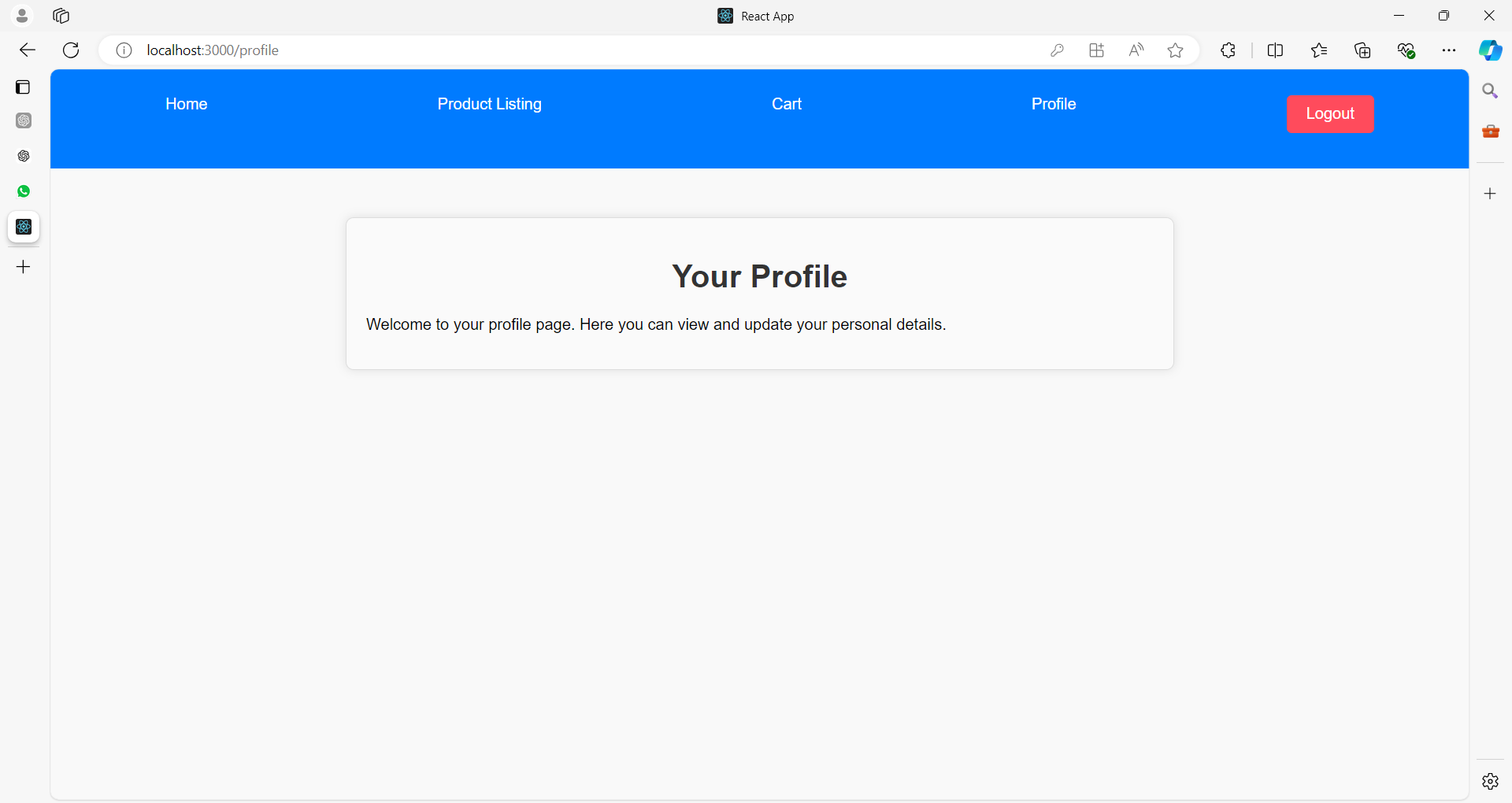




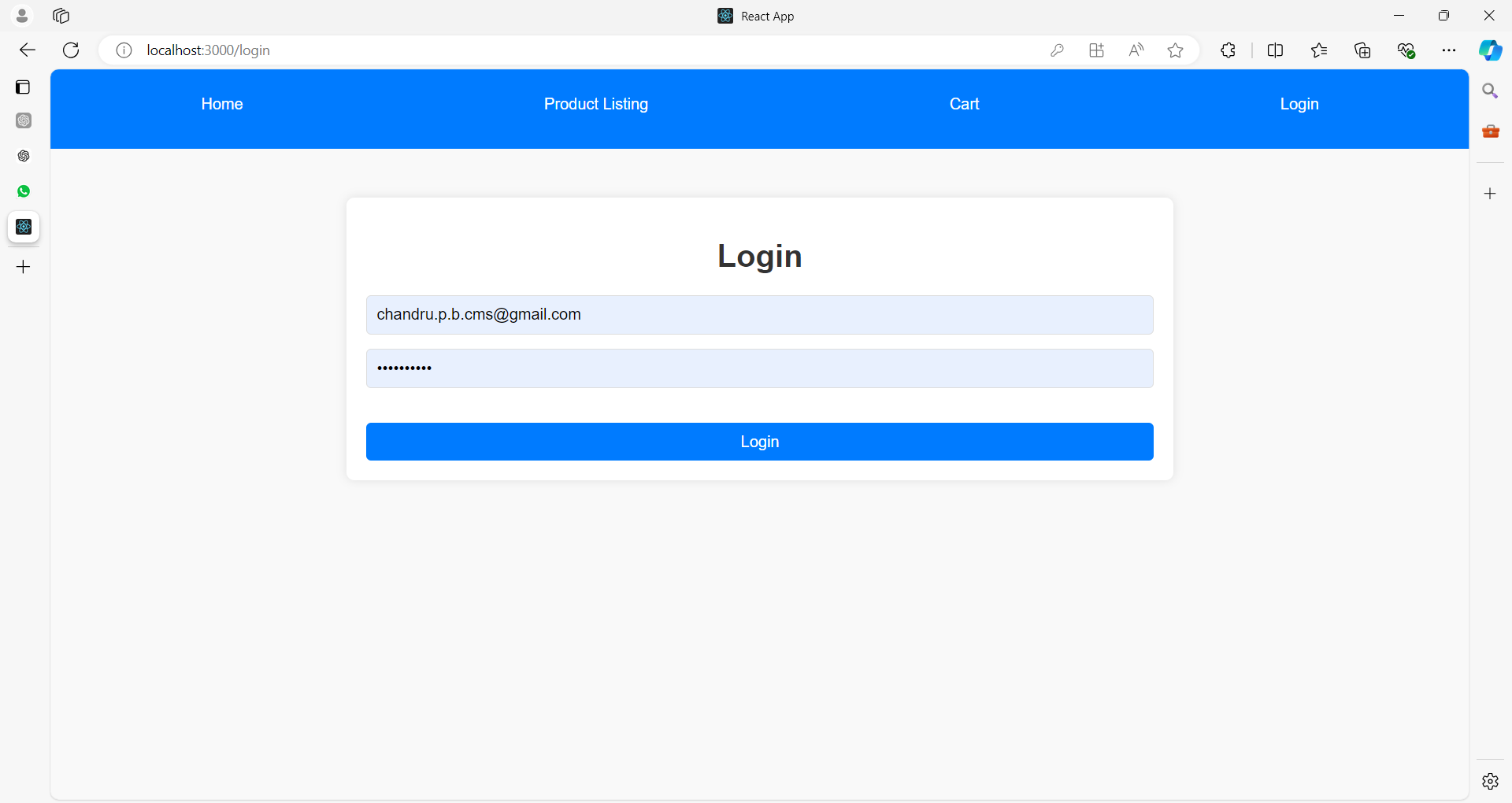








**After clicking on Logout button**



**Project Structure**

Here's a high-level overview of the project structure:

