

# Cours Angular : Guards et Interceptors

## 1. Introduction

Dans Angular, **les guards** protègent les routes et **les interceptors** permettent d'intercepter et de modifier les requêtes HTTP. Ils sont essentiels pour la **sécurité**, la **gestion des erreurs** et le **contrôle de flux**.

Objectifs : - Comprendre les différents types de guards - Créer et utiliser des guards pour sécuriser les routes - Comprendre le fonctionnement des interceptors - Créer des interceptors pour gérer tokens et erreurs - Appliquer les bonnes pratiques professionnelles

---

## 2. Les Guards

### 2.1 Qu'est-ce qu'un guard ?

Un **guard** est une classe Angular qui décide si une route peut être activée, désactivée, ou chargée.

### 2.2 Types de guards

- **CanActivate** : protège l'accès à une route
  - **CanActivateChild** : protège les routes enfants
  - **CanDeactivate** : vérifie avant de quitter un composant
  - **CanLoad** : empêche le chargement d'un module lazy loaded
- 

### 2.3 Exemple CanActivate

```
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';

@Injectable({ providedIn: 'root' })
export class AuthGuard implements CanActivate {

  constructor(private router: Router) {}

  canActivate(): boolean {
    const isLoggedIn = false; // logique réelle
    if (!isLoggedIn) {
      this.router.navigate(['/login']);
      return false;
    }
    return true;
  }
}
```

## 2.4 Utilisation dans le routing

```
const routes = [
  { path: 'dashboard', component: DashboardComponent, canActivate:
    [AuthGuard] }
];
```

## 2.5 Exemple CanDeactivate

```
export interface CanComponentDeactivate {
  canDeactivate: () => boolean;
}

@Injectable({ providedIn: 'root' })
export class UnsavedChangesGuard implements
  CanDeactivate<CanComponentDeactivate> {
  canDeactivate(component: CanComponentDeactivate): boolean {
    return component.canDeactivate ? component.canDeactivate() : true;
  }
}
```

# 3. Les Interceptors

## 3.1 Qu'est-ce qu'un interceptor ?

Un **interceptor HTTP** est une classe qui intercepte toutes les requêtes et réponses HTTP.

## 3.2 Création d'un interceptor

```
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent } from
  '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler):
    Observable<HttpEvent<any>> {
    const cloned = req.clone({ headers: req.headers.set('Authorization',
      'Bearer token') });
    return next.handle(cloned);
  }
}
```

### 3.3 Déclaration dans AppModule

```
providers: [  
  { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true }  
]
```

### 3.4 Gestion globale des erreurs

```
import { catchError } from 'rxjs/operators';  
import { throwError } from 'rxjs';  
  
intercept(req, next) {  
  return next.handle(req).pipe(  
    catchError(err => {  
      console.error('Erreur HTTP', err);  
      return throwError(err);  
    })  
  );  
}
```

## 4. Bonnes pratiques

- Utiliser **guards** pour toutes les routes sensibles
- Préférer **CanActivate** / **CanLoad** pour les modules lazy loaded
- Intercepteurs pour ajouter les tokens, logs et gestion globale des erreurs
- Centraliser la logique dans des services réutilisables
- Toujours utiliser `multi: true` pour les interceptors

## 5. Cas pratique complet

### 5.1 AuthGuard + HTTP Interceptor

**AuthGuard** : protège `/dashboard` **Interceptor** : ajoute le token JWT à toutes les requêtes

```
// AuthGuard  
canActivate(): boolean {  
  if (!this.authService.isLoggedIn()) {  
    this.router.navigate(['/login']);  
    return false;  
  }  
  return true;  
}
```

```
// Interceptor
intercept(req, next) {
  const token = this.authService.getToken();
  const cloned = req.clone({ headers: req.headers.set('Authorization',
`Bearer ${token}`) });
  return next.handle(cloned);
}
```

---

## 6. Résumé

- **Guards** = sécurité et contrôle de navigation
- **Interceptors** = manipulation globale des requêtes HTTP
- Angular permet de centraliser la sécurité et la gestion des données de manière réactive et maintenable

---

👉 Prochaine étape recommandée : **Resolvers, Guards avancés et Interceptors combinés avec state management**