Angular

Dans une application web, **vous avez besoin de communiquer** entre les Components, et avec le monde extérieur à travers les APIs

Requêtes AJAX Rappels
- Définition
- L'objet XmlHttpRequest
- Utiliser fetch (son retour de type Promise)

---

1 LES SERVICES

2 LES SUBJECTS

3 LA LIBRAIRIE HTTPCLIENT

1 POURQUOI UTILISER DES SERVICES

COMPONENT A

Imaginez que vous avez besoin d'une donnée dans un component A

1 POURQUOI UTILISER DES SERVICES

Imaginez maintenant que vous avez besoin de la même donnée dans un component B. C'est particulièrement inefficace. Vous répétez le même code.

La solution? Les services

1 POURQUOI UTILISER DES SERVICES

Le principe du service est de permettre à plusieurs components d'accéder à des données et des méthodes partagées.

2 LES SUBJECTS - distribuer l'information au moment où elle change

Le rôle du subject :

Maintenir une variable utilisable par tous les components, qui soit à la fois

➔ **une donnée modifiable** par n'importe quel component

➔ **une source de données sur laquelle tous les component peuvent s'abonner**

Un Subject RxJs permet à la fois d'envoyer des données (.next()) et de s'y abonner (.subscribe())

FAIRE DES REQUÊTES HTTP AVEC HttpClient

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

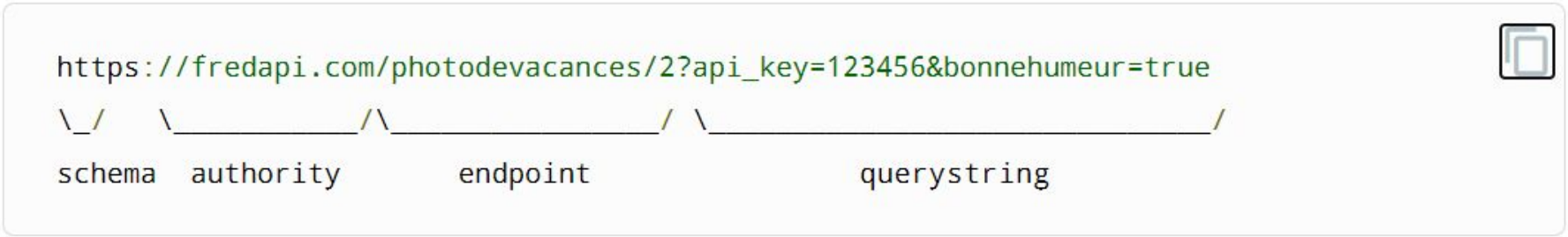
- Dans app.module.ts
  Charger HttpClientModule

- Dans un component ou un service
  Injecter HttpClient

- On peut faire des http requests
  .get(), .post(), .put(), .delete()

les méthodes de HttpClient retourne des Observables. On va pouvoir s'y abonner à l'aide de la méthode **.subscribe(data => …)** pour récupérer les données et gérer les erreurs.

LES SERVICES WEB API



Une API propose des urls comme interface d'accès à ses données. Le développeur.euse peut alors requêter des ressources, le plus souvent pour alimenter une interface web ou mobile.

La documentation d'une API respecte la structure des Urls :

- L'url de nom de domaine de l'API
- Le endpoint (la ressource à demander)
- La queryString (les options)

Exemple : https://api.themoviedb.org/3/movie/550?api_key=efdeb661aaa006b1e4f36f990a5fd8fd

3 UTILISER LA LIBRAIRIE HTTPCLIENT

COMPONENT A

```
constructor(private productService:ProductService) {}

ngOnInit() {
      this.productService.getProducts()
          .subscribe( response =>
              this.products = response
          )
}
```

ContactService

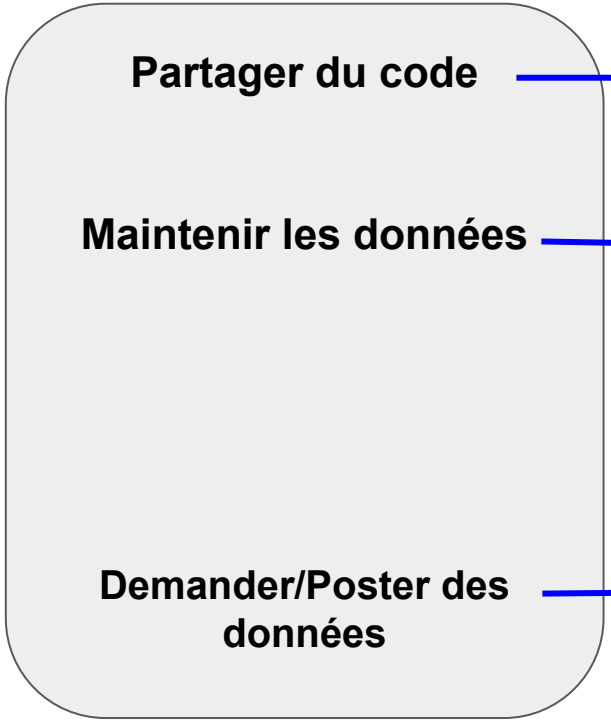
```
import {HttpClient} from '@angular/common/http';
constructor(private http:HttpClient) {}

getProducts() {
   return this.http.get(apiUrl);
}
```

Les méthodes de HttpClient (.get(), .post(), .delete(), .put())
retournent un Observable

Nos components peuvent alors s'y abonner
avec .subscribe() afin de récupérer la réponse

https://angular.io/guide/http

Conclusion : à chaque problème sa solution



Des services. Ils hébergent du code pouvant être partagé entre les components.

Des subjects / behaviorSubjects. Un type d'*Observable* (on peut envoyer une nouvelle donnée et souscrire à ce flux) qui permet de maintenir une donnée constamment à jour pour tous les components de l'application.

La class HttpClient. Elle permet d'effectuer des requêtes AJAX

https://nicolasfazio.ch/programmation/angular/angular-creer-service-reactif-observables
https://blog.bitsrc.io/3-ways-to-debounce-http-requests-in-angular-c407eb165ada