Angular CanActivate Guard Example

14 Comments / 7 minutes of reading / March 9, 2023

← Route Guards | Angular Tutorial | CanActivateChild →

The Angular CanActivate guard runs before we navigate to a route allowing us to cancel the navigation. In this tutorial, we will learn what is `CanActivate` guard is and how to use it to protect the route. We will build a simple `CanActivate` Example app to show you how to use it in real application

Table of Contents

- What is CanActivate Guard
  - Use cases for the CanActivate Guard
- How to use CanActivate Guard
- CanActivate guard Example
  - LoginComponent
  - ProductComponent
  - Other Component
  - CanActivate Guard
- Reference

What is CanActivate Guard

The Angular CanActivate guard decides, if a `route` can be activated (or 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 for the CanActivate Guard

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

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

How to use CanActivate Guard

First, we need to create a Angular Service.

The service must import & implement the `CanActivate` Interface. This Interface is defined in the `@angular/router` module. The Interface has one method i.e. `canActivate`. We need to implement it in our Service. The details of the `CanActivate` interface is as shown below.

```
1
2 interface CanActivate {
3   canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boo
4 }
5
```

The method gets the instance of the `ActivatedRouteSnapshot` & `RouterStateSnapshot`. We can use this to get access to the route parameter, query parameter etc.

The guard must return true/false or a UrlTree . The return value can be in the form of observable or a promise or a simple `boolean` value.

A route can have more than one `canActivate` guard.

If **all guards** returns `true`, navigation to the route will continue.

If **any one of the guard** returns `false`, navigation will be cancelled.

If **any one of the guard** returns a `UrlTree`, current navigation will be cancelled and a new navigation will be kicked off to the `UrlTree` returned from the guard.

The example of `canActivate` guard is as follows

```
1
2  import { Injectable } from '@angular/core';
3  import { Router, CanActivate, ActivatedRouteSnapshot,RouterStateSnapshot } from '@angu
4
5
6  @Injectable()
7  export class AuthGuardService implements CanActivate {
8
9     constructor(private _router:Router ) {
10    }
11
12    canActivate(route: ActivatedRouteSnapshot,
13              state: RouterStateSnapshot): boolean {
14
15       //check some condition
16       if (someCondition)  {
```

```
17            alert('You are not allowed to view this page');
18            //redirect to login/home page etc
19            //return false to cancel the navigation
20            return false;
21        }
22        return true;
23    }
24
25 }
26
```

Update the route definition with the `canActivate` guard as shown below. You can apply more than one guard to a route and a route can have more than one guard

```
1
2 { path: 'product', component: ProductComponent, canActivate : [AuthGuardService] },
3
```

CanActivate guard Example

In our example application, we will create three components. The `HomeComponent` & `ContactComponent` are not protected and can be accessed any user. The user must log in into the system to access the `ProductComponent` ..We also need a `LoginComponent` to handle the user login.

LoginComponent

The following is the code for `LoginComponent` and associated `AuthService`

[tabby title="login.component.ts"]

```
1
2 import { Component, OnInit } from '@angular/core';
3 import { FormControl, FormGroup } from '@angular/forms';
4 import { Router, ActivatedRoute } from '@angular/router';
5 import { AuthService } from './auth.service';
6
```

```
@Component({
  templateUrl: './login.component.html',
  styles: [``]
})
export class LoginComponent implements OnInit {

   invalidCredentialMsg: string;
   username:string;
   password:string;
   retUrl:string="home";

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

   ngOnInit() {
      this.activatedRoute.queryParamMap
            .subscribe(params => {
         this.retUrl = params.get('retUrl');
         console.log( 'LoginComponent/ngOnInit '+ this.retUrl);
      });
   }

   onFormSubmit(loginForm) {
     this.authService.login(loginForm.value.username, loginForm.value.password).subscrib
        console.log( 'return to '+ this.retUrl);
        if (this.retUrl!=null) {
            this.router.navigate( [this.retUrl]);
        } else {
            this.router.navigate( ['home']);
        }
     });
   }
}

