Angular Identity Management

Le service Users (avec Web API)

La classe `HttpClient` (du module `HttpClientModule`) permet de faire des appels à des services API.

- Les méthodes `get`, `delete`, `post`, `put`, … retourne un type `Observable`.
- Ces méthodes prennent plusieurs paramètres et par défaut les paramètres `observe` et `responseType` valent respectivement `body` et `json`. Ce qui permet d'obtenir un objet JavaScript depuis la réponse JSON contenue dans le body.
- Ces méthodes sont "typables", exemple : `client.get<UserLdap>(url).subscribe( … );`
- L'obtention des résultats se fait avec la méthode subscribe.

Simulation de données fictives améliorée

Afin de simuler un serveur API nous allons utiliser le module In-memory Web API. Ce module va permettre de faire des requêtes HTTP comme si nous le faisions sur une vraie API.

Installation du module

Il faut d'abord installer le package pour Angular : `yarn add angular-in-memory-web-api --save`
Puis il faut importer le module dans l'application, nous allons créer par la suite un service InMemoryUsersService pour injecter les données. Nous utiliserons le service UsersService juste après.

Modifiez le fichier ldap-management.module.ts

```
@NgModule({
 declarations: [
   ...
 ],
 imports: [
   ...
   HttpClientModule,
   // The HttpClientInMemoryWebApiModule module intercepts HTTP requests
   // and returns simulated server responses.
   // Remove it when a real server is ready to receive requests.
   HttpClientInMemoryWebApiModule.forRoot(
     InMemoryUsersService, {dataEncapsulation: false}
   )
 ]
})
export class LdapManagementModule { }
```

Création du service InMemoryUsersService

Nous allons donc créer ce service pour injecter les données.
Commande (dans le répertoire src/app/service) : `ng generate service InMemoryUsers`
N'oubliez pas d'importer la classe InMemoryUsersService dans le module.

Le code sera le suivant :

```
@Injectable({
  providedIn: 'root'
})
export class InMemoryUsersService  implements InMemoryDbService {
  createDb() {
    console.log('InMemoryUsersService.createDb');
    const users: UserLdap[] = LDAP_USERS;
    return {users};
  }

  // Overrides the genId method to ensure that a user always has an id.
  // If the users array is empty,
  // the method below returns the initial number (4).
  // if the users array is not empty, the method below returns the highest
  // user id + 1.
  genId(users: UserLdap[]): number {
    console.log('InMemoryUsersService.genId');
    return users.length > 0 ? Math.max(...users.map(user => user.id)) + 1 : 4;
  }
}
```

Modification de plusieurs classes

Comme le service InMemoryUsersService impose d'utiliser un identifiant entier nommé "id", nous devons modifier notre classe UserLdap pour l'ajouter et de ce fait cela entraînera plusieurs modifications.

Classe UserLdap

```
export interface UserLdap {
  id: number; // A ajouter pour InMemoryUsersService
  login: string;
  ...
```

Fichier ldap-mock-data.ts

Pour chaque élément, il faut ajouter le champ id :

```
export const LDAP_USERS: UserLdap[] = [
  {
    id: 1, // A ajouter pour InMemoryUsersService
    ...
```

Composant LdapListComponent

Si vous le souhaitez, vous pouvez utiliser un SnackBar pour afficher un message lors de l'ajout et l'édition d'un utilisateur.

Modification du service UsersService

Nous allons modifier le service pour utiliser des requêtes HTTP (ces requêtes seront interceptées par le service InMemoryUsersService grâce à l'API In-Memory).

Pour utiliser les requêtes HTTP, il faut ajouter le module HttpClient dans le module principal (AppModule) :

```
@NgModule({
  imports: [
    ...
    HttpClientModule,
    // The HttpClientInMemoryWebApiModule module intercepts HTTP requests
    // and returns simulated server responses.
    // Remove it when a real server is ready to receive requests.
    HttpClientInMemoryWebApiModule.forRoot(
        InMemoryUsersService, options: {dataEncapsulation: false}
    )
  ],
  ...
})
export class AppModule { }
```

Puis il faut importer le client http dans le service UsersService:

```
constructor(private http: HttpClient) {}
```

Comme nous utilisons l'API In-Memory, il faut ajouter l'Url d'accès au service :
`private usersUrl = 'api/users';`

Remarque : Il est important de noter que le mot *users* de 'api/*users*' doit être identique à la variable *users* de `const users: UserLdap[] = LDAP_USERS;` de la classe InMemoryUsersService.

