Chapitre 10 Template Driven Forms

Vous avez deux types de formulaire avec Angular :

- Template driven Forms : création dans le template du formulaire. Quand Angular parcourt le template et qu'il rencontre la balise form il créera l'objet correspondant exploitable dans le TypeScript.
- Reactive Form création dynamique dans le TypeScript, elle offre plus de possibilité.

Template Driven form

Nous allons développer un petit moteur de recherche dans notre application.

Dans le template, dans la balise input vous devez mettre ngModel et un attribut name sur chaque input pour justement câbler cela. Ainsi Angular fera le lien entre le template et le TypeScript.

La directive ngModel permet de définir et de contrôler un champ du formulaire dans Angular.

Un simple attribut « name » pour sa part définit une clé/valeur qu'Angular exploitera pour récupérer la valeur de tel ou tel champ.

Il faudra également définir l'événement (ngSubmit) : une méthode dans le TypeScript permettra de récupérer l'objet formulaire du template dans le code TypeScript.

Il faudra définir une référence locale pour accéder au formulaire dans le code HTML.

Préparation

Importez le module FormsModule (template driven forms), dans le fichier app.module.ts :

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { AlbumsComponent } from './albums/albums.component';
import { AlbumDetailsComponent } from './album-details/album-details.component';

@NgModule({
  declarations: [
    AppComponent,
```

```
    AlbumsComponent,
    AlbumDetailsComponent
  ],
  imports: [
    BrowserModule,
    FormsModule // importez le module
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Exercice 16 (mise en place)

Créez le component search, puis placez le sélecteur de ce component dans le template du component AlbumsComponent. Vous retirerez le HTML du formulaire de recherche dans la navigation principale (template app.component.html) et vous le placerez dans le template albums.component.html, ce sera plus simple par la suite pour afficher les résultats de la recherche dans le component AlbumsComponent :

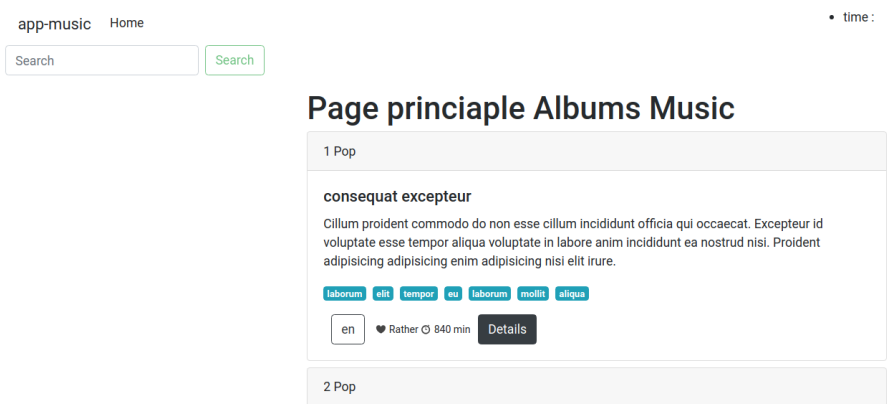

Figure 1: component search

Exercice 17 (formulaire/application)

Nous allons maintenant travailler dans le component SearchComponent.

Voici le code que vous devez écrire :

```
<!--
Méthode onSubmit à définir dans le TypeScript.
```

```
Vous devez également définir une référence locale #formSearch.
Ainsi que les directives ngModel pour lié un champ à son nom (data-binding)
-->
<form class="form-inline my-2 my-lg-0"
(ngSubmit)="onSubmit(formSearch)"
#formSearch="ngForm"
>
<input name="word"
ngModel required
class="form-control mr-sm-2"
type="search"
placeholder="Search"
aria-label="Search"
>
<!-- Directive disabled pour désactiver le formulaire si il n'est pas valide -->
<button
[disabled]="formSearch.invalid"
class="btn btn-outline-success my-2 my-sm-0"
type="submit"
>
Search
</button>
</form>
```

ngModel

Angular parcourt le template lorsqu'il trouve la balise **form** il crée cet objet, utilisable dans le TypeScript. Il faut préciser à Angular quel champ est un "controls", un champ à enregistrer afin de pouvoir récupérer sa valeur dans le TypeScript. Définissez un **name** et ajoutez la directive **ngModel** dans la balise input. Cette relation est du one-way data-binding (unidirectionnelle) elle est suffisante pour récupérer les valeurs d'un formulaire lors de sa soumission.

Il existe une autre syntaxe pour ngModel qui relie le name au TypeScript dans une relation two-way data-binding :

```
[(ngModel)]="name"
```

Notez que l'attribut required dans le HTML empêchera une soumission sans contenu.

Il faudra également importer NgForm, le type de l'objet form récupéré par Angular dans la méthode onSubmit (méthode appelée à la soumission grâce à l'événement ngSubmit) :

```
import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms'; // template-driven
```

```
import { AlbumService } from '../album.service';
import { Album } from '../album';

@Component({
  selector: 'app-search',
  templateUrl: './search.component.html',
  styleUrls: ['./search.component.scss']
})
export class SearchComponent implements OnInit {

  constructor() { }

  ngOnInit() {}

  onSubmit(form: NgForm): void {
     // récupération des données du formulaire
      console.log(form);
  }
}
```

Pour récupérer les valeurs d'un formulaire vous utiliserez la méthode suivante form.value[NameAttribute]. Ici on a qu'une seule paire de clé/valeur, mais vous devez fonctionner comme cela pour tous les champs d'un formulaire (ngModel + name) pour le template driven forms :

```
 onSubmit(form: NgForm): void {
      console.log(form.value['word']); // récupération d'une valeur spécifique
  }
```

Exercice 18

Vous allez maintenant créer une méthode dans le service AlbumService, afin d'effectuer une recherche sur les titres des albums à partir d'un mot saisi dans le formulaire. Vous appelerez cette méthode search dans votre service.

Testez votre méthode avec un simple console.log dans le component SearchComponent dans un premier temps.

Indications : pensez à faire de l'injection de dépendance pour votre service AlbumService.

Exercice 19

Mettez en place maintenant la communication entre l'enfant et le parent et essayez d'afficher le résultat de votre recherche dans la page principale : al-

bums.component.html

Indications : revoyez ce que nous avons déjà réalisé à ce sujet : ici l'enfant doit émettre le résultat de la recherche au parent.

Exercice 20

En utilisant l'attribut directive **ngModelChange** qui permet de détecter les changements du modèle ngModel et la syntaxe **banana in a box** [(ngModel)] pour lier la propriété word et une fois la recherche effectuée remettre à jour les albums sur la page principale (AlbumsComponent).

La syntaxe **banana in a box** permet de lier la propriété "word" dans les deux sens vue et TypeScript : data-binding two-way.

Indications : utilisez un EventEmitter pour signifier au Component AlbumsComponent qu'il faut mettre à jour les albums.

```
<input name="word" [(ngModel)]="word" (ngModelChange)="onChangeEmit($event)" />
```