Cours Angular : Le Routage (Angular Router)

1. Introduction

Le **routage Angular** permet de gérer la navigation entre les différentes pages (vues) d'une application **Single Page Application (SPA)**, sans rechargement de page.

Objectifs de ce cours : - Comprendre le rôle du routage - Configurer Angular Router - Créer des routes simples et avancées - Utiliser les paramètres de route - Mettre en place la navigation - Sécuriser les routes avec les guards

---

2. Qu'est-ce que le Router Angular ?

Le **Router** est un module Angular qui : - Associe une **URL** à un **composant** - Charge dynamiquement le composant correspondant - Gère l'historique de navigation

Exemple : - `/login` → `LoginComponent` - `/dashboard` → `DashboardComponent`

---

3. Mise en place du routage

3.1 Création du module de routage

Lors de la création du projet :

```
ng new my-app --routing
```

Ou manuellement :

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

3.2 Configuration des routes

Fichier `app-routing.module.ts` :

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { LoginComponent } from './login/login.component';

const routes: Routes = [
```

```
  { path: '', component: HomeComponent },
  { path: 'login', component: LoginComponent }
];

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

---

4. Affichage des routes

4.1 `router-outlet`

Le composant chargé par le router s'affiche dans :

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

👉Généralement placé dans `app.component.html`.

---

5. Navigation entre les pages

5.1 Navigation avec `routerLink`

```
<a routerLink="/login">Connexion</a>
<button routerLink="/dashboard">Dashboard</button>
```

Avec paramètres :

```
<a [routerLink]="['/user', userId]">Profil</a>
```

5.2 Navigation programmatique

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

constructor(private router: Router) {}

login() {
  this.router.navigate(['/dashboard']);
}
```

---

6. Routes avec paramètres

6.1 Définition

```
{ path: 'user/:id', component: UserComponent }
```

6.2 Récupération des paramètres

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

constructor(private route: ActivatedRoute) {}

ngOnInit() {
  const id = this.route.snapshot.paramMap.get('id');
}
```

Ou en mode réactif :

```
this.route.params.subscribe(params => {
  console.log(params['id']);
});
```

---

7. Routes enfants (Child Routes)

```
const routes: Routes = [
  {
    path: 'dashboard',
    component: DashboardComponent,
    children: [
      { path: 'profile', component: ProfileComponent },
      { path: 'settings', component: SettingsComponent }
    ]
  }
];
```

Navigation :

```
<a routerLink="/dashboard/profile">Profil</a>
```

---

8. Redirections et routes wildcard

8.1 Redirection

```
{ path: '', redirectTo: '/login', pathMatch: 'full' }
```

8.2 Route 404

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

---

9. Guards (sécurisation des routes)

Les **guards** contrôlent l'accès aux routes.

Types principaux : - `CanActivate` - `CanActivateChild` - `CanDeactivate`

9.1 Exemple `CanActivate`

```
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';

@Injectable({ providedIn: 'root' })
export class AuthGuard implements CanActivate {

  constructor(private router: Router) {}

  canActivate(): boolean {
    const isLogged = true;

    if (!isLogged) {
      this.router.navigate(['/login']);
      return false;
    }
    return true;
  }
}
```

Utilisation :

```
{ path: 'dashboard', component: DashboardComponent, canActivate:
[AuthGuard] }
```

10. Lazy Loading (chargement paresseux)

Permet d'améliorer les performances en chargeant les modules **uniquement si nécessaire**.

```
{
  path: 'admin',
  loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule)
}
```

11. Cas pratique complet

Objectif

Créer une navigation sécurisée avec paramètres.

```
const routes: Routes = [
  { path: 'login', component: LoginComponent },
  { path: 'products', component: ProductsComponent, canActivate:
[AuthGuard] },
  { path: 'product/:id', component: ProductDetailComponent },
  { path: '**', component: NotFoundComponent }
];
```

12. Bonnes pratiques

- Utiliser le lazy loading pour les gros modules
- Centraliser les routes dans des modules dédiés
- Toujours prévoir une route 404
- Protéger les routes sensibles avec des guards

13. Résumé

- Angular Router gère la navigation SPA
- Les routes lient URL ↔ composant
- `routerLink` et `navigate()` pour naviguer
- Guards pour la sécurité
- Lazy loading pour la performance

👉Prochaine étape recommandée : **Resolvers, guards avancés et navigation avancée**