

PRÉSENTATION DES SERVICES (ANGULAR)

INTRODUCTION AUX SERVICES

Les **services** sont des éléments clés dans le développement d'applications **Angular** pour assurer la **modularité**, la **réutilisabilité** et la **séparation des préoccupations**.

DÉFINITION D'UN SERVICE

Un service est une **classe** avec un **objectif spécifique**, généralement utilisée pour organiser la **logique de l'application** en dehors des **composants**.

POURQUOI UTILISER DES SERVICES ?

- Réutilisation du code
- Modularité
- Séparation des préoccupations
- Facilite les tests

COHÉRENCE AVEC LES PRINCIPES DE LA PROGRAMMATION MODULAIRE

Les **services** permettent de suivre les principes de la **programmation modulaire** en regroupant les fonctionnalités relatives en classes indépendantes.

OÙ PLACER LES SERVICES DANS L'ARCHITECTURE D'UNE APPLICATION ANGULAR

Les services doivent être placés dans des **dossiers spécifiques** pour les organiser et faciliter leur maintenance. Les structures recommandées incluent :

- Un dossier `services` au niveau **racine** de l'application
- Un dossier `services` dans chaque **module fonctionnel**.

CRÉATION D'UN SERVICE

CRÉATION D'UN SERVICE

Les services peuvent être créés à l'aide de l'**Angular CLI** ou **manuellement** en écrivant du code.

- Angular CLI :

```
ng generate service nom-du-service
```

- Manuellement :
 1. Créez un fichier avec l'extension `.service.ts`
 2. Importez les éléments nécessaires et définissez une classe pour le service
 3. N'oubliez pas de décorer la classe avec `@Injectable`

UTILISATION DE L'OUTIL ANGULAR CLI

COMMANDES POUR CRÉER UN SERVICE

Pour créer un service avec **Angular CLI**, utilisez cette commande :

```
ng generate service my-service
```

ou raccourci :

```
ng g s my-service
```

STRUCTURE DU FICHIER GÉNÉRÉ

Angular CLI génère deux fichiers :

Fichier	Contenu
<code>my-service.service.ts</code>	Contient la classe de service
<code>my-service.service.spec.ts</code>	Contient le fichier de test unitaire

CRÉATION MANUELLE DU SERVICE

IMPORTATION DES DÉPENDANCES NÉCESSAIRES

Pour créer un service manuellement, commencez par importer **Injectable** :

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

ANNOTATION @Injectable

Le décorateur `@Injectable()` indique qu'une classe peut être **injectée** comme dépendance :

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

DÉCLARATION DE LA CLASSE

Définissez ensuite la classe du **service** :

```
export class MyService {  
  // Votre code ici  
}
```

INJECTION DE DÉPENDANCES

COMPRÉHENSION DU MÉCANISME D'INJECTION DE DÉPENDANCES

L'**injection de dépendances** est un mécanisme utilisé pour **résoudre** et gérer les **dépendances** entre les classes.

Avantages

Meilleure modularité

Encourager la réutilisabilité du code

Faciliter les tests

Exemples d'utilisation

Services Angular

Fournisseurs de données

Mocks pour les tests

RÉSOLUTION DES DÉPENDANCES

Angular résout les **dépendances** en créant et en **injectant** les instances de service appropriées.

SCOPES DES INSTANCES DE SERVICE

Le **scope** détermine la portée de l'instance de service. Par défaut, les services sont **singletons**, mais peuvent être configurés pour avoir des portées différentes.

Types de portée	Description
Singleton	Une seule instance du service est créée et partagée entre tous les utilisateurs.
Transitif	Une nouvelle instance du service est créée à chaque utilisation.

