Angular : guard

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Angular

Guard ?

- Une guard permet de contrôler (autoriser, interdire) l'accès à une `route` ou le départ depuis une `route`
- Traditionnellement (jusqu'à **Angular 14**), les Guards sont des classes : un service décoré par `@Injectable`.
- Depuis **Angular 15**, il est possible d'utiliser des fonctions pour définir les Guards, ce que l'on appelle les **Functional Guards**.

Angular

Guard ?

- Une guard permet de contrôler (autoriser, interdire) l'accès à une `route` ou le départ depuis une `route`
- Traditionnellement (jusqu'à **Angular 14**), les Guards sont des classes : un service décoré par `@Injectable`.
- Depuis **Angular 15**, il est possible d'utiliser des fonctions pour définir les Guards, ce que l'on appelle les **Functional Guards**.

Depuis **Angular 16**, les **Functional Guards** sont fortement encouragés .

Angular

Les interfaces ou les **Functional Guards**

- `CanActivate` ou `CanActivateFn` : vérifie si un utilisateur peut visiter une route.
- `CanDeactivate` ou `CanDeactivateFn` : vérifie si un utilisateur peut quitter une route.
- `CanActivateChild` ou `CanActivateChildFn` : vérifie si un utilisateur peut visiter les routes enfants.
- `CanMatch` ou `CanMatchFn` : vérifie si une Route donnée (et le composant ou le module qu'elle charge) peut être utilisée pour correspondre à l'URL courante.
- `CanLoad` ou `CanLoadFn` [Dépréciée] : vérifie si un utilisateur peut aller sur une route d'un module défini avec un lazy loading

Angular

Exemple

- On veut que l'accès à la route `/adresse` soit seulement autorisé aux utilisateurs authentifiés
- On va créer une guard `auth` et l'associer à la route `adresse`

Angular

Pour créer une guard

```
ng generate guard guard-name
```

Angular

Pour créer une guard

```
ng generate guard guard-name
```

Ou le raccourci

```
ng g g guard-name
```

Angular

Pour notre exemple, on va créer une garde qui vérifie si un utilisateur est authentifié avant de charger certaines routes

```
ng g g guards/auth
```

Angular

Pour notre exemple, on va créer une garde qui vérifie si un utilisateur est authentifié avant de charger certaines routes

```
ng g g guards/auth
```

Dans le menu qui s'affiche

- Pointer sur `CanActivate`
- Puis cliquer une première fois sur `espace` et une deuxième sur `entrée`

Angular

On peut aussi préciser dans la commande l'interface à implémenter

```
ng g g guards/auth -implements CanActivate
```

Angular

On peut aussi préciser dans la commande l'interface à implémenter

```
ng g g guards/auth -implements CanActivate
```

Résultat

```
CREATE src/app/guards/auth.guard.spec.ts (346 bytes)
CREATE src/app/guards/auth.guard.ts (456 bytes)
```

Angular

Créons une interface `User`

```
ng g i interfaces/user
```

Angular

Créons une interface `User`

```
ng g i interfaces/user
```

Considérons le contenu suivant pour `user.ts`

```
export interface User {
    username?: string;
    password?: string;
}
```

Angular

Contenu du `auth.guard.ts`

```
import { CanActivateFn } from '@angular/router';

export const authGuard: CanActivateFn = (route, state) => {
  return true;
};
```

Angular

Contenu du `auth.guard.ts`

```
import { CanActivateFn } from '@angular/router';

export const authGuard: CanActivateFn = (route, state) => {
  return true;
};
```

Explication

- `route` de type `ActivatedRouteSnapshot` : contient des informations comme les paramètres envoyés pour la route demandée...
- `state` de type`RouterStateSnapshot` :contient des informations comme l'URL de la route demandée
- La fonction de type `CanActivateFn` ne fait aucun contrôle car elle retourne toujours `true`

Angular

Dans `app.routing.ts`**, associons** `authGuard` **à la route** `/adresse`

```
const routes: Routes = [
  {
    path: 'adresse',
    component: AdresseComponent,
    canActivate: [authGuard]
  },
  // les autres routes
];
```

Angular

Créons le composant que nous utiliserons pour l'authentification

```
ng g c components/auth
```

Angular

Créons le composant que nous utiliserons pour l'authentification

```
ng g c components/auth
```

Dans `app-routing.module.ts`**, associons la route** `/auth` **au composant** `AuthComponent`

```
const routes: Routes = [
  { path: 'auth', component: AuthComponent },

  // les autres routes
];
```

Angular

Contenu de `auth.component.html`

```
<h1>Page d'authentification</h1>
<form #monForm=ngForm (ngSubmit)=login()>
    <div>
        Nom d'utilisateur :
        <input type=text [(ngModel)]=user.username name=username
           required>
    </div>
    <div>
        Mot de passe :
        <input type=password [(ngModel)]=user.password name=password
           required>
        <button  type=submit [disabled]="monForm.invalid">
            Se connecter
        </button>
    </div>
    <div [hidden]='erreur'>Identifiants incorrects</div>
</form>
```