Variables d'environnement

Nous avons défini "en dur" l'Url d'accès au service. Ceci est valable dans le cadre d'un TP mais il est préférable de définir les variables dans les fichiers d'environnement de Angular dans le répertoire "src/environments".
Le répertoire contient deux fichiers pour deux types de "build" (voir Building and serving Angular apps) :

- Le fichier "environments.ts" est utilisé pour le build
- Le fichier "environments.prod.ts" pour un build production

Lors d'un build production, le fichier "environments.ts" est remplacé (écrasé) par le fichier "environments.prod.ts".

Nous aurions pu définir l'Url d'accès dan le fichier "environments.ts":

```
export const environment = {
 production: false,
 usersApiUrl: 'api/users'
};
```

Puis obtenir sa valeur de cette façon :

```
export class UsersService {

private usersUrl = '';

constructor(private http: HttpClient) {
 this.usersUrl = environment.usersApiUrl;
}
...
```

Obtenir tous les utilisateurs

Pour obtenir tous les utilisateurs, nous utilisons la méthode get de la classe HttpClient :

```
getUsers(): Observable<UserLdap[]> {
  return this.http.get<UserLdap[]>(this.usersUrl);
}
```

La méthode get :

- accepte un template : nous lui indiquons d'utiliser une liste d'objets UserLdap
- nécessite uniquement une URL

Obtenir un utilisateur

Pour obtenir un utilisateur, nous utilisons aussi la méthode get de la classe HttpClient en lui indiquant l'id de l'utilisateur :

```
getUser(id: number): Observable<UserLdap> {
  return this.http.get<UserLdap>( url: this.usersUrl + '/' + id);
}
```

Ajouter un utilisateur

Pour ajouter un utilisateur, nous utilisons la méthode post de la classe HttpClient :

```
addUser(user: UserLdap): Observable<UserLdap> {
  return this.http.post<UserLdap>(this.usersUrl, user, options: {
    headers: this.httpOptions
  });
}
```

Il faut ajouter la donnée privée suivante pour définir les options :

```
private httpOptions = new HttpHeaders({'Content-Type': 'application/json'});
```

La méthode post :

- accepte un template : nous lui indiquons d'utiliser un objet UserLdap
- nécessite une URL
- nécessite des données : l'utilisateur à ajouter
- peut avoir des options : dans notre cas nous indiquons utiliser du JSON

Modifier un utilisateur

Pour modifier un utilisateur, nous utilisons la méthode put de la classe HttpClient qui a les mêmes paramètres que la méthode post précédente :

```
updateUser(user: UserLdap): Observable<UserLdap> {
  // Modification de l'utilisateur
  return this.http.put<UserLdap>( url: this.usersUrl + '/' + user.id,
      user, options: { headers: this.httpOptions});
}
```

Supprimer un utilisateur

Pour supprimer un utilisateur, nous utilisons la méthode delete de la classe HttpClient :

```
deleteUser(id: number): Observable<UserLdap> {
  return this.http.delete<UserLdap>( url: this.usersUrl + '/' + id, options: {
    headers: this.httpOptions
  });
}
```

La méthode delete :

- accepte un template : nous lui indiquons d'utiliser un objet UserLdap
- nécessite une URL
- peut avoir des options : dans notre cas nous indiquons utiliser du JSON

Remarque :

Plutôt que d'utiliser de "fausses" données, nous pourrions faire appel à l'API REST "UsersManagements" (Microservice Identity).

Vous avez tous les éléments pour le faire; néanmoins il faut supprimer la classe InMemoryUsersService qui sera alors inutile et changer l'URL de la classe UsersService.

Voir https://angular.io/guide/http

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/