esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

Les services dans Angular

- Injection de dépendances
- Les services
- Les observables
- Le service HttpClient

Injection de dépendances

Injection des dépendances

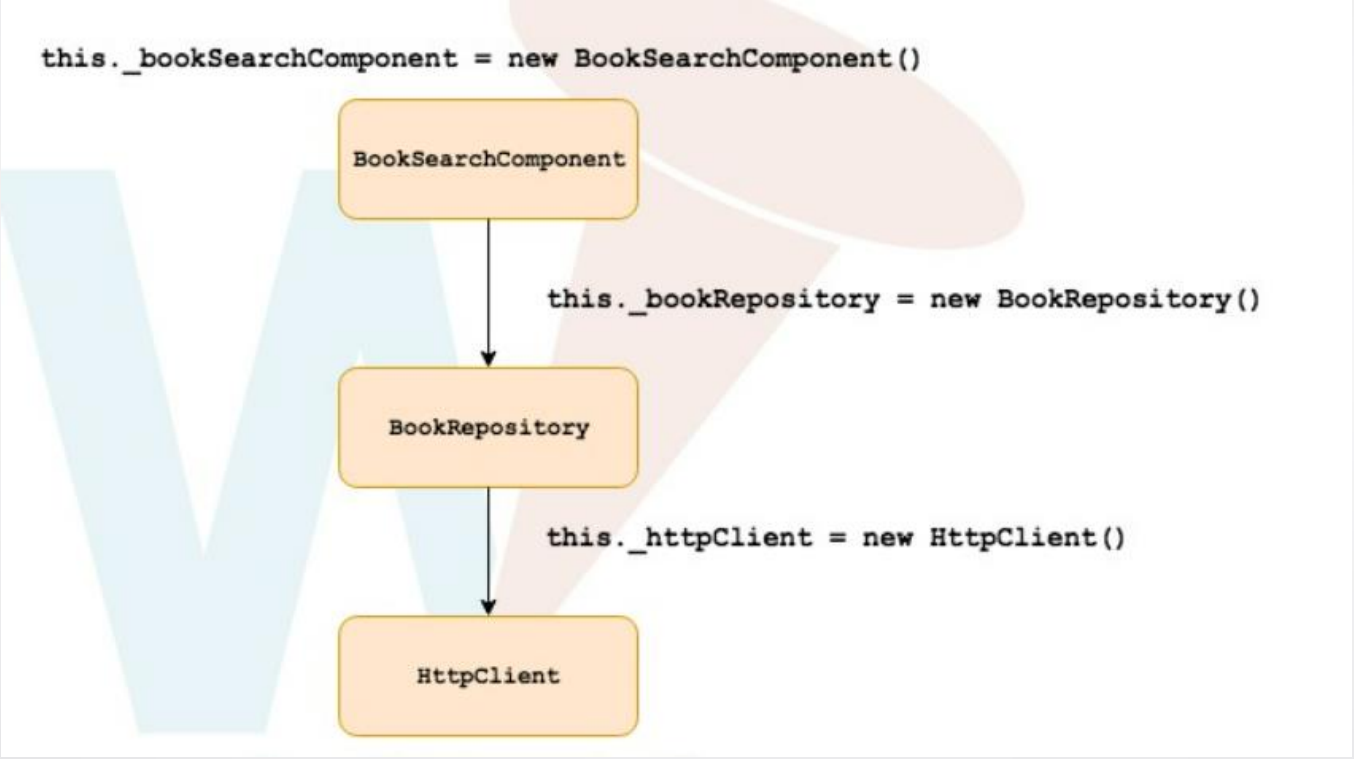
- C’est un design pattern qui consiste à séparer l’instanciation d’une dépendance et son utilisation.
- L’injector du système d’injection de dépendance permet de créer ou réutiliser une dépendance.
- Il appelle uniquement les dépendances dont l’application aura besoin lors de l’exécution..
- Les services dans la plus part du temps sont utilisées comme des singletons çad un service est créé seulement une fois pour le cycle de vie de l’application.

