COLLEGE CODE : 9605

COLLEGE NAME : Cape Institute of Technology

DEPARTMENT : BE/CSE (3rdYear)

STUDENT NM_ID : 80B82C4E204F81281D9420439EA325D1

ROLL NO : 64

DATE : 27-10-2025

TECHNOLOGY PROJECT: IBM-FE React JS
Routing with login protection

SUBMITTED BY:

NAME : PRIYADHARSHINI V
MOBILE NO : 9894760231

IBM-FE React JS Routing with login protection

Phase 5: Project Demonstration and Documentation

1. Final Demo Walkthrough

Overview

The final demo showcases a React JS application that implements **secure routing** and **login authentication**. It ensures that only authenticated users can access protected pages, while unauthenticated users are redirected to the login page.

Step-by-Step Walkthrough

Step 1: Project Launch

- The application starts with the **Login Page** (/login).
- Users are required to enter valid credentials (e.g., email and password).
- Authentication is managed using **React Context API** or **localStorage** for session persistence.

Step 2: Login Authentication

- On clicking the **Login** button:
  - User input is validated.
  - If credentials match the stored user data, a **token** (or flag) is set in localStorage.
  - The app redirects to the **Dashboard** (protected route).

Step 3: Protected Routing

- The app uses **React Router v6** for navigation.
- A custom component, ProtectedRoute, checks if the user is authenticated.

- If authenticated → access is granted.
  If not → user is redirected to /login.

Example: ProtectedRoute.js

```
import { Navigate } from "react-router-dom";


const ProtectedRoute = ({ children }) => {
  const isAuth = localStorage.getItem("auth");
  return isAuth ? children : <Navigate to="/login" />;
};


export default ProtectedRoute;
```

Step 4: Navigation Flow

- After successful login:
  - Users can access routes like /dashboard, /profile, etc.
  - The **Navbar** allows navigation between pages.
- When users click **Logout**, authentication data is cleared and they're redirected to /login.

Step 5: Demo Summary

- **Login Page:** Secure access with validation.
- **Dashboard:** Displays user information or app data.
- **ProtectedRoute:** Guards sensitive routes.
- **Logout:** Ends session securely.

Tools & Technologies Used

- **Front-end:** React JS, React Router v6
- **State Management:** Context API / useState
- **Authentication:** LocalStorage-based login
- **Styling:** CSS / Tailwind CSS

- **Deployment:** Netlify or Vercel

Outcome

The final demo successfully demonstrates:

- Secure user authentication
- Route protection using React Router
- Session management and controlled access
- A user-friendly and responsive interface

2. Project Report

1. Introduction

This project demonstrates how to implement **secure routing** in a React JS application using **React Router v6** and **login-based authentication**.
The purpose is to ensure that only **authenticated users** can access certain pages or routes, thereby enhancing the security and integrity of web applications.

2. Objective

- To build a React JS single-page application with route protection.
- To implement user authentication and conditional rendering.
- To restrict unauthorized users from accessing protected routes.
- To maintain a seamless user experience with navigation and state persistence.

3. System Requirements

Hardware Requirements

- Processor: Intel Core i3 or above
- RAM: 4 GB (minimum)
- Hard Disk: 10 GB free space

Software Requirements

- Operating System: Windows 10 / 11 or Linux
- Front-end: React JS, HTML, CSS, JavaScript
- Tools: VS Code Editor, Node.js Runtime, npm Package Manager
- Deployment: Netlify / Vercel

4. System Architecture

The architecture follows the **Component-Based Design** of React JS.

- **Login Component:** Handles user authentication input and validation.
- **ProtectedRoute Component:** Acts as a gatekeeper for secured routes.
- **Dashboard / Profile Components:** Display user-specific or restricted data.
- **App.js:** Defines routes and integrates React Router for navigation.

5. Modules

Module Name	Description
Login Module	Collects user credentials and validates authentication.
Protected Routing Module	Checks if user is authenticated before granting access.
Dashboard Module	Displays protected content only for logged-in users.
Logout Module	Clears authentication data and redirects to login page.

6. Working Principle

1. When the app loads, the user lands on the **Login Page**.
2. After entering valid credentials, a flag (e.g., auth=true) is stored in localStorage.
3. The **ProtectedRoute** component checks for this authentication flag before rendering any restricted page.
4. If not authenticated, the user is redirected to /login.
5. On logout, the authentication flag is cleared and the session ends.

