Services

les services sont des fichiers permettant de centraliser des actions qui peuvent se retrouver dans plusieurs components afin de ne pas avoir à se répéter et retrouver son code plus facilement.

Création d'un service

Créons un nouveau service :

```
# Je génère mon service dans recette et non plus dans app.
ng generate service recette/recette --dry-run
# L'option --dry-run permet de simuler la commande
ng generate service recette/recette
# Si tout va bien, on relance la commande normal
```

Notre service a bien été généré. On remarquera que dans ce fichier "**recette.service.ts**" le décorateur ne s'appelle pas "**service**" mais "**Injectable**", il permet d'indiquer à Angular que notre service peut se voir injecter à d'autres endroits.

La propriété "**providedIn**" indique où est ce que l'instance de notre service sera disponible. Par défaut on verra "**root**" indiquant que la même instance de notre service sera disponible partout. "**Injectable**" se trouve déjà par défaut dans les pipes, components ou directive.

Créons trois méthodes dans dans notre service et supprimons ce qui est inutile :

```
getRecetteList():Recette[]
{
    return RECETTES
}
// accompagné des imports correspondant
getRecetteById(recetteId: number): Recette|undefined
{
    return RECETTES.find(rec=> rec.id === recetteId);
}
getRecetteTypeList():string[]
{
    return Object.values(Types);
}
// on importe l'enum "Types" pour transfomer ses valeurs en tableau.
```

Avoir ces fonctions va me permettre de retirer de la logique répétitive de mes composants afin de tout centraliser dans un seul fichier.

Utilisation d'un service

Rendons nous dans "**liste-recette.component.ts**". Les services sont automatiquement instancié par Angular, nous n'avons donc aucune raison d'utiliser le mot clef "**new**", on va juste avoir besoin de l'importer de le déclarer dans notre classe :

```
import { RecetteService } from '../recette.service';
/* ... */
constructor(
    private router: Router,
    private recetteService: RecetteService
){}
```

Ensuite je vais retirer la constante "**RECETTES**" des imports et changer la ligne suivante :

```
receteList: Recette[] = [];
// Puis dans "ngOnInit"
this.receteList = this.recetteService.getRecetteList();
```

L'avantage de faire comme ceci, vient du fait que si je change la façon dont je récupère mes recettes, Je n'aurais pas à fouiller tout mes fichiers pour changer les différentes méthodes mais juste à changer la méthode de mon service.

On va maintenant faire de même dans "**détail-recette.component.ts**" :

```
/*
    J'ajoute le service au constructor.
    Je déclare ma propriété.
*/
receteList: Recette[] = RECETTES;
// Puis dans "ngOnInit" je remplace :
this.recette = this.receteList.find(rec=> rec.id === receteId);
// par :
this.recette = this.receteService.getRecetteById(receteId);
```

Toute la logique de récupération des données est géré par mon service et non plus par mes composants, si je veux la modifier je n'aurais qu'un seul fichier à modifier.

Un peu de théorie

Le modèle d'injection de dépendance permet aux classes de recevoir des dépendances depuis une source externe plutôt que de les créer elle même. C'est un "**injecteur**" qui va transmettre aux classes qui en ont besoin les services demandé. On retrouvera la création de cet injecteur dans le fichier "**main.ts**":

```
platformBrowserDynamic().bootstrapModule(AppModule)
```

On a pas besoin de s'occuper des "**injecteurs**" mais par contre on peut modifier les "**fournisseurs**".

Prenons notre "**recette.service.ts**" il n'a pas besoin d'être disponible dans toute l'application, seulement dans le module recette.

Dirigeons nous donc vers "**recette.service.ts**" et retirons l'option suivante du décorateur :

```
{ providedIn: 'root' }
// on gardera le décorateur vide
```

Puis allons dans "**recette.module.ts**" et ajoutons à la suite de "**import**" dans le décorateur :

```
providers: [RecetteService]
```

Notre application fonctionne toujours aussi bien, mais maintenant travaille légèrement mieux en ayant le service recette disponible uniquement dans le module recette.

> C'est bien plus rare de le faire mais on peut aussi insérer cette dernière ligne dans un composant directement Cela pourrait poser problème car l'instance sera unique au composant, mais c'est possible.