



Accès à un serveur avec HttpClient

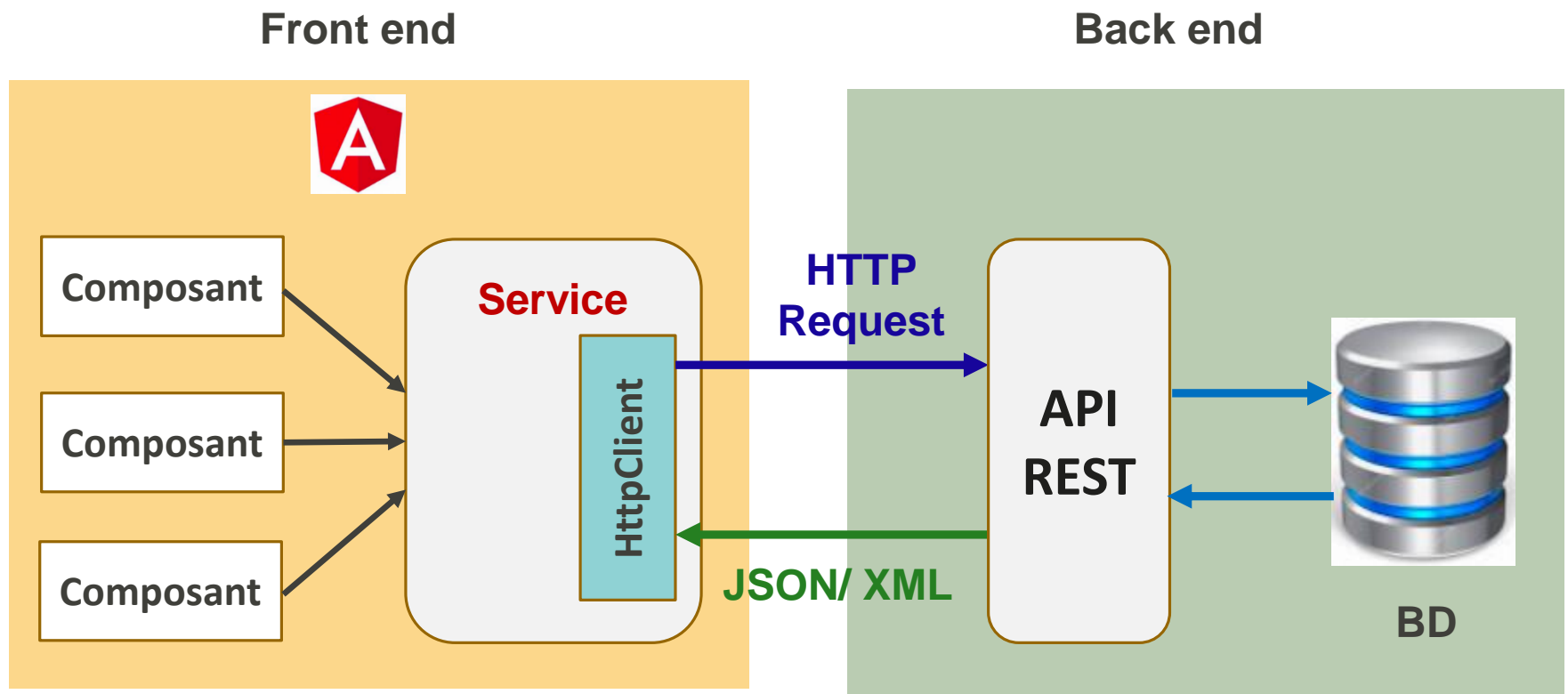
Enseignante: Amel TRIKI

Année Universitaire 2022-2023

Plan

- Communication avec le back end
- Le protocole HTTP
- API REST
- Le serveur JSON
- Le service HttpClient
- Les observables
- Exemple illustratif d'un accès au serveur
- Le pipe async

Communication avec le back end



Le protocole HTTP (1/2)

- La majorité des applications Front end communiquent avec le serveur via le protocole **HTTP** (**H**yper**T**ext **T**ransfer **P**rotocol)



- HTTP fournit principalement 4 verbes pour la manipulation des ressources: **GET** , **POST**, **PUT** et **DELETE**

Exemple: GET <https://api.site.tn/voiture?immat=141tu4862>

Le protocole HTTP (2/2)