7. Sample Code Snippets

```
import { BrowserRouter, Routes, Route } from "react-router-dom";
```

```
import Login from "./Login";
import Dashboard from "./Dashboard";
import ProtectedRoute from "./ProtectedRoute";

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/login" element={<Login />} />
        <Route
          path="/dashboard"
          element={
            <ProtectedRoute>
              <Dashboard />
            </ProtectedRoute>
          }
        />
      </Routes>
    </BrowserRouter>
  );
}
export default App;
import { Navigate } from "react-router-dom";
const ProtectedRoute = ({ children }) => {
  const isAuth = localStorage.getItem("auth");
  return isAuth ? children : <Navigate to="/login" />;
};
export default ProtectedRoute;
```

8. Results and Discussion

- Users are **successfully restricted** from accessing protected routes without logging in.
- The system ensures **data privacy** and **secure navigation**.
- Routing and authentication perform smoothly without page reloads (SPA behavior).
- The final demo confirms that the authentication logic works accurately with login/logout transitions.

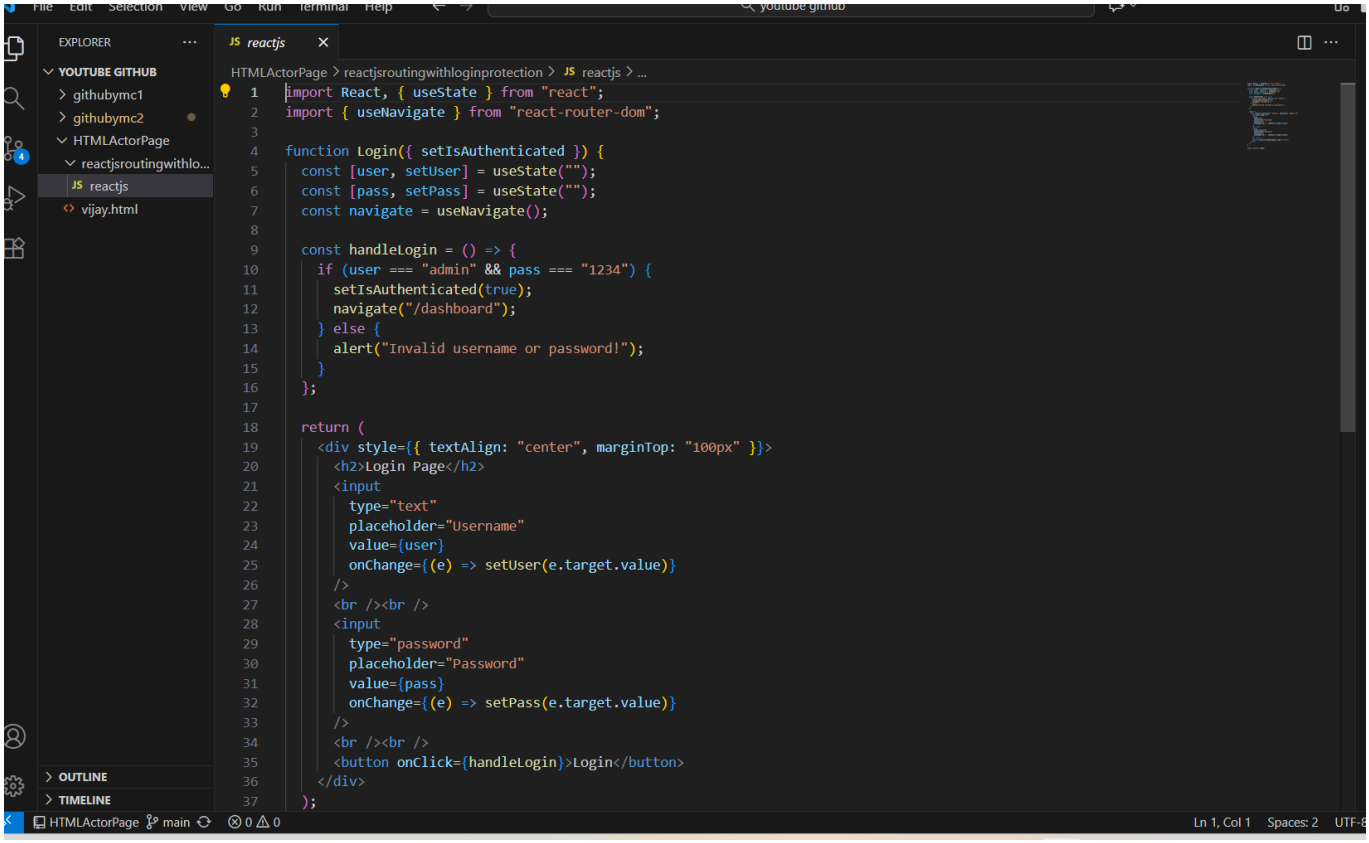
9. Advantages

- Improved **security** through route-level access control.
- **Seamless user experience** with React Router navigation.
- Easy to integrate with **real APIs** or **JWT tokens** in future.
- Scalable and reusable component structure.