**ecommerce-app/**

**│**

**├── public/**

**│ └── users.json**

**│**

**├── src/**

**│ ├── components/**

**│ │ ├── Home.js**

**│ │ ├── ProductListing.js**

**│ │ ├── ProductDetails.js**

**│ │ ├── Cart.js**

**│ │ ├── Profile.js**

**│ │ ├── Login.js**

**│ │ ├── Logout.js**

**│ │**

**│ ├── css/**

**│ │ ├── Home.css**

**│ │ ├── ProductListing.css**

**│ │ ├── ProductDetails.css**

**│ │ ├── Cart.css**

**│ │ ├── Profile.css**

**│ │ ├── Login.css**

**│ │ └── Logout.css**

**│ │**

**│ ├── context/**

**│ │ └── SessionContext.js**

**│ │**

**│ ├── App.js**

**│ ├── index.js**

**│ └── routes/**

**│ └── PrivateRoute.js**

**│**

**└── package.json**

**Step 1: Set Up the JSON Data**

First, create a **users.json** file in the public/ directory to simulate a database.

// public/users.json

[

{

"id": 1,

"username": "john\_doe",

"password": "password123",

"email": "john@example.com",

"cart": []

},

{

"id": 2,

"username": "jane\_doe",

"password": "password456",

"email": "jane@example.com",

"cart": []

},

{

"id": 3,

"username": "rohit\_sharma",

"password": "rohit789",

"email": "rohit@india.com",

"cart": []

},

{

"id": 4,

"username": "anita\_mishra",

"password": "anita321",

"email": "anita@india.com",

"cart": []

},

{

"id": 5,

"username": "emma\_james",

"password": "emma456",

"email": "emma@example.com",

"cart": []

},

{

"id": 6,

"username": "liam\_smith",

"password": "liam789",

"email": "liam@example.com",

"cart": []

},

{

"id": 7,

"username": "raj\_kumar",

"password": "raj1234",

"email": "raj@india.com",

"cart": []

},

{

"id": 8,

"username": "neha\_patel",

"password": "neha567",

"email": "neha@india.com",

"cart": []

},

{

"id": 9,

"username": "oliver\_brown",

"password": "oliver890",

"email": "oliver@example.com",

"cart": []

},

{

"id": 10,

"username": "sophia\_johnson",

"password": "sophia234",

"email": "sophia@example.com",

"cart": []

},

{

"id": 11,

"username": "arjun\_singh",

"password": "arjun345",

"email": "arjun@india.com",

"cart": []

},

{

"id": 12,

"username": "priya\_sharma",

"password": "priya678",

"email": "priya@india.com",

"cart": []

},

{

"id": 13,

"username": "michael\_clark",

"password": "michael901",

"email": "michael@example.com",

"cart": []

},

{

"id": 14,

"username": "ella\_davis",

"password": "ella345",

"email": "ella@example.com",

"cart": []

},

{

"id": 15,

"username": "nina\_verma",

"password": "nina5678",

"email": "nina@india.com",

"cart": []

},

{

"id": 16,

"username": "ajay\_mehta",

"password": "ajay1234",

"email": "ajay@india.com",

"cart": []

},

{

"id": 17,

"username": "madison\_lee",

"password": "madison678",

"email": "madison@example.com",

"cart": []

},

{

"id": 18,

"username": "jason\_wilson",

"password": "jason7890",

"email": "jason@example.com",

"cart": []

},

{

"id": 19,

"username": "sonia\_gupta",

"password": "sonia2345",

"email": "sonia@india.com",

"cart": []

},

{

"id": 20,

"username": "vivek\_jain",

"password": "vivek6789",

"email": "vivek@india.com",

"cart": []

}

]

**Step 2: Create the Session Context**

Below is a simplified React application that demonstrates session management.

This example includes basic components for Home, Product Listing, Product Details, Cart, Profile, Login, and Logout functionalities.

The app uses **React Context** to manage user sessions.

