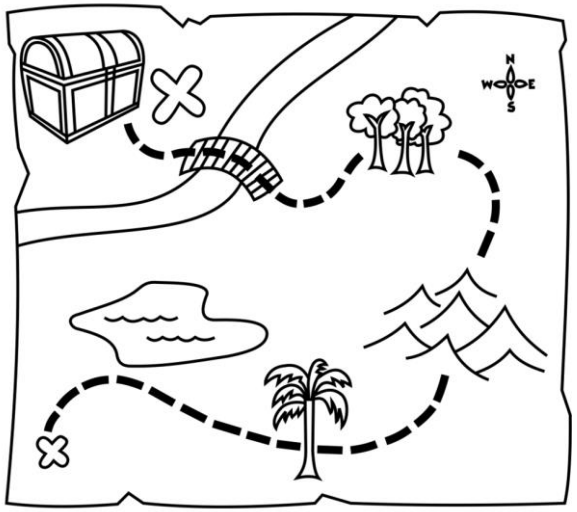


Routing Angular

Animé par Mazen Gharbi

Routing

▷ Sans « **deep-linking** », les utilisateurs ne peuvent pas partager des URLs pour accéder à une vue ou plus exactement une "route".

▷ Angular offre 2 possibilité pour l'affichage de vos urls :

- Hashbang mode
- HTML5 mode

▷ Chaque "route" doit avoir une URL unique et explicite permettant de la retrouver ;

Mise en place du routing

▷ Pour mettre en place le routing en Angular, il est nécessaire d'utiliser le module « RouterModule » fournit par « @angular/router »

app.module.ts

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    ReactiveFormsModule,
    RouterModule.forRoot(APP_ROUTES, {useHash: false})
  ],
  providers: [
    APIHelper,
  ],
  bootstrap: [AppComponent]
})
```

Nous créons cette variable !

app.routes.ts

```
import {LoginComponent} from './login/login.component';
import {HomeComponent} from './home/home.component';

export const APP_ROUTES = [
    {
        component: LoginComponent,
        path: 'login'
    },
    {
        component: HomeComponent,
        path: 'home'
    },
    {
        redirectTo: '/login',
        path: '**'
    },
];
```

Mise en place du routing

▷ Pour indiquer où le composant associé à la "route" doit être chargé, nous utiliserons le composant « RouterOutlet »

app.component.html

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

Naviguer entre nos routes

- Dans le template :

```
<a [routerLink]="['./hello']">Hello</a>
<a [routerLink]="['./forum', 'category']">Contact</a>
```

- Dans le component :

```
constructor (private router: Router, private activatedRoute: ActivatedRoute) {
    this.router.navigate(['./list-tickets'], {relativeTo: this.activatedRoute});
}
```

Préférez l'utilisation des chemins relatifs.

Routes enfants

▷ Pour éviter de configurer toutes les "routes" à la racine de l'application, Angular permet de définir une arborescence de "routes".
▷ Le "root component" définit les "routes" des "child components".

▷ Les "child components" définissent les "routes" de leurs "child components" et ainsi de suite...

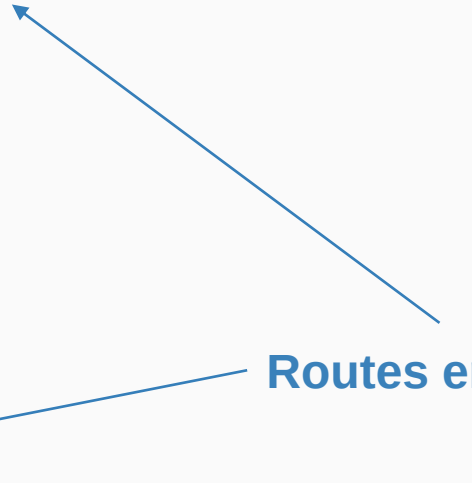
▷ On peut ainsi imaginer plusieurs équipes développer des applications différentes que l'on peut ensuite intégrer dans une application racine.

Routes enfants

app.routes.ts

