Cours Angular : Services et Injection de Dépendances

1. Introduction

Les **services** et l'**injection de dépendances (DI)** sont au cœur de l'architecture Angular.

Objectifs : - Comprendre le rôle des services - Créer et utiliser un service - Comprendre l'injection de dépendances - Gérer le scope et la réutilisabilité - Appliquer les bonnes pratiques en entreprise

---

2. Qu'est-ce qu'un service ?

Un service est une **classe TypeScript** qui contient : - Des **logiques métier** - Des **fonctions réutilisables** - Des **données partagées** entre composants

Exemple classique : - Gestion des utilisateurs - Appels HTTP - Authentification

---

3. Création d'un service

3.1 Commande CLI

```
ng generate service user
```

ou

```
ng g s user
```

3.2 Structure d'un service

```
import { Injectable } from '@angular/core';

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

  users = ['Ali', 'Fatou', 'Jean'];

  constructor() {}
```

```
  getUsers() {
    return this.users;
  }
}
```

- `@Injectable({ providedIn: 'root' })` : rend le service **singleton global**.

---

4. Injection de dépendances (DI)

4.1 Principe

DI = fournir **automatiquement** aux composants ou services les instances nécessaires.

4.2 Exemple dans un composant

```
import { Component } from '@angular/core';
import { UserService } from './user.service';

@Component({ selector: 'app-user', templateUrl: './user.component.html' })
export class UserComponent {
  users: string[] = [];

  constructor(private userService: UserService) {
    this.users = this.userService.getUsers();
  }
}
```

👉Angular crée automatiquement une instance de `UserService` et l'injecte.

---

5. Scope des services

- `providedIn: 'root'` → Singleton global
- `providedIn: 'platform'` → Singleton par plateforme
- `providedIn: 'any'` → instance par module lazy loaded
- Déclaration dans `providers` d'un module ou composant → instance locale

---

6. Services et HTTP

6.1 Exemple avec HttpClient

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';

@Injectable({ providedIn: 'root' })
```

```
export class ApiService {
  constructor(private http: HttpClient) {}

  getUsers() {
    return this.http.get<User[]>('/api/users');
  }
}
```

6.2 Utilisation dans un composant

```
users$ = this.apiService.getUsers();
```

```
<ul>
  <li *ngFor="let user of users$ | async">{{ user.name }}</li>
</ul>
```

---

7. Services partagés et réutilisables

- Stockage de l'état global (AuthService, CartService)
- Logique métier centralisée
- Communication entre composants (via Subject / BehaviorSubject)

7.1 Exemple Subject pour partage de données

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

@Injectable({ providedIn: 'root' })
export class MessageService {
  private messageSource = new BehaviorSubject<string>('');
  currentMessage$ = this.messageSource.asObservable();

  changeMessage(message: string) {
    this.messageSource.next(message);
  }
}
```

```
constructor(private messageService: MessageService) {
  this.messageService.currentMessage$.subscribe(msg => console.log(msg));
}
```

---

8. Lazy services et modules

- Un service dans un module lazy loaded est **instancié uniquement pour ce module**.
- Permet de **scinder le code et la mémoire**.

---

9. Bonnes pratiques

- Toujours utiliser `providedIn: 'root'` pour les services globaux
- Ne pas mélanger logique UI et logique métier
- Services = fonctions, données, API
- Utiliser RxJS pour le partage de données réactif
- Limiter l'injection directe dans le template (préférer le composant)

---

10. Résumé

- Services = logique réutilisable
- DI = injection automatique d'instances
- Singleton vs instance locale selon le scope
- Services combinés avec RxJS = partage réactif des données
- Angular encourage une architecture **modulaire, testable et maintenable**

👉Prochaine étape recommandée : **Services avancés + intercepteurs HTTP + state management**