Les routes

Avec Angular nous somme sur une SPA (Single Page Application) donc nous n'allons pas vraiment changer de page mais simplement changer de composant. Cela dit Angular simulera le changement de page pour le navigateur, si on appui sur la bouton précédent, cela sera bien prit en compte.

On va maintenant créer deux composants, un qui affichera la liste de nos recettes et l'autre qui donnera le détail de l'une de ces recettes.

```
ng generate component liste-recette
ng generate component detail-recette
```

Pour chacun d'entre eux, **Angular/cli** a créé un fichier html, un ts et un css (et un spec.ts si activé). Il a aussi modifié le "**app.module.ts**" pour les importer et les ajouter aux declarations. On notera que comme cela commencerait à faire de nombreux fichiers où l'on pourrait se perdre. Angular/cli a créé ces fichiers dans des dossiers à part.

Création des routes

Maintenant que nous avons nos nouveaux composants, il va falloir les relier à des routes. Pour cela ouvrons "**app-routing.module.ts**" et intéressons nous à la ligne suivante :

```
const routes: Routes = [];
```

Dans ce tableau nous allons ajouter des objets pour chaque routes que l'on souhaite gérer.

```
{path: "recettes", component: ListeRecetteComponent},
{path: "recette/:id", component: DetailRecetteComponent},
{path: "", redirectTo: "recettes", pathMatch: "full"}
```

Une route classique contient une propriété "**path**" qui sera l'URL de notre route. Et une propriété "**component**" pour indiquer quel composant charger avec cet url. On notera la seconde route dont le "**path**" contient "**:id**", cela indique à Angular que cette seconde partie de l'adresse sera variable. Actuellement pour Angular ce chemin pourrait être "*recette/2*" ou "*recette/truc*" peu importe. On voit que mon troisième cas n'a pas de "**component**" mais un "**redirectTo**". C'est simplement une redirection vers une autre route. Ensuite l'option "**pathMatch**" indique que le chemin doit être exactement celui de la propriété "path". prenons ce cas :

- route 1 = "bidule";
- route 2 = "bidule/truc"

Si on ne met pas "full" il pourrait considérer qu'il peut aller sur la route 1 au lieu de la 2 puisque commençant par la même chose.

Revenons au html de notre composant racine, "**app.component.html**" Si vous vous souvenez, il y a une balise étrange que nous n'avons jusqu'ici pas touché.

```
<router-outlet></router-outlet>
```

Cette balise spécifique accueillera le contenu de toute nos routes. à chaque fois que l'on changera de page, ce sera le contenu html de cette balise qui changera. Cela crée un lien "*parent - enfant*" entre notre composant racine et nos composants liste et détail.

Maintenant si on teste nos url, on verra écrit "*nomDuComposant works!*"

vider le composant racine

Nos routes fonctionnent mais du coup, tout le fonctionnent de notre liste aurait plus sa place dans le composant "**liste-recette**" que dans le composant racine. Commençons par couper tout le html sauf le titre de "**app.component.html**" et plaçons le dans "**liste-recette.component.html**" Pareillement je vais venir récupérer toute la logique qui se trouvait dans "**app.component.ts**" et la coller dans "**liste-recette.component.ts**". Il ne faudra pas oublier d'importer les classes utile dans ce dernier et supprimer ce qui est devenu inutile dans le composant racine.

Si on ouvre notre page d'**accueil** ou "**/recettes**" on verra aussi que notre css ne fonctionne plus. Pourquoi? Car Angular utilise un fonctionnement proche du shadow DOM ce qui fait que chaque composant à son propre fichier CSS qui n'impactera pas les autres composants. Mais bien évidement, si on a du css qui doit s'appliquer à absolument tout notre site, il y a le fichier "**src/styles.css**" qui est hors de notre dossier "**app**" qui sert de css "*global*". Il impactera alors tout. Dans notre cas, nous nous contenteront de couper le css de "**app.component.css**" et de le coller dans "**liste-recette.component.css**"

Maintenant si nous changeons de page avec par exemple "**/recette/2**" notre liste disparaîtra pour afficher "*detail-recette works!*"

Détail d'une recette

Pour faire fonctionner cette route, la première chose à faire c'est de récupérer le paramètre variable de la route. Pour cela on va importer la classe ActivatedRoute venant du router d'Angular dans notre fichier "**detail-recette.component.ts**" :

```
import { ActivatedRoute} from '@angular/router';
/* Dans la classe : */
constructor(private route: ActivatedRoute) {}
```

On va ensuite pouvoir récupérer le paramètre de notre route dans "**ngOnInit()**":

```
// Pensez aussi à implémenter l'interface.
ngOnInit()
{
    const recetteId: number =
```

```
parseInt(this.route.snapshot.paramMap.get("id")??"");
    /*
        la propriété snapshot nous retourne un instantané de la route actuelle
        la propriété paramMap nous retourne un objet contenant tous nos paramètres
de route
        la méthode "get" récupère le paramètre donné en argument, c'est le nom que
l'on a défini dans le router
    */
}
```

Je fais passer le tout en "**parseInt**" car on souhaite récupérer un nombre, mais typescript n'est pas content car "**parseInt**" n'accepte que les string et "**get**" peut nous retourner "**null**" alors je lui dit "**??**" pour donner à "**parseInt**" une chaîne de caractère vide si c'est "**null**";

