Angular Identity Management

La sécurité

L'authentification selon Angular

Afin de mettre en place l'authentification, nous allons ajouter la fonctionnalité "guard" d'Angular.

La classe "guard" permet de contrôler l'accès aux routes grâce à des méthodes :

- Si la méthode retourne "true" alors la navigation continue
- Si la méthode retourne "false" alors la navigation s'arrête et l'utilisateur ne peut aller plus loin sauf si la méthode retourne une nouvelle navigation
- Si la méthode retourne un arbre de navigation "UrlTree" alors une nouvelle navigation commence.

Les méthodes de la classe Guard peuvent être synchrone mais il est fortement conseillé de les faire asynchrone : retourne Observable<boolean>.

Le routeur prend en charge plusieurs interfaces de "guard" :

- CanActivate pour naviguer vers une route.
- CanActivateChild pour naviguer vers une route enfant.
- CanDeactivate pour quitter la navigation et aller vers une autre .
- Resolve pour récupérer des données de route avant l'activation de la route.
- CanLoad pour charger un module asynchrone afin de continuer la navigation.

Nous allons mettre en place la sécurité :

1. Création du service AuthenticationService : service simulant une gestion d'authentification depuis une API JWT (Json Web Token)
2. Création du composant AuthGuard : classe Guard personnalisée
3. Création de la page de login
4. Ajout du composant AuthGuard au routage
5. Ajout de la méthode "logout" à la "navbar"
6. Modification des composants HTML pour mettre à jour le layout

(Voir article Angular Authentication: Using Route Guards - Ryan Chenkie)

Création du service AuthenticationService

Créez un répertoire "security" dans le répertoire "src/app".
Dans le répertoire "security", exécutez la commande suivante pour générer le service : ng generate service Authentication
Ce service va vérifier la connexion (méthode isLoggedIn), gérer le token, exécuter le login et le logout.

Le code sera le suivant :

```typescript
interface AuthenticationResponse {
  status: boolean;
  token: string;
  message: string;
}

@Injectable({
  providedIn: 'root'
})
export class AuthenticationService {
  redirectUrl = '/';

  constructor() { }

  static isLoggedIn() {
    // Checks if there is a saved token and it's still valid
    const token = AuthenticationService.getToken();
    console.log('token=' + token);
    return !!token && !AuthenticationService.isTokenExpired(token);
  }

  static isTokenExpired(token: string) {
    /*
    // Le vrai code
    try {
      const decoded = jwt_decode(token);
      return decoded.exp < Date.now() / 1000;
    } catch (err) {
      return false;
    }*/

    // La simulation
    return false;
  }

  static setToken(idToken: string) {
    // Saves user token to sessionStorage
    sessionStorage.setItem('id_token', idToken);
  }

  static getToken() {
    // Retrieves the user token from sessionStorage
    return sessionStorage.getItem( key: 'id_token');
  }
```

```
  static logout() {
    // Clear user token and profile data from sessionStorage
    sessionStorage.removeItem( key: 'id_token');
  }

  loginWithRole(username, password, role): Observable<AuthenticationResponse> {
  /*
  // Le vrai code
  const url = `${this.authenticationUrl}/login`;
  const httpOptions = {
    headers: new HttpHeaders({
      'Content-Type': 'application/json'
    })
  };

  return this.httpClient.request<AuthenticationResponse>('POST', url, {
    body: {
      username,
      password,
      role
    },
    headers: httpOptions.headers
  }).pipe(
    tap((data: AuthenticationResponse)
      => AuthenticationService.setToken(data.token))
     // Setting the token in sessionStorage)
  );
  */

    // La simulation
    const response: AuthenticationResponse
        = {status: true, message: 'HTTP 200', token: 'atoken'};
    AuthenticationService.setToken('token');
    return of(response);
  }
}
```

Génération du composant AuthGuard

Dans le répertoire “security’, exécutez la commande suivante pour générer la classe AuthGuard (prenez uniquement l’option “CanActivate”) : `ng generate guard auth`

