| Auth Routing  *"A React application implementing authentication-based routing using Zustand and React Router v6 to protect pages and manage user sessions."*  **Developed By: Gaurvi Paneri**  13 August, 2025  BrainyBeam Placement Task |
| --- |

# **Table of Contents**

1. **Introduction** *Overview and purpose of the Auth Routing*
2. **Objective** *What app aims to achieve and why*
3. **Features** *Description of key functionalities*
4. **Technology Stack** *Tools and libraries used in the project*
5. **Application Architecture** *Explanation of frontend structure and data flow*
6. **Implementation Details** *Key code logic, components, and state management*
7. **Usage Instructions** *How to run and use the app locally*
8. **Challenges and Solutions** *Challenges that were faced along with their solutions*
9. **ScreenShots**

*ScreenShots of the Output*

1. **Conclusion** *Summary of the project and learnings*
2. **References** *Resources and documentation referred to during development*
3. **Introduction**

This project is an authentication-based React application that controls routing to ensure only authenticated users can access protected pages. If a user attempts to access a protected route without logging in, they are redirected to the login page. Zustand is used for managing authentication state, enabling smooth login/logout handling and secure access to the protected dashboard.

1. **Objective**

The objectives of this applications are as follows:

* To restrict access to specific pages until the user is authenticated.
* To maintain the user's login and logout state locally for seamless session management.

1. **Features**

**Key features of this applications are as follows:**

* **Protected Dashboard –** Accessible only to authenticated users**.**
* **Login & Logout Functionality –** Allows users to securely sign in and sign out.
* **Automatic Redirection –** Redirects unauthenticated users to the login page when trying to access protected routes.
* **State Management with Zustand –** Stores and manages authentication state locally for smooth navigation.
* **Route Control with React Router v6 –** Handles public and private routes efficiently.

1. **Technologies Used**

* **React with Vite:** for fast and easy project setup
* **Styled Components:** for modular and maintainable component styling
* **Zustand:** for managing the user authentication state
* **React Router Dom:** or handling public and protected routes.

1. **Application Architecture**

**1. Folder Structure**

**Auth\_Routing**

**├──src**

**│ ├── components**

**│ └── ProtectedRoutes.jsx→***Contains the logic to protect dashboard route*

**│ ├── pages**

**│ ├──Login.jsx→***Navigates to dashboard if the user is authenticated*

**│ └── DashBoard.jsx→***Accessible only upon authentication*

**│ ├── store**

**│ └── auth.jsx→***stores the authentication state locally using Zustand*

**│ ├──App.css→***For styling main app layout*

**│ ├── App.jsx→***Defines routes and wraps dashboard in protected routes*

**└── main.jsx→***Application Entry point*

**├──public**

**├──index.html**

**└──package.json**

**2. Data Flow**

* **Login.jsx**
  + The user enters a username and password.
  + On successful input, the application updates the authentication state and navigates the user to the dashboard page.
* **auth.jsx (Store)**
  + Initializes the isLoggedIn state as false.
  + When credentials(Username and Password) are entered, it updates isLoggedIn to true.
  + On logout, it resets isLoggedIn back to false.
* **Dashboard.jsx**
  + This page is accessible only if **isLoggedIn is true**.
  + Contains a Logout button that triggers a state update, setting **isLoggedIn to false** and redirecting to the login page.
  + If a user tries to access this route directly via the browser’s address bar without logging in, the **ProtectedRoutes** component will block access and redirect them to the login page.

1. **Implementation Details**

* **State Management:**  
  Used Zustand to create a centralized store (auth.jsx) that maintains the isLoggedIn state. Initially set to false, it updates to true when the user logs in and back to false when logging out.
* **Protected Routes:**  
  Created a ProtectedRoutes component that checks the isLoggedIn state. If the state is false, it redirects the user to the login page instead of rendering the protected component.
* **Routing Setup:** Used React Router DOM to define application routes in App.jsx. Wrapped the Dashboard route with ProtectedRoutes to ensure it’s accessible only to logged-in users.
* **Login Flow:**  
   In Login.jsx, the user enters their username and password. On form submission, the isLoggedIn state in the store is updated, and the user is navigated to the dashboard.
* **Logout Flow:**  
   In Dashboard.jsx, clicking the Logout button resets isLoggedIn to false and redirects back to the login page.

