Angular : JWT et HttpInterceptor

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

Objectif de ce chapitre

- Considérons les fichiers suivant qui permettent de communiquer avec une ressource REST (le code est donné dans les slides suivantes)
  - `personne.ts` (l'interface)
  - `personne.ts` (le composant)
  - `personne.html`
  - `personne.css`
  - `personne.ts` (le service)
- Les ressources REST nécessite une authentification (avec JWT) et une autorisation

Angular

Objectif de ce chapitre

- Considérons les fichiers suivant qui permettent de communiquer avec une ressource REST (le code est donné dans les slides suivantes)
  - `personne.ts` (l'interface)
  - `personne.ts` (le composant)
  - `personne.html`
  - `personne.css`
  - `personne.ts` (le service)
- Les ressources REST nécessite une authentification (avec JWT) et une autorisation

L'URL dans le service est à modifier.

Angular

Contenu de `personne.ts`

```
export interface Personne {
    id?: number;
    nom?: string;
    prenom?: string;
}
```

Angular

Contenu de `personne.service.ts`

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Personne } from '../interfaces/personne';
import { environment } from '../../environments/environment.development';

@Injectable({
  providedIn: 'root'
})
export class PersonneService {

  private url: string

  constructor(private http: HttpClient) {
    this.url = environment.BACKEND_URL + '/personnes'
  }

  getAll() {
    return this.http.get<Array<Personne>>(this.url);
  }
  addPerson(p: Personne) {
    return this.http.post(this.url, p);
  }
}
```

Angular

Contenu de `personne.html`

```
<h3> Pour ajouter une personne</h3>
<form #monForm=ngForm (ngSubmit)=ajouterPersonne()>
  <div>
    Nom : <input type=text name=nom [(ngModel)]=personne.nom required #nom="ngModel">
  </div>
  <div [hidden]="nom.valid || nom.pristine">
    Le nom est obligatoire
  </div>
  <div>
    Prénom : <input type=text name=prenom [(ngModel)]=personne.prenom required #prenom="ngModel
       ">
  </div>
    <div [hidden]="prenom.valid || prenom.pristine">
      Le prénom est obligatoire
    </div>
  <div>
    <button [disabled]=!monForm.valid>
      ajouter
    </button>
  </div>
</form>

<h3> Liste de personnes </h3>
<ul>
  <li *ngFor="let elt of personnes">
    {{ elt.prenom }}  {{ elt.nom }}
  </li>
</ul>
```

Angular

Contenu de `personne.css`

```
.ng-invalid:not(form){
    border-left: 5px  solid red;
}
.ng-valid:not(form){
    border-left: 5px  solid green;
}
```

Contenu de `personne..ts`

```
// les imports
@Component({
  selector: 'app-personne',
  templateUrl: './personne.html',
  styleUrls: ['./personne.css']
})
export class PersonneComponent implements OnInit {
  personne: Personne = {};
  personnes: Array <Personne> = [];
  constructor(private personneService: PersonneService) {  }
  ngOnInit() {
    this.personneService.getAll().subscribe(res => {
      this.personnes = res;
    });
  }
  ajouterPersonne() {
    this.personneService.addPerson(this.personne).subscribe(res => {
      this.personne = res
    });
  }
}
```

Angular

Exercice

Pour chaque personne, ajoutez

- un bouton `supprimer` qui permet de supprimer la personne associée de la base de données
- un lien `modifier` qui permet d'afficher les données relatives à la personne associée dans un formulaire du composant `edit-personne` afin de permettre la modification de ces données.

Angular

Considérons le contenu suivant pour `User`

```
export interface User {
    username?: string
    password?: string
    grantType: string
    refreshToken?: string
}
```

Angular

Considérons le contenu suivant pour `User`

```
export interface User {
    username?: string
    password?: string
    grantType: string
    refreshToken?: string
}
```


Et le contenu suivant pour `Token`

```
export interface Token {
    accessToken: string
    refreshToken: string
}
```

Angular

Commençons par créer le service `auth`

```
ng g s services/auth
```

Commençons par modifier le contenu de `auth.service.ts` en ajoutons une méthode `login` qui utilise la méthode HTTP `POST` pour interroger le backend

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { environment } from '../../environments/environment.development
  ';
import { User } from '../models/user';
import { Observable } from 'rxjs';
import { Token } from '../models/token';

@Injectable({
  providedIn: 'root'
})
export class AuthService {

  private url: string
  constructor(private http: HttpClient) {
    this.url = environment.BACKEND_URL + '/authenticate'
  }

  findByUsernameAndPassword(user: User): Observable<Token> {
    return this.http.post<Token>(this.url, user)
  }
}
```