La classe va vérifier si la personne est logguée et le cas échéant va rediriger l'utilisateur sur la page de

login. Le code sera le suivant :

```
@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(
    private authenticationService: AuthenticationService,
    private router: Router) {}

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> |
      Promise<boolean | UrlTree> | boolean | UrlTree {
    return this.checkLogin(state.url);
  }

  private checkLogin(url: string): boolean {
    if (AuthenticationService.isLoggedIn()) {
      return true;
    }

    // On conserve le lien demandé
    this.authenticationService.redirectUrl = url;

    // Redirection vers la page de login
    this.router.navigate( commands: ['/login']);
    return false;
  }
}
```

Création de la page de login

Toujours dans le répertoire “security’, exécutez la commande suivante pour générer le composant : `ng generate component login --module=app`

Le HTML

Le HTML est classique :

```
<div class="signin-content">
  <mat-card>
    <mat-card-title>Gestion des utilisateurs</mat-card-title>
    <mat-card-content>
      <form [formGroup]="form" (ngSubmit)="onSubmit()">
        <mat-form-field class="full-width-input">
          <input matInput placeholder="User" formControlName="userName" required>
          <mat-error *ngIf="isFieldInvalid( field: 'userName')">
            Login incorrect
          </mat-error>
        </mat-form-field>
        <mat-form-field class="full-width-input">
          <input matInput type="password" placeholder="Password"
                 formControlName="password" required>
          <mat-error *ngIf="isFieldInvalid( field: 'password')">
            Mot de passe incorrect
          </mat-error>
        </mat-form-field>

        <button mat-raised-button color="primary"
                [disabled]="processRunning || !form.valid">
          <mat-icon *ngIf="processRunning">
            <mat-spinner diameter="16"></mat-spinner>
          </mat-icon>
          Connexion</button>
      </form>
    </mat-card-content>
  </mat-card>
</div>
```

Il faut ajouter le module MatCardModule au module Material :

```
@NgModule({
 exports: [
    ...
  MatCardModule,
 ],
  ...
```

```
})
export class AppMaterialModule { }
```

Le CSS

Il est minimal:

```
mat-card {
 max-width: 400px;
 margin: 2em auto;
 text-align: center;
}
.signin-content {
 padding: 60px 1rem;
}
.full-width-input {
 width: 100%;
}
```

Le composant

Le composant login va vérifier les entrées utilisateurs et appeler le service “AuthenticationService” pour logguer l'utilisateur. le code sera le suivant :

```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})
export class LoginComponent implements OnInit {
  form: FormGroup;
  processRunning = false;
  private formSubmitAttempt: boolean;

  constructor(
    private fb: FormBuilder,
    private authenticationService: AuthenticationService,
    public router: Router,
    private snackBar: MatSnackBar
  ) {}

  ngOnInit() {
    this.form = this.fb.group( controlsConfig: {
      userName: ['', Validators.required],
      password: ['', Validators.required]
    });
  }
```

```
  isFieldInvalid(field: string) {
    return (
      (!this.form.get(field).valid && this.form.get(field).touched) ||
      (this.form.get(field).untouched && this.formSubmitAttempt)
    );
  }

  onSubmit() {
    if (this.form.valid) {
      this.processRunning = true;
      this.authenticationService.loginWithRole(
        this.form.get('userName').value,
        this.form.get('password').value,
        role: 'ROLE_SUPER_ADMIN'
      ).subscribe( next: () => {
          if (AuthenticationService.isLoggedIn()) {
            this.processRunning = false;
            this.router.navigate( commands: [this.authenticationService.redirectUrl]);
          } else {
            throw new Error();
          }
        },
        error: (error: HttpErrorResponse) => {
          this.processRunning = false;
          this.snackBar.open( message: 'Login ou mot de passe invalide !',  action: 'X');
        });
    }
    this.formSubmitAttempt = true;
  }
}
```

Ajout du composant AuthGuard au routage

Il nous faut modifier les routes dans app-routing.module.ts et ldap-management-routing.module.ts.

