**9530**

**ST.MOTHER THERESA ENGINEERING COLLEGE**

COMPUTER SCIENCE AND ENGINEERING

**NM-ID:**7930244674B12BB982040D1318B703FE3

**REG NO:**953023104074

**DATE:**16-09-2025

**Completed the project named as**

**PHASE-2**

**ROUTING WITH LOGIN PROTECTION**

SUBMITTED BY,

M.MUTHUMARI

**PH NO:**7845106149

# Phase 2 Project Report

## Topic: Routing with Login Protection

## Problem Statement

The project aims to provide a secure and seamless navigation system where users can access only the routes they are authorized to view, ensuring that sensitive pages are protected with login authentication.

## Objective

Design and develop a front-end application with protected routing that integrates with a login mechanism to ensure data security and user authentication.

## Aim 1

Implement client-side routing using React Router to manage navigation between public and protected pages.

## Aim 2

Integrate authentication using JWT or session storage to protect routes and provide role-based access.

## Tech Stack Selection

| Layer | Technology |
|---|---|
| Frontend | React.js + React Router |
| Backend (suggested) | Node.js + Express |
| Authentication | JWT (JSON Web Token) |
| Database | MongoDB (or any NoSQL/SQL) |
| Styling | Tailwind CSS or plain CSS |

## UI Structure

• Login Page • Dashboard (Protected) • Profile Page (Protected) • Public Home Page • Error / Unauthorized Access Page

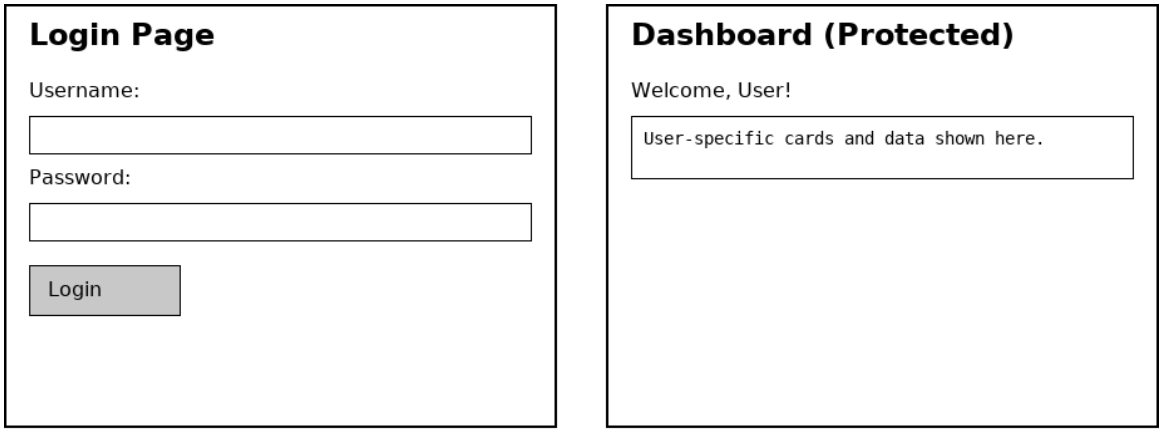
## Data Handling Approach

Credentials are sent over HTTPS to the backend authentication endpoint. On success, the backend returns a JWT token which is stored in localStorage or an HTTP-only cookie. Subsequent requests include the token for authorization.

## Component Design

• Login Component • ProtectedRoute Component (checks auth) • Dashboard Component • Navbar / Logout Component

## Frontend Output (Mockup)




Mockup: Click Login to authenticate → redirected to Dashboard if token exists (simulated).

## Basic Flow Chart

Start → Visit Login Page → Submit Credentials → Auth API → If success → Store token and Redirect to Dashboard → On protected route access, check token → If valid, allow; else redirect to Login.

## Program (Frontend - React Example)

```
// App.jsx (simplified)
import React, { useState, useEffect } from 'react';
import { BrowserRouter as Router, Route, Routes, Navigate } from 'react-router-dom';

const fakeAuthApi = (username, password) => {
  // Simulate API: returns a token string on correct credentials
  if (username === 'user' && password === 'pass') return 'fake-jwt-token';
  return null;
};

function Login({ setAuth }) {
  const [user, setUser] = useState('');
  const [pass, setPass] = useState('');
  const handleLogin = () => {
    const token = fakeAuthApi(user, pass);
    if (token) {
      localStorage.setItem('token', token);
      setAuth(true);
    } else {
      alert('Invalid credentials');
    }
  };
  return (
    <div>
      <input placeholder="Username" value={user} onChange={e => setUser(e.target.value)} />
      <input placeholder="Password" type="password" value={pass} onChange={e => setPass(e.target.value)} />
      <button onClick={handleLogin}>Login</button>
    </div>
  );
}

function Dashboard() {
  return <h2>Welcome to Dashboard</h2>;
}
function ProtectedRoute({ auth, children }) {
```

```
  return auth ? children : <Navigate to="/" />;
}

export default function App() {
  const [auth, setAuth] = useState(!!localStorage.getItem('token'));
  useEffect(() => {
    setAuth(!!localStorage.getItem('token'));
  }, []);
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Login setAuth={setAuth} />} />
        <Route path="/dashboard" element={<ProtectedRoute auth={auth}><Dashboard /></ProtectedRoute>} />
      </Routes>
    </Router>
  );
}
```

## Output (Expected)

1. Enter username: 'user' and password: 'pass' in the Login mockup to simulate a successful login. 2. On success, a token is stored in localStorage and the user is redirected to /dashboard. 3. Accessing /dashboard without token redirects to Login page.

## Conclusion

This Phase 2 report includes the complete design, a frontend program example, and a visual mockup of the UI output. Next steps: connect to a real backend, replace fakeAuthApi with real API calls, and add role-based access control.