Le routing dans Angular

Framework Front-end

Table des matières

1. On ne faisait pas une SPA à la base ?

Il est vrai de dire qu'Angular est à la base prévu pour fournir des **Single Page Applications**, donc normalement on ne devrait pas avoir à se poser la question des URLs du site.

Cependant, il peut arriver que nous ayons besoin d'afficher des contenus supplémentaires sur différentes « pages », afin par exemple de ne pas encombrer la page d'accueil.

C'est là qu'arrive le **routing** : nous allons dire à Angular quels composants charger en fonction de la route qu'il va trouver dans l'URL, et lui préciser dans quel emplacement charger le contenu.

2. RouterOutlet et la balise <router-outlet/>

RouterOutlet est une classe qui va devoir être incluse dans notre projet. Elle va aller de paire avec une balise <router-outlet/> dans laquelle s'inscrira le contenu que le Router va aller chercher.

On va commencer par imaginer notre projet comme ceci :

- Un composant racine qui inclut un composant Menu, ainsi que plusieurs pages sous le menu
- Un composant Menu, inclus dans le composant Racine,
- Un composant Homepage, qui sera affiché sous le menu par défaut
- Un composant QuiSommesNous, qui ne sera pas affiché par défaut mais que devra l'être quand on clique sur le bon lien
- Un composant Contact, qui lui aussi sera affiché seulement quand il est appelé

Commençons par inclure la classe RouterOutlet dans notre composant racine, dans **app.ts**

```
import { Component, signal } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { Menu } from "./menu/menu";

@Component({
  selector: 'app-root',
  imports: [RouterOutlet, Menu],
  templateUrl: './app.html',
  styleUrl: './app.scss'
})
export class App {
  protected readonly title = signal('website');
}
```

Puis dans le template de notre composant racine, nous allons prévoir seulement ces deux balises :

```
<app-menu></app-menu>
<router-outlet />
```

3. app.routes.ts

Dans ce fichier nous allons déclarer toutes les **routes** de notre application, sous la forme d'un tableau dans la constante **routes**. On importe dans ce fichier tous les composants dont on va avoir besoin pour gérer les redirections, puis on décrit, pour chaque route :

- Un chemin, le **path**. C'est l'URL qui va nous permettre d'accéder au contenu souhaité
- Un composant **component**, c'est le composant qui va être chargé lorsque la route va être appelée

```
import { Routes } from '@angular/router';
import { Homepage } from './homepage/homepage';
import { QuiSommesNous } from './qui-sommes-nous/qui-sommes-nous';
import { Contact } from './contact/contact';

export const routes: Routes = [
    {
        path: '',
        component: Homepage
    },
    {
        path: 'qui-sommes-nous',
        component: QuiSommesNous,
    },
    {
        path: 'contact',
        component: Contact,
    }
];
```

4. RouterLink

Enfin, dans notre template, nous allons avoir besoin de déclarer nos liens comme étant dépendant du router. Pour cela, nous allons utiliser **RouterLink**, une classe à importer comme ceci dans notre menu par exemple :

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

import { RouterLink } from '@angular/router';

@Component({
  selector: 'app-menu',
  imports: [RouterLink],
  templateUrl: './menu.html',
  styleUrl: './menu.scss',
})
export class Menu {

}
```

Une fois que cette classe est importée, nous pouvons l'utiliser comme ceci en tant qu'attribut de nos balises <a>, en **remplaçant** l'attribut **href** :

```
<a class="nav-link mx-2" routerLink="qui-sommes-nous">
      Qui-sommes-nous ?
</a>
```

La valeur que nous donnons à l'attribut routerLink doit correspondre à l'un des **path** que nous avons décrit dans **app.routes.ts**

5. Lazy Loading ?

Et si on s'évitait de charger tous les composants d'un coup pour laisser Angular choisir quoi charger selon le besoin de l'utilisateur ?

De manière générale, en informatique, on appelle « **lazy loading** » le fait de ne charger une ressource que lorsqu'elle est visible. Cela se dit pour les images, les fichiers, et… les composants Angular.

Pour utiliser cela dans notre router, on va utiliser la méthode loadComponent() comme ceci :

```
import { Routes } from '@angular/router';

export const routes: Routes = [
    {
        path: '',
        loadComponent: () => import('./components/homepage/homepage').then(m => m.Homepage)
    },
    {
        path: 'qui-sommes-nous',
        loadComponent: () => import('./components/qui-sommes-nous/qui-sommes-nous').then(m =>
m.QuiSommesNous)
    },
    {
        path: 'contact',
        loadComponent: () => import('./components/contact/contact').then(m => m.Contact)
    }
];
```

Le résultat attendu est rigoureusement identique, mais cette façon de faire permet d'alléger le chargement initial du site et donc d'optimiser son temps de chargement global. Et donc son référencement ☺

6. Récapitulons

- Nous avons décrit nos **routes** dans le fichier app.routes.ts, avec ou sans Lazy Loading
- Nous avons importé la classe **RouterOutlet** dans la composant racine

- Nous avons prévu un **emplacement** dans le composant racine pour accueillir le contenu qui sera chargé : la balise **<router-outlet/>**
- Notre menu, inclus par défaut dans le composant racine, va nous permettre de naviguer vers les URLs suivantes : « », « qui-sommes-nous », « contact » grâce au **RouterLink**
- Le router va **intercepter** ces URLs et s'en servir pour charger les composants correspondants dans le **<router-outlet/>**