**1. src/context/SessionContext.js**

This file provides session context for the entire application.

import React, { createContext, useState, useContext }

from 'react';

// Create a Context for session management

const SessionContext = createContext();

// Create a provider component

export const SessionProvider = ({ children }) => {

  const [session, setSession] = useState(null);

  const login = (user) => {

    setSession(user);

  };

  const logout = () => {

    setSession(null);

  };

  return (

    <SessionContext.Provider value={{ session, login, logout }}>

      {children}

    </SessionContext.Provider>

  );

};

// Custom hook to use session context

export const useSession = () => {

  return useContext(SessionContext);

};

**2. src/routes/PrivateRoute.js**

A route component that only allows access if the user is authenticated.

import React from 'react';

import { Navigate } from 'react-router-dom';

import { useSession } from '../context/SessionContext';

const PrivateRoute = ({ element }) => {

  const { session } = useSession();

  return session ? element : <Navigate to="/login" />;

};

export default PrivateRoute;

**3. src/App.js**

The main app component which includes routing and header.

import React from 'react';

import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';

import { SessionProvider, useSession } from './context/SessionContext';

import Home from './components/Home';

import ProductListing from './components/ProductListing';

import ProductDetails from './components/ProductDetails';

import Cart from './components/Cart';

import Profile from './components/Profile';

import Login from './components/Login';

import Logout from './components/Logout';

import PrivateRoute from './routes/PrivateRoute';

const Header = () => {

  const { session, logout } = useSession();

  return (

    <header>

      <nav>

        <ul>

          <li><Link to="/">Home</Link></li>

          <li><Link to="/products">Product Listing</Link></li>

          <li><Link to="/cart">Cart</Link></li>

{session ? (

            <>

              <li><Link to="/profile">Profile</Link></li>

              <li><button onClick={logout}>Logout</button></li>

            </>

          ) : (

            <li><Link to="/login">Login</Link></li>

          )}

        </ul>

      </nav>

    </header>

  );

};

function App() {

  return (

    <SessionProvider>

      <Router>

        <Header />

        <Routes>

          <Route path="/" element={<Home />} />

          <Route path="/products" element={<ProductListing />} />

          <Route path="/products/:id" element={<ProductDetails />} />

          <Route path="/cart" element={<PrivateRoute element={<Cart />} />} />

          <Route path="/profile" element={<PrivateRoute element={<Profile />} />} />

          <Route path="/login" element={<Login />} />

          <Route path="/logout" element={<Logout />} />

        </Routes>

      </Router>

    </SessionProvider>

  );

}

export default App;

**4. Component Files**

Here’s a brief outline of each component:

**src/components/Home.js**

import React from 'react';

import '../App.css'; // Import the CSS file

const Home = () => {

  return (

    <div className="container">

      <h1 className="text-center">Welcome to the Home Page</h1>

      <p className="mt-20">

        This is the homepage of your application. You can explore products, view your cart, and more.

      </p>

    </div>

  );

};

export default Home;

**src/components/ProductListing.js**

import React from 'react';

import { Link } from 'react-router-dom';

import '../App.css'; // Import the CSS file

const ProductListing = () => {

  return (

    <div className="container">

      <h1 className="text-center">Product Listing</h1>

      <ul className="mt-20">

        {/\* Sample product links \*/}

        <li><Link to="/products/1">Product 1</Link></li>

        <li><Link to="/products/2">Product 2</Link></li>

      </ul>

    </div>

  );

};

export default ProductListing;

**src/components/ProductDetails.js**

import React from 'react';

import { useParams } from 'react-router-dom';

import '../App.css'; // Import the CSS file

const ProductDetails = () => {

  const { id } = useParams();

  return (

    <div className="container product-details">

      <h1 className="text-center">Product Details for Product {id}</h1>

      <p className="mt-20">

        Here you will find all the details about the product, including specifications, price, and availability.

      </p>

      {/\* Additional product details would be dynamically displayed here \*/}

    </div>

  );

};

export default ProductDetails;

**src/components/Cart.js**

import React from 'react';

import '../App.css'; // Ensure you import the CSS file

const Cart = () => {

  return (

    <div className="container cart-section">

      <h1 className="text-center">Your Cart</h1>

      <p className="mt-20">No items in your cart yet.</p>

      {/\* Cart items would be dynamically displayed here \*/}

    </div>

  );

};

export default Cart;

**src/components/Profile.js**

import React from 'react';

import '../App.css'; // Import the CSS file

const Profile = () => {

  return (

    <div className="container profile-section">

      <h1 className="text-center">Your Profile</h1>

      <p className="mt-20">

        Welcome to your profile page. Here you can view and update your personal details.

