Angular route guard allows us to grant or remove access to certain parts of the navigation. Another route guard, the CanDeactivate guard, enables you to prevent a user from accidentally leaving a component with unsaved changes.

**Why do we need Angular guards?**

To prevent unauthorized access to certain parts of our navigation, use route guards in Angular.

The client-side route guards like this are not meant to be a security feature. However, they won’t prevent a smart user from figuring out how to get to the protected routes.

3.2M

Such security should be implemented on the server-side. So you need to develop the logic for the server-side, and based on the response, we will change the routes.

Route guards are instead meant as a way to improve the UX for your apps.

**Types of routing guards**

Route guards in Angular can prevent users from navigating to parts of an app without authorization.

There are 4 route guards available in Angular.

1. **CanActivate**: It controls if a route can be activated.
2. **CanActivateChild**: It controls if children of a route can be activated.
3. **CanLoad**: It controls if a route can even be loaded. This becomes useful for lazy-loaded feature modules. They won’t also load if the guard returns false.
4. **CanDeactivate**: It controls if the user can leave a route. **Note that** this guard doesn’t prevent the user from closing the browser tab or navigating to a different address. It only prevents actions from within the application itself.

To use route guards, consider using component-less routes as this facilitates guarding child routes.

**How to Create Guard Service in Angular**

To create a service for your guard, type the following command.

ng **generate** guard your-guard-name

In your guard class, implement the guard you want to use. The following example uses CanActivate to guard the route.

export **class** **YourGuard** **implements** **CanActivate** {

canActivate(

next: ActivatedRouteSnapshot,

state: RouterStateSnapshot): **boolean** {

*// your logic goes here*

}

}

**Angular Route Guard Example**

In traditional server-side web applications, the application would check the permissions on the server and return the 403 error page if the user didn’t have permissions, or perhaps redirect them to a /auth/login/ page if they were not signed in.

We want to have the same functionality on our client-side SPA, and with **Router Guards,** we can implement that kind of functionality.

With **Router Guards,** we can prevent the users from accessing areas that they’re not permitted to access, or we can ask them for confirmation when leaving the particular area. Let’s perform a practical and see how we can set up a guard for our angular application.

**Step 1: Install the Angular 12 project.**

To create a new Angular 12 project, type the following command.

ng **new** angularguard

While creating a new project, please enable the routing so that the **app-routing.module.ts**file is created.

Now, go inside the project and create the following two components.

1. HomeComponent
2. DashboardComponent

Type the following commands to create components.

ng g c home --skipTests=true

ng g c dashboard --skipTests=true

From these components, we will prevent access to the dashboard component if the user is not logged in. Otherwise, the user can access the dashboard component.

That means we will set the auth guard to the dashboard component. So if the auth service returns true, then the user is authenticated; otherwise, it is not. Based on true or false, we will prevent access to the component.

**Step 2: Setup routing**

Now, let’s set up the routing for these two components.

Write the following code inside the **app-routing.module.ts**file.

*// app-routing.module.ts*

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

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

**import** { HomeComponent } from './home/home.component';

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

const routes: Routes = [

{ path: '', component: AppComponent },

{ path: 'home', component: HomeComponent},

{ path: 'dashboard', component: DashboardComponent }

];

**@NgModule**({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export **class** **AppRoutingModule** { }

We have defined three routes.

1. **‘/’** route for AppComponent.
2. **‘home’** route for HomeComponent
3. **‘dashboard’** route for DashboardComponent

Now, edit the **app.component.html** file and write the following code.

*<!-- app.component.html -->*

<p>Angular Auth Guard Example</p>

<router-outlet></router-outlet>

So, now you have three navigation parts in your angular 12 application.

1. **http://localhost:4200/**
2. **http://localhost:4200/home**
3. **http://localhost:4200/dashboard**

Right now, every part is accessible.

**Step 3: Create an auth service**

The auth service is responsible for returning a boolean value. If it will return **true,** then the user is logged in; otherwise, it is not logged in and returns **false**.

To create a service in Angular, type the following command.

ng g s auth --skipTests=true

The next step is to register the auth service inside the **app-routing.module.ts**file.

*// app-routing.module.ts*

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

**@NgModule**({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule],

providers: [AuthService]

})

Now, write the following function inside the **auth.service.ts** file.

*// auth.service.ts*

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

**@Injectable**({

providedIn: 'root'

})

**export** **class** AuthService {

**constructor**() { }

isLoggedIn(): boolean {

**return** false;

}

}

Right now, we are returning false. That means the user is not authenticated.

For proper authentication in Angular, please check out the [Angular JWT Authentication](https://appdividend.com/2020/07/09/angular-authentication-system-login-and-registration-in-angular/) Tutorial.

