**1. Create an Auth Guard**

First, create the AuthGuard service using Angular CLI or manually.

bash

ng generate guard auth

This will generate a guard file auth.guard.ts. In this guard, you can implement the logic that determines whether the user can access the route or not.

**2. Implement the Auth Guard Logic**

In the generated auth.guard.ts, implement the CanActivate or CanLoad methods. The most common one is CanActivate, which is used to determine if a route can be activated.

Here’s an example of a simple AuthGuard:

// src/app/auth.guard.ts

import { Injectable } from '@angular/core';

import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router } from '@angular/router';

import { Observable } from 'rxjs';

import { AuthService } from './auth.service'; // A service for authentication logic

@Injectable({

providedIn: 'root'

})

export class AuthGuard implements CanActivate {

constructor(private authService: AuthService, private router: Router) {}

canActivate(

route: ActivatedRouteSnapshot,

state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {

// Check if the user is authenticated

if (this.authService.isAuthenticated()) {

return true; // Allow access to the route

}

// If not authenticated, redirect to login page

this.router.navigate(['/login']);

return false; // Prevent access to the route

}

}

**3. Implement Authentication Service**

In the example above, we use an AuthService to check if the user is authenticated. Here’s an example of a simple AuthService that manages the authentication status.

// src/app/auth.service.ts

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class AuthService {

private authenticated = false;

constructor() {}

login() {

this.authenticated = true;

}

logout() {

this.authenticated = false;

}

isAuthenticated(): boolean {

return this.authenticated;

}

}

**4. Configure Routes to Use the Auth Guard**

In your app-routing.module.ts, apply the AuthGuard to the routes that you want to protect. For example, you might want to protect a dashboard route so that only authenticated users can access it.

// src/app/app-routing.module.ts

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { DashboardComponent } from './dashboard/dashboard.component';

import { LoginComponent } from './login/login.component';

import { AuthGuard } from './auth.guard';

const routes: Routes = [

{ path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard] }, // Protected route

{ path: 'login', component: LoginComponent }, // Login route

{ path: '', redirectTo: '/login', pathMatch: 'full' }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule {}

**5. Test the Guard**

To test the guard, simulate an authentication flow:

* Initially, the user will be redirected to the login page.
* After logging in (by calling authService.login()), the user should be able to access the protected route (/dashboard).

You can add the login logic in the LoginComponent like this:

// src/app/login/login.component.ts

import { Component } from '@angular/core';

import { AuthService } from '../auth.service';

import { Router } from '@angular/router';

@Component({

selector: 'app-login',

template: `

<button (click)="login()">Login</button>

`

})

export class LoginComponent {

constructor(private authService: AuthService, private router: Router) {}

login() {

this.authService.login();

this.router.navigate(['/dashboard']);

}

}

**6. Optional: Using CanLoad**

If you want to protect entire lazy-loaded modules, use CanLoad instead of CanActivate. This ensures that the route is not even loaded if the guard condition fails.

Here’s an example of how to use CanLoad:

// src/app/auth.guard.ts

import { Injectable } from '@angular/core';

import { CanLoad, Route, UrlSegment, Router } from '@angular/router';

import { AuthService } from './auth.service';

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class AuthGuard implements CanLoad {

constructor(private authService: AuthService, private router: Router) {}

canLoad(

route: Route,

segments: UrlSegment[]): Observable<boolean> | Promise<boolean> | boolean {

if (this.authService.isAuthenticated()) {

return true;

}

this.router.navigate(['/login']);

return false;

}

}