```

[tabby title="login.component.html"]

```

<h3>Login Form</h3>

<div>
 <form #loginForm="ngForm" (ngSubmit)="onFormSubmit(loginForm)">
  <p>User Name: <input type='text'  name='username' ngModel></p>
  <p>Password: <input type="password"  name="password" ngModel></p>
```

```
  <p><button type="submit">Submit</button></p>
 </form>
</div>

```

[tabby title=”auth.service.ts”]

```

import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/of';
import 'rxjs/add/operator/map';
import { of } from 'rxjs';

@Injectable()
export class AuthService {

    private isloggedIn: boolean;
    private userName:string;

    constructor() {
        this.isloggedIn=false;
    }

    login(username: string, password:string) {

        //Assuming users are provided the correct credentials.
        //In real app you will query the database to verify.
        this.isloggedIn=true;
        this.userName=username;
        return of(this.isloggedIn);
    }

```

```
27    isUserLoggedIn(): boolean {
28       return this.isloggedIn;
29    }
30
31    isAdminUser():boolean {
32       if (this.userName=='Admin') {
33          return true;
34       }
35       return false;
36    }
37
38    logoutUser(): void{
39       this.isloggedIn = false;
40    }
41
42 }
43
```

[tabbyending]

The AuthService checks whether the user is allowed to login. It has the method to login & logout the users. Our implementation of the login method does not check for anything. It just marks the user as logged in.

ProductComponent

The ProductComponent is our protected component. Only the logged in users can access this. This component displays the list of Products, which it gets from the ProductService .

[tabby title="product.component.ts"]

```
1
2  import { Component, OnInit } from '@angular/core';
3  import { ProductService } from './product.service';
4  import { Product } from './Product';
5
6
7  @Component({
```

```
  templateUrl: "product.component.html",
})
export class ProductComponent
{

  products:Product[];
  constructor(private productService: ProductService){
  }

  ngOnInit() {

    this.productService.getProducts()
      .subscribe(data => {
        this.products=data;
      })
  }

}

```

[tabby title="product.component.html"]

```

<h1>Product List</h1>
  <p> This is a protected component </p>

  <div class='table-responsive'>
  <table class='table'>
     <thead>
        <tr>
           <th>Name</th>
           <th>Price</th>
        </tr>
     </thead>
     <tbody>
        <tr *ngFor="let product of products;">
           <td><a>{{product.name}} </a> </td>
           <td>{{product.price}}</td>
        </tr>
     </tbody>
    </table>
 </div>

```

[tabby title="product.service.ts"]

```
import {Product} from './Product'
import { of, Observable, throwError} from 'rxjs';
import { delay, map } from 'rxjs/internal/operators';

export class ProductService{

   products: Product[];

   public constructor() {
      this.products=[
         new Product(1,'Memory Card',500),
         new Product(2,'Pen Drive',750),
         new Product(3,'Power Bank',100),
         new Product(4,'Computer',100),
         new Product(5,'Laptop',100),
         new Product(6,'Printer',100),
      ]
   }

   public getProducts(): Observable<Product[]> {
      return of(this.products) ;
   }

   public getProduct(id): Observable<Product> {
      var Product= this.products.find(i => i.productID==id)
      return of(Product) ;
   }

}

```

[tabby title="product.ts"]

```
export class Product {

   constructor(productID:number,    name: string ,   price:number) {
      this.productID=productID;
      this.name=name;
      this.price=price;
   }
```

```
 9
10    productID:number ;
11    name: string ;
12    price:number;
13
14 }
15
```

[tabbyending]

Other Component

[tabby title="app.component.ts"]

```
 1
 2 import { Component } from '@angular/core';
 3 import { AuthService } from './auth.service';
 4 import { Router } from '@angular/router';
 5
 6 @Component({
 7   selector: 'app-root',
 8   templateUrl: './app.component.html'
 9 })
10 export class AppComponent {
11   title = 'Routing Module - Route Guards Demo';
12
13   constructor (private authService:AuthService,
14               private router:Router) {
15   }
16
17   logout() {
18     this.authService.logoutUser();
19     this.router.navigate(['home']);
20   }
21
22 }
23
```