- HTTP définit 40 codes de statut répertoriés en 5 catégories

Code	Signification
1XX	Message d'information Exp: 101 Demande de changement de protocole
2XX	Succès de la requête client Exp: 201 Document créé
3XX	Redirection de la requête client Exp: 302 Le document a changé d'adresse temporairement
4XX	Requête client incomplète Exp: 404 document introuvable
5XX	Erreur du serveur Exp: 500 Erreur inattendue au niveau du serveur

API REST (1/3)

- **API** (Application Programming Interface) permet l'interaction entre différentes applications
- Une API liste les opérations possibles où chaque action est accessible via une url spécifique

Exemple: <https://api.github.com/users/trikiiset>

- Une **API REST** (Representational State Transfer) suit un style d'architecture qui:
 - Utilise explicitement les méthodes HTTP
 - Utilise l'URI pour exposer la logique métier
 - Renvoie une réponse au format JSON, XML, HTML, ...

API REST (2/3)

- Une API REST se base sur les **URI** pour identifier une ressource
- Les URI sont construites de façon hiérarchique avec une certaine sémantique

Exemple:

- *Liste des produits*

`http://monsite.com/produits`

- *Affichage du produit d'id 15*

`http://monsite.com/produits/15`

- *Filtre et tri sur des produits:*

`http://monsite.com/produits?type=info&tri=asc`

API REST (3/3)

Exemples:

- Tous les commentaires du produit d'id 15

<http://monsite.com/produits/15/comments>

- Commentaire précis d'un produit

<http://monsite.com/produits/15/comments/25>

Exemples:

Pour décrire une action à réaliser sur les ressources

- Afficher tous les produits : **GET** <http://monsite.com/produits>
- Créer un produit : **POST** <http://monsite.com/produits>
- Effacer le produit d'id 15 : **DELETE** <http://monsite.com/produits/15>

Le serveur JSON

- Le serveur **JSON** est un package **npm** permettant de créer des API REST au format **json**
- Il est destiné aux développeurs front end pour effectuer des opérations **CRUD** (**C**reate, **R**ead, **U**ppdate, **D**eleate) sans disposer de backend à proprement dit : il est utilisé en tant que serveur de **test**

Configuration d'un serveur JSON

1

Installation du serveur json:

npm install -g json-server

2

Création d'un fichier json avec des données

Exemple: bd.json

```
{  "produits":[    {"id":1, "libelle":"stylo", "prix":0.45},    {"id":2, "libelle":"livre", "prix":5.2}  ]}
```

3

Lancement du serveur json

json-server --watch bd.json

4

Vérification du fonctionnement

http://localhost:3000/produits

Le service HttpClient

- **HttpClient** est un service permettant de communiquer avec le serveur
- **HttpClient** est inclus dans **HttpClientModule** défini dans **app.module.ts**

```
import {HttpClientModule} from "@angular/common/http";  
...  
imports: [  
  BrowserModule,  
  HttpClientModule  
],
```

- **HttpClient** définit des méthodes permettant d'accéder au serveur pour effectuer des opérations et renvoient un objet de type **Observable**

Quelques méthodes de HttpClient

HttpClient offre différents méthodes

Méthode	Paramètres	Rôle
get	(url [,httpOptions])	Récupération de données
post	(url, données [,httpOptions])	Ajout de données
put	(url, données [,httpOptions])	Remplacement de données
delete	(url [,httpOptions])	Suppression
patch	(url, données [,httpOptions])	Modification de données

- Généralement, les données représentent un **objet**
- Toutes les méthodes peuvent définir en plus un attribut décrivant un objet de type **HttpOptions**

Les observables

- Les différentes méthodes de HttpClient renvoient un **observable**
- Un observable permet de gérer des flux asynchrones (ce qui est le cas des réponses HTTP)
- Un observable est défini dans RxJS

```
import { Observable } from 'rxjs';
```

- Un observable émet des valeurs quand celles-ci sont disponibles

Principe de l'observable

Le principe de l'observable est très proche de celui d'une chaîne Youtube:

- | | |
|---|--|
| <ul style="list-style-type: none">■ Chaîne Youtube■ Personne abonnée■ Il faut demander à s'abonner à une chaîne youtube■ Quand il y a une nouvelle vidéo, une notification est diffusée aux abonnés■ Un abonné peut demander de se désabonner | <ul style="list-style-type: none">■ Observable■ Observer■ Méthode subscribre de l'observable■ Fonction callback de l'observable■ Méthode unsubscribe de l'observable |
|---|--|

