esprit Se former autrement HONORIS UNITED UNIVERSITIES	Année Universitaire : 2025-2026 ANGULAR

Workshop n°9 :

Manipulation des services Angular

Les objectifs :

- Créer et injecter un service.
- Consommer des API REST à travers le service HttpClient
- Manipuler différentes méthodes du service HttpClient : GET, POST, PUT, DELETE
- Manipuler des objets HttpRequest, HttpResponse et HttpErrorResponse
- Manipuler quelques opérateurs RxJS
- Manipuler les méthodes des observables pipe() et subscribe()
- Comprendre et manipuler les observables
- Gérer les erreurs avec catchError() et throwError()

Le rendu final :

Une application dont les informations provient d'un backend.

Mise en place du backend – JSON Server:

1- Ouvrir votre invite de commande et taper la commande suivante :
npm i --global json-server
2- Créer deux dossiers « **backend/data** ».
3- A l'intérieur du dossier « **backend/data**», taper la commande :
`json-server --watch db.json`
4- Vous devez obtenir le résultat suivant dans votre cmd :

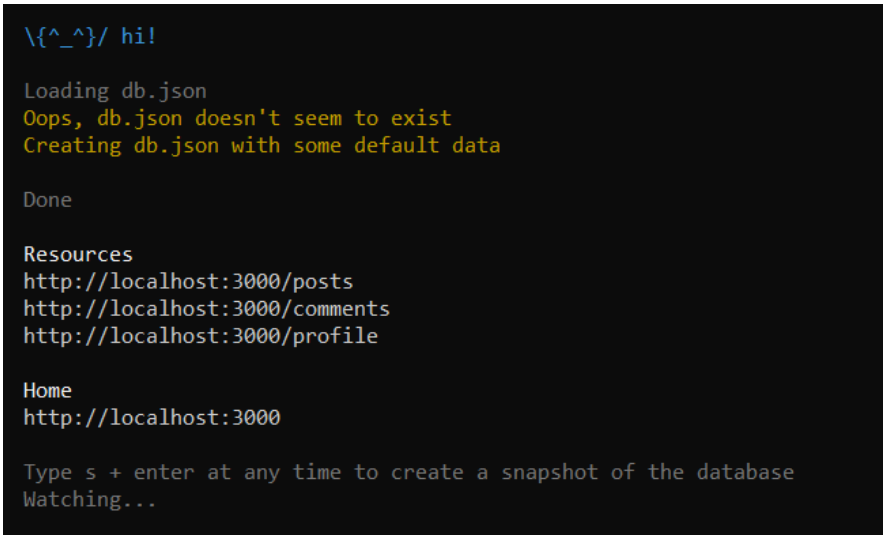

Figure 1 : json-server –watch db.json

5- Tester les différentes urls affichés dans Figure 1.
6- Ouvrir le fichier data/db.json et comparer son contenu avec ce que vous avez obtenu dans la question 5.
7- Ajouter dans le fichier db.json le contenu suivant :

```
"events": [
  {
   "id": 1,
   "title": "Angular Summit",
   "description": "Conférence sur Angular et l'écosystème front-end",
   "date": "2025-11-10",
   "place": "Tunis",
   "price": 50,
   "organizerId": 1,
   "imageUrl": "images/event1.PNG",
   "nbPlaces": 25,
   "nbLikes": 0
  },
  {
   "id": 2,
   "title": "Web dev days",
   "description": "Journée dédiée aux frameworks web modernes.",
   "date": "2025-01-05",
   "place": "Ariana",
   "price": 30,
   "organizerId": 1,
   "imageUrl": "images/event2.PNG",
   "nbPlaces": 0,
   "nbLikes": 0
  }]
```

8- Taper l'url : localhost:3000/events

Exercice 1 : Récupération de la liste des évènements à partir du backend

A- Récupération des données à travers HttpClient – méthode get()

1- Créer un service **EventsService** sous **data-access/services** si ceci n'est pas encore fait à travers la commande

```
ng g s data-access/services/events
```

2- Définir dans providers de @NgModule de AppModule « **provideHttpClient**() » comme suit :

```
providers: [provideHttpClient()],
```

3- Injecter le service **HttpClient** au niveau de ce service :

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

4- Ajouter une nouvelle propriété apiEventsUrl de type string comme indiqué ci-dessous :

```
private apiEventsUrl = 'http://localhost:3000/events';
```

5- Ajouter la méthodes **getAllEventsFromBackend()** qui retourne la liste des évènements à travers l'appel de l'api correspondante à apiEventsUrl comme suit :

```
getAllEventsFromBacked(): Observable<Event[]> {
  return this._http.get<Event[]>(this.apiEventsUrl)
}
```

6- Injecter le service EventsService dans le composant ListEventsComponent si ceci n'est pas encore fait. Ensuite, au niveau de la méthode ngOnInit() du composant ListEventsComponent, appeler la méthode getAllEventsFromBackend() du service EventsService afin de récupérer la liste des événements.

```
listEvent : Event[] =[];
constructor(private es:EventsService){}
ngOnInit(){
  this.es.getAllEventsFromBacked().subscribe({
    next:(response)=>this.listEvent=response,
}
```

7- Vérifier que les données affichées sont bien récupérées à partir du backend.

