

Par NIDHAL JELASSI
nidhal.jelassi@fsegt.utm.tn

PROGRAMMATION WEB AVANCÉE

ANGULAR



Chapitre 6 : Le routing

OBJECTIFS

- ▶ Définir le routeur d'Angular
- ▶ Définir une route
- ▶ Déclencher une route à partir d'un composant
- ▶ Ajouter des paramètres à une route
- ▶ Récupérer les paramètres d'une route à partir du composant.
- ▶ Préfixer un ensemble de routes
- ▶ Gérer les routes innexistantes

UN SYSTÈME DE ROUTING...POURQUOI ?

- ▶ Un système de routing permet d'associer une route (url - path) à un traitement particulier.
- ▶ Angular produit une SPA. Pourquoi alors parle-on de route ?
- ▶ L'objectif est donc de :
 - ▶ Séparer différentes fonctionnalités du système
 - ▶ Maintenir l'état de l'application
 - ▶ Ajouter des règles de protection

UTILITÉ

- ▶ *Que risque t-on d'avoir si on n'utilise pas un système de routing ?*
- ▶ On ne peut plus rafraichir notre page
- ▶ Perdre tout son contexte à chaque changement de page
- ▶ Impossibilité de partager nos pages

CRÉATION D'UN SYSTÈME DE ROUTING

1. Indiquer au routeur comment composer les urls en ajoutant dans le head la balise suivante : `<base href="/">`
2. Créer un fichier 'app.routing.ts'
3. Importer le service de routing d'Angular : `import { RouterModule, Routes } from '@angular/router';`
 - ▶ Le **RouterModule** va permettre de configurer les routes dans notre projet.
 - ▶ Le **Routes** va permettre de créer les routes

CRÉATION D'UN SYSTÈME DE ROUTING

4. Créer une **constante** qui est un **tableau d'objets** de type **Routes** représentant chacun la route à décrire.
5. Intégrer les routes à notre application dans le **AppModule** à travers le **RouterModule** et sa méthode **forRoot**.

```
1 import { Routes, RouterModule } from '@angular/router';
2 import { HomeComponent } from './home/home.component';
3 import { ServersComponent } from './servers/servers.component';
4 import { UsersComponent } from './users/users.component';
5
6 const routes : Routes = [
7   { path : '', component: HomeComponent },
8   { path : 'servers', component: ServersComponent },
9   { path : 'users', component: UsersComponent }
10 ];
11
12 export const ROUTING = RouterModule.forRoot(routes);
```

```
import { ServersService } from './servers/server
import { ROUTING } from './app.routing';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    UsersComponent,
    ServersComponent,
    UserComponent,
    EditServerComponent,
    ServerComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    ROUTING
  ],
```

UTILISATION DU SYS DE ROUTING

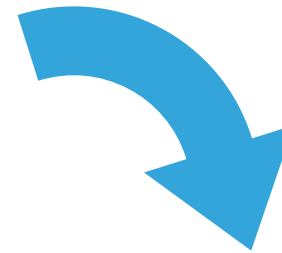
- ▶ Pour indiquer à Angular où est-ce qu'il doit charger les vues spécifiques aux routes nous utilisons le **router-outlet**.
- ▶ Router-outlet est une directive qui permet de spécifier l'endroit où la vue va être chargée.
- ▶ Sa syntaxe est **<router-outlet></router-outlet>**

```
<div class="container">
  <div class="row">
    <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
      <ul class="nav nav-tabs">
        <li role="presentation" class="active"><a href="#">Home</a></li>
        <li role="presentation"><a href="#">Servers</a></li>
        <li role="presentation"><a href="#">Users</a></li>
      </ul>
    </div>
  </div>
  <div class="row">
    <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
      <app-home></app-home>
    </div>
  </div>
</div>

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

EN PRATIQUE

```
<div class="container">
  <div class="row">
    <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
      <ul class="nav nav-tabs">
        <li role="presentation" class="active"><a href="#">Home</a></li>
        <li role="presentation"><a href="#">Servers</a></li>
        <li role="presentation"><a href="#">Users</a></li>
      </ul>
    </div>
  </div>
  <div class="row">
    <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
      <app-home></app-home>
    </div>
  </div>
  <div class="row">
    <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
      <app-users></app-users>
    </div>
  </div>
  <div class="row">
    <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
      <app-servers></app-servers>
    </div>
  </div>
</div>
```