      </p>

      {/\* Profile information would be dynamically displayed here \*/}

    </div>

  );

};

export default Profile;

**src/components/Login.js**

import React, { useState } from 'react';

import { useNavigate } from 'react-router-dom';

import { useSession } from '../context/SessionContext';

import '../App.css'; // Import the CSS file

const Login = () => {

  const [username, setUsername] = useState('');

  const [password, setPassword] = useState('');

  const { login } = useSession();

  const navigate = useNavigate();

  const handleLogin = () => {

    // Perform authentication (mocked here)

    login({ username });

    navigate('/');

  };

  return (

    <div className="container">

      <h1 className="text-center">Login</h1>

      <form>

        <input

          type="text"

          placeholder="Username"

          value={username}

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

        />

        <input

          type="password"

          placeholder="Password"

          value={password}

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

        />

        <button type="button" onClick={handleLogin} className="mt-20">Login</button>

      </form>

    </div>

  );

};

export default Login;

**src/components/Logout.js**

import React, { useEffect } from 'react';

import { useNavigate } from 'react-router-dom';

import { useSession } from '../context/SessionContext';

import '../App.css'; // Import the CSS file

const Logout = () => {

  const { logout } = useSession();

  const navigate = useNavigate();

  useEffect(() => {

    logout();

    navigate('/');

  }, [logout, navigate]);

  return (

    <div className="container text-center">

      <h1>Logging out...</h1>

    </div>

  );

};

export default Logout;

**Explanation of the Components and Routing**

* **Home Component:** Displays a welcome message.
* **ProductListing Component:** Lists all products with links to product details.
* **ProductDetails Component:** Shows details of a single product.
* **Cart Component:** A placeholder for the cart functionality (could be expanded later).
* **Profile Component:** Shows user information if logged in.
* **Login Component:** Allows users to log in by matching credentials from users.json.
* **Logout Component:** Logs out the user and redirects them to the home page.
* **PrivateRoute Component:** A route wrapper that ensures users are authenticated before accessing certain routes.

**Step 6: Run Your Application**

Make sure all dependencies are installed and start your development server:

npm install

npm start

Visit http://localhost:3000 in your browser to interact with the application. You can navigate between different pages, log in, and see how session management works.

**Summary**

In this application:

* **Session Management:** Handled by React Context API and local storage to maintain user login state across page reloads.
* **Routing:** Uses React Router for navigation between different pages.
* **JSON Data:** Simulates user data storage and retrieval.

This example provides a solid foundation for an eCommerce application with basic session management. You can further expand this by adding functionalities such as:

* **Cart Management:** Allow users to add products to their cart and view cart details.
* **Product Management:** Fetch product data from an API or a more complex JSON file.
* **User Registration:** Add functionality for new users to register.
* **Order Management:** Allow users to place and view orders.

**Example 03**

Same example with cookies

To integrate cookies for session management in the eCommerce React application, we'll use cookies to store session data (such as authentication tokens). Here’s how you can modify the example to use cookies instead of local storage.

**Step 1: Install Cookie Handling Library**

First, you need a library to handle cookies. js-cookie is a popular choice.

npm install js-cookie

**Step 2: Update the Session Context**

Modify SessionContext.js to use cookies for storing session data.

// src/context/SessionContext.js

import React, { createContext, useState, useEffect, useContext } from 'react';

import Cookies from 'js-cookie';

const SessionContext = createContext();

export const SessionProvider = ({ children }) => {

const [session, setSession] = useState(null);

useEffect(() => {

const storedSession = Cookies.get('session');

if (storedSession) {

setSession(JSON.parse(storedSession));

}

}, []);

useEffect(() => {

if (session) {

Cookies.set('session', JSON.stringify(session), { expires: 1 }); // Cookie expires in 1 day

} else {

Cookies.remove('session');

}

}, [session]);

const login = (user) => setSession(user);

const logout = () => setSession(null);

return (

<SessionContext.Provider value={{ session, login, logout }}>

{children}

</SessionContext.Provider>

);

};

export const useSession = () => useContext(SessionContext);

**Step 3: Update the Login Component**

Make sure the Login component uses the updated session management logic.