Le module principal

Modifiez les routes dans le module app-routing.module.ts ainsi pour avoir accès à la page login :

```
const routes: Routes = [
 { path: 'login', component: LoginComponent },
 { path: '', redirectTo: 'users/list', pathMatch: 'full' },
 { path: '**', component: PageNotFoundComponent }
];
```

Le module Ldap Management

Nous avons créé ce module pour “ranger” nos classes mais aussi pour organiser les routes afin de mettre en place la sécurité.
Modifiez le fichier ldap-management-routing.module.ts ainsi pour mettre en place le composant AuthGuard :

```
const adminRoutes: Routes = [
 {
   path: 'users',
   component: LdapComponent,
   canActivate: [AuthGuard],
   children: [
     { path: 'list', component: LdapListComponent },
     { path: 'add', component: LdapAddComponent },
     { path: ':id', component: LdapEditComponent },
     { path: '', redirectTo: '/users/list', pathMatch: 'full' },
   ]
 }
];

@NgModule({
 imports: [RouterModule.forChild(adminRoutes)],
 exports: [RouterModule]
})
export class LdapManagementRoutingModule { }
```

Ainsi toutes les routes enfants de “user” sont soumises à la sécurité de AuthGuard.

Ajout de la méthode “logout” à la “navbar”

Nous allons pour finir, câbler le bouton “logout” du composant “navbar”.
Modifiez le code du constructeur et de la méthode “logout” de navbar.component.ts ainsi :

```
constructor(
 private breakpointObserver: BreakpointObserver,
 private authenticationService: AuthenticationService,
 private router: Router
) {}

logout() {
 AuthenticationService.logout();
 this.router.navigate(['/login']);
```

```
}
```

Modification des composants HTML pour mettre à jour le layout

Si vous relancez l'application, la page de connexion est à “l’intérieur” du cadre.
Il va falloir modifier toutes les pages HTML pour les inclure dans le composant “navbar”.

Composant NavBar

Mais dans un premier temps, nous allons modifier “navbar.component.html”, pour ne pas utiliser de template via le routage mais plutôt via un composant enfant.
Trouvez le code suivant :

```
<mat-sidenav-content>
 <!-- Content Here -->
 <div class="main_content">
   <router-outlet></router-outlet>
 </div>
</mat-sidenav-content>
```

Et changez le avec le code ci-après :

```
<mat-sidenav-content>
 <!-- Content Here -->
 <div class="main_content">
   <ng-content></ng-content>
 </div>
</mat-sidenav-content>
```

Composant principal

Le code HTML (app.component.html) doit être le suivant :

```
<router-outlet></router-outlet>
```

Composants LdapDetail et LdapList

Le code HTML de chaque composant doit être entouré de la balise “<app-navbar>” :

```
<app-navbar>
  <!-- Remettre le code ici -->
</app-navbar>
```

Liens

https://angular.io/start

https://blog.angular.io/

https://blog.angular-university.io/

https://guide-angular.wishtack.io/

https://openclassrooms.com/fr/courses/4668271-developpez-des-applications-web-avec-angular

https://www.typescriptlang.org/docs/home.html

https://blog.soat.fr/

Parent Child Two way binding
https://medium.com/@preethi.s/angular-custom-two-way-data-binding-3e618309d6c7

Organisation par module
Mise en place de la sécurité
https://angular.io/guide/route

Login:
https://loiane.com/2017/08/angular-hide-navbar-login-page/

Dialog:
https://blog.angular-university.io/angular-material-dialog/

Angular Material
https://material.angular.io/

https://medium.com/@ismapro/first-steps-with-angular-7-with-angular-cli-and-angular-material-d69f55d8ac51

https://www.positronx.io/create-angular-material-8-custom-theme/

https://akveo.github.io/ngx-admin/

https://auth0.com/blog/creating-beautiful-apps-with-angular-material/