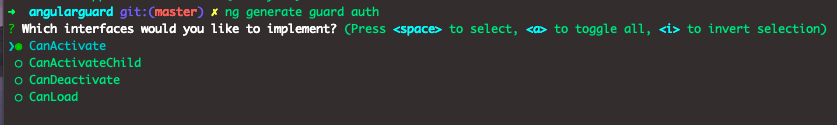
Okay, let’s create an auth guard.

**Step 4: Create an Angular route guard as a service**

To create a service for your guard, type the following command.

ng **generate** guard auth --skipTests=true

You will get to choose which type of guard you want to create, as in the following image.

[](https://appdividend.com/wp-content/uploads/2020/07/Angular-Route-Guard-Example.png)

I am choosing the CanActivate guard.

Your **auth.guard.ts**file will be created and looks like this.

*// auth.guard.ts*

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

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

**import** { Observable } **from** 'rxjs';

**@Injectable**({

providedIn: 'root'

})

**export** **class** AuthGuard **implements** CanActivate {

canActivate(

next: ActivatedRouteSnapshot,

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

**return** true;

}

}

Interface that a class can implement to be a guard deciding if the route can be activated. If all the guards return true, navigation will continue. If any guard returns false, the navigation will be canceled.

If any guard returns the UrlTree, current navigation will be canceled, and new navigation will be kicked off to the UrlTree returned from a guard.

Now, import the **auth.service.ts**file inside this guard.

After that, we will complete the **canActivate()**function.

Based on if the user is authenticated or not, the canActivate() function will return **true**or**false.**

Write the following code inside the **auth.guard.ts**file.

*// auth.guard.ts*

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

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

**import** { Observable } **from** 'rxjs';

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

**@Injectable**({

providedIn: 'root'

})

**export** **class** AuthGuard **implements** CanActivate {

**constructor**(**private** auth: AuthService) {}

canActivate(

next: ActivatedRouteSnapshot,

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

**if** (**this**.auth.isLoggedIn()) {

**return** true;

}

window.alert('You don\'t have permission to view this page');

**return** false;

}

}

Here, if the user is logged in, it won’t prevent accessing the page; otherwise, we will show the user alert that you don’t have permission to view this page.

You can see that we have injected the auth service in the constructor to use its function.

If the user is logged in, the guard passes and lets the user through.

If the user is not logged in, the guard fails, we show the user an alert, and the page doesn’t navigate to the intended URL.

**Step 5: Attach the Auth Guard in the routing module.**

In your routing module, use the appropriate property in your **routes** configuration.

Add the following code inside the **app-routing.module.ts**file.

*// app-routing.module.ts*

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

const routes: Routes = [

{ path: '', component: HomeComponent },

{ path: 'home', component: HomeComponent},

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

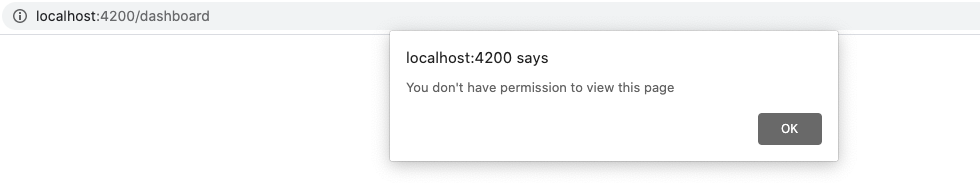
];

Here, **canActivate** tells the router to mediate navigation to this particular route.

Okay, now, let’s try to access the **dashboard route.**

Go to the browser and type this URL: **http://localhost:4200/dashboard.**

You will see something like this.

[](https://appdividend.com/wp-content/uploads/2020/07/Angular-10-Guards.png)

That is it. We have successfully prevented access using the Auth guard.

Now, let’s return **true**from **auth.service.ts** file’s **isLoggedIn()**function.

*// auth.service.ts*

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

**@Injectable**({

providedIn: 'root'

})

**export** **class** AuthService {

**constructor**() { }

isLoggedIn(): boolean {

**return** true;

}

}

Now, we should be able to access the dashboard component or route, or page.

Let’s try one more time. Go to the **http://localhost:4200/dashboard.**

Bingo!! You now have access to the page. See the following output.

[](https://appdividend.com/wp-content/uploads/2020/07/Auth-guard-example.png)

I have also put the code on Github. So, please check out that as well.

[GITHUB CODE](https://github.com/KrunalLathiya/AngularRouteGuard)

For more information on route guards, check out official [Angular documentation](https://angular.io/guide/router).

**Conclusion**

In this Angular Route guards tutorial, we have seen the following things.

1. How to create a guard using a default command.
2. How to inject Angular Service into Guard Service.
3. How to use canActivate guard to prevent access to a specific page.

That is it for the Angular route guard example.