Il nous faudra aussi ajouter deux propriétés à notre classe et importer ce qui correspond :

```
import { Recette } from '../Recette';
import { RECETTES } from '../RecetteList';
/* Dans la classe : */
receteList: Recette[] = RECETTES;
recette: Recette|undefined;
```

Et on va maintenant pouvoir réutiliser une ligne de code faite dans notre méthode "**selectRecette**" qui est dans notre "**ListeRecetteComponent**" et l'ajouter à notre "**ngOnInit()**" :

```
this.recette = this.receteList.find(rec=> rec.id === receteId);
```

Maintenant on peut tester notre adresse en faisant un "**console.log**", elle affichera notre recette si l'id correspond on à l'une d'entre elles, sinon on a undefined;

Exercice 4

Consigne dans le fichier "**exercice-4.md**.

Correction

Voir les fichiers "**exercice-4.html**" et "**exercice-4.scss**"

Navigation

Allons dans notre composant racine "**app.component.html**" et ajoutons une petite barre de navigation

```
<nav><a href="#" class="logo">{{title}}</a></nav>
<!-- Pour l'instant c'est un lien banale, qui n'a rien d'une SPA -->
```

Avec un css qui correspond :

```
nav{
    background-color: steelblue;
    padding: 1rem;
    text-align: center;
    .logo{
        font-size: 2.5rem;
        color: black;
        text-transform: capitalize;
    }
}
```

Ensuite il nous faudrait pouvoir naviguer entre nos pages, allons dans "**detail-recette.component.html**" et ajoutons la ligne suivante :

```
<button (click)="goToRecetteList()">Retour</button>
```

Avec un peu de css :

```
button{
    float: right;
    margin: 1rem;
    padding: 1rem;
    font-size: 2rem;
}
```

Puis allons ajouter cette méthode dans le fichier "**.ts**" correspondant.

```
goToRecetteList(){}
```

Il nous faudra aussi utiliser le router de Angular que l'on va ajouter au "**constructor**".

```
constructor(private route: ActivatedRoute, private router: Router) { }
```

Enfin on va ajouter à "**goToRecetteList**" la ligne suivante :

```
this.router.navigate(["/recettes"]);
// Comme son nom l'indique, la méthode navigate permet la navigation dans notre
application
// ! Attention, elle prend bien un tableau et non directement un string, on verra
après pourquoi
```

Maintenant dirigeons nous vers "**liste-recette.component.ts**" et ajoutons le router au constructeur là aussi. Puis créons une méthode "**goToRecette**" qui prendra une recette en paramètre.

```
constructor(private router: Router){}
goToRecette(recette: Recette){}
```

Nous allons appeler la méthode navigate mais avec une subtilité.

```
this.router.navigate(["/recette", recette.id]);
// On place en second élément de notre tableau, le paramètre de notre chemin
```

Il ne nous reste plus qu'à ajouter un évènement sur chacune des cartes de nos recettes, pour cela je vais juste ajouter :

```
(click)="goToRecette(rec)"
```

là où il y a mon "***ngFor**" sur l'élément que je souhaite cliquable, dans mon cas, la carte en entier.

Gérer les 404

Pour gérer nos 404, on va déjà créer un nouveau composant qui tiendra ce rôle:

```
ng generate component page-not-found
```

Ensuite on va ajouter à nos routes :

```
{path: "**", component: PageNotFoundComponent}
```

Attention à bien mettre cette route en dernier ! Car Angular lit les routes dans l'ordre. Si une route correspond, il retourne celle ci sans vérifier les autres. Or, ici on voit que j'ai mit "**" en path, ce symbole signifie "Toute les routes", donc peu importe ce que l'utilisateur met comme URL, cela va me retourner notre 404 si aucune des précédentes ne correspond.

Ensuite dans notre "**page-not-found.component.ts**" *Je peux retirer le "constructor" et le "ngOnInit" si présent. On ne les utilisera pas*. Enfin rendons nous dans "**page-not-found.component.scss**" et ajoutons quelques lignes.

```
.notFound{
    width: 80%;
```

```
    margin: 2rem auto;
    text-align: center;
    *{
        margin: 1rem;
    }
    a{
        display: block;
        font-size: 2rem;
    }
}
```

Puis côté html dans "**page-not-found.component.html**" :

```
<div class='notFound'>
    <h1>Cette page n'existe pas !</h1>
    <p>
    Cela dit, vous pouvez retourner à la liste en cliquant ici :
    <a routerLink='/recettes'>Voir plus de recette !</a>
    </p>
</div>
```

Vous remarquerez sur notre lien l'attribut "**routerLink**". Il permet aussi de faire un lien. De façon plus simple, si vous n'avez pas une grosse logique à mettre en place pour définir votre lien. En arrière plan, cette "**directive d'attribut**" se contente d'utiliser elle aussi la méthode "**navigate**" du routeur. Donc vous pouvez utiliser celle qui vous plaît.

Je vais en profiter pour changer ma navigation dans mon composant racine :

```
<a routerLink="/" class="logo">{{title}}</a>
```