Service HttPClient

Plan

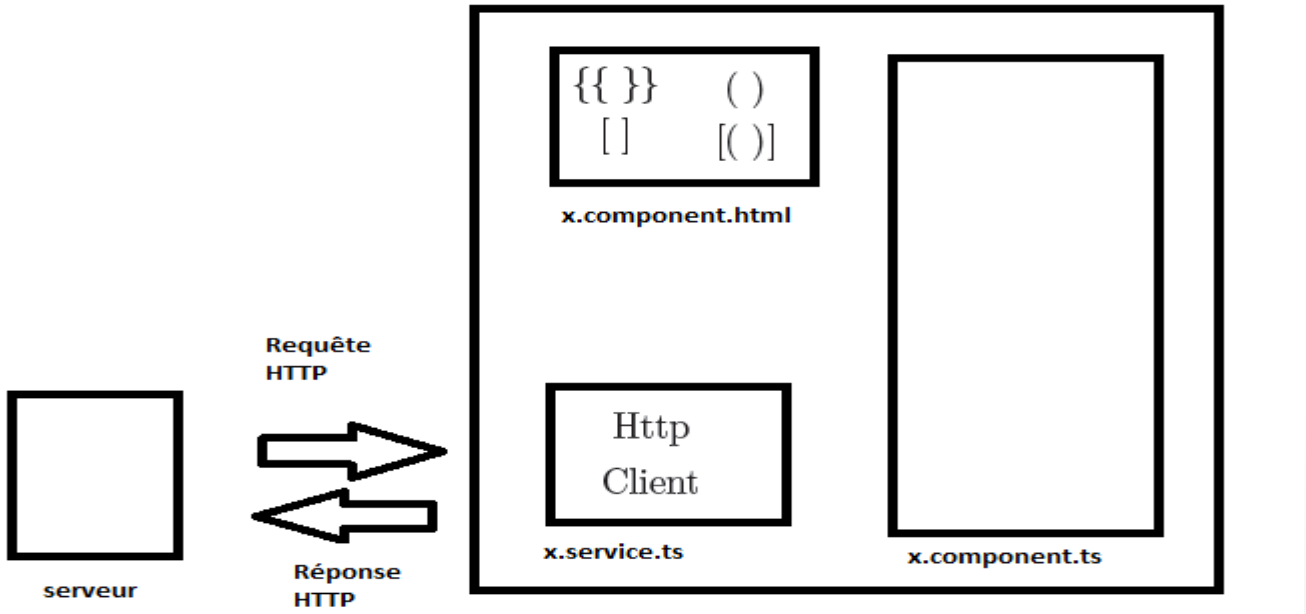
- Introduction
- Classe HttpClient
- Protocole HTTP
- Mise en œuvre de HttpClient
- Exemple d’application
- Fichiers d’environnement
- Package concurently

Introduction

- Angular assure la communication avec le serveur à travers un module nommé HttpModule (HttpClientModule depuis la version 5) facilitant la réalisation de requête HTTP vers n'importe quel serveur via les classes suivantes :
- HttpClient
- HttpHeaders
- HttpInterceptor
- HttpRequest
- ...
- Pour le coté serveur : on peut utiliser un serveur JSON qui recevra des requêtes HTTP envoyées par Angular et retourner une réponse

Classe HttpClient

- « HttpClient est disponible sous forme de classe injectable, avec des méthodes pour effectuer des requêtes HTTP. Chaque méthode de requête a plusieurs signatures et le type de retour varie en fonction de la signature appelée (principalement les valeurs de observe et responseType).”



Classe HttpClient

- Avantages du service HttpClient
- Test facile
- Objets de requête et de réponse typés
- Interception de demande
- Interception de réponse
- Support observable des API
- Gestion facile des erreurs

Classe HttpClient

- Méthodes du service HTTPClient
- request()
- delete()
- get()
- patch()
- post()
- put()
- head()
- jsonp()
- options()

Protocole HTTP

- Hypertext Transfer Protocol
- Protocole de communication entre client et serveur
- Basé sur la notion requête - réponse appelée généralement (HTTP Request - HTTP Response)
- Plusieurs types de requêtes = méthodes HTTP
- GET: récupérer une ressource
- POST: création d'une nouvelle ressources
- DELETE: suppression d'une ressource
- PUT: création ou substitution d'une ressource
- Toutes ces méthodes prennent en paramètre l'adresse du serveur (+ pour certaines méthodes les données à manipuler)

