Angular CanActivateChild Example

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

← CanActivate | Angular Tutorial | CanDeactivate Guard →

The Angular `CanActivateChild` guard runs before we navigate to a child route. In this tutorial, we will learn what is `CanActivateChild` guard is and how to use it to protect the child routes. We will build a simple Angular `CanActivateChild` example app to show you how to use it in the real application.

Table of Contents

What is CanActivateChild Guard

The `CanActivatechild` 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.

Difference between CanActivate & CanActivateChild

Consider the following routes.

The `ProductComponent` displays the list of products. We have attached the canActivate guard to the product route. The canActivate guard blocks access to the route, if the user is not logged in. This guard protects both the `product` route and all its children.

```
{ path: 'product', component: ProductComponent, canActivate : [AuthGuardService],
    children: [
    { path: 'view/:id', component: ProductViewComponent },
    { path: 'edit/:id', component: ProductEditComponent },
    { path: 'add', component: ProductAddComponent }
    ]
 },
```

Now, consider the case where we want all users to view the `ProductComponent`, but only the Admin user can view any of its child routes

We can create another guard `ProductGuardService` that implements the `canActivate` guard and attach it to each of those child routes as shown below.

```
{ path: 'product', component: ProductComponent, canActivate : [AuthGuardService],
    children: [
    { path: 'view/:id', component: ProductViewComponent, canActivate : [ProductGuardSer
    { path: 'edit/:id', component: ProductEditComponent, canActivate : [ProductGuardServi
    { path: 'add', component: ProductAddComponent, canActivate : [ProductGuardService]
    ]
 },
```

Another way is to use the `CanActivateChild` guard and attach it to the product route as shown below. When Angular sees a `canActivateChild` guard attached to the parent

route, it invokes it every time the user tries to navigate to the child route. Hence instead of attaching Guard service to every child, you can attach it to the parent route.

```
{ path: 'product', component: ProductComponent, canActivate : [AuthGuardService],
canActivateChild : [AdminGuardService],
    children: [
    {  path: 'view/:id', component: ProductViewComponent  },
    {  path: 'edit/:id', component: ProductEditComponent  },
    {  path: 'add', component: ProductAddComponent }
    ]
 },
```

How to Create CanActivateChild Guard

Just like all other Angular Guards, we need to create an Angular Service. The service must import & implement the `CanActivateChild` Interface. The Interface is defined in the `@angular/router module` . The Interface has one method i.e. `canActivateChild` . The details of the `CanActivateChild` interface are as shown below.

```
interface CanActivateChild {
  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boo
}
```

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 . It can return these values either as an observable or a promise or as a simple Boolean value.

If all guards return **true**, navigation will continue.

If any guards return **false**, navigation will be cancelled.

If any 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 `canActivateChild` guard is as follows

```
import { Injectable } from '@angular/core';
import { Router, CanActivate, ActivatedRouteSnapshot,RouterStateSnapshot } from '@angu


@Injectable()
export class AuthGuardService implements CanActivateChild {

   constructor(private _router:Router ) {
   }

   canActivate(route: ActivatedRouteSnapshot,
            state: RouterStateSnapshot): boolean {

      //check some condition
      if (someCondition)  {
         alert('You are not allowed to view this page');
         //redirect to login/home page etc
         //return false to cancel the navigation
         return false;
      }
      return true;
   }
```

```
24
25 }
26
```

Angular CanActivateChild Example

In our example application, the `HomeComponent` & `ContactComponent` are not protected and can be accessed by any user.

The user must log in to the system to access the `ProductComponent` ..

The ProdudctEditComponent, and ProductViewComponets are child components of the ProductComponent

We also need `LoginComponent` to handle the user login.

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
7 @Component({
8   templateUrl: './login.component.html',
9   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']);
        }
     });
   }
}

```

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>

```

auth.service.ts

```
1
2  import { Injectable } from '@angular/core';
3  import { Observable } from 'rxjs/Observable';
4  import 'rxjs/add/observable/of';
5  import 'rxjs/add/operator/map';
6  import { of } from 'rxjs';
7
8
9  @Injectable()
10 export class AuthService {
11
12    private isloggedIn: boolean;
13    private userName:string;
14
15    constructor() {
16       this.isloggedIn=false;
17    }
18
19    login(username: string, password:string) {
20
21       //Assuming users are provided the correct credentials.
22       //In real app you will query the database to verify.
23       this.isloggedIn=true;
24       this.userName=username;
25       return of(this.isloggedIn);
26    }
27
28    isUserLoggedIn(): boolean {
29       return this.isloggedIn;
30    }
31
32    isAdminUser():boolean {
33       if (this.userName=='Admin') {
34          return true;
35       }
36       return false;
37    }
38
39    logoutUser(): void{
40       this.isloggedIn = false;
41    }
42
43 }
44
```

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. `isAdminUser()` method returns true if the user is logged in with the user name "Admin".

Product Component

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`.

