Atelier 05 Angular :

Consommer des api REST à l'aide de HttpClient

L'objectif général de cet atelier est de vous apprendre comment consommer des api Rest fournis par notre application Backend Spring boot, à l'aide du module HttpClient.

Objectifs :

1. Ajouter le provider provideHttpClient(),
2. Utiliser HttpClient pour appeler l'api REST qui retourner tous les produits,
3. Utiliser HttpClient pour appeler l'api REST qui ajoute un nouveau produit,
4. Utiliser HttpClient pour appeler l'api REST qui supprime un produit,
5. Utiliser HttpClient pour appeler l'api REST qui modifie un produit,
6. Utiliser HttpClient pour appeler l'api REST concernant les catégories.

Ajouter le provider provideHttpClient()

1. Ajouter le provider provideHttpClient() au fichier app.config.ts

```
import { provideHttpClient } from '@angular/common/http';
```

```
 providers: [provideZoneChangeDetection({ eventCoalescing: true }),
              provideRouter(routes),
              provideHttpClient()
            ]
```

Utiliser HttpClient pour appeler l'api REST qui retourner tous les produits :

2. Modifier le fichier produit.service.ts :

```
…
import { Observable } from 'rxjs';
import { HttpClient, HttpHeaders } from '@angular/common/http';
const httpOptions = {
  headers: new HttpHeaders( {'Content-Type': 'application/json'} )
};
```

```
…
export class ProduitService {
  apiURL: string = 'http://localhost:8080/produits/api';
 constructor(private http : HttpClient) {

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

3. Modifier le fichier produit.component.ts :

```
 ngOnInit(): void {
    this.produitService.listeProduit().subscribe(prods => {
      console.log(prods);
      this.produits = prods;
    });
  }
```

4. Testez votre travail

Utiliser HttpClient pour appeler l'api REST qui ajoute un nouveau produit :

5. Modifier la méthode *ajouterProduit()* du fichier produit.service.ts :

```
 ajouterProduit( prod: Produit):Observable<Produit>{
      return this.http.post<Produit>(this.apiURL, prod, httpOptions);
    }
```

6. Modifier la méthode *ajouterProduit()* de la classe AddProduitComponent :

```
addProduit(){
    this.produitService.ajouterProduit(this.newProduit)
    .subscribe(prod => {
      console.log(prod);
       this.router.navigate(['produits']);
      });
   }
```

7. Testez votre travail, en ajoutant un nouveau produit.

Utiliser HttpClient pour appeler l'api REST qui supprime un produit :

8. Modifier la méthode *supprimerProduit()* de la classe *ProduitService* :

```
supprimerProduit(id : number) {
     const url = `${this.apiURL}/${id}`;
      return this.http.delete(url, httpOptions);
          }
```

9. Modifier la méthode *supprimerProduit()* de la classe ProduitsComponent:

```
  ngOnInit(): void {
    this.chargerProduits();
  }


chargerProduits(){
    this.produitService.listeProduit().subscribe(prods => {
      console.log(prods);
      this.produits = prods;
    });
  }

  supprimerProduit(p: Produit)
    {
      let conf = confirm("Etes-vous sûr ?");
      if (conf)
      this.produitService.supprimerProduit(p.idProduit).subscribe(() => {
        console.log("produit supprimé");
        this.chargerProduits();
           });
    }
```

Utiliser HttpClient pour appeler l'api REST qui modifie un produit :

10. Modifier la méthode *consulterProduit()* de la classe *ProduitService* :

```
 consulterProduit(id: number): Observable<Produit> {
      const url = `${this.apiURL}/${id}`;
      return this.http.get<Produit>(url);
    }
```

11. Modifier la méthode ***ngOnInit()*** de la classe *UpdateProduitComponent :*

```
ngOnInit() {
  this.produitService.consulterProduit(this.activatedRoute.snapshot.params['id']).