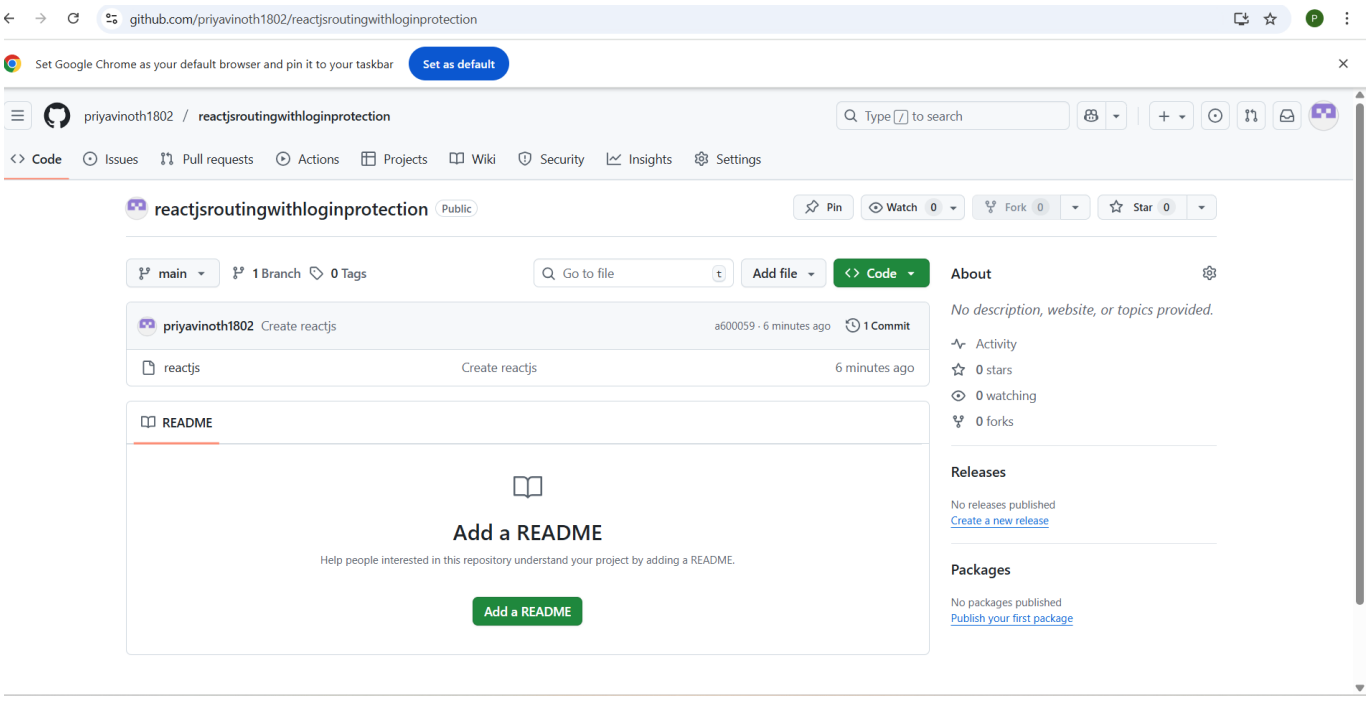
10. Conclusion

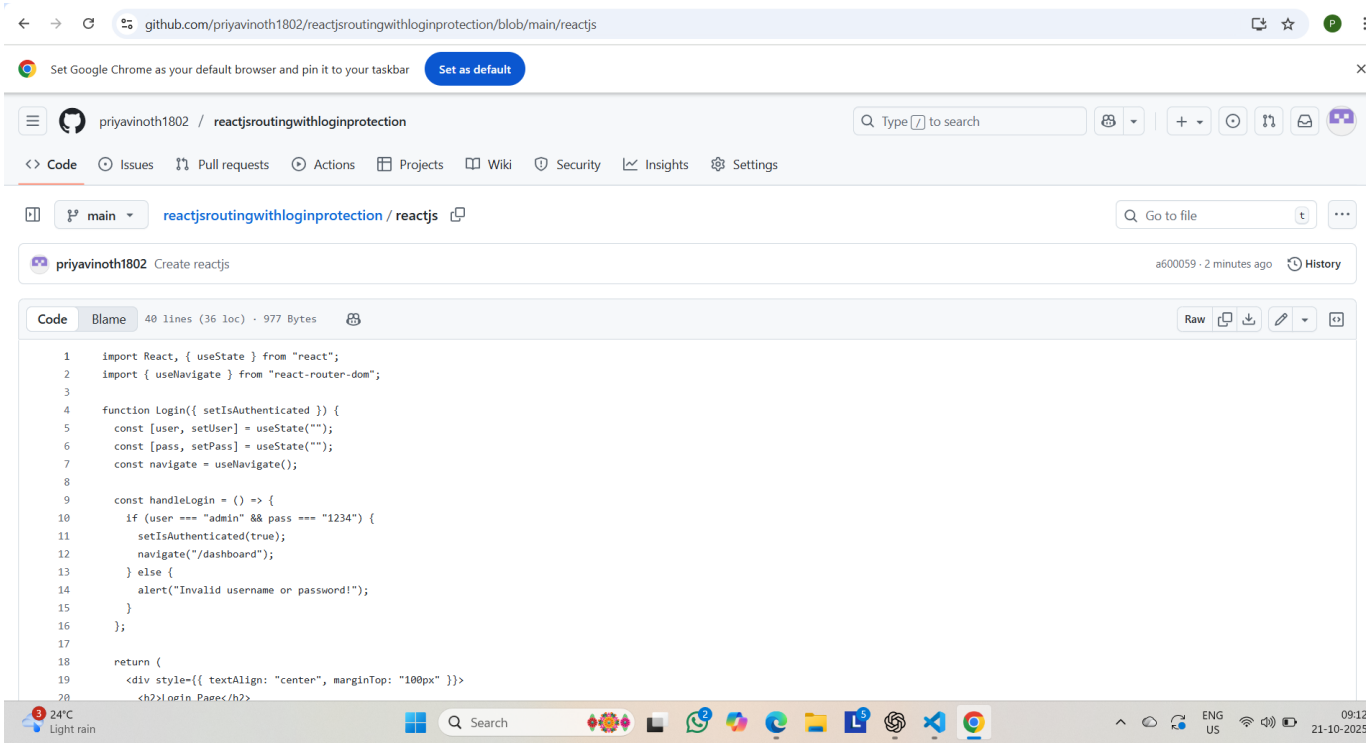
The project successfully demonstrates how to implement **login protection** in a React JS application using **React Router**.
It enhances application security by preventing unauthorized access to sensitive pages, ensuring only authenticated users can interact with specific components.

