chapitre2
Routage et Navigation

Introduction au Routage

- En général, on ne cherche pas à afficher tous les composants au même temps dans le bloc principal.
- On définit plutôt un chemin pour chaque composant.
- Un système de routing permet d’associer une route (url - path) à un traitement particulier.
- Il s'agit des instructions d'affichage à suivre pour chaque URL, c'est-à-dire quel(s) component(s) il faut afficher à quel(s) endroit(s) pour un URL donné.
- Le routeur angular permet la navigation d'une vue à l'autre lorsque les utilisateurs exécutent des tâches d'application.

AppRouting module

- Angular-CLI nous propose le module applicatif **AppRoutingModule** permettant d'ajouter la fonctionnalité de routage à une application :

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
```

✓**Routes** : est un Array contenant la déclaration des Routes

✓**RouterModule** : est un module regroupant les directives et les services paramétrables permettant de remplir la fonctionnalité de routage.

Création d'un système de Routing

- La constante **routes** représente les chemins à déclarer dans un projet Angular:
  - ➢Chaque objet {**path**: « url du composant », **component**: « la class du composant concerné »}
- Intégrer les routes à notre application dans le **App-routingModule** à travers le **RouterModule** et sa méthode ***forRoot***

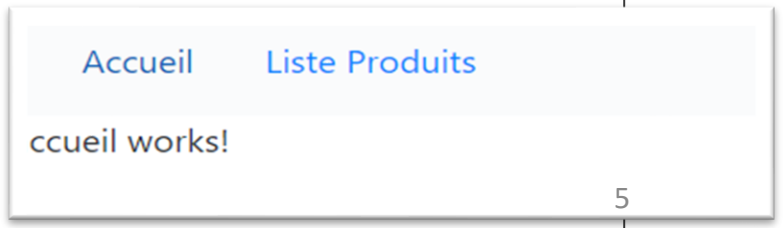
```
const routes: Routes = [

    {path:"products",component:ListeProduitsComponent},
    {path:"accueil",component:AccueilComponent},
    {path:"products/:id",component:DetailsComponent},
    {path:"",redirectTo:"/accueil",pathMatch:'full'},
    {path:"**", component:PageNotFoundComponent }

];
```

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
```

Application de routage

- Pour appliquer un routage dans une vue (HTML Template), on a besoin des directives suivantes:
  - **routerLink:** c'est une directive prenant en valeur le chemin (path) indiqué dans la table routes
  - **Router-outlet** est une directive qui permet de spécifier l'endroit ou la vue va être chargée. Sa syntaxe est **<router-outlet></router-outlet>**
- **Exemple:** Quand l'utilisateur tape http://localhost:4200/accueil , le routeur cherche et charge le composant *AccueilComponent* et l'affiche dans un élément **<router-outlet></router-outlet>** . Cet élément est sensé se trouver dans la vue du composant racine.

```
<nav class='navbar navbar-expand navbar-light bg-light'>
      <ul class='nav nav-pills'>
        <li><a class='nav-link' routerLink="/accueil">Accueil</a></li>
        <li><a class='nav-link' routerLink="/products">Liste Produits</a></li>
      </ul>
    </nav>
<!-- <app-liste-produits></app-liste-produits> -->
<router-outlet></router-outlet>
```



Redirection

- **redirectTo:** est une propriété ajouté dans l'objet du route pour rediriger une route
- Cette propriété permet d'indiquer vers quelle route le path doit être redirigé.

```
const routes: Routes = [

    {path:"products",component:ListeProduitsComponent},
    {path:"accueil",component:AccueilComponent},
    {path:"products/:id",component:DetailsComponent},
    {path:"",redirectTo:"/accueil",pathMatch:'full'},
    {path:"**", component:PageNotFoundComponent }

];
```

- *Sans la partie pathMatch: 'full' (pour les chemins vides), toutes les routes déclarées après cette dernière ne seront pas accessibles.*
- *pathMatch: 'full' ne laisse donc passer que les requêtes dont le chemin correspond **exactement** au chemin vide*
- *La deuxième valeur possible pour pathMatch est 'prefix'*