// src/components/Login.js

import React, { useState } from 'react';

import { useHistory } from 'react-router-dom';

import { useSession } from '../context/SessionContext';

const Login = () => {

const [username, setUsername] = useState('');

const [password, setPassword] = useState('');

const { login } = useSession();

const history = useHistory();

const handleLogin = async (e) => {

e.preventDefault();

// Fetch users from the JSON file

const response = await fetch('/users.json');

const users = await response.json();

const user = users.find(

(u) => u.username === username && u.password === password

);

if (user) {

login(user);

history.push('/profile');

} else {

alert('Invalid username or password');

}

};

return (

<div>

<h2>Login</h2>

<form onSubmit={handleLogin}>

<input

type="text"

placeholder="Username"

value={username}

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

/>

<input

type="password"

placeholder="Password"

value={password}

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

/>

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

</form>

</div>

);

};

export default Login;

**Step 4: Update the Logout Component**

Ensure the Logout component clears the session cookie when logging out.

// src/components/Logout.js

import React, { useEffect } from 'react';

import { useSession } from '../context/SessionContext';

import { useHistory } from 'react-router-dom';

const Logout = () => {

const { logout } = useSession();

const history = useHistory();

useEffect(() => {

logout();

history.push('/');

}, [logout, history]);

return <div>You have been logged out.</div>;

};

export default Logout;

**Step 5: Update Routing and Other Components**

The rest of the application remains unchanged. Ensure that your App.js and other components use the updated SessionContext for session management.

Here’s a summary of what you need to check:

* **Home Component:** Remains unchanged.
* **ProductListing Component:** Remains unchanged.
* **ProductDetails Component:** Remains unchanged.
* **Cart Component:** Remains unchanged.
* **Profile Component:** Remains unchanged.
* **Login Component:** Updated to handle cookies.
* **Logout Component:** Updated to handle cookies.

**Step 6: Test the Application**

Make sure all dependencies are installed and start your development server:

npm install

npm start

Visit http://localhost:3000 in your browser. You should be able to log in, log out, and see how session management works with cookies.

**Summary**

In this updated example, we've replaced local storage with cookies for session management:

* **Cookies:** Used for storing session data (authentication tokens) with expiration settings.
* **Session Context:** Updated to use js-cookie for setting and removing cookies.
* **Login/Logout:** Adjusted to handle cookies properly.

This approach helps manage sessions more securely and is suitable for many applications that require persistent sessions across page reloads.

**Example 04**

Same example with redux

Let's refactor the eCommerce application to use Redux for state management instead of the Context API. Redux will handle the session management, product listings, and cart state. Here's how you can structure and implement the application with Redux:

**Project Structure**

The project structure will be similar to the previous setup, with added Redux-specific files:

ecommerce-app/

│

├── public/

│ └── users.json

│

├── src/

│ ├── components/

│ │ ├── Home.js

│ │ ├── ProductListing.js

│ │ ├── ProductDetails.js

│ │ ├── Cart.js

│ │ ├── Profile.js

│ │ ├── Login.js

│ │ └── Logout.js

│ │

│ ├── redux/

│ │ ├── actions/

│ │ │ ├── sessionActions.js

│ │ │ └── cartActions.js

│ │ ├── reducers/

│ │ │ ├── sessionReducer.js

│ │ │ ├── cartReducer.js

│ │ │ └── rootReducer.js

│ │ └── store.js

│ │

│ ├── App.js

│ ├── index.js

│ └── routes/

│ └── PrivateRoute.js

└── package.json

**Step 1: Install Redux and React-Redux**

First, install Redux and React-Redux.

npm install redux react-redux

**Step 2: Set Up Redux**

Create the Redux store, actions, and reducers.

**Store Configuration**

Create store.js to configure the Redux store.

// src/redux/store.js

import { createStore, combineReducers } from 'redux';

import sessionReducer from './reducers/sessionReducer';

import cartReducer from './reducers/cartReducer';

const rootReducer = combineReducers({

session: sessionReducer,

cart: cartReducer,

});

const store = createStore(rootReducer);

export default store;

**Session Actions**

Define actions for managing the session.