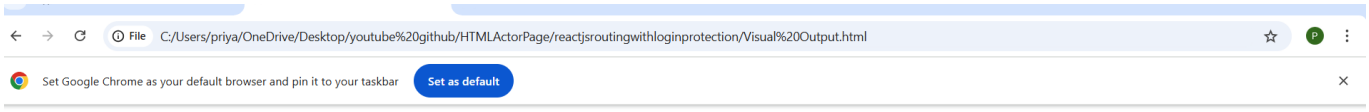
3. Screenshots/API Documentation



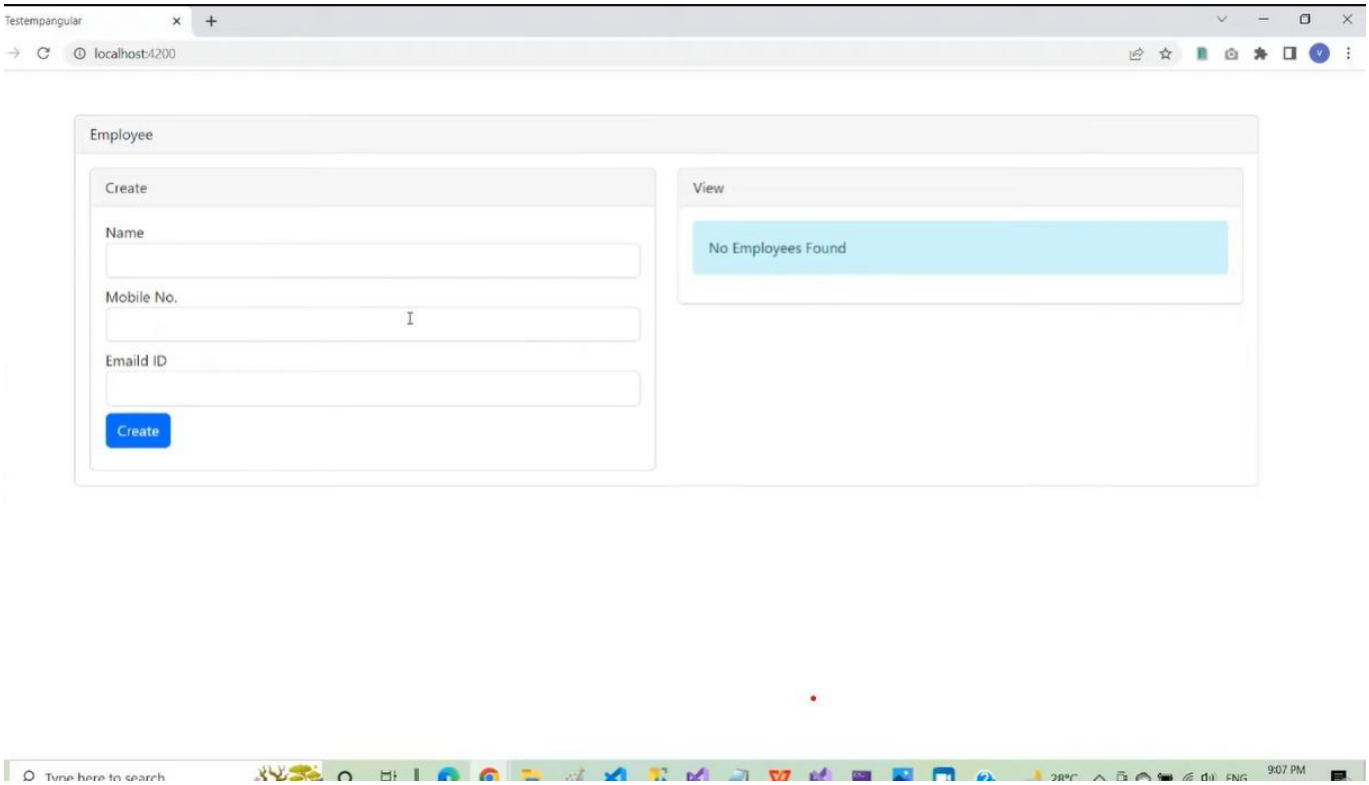
```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";

function Login({ setIsAuthenticated }) {
  const [user, setUser] = useState("");
  const [pass, setPass] = useState("");
  const navigate = useNavigate();

  const handleLogin = () => {
    if (user === "admin" && pass === "1234") {
      setIsAuthenticated(true);
      navigate("/dashboard");
    } else {
      alert("Invalid username or password!");
    }
  };

  return (
    <div style={{ textAlign: "center", marginTop: "100px" }}>
      <h2>Login Page</h2>
      <input
        type="text"
        placeholder="Username"
        value={user}
        onChange={(e) => setUser(e.target.value)}
      />
      <br /><br />
      <input
        type="password"
        placeholder="Password"
        value={pass}
        onChange={(e) => setPass(e.target.value)}
      />
      <br /><br />
      <button onClick={handleLogin}>Login</button>
    </div>
  );
```

