Module 10

What to learn

Routing module
Integrate routing with your app

Practice Exercise

Practice 1
Do the hands on from following url https://angular.io/guide/router-tutorial-toh for milestone2 and 3
Practice 2
Implement an **Auth Guard** in Angular to protect the **Admin Dashboard** (/admin) and **User Dashboard** (/user) routes. The **Admin Dashboard** should be accessible only to users with the **Admin** role, while the **User Dashboard** should be accessible to any authenticated user. Redirect unauthenticated users to the **login page** (/login) when they try to access these protected routes.

Assignment Exercise

Assignment 1
With Reference to the Day16 Assignment Create a Separate Module for HTML/CSS and JavaScript and do the routing accordingly.
Assignment 2

Use Case: Angular Application with Auth Guard and Lazy Loading

Scenario:

You are tasked with building an **Angular application** with multiple modules. Some modules are publicly accessible, while others require authentication to access. You will need to implement **lazy loading** for efficiency and **auth guards** to protect specific routes.

Features:

Public Routes:

Home Page (/home): Public route accessible by any user.

About Page (/about): Public route accessible by any user.

Protected Routes:

Dashboard Page (/dashboard): Only accessible to authenticated users.
Settings Page (/dashboard/settings): A child route under **Dashboard**, also protected.

Modules:

Public Module (PublicModule): Lazy-loaded module containing **Home** and **About** pages.
Dashboard Module (DashboardModule): Lazy-loaded module containing **Dashboard** and **Settings** pages.
Auth Module: Handles login and authentication logic.

Auth Guard:

Protect the **Dashboard** and **Settings** routes with an **Auth Guard** that checks if the user is authenticated.
If a user tries to access these routes without being authenticated, redirect them to the **login page**.

Login and Logout:

A **login page** that allows users to enter their credentials and authenticate.
A **logout function** that clears the authentication token and redirects the user to the **home page**.

Route Configuration:

App Routing (app-routing.module.ts):

Set up routes for the **Public Module** (/home, /about) and **Dashboard Module** (/dashboard).
Lazy load the **PublicModule** for home and about pages.
Lazy load the **DashboardModule** for the dashboard and settings pages.
Protect **Dashboard** and **Settings** routes with the **Auth Guard**.

Function Flow:

Routing Flow:

The **app-routing.module.ts** file configures the routes:

The root / redirects to the public pages (Home and About).
The /dashboard route is lazily loaded using loadChildren. It's protected by the **Auth Guard**.
The /dashboard/settings route is lazily loaded as a child of /dashboard, and it's also protected by the **Auth Guard**.

Auth Guard Flow:

The **AuthGuard** intercepts the navigation to protected routes (/dashboard, /dashboard/settings):

If the user is authenticated (based on a token stored in local storage), the route is activated.

If the user is not authenticated, the **AuthGuard** redirects the user to the **login page** (/login).

Login Flow:

On accessing the **login page**, the user is prompted to enter credentials (username, password).

When the user submits the login form:

The **AuthService** checks if the credentials are valid.

If valid, the **AuthService** stores a token in **localStorage** to mark the user as authenticated.

The user is redirected to the **dashboard** page (/dashboard).

If invalid, an error message is displayed, and the user stays on the login page.

Logout Flow:

When the user clicks the **logout** button:

The **AuthService** clears the authentication token from **localStorage**.

The user is redirected to the **home page** (/home).

Protected Route Flow:

If an authenticated user tries to access the **dashboard** or **settings** route directly:

The **AuthGuard** checks for the authentication token in **localStorage**.

If the token is valid, the route is activated, and the user can view the page.

If the token is not valid or absent, the user is redirected to the **login page**.

Implementation Steps:

Set Up App Routing (app-routing.module.ts):

Create routes for **public** and **protected** pages.

Use **lazy loading** for **PublicModule** and **DashboardModule**.

Implement the **AuthGuard** on the **Dashboard** and **Settings** routes.

Create Auth Guard (auth.guard.ts):

Implement the **AuthGuard** to protect routes.

Redirect unauthorized users to the **login page**.

Create Auth Service (auth.service.ts):

Implement methods for **login**, **logout**, and **checking authentication** status using **localStorage**.

Create Login Component (login.component.ts):

Implement the login form, authenticate the user, and store the authentication token.

Create Public Module (public.module.ts):

Implement the **Home** and **About** pages as lazily-loaded components.

Create Dashboard Module (dashboard.module.ts):

Implement the **Dashboard** and **Settings** pages as lazily-loaded components.

Function Flow Diagram:

User navigates to /home or /about → The route is publicly accessible and loads the respective component.

User navigates to /dashboard or /dashboard/settings → Auth Guard checks if the user is authenticated:

If **Authenticated** → Route is activated and the component is displayed.

If **Not Authenticated** → Redirect to /login page.

User logs in → AuthService stores token in **localStorage** → Redirect to /dashboard.

User clicks logout → AuthService removes token from **localStorage** → Redirect to /home.

Deliverables:

Routing Configuration for lazy loading and auth guards.

Auth Guard that prevents unauthorized access.

Login/Logout logic to authenticate users and manage sessions.

Modular, Lazy-Loaded Application for performance optimization.

Online Reference

No online Reference

Introduction

the basics

course project-basics

debugging

components & databinding deep dive

course project – components & databinding

directives deep dive

Using Services & Dependency Injection

Course Project – Services & Dependency Injection

Changing Pages with Routing

Course Project – Routing

Handling Forms in Angular Apps

Course Project-Forms

Using Pipes to Transform Output

Making Http Requests