Préparons le formulaire d'authentification (`auth.html`)

```
<h2 class="text-primary my-5">Page de connexion</h2>
<form (ngSubmit)="seConnecter()" #form="ngForm">
    <div>
        <label for="username">Nom d'utilisateur</label>
        <input type="text" name="username" [(ngModel)]="user.username"
          required minlength="3">
    </div>
    <div>
        <label for="password">Mot de passe</label>
        <input type="password" name="password" [(ngModel)]="user.
          password" minlength="3">
    </div>
    <div>
        <button [disabled]="form.invalid">
            Se connecter
        </button>
    </div>
</form>
@if (erreur()) {
<div class="alert alert-danger my-5" role="alert">
    {{ erreur() }}
</div>
}
```

Dans `auth.ts`, on utilise la méthode `findByUsernameAndPassword` de `auth.service.ts`. si on reçoit un token l'utilisateur existe dans la base (on ajoute donc le token au `localstorage`)

```
import { Component, signal } from '@angular/core';
import { User } from '../../models/user';
import { FormsModule } from '@angular/forms';
import { Router } from '@angular/router';
import { AuthService } from '../../services/auth';

@Component({
  selector: 'app-auth',
  imports: [FormsModule],
  templateUrl: './auth.html',
  styleUrl: './auth.css',
})
export class AuthComponent {
  erreur = signal<string | null>(null)
  user: User = { username: '', password: '', grantType: 'PASSWORD' }
  constructor(
    private router: Router,
    private authService: AuthService,
  ) { }
  seConnecter() {
    this.authService.findByUsernameAndPassword(this.user).subscribe({
      next: (res) => {
        localStorage.setItem('tokens', JSON.stringify(res))
        localStorage.setItem('user', JSON.stringify(this.user))
        this.router.navigateByUrl('/personne')
      },
      error: (err) => this.erreur.set("Identifiants incorrects")
    })
  }
}
```

Angular

Modifions maintenant le contenu de `personne.service.ts` pour utiliser le token

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Personne } from '../interfaces/personne';
import { map } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class PersonneService {

  private url: string
  private headers: HttpHeaders;

  constructor(private http: HttpClient) {
    this.url = environment.BACKEND_URL + '/personnes'
    const tokensString: string = localStorage.getItem('tokens') ?? ''
    let tokens: Token = JSON.parse(tokensString)
    this.headers = new HttpHeaders().set('Authorization', `Bearer ${tokens.accessToken}`);
  }

  getAll() {
    return this.http.get(this.url, { headers: this.headers });
  }

  addPerson(p: Personne) {
    return this.http.post(this.url, p, { headers: this.headers });
  }
}
```

Angular

Deux problèmes avec la solution précédente

- Code répétitif (le **HttpHeaders** dans toutes les requêtes **HTTP**)
- Sans authentification, on peut accéder au composant `personne`

Angular

Deux problèmes avec la solution précédente

- Code répétitif (le **HttpHeaders** dans toutes les requêtes **HTTP**)
- Sans authentification, on peut accéder au composant `personne`

Solutions

- pour le premier problème : **HttpInterceptor**
- pour le deuxième : **Guard**

Angular

jwt-decode

- Package **Node.js**
- Écrit en **TypeScript**
- Permettant de décoder un jeton **JWT** et retourner un objet **JavaScript**
- `https://www.npmjs.com/package/jwt-decode`

Angular

Pour l'installer

```
npm install jwt-decode
```

Angular

Exemple

```
var decoded = jwtDecode(token);

console.log(decoded);
```

Angular

Commençons par créer un intercepteur `auth` dans `services`

```
ng g interceptor services/auth
```

Angular

Code généré de `auth.interceptor.ts`

```
export const authInterceptor: HttpInterceptorFn = (req, next) => {
  return next(req);
};
```

Angular

Déclarons l'intercepteur dans `app.config.ts`