Injection des dépendances

Soit un composant BookSearchComponent qui utilise un service BookRepository. Ce dernier utilise à son tour un service HttpClient

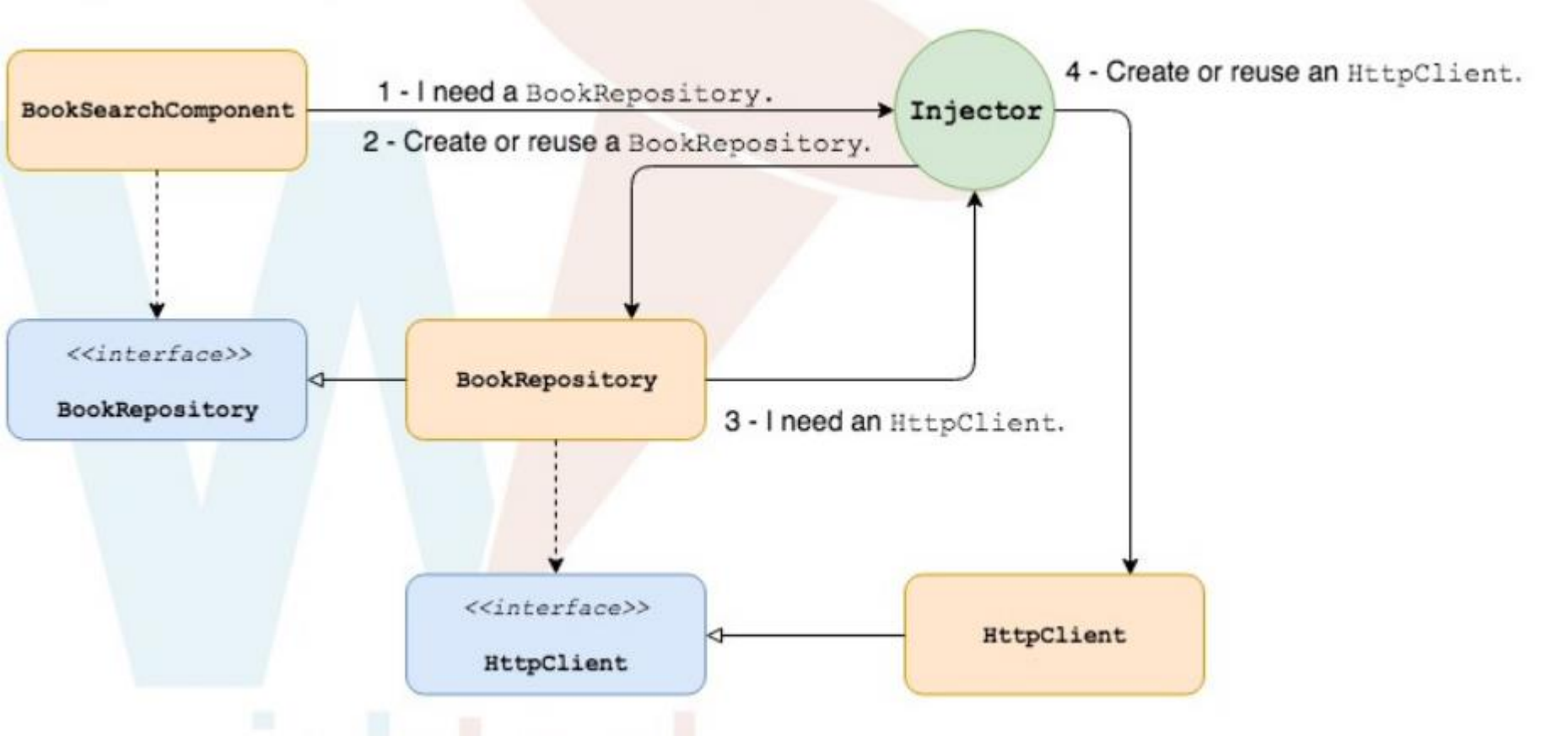
Injection des dépendances

- Sans injection de dépendances



Injection des dépendances

- Avec injection de dépendances



Les services

Service - Définition

- Un service est une ressource réutilisable et partagée entre plusieurs éléments.
- Il s’agit généralement d’une classe ayant un objectif étroit et bien défini.
- Il offre des propriétés et des fonctionnalités dont l’application a besoin