1. **Usage Instructions**

### **Running the App Locally**

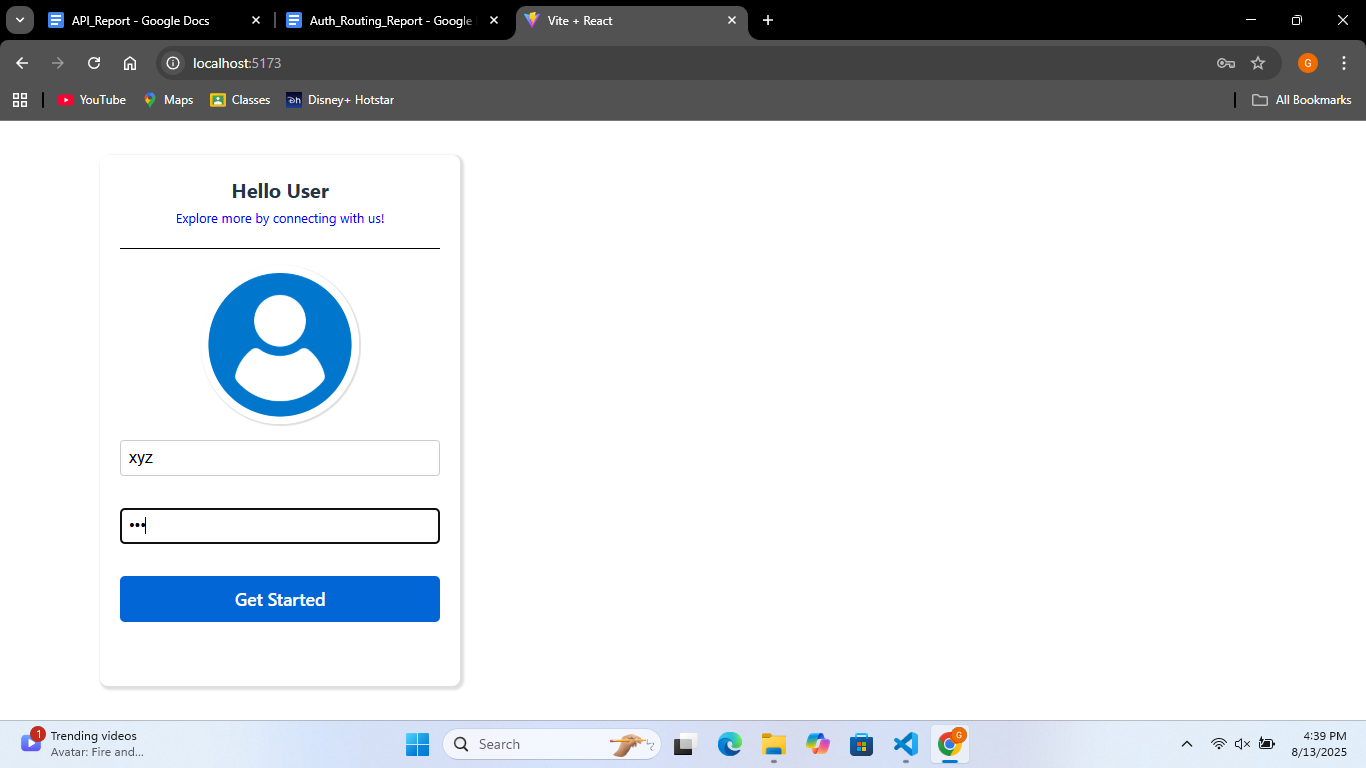
### **Prerequisites:**

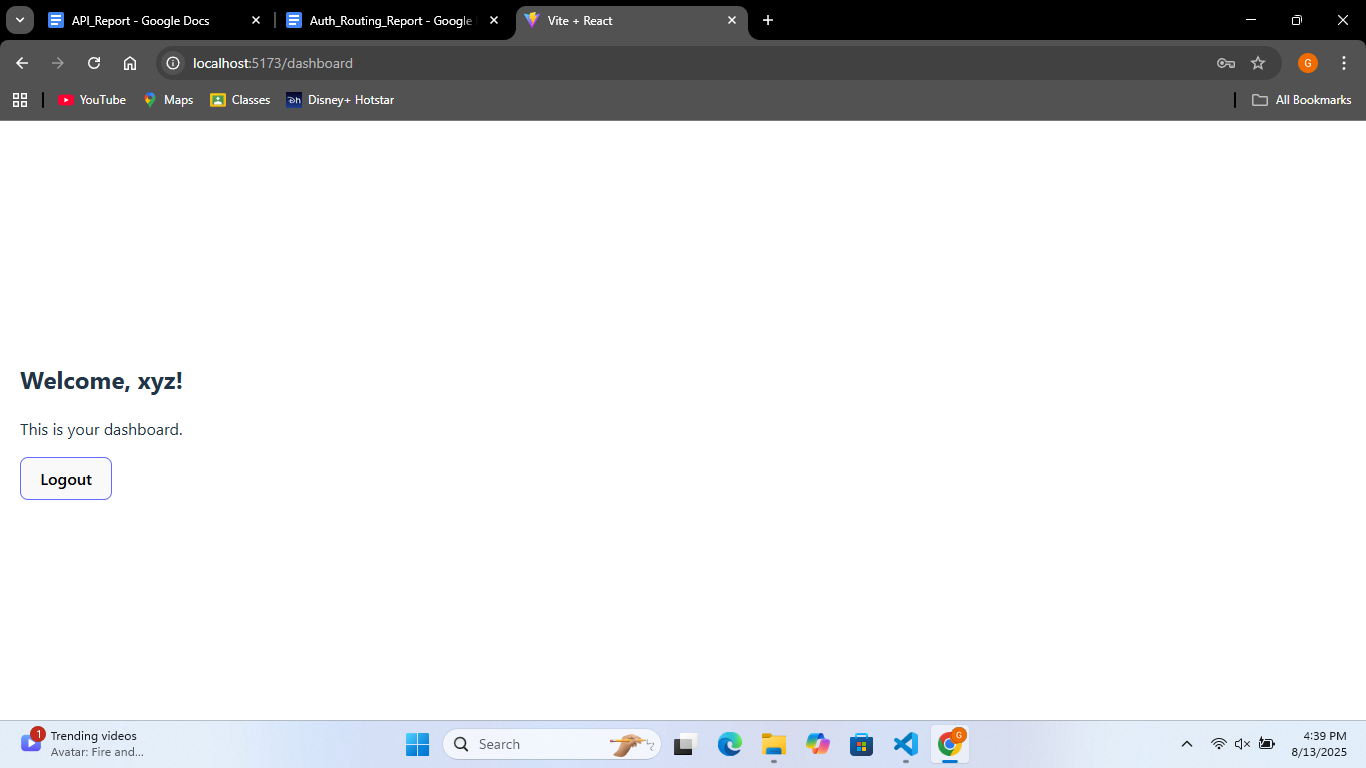
* + Make sure you have **Node.js (v16 or higher)** installed on your system.
  + A modern web browser (Chrome, Firefox, Edge, etc.)
* **Installation:**
  + Clone the project repository to your local machine:  
     **git clone React\_Task\_3\_GaurviPaneri**
  + Navigate to the project folder:  
    **cd React\_Task\_3\_GaurviPaneri**
  + Install dependencies:  
    **npm install**
* **Start the Development Server:**
  + Run the app locally using Vite:  
    **npm run dev**
  + Open the URL printed in the terminal (usually http://localhost:5173) in your browser.
* **Using the App:**Open the application in a web browser to view the Login page.  
  + Enter a username and password to navigate to the Dashboard page.
  + On the dashboard, clicking the Logout button redirects the user back to the login page.
  + To access the dashboard again, the user must re-enter their credentials.
  + If the user tries to navigate directly to **/dashboard** from the browser without being logged in, they are redirected to the login page.
  + If the page is refreshed while on the dashboard, the authentication state resets, and the user is redirected back to the login page.

1. **Challenges and Solutions**

* No major challenges were encountered during implementation as the concepts of authentication state management and protected routing had been practiced in a project I created earlier.
* Prior experience with React Router and Zustand made the task straightforward, enabling smooth and error-free development.

1. **ScreenShots**

****

****

1. **Conclusion**

This project successfully demonstrates authentication-based routing in a React application using Zustand for state management and React Router DOM for route control. It ensures that protected pages, such as the dashboard, are accessible only to logged-in users and redirects unauthorized access attempts to the login page, maintaining secure navigation flow.

1. **References**

* [**Vite Documentation**](https://vite.dev/guide/)
* [**React Documentation**](https://react.dev/)
* [**Zustand Documentation**](https://zustand.docs.pmnd.rs/getting-started/introduction)
* [**React Router Documentation**](https://www.npmjs.com/package/react-router-dom)