Abonnement à un observable

```
objetObservable.subscribe(  
  value => {  
    // valueCallback  
  },  
  error => {  
    // errorCallback  
  },  
  () => {  
    // completeCallback  
  }  
)
```

Obligatoire: fonction qui s'exécute à chaque fois qu'une donnée est disponible

Optionnel: fonction qui s'exécute en cas d'erreur

Optionnel: fonction qui s'exécute quand le flux est terminé

Une requête n'est exécutée que lorsque la méthode **subscribe** est appelée

Exemple illustratif d'un accès à un serveur JSON

- Enoncé
- Etapes
- Implémentation du service
- Récupération des produits
- Ajout d'un produit
- Modification d'un produit
- Suppression d'un produit

Enoncé

On souhaite développer une application permettant de :

- Afficher la liste des produits
- Ajouter un nouveau produit
- Modifier un produit
- Supprimer un produit

Opérations sur les produits

- Id: 1 - stylo - Prix: 0.45 [Supprimer](#) [Modifier](#)
- Id: 2 - livre - Prix: 5.2 [Supprimer](#) [Modifier](#)

Libellé

Prix

Etapes (1/2)

- 1) Création d'un fichier **bd.json** qui joue le rôle d'un Fake API REST et lancement du serveur:

json-server --watch bd.json

- 2) Définition d'une classe **Produit**

```
export class Produit {  
  id: number;  
  libelle:string;  
  prix:number;  
}
```

```
{  
  "produits": [  
    {  
      "libelle": "stylo",  
      "prix": 0.45,  
      "id": 1  
    },  
    {  
      "libelle": "livre",  
      "prix": 5.2,  
      "id": 2  
    }  
  ]  
}
```

bd.json

Etapes (2/2)

3) Importation de **HttpClientModule** dans **app.module.ts**

```
import { HttpClientModule } from "@angular/common/http";  
...  
  imports: [  
    BrowserModule,  
    HttpClientModule  
  ],
```

- 4) Implémentation des méthodes d'accès au serveur dans le service
- 5) Implémentation de la classe **ProduitComponent** en **s'abonnant** aux différentes méthodes
- 6) Implémentation du template du composant

Implémentation du service

produit.service.ts

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { Produit } from '../produit';

const URL = "http://localhost:3000/produits";

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

  constructor(private http: HttpClient) { }

}
```

Url permettant
l'accès à la
collection *produits*

Injection du
service HttpClient

Récupération des produits

produit.service.ts

Ce n'est pas obligatoire, c'est pour typer la réponse

```
getProduits():Observable<Produit[]>{  
    return this.http.get<Produit[]>(URL);  
}
```

Méthode **get** de
HttpClient

URL pour accéder à la
collection **produits**
dans bd.json

Abonnement à getProduits

produit.component.ts

```
export class ProduitComponent implements OnInit {  
  lesProduits: Produit[];  
  
  constructor(private produitService: ProduitService) { }  
  ngOnInit(): void {    afficherProduits();  }  
  
  afficherProduits(){  
    this.produitService.getProduits()  
    .subscribe( data => this.lesProduits = data)  
  }  
}
```

Implémentation du template

Affichage de la liste des produits

produit.component.html

```
<h3>Opérations sur les produits</h3>
<ul>
  <li *ngFor= "let p of lesProduits">
    Id: {{p.id}} - {{p.libelle}} - Prix: {{p.prix}}
  </li>
</ul>
```

Ajout d'un produit

produit.service.ts

La réponse HTTP contiendra l'objet de type **Produit** qui a été ajouté

```
addProduit(p:Produit):Observable<Produit>{  
    return this.http.post<Produit>(URL, p);  
}
```

Méthode **post** de
HttpClient

URL pour accéder à la
collection **produits**
dans bd.json

Objet de type
Produit à ajouter

Abonnement à addProduit

produit.component.ts

```
productForm: FormGroup;
constructor(private produitService: ProduitService,
             private FormBuilder: FormBuilder) { }

ngOnInit(): void {
  this.productForm = this.formBuilder.group(
    { libelle: '', prix: 0 }
  );
  this.afficherProduits();
}

onAjouter(){
  this.produitService.addProduit(this.productForm.value)
    .subscribe(data => console.log(data));
}
```

Implémentation du template

Ajout d'un produit

produit.component.html