```
import {LoginComponent} from './login/login.component';
import {HomeComponent} from './home/home.component';
import {HOME_ROUTES} from './home/home.routes';

export const APP_ROUTES = [
   {
      component: LoginComponent,
      path: 'login'
   },
   {
      component: HomeComponent,
      path: 'home'
      children: HOME_ROUTES
   },
   {
      redirectTo: 'login',
      path: '**'
   },
];
```



Routes enfants !

Routes enfants

home/home.routes.ts

```
import {ListTicketsComponent} from './list-tickets/list-tickets.component';
import {AddTicketComponent} from './add-ticket/add-ticket.component';
import { Route } from '@angular/router';

export const HOME_ROUTES: Route[] = [
   {
      component: ListTicketsComponent,
      path: 'list-tickets'
   },
   {
      component: AddTicketComponent,
      path: 'ticket/add'
   },
   {
      redirectTo: 'list-tickets',
      path: '**'
   }
];
```

Paramètres de route

▷ Nous pourrions avoir besoin d'afficher un contenu différent en fonction des paramètres transmis à la "route"
▷ Les paramètres obligatoires doivent être déclarés dans le "path" de la "route"

home/home.routes.ts

```
{
  component: AddTicketComponent,
  path: 'ticket/edit/:idTicket'
}
```

▷ Puis lors de l'appel de la route :

```
this.router.navigate(['../ticket', 'edit', index], {relativeTo: this.activatedRoute});
```

Paramètres de route

▷ Une fois le paramètre envoyé, il va être nécessaire de le récupérer dans le composant ciblé

add-ticket.component.ts

```
this.activatedRoute.params.subscribe((params) => {
      this.paramId = params.idTicket;
});
```

Pourquoi un observable?

▷ En cas de changement des paramètres (navigation vers le même composant mais avec des paramètres différents), Angular réutilisera le même composant et il faut donc mettre à jour les données du composant.

Paramètres de route

▷ Angular propose un snapshot de la route au moment de son appel pour récupérer les paramètres à travers un objet et non plus un observable

add-ticket.component.ts

```
let paramId = +this.route.snapshot.params.idTicket;
```

Lazy loading

▷ Encore l'une des plus grandes nouveautés d'Angular, le « lazy loading » permet de charger de façon asynchrone les modules et leurs dépendances au moment où une route est sollicitée.

▷ Le point le plus magique du "lazy loading" est qu'il suffit de modifier une ligne pour l'activer.

▷ Le lazy-loading vous permettra d’optimiser votre application en évitant de charger l’entièreté de votre application au lancement

Lazy loading

```
{
  path: 'projects',
  loadChildren: './+printing-projects/printing-projects.module#PrintingProjectsModule'
},
{
  path: 'my-projects',
  loadChildren: './+my-projects/my-projects.module#MyProjectsModule'
},
{
  path: 'opportunities',
  loadChildren: () => import('./lazy/lazy.module').then(m => m.NomModule)
},
```

ANGULAR 8

Route Lifecycle Hooks

▷ Il est possible de contrôler l'accès à une route grace à la propriété « canActivate »

app.routes.ts

```
{
  component: HomeComponent,
  path: 'home',
  canActivate: [LoggedGuardService],
  children: HOME_ROUTES
}
```

▷ canActivate prend en paramètre un tableau de Dependency Injections !

Route Lifecycle Hooks

logged-guard.service.ts

```
import {Injectable} from '@angular/core';
import {CanActivate, Router} from "@angular/router";
import {UserService} from "../../../../shared/providers/user/user.service";

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

  constructor(private user: UserService, private router: Router) { }

  canActivate() {
      return this.user.isLogged().then((res) => {
          if (res) {
              return true;
          } else {
              this.router.navigate(['./login']);
              return false;
          }
      });
  }
}
```

Route Lifecycle Hooks

▷ Inversement, « canDeactivate » permet de contrôler la sortie d'une route

app.routes.ts

```
{
  component: HomeComponent,
  path: 'home',
  canDeactivate: [CanLeaveGuardService],
  children: HOME_ROUTES
}
```

Questions