Les modules

Jusqu'ici nous ne possédons qu'un seul module, celui par défaut, "**app.module.ts**". C'est suffisant pour une petite application mais si vous souhaitez voir plus grand, vous allez finir par vous perdre dans tous vos fichiers.

Une bonne application Angular est modulaire, on parle souvent de module de fonctionnalités, car on va créer un module pour chaque fonctionnalité de notre application.

un module est une classe avec un décorateur "**@ngModule**", ce décorateur contient les propriétés :

- imports qui permet d'importer d'autres modules.
- exports qui permet d'exporter des classes du modules.
- declarations qui contient tous les components, directive et pipe du module
- bootstrap qui ne concerne que le module racine.

Attention les modules qui sont importé et exporté d'Angular ne sont pas les mêmes que les modules JS, bien que les deux soit complémentaires.

Maintenant créons un nouveau module :

```
ng generate module recette
```

Angular/cli nous a bien créé un nouveau dossier contenant "**recette.module.ts**".

Et bien on va profiter de ce nouveau dossier pour faire du rangement et venir déplacer tout ce qui concerne nos recettes dedans :

- detail-recette/
- liste-recette/
- Recette.ts
- RecetteList.ts
- border-card.directive.ts
- type-color.pipe.ts

Et les possibles fichiers "**.specs.ts**".

Avec tout ces déplacements on va se retrouver avec beaucoup d'imports qui peuvent ne plus être bon. Corrigeons cela de ce pas.

Ouvrons maintenant "**recette.module.ts**" On trouvera par défaut "**ngModule**" évidement puisque c'est un module,mais aussi "**commonModule**" qui sera utile dans n'importe quel module puisque dedans se trouve par exemple les "**directives structurels**" que sont "**ngFor**" et "**ngIf**".

Je vais ajouter à mes déclarations :

- ListeRecetteComponent,
- DetailRecetteComponent,

- BorderCardDirective,
- TypeColorPipe

et faire les imports qui correspondent :

```
import { ListeRecetteComponent } from './liste-recette/liste-recette.component';
import { DetailRecetteComponent } from './detail-recette/detail-
recette.component';
import { BorderCardDirective } from './border-card.directive';
import { TypeColorPipe } from './type-color.pipe';
```

Pour que Angular arrête de provoquer une erreur, je pourrais retirer de mes composants, directives et pipes tous les "imports" et les "standalone". Effectivement, maintenant ils ne sont pas "standalone" mais une partie d'un module. Et c'est le module qui s'occupe de faire les imports pour tout ses composants, plus besoin de refaire chaque import dans chaque composant.

Il ne me reste plus qu'à importer mon module, pour cela deux solutions, soit je l'importe directement dans mon **app.component.ts** soit via mes routes.

Mon module peut avoir ses propres routes ne dépendant pas du reste de l'application, Créons un fichier **recette.routes.ts** :

```
import { Routes } from "@angular/router";
import { ListeRecetteComponent } from "./liste-recette/liste-recette.component";
import { DetailRecetteComponent } from "./detail-recette/detail-
recette.component";

export const recetteRoutes: Routes = [
        {path: "", component: ListeRecetteComponent},
        {path: ":id", component: DetailRecetteComponent}
    ];
/*
    On notera la disparition du mot "recettes" de nos chemins, nous allons voir
pourquoi sous peu.
    Puis dans les imports du fichier recette.module.ts :
*/
RouterModule.forChild(recetteRoutes)
```

On pensera bien à supprimer les routes en doublon dans "**app.routes.ts**" et on ajoutera cela à nos routes :

```
/*
    Avec cette route, on importe les routes de notre module recette, et nous les
faisons précéder du chemin "recettes"
    Nous avons donc de nouveau "recettes" et "recettes/:id" comme chemin
*/
{path: "recettes", loadChildren:
()=>import("./recette/recette.module").then(mod=>mod.RecetteModule)},
```

Puis on ira supprimer tout ce qui concerne les recettes dans "**app.component.ts**"

Une fois tout corrigé, relancer ng serve peut être utile pour être sûr qu'il a bien prit en compte les changements.

Bien sûr dans le cas d'une application classique, on réfléchi avant de coder à la structure et aux modules que comportera notre application. Nous n'aurons pas besoin de tout déplacer comme cela.