Service - Intérêts

- Réduire le couplage entre composants ou classes.
- Assurer la modularité et la réutilisabilité
- Un composant se concentre principalement sur la présentation des données et délègue l'accès aux données à aux services.
- Les services sont un excellent moyen de partager des informations entre des classes qui ne se connaissent pas

Service - Création

- Nous pouvons écrire manuellement la classe du service tout en respectant la charte associée.
- Angular CLI permet de générer un service à l'aide de la commande **ng generate service nomservice**
- Si un service est à créer dans un répertoire bien déterminé, il faut alors préciser le nom du répertoire au niveau la commande **ng generate service rep\nomservice**

Service - déclaration

- Pour déclarer qu'une classe est un service dans Angular, il suffit de créer une classe TypeScript et de la décorer avec le décorateur @Injectable().

```
@Injectable()
export class UsersService { ... }
```

Service - Injection

- Un service est injecté au niveau de la classe qu'elle va l'utiliser. Cette classe peut être un service ou un composant.
- Le code suivant permet d'injecter un service appelé DataService au niveau du composant AppComponent. Il s'agit de le définir comme argument du constructeur de la classe.

```
export class AppComponent {
constructor(private dataService:DataService) { }
}
```

Provider - Définition

Afin de pouvoir instancier un service, Angular a besoin d'un "provider" lui indiquant **comment produire l'instance de ce service**. Cela se fait généralement via la propriété

- providedIn de @Injectable
- providers de la configuration du module associé
- providers du composant

Vous devez alors enregistrer au moins un fournisseur « provider » du service à injecter.

Provider - Déclaration

Avantages de providedIn:

- Simple et rapide à implémenter
- Ne charger le code des services qu'à la première injection
- Si le service n'est jamais utilisé, le code du service sera entièrement retiré du build final de l'application

Provider - Erreur

- Au cas où aucun provider n’est enregistrée pour un service XService vous obtenez l’erreur suivante:

```
StaticInjectorError(xModule)[xxxxxxx -> xxxxxxxx]:
StaticInjectorError(Platform: core)[xxxxxx -> xxxxxxx]:
NullInjectorError: No provider for XService!
```

- Il faut alors définir le provider de XService

Services - Squelette

- **@Injectable()**: décorateur qui permet de fournir les métadonnées permettant à Angular d'injecter le service dans un composant en tant que dépendance.
- **providedIn: 'root'** : fournir le service au niveau racine. Ce provider est généré par défaut en créant un service avec l'outil CLI. Angular crée une instance unique et partagée de ce service et l'injecte dans toute classe qui le demande

```
import { Injectable } from '@angular/cor
@Injectable({
  providedIn: 'root'
})
export class UserService {

  constructor() {}

}
```

ProvidedIn

provider	description
@Injectable({ providedIn: 'root' })	Une seule instance du service disponible dans toute l'application (y compris les modules lazy loaded)
@Injectable({ providedIn: 'any' })	Une instance du service est crée **pour chaque module lazy loaded** Le reste de l'application y compris les modules qui ne sont pas lazy loaded partage l'instance disponible au niveau du module racine.
@Injectable({ providedIn: 'plateform' })	Ceci est utile si vous avez plusieurs applications anangular en cours d'exécution sur une seule page. Il s'agit d'une option utile si vous utilisez Angular Elements, où ils peuvent partager une seule instance de service entre eux

Providers

provider	description
@NgModule({ ….. providers: [SomeService] })	L'enregistrement du service dans un @NgModule le rendra disponible uniquement dans ce module (Singleton dans la portée du module) RQ: L'utilisation de cette façon avec providedIn: 'root' rend le service singleton pour le reste de l'application, tandis qu'il crée une instance distincte pour ce module
@Component({ ….. providers: [SomeService] })	L'enregistrement du service dans un @Component le rendra disponible pour ce composant avec une nouvelle instance du service pour chaque instance de composant.

Exemple d’application

- Soit un service appelé LogService

```
@Injectable({
providedIn : ‘root’
})
export class LogService {
log(msg: any) { console.log(msg); }
error(msg: any) { console.error(msg); }
warn(msg: any) { console.warn(msg); }
}
```

- Ce service offre trois méthodes : log(msg), error(msg) et warn(msg). Ces méthodes sont souvent utilisées par divers classes.