Mise en œuvre de HttpClient

- Préparation de la communication avec le serveur:
- On importe le module HttPClient Module dans le module principal
- On injecte le service HttPClient
- On fait appel à la méthode HttpClient.get() pour récupérer les données d'un serveur. La méthode asynchrone envoie une requête HTTP et renvoie un Observable qui émet les données demandées lorsque la réponse est reçue
- La méthode get() prend deux arguments ; l'URL du point de terminaison à partir de laquelle récupérer et un objet d'options utilisé pour configurer la demande
- On peut préciser le type des données retournées par la méthode get() « observable »

Mise en œuvre de HttpClient

- Serveur **HTTP** de test Front-End pour les développeurs Front-End
- Open-source
- Utilisant par défaut le port 3000
- Documentation : https://github.com/typicode/json-server

```
npm install -g json-server
```

Exemple d'application

- Exemple:
- Créons un fichier personnes.json qui va nous servir de base de données et qui sera situé au même niveau que le dossier du notre projet

```
{
  "personnes": [
    {
      "nom": "wick",
      "prenom": "john",
      "id": 1
    },
    {
      "nom": "green",
      "prenom": "bob",
      "id": 2
    }
  ]
}
```

Exemple d'application

- On affiche la liste des éléments de notre base en se plaçant dans le dossier contenant notre fichier json et en exécutant les commande suivantes:
- On lance le serveur en utilisant un numéro de port 5555 et en précisant le nom de base de données

- On affiche le résultat

```
json-server -p 5555 personnes.json
```

```
Resources http://localhost:5555/personnes
```

Exemple d'application

- http://localhost:5555/personnes
- C'est l'URL qui sera utilisée par le client pour réaliser des requêtes HTTP
- Si on copie cette adresse et qu'on la colle dans le navigateur, on obtient toutes les données de notre base de données

Exemple d'application

Pour récupérer la liste de toutes les personnes

- GET : http://localhost:5555/personnes

Pour récupérer une personne selon l'identifiant (33 par exemple)

- GET : http://localhost:5555/personnes/33

Pour ajouter une nouvelle personne

- POST : http://localhost:5555/personnes

Pour modifier les valeurs d'une personne existante (ayant l'identifiant 33)

- PUT : http://localhost:5555/personnes/33

Pour supprimer une personne existante (ayant l'identifiant 33)

- DELETE : http://localhost:5555/personnes/33

Exemple d'application

- Les données sont saisies (ou affichées) dans le template .component.html
- La classe .component.ts peut récupérer des données dont la source est le template .component.html pour les passer au service ou récupérer des données dont la source est le service pour les passer au template .component.html.
- En faisant une injection de dépendance du service .service.ts dans .component.ts, ce dernier peut l'utiliser pour persister ou récupérer des données
- En faisant une injection de dépendance de la classe HttpClient dans .service.ts, ce dernier peut effectuer des requêtes HTTP en précisant chaque fois la méthode et l'URL.

Exemple d'application

- On considère le service personne

```
import { Injectable } from '@angular/core';
import { Personne } from '../interfaces/personne';

@Injectable({
  providedIn: 'root'
})
export class PersonneService {
  url = 'http://localhost:5555/personnes/';
  constructor() {

  }
  getAll(){

  }
  addPerson(p: Personne){

  }
}
```

Exemple d'application

- Dans le constructeur de la classe personne on ajoute le service personne.service

```
import { Component, OnInit } from '@angular/core';
import { Personne } from '../../interfaces/personne';
import { PersonneService } from '../../services/personne.service';

@Component({
  selector: 'app-personne',
  templateUrl: './personne.component.html',
  styleUrls: ['./personne.component.css']
})
export class PersonneComponent implements OnInit {
  personne: Personne = {};
  personnes: Array <Personne> = [];

  constructor(private personneService: PersonneService) {  }

  ngOnInit() {
    this.personnes = this.personneService.getAll();
  }
}
```

Exemple d'application

- Le résultat s'affiche dans le template:

```
<h2>Liste des personnes</h2>
<ul>
  <li *ngFor="let elt of personnes">
    {{ elt.prenom }}  {{ elt.nom }}
  </li>
</ul>
```

- On injecte le service HttpCLient dans le service personne:

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

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

  url = 'http://localhost:5555/personnes/';

  constructor(private http: HttpClient) {

  }
  getAll(){

  }
  addPerson(p: Personne){

  }
}
```

Exemple d'application

- On appelle HttpClient dans le module principal:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
//+ les autres imports

@NgModule({
  declarations: [
    AppComponent,
    AdresseComponent,
    PersonneComponent,
    FormulaireComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [PersonneService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Exemple d'application

- Requête http afin de récupérer la liste de personnes

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

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

  url = 'http://localhost:5555/personnes/';

  constructor(private http: HttpClient) {

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

  }
}
```

Exemple d'application

- Dans le composant personne.component.ts il faut utiliser la méthode subscribe pour récupérer la liste de personne

```
import { Component, OnInit } from '@angular/core';
import { Personne } from '../../interfaces/personne';
import { PersonneService } from '../../services/personne.service';

@Component({
  selector: 'app-personne',
  templateUrl: './personne.component.html',
  styleUrls: ['./personne.component.css']
})
export class PersonneComponent implements OnInit {
  personne: Personne = {};
  personnes: Array <Personne> = [];

  constructor(private personneService: PersonneService) {  }

  ngOnInit(): void {
    this.personneService.getAll().subscribe(res => {
      this.personnes = res;
    });
  }
}
```

Exemple d'application

- Formulaire d'ajout d'une personne

```
<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>
<!-- garder la partie permettant d'afficher les personnes -->
```

Exemple d'application

- Préparation d'une requête http d'ajout d'une personne

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

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

  url = 'http://localhost:5555/personnes/';

  constructor(private http: HttpClient) { }

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

Exemple d'application

- On ajoute la méthode ajouterPersonne() dans le composant personne.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Personne } from '../../interfaces/personne';
import { PersonneService } from '../../services/personne.service';

@Component({
  selector: 'app-personne',
  templateUrl: './personne.component.html',
  styleUrls: ['./personne.component.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 => {
      console.log(res);
    });
  }
}
```

Explication

- Lors de l’ajout:
- La personne est ajoutée dans le fichier personnes.json
- Mais elle n’est pas affichée dans la page qu’après actualisation
- Car on ne met pas à jour la liste des personnes
- Soit on ajoute la personne au tableau personnes de la classe personne.component.ts lorsqu’on l’ajoute dans notre fichier personnes.json
- Soit on refait appel à la méthode getAll() de notre service personne.service.ts dans personne.component.ts lorsqu’on l’ajoute dans notre fichier personnes.json pour mettre à jour le tableau personnes

Exemple d'application

- Modifions la méthode ajouterPersonne() dans personne.component.ts

```
// les imports
@Component({
  selector: 'app-personne',
  templateUrl: './personne.component.html',
  styleUrls: ['./personne.component.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.personneService.getAll().subscribe(result => {
        this.personnes = result;
      });
    });
  }
}
```

Fichiers d'environnement

- Le répertoire environments contient deux fichiers permettant de définir les variables d'environnement d'une application Angular
- environment.ts pour le mode développement
- environment.prod.ts pour le mode production
- Angular remplacera automatiquement le premier fichier par le deuxième fichier à chaque fois exécution de ng build --prod.

Fichiers d'environnement

- Contenu initial de environment.ts

```
export const environment = {
  production: false,
};
```

- Ajoutons dans environment.ts l'adresse le préfixe de nos API REST

```
export const environment = {
  production: false,
  APIEndpoint: 'http://localhost:5555/'
};
```

Fichiers d'environnement

- Modifions personne.service.ts et utilisons la variable ajoutée dans environment.ts

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

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

  url: string;

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

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

Package concurently

- On doit utiliser le package concurrently pour ne démarrer le serveur JSON et Angular simultanément.
- Package **NodeJS**
- Donne la main pour exécuter plusieurs commandes simultanément
- Syntaxe concurrently "command1 arg" "command2 arg"

Package concurently

- Installation:
- En exécutant la commande:

```
npm install concurrently --save
```

- Le script suivant sera consulté et particulièrement la partie start

```
npm start
```

```
"scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
},
```

Package concurently

- On modifie la section start pour que les deux serveurs pour démarrer les deux serveurs simultanément

```
"scripts": {
    "ng": "ng",
    "start": "concurrently \"ng serve\" \"json-server -p 5555 personnes
      .json\"",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
},
```

Merci pour votre attention