Angular

Contenu de `auth.component.ts`

```
import { Component, OnInit } from '@angular/core';
import { User } from 'src/app/interfaces/user';
import { AuthService } from 'src/app/services/auth.service';
import { Router } from '@angular/router';
@Component({
  selector: 'app-auth',
  templateUrl: './auth.component.html',
  styleUrls: ['./auth.component.css']
})
export class AuthComponent implements OnInit {
  user: User = {};
  erreur = true;
  constructor(private router: Router) { }
  ngOnInit() { }

  login() {
    if (this.login === 'wick' && this.password === 'john') {
      localStorage.setItem('isConnected', 'true');
      this.router.navigateByUrl('/cours/personne');
    } else {
      this.erreur = false;
    }
  }
}
```

Angular

Mettons à jour le contenu de `auth.guard.ts`

```
import { inject } from '@angular/core';
import { CanActivateFn, Router } from '@angular/router';

export const authGuard: CanActivateFn = (route, state) => {
  const router = inject(Router);
  if (Boolean(localStorage.getItem('isConnected'))) {
    return true;
  } else {
    return router.createUrlTree(['/auth']);
  }
};
```

Angular

Explication

- `createUrlTree` construit simplement un arbre d'**URL** (une représentation de la route) sans effectuer la navigation et retourne un objet `UrlTree`.
- Dans un guard, **Angular** intercepte le `UrlTree` et redirige automatiquement.
- Les guards ne doivent pas provoquer eux-mêmes la navigation, mais laisser le routeur décider.

Angular

Pour tester

- Essayez d'accéder à la route `/adresse` sans authentification
- Authentifiez-vous avec les identifiants `wick` et `john` et réessayez

Angular

Exécutons la commande suivante

```
ng g g guards/leave -implements CanDeactivate
```

Angular

Exécutons la commande suivante

```
ng g g guards/leave -implements CanDeactivate
```

Le résultat

```
CREATE src/app/guards/leave.guard.spec.ts (352 bytes)
CREATE src/app/guards/leave.guard.ts (472 bytes)
```

Angular

Code généré

```
import { CanDeactivateFn } from '@angular/router';

export const leaveGuard: CanDeactivateFn<unknown> = (component, currentRoute,
   currentState, nextState) => {
  return true;
};
```

Angular

Code généré

```
import { CanDeactivateFn } from '@angular/router';

export const leaveGuard: CanDeactivateFn<unknown> = (component, currentRoute,
   currentState, nextState) => {
  return true;
};
```


Ensuite

Remplacez `unknown` par `FormulaireComponent`

Angular

Nouveau code

```
import { CanDeactivateFn } from '@angular/router';
import { FormulaireComponent } from '../components/formulaire/formulaire';

export const leaveGuard: CanDeactivateFn<FormulaireComponent> = (component,
   currentRoute, currentState, nextState) => {
  return true;
};
```

Angular

Nouveau code

```
import { CanDeactivateFn } from '@angular/router';
import { FormulaireComponent } from '../components/formulaire/formulaire';

export const leaveGuard: CanDeactivateFn<FormulaireComponent> = (component,
   currentRoute, currentState, nextState) => {
  return true;
};
```


Objectif

- On ajoute le lien suivant dans `formulaire.html` :

```
<a routerLink='/personne'>Personne</a>
```

- Si l'utilisateur remplit les deux champs `nom` et `prenom` dans `formulaire.html` et clique sur le lien pour aller sur le composant `personne`, on lui demande une confirmation.

Angular

Modifions la garde

```
import { CanDeactivateFn } from '@angular/router';
import { FormulaireComponent } from '../components/formulaire/
   formulaire';

export const leaveGuard: CanDeactivateFn<FormulaireComponent> = (
   component, currentRoute, currentState, nextState) => {

  return component.personne.nom === undefined ||
    component.personne.prenom === undefined ||
    component.personne.nom?.length === 0 ||
    component.personne.prenom?.length === 0 ||
    confirm('voulez-vous vraiement quitter ?');
}
```

Angular

Associons `leaveGuard` à la route `/formulaire` dans `cours-routing.module.ts`

```
const routes: Routes = [
  {
    path: 'adresse',
    component: AdresseComponent,
    canActivate: [AuthGuard]
  },
  {
    path: 'formulaire',
    component: FormulaireComponent,
    canDeactivate: [leaveGuard]
  },
  // le reste des routes
];
```

Angular

Il est possible d'associer plusieurs guards à une route

```
const routes: Routes = [
  {
    path: 'adresse',
    component: AdresseComponent,
    canActivate: [AuthGuard]
  },
  {
    path: 'formulaire',
    component: FormulaireComponent,
    canActivate: [authGuard],
    canDeactivate: [leaveGuard],
  },
  // le reste des routes
];
```