PARTIE 3: UTILISATION DE SERVICES

OBJECTIFS ?

- Pourquoi a-t-on besoin de cette notion de “service” ?
- Comment fonctionnent les services en Angular ?

POURQUOI DES SERVICES ?

- L’objectif d'un service est de contenir tout ou partie de la logique fonctionnelle d'une application.
- Ils sont là pour fournir les données aux composants qui avant tout “présentent” ces données.

Comme avec Vue.js et Pinia, il est possible d'utiliser un store pour centraliser les états partagés (ex: `NgRx`), mais il est aussi courant de les trouver dans les services (*statefull services*).

UTILISATION DE SERVICES

- Un exemple typique est d'encapsuler les interactions avec une API dans un service pour que les composants n'aient pas à se soucier des aspects liés à la communication (e.g., requêtes HTTP)
- Les services peuvent utiliser d’autres services.

CRÉER UN SERVICE (1/2)

```
ng generate service logging-service
```

```
import { Injectable } from '@angular/core';
@Injectable({
  providedIn: 'root'
})
export class LoggingService {

}
```

- Le décorateur `@Injectable` permet d'indiquer que la classe `LoggingService` peut être injectée.
- La métadonnée `providedIn` à `root` du décorateur `@Injectable` ajoute le service au registre des composants injectables n'importe où dans l'application.
- L'Injector de l'application créera une instance du service la première fois qu'il sera demandé et la transmettra ensuite à chaque composant, service ou fonction qui en fera la demande. C'est un *design pattern* **singleton**.

CRÉER UN SERVICE (2/2)

```
import { Injectable } from '@angular/core';
@Injectable({
  providedIn: 'root'
})
export class LoggingService {
  log(text: string): void{
    console.log(text)
  }
}
```

- Les méthodes / propriétés publiques de cette classe seront exposées et utilisables par les autres élements (composants/services / fonctions) de l'application, dans lesquels on aura défini ce service comme une dépendance.

L'INJECTION DE DÉPENDANCES (DEPENDENCY INJECTION - DI)?

- Lorsqu'on a besoin de faire appel à des fonctionnalités qui sont définies dans d’autres parties de l’application (un service, par exemple), on fait appel au mécanisme d'**injection de dépendances**.

UTILISER UN SERVICE

- Une fois qu'une dépendance, comme un service, est enregistrée, un composant peut la définir comme une dépendance requise.
- L'injection de dépendances est utilisée pour passer une instance d'une classe (le service) à un objet dépendant (composant, service, fonction).

```
...
class MonComposant {
  private readonly loggingService = inject(LoggingService);
  ...
}
```

Le modificateur d'accès *private* permet d'empêcher que le service soit utilisé dans le *template*.

EXERCICE (1/3)

- Créez un service MessagesService avec la commande :

```
ng g s messages-service
```

- Vous commencez par déplacer la déclaration du tableau `tabMessages` au niveau du service `MessagesService`

```
// signal pour l'état global
  private messages = signal<MessageModel[]>([{
    id: 1,
    text: 'Congratulations! Your app is running. 🎉',
    category: 'validation'}, ...
  ]);

  // signal en lecture seule pour les composants
  readonly tabMessages = this.messages.asReadonly();
```

- `tabMessages` est la version exposée aux composants de l'application. En lecture seule, il empêche des modifications directes par les composants.

EXERCICE (2/3)

- Implémentez ensuite
  - une méthode `fillWithExamples`qui remplace la valeur associée à l'état global `messages` avec le tableau de nos trois exemples de messages.
  - une méthode `delete` pour supprimer un message à partir de son `id`
  - une autre méthode `add` qui à partir d'un texte requis et d'une catégorie optionnelle ajoute un message au tableau.

Les identifiants doivent commencer à 1 et augmenter de 1 à chaque nouvelle insertion.

EXERCICE (3/3)

- Injectez le service `MessagesService` dans le composants App et créer une propriété dans ce composant qui récupèrera le signal du service.
- Utilisez le service et vérifier que votre application fonctionne parfaitement
- Remplacez les fichiers de tests du composant App et Alert, ainsi que celui du service par ceux fournis dans le zip
- Lancez ces tests unitaires => `ng test`

ALLER PLUS LOIN: PLUSIEURS INSTANCE D'UN SERVICE

Il arrive qu'un service soit plutôt "enregistré" comme tel au niveau du lanceur de l'application.

```
import { Injectable } from '@angular/core';
@Injectable() // sans providedIn: 'root'
export class MessageService {
  ...
}
```

```
// main.ts
  bootstrapApplication(AppComponent, {
      providers: [MessageService]
    });
```

- C'est cette méthode qui sera utilisée dans les cas, rares, où on a besoin de plusieurs instances du service dans l'application.
- Ce tableau de *providers* peut faire partie des métadonnées de `@Component` si on veut limiter la portée d'une instance à ce