Angular Guards Tutorial

3 Comments / 9 minutes of reading / February 1, 2024

← ActivatedRoute | Angular Tutorial | CanActivate Guard →

Angular Route Guards help us prevent the user from accessing certain parts of the applications under specific conditions. We can also use it to perform some actions before navigating to a route or leaving the route. The use cases for route guards are authorization, authentication, data collection, etc. The Angular supports several guards like CanActivate, CanDeactivate, Resolve, CanLoad, CanActivateChild, etc. This article will explore Angular Guards in detail by building an example Angular Guards application.

Table of Contents

Angular Route Guards

We use the Angular Guards to control whether users can navigate to or away from the current route.

We looked at how to configure our routes and navigate to the different parts of our application in our Angular Router Tutorial. Allowing the user to navigate all application parts is not a good idea. We must restrict the user until the user performs specific actions like logging in etc. Angular provides the **Route Guards** for this purpose.

One of the common scenarios where we use Route guards is authentication. We want our app to stop the unauthorized user from accessing the protected route. We achieve this by using the **CanActivate guard**, which angular invokes when the user tries to navigate into the protected route. Then we hook into the **CanActivate guard** and use the authentication service to check whether the user is authorized to use the route or not. If he is not authenticated, then we can redirect the user to the login page.

Uses of Angular Route Guards

- To Confirm the navigational operation
- Asking whether to save before moving away from a view
- Allow access to certain parts of the application to specific users.

- Validating the route parameters before navigating to the route
- Fetching some data before you display the component.

Types of Route Guards

The Angular Router supports six different guards, which you can use to protect the route.

1. CanActivate
2. CanDeactivate
3. Resolve
4. CanLoad
5. CanActivateChild
6. CanMatch

CanActivate

The Angular CanActivate guard decides if a route can be activated (or the component gets rendered). We use this guard when we want to check on some condition, before activating the component or showing it to the user. This allows us to cancel the navigation.

Use cases

- Checking if a user has logged in
- Checking if a user has permission

One of the use cases of this guard is to check if the user has logged in to the system. If the user has not logged in, then the guard can redirect him to the login page.

Read: Angular CanActivate Guard Example

CanDeactivate

This Angular Guard decides if the user can leave the component (navigate away from the current route). This route is useful when the user might have some pending changes, which is not yet saved. The CanDeactivate route allows us to ask for user confirmation before leaving the component. You might ask the user if it's OK to discard pending changes rather than save them.

Read: Angular CanDeactivate Guard Example

Resolve

This guard delays the activation of the route until some tasks are complete. You can use the guard to pre-fetch the data from the backend API before activating the route.

CanLoad

The `CanLoad` Guard prevents the loading of the Lazy Loaded Module. We generally use this guard when we do not want to unauthorized users to be able even to see the source code of the module.

This guard works similarly to CanActivate guard with one difference. The CanActivate guard prevents a particular route from being accessed. The CanLoad prevents the

entire lazy-loaded module from being downloaded, protecting all the `routes` within that module.

Read: Angular CanLoad Guard Example

CanActivateChild

CanActivateChild guard determines whether a child route can be activated. This guard is very similar to CanActivateGuard. We apply this guard to the parent route. The Angular invokes this guard whenever the user tries to navigate to any of its child routes. This allows us to check some conditions and decide whether to proceed with the navigation or cancel it.

Read: Angular CanActivateChild Guard Example

How to Build Angular Route Guards

Building the Guards is very easy.

1. Build the Guard as Service.
2. Implement the Guard Method in the Service
3. Register the Guard Service in the Root Module
4. Update the Routes to use the guards

1. Build the Guard as a Service

Building the Guard Service is as simple as building any other Angular Service. You need to import the corresponding guard from the Angular Router Library using the Import statement. For Example use `CanActivate` Guard import the `CanActivate` in the import `the CanActivate` in the import statement

```

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

```

Next, create the Guard class, which implements the selected guard Interface as shown below.

```

@Injectable()
export class ProductGuardService implements CanActivate {}

```

You can also inject other services into the Guards using the Dependency Injection

2. Implement the Guard Method

The next step is to create the Guard Method. The name of the Guard method is the same as the Guard it implements. For Example, to implement the `CanActivate` guard, create a method `CanActivate`

```

canActivate(): boolean {
    // Check weather the route can be activated;
    return true;
    // or false if you want to cancel the navigation;
}

```

```
7
```

The return value from the Guard

The guard method must return either a True or a False value.

If it returns true, the navigation process continues. if it returns false, the navigation process stops, and the user stays put.

The above method returns a True value. The Guard can also return an Observable or a Promise, which eventually returns a True or false. The Angular will keep the user waiting until the guard returns true or false.

The guard can also tell the router to navigate elsewhere, effectively canceling the current navigation.

3. Register the Guard as Service in Module

As mentioned earlier, guards are nothing but services. We need to register them with the providers array of the Angular Module as shown below.

```
1
2 providers: [ProductService,ProductGuardService]
3
```

The Angular router requires the Guards and all other services the guard depends on to be available during the navigation. Hence, the guards must be provided at the module level. This allows the router to access the guards using the Dependency Injection.

4. Update the Routes

Finally, we need to add the guards to the routes array, as shown below

```
{ path: 'product', component: ProductComponent, canActivate : [ProductGuardService]
}
```

The above code adds the `canActivate` guard (ProductGuardService) to the `Product` route.