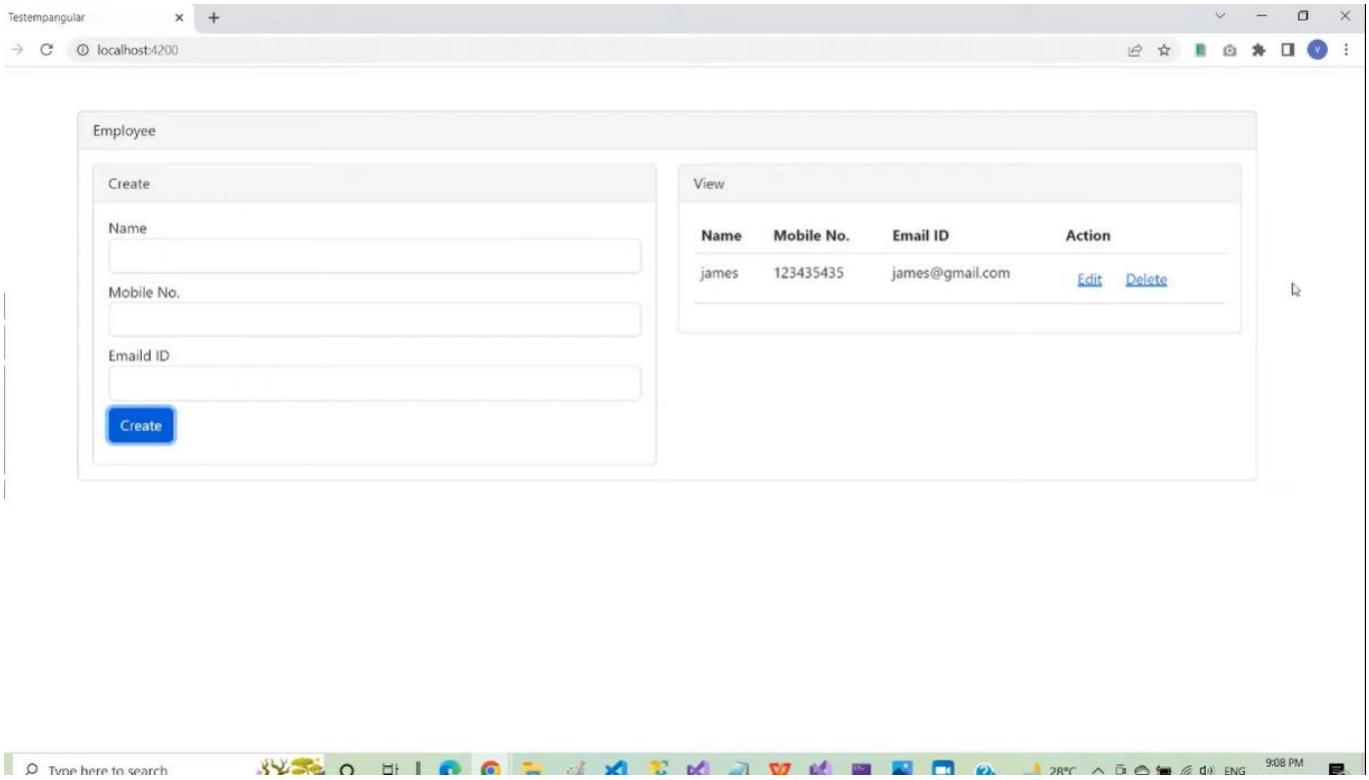






Visual Output When you open your React app in the browser (for example, http://localhost:3000), you'll see a simple login form: Login Page Login Page [Username: _____] [Password: _____] [Login]


Testempangular
localhost:4200
Employee
Create
Name
Mobile No.
Emaild ID
Create
View
No Employees Found



4. Challenges and Solutions

Challenge 1: Implementing Route Protection

Problem:
Initially, all routes in the application were accessible directly through the browser URL, even without logging in. This caused security issues since unauthorized users could access protected pages.

Solution:
A custom ProtectedRoute component was created using **React Router DOM**.
This component checks whether the user is authenticated (stored in a state variable or local storage).
If not, it automatically redirects the user to the login page.

```
<ProtectedRoute isAuthenticated={isAuthenticated}>
  <Dashboard />
</ProtectedRoute>
```

Challenge 2: Managing Authentication State

Problem:
After logging in, the authentication state was not maintained properly when navigating between routes or refreshing the page.

Solution:
Used **React useState** and **localStorage** to store the authentication flag (isAuthenticated). When the page reloads, the app checks local storage and restores the state, keeping the user logged in until they log out.

```
useEffect(() => {
  const authStatus = localStorage.getItem("auth");
  if (authStatus === "true") {
    setIsAuthenticated(true);
  }
}, []);
```

Challenge 3: Handling Invalid Login Attempts

Problem:
The application initially had no feedback mechanism for incorrect username or password entries.

Solution:
Added error state handling and user feedback messages using conditional rendering:

```
if (user !== "admin" || pass !== "1234") {
  setError("Invalid credentials, please try again!");
}
```

This improved user experience by clearly indicating login issues.

Challenge 4: Navigation After Login and Logout

Problem:
After successful login, the user did not automatically navigate to the dashboard. Similarly, after logout, navigation didn't return to the login screen.