   subscribe( prod =>{ this.currentProduit = prod; } ) ;

  }
```

12. Modifier la méthode update*Produit()* de la classe *ProduitService* :

```
updateProduit(prod :Produit) : Observable<Produit>
    {
    return this.http.put<Produit>(this.apiURL, prod, httpOptions);

    }
```

13. Modifier la méthode *updateProduit()* de la classe *UpdateProduitComponent*:

```
updateProduit() {
    this.produitService.updateProduit(this.currentProduit).subscribe(prod => {
      this.router.navigate(['produits']); }
        );
  }
```

14. Testez la modification d'un produit.

Utiliser HttpClient pour appeler l'api REST concernant les categories

15. **Angular :** Enlevez les commentaires des listes déroulantes contenant les categories des produits,
16. **Spring Boot :** Au niveau de l'application Spring boot, Créer l'interface CategorieRepository,

```
package com.nadhem.produits.repos;

import org.springframework.data.jpa.repository.JpaRepository;
import com.nadhem.produits.entities.Categorie;

public interface CategorieRepository extends JpaRepository<Categorie, Long> {

}
```

17. Toujours, Au niveau de l'application Spring boot, Créer la classe CategorieRESTController,

```
package com.nadhem.produits.restcontrollers;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import com.nadhem.produits.entities.Categorie;
import com.nadhem.produits.repos.CategorieRepository;

@RestController
@RequestMapping("/api/cat")
@CrossOrigin("*")
public class CategorieRESTController {

	@Autowired
	CategorieRepository categorieRepository;

	@RequestMapping(method=RequestMethod.GET)
	public List<Categorie> getAllCategories()
	{
		return categorieRepository.findAll();
	}

	@RequestMapping(value="/{id}",method = RequestMethod.GET)
	public Categorie getCategorieById(@PathVariable("id") Long id) {
		return categorieRepository.findById(id).get();
		}
}
```

18. Testez l'api retourant toutes les catégories :

http://localhost:8080/produits/api/cat/

19. Modifier la méthode listeCategories de la classe ProduitService :

```
listeCategories():Observable<Categorie[]>{
          return this.http.get<Categorie[]>(this.apiURL+"/cat");
        }
```

20. Modifier la méthode ngOninit() de la classe addProduitComponent

```
ngOnInit(): void {
  this.produitService.listeCategories().
  subscribe(cats => {this.categories = cats;
                     console.log(cats);
 });
 }
```

21. Modifier la méthode addProduit() de la classe addProduitComponent

```
addProduit(){
  this.newProduit.categorie = this.categories.find(cat => cat.idCat == this.newIdCat)!;
   this.produitService.ajouterProduit(this.newProduit)
                   .subscribe(prod => {
                   console.log(prod);
                   this.router.navigate(['produits']);
                   });
   }
```

22. Testez votre travail en ajoutant un nouveau produit

Modification des catégories des produits

23. Modifier la méthode ngOninit() de la classe UpdateProduitComponent

```
ngOnInit(): void {
   this.produitService.listeCategories().
   subscribe(cats => {this.categories = cats;
                      console.log(cats);
  });
   this.produitService.consulterProduit(this.activatedRoute.snapshot.param
s['id']).
   subscribe( prod =>{ this.currentProduit = prod;
                       this.updatedCatId =
this.currentProduit.categorie.idCat;
   } ) ;
   }
```

24. Modifier la méthode updateProduit() de la classe updateProduitComponent

```
updateProduit() {
  this.currentProduit.categorie = this.categories.
                find(cat => cat.idCat == this.updatedCatId)!;
       this.produitService.updateProduit(this.currentProduit).subscribe(prod => {
      this.router.navigate(['produits']); }
      );
  }
```

25. Testez votre travail en modifiant la catégorie d'un nouveau produit

Utiliser les variables d'environnement

ng generate environments

```
export const environment = {

    apiURL :'http://localhost:8080/produits/api'
};
```