```
<div class="container">
  <div class="row">
    <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
      <ul class="nav nav-tabs">
        <li role="presentation" class="active"><a href="#">Home</a></li>
        <li role="presentation"><a href="#">Servers</a></li>
        <li role="presentation"><a href="#">Users</a></li>
      </ul>
    </div>
  </div>
  <div class="row">
    <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
      <app-home></app-home>
    </div>
  </div>

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


SYNTAXE MINIMALISTE D'UNE ROUTE

- ▶ Une route est un objet.
- ▶ Les deux propriétés essentielles sont **path** et **component**.
- ▶ **path** permet de spécifier l'URI. Cette url ne doit pas commencer par un **/**.
- ▶ **component** permet de spécifier le composant à exécuter.

```
{path: ' ',component: CvComponent },  
{path: 'onlyHeader',component: HeaderComponent }
```

EXERCICE

- ▶ Configurer votre routing
- ▶ Créer deux composants
- ▶ Créer deux routes qui pointent sur ces deux composants
- ▶ Vérifier le fonctionnement de votre routing



DÉCLENCHER UNE ROUTE (ROUTERLINK)

- ▶ L'idée intuitive pour déclencher une route est d'utiliser la balise **a** et son attribut href. Quel problème cette idée risque de poser ?
- ▶ L'utilisation de **<a href >** va déclencher le chargement de la page ce qui est **inconcevable pour une SPA**.
- ▶ La solution proposée par le **router** d'Angular est l'utilisation de la directive **routerLink** qui comme son nom l'indique liera la directive à la route que nous souhaitons déclencher sans recharger la page.
- ▶ Exemple :

```
<li ><a [routerLink]= '['cv']" routerLinkActive="active">Gérer les  
cvs</a></li>
```

EXERCICE



- ▶ Faites en sorte d'avoir un composant dans votre application qui permet d'afficher l'ensemble de vos liens.
- ▶ En cliquant sur un lien, le composant qui lui est associé doit être affiché.

DÉCLENCHER UNE ROUTE (ROUTERLINK)

- **routerLinkActiveOptions** + **exact** permet de définir que la classe CSS active est ajoutée si le chemin correspond exactement. Sans cela par exemple pour la route « /users » ayant des childrens, le lien serait toujours actif que l'on sur « /users/id » ou « users/about ».

```
<div class="row">
  <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
    <ul class="nav nav-tabs">
      <li role="presentation"
        routerLinkActive="active"
        [routerLinkActiveOptions]="{exact: true}">
        <a routerLink="/">Home</a>
      </li>
      <li role="presentation"
        routerLinkActive="active">
        <a routerLink="servers">Servers</a>
      </li>
      <li role="presentation"
        routerLinkActive="active">
        <a [routerLink]="['users']">Users</a>
      </li>
    </ul>
  </div>
```

DÉCLENCHER UNE ROUTE À PARTIR DU COMPOSANT

- ▶ Afin de déclencher une route à travers le composant on utilise l'objet Router et sa méthode **navigate**.
- ▶ Cette méthode prend le même paramètre que le **routerLink**, à savoir un tableau contenant la description de la route.
- ▶ Afin d'utiliser le Router, il faut l'importer de l'**@angular/router** et l'injecter dans votre composant.

```
onLoadServer(id: number) {  
  // traitement à faire  
  this.router.navigate(['/servers', id])  
}
```

EXERCICE



- ▶ Créer un composant appelé RouterSimulator
- ▶ Dans ce composant créer une liste déroulante contenant le nom des différentes routes de votre application.
- ▶ Ajouter ce composant au même niveau que le header et que le `<router-outlet>`.
- ▶ En sélectionnant le nom d'un composant, il doit apparaître dans le `<router-outlet>` simulant ainsi le fonctionnement d'un routeur.