// src/redux/actions/sessionActions.js

export const LOGIN = 'LOGIN';

export const LOGOUT = 'LOGOUT';

export const login = (user) => ({

type: LOGIN,

payload: user,

});

export const logout = () => ({

type: LOGOUT,

});

**Cart Actions**

Define actions for managing the cart (for simplicity, we'll add products to the cart).

// src/redux/actions/cartActions.js

export const ADD\_TO\_CART = 'ADD\_TO\_CART';

export const REMOVE\_FROM\_CART = 'REMOVE\_FROM\_CART';

export const addToCart = (product) => ({

type: ADD\_TO\_CART,

payload: product,

});

export const removeFromCart = (productId) => ({

type: REMOVE\_FROM\_CART,

payload: productId,

});

**Session**

**Session Reducer**

Create a reducer to handle session-related actions.

// src/redux/reducers/sessionReducer.js

import { LOGIN, LOGOUT } from '../actions/sessionActions';

const initialState = {

user: null,

};

const sessionReducer = (state = initialState, action) => {

switch (action.type) {

case LOGIN:

return {

...state,

user: action.payload,

};

case LOGOUT:

return {

...state,

user: null,

};

default:

return state;

}

};

export default sessionReducer;

**Cart Reducer**

Create a reducer to handle cart-related actions.

// src/redux/reducers/cartReducer.js

import { ADD\_TO\_CART, REMOVE\_FROM\_CART } from '../actions/cartActions';

const initialState = {

items: [],

};

const cartReducer = (state = initialState, action) => {

switch (action.type) {

case ADD\_TO\_CART:

return {

...state,

items: [...state.items, action.payload],

};

case REMOVE\_FROM\_CART:

return {

...state,

items: state.items.filter(item => item.id !== action.payload),

};

default:

return state;

}

};

export default cartReducer;

**Root Reducer**

Combine reducers if you have multiple reducers.

// src/redux/reducers/rootReducer.js

import { combineReducers } from 'redux';

import sessionReducer from './sessionReducer';

import cartReducer from './cartReducer';

const rootReducer = combineReducers({

session: sessionReducer,

cart: cartReducer,

});

export default rootReducer;

**Step 3: Update index.js to Provide the Store**

Wrap the application with the Redux Provider.

// src/index.js

import React from 'react';

import ReactDOM from 'react-dom';

import { Provider } from 'react-redux';

import store from './redux/store';

import App from './App';

ReactDOM.render(

<Provider store={store}>

<App />

</Provider>,

document.getElementById('root')

);

**Step 4: Update Components to Use Redux**

**Login Component**

Update the Login component to dispatch login actions.

// src/components/Login.js

import React, { useState } from 'react';

import { useDispatch } from 'react-redux';

import { useHistory } from 'react-router-dom';

import { login } from '../redux/actions/sessionActions';

const Login = () => {

const [username, setUsername] = useState('');

const [password, setPassword] = useState('');

const dispatch = useDispatch();

const history = useHistory();

const handleLogin = async (e) => {

e.preventDefault();

// Fetch users from the JSON file

const response = await fetch('/users.json');

const users = await response.json();

const user = users.find(

(u) => u.username === username && u.password === password

);

if (user) {

dispatch(login(user));

history.push('/profile');

} else {

alert('Invalid username or password');

}

};

return (

<div>

<h2>Login</h2>

<form onSubmit={handleLogin}>

<input

type="text"

placeholder="Username"

value={username}

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

/>

<input

type="password"

placeholder="Password"

value={password}

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

/>

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

</form>

</div>

);

};

export default Login;

**Logout Component**

Update the Logout component to dispatch logout actions.

// src/components/Logout.js

import React, { useEffect } from 'react';

import { useDispatch } from 'react-redux';

import { useHistory } from 'react-router-dom';

import { logout } from '../redux/actions/sessionActions';

const Logout = () => {

const dispatch = useDispatch();

const history = useHistory();

useEffect(() => {

dispatch(logout());

history.push('/');

}, [dispatch, history]);

return <div>You have been logged out.</div>;

};

export default Logout;

**Profile Component**

Update the Profile component to use Redux state.

