Chapitre 16 Reactive Forms

Nous allons voir maintenant une autre manière de gérer les formulaires. Celle-ci est plus souple et dynamique.

Notez que nous avons importé déjà ReactiveForms dans le ShareModule.

Un formulaire réactive est de type FormGroup, il regroupe plusieurs FromControl permettant d'interargir avec les inputs du formulaire depuis le TypeScript.

Exercice 8

Créez le formulaire d'ajout d'un album : AddAlbumComponent dans le module AdminModule.

Créez également la route suivante dans le module AdminModule (attention au dépendance). Pensez également à ajouter un bouton "Add album" dans la page admin listant les albums à administrer :

```
// Dans l'AdminModule définition des routes
const routes: Routes = [
  { path: 'admin/add', canActivate: [GuardService], component: AddAlbumComponent },
]
@NgModule({
  imports: [
    CommonModule,
    ShareModule,
    RouterModule.forChild(routes) // définition des routes dans le sous-module
  ],
  declarations: [AlbumComponent, AddAlbumComponent],
  exports: [AlbumComponent, RouterModule]
})
export class AdminModule { }
```

Puis injectez les dépendances suivantes : FormGroup, FormBuilder, FormControl et Validator. Importez également AlbumService que nous utiliserons.

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, Validators, FormControl, FormGroup } from '@angular/forms';
import { AlbumService } from '../../album.service';

@Component({
  selector: 'app-add-album',
  templateUrl: './add-album.component.html',
  styleUrls: ['./add-album.component.scss']
})
```

```
export class AddAlbumComponent implements OnInit {

  constructor(private fb : FormBuilder, private aS : AlbumService) { }

  ngOnInit() {
  }

}
```

Définissez maintenant le formulaire en relation avec le TypeScript :

```
<form [formGroup]="albumForm" >
</form>
```

Pour définir un champ input (par exemple pour l'input name dans notre formulaire) vous devez écrire :

```
<form [formGroup]="albumForm" >
 <div class="form-group">
    <label for="name">Name :</label>
    <input formControlName="name" type="text" class="form-control"
    id="name" aria-describedby="emailHelp" placeholder="Enter artist name">
    <!-- Permet de valider les règles définies dans le TypeScript -->
    <div *ngIf="name.invalid && name.touched" class="alert alert-danger">
      <div *ngIf="name.errors.required">
        Name is required
      </div>
      <div *ngIf="name.errors.minlength">
        Name must be at least 5 characters long.
      </div>
    </div>
  </div>
</form>
```

Dans le TypeScript il faudra relier ce formulaire de la manière suivante. Définissez des méthodes get pour chaque input comme dans l'exemple suivant, ces méthodes sont utilisées par Angular pour récupérer les données de validation dans le TypeScript.

```
this.fb.group({
    name : new FormControl('', [
          Validators.required,
          Validators.minLength(5)
        ]),
})

  get name() {
    return this.albumForm.get('name');
  }
```

Terminez l'exercice en reliant le formulaire au TypeScript comme dans l'exemple donné ci-dessus. Créez une méthode onSubmit pour récupérer le contenu du formulaire dans le TypeScript :

```
onSubmit() {
    console.log(this.albumForm.value['name'])
}
```

Exercice 9 (ajout)

Faites la mise à jour côté base de données à l'aide du service AlbumService. Dans le module HttpClient utilisez la méthode post et passez un objet de type Album; Firebase enregistrera ce nouvel album dans notre base de données music.

```
// ...

  addAlbum(album: Album): Observable<void> {
    return this.http.post<void>(this.albumsUrl + '/.json', album);
  }
```

Gérez enfin la redirection vers la page admin dans la méthode complete de l'Observable et renvoyez par la méthode addAlbum de notre service AlbumService :

```
// Dans le component AddAlbumComponent
this.aS.addAlbum(album).subscribe(
      album => { console.log(album) },
      error => console.error(error),
      () => {
        this.router.navigate(['/admin'], { queryParams: { message: 'success' } });
      }
    );
```

Exercice 10 (mise à jour)

Vous allez maintenant gérer la mise à jour d'un Album. Ajoutez un bouton update pour chaque album dans cette page.

Récupérez l'identifant créé par Firebase, puis enregistrez celui-ci dans la propriété id de l'Album.

```
map(albums => {
        let Albums: Album[] = [];
        _.forEach(albums, (v, k) => {
          v.id = k;
          Albums.push(v);
```

```
  });

  return Albums;
}),
```

Exercice 11 (suppression)

Ajoutez un bouton de suppression dans l'administration des albums et rendez fonctionnelle la suppresion d'un ablum. Vous utiliserez la méthode delete du service HttpClient.