UTILISATION DES SERVICES DANS LES COMPOSANTS

SYNTAXE POUR INJECTER UN SERVICE

Pour injecter un service, ajoutez-le comme argument du **constructeur** d'un composant.

```
constructor(private monService: MonService) {}
```

ACCÈS AUX MÉTHODES ET PROPRIÉTÉS DU SERVICE

Accédez aux méthodes et propriétés du **service injecté** en utilisant la syntaxe suivante :

```
this.monService.maMethode();
```

UTILISATION DES SERVICES DANS D'AUTRES SERVICES

SYNTAXE POUR INJECTER UN SERVICE

Pour injecter un **service** dans un autre service, ajoutez-le dans le **constructeur**.

```
constructor(private autreService: AutreService) {}
```


GESTION DES DÉPENDANCES CIRCULAIRES

Pour gérer les **dépendances circulaires**, utilisez l'annotation `@Inject` et le type `**forwardRef**`.

```
constructor(@Inject(forwardRef(() => AutreService)) private autreService: AutreService) {}
```

FOURNISSEURS DE SERVICES

DÉCLARATION DE FOURNISSEURS

REGISTERING DANS APPMODULE

Pour déclarer un service comme **fournisseur**, ajoutez-le dans le tableau `providers` de `**@NgModule**` dans `AppModule` :

```
import { MonService } from './mon-service.service';

@NgModule({
  providers: [MonService]
})
export class AppModule { }
```

REGISTERING DANS UN MODULE SPÉCIFIQUE

Il est également possible d'enregistrer un **service** dans un **module spécifique**:

```
import { MonService } from './mon-service.service';

@NgModule({
  providers: [MonService]
})
export class MonModule { }
```

FOURNISSEURS DE PORTÉE

PROVIDEDIN: 'ROOT'

Pour rendre un service **disponible dans toute l'application**, utilisez l'option `providedIn` avec la valeur `'root'`:

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

PROVIDERS ARRAY D'UN MODULE/COMPOSANT

Pour limiter la **portée** d'un service, ajoutez-le au tableau `providers` d'un **module** ou d'un **composant** spécifique.

```
@Component ({  
  providers: [MonService]  
})  
export class MonComposant { }
```

L'**instance** du service sera propre à ce composant et à tous ses **enfants**. Chaque composant créera une nouvelle instance du service.

COMMUNICATION ENTRE COMPOSANTS VIA SERVICES

LES OBSERVABLES

COMPRÉHENSION DES OBSERVABLES

Les **observables** sont des objets qui émettent des données sur le temps. Ils sont utilisés pour traiter les flux de données **asynchrones** et sont au cœur de la programmation **réactive** d'Angular.

CRÉER ET S'ABONNER AUX OBSERVABLES

Pour créer un observable:

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

const monObservable = new Observable(observer => {
  observer.next('Données 1');
  observer.next('Données 2');
});
```

Pour s'abonner à un observable:

```
monObservable.subscribe(
  data => console.log(data),
  error => console.error(error),
  () => console.log('Terminé')
);
```

CHAÎNAGE DES OPÉRATEURS ET GESTION DES ERREURS

Les **opérateurs** permettent de manipuler les données émises par les **observables**. Pour utiliser les opérateurs, il faut utiliser la méthode `pipe()`:

```
import { filter, map } from 'rxjs/operators';

monObservable.pipe(
  filter(data => data % 2 === 0),
  map(data => data * 2)
).subscribe(...)
```

IMPLÉMENTATION D'UN ÉMETTEUR/RÉCEPTEUR D'ÉVÉNEMENTS

UTILISATION D'UN SERVICE POUR TRANSMETTRE LES DONNÉES

Un service peut être utilisé comme intermédiaire pour transmettre des données entre les **composants** en utilisant les **observables**.

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

@Injectable()
export class EmetteurService {
  private sujet = new Subject<any>();

  envoyerDonnees(data: any) {
    this.sujet.next(data);
  }

  recevoirDonnees(): Observable<any> {
    return this.sujet.asObservable();
  }
}
```

EXEMPLES D'UTILISATION DANS DES COMPOSANTS

Dans le composant **émetteur**:

```
constructor(private emetteurService: EmetteurService) { }  
  
envoyer() {  
  this.emetteurService.envoyerDonnees('Données envoyées');  
}
```

Dans le composant **récepteur**:

```
constructor(private emetteurService: EmetteurService) {  
  this.emetteurService.recevoirDonnees().subscribe(data => {  
    console.log(data);  
  });  
}
```