```
<form [formGroup]="productForm" >

  <label>Libellé</label>
  <input type="text" formControlName="libelle">

  <div class="form-group">
    <label>Prix</label>
    <input type="number" formControlName="prix">
  </div>
  <button type="button" (click)="onAjouter()">Ajouter</button>

</form>
```

Modification d'un produit

produit.service.ts

On peut l'omettre si on ne cherche pas à récupérer le produit ajouté

```
updateProduit(id:number, p:Produit):Observable<Produit>{  
    return this.http.put<Produit>(URL+"/"+ id, p);  
}
```

Ecriture équivalente

```
return this.http.put<Produit>(` ${URL}/${id}`, p);
```

Abonnement à updateProduit

produit.component.ts

```
export class ProduitComponent implements OnInit {  
  productForm: FormGroup;  
  ...  
  
  onModifier(id:number){  
    this.produitService.updateProduit(id, this.productForm.value)  
    .subscribe(data => console.log(data));  
  }  
}
```

Implémentation du template

Modification d'un produit donné

produit.component.html

```
<h3>Opérations sur les produits</h3>
<ul>
  <li *ngFor= "let p of lesProduits">
    Id: {{p.id}} - {{p.libelle}} - Prix: {{p.prix}}

    <button type="button" class="btn btn-
link" (click)="onModifier(p.id)">Modifier</button>
  </li>
</ul>
```

Suppression d'un produit

produit.service.ts

```
deleteProduit(id:number){  
    return this.http.delete(URL+"/"+ id);  
}
```

Ecriture équivalente



```
return this.http.delete(`${URL}/${id}`);
```

Abonnement à deleteProduit

produit.component.ts

```
export class ProduitComponent implements OnInit {  
  productForm: FormGroup;  
  ...  
  
  onSupprimer(id:number){  
    this.produitService.deleteProduit(id)  
    .subscribe();  
  }  
}
```

Le pipe async

Le pipe **async** souscrit à un objet asynchrone (Observable dans notre cas) et insère les soumissions dans le DOM

Syntaxe dans le template:

```
ObjetAsynchrone$ | async
```



async n'est pas un pipe de formatage des données

Exemple (1/2)

On reprend l'exemple où on veut afficher la liste des produits

Convention de nommage: le nom de l'observable finit par \$

produit.component.ts

```
export class ProduitComponent implements OnInit {  
  lesProduits$!: Observable<Produit[]>;  
  
  constructor(private produitService: ProduitService) { }  
  ngOnInit(): void {    afficherProduits();  }  
  
  afficherProduits(){  
    this.lesProduits$ = this.produitService.getProduits();  
  }  
}
```

Exemple 2/2

Affichage de la liste des produits

produit.component.html

Nom de l'observable

```
<h3>Opérations sur les produits</h3>
<ul>
  <li *ngFor= "let p of lesProduits$" | async>
    Id: {{p.id}} - {{p.libelle}} - Prix: {{p.prix}}
  </li>
</ul>
```

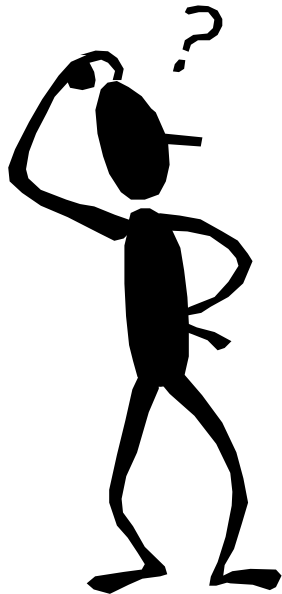
Aller plus loin

Ce qui a été présenté dans ce cours est très succinct, il est possible de:

- Définir des requêtes avec des options
- Récupérer toute la réponse HTTP (avec l'entête)
- Gérer les erreurs HTTP
- Appliquer des opérateurs de bas et de haut niveau sur les observables
-



Des questions?



Références

- <https://angular.io/guide/http>
- <https://github.com/typicode/json-server>
- <https://remotestack.io/angular-httpclient-service-example-tutorial/>
- <https://www.techiediaries.com/angular-14/>