# Angular 7|6 Tutorial Course: Securing the UI with Router Guards and UrlTree Parsed Routes (for Login Redirects)

In the previous tutorial, we have added routing in our developer portfolio web application created with Angular 7. Let's now secure the UI with router guards.

We'll be learning how to use Router Guards and UrlTree data structures to protect the UI if the user is not logged in and redirect them to the login interface if they don't have access to a specific route.

The admin interface can be only accessed by the website owner so we need to use Guards to protect the components of the admin module and only allow access to them if the user is logged in.

First, you need to create a guard. Run the following command in your terminal to generate a guard service:

$ ng g guard admin/admin

***Note****: We prefix the guard name with the admin/ path to generate it inside the admin folder for the matter of code organization.*

Two src/app/admin/admin.guard.spec.ts and src/app/admin/admin.guard.ts files will be generated. Open the src/app/admin/admin.guard.ts file, you should see the following code:

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

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

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class AdminGuard implements CanActivate {

canActivate(

next: ActivatedRouteSnapshot,

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

return true;

}

}

From the code, you see that a guard is simply a service that implements the CanActivate interface and overrides the canActivate() method. In this case, it always returns true which means access will be always granted to the user when this guard is applied to a route.

***Note****: There are other types of Guards such as:*

* *CanActivateChild: used to allow or disallow access to child routes.*
* *CanDeactivate: used to allow or deny exit from route.*
* *Resolve`: used for doing operations (resolve data) just before route activation etc.*

Let's change the method to only allow access if the user is logged in. First, you need to import AuthServiceand inject it via the AdminGuard service and next you need to call the isLoggedIn property in the canActivate() method to check if the user is logged in and return true or false;

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

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

import { Observable } from 'rxjs';

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

@Injectable({

providedIn: 'root'

})

export class AdminGuard implements CanActivate {

constructor(private authService: AuthService){}

canActivate(

next: ActivatedRouteSnapshot,

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

return this.authService.isLoggedIn;

}

}

The canActivate() method will return true if the user is logged in or false otherwise.

The canActivate() method is passed many arguments which makes it easy to detrmine if the guard needs to allow or disallow access to certain route(s):

1. next: ActivatedRouteSnapshotwhich is the next route that will be activated if the guard is allowing access,
2. state: RouterStateSnapshotwhich is the next router state if the guard is allowing access.

Now, you need to apply the guard to the routes you need to protect using the canActivate property of the path object. Open the src/app/admin/admin-routing.module.ts file and update it accordingly:

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

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

import { ProjectComponent } from './project/project.component';

import { ProjectListComponent } from './project-list/project-list.component';

import { ProjectCreateComponent } from './project-create/project-create.component';

import { ProjectUpdateComponent } from './project-update/project-update.component';

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

import { AdminGuard } from './admin.guard';

const routes: Routes = [

{

path: 'admin',

component: ProjectComponent,

children: [

{

path: 'list',

component: ProjectListComponent,

canActivate: [AdminGuard]

},

{

path: 'create',

component: ProjectCreateComponent,

canActivate: [AdminGuard]

},

{

path: 'update',

component: ProjectUpdateComponent,

canActivate: [AdminGuard]

},

{

path: 'login',

component: LoginComponent

}

]

}

];

@NgModule({

imports: [RouterModule.forChild(routes)],

exports: [RouterModule]

})

export class AdminRoutingModule { }

We protected the ProjectListComponent, ProjectCreateComponent and ProjectUpdateComponentcomponents of the admin module from non logged in users.

***Note****: The canActivate property of the path object takes an array which means you can register multiple guards.*

Before Angular 7.1, route guards can only return a boolean, Promise<boolean> or Observable<boolean> (asynchronous boolean objects) to tell the router if the route can be activated or not. But now, you can also return an UrlTree variable which provides the new router state (route) that should be activated.

According to the [Angular docs](https://angular.io/api/router/UrlTree) an UrlTree is a data structure that represents a parsed URL.

***Note****: You can create an UrlTree by calling the parseUrl() or createUrlTree() method of the Router object.*

Now, let's change our router guard to redirect the users to the /admin/login route if they try to access the protected admin components without being logged in first:

* First, we import and inject the router,
* Next, we update the canActivate() method to return an UrlTree corresponding to the /admin/login route.

This is the full code of the AdminGuard service:

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

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

import { Observable } from 'rxjs';

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

@Injectable({

providedIn: 'root'

})

export class AdminGuard implements CanActivate {

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

{}

canActivate(

next: ActivatedRouteSnapshot,

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

if(this.authService.isLoggedIn){

return true;

}

else{

return this.router.parseUrl("/admin/login");

}

}

}

We check if the user is logged in and we return true, otherwise we return the result from the parseUrl("/admin/login") method of the injected router instance which is an UrlTreeof the /admin/login route.

Now, go to your application, if you visit any protected route without logging in you will be redirected to the /admin/login route where you can login.

Check out all parts:

* [Angular 7|6 Tutorial Course: CLI, Components, Routing & Bootstrap 4](https://www.techiediaries.com/angular-course),
* [Angular 7|6 Tutorial Course: Angular NgModules (Feature and Root Modules)](https://www.techiediaries.com/angular-course-modules),
* [Angular 7|6 Tutorial Course: Nested Router-Outlet, Child Routes & forChild()](https://www.techiediaries.com/angular-course-child-routes),
* [Angular 7|6 Tutorial Course: Authentication with Firebase (Email & Password)](https://www.techiediaries.com/angular-course-firebase-authentication),
* Angular 7|6 Tutorial Course: Securing the UI with Router Guards and UrlTree Parsed Routes

## Conclusion

As a recap, we’ve seen how to use route guards some new features introduced in Angular v7.1 enables you to redirect to another route by using a UrlTree parsed route.

In this tutorial, we've used Angular Guards and UrlTree structures that correspond to parsed routes to disallow access to certain routes if users are not logged in. In the next tutorial, we'll proceed by implementing the CRUD operations of the admin interface which allows the portfolio owner to add projects to their website. We'll be using Firestore as our persistence layer.