ROUTING & NAVIGATION

ULT – DÉPARTEMENT GÉNIE INFORMATIQUE

Module: Projet Framework et Développement

Public cible: 5° Génie Logiciel

Houcem Hedhly

houssem.hedhli@ult-tunisie.com

2021/2022

PLAN

I. Principe de base
II. Méthodes d'intégration
III. Exemple
IV. Cas pratiques

I. PRINCIPE DE BASE

- SPA: changer ce que voit l'utilisateur en réponse à ses interactions avec l'application.
- Afficher et/ou cacher une partie de l'interface (View) et on ne charge pas toute la page.
- Une partie de l'interface = Template(s) de un (ou plusieurs) component(s).
- Pour gérer la navigation d'une View à l'autre, on utilise le module Angular prédéfini `Router`.
- Active la navigation par l'interprétation une URL comme une instruction pour charger View.

II. MÉTHODES D'INTÉGRATION

- Scénario 1: créer une application, à l'aide de la CLI, en activant le module de routing dès le départ.
- Scénario 2: ajouter le module de routing dans une application existante.
  - Méthode 1: un module dédié au routing
  - Méthode 2: intégrer le module de routing dans le module de l'application.
- Ajouter un élément `<base href>` dans le fichier `index.html`

```
<base href="/">
```

II. MÉTHODES D'INTÉGRATION

Scénario 1:

- Créer une application, à l'aide de la CLI en activant le module Router.
- Intègre un NgModule appelé AppRoutingModule qui permet de configurer et de définir les routes.

```
$ ng new app-name --routing
```

II. MÉTHODES D'INTÉGRATION

Scénario 2 : ajouter un module de routing dans une application existante.

Méthode 1: générer un `NgModule` appelé `AppRoutingModule` dédié au routing, et qui permet de configurer et de définir les routes.

```
$ ng generate module app-routing --flat --module=app
```

- `--flat` : empêche de créer un dossier séparé pour le nouveau module.
- `--module` : importer le nouveau module dans le AppModule.

II. MÉTHODES D'INTÉGRATION

Scénario 2 : ajouter un module de routing dans une application existante.

Méthode 2: intégrer le module de routing dans le module auquel il appartient.

- Etape 1: importer `RouterModule` et `Routes` dans le `AppModule`.

```
// src/app/app.module.ts
import { RouterModule, Routes } from '@angular/router'
```

II. MÉTHODES D'INTÉGRATION

- Etape 2: créer un tableau qui contiendra la définition des routes.

```
// src/app/app.module.ts
const routes: Routes = []
```

- First-match wins.
- Les routes les plus précis doivent être mentionné au-dessus des rotes les moins précis.

II. MÉTHODES D'INTÉGRATION

- Etape 3: ajouter le module `RouterModule` dans le tableau des importation du décorateur `NgModule`.

```
// src/app/app.module.ts
@NgModule({
  imports: [BrowserModule, RouterModule.forRoot(routes)],
  declarations: [AppComponent, HomeComponent],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

II. MÉTHODES D'INTÉGRATION

- Etape 4: ajouter un moyen pour les routes pour faire l'affichage. Ajouter le `router-outlet` au `AppComponent`:

```
// src/app/app.component.ts
<router-outlet></router-outlet>
```

III. EXEMPLE

- Définir les routes dans le tableau `routes`.
- Chaque route est un objet JavaScript à deux propriétés:
  - `path`: l'URL du route
  - `component`: le nom du component qui sera affiché quand cette URL est sollicitée.

```
const routes: Routes = [
  { path: 'first-url', component: FirstComponent },
  { path: 'second-url', component: SecondComponent }
];
```

III. EXEMPLE

- Ajouter les routes dans l'application.
- Dans l'élément HTML classique `<a>`, on remplace l'attribut `href` par `routerLink`:

```
<nav>
  <ul>
    <li><a routerLink="/first-url">First Component</a></li>
    <li><a routerLink="/second-url">Second Component</a></li>
  </ul>
</nav>
<!-- The routed views render in the <router-outlet>-->
<router-outlet></router-outlet>
```

IV. CAS PRATIQUE #1: WILDCARD ROUTE

- Règle de base: gérer les requêtes aux pages qui n'existent pas.
- Solution: créer un **wildcard route** (route joker): `{ path: '**', component: }`
- Le module Router sélectionne cette route chaque fois l'URL demandée à aucun chemin dans le tableau des routes.
- Utiliser un component qui affiche une page **`404 Not FOUND`** comme réponse.
- Idéalement, on place le wildcard route en dernier rang dans le tableau des routes.

```
{ path: '**', component: NotFoundComponent}
```

IV. CAS PRATIQUE #2: REDIRECTION

- Définir une route :
  - `path`: chemin demandé qui nécessite d'être redirigé.
  - `redirectTo`: le chemin cible de la redirection.
  - `pathMatch`: défini comment faire correspondre l'URL.

```
const routes: Routes = [
  { path: 'first-url', component: firstComponent },
  { path: 'second-url', component: secondComponent },
  { path: '',   redirectTo: '/first-url', pathMatch: 'full' },
  { path: '**', component: NotFoundComponent}
];
```

IV. CAS PRATIQUE #3: ROUTE PARAMS

- Importer `Router`, `AtivatedRoute` et `ParamMap` dans le component en question:

```
import { Router, ActivatedRoute, ParamMap } from '@angular/router';
```

- Injecter une instance de `ActivatedRoute` dans le constructeur du component:

```
constructor(private route: ActivatedRoute) {}
```

- Accéder aux paramètres de l'URL dans la méthode `ngOnInit()`:

```
ngOnInit() {
  this.route.queryParams.subscribe(params => {
    this.name = params['name'];
  });
}
```