Exemple d’application

- Injecter ce service au niveau du (composant ou service ou classe) qui a besoin

```
constructor(private lservice:LogService){}
```

- Utiliser l’une ou les méthodes de ce service en fonction du besoin.

```
log(){
this.lservice.log("finish");
}
```

Les observables

Les observables - Définition

- Les observables représentent un nouveau standard pour la gestion des données asynchrones
- Les observables sont des collections paresseuses de valeurs multiples dans le temps.
- Les observables sont des collections paresseuses c'est à dire les collections ne sont envoyés que pour les inscrits.
- Les observables sont des collections de valeurs multiples dans le temps c'est-à-dire l'observable peut envoyers plusieurs valeurs au fil des temps d'une façon asynchrone.
- Les observables sont annulables (unsubscribe())

Les observables - Définition

- Ils ne sont pas une caractéristique spécifique à Angular.
- Angular utilise largement les observables dans le système d'événements et le service HTTP.
- Angular entrain d'utiliser les observables de la bibliothèque RxJS (bibliothèque Reactive X pour le langage JavaScript)

Les observables dans Angular

Angular utilise les observables comme interface pour gérer diverses opérations asynchrones courantes. Tels que:

- Les classes EventEmitter extends observable.
- Les modules Router et Forms utilisent des observables pour écouter et répondre aux événements entrés par l'utilisateur.
- Le module HttpClientModule utilise des observables pour gérer les requêtes et les réponses HTTP.

```
getPizzas(): Observable <Pizza[]> {
 return this.http.get<Pizza[]>(urlAPI) ;
}
```

Les observables - inscription

- L’inscription à un observable peut se faire avec:
- Les pipes asynchrones
- La fonction subscribe
- Une fois l’inscription à un observable est établie, ce dernier émet les collections.
- Un traitement doit être fait une fois les collections sont émises. Ce traitement est à définir à l’aide de la fonction subscribe ou la pipe.

La méthode subscribe

- La fonction subscribe() permet de s'inscrire à un observable pour récupérer les collections.
- Elle possède trois arguments (résultat=> quoi faire avec, erreur=> quoi faire, terminé=>notifier())

```
myObservable.subscribe(
x => console.log('Observer got a next value: ' + x),
err => console.error('Observer got an error: ' + err),
() => console.log('Observer got a complete notification')
);
```

Le service HttpClient

HttpClientModule et HttpClient

- La plupart des applications front-end communiquent avec des services principaux via le protocole HTTP.
- Angular propose un service appelé HttpClient qui offre un ensemble de fonctions à utiliser pour récupérer ou envoyer des données
- Pour l'utiliser il faut importer le module **HttpClientModule** dans AppModule et importer le service **HttpClient** au niveau de la classe dans laquelle a été injecté. Les deux à importer à partir du package **@angular/common/http**

Les méthodes Http

- **GET**
  - La méthode qui permet de demander une ressource.
- **HEAD**
  - Cette méthode ne demande que des informations sur la ressource, sans demander la ressource elle-même.
- **POST**
  - Cette méthode permet d'ajouter une ressource sur le serveur.
- **OPTIONS**
  - Cette méthode permet d'obtenir les options de communication d'une ressource ou du serveur en général.
- **PUT**
  - Cette méthode doit être utilisée lorsqu'une requête modifie la ressource.
- **DELETE**
  - Cette méthode permet de supprimer une ressource du serveur.

Exemple

- Exemple d'utilisation de la méthode GET du service HttpClient:

1-Injecter le service HttpClient au niveau du service:

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

2- Définir la méthode qui appelle la méthode get()

```
getFilmList():Observable<film[]>{
  return this.http.get<film[]>(this.url);
}
```

3- Injecter le service FilmsService au niveau du composant:

```
constructor(private service:FilmsService) {}
```

4- Appeler la méthode getFilmList() et récupérer la liste des films

```
getFilms() {
this.service.getFilmList().subscribe(
(data:film[])=>{this.list = data});
}
```

Néthographie

- https://angular.io/guide/
- https://guide-angular.wishtack.io/angular

Merci de votre attention