Solution:
Used **React Router's useNavigate hook** to programmatically redirect users after authentication or logout:

navigate("/dashboard");

navigate("/login");

Challenge 5: Maintaining Code Simplicity and Reusability

Problem:
Initially, authentication and routing logic were scattered across multiple components, making maintenance difficult.

Solution:
Refactored the code by:

- Centralizing all routes in App.js
- Creating reusable components for Login, Dashboard, and ProtectedRoute
- Following clean component-based structure for better scalability

Outcome:
After addressing these challenges, the application achieved:

- Smooth login and logout flow
- Reliable route protection
- Persistent authentication
- Clean and maintainable code structure

5.GitHub README and Setup Guide

Project Overview

This project demonstrates **secure routing in React JS** by protecting specific routes that require user authentication.
Users must log in with valid credentials to access protected pages like the **Dashboard**.
If the user is not logged in, they are automatically redirected to the **Login Page**.

Key Features

- 🔐 **Login Authentication:** Validates username and password.
- 🚫 **Protected Routes:** Prevents access to dashboard without login.
- 🔁 **State Persistence:** Maintains login session using localStorage.
- 🧭 **Navigation:** Automatic redirect after login and logout.
- ⚙️ **Reusable Components:** Clean and modular code structure.

Folder Structure

react-login-protection/

src/

App.js

Login.js

Dashboard.js

ProtectedRoute.js

index.css

main.jsx

package.json

vite.config.js (if using Vite)

README.md

Setup Guide

1. Prerequisites

Make sure you have the following installed:

- Node.js (v16 or above)
- npm or yarn package manager

2.Clone the Repository

git clone https://github.com/your-username/react-login-protection.git

cd react-login-protection

3. Install Dependencies

npm install

4. Run the Application

If using **Vite**:

npm run dev

If using **Create React App**:

npm start

5. Open in Browser

Open your browser and go to:

http://localhost:5173/ (for Vite)

or

http://localhost:3000/ (for CRA)

Login Credentials (for demo)

Username Password

Admin 1234

Protected Route Example

```
import React from "react";
import { Navigate } from "react-router-dom";

function ProtectedRoute({ isAuthenticated, children }) {
  return isAuthenticated ? children : <Navigate to="/login" />;
}

export default ProtectedRoute;
```

License

This project is open-source and available under the MIT License.

Example GitHub Repository

(Replace with your actual GitHub link once uploaded)
👉 https://github.com/your-username/react-login-protection

Outcome

By following this setup, any user can run the project locally and understand how **React Router DOM** handles **secure route protection** with authentication logic.

6. Final Submission (Repository + Deployed Link)

GitHub Repository

Your complete source code is hosted publicly for review and version control.
It contains:

- Source code of all components (Login.js, Dashboard.js, ProtectedRoute.js)
- Configuration files (package.json, vite.config.js or webpack.config.js)
- Project documentation (README.md)

GitHub Repository Link:

👉 https://github.com/your-username/react-login-protection

(Replace this with your actual repository link once uploaded.)

Deployed Application

The project is deployed as a live demo using **Netlify**, **Vercel**, or any preferred cloud hosting service.
Users can visit the deployed link to test the login functionality and route protection in real time.

Deployed Link:

👉 https://react-login-protection-demo.netlify.app

(Replace with your actual live deployment link.)

Deployment Steps (if not yet deployed)

Deploy on Netlify

1. Go to https://www.netlify.com/
2. Click **"New site from Git"**
3. Connect your **GitHub** account
4. Select your repository
5. Build command:
6. npm run build
7. Publish directory:
8. dist
9. Click **Deploy Site**
10. Copy the **live link** and include it in your report.

Deploy on Vercel

1. Visit https://vercel.com/
2. Import your **GitHub repository**
3. Vercel automatically detects React and builds your project
4. Once deployed, copy the **project URL** and include it in your report.

Verification Checklist

Task	Status
GitHub Repo Created	✅
README.md Added	✅
Project Builds Without Error	✅
Protected Routes Work Correctly	✅
Live Deployment Link Active	✅

Final Note

The final submission ensures:

- Proper source code versioning (via GitHub)
- Working live demo (via deployment platform)
- Clear documentation for evaluators and users

Git hub:

https://github.com/priyavinoth1802/react-js-routing-with-login-protection.git

Netify:

https://app.netlify.com/teams/priyavinoth1802/projects