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({
8    templateUrl: "product.component.html",
9  })
10 export class ProductComponent
11 {
12
13   products:Product[];
14   constructor(private productService: ProductService){
15   }
16
17   ngOnInit() {
18
19     this.productService.getProducts()
20       .subscribe(data => {
21         this.products=data;
22       })
23   }
24
25 }
26
```

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>
           <th></th>
           <th></th>
        </tr>
     </thead>
     <tbody>
        <tr *ngFor="let product of products;">
           <td><a [routerLink]="['/product/view',product.productID]">{{product.name}}
           <td>{{product.price}}</td>
           <td><a [routerLink]="['/product/view',product.productID]">View</a></td>
           <td><a [routerLink]="['/product/edit',product.productID]">Edit</a></td>
        </tr>
     </tbody>
   </table>
 </div>

 <router-outlet></router-outlet>

```

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);
   }

}

```

product.ts

```

export class Product {

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

   productID:number ;
   name: string ;
   price:number;

}

```

We have Product add, edit & view components.

product-add.component.ts

```
1
2  import { Component, OnInit } from '@angular/core';
3  import { Product } from './Product';
4  import { ProductService } from './product.service';
5  import { ActivatedRoute } from '@angular/router';
6
7
8
9  @Component({
10   template: `<h1>Add Product</h1>`,
11 })
12 export class ProductAddComponent
13 {
14
15   product:Product;
16
17   constructor(private productService: ProductService,
18             private route:ActivatedRoute ){
19   }
20
21
22   ngOnInit() {
23   }
24
25 }
26
```

product-edit.component.ts

```
1
2  import { Component, OnInit } from '@angular/core';
3  import { Product } from './Product';
4  import { ProductService } from './product.service';
5  import { ActivatedRoute } from '@angular/router';
6
7
8  @Component({
9   template: `<h1>Edit Product</h1>`,
10 })
11 export class ProductEditComponent
12 {
13
14     product:Product
```

```
15
16    constructor(private productService: ProductService,
17              private route:ActivatedRoute ){
18    }
19
20
21    ngOnInit() {
22
23    let id=this.route.snapshot.params['id'];
24
25    this.productService.getProduct(id)
26       .subscribe(data => {
27          this.product=data;
28       })
29    }
30 }
31
```

product-view.component.ts

```
1
2  import { Component, OnInit } from '@angular/core';
3  import { Product } from './Product';
4  import { ProductService } from './product.service';
5  import { ActivatedRoute } from '@angular/router';
6
7
8  @Component({
9   template: `<h1>View Product</h1>`,
10 })
11 export class ProductViewComponent
12 {
13
14    product:Product
15
16    constructor(private productService: ProductService,
17              private route:ActivatedRoute ){
18    }
19
20
21    ngOnInit() {
22
23
24       let id=this.route.snapshot.params['id'];
25
26       this.productService.getProduct(id)
27          .subscribe(data => {
28             this.product=data;
```

```
        })
    }

}

```

Other Components

app.component.ts

```
import { Component } from '@angular/core';
import { AuthService } from './auth.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  title = 'Routing Module - Route Guards Demo';

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

  logout() {
    this.authService.logoutUser();
    this.router.navigate(['home']);
  }

}


```

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>

```

home.component.ts

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

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

export class HomeComponent {
}

```

contact.component.ts

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

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

```

CanActivateChild Guard

Next, we will build CanActivate guard, which will check whether the users are logged in or not. The users are redirected to the login page if they are not logged in.

Also, CanActivateChild guard, which checks whether the user is Admin User. Only the Admin Users are allowed to navigate to the ProductEditComponent & ProductViewComponent.

auth-guard.service.ts

```
1
2
3  import { Injectable } from '@angular/core';
4  import { Router, CanActivate, ActivatedRouteSnapshot,RouterStateSnapshot, UrlTree, CanA
5  import { AuthService } from './auth.service';
6
7
8  @Injectable()
9  export class AuthGuardService implements CanActivate , CanActivateChild {
10
11     constructor(private router:Router, private authService: AuthService ) {
12
13     }
14
15     canActivate(route: ActivatedRouteSnapshot,
16                 state: RouterStateSnapshot): boolean|UrlTree {
17
18        console.log('canActivate on '+route.url);
19
20        if (!this.authService.isUserLoggedIn()) {
21           alert('You are not allowed to view this page. You are redirected to login Page');
22           this.router.navigate(["login"],{ queryParams: { retUrl: route.url } });
```

```
23          return false;
24
25          //var urlTree = this.router.createUrlTree(['login']);
26          //return urlTree;
27       }
28
29       return true;
30    }
31
32    canActivateChild(route: ActivatedRouteSnapshot,
33                state: RouterStateSnapshot): boolean|UrlTree {
34
35
36
37       if (!this.authService.isAdminUser()) {
38          alert('You are not allowed to view this page');
39          return false;
40       }
41
42
43       return true;
44 }
45
46 }
47
```

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

The AuthGuardService implements both CanActivate & CanActivateChild interface

Inject the AuthServce in the constructor of the Guard

In the CanActivate method, we will redirect the user to 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.

While in `CanActivateChild` method, we just check if the user is Admin, then we will return true else return false

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

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 import { ProductViewComponent } from './product-view.component';
11 import { ProductAddComponent } from './product-add.component';
12 import { ProductEditComponent } from './product-edit.component';
13
14
15 export const appRoutes: Routes = [
16   { path: 'home', component: HomeComponent },
17   { path: 'login', component:LoginComponent},
18   { path: 'contact', component: ContactComponent },
19   { path: 'product', component: ProductComponent, canActivate : [AuthGuardService] ,
20      canActivateChild : [AuthGuardService],
21      children: [
22      {  path: 'view/:id', component: ProductViewComponent  },
23      {  path: 'edit/:id', component: ProductEditComponent  },
24      {  path: 'add', component: ProductAddComponent }
25      ]
26   },
27
28   { path: '', redirectTo: 'home', pathMatch: 'full' },
29 ];
30
31
```

app.module.ts

```
1
2  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 { ProductAddComponent } from './product-add.component';
import { ProductViewComponent } from './product-view.component';
import { ProductEditComponent } from './product-edit.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 { }


```

We apply both guards on the product route. The canActivate guards protect the product route and canActivateChild protects all its child routes.

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