When the user navigates to the Product route, Angular calls the canActivate method from the ProductGuardService. If the method returns true, then the `ProductComponent` is rendered.

You can add more than one guard, as shown below

```
{ path: 'product',
  component: ProductComponent,
  canActivate : [ProductGuardService, AnotherProductGuardService ]
}
```

The syntax for adding other guards is also similar

```
{ path: 'product', component,
```

```
    canActivate : any[],
    canActivateChild: any[],
    canDeactivate: any[],
    canLoad: any[],
    resolve: any[]
}

```

Order of execution of route guards

A route can have multiple guards, and you can have guards at every level of a routing hierarchy.

CanDeactivate() and CanActivateChild() guards are always checked first. The checking starts from the deepest child route to the top.

CanActivate() Guard is checked next, and checking starts from the top to the deepest child route.

CanLoad() Is invoked next If the feature module is to be loaded asynchronously.

Resolve() Guard is invoked last.

The Angular Router cancels the navigation If any of the guards return false.

Angular Guards Example

Let us update the App, which we built in the previous tutorials on Angular Routers, and use the CanActivate Guard to prevent the user from activating the ProductComponent

The source code for this tutorial is available on GitHub

Guard Service

We create Guard classes as a service. Create a file named product-guard.service.ts in the src/app folder and add the following code

```
1
2  import { Injectable } from '@angular/core';
3  import { Router, CanActivate, ActivatedRouteSnapshot,RouterStateSnapshot } from '@angu
4
5  @Injectable()
6  export class ProductGuardService implements CanActivate {
7
8    constructor(private _router:Router ) {
9    }
10
11   canActivate(route: ActivatedRouteSnapshot,
12               state: RouterStateSnapshot): boolean {
13       console.log("canActivate");      //return true
14      //remove comments to return true
15       alert('You are not allowed to view this page. You are redirected to Home Page');
16       this._router.navigate(["home"]);
17       return false;
18  } }
19
```

First, we need to import the Router, CanActivate, ActivatedRouteSnapshot & RouterStateSnapshot libraries from the angular/core package

```
1
```

```
2 import { Router, CanActivate, ActivatedRouteSnapshot,RouterStateSnapshot } from '@angul
3
```

Define the ProductGuardService, which implements the CanActivate Interface

```
1
2 export class ProductGuardService implements CanActivate {
3
```

Finally, define the `canActivate` method

The `canActivate` method accepts two arguments. The first argument is an `ActivatedRouteSnapshot` object, which describes the route that is being navigated to using the properties. The second argument is a `RouterStateSnapshot` object, which describes the current route through a single property called URL.

```
1
2 canActivate(route: ActivatedRouteSnapshot,
3         state: RouterStateSnapshot): boolean {
4
5    console.log("canActivate");     //return true
6    //remove comments to return true
7    alert('You are not allowed to view this page. You are redirected to Home Page');
8    //this._router.navigate(["home"]); //navigate to some other route;
9    return false;
10 }
11
```

Import the Guard in the Root Module

```
1
2 import { BrowserModule } from '@angular/platform-browser';
3 import { NgModule } from '@angular/core';
4 import { FormsModule } from '@angular/forms';
5 import { HttpModule } from '@angular/http';
6 import { RouterModule } from '@angular/router';
7 import { AppComponent } from './app.component';
```

```
import { HomeComponent} from './home.component';
import { ContactComponent} from './contact.component';
import { ProductComponent} from './product.component';
import { ErrorComponent} from './error.component';
import { ProductDetailComponent} from './product-detail.component';
import { ProductService } from './product.service';
import { ProductGuardService } from './product-guard.service';
import { appRoutes } from './app.routes';

@NgModule({
  declarations: [ AppComponent,HomeComponent,ContactComponent,
                ProductComponent,ErrorComponent,
                ProductDetailComponent],
  imports:      [ BrowserModule, FormsModule,HttpModule,
                RouterModule.forRoot(appRoutes)],
  providers:    [ ProductService,ProductGuardService ],
  bootstrap:    [ AppComponent]
})
export class AppModule { }

```

First, Import the Guard Service as shown below

```

import { ProductGuardService } from './product-guard.service';

```

Next, Register it using the `Providers` metadata so that that router can use it. Remember that Guards must be provided at the angular module level

```

providers: [ProductService,ProductGuardService],

```

Update the Routes

Finally, Update the `app.routes` class

```

```

```
import { Routes } from '@angular/router';
import { HomeComponent} from './home.component';
import { ContactComponent} from './contact.component';
import { ProductComponent} from './product.component';
import { ErrorComponent} from './error.component';
import { ProductDetailComponent} from './product-detail.component';
import { ProductGuardService } from './product-guard.service';

export const appRoutes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'contact', component: ContactComponent },
  { path: 'product', component: ProductComponent, canActivate :[ProductGuardService] },
  { path: 'product/:id', component: ProductDetailComponent },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: '**', component: ErrorComponent }
];

```

The only change we have made is to attach the `ProductGuardService` to the `CanActivate` guard

```

{ path: 'product',
  component: ProductComponent,
  canActivate : [ProductGuardService] },

```

Test the Guard. Run the app, and you will see the alert message “You are not allowed to view this page”. You are redirected to the Home Page.

Summary

The angular Guards are a great tool that helps us to protect the route. They also help us run some logic, get data from the back-end server, etc. You can also create multiple guards against a single route or use the same guard against multiple routes.

← ActivatedRoute | Angular Tutorial | CanActivate Guard →