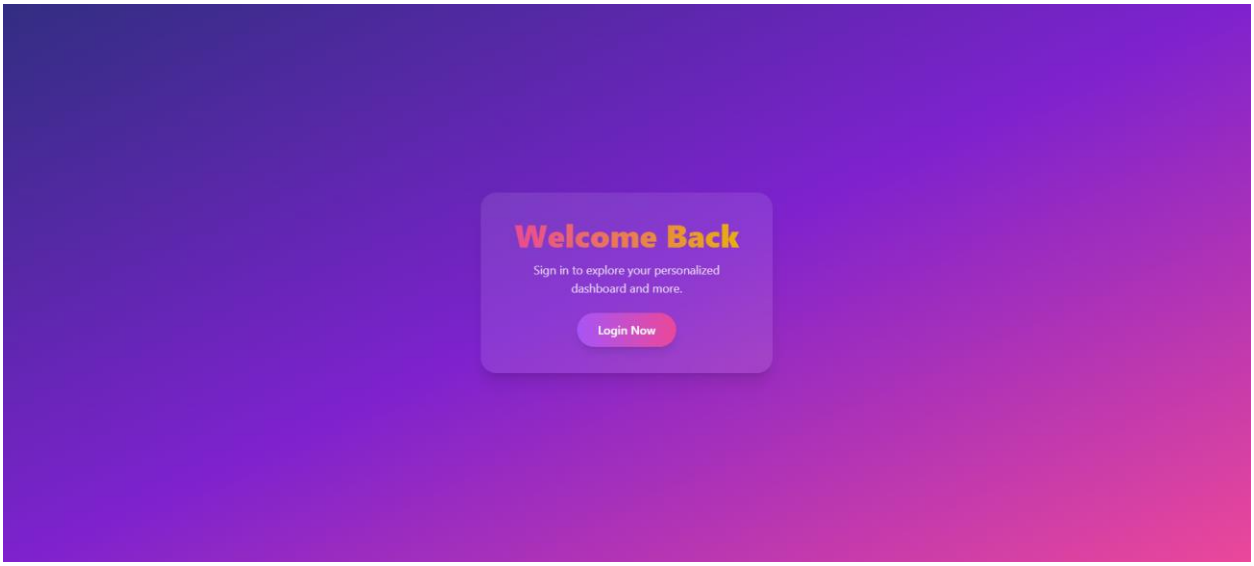
1. **Task Description**

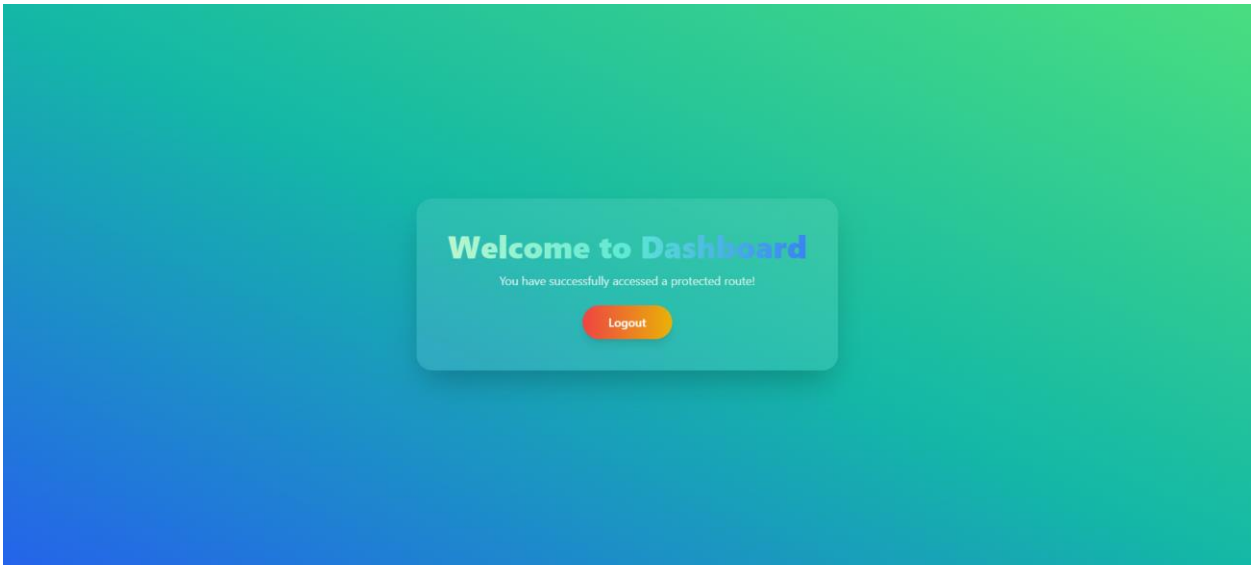
   Develop a role-based authorization system to control access to specific routes or components.

2. **Task Output Screenshot**

   LOGIN PAGE –



   LOGOUT PAGE –



3. Widget/Algorithm Used In Task

1. **BrowserRouter (as Router from react-router-dom):**

- ❖ Wraps the application to provide routing capabilities.
- ❖ Enables navigation between different URL paths without reloading the page.

2. **Routes (from react-router-dom):**

- ➢ A container for Route components to manage routing within the application.

3. **Route (from react-router-dom):**

- ➢ Defines individual routes in the application. Each route specifies:
- ➢ **Path:** The URL to match.
- ➢ **Element:** The component to render when the path matches.

4. **AuthProvider (Custom Context Provider):**

- ➢ Provides authentication state and functionality to all child components using React Context API.

5. **ProtectedRoute (Custom Component):**

- ➢ Wraps restricted components to enforce authentication.
- ➢ Ensures only authenticated users can access specific routes like the Dashboard.

6. **Custom Components/Pages:**

- ➢ Home: Represents the home page.
- ➢ Dashboard: Represents the dashboard page for authenticated users.

Algorithm Used:

Routing Algorithm (Pattern Matching):

- **Task:** Match the current URL path to a specific route and render the associated component.
- **Steps:**
    1. The Router captures the URL path.
    2. Routes iterates through the child Route components to find a match.
    3. The first Route that matches renders its corresponding component.

Authentication Check Algorithm (ProtectedRoute):

- **Task:** Protect specific routes and ensure only authenticated users can access them.
- **Steps:**
    1. ProtectedRoute checks if the user is authenticated via AuthProvider.
    2. If authenticated, it renders the child component (Dashboard).
    3. If not authenticated, it redirects the user to an appropriate fallback, typically a login page or Home.

Context API Algorithm (AuthProvider):

- **Task:** Manage and provide authentication state globally within the application.
- **Steps:**
    1. AuthProvider wraps the app, providing authentication state via React Context.
    2. Any child component can access the authentication state using the useContext hook.
    3. ProtectedRoute consumes this state to enforce access restrictions.