Route parametres

```
const routes: Routes = [

    {path:"products",component:ListeProduitsComponent},
    {path:"accueil",component:AccueilComponent},
    {path:"products/:id",component:DetailsComponent},
    {path:"",redirectTo:"/accueil",pathMatch:'full'},
    {path:"**", component:PageNotFoundComponent }

];
```

```
<a routerLink="/products/{{p.id}}">{{p.nom}}</a>
```

ou

```
<a [routerLink]="['/products',p.id]">{{p.nom}}<br></a>
```

Récupération des paramètres

1. Importer ***ActivatedRoute*** qui nous permettra de récupérer les paramètres de la route.
2. Injecter ActivatedRoute au niveau du composant.
3. Affecter le paramètre à une variable du composant

Injection de dépendances: *consiste à créer dynamiquement (injecter) les dépendances entre les différents objets en s'appuyant sur une description (fichier de configuration ou métadonnées) ou de manière programmatique.*
Ainsi les dépendances entre composants logiciels ne sont plus exprimées dans le code de manière statique mais déterminées dynamiquement à l'exécution.

- L'affectation peut être faite de plusieurs façons:
  - **Méthode 1**: un snapshot de l'URL via le ***ActivatedRoute***

```
import { ActivatedRoute } from '@angular/router';
constructor(private ar:ActivatedRoute) { }
ngOnInit() {
  this.id=this.ar.snapshot.params.id;
}
```

ou

```
ngOnInit() {
  this.id=+this.ar.snapshot.paramMap.get('id')
}
```

  - **Méthode 2**: Via Observable : la fonction subscribe

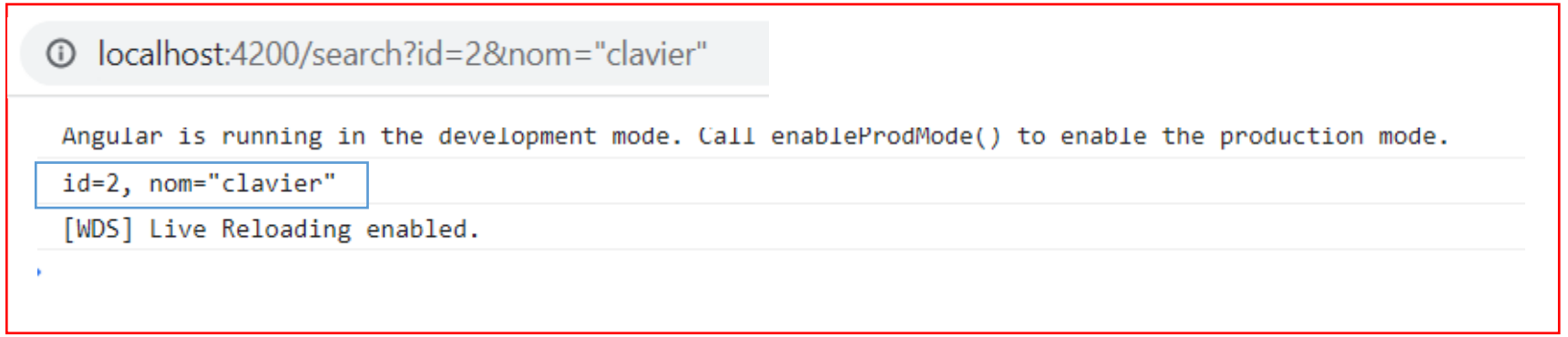
```
ngOnInit() {  this.ar.paramMap.subscribe((params:ParamMap)=>this.id=+params.get('id')) }
```

Récupération des paramètres

- Une autre méthode pour récupérer des données : **queryParams**
  - Les queryParameters sont les paramètres envoyé à travers une requête GET.
  - Identifié avec le ?.

```
ngOnInit() {

        this.id=this.ar.snapshot.queryParams.id;
        console.log("id="+this.id+", nom="+this.ar.snapshot.queryParams.nom);
}
```

```
{path:"search",component:DetailsComponent},
```



Application du route via 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 Route, il faut l'importer de l'@angular/router et l'injecter dans votre composant.

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

constructor(private ar:ActivatedRoute,private router:Router) { }

retour()
{
  this.router.navigate(['/products']);
}
```

```
<button class="btn btn-primary" (click)=retour()>back</button>
```

Wildcard route

- Le wildcard route permet de rediriger vers un composant si le chemin indiqué n'existe pas (The not found page):

```
const routes: Routes = [

    {path:"products",component:ListeProduitsComponent},
    {path:"accueil",component:AccueilComponent},
    {path:"products/:id",component:DetailsComponent},
    {path:"",redirectTo:"/accueil",pathMatch:'full'},
    {path:"**", component:PageNotFoundComponent }

];
```

- ***NB: Il faut faire attention à l'emplacement du Wildcard route***