LES PARAMÈTRES D'UNE ROUTE

- ▶ Afin de spécifier à notre router qu'un segment d'une route est un paramètre, il suffit d'y ajouter ':' devant le nom de ce segment.
- ▶ Exemple : **/user/:id** permet de dire que la route contient au début **user** ensuite un paramètre de route appelé **id**.

```
const appRoutes: Routes = [  
  { path: '', component: HomeComponent },  
  { path: 'users', component: UsersComponent, children: [  
    { path: 'about', component: AboutComponent },  
    { path: ':id/:name', component: UserComponent }  
  ] },  
]
```


RÉCUPÉRATION DES PARAMÈTRES

- ▶ Afin de récupérer les paramètres d'une route au niveau d'un composant on doit procéder comme suit :
- 1. Importer **ActivatedRoute** qui nous permettra de récupérer les paramètres de la route.
- 2. Injecter **ActivatedRoute** au niveau du composant (c'est un service).
- 3. Affecter le paramètre à une variable du composant en s'inscrivant avec la méthode **subscribe** à **l'observable params** de notre **ActivatedRoute**. Cette variable retourne un tableau de l'ensemble des paramètres.
- ▶ Syntaxe : `activatedRoute.params.subscribe(params=>{this.monParam=params['param']});`

RÉCUPÉRATION DES PARAMÈTRES



QUELLE EST LA DIFFÉRENCE
ENTRE LES 2 TECHNIQUES ?

```
export class UserComponent implements OnInit, OnDestroy {  
  user: {id: number, name: string};  
  paramsSubscription: Subscription;  
  
  constructor(private route: ActivatedRoute) { }  
  
  ngOnInit() {  
    this.user = {  
      id: this.route.snapshot.params['id'],  
      name: this.route.snapshot.params['name']  
    };  
  
    this.paramsSubscription = this.route.params  
      .subscribe(  
        (params: Params) => {  
          this.user.id = params['id'];  
          this.user.name = params['name'];  
        }  
      );  
  }  
  
  ngOnDestroy() {  
    this.paramsSubscription.unsubscribe();  
  }  
}
```

1ère technique (avec snapshot)

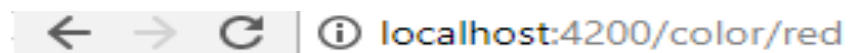
2ème technique (avec un observable)

SUBSCRIBE / UNSUBSCRIBE

- ▶ La méthode **subscribe** permet de s'inscrire à un observable.
- ▶ **Problème** : Cette souscription reste valide même après la disparition de la variable ce qui sature la mémoire pour rien.
- ▶ **Solution** : Se Désinscrire à la mort du composant donc dans le `ngOnDestroy()`.

EXERCICE

- ▶ Reprendre le composant qui permet de changer la couleur de la DIV
- ▶ Ajouter lui une route
- ▶ Faire en sorte que cette route soit de cette forme **/color:couleur** et qui permettra d'affecter la couleur récupérée par la route comme couleur par défaut du DIV.

A screenshot of a web browser's address bar. It features navigation icons (back, forward, refresh) and an information icon on the left. The address bar contains the text 'localhost:4200/color/red'.

LES QUERY PARAMETERS

- ▶ Les **queryParams** sont les paramètres envoyés à travers une requête **GET**.
- ▶ Identifié avec le **?**.
- ▶ Afin d'insérer un **queryParams** on dispose de deux méthodes :

1ère méthode :

- ▶ On ajoute dans la méthode `navigate` du Router un second paramètre de type objet.

LES QUERY PARAMETERS

- ▶ L'une des propriétés de cet objet est aussi un objet dont la clé est queryParams dont le contenu est aussi un objet content les identifiants des queryParams et leurs valeurs.

```
this.router.navigate([' /about', this.id], {queryParams: { 'qpVar': 'je suis un qp' } });
```

2ème méthode :

- ▶ On intègre à notre routerLink de la manière suivante :

```
<a [routerLink]="[' /about/10']" [queryParams]="{qpVar: 'je suis un qp bindé avec le routerLink'}">About</a>
```