[tabby title="app.component.html"]

```
<div class="container">

<nav class="navbar navbar-default">
  <div class="container-fluid">
    <div class="navbar-header">
      <a class="navbar-brand" [routerLink]="['/']"><strong> {{title}} </strong></a>
    </div>
    <ul class="nav navbar-nav">

        <li><a [routerLink]="['home']">Home</a></li>
        <li><a [routerLink]="['product']">Product</a></li>
        <li><a [routerLink]="['contact']">Contact us</a></li>
        <li><a [routerLink]="['login']">Login</a></li>
        <li><a [routerLink]="" (click)="logout()">Log out</a></li>

    </ul>
  </div>
</nav>

 <router-outlet></router-outlet>

</div>

```

[tabby title="home.component.ts"]

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

@Component({
    template: `<h1>Welcome!</h1>
            <p>This is Home Component </p>
          `
})
```

```
export class HomeComponent {
}
```

[tabby title="contact.component.ts"]

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

@Component({
    template: `<h1>Contact Us</h1>
            <p>TekTutorialsHub </p>
            `
})
export class ContactComponent {
}
```

[tabbyending]

CanActivate Guard

Finally, we build a CanActivate guard, which will check whether the users are logged in or not. If users are not logged in, then they are redirected to the login page.

[tabby title="auth-guard.service.ts"]

```
import { Injectable } from '@angular/core';
import { Router, CanActivate, ActivatedRouteSnapshot,RouterStateSnapshot, UrlTree } from
import { AuthService } from './auth.service';


@Injectable()
export class AuthGuardService implements CanActivate {

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

```

```
12    }
13
14    canActivate(route: ActivatedRouteSnapshot,
15              state: RouterStateSnapshot): boolean|UrlTree {
16
17       if (!this.authService.isUserLoggedIn()) {
18          alert('You are not allowed to view this page. You are redirected to login Page');
19
20          this.router.navigate(["login"],{ queryParams: { retUrl: route.url} });
21          return false;
22
23          //var urlTree = this.router.createUrlTree(['login']);
24          //return urlTree;
25       }
26
27       return true;
28    }
29
30 }
31
```

[tabbyending]

First, we import the CanActivate from the @angular/router module.

The AuthGuardService implements the CanActivate interface

Inject the AuthServce in the constructor of the Guard

In the `CanActivate` method, we will redirect the user the login page, if the user is not logged in. To cancel the navigation ,we must either return `false` or `urlTree` as shown in the example above.

Next, we will update the route definition and use the guard in all the routes, which we want to protect

[tabby title="app.routes.ts"]

```
1
2  import { Routes } from '@angular/router';
3
4  import { HomeComponent} from './home.component'
5  import { ContactComponent} from './contact.component'
6  import { ProductComponent} from './product.component'
7
8  import { AuthGuardService } from './auth-guard.service';
9  import { LoginComponent } from './login.component';
10
11
12 export const appRoutes: Routes = [
13   { path: 'home', component: HomeComponent },
14   { path: 'login', component:LoginComponent},
15   { path: 'contact', component: ContactComponent },
16   { path: 'product', component: ProductComponent, canActivate : [AuthGuardService] },
17   { path: '', redirectTo: 'home', pathMatch: 'full' },
18 ];
19
```

[tabbyending]

Finally, register the service in the `app.module` .

[tabby title="app.module.ts"]

```
1
```

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpModule } from '@angular/http';
import { FormsModule }    from '@angular/forms';

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

import { AppComponent } from './app.component';
import { HomeComponent} from './home.component'
import { ContactComponent} from './contact.component'
import { ProductComponent} from './product.component'

import { AuthGuardService } from './auth-guard.service';

import { appRoutes } from './app.routes';
import { AuthService } from './auth.service';
import { LoginComponent } from './login.component';
import { ProductService } from './product.service';

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


```

[tabbyending]

Run the app. You can access the Product page only if you login as shown in the image below.