



Router Guards

Securing parts of your route

Guard Types

- Four types of guards:
 - `CanActivate` – decides if a route can be activated
 - `CanActivateChild` – decides if children of a route can be activated
 - `CanDeactivate` – decides if a route can be deactivated
 - `CanLoad` – decides if a module can be loaded lazily


Defining Guards

- Multiple ways (as functions or as classes)
- Regardless, it needs to return a
 - `Observable<boolean>`,
 - `Promise<boolean>` or
 - `boolean`.
- Defined in `@NgModule`, or as a separate class

Guards as a function

- Define a token and a guard function. For example in `app.module.ts`.

```
@NgModule({  
  ...  
  providers: [  
    CityService,  
    AuthService,  
    {  
      provide: 'CanAlwaysActivateGuard', // Guard as a function  
      useValue: guardFunction  
    },  
    CanActivateViaAuthGuard,  
    CanDeactivateGuard  
  ],  
  bootstrap: [MainComponent]  
})  
export class AppModule {}
```



```
export function guardFunction() {  
  console.log('Route requested');  
  return true; // do validation or other stuff here  
}
```

Use the guard token in app.routes

// app.routes.ts

```
...  
export const AppRoutes: Routes = [  
  ...  
  {  
    path: 'home',  
    component: AppComponent,  
    canActivate: ['CanAlwaysActivateGuard'] // Defined in app.module.ts  
  },  
  ...  
];
```



**(re)use of string
token**

You *can* have multiple tokens/functions, guarding your route

Guards as a class – *most used*

- Used: when the guard needs Dependency Injection
- Common use: with some kind of Authentication Service.
- All about Implementing interfaces!
 - `canActivate()`
 - `canActivateChild()`
 - `canDeActivate()`

canActivateViaAuthGuard.ts

```
// canActivateViaAuthGuard.ts
```

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

```
@Injectable()
```

```
export class CanActivateViaAuthGuard implements CanActivate {
```

```
  constructor(private authService: AuthService) {}
```

```
  canActivate() {  
    return this.authService.isLoggedIn();  
  }
```

```
}
```

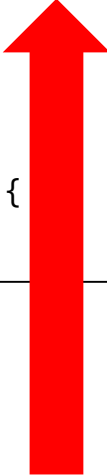
Class/Guard name

Auth Service

**Interface
implementation**

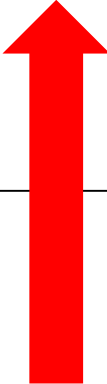
Register Guard class on module and routes

```
// app.module.ts
...
@NgModule({
  ...
  providers : [
    ...,
    AuthService,
    CanActivateViaAuthGuard
  ],
  ...
})
export class AppModule {
}
```



```
// app.routes.ts
...
import {CanActivateViaAuthGuard} from "./canActivateViaAuthGuard";

export const AppRoutes: Routes = [
  ...
  {
    path      : 'add',
    component : CityAddComponent,
    canActivate: [CanActivateViaAuthGuard]
  },
  ...
];
```




Deactivating routes

- Called when navigating *away* from a route
- Same approach as `CanActivate` route

```
// canDeactivateGuard.ts
import {Injectable} from '@angular/core';
import {CanDeactivate} from '@angular/router';
import {CanDeactivateComponent} from './canDeactivate.component';

@Injectable()
export class CanDeactivateGuard implements CanDeactivate<CanDeactivateComponent> {

  canDeactivate(target: CanDeactivateComponent) {
    // Can the user deactivate the route? Test for changes here!
    // For now, return Yes/Nope from the browser confirm dialog.
    if (target.hasChanges()) {
      return window.confirm('Do you really want to cancel? There might be unsaved changes');
    }
    return true;
  }
}
```



Add guard to routes

```
// app.routes.ts


...

import {CanDeactivateComponent} from "./canDeactivate.component";
import {CanDeactivateGuard} from "./canDeactivateGuard";

export const AppRoutes: Routes = [

  ...

  {
    path      : 'deactivate',
    component : CanDeactivateComponent,
    canDeactivate: [CanDeactivateGuard]
  },
  ...
];
```




Create DeactivateComponent

- Add implementation of `.hasChanges()`!

```
// ...
export class CanDeactivateComponent implements OnInit {
  // Properties voor de component/class
  myForm: FormGroup = new FormGroup({
    txtInput: new FormControl()
  });

  constructor(private route: Router) { }
  ngOnInit() {}

  moveAway() {
    this.route.navigate(['/home']);
  }
  hasChanges(){
    return this.myForm.dirty; // return state of the form
  }
}
```



Workshop

- Use the example `../150-router-guards`
- Check the example code and make yourself familiar with the code flow through the application

OR:

- Work with your own example from this morning and implement a `CanActivateGuard` class