```
import { ApplicationConfig, provideBrowserGlobalErrorListeners, provideZoneChangeDetection }
   from '@angular/core';
import { provideRouter, withComponentInputBinding } from '@angular/router';

import { routes } from './app.routes';
import { provideHttpClient, withInterceptors, withInterceptorsFromDi } from '@angular/common/
   http';
import { authInterceptor } from './services/auth-interceptor';

export const appConfig: ApplicationConfig = {
  providers: [
    provideBrowserGlobalErrorListeners(),
    provideZoneChangeDetection({ eventCoalescing: true }),
    provideRouter(routes, withComponentInputBinding()),
    provideHttpClient(withInterceptorsFromDi(), withInterceptors([authInterceptor]))
  ]
};
```

Préparons l'intercepteur pour modifier les requêtes sortantes en ajoutant le token sauf pour celle qui demande le jeton (à l'authentification)

```
import { HttpInterceptorFn } from '@angular/common/http';

const AUTH_URL = 'http://localhost:8080/authenticate';

export const authInterceptor: HttpInterceptorFn = (req, next) => {
  if (req.url === AUTH_URL) {
    return next(req);
  }

  const tokensString: string = localStorage.getItem('tokens') ?? ''

  if (!tokensString) {
    return next(req);
  }
  let tokens: Token = JSON.parse(tokensString)

  const cloned = req.clone({
    setHeaders: { 'Authorization': `Bearer ${tokens.accessToken}` }
  });
  return next(cloned);
};
```

Angular

Modifions maintenant `personne.service.ts` pour ne plus ajouter le token

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Personne } from '../interfaces/personne';
import { map } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class PersonneService {

  private url: string

  constructor(private http: HttpClient) {
    this.url = environment.BACKEND_URL + '/personnes'
  }

  getAll() {
    return this.http.get(this.url);
  }

  addPerson(p: Personne) {
    return this.http.post(this.url, p);
  }
}
```

Angular

Commençons par modifier notre service d'authentification

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { environment } from '../../environments/environment.development';
import { User } from '../models/user';
import { Observable } from 'rxjs';
import { jwtDecode } from 'jwt-decode';
import { Token } from '../models/token';

@Injectable({
  providedIn: 'root',
})
export class AuthService {
  url: string
  constructor(private http: HttpClient) {
    this.url = environment.BACKEND_URL + '/authenticate'
  }

  getTokens(user: User): Observable<Token> {
    return this.http.post<Token>(this.url, user)
  }

  isExpired(token: string): Boolean {
    const decoded = jwtDecode(token);
    return Math.round(Date.now() / 1000) > (decoded.exp ?? 1);
  }
}
```

L'intercepteur utilisera le `refreshToken` pour obtenir un nouveau `accessToken` si ce dernier expire

```
const AUTH_URL = `${environment.BACKEND_URL}/authenticate`

export const authInterceptor: HttpInterceptorFn = (req, next) => {
  const authService = inject(AuthService)
  if (req.url == AUTH_URL) {
    return next(req);
  }
  const tokensString: string = localStorage.getItem('tokens') ?? ''
  if (!tokensString) {
    return next(req);
  }
  let tokens: Token = JSON.parse(tokensString)
  if (!authService.isExpired(tokens.accessToken)) {
    const cloned = req.clone({
      setHeaders: { 'Authorization': `Bearer ${tokens.accessToken}` }
    })
    return next(cloned)
  }
  return authService.getTokens({
    grantType: "REFRESH_TOKEN",
    refreshToken: tokens.refreshToken
  })
    .pipe(
      switchMap(res => {
        localStorage.setItem('tokens', JSON.stringify(res))
        const cloned = req.clone({
          setHeaders: { 'Authorization': `Bearer ${tokens.accessToken}` }
        })
        return next(cloned)
      })
    )
};
```

Angular

Attention le code suivant ne fonctionnera pas correctement

```
const AUTH_URL = `${environment.BACKEND_URL}/authenticate`

export const authInterceptor: HttpInterceptorFn = (req, next) => {
  const authService = inject(AuthService)
  if (req.url == AUTH_URL) {
    return next(req);
  }
  const tokensString: string = localStorage.getItem('tokens') ?? ''
  if (!tokensString) {
    return next(req);
  }
  let tokens: Token = JSON.parse(tokensString)
  if (authService.isExpired(tokens.accessToken)) {
    authService.getTokensUsingRefreshToken({
      grantType: "REFRESH_TOKEN",
      refreshToken: tokens.refreshToken
    }).subscribe(res => {
      localStorage.setItem('tokens', JSON.stringify(res))
    })
  }
  const cloned = req.clone({
    setHeaders: { 'Authorization': `Bearer ${tokens.accessToken}` }
  })
  return next(cloned)
};
```

Angular

Explication

- `subscribe` casses la chaîne **RxJS** et on ne peut plus attendre le résultat dans l'interceptor.
- Un interceptor doit toujours retourner un Observable, jamais faire un subscribe lui-même.
- Il faut chaîner l'appel de refresh avec `pipe` + `switchMap` (ou autre opérateur) et ne retourner `next(cloned)` qu'après avoir reçu les nouveaux tokens.

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
```