8- Au niveau de votre navigateur, ouvrir DevTools (clic droit –>inspecter) et aller à l'onglet « Network », vous allez voir votre requête.

B- Manipulation de l'objet HttpRequest : Faire une requête GET avec params + headers

9- Nous souhaitons que notre requête soit de la forme :
GET http://localhost:3000/events?role=admin&active=true
Avec :

- Un header : Authorization: Bearer 123
- Un header : Content-Type: application/json

Ajouter les options nécessaires au niveau de votre requête.

```
getAllEventsFromBacked(): Observable<Event[]> {

return this._http.get<Event[]>(this.apiEventsUrl,{

params: { active: 'true', sort: 'name' },
```

```
headers: {'Authorization': 'Bearer 123', 'Content-Type': 'application/json'},

})}
```

C- HttpRequest – option observe

10- Ajouter à la requête de la méthode ci-dessus l'option observe : 'response' qui permet de récupérer la réponse complète

```
getAllEventsFromBacked(): Observable<HttpResponse<Event[]>> {
  return this._http.get<Event[]>(this.apiEventsUrl,{
    params: { active: 'true', sort: 'name' },
    headers: {'Authorization': 'Bearer 123', 'Content-Type': 'application/json'},
    observe:'response'
})}
```

D- Manipulation de HttpResponse

11- Au niveau de votre composant, mettez à jour votre code pour récupérer la liste des évènements à travers le **body** de la réponse et afficher dans la console le statut **« status »** et le texte du statut **« statusText »** de la réponse.

```
ngOnInit(){
  this.es.getAllEventsFromBacked().subscribe({
    next:(response)=>{
        this.listEvent=response.body;
        console.log(response.status+" : " +response.statusText)},
     });
  }
```

E- Manipulation de HttpErrorResponse : Retourner les erreurs avec catchError() et throwError()

Nous souhaitons maintenant gérer les erreurs qui peuvent être commises de sorte que si une erreur survient, la méthode du service retourne tout l'objet HttpErrorResponse. Pour cela :

12- Mettez à jour votre méthode getAlleventsFromBackend() en ajoutant catchError() et throwError() comme suit :

```
getAllEventsFromBacked(): Observable<HttpResponse<Event[]>> {
return this._http.get<Event[]>(this.apiEventsUrl,{
```

```
        params: { active: 'true', sort: 'name' },
        headers: {'Authorization': 'Bearer 123', 'Content-Type':
'application/json'},
        observe:'response'
}).pipe(
  catchError((error:HttpErrorResponse)=>{return
throwError(()=>error)})
  )
```

13- Au niveau du composant, récupérer l'erreur et afficher le message d'erreur envoyé par Angular dans la console comme suit :

```
this.es.getAllEventsFromBacked().subscribe({
    next:(response)=>{
      this.listEvent=response.body;
      console.log(response.status+" : " +response.statusText)},
    error:(error)=>console.log(error?.message)});
  }
```

14- Afin de tester l'erreur, provoquer une erreur coté serveur, par exemple modifier dans le fichier db.json « events » par « event ».

15- Essayer d'afficher le message généré par votre backend en remplaçant « error?.message » par à « error?.error?.message », vous devez voir « undefined » dans votre console.

NB : n'oubliez pas de corriger votre db.json pour continuer le reste de l'atelier.

Exercice 2 : Transformer une réponse HTTP et gérer les erreurs

1- **Envoyer une requête GET** vers l'URL : http://localhost:3000/events dans une nouvelle méthode de votre service appelé **getExpensiveEvents()**

2- **Transformer la réponse en utilisant des opérateurs RxJS** :

- Garder uniquement les évènements dont price > 50 (filter)

```
map(events => events.filter(e => e.price > 50)),
```

NB : **.filter()** est la méthode JS et non pas un opérateur RxJS

- Transformer chaque évènement en objet simplifié :

{title: string, finalPrice: number} où finalPrice = price * 1.2 (TVA 20%)

```
map(events =>
      events.map(e => ({
        title: e.title,
        finalPrice: e.price * 1.2
      }))
    ),
```

NB : le 1er map est l'opérateur de RxJS, le 2ème map est une fonction JS

3- **Gérer les erreurs** :

- Si le backend renvoie une erreur, tu dois retourner un *Observable* contenant **un tableau vide**

```
catchError(() => of([]))
```

4- **Afficher le résultat dans la console au niveau du composant ListEventsComponent.**

Exercice 3 : Création d'un service interne pour la gestion des erreurs

1- Créer un service appelé « ErrorService » sous **shared/services** qui permet de centraliser la gestion des erreurs à travers sa méthode handleError() comme suit :

```
handleError(error: HttpErrorResponse) {
  let message = '';

  if (error.error instanceof ErrorEvent) {
   message = `Erreur réseau : ${error.error.message}`;
  } else {
    message = `Erreur serveur (${error.status}) : ${error.message}`;
  }
  console.log(message);

  return throwError(() => message);
 }
```

2- Au niveau du service EventsService, utiliser la méthode handleError() pour retourner l'erreur lorsqu'elle est produite.

NB : pour tester votre code il faut faire une erreur coté serveur comme on a afit avant (par exemple changer events dans db.json)