RÉCUPÉRER LES QUERYPARAMETERS

- ▶ Les **queryParams** sont récupérable de la même façon que les paramètres.
- ▶ Afin de récupérer les paramètres d'une root au niveau d'un composant on doit procéder comme suit :
- ▶ Importer **ActivatedRoute** qui nous permettra de récupérer les paramètres de la root.
- ▶ Injecter **ActivatedRoute** au niveau du composant.
- ▶ Affecter le paramètre à une variable du composant en utilisant la méthode **subscribe** pour se souscrire à **params** de notre **ActivatedRoute**.
- ▶ Syntaxe :

```
activatedRouter.queryParams.subscribe(  
  (queryParams: any) => (this.monQp=queryParam[ 'qpVar' ] )  
);
```

CHILD ROUTES

- ▶ Certains composants ne sont visibles qu'à l'intérieur d'autres composants.
- ▶ Prenons l'exemple d'un objet Personne. En accédant à la route `/personne/:id` nous avons l'affichage de la personne et nous aimerions avoir deux boutons. Un pour éditer la Personne (route **`/personne/:id/editer`**). L'autre pour afficher ces détails (route **`/personne/:id/aperçu`**).
- ▶ L'idée est de préfixer nos routes.
- ▶ Afin de mettre en place ce processus nous procédons comme suit :
 - ▶ Nous définissons le préfixe avec la propriété **`path`**.
 - ▶ Nous y ajoutons la propriété **`children`** qui contiendra le tableau des routes. Chaque route de ce tableau sera préfixé avec la route définie dans `path`.

CHILD ROUTES + TEMPLATE

```
const appRoutes: Routes = [  
  { path: '', component: HomeComponent },  
  { path: 'users', component: UsersComponent, children: [  
    { path: 'about', component: AboutComponent },  
    { path: ':id/:name', component: UserComponent }  
  ] },  
]
```

- ▶ Supposons que nous voulons avoir un Template central avec des données fixes et des parties variables dans le même template.
- ▶ En changeant les routes, le contenu principal doit rester le même et la partie variable doit changer selon la route.

CHILD ROUTES

- ▶ Afin de mettre en place ce processus nous procédons comme suit :
- ▶ Nous définissons le préfixe avec la propriété `path`. On lui associe le composant Père.
- ▶ Nous y ajoutons la propriété **children** qui contiendra le tableau des routes. Chaque route de ce tableau sera préfixé avec la route définie dans `path`.
- ▶ Nous ajoutons la balise `<router-outlet></router-outlet>` dans le Template père.

RedirectTo

- ▶ Afin de rediriger une route il suffit d'ajouter une propriété dans l'objet route qui est **redirectTo**. Cette propriété permet d'indiquer vers quelle route le **path** doit être redirigé. Si la route n'a pas encore été matché, alors les routes commençant par ce path seront redirigées.
- ▶ Une autre propriété peut être utilisé qui est la propriété pathMatch. Cette propriété permet de définir comment le matching des path est exécuté. Avec la valeur '**full**', elle spécifie au routeur de ne faire la redirection que si le path exact est matché.

RedirectTo

```
const APP_Routes:Routes =[
  {path: '', component: HomeComponent},
  {path: 'about', redirectTo: '/', pathMatch: 'full'},
  {path: 'about/:param', component: AboutComponent},
  {path: 'about/:param', component: AboutComponent,
    children: FILS_ROUTE},
]
```

ERROR 404

- ▶ Afin de rediriger une route inexistante vers une page d'erreur, il suffit de garder la même syntaxe de redirection et de mettre dans la propriété `path` `'**'`.

```
▶ const APP_ROUTE: Routes = [  
  { path: '', redirectTo: 'cv', pathMatch: 'full' },  
  { path: 'lampe', component: ColorComponent },  
  { path: 'login', component: LoginComponent },  
  { path: 'error', component: ErrorPageComponent },  
  { path: '**', component: ErrorPageComponent }  
];
```

RETOUR AU PROJET...

- ▶ Créer un composant HeaderComponent contenant votre navbar.
- ▶ Ajouter les fonctionnalités suivante à votre CV:
- ▶ Une page détail qui va afficher les détails d'un CV.
- ▶ Un bouton dans chaque cv qui au click vous renvoie vers la page détails.
- ▶ Dans la page détail, un bouton delete qui au click supprime ce cv et vous renvoi à la liste des cvs.