// src/components/Profile.js

import React from 'react';

import { useSelector } from 'react-redux';

const Profile = () => {

const user = useSelector((state) => state.session.user);

if (!user) {

return <div>Please log in to view your profile.</div>;

}

return (

<div>

<h2>Profile</h2>

<p>Username: {user.username}</p>

<p>Email: {user.email}</p>

</div>

);

};

export default Profile;

**Cart Component**

Update the Cart component to use Redux state.

// src/components/Cart.js

import React from 'react';

import { useSelector, useDispatch } from 'react-redux';

import { removeFromCart } from '../redux/actions/cartActions';

const Cart = () => {

const cartItems = useSelector((state) => state.cart.items);

const dispatch = useDispatch();

const handleRemove = (id) => {

dispatch(removeFromCart(id));

};

return (

<div>

<h2>Your Cart</h2>

{cartItems.length === 0 ? (

<p>Your cart is currently empty.</p>

) : (

<ul>

{cartItems.map((item) => (

<li key={item.id}>

{item.name} - ${item.price}

<button onClick={() => handleRemove(item.id)}>Remove</button>

</li>

))}

</ul>

)}

</div>

);

};

export default Cart;

**ProductListing Component**

Update the ProductListing component to include add-to-cart functionality.

// src/components/ProductListing.js

import React from 'react';

import { Link } from 'react-router-dom';

import { useDispatch } from 'react-redux';

import { addToCart } from '../redux/actions/cartActions';

const products = [

{ id: 1, name: 'Product 1', price: 50 },

{ id: 2, name: 'Product 2', price: 100 },

{ id: 3, name: 'Product 3', price: 150 },

];

const ProductListing = () => {

const dispatch = useDispatch();

const handleAddToCart = (product) => {

dispatch(addToCart(product));

};

return (

<div>

<h2>Products</h2>

<ul>

{products.map((product) => (

<li key={product.id}>

<Link to={`/product/${product.id}`}>

{product.name} - ${product.price}

</Link>

<button onClick={() => handleAddToCart(product)}>Add to Cart</button>

</li>

))}

</ul>

</div>

);

};

export default ProductListing;

**ProductDetails Component**

Update the ProductDetails component to use Redux state for cart management.

// src/components/ProductDetails.js

import React from 'react';

import { useParams } from 'react-router-dom';

import { useDispatch } from 'react-redux';

import { addToCart } from '../redux/actions/cartActions';

const products = [

{ id: 1, name: 'Product 1', price: 50 },

{ id: 2, name: 'Product 2', price: 100 },

{ id: 3, name: 'Product 3', price: 150 },

];

const ProductDetails = () => {

const { id } = useParams();

const product = products.find((p) => p.id === parseInt(id));

const dispatch = useDispatch();

const handleAddToCart = () => {

dispatch(addToCart(product));

};

return (

<div>

<h2>{product.name}</h2>

<p>Price: ${product.price}</p>

<button onClick={handleAddToCart}>Add to Cart</button>

</div>

);

};

export default ProductDetails;

**Step 5: Update Routing**

Make sure your App.js includes the updated routes.

// src/App.js

import React from 'react';

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

import Home from './components/Home';

import ProductListing from './components/ProductListing';

import ProductDetails from './components/ProductDetails';

import Cart from './components/Cart';

import Profile from './components/Profile';

import Login from './components/Login';

import Logout from './components/Logout';

import PrivateRoute from './routes/PrivateRoute';

function App() {

return (

<Router>

<Switch>

<Route path="/" exact component={Home} />

<Route path="/products" exact component={ProductListing} />

<Route path="/product/:id" component={ProductDetails} />

<Route path="/cart" component={Cart} />

<Route path="/login" component={Login} />

<Route path="/logout" component={Logout} />

<PrivateRoute path="/profile" component={Profile} />

</Switch>

</Router>

);

}

export default App;

**Summary**

In this Redux-based example:

* **Redux Store:** Manages state for sessions and the cart.
* **Reducers:** Handle session and cart actions.
* **Actions:** Define login, logout, add-to-cart, and remove-from-cart actions.
* **Components:** Use useDispatch and useSelector to interact with Redux state.

You can expand this basic structure to include more features like product management, user registration, and order management.