Le résultat

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

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

Modifions le contenu de `auth.guard.ts` pour rediriger vers la page d'authentification s'il n'y a pas de jeton en session

```
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

Associons `AuthGuard` **aux différentes routes dans** `app.routes.ts`

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { PersonneComponent } from './composants/personne/personne.component';
import { EditPersonneComponent } from './composants/edit-personne/edit-personne.component';
import { AuthComponent } from './composants/auth/auth.component';
import { authGuard } from './services/auth.guard';

const routes: Routes = [
  { path: 'auth', component: AuthComponent },
  { path: 'editPersonne/:id', component: EditPersonneComponent, canActivate: [authGuard] },
  { path: 'personne', component: PersonneComponent, canActivate: [authGuard] },
];
```

Angular

Pour éviter les répétitions dans `app-routing.module.ts`

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { PersonneComponent } from './composants/personne/personne.component';
import { EditPersonneComponent } from './composants/edit-personne/edit-personne.component';
import { AuthComponent } from './composants/auth/auth.component';
import { AuthGuard } from './services/auth.guard';

const routes: Routes = [

  { path: 'auth', component: AuthComponent },
  {
    path: '', canActivate: [AuthGuard], children: [
      { path: 'editPersonne/:id', component: EditPersonneComponent },
      { path: 'personne', component: PersonneComponent }
    ]}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Angular

Pour se déconnecter, il suffit de

- supprimer les tokens du `localStorage`
- rediriger l'utilisateur vers la page d'authentification

Angular

Exercice

Dans le composant `header`, on veut afficher

- un bouton `déconnexion` si un jeton est stocké dans le `WebStorage`
- un lien `connexion` qui redirige vers la page d'authentification sinon.

Angular

Commençons par créer un service `login-logout`

```
import { Injectable } from '@angular/core';
import { Subject } from 'rxjs';

@Injectable({
  providedIn: 'root',
})
export class LoginLogoutService {

  private subject = new Subject<boolean>()

  isConnected(state: boolean) {
    this.subject.next(state)
  }

  getSubject() {
    return this.subject
  }
}
```

Angular

Utilisons ce service au moment de la connexion

```
@Component({
  selector: 'app-auth',
  imports: [FormsModule],
  templateUrl: './auth.html',
  styleUrl: './auth.css',
})
export class AuthComponent {

  erreur = signal<string | null>(null)
  user: User = { username: '', password: '', grantType: 'PASSWORD' }

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

  seConnecter() {
    this.authService.findByUsernameAndPassword(this.user).subscribe({
      next: (res) => {
        localStorage.setItem('tokens', JSON.stringify(res))
        localStorage.setItem('user', JSON.stringify(this.user))
        this.logService.isConnected(true)
        this.router.navigateByUrl('/personne')
      },
      error: (err) => this.erreur.set("Identifiants incorrects")
    })
  }
}
```

Angular

Utilisons ce service également dans le composant `header`

```
@Component({
  selector: 'app-header',
  imports: [],
  templateUrl: './header.html',
  styleUrl: './header.css',
})
export class HeaderComponent {

  title = 'angular-standalone';
  isConnected = signal(!!localStorage.getItem('tokens'))

  constructor(
    private router: Router,
    private logService: LoginLogoutService
  ) {
    this.logService.getSubject().subscribe(v => this.isConnected.set(v))
  }

  login() {
    this.router.navigateByUrl('/auth')
  }

  logout() {
    localStorage.removeItem('tokens')
    localStorage.removeItem('user')
    this.logService.isConnected(false)
    this.router.navigateByUrl('/')
  }
}
```

Angular

Et enfin `header.html`

```
<header>
    <h1 class="text-primary my-5">
        Welcome to {{ title }}
        <i style="color: tomato;" class="fa-brands fa-angular"></i>!
        <button class="btn btn-link" (click)="login()" [hidden]="isConnected()">
            Login
        </button>
        <button class="btn btn-link" (click)="logout()" [hidden]="!isConnected()">
            Logout
        </button>
    </h1>
</header>
```