HttpClient

Introduction

Créez une branche spécifique pour cette partie. En effet, nous allons utiliser maintenant un module pour communiquer avec une base de données distante. Nous devrons modifier l'approche pour interroger les données, ceci aura des impacts dans le service AlbumService et les components suivants : AlbumsComponent, AlbumDetailsComponent, AlbumDescriptionComponent.

```
# vérification du statut de la branche actuelle
git status

# fixez les modifications déjà apportées dans votre application
git add .
git commit -m "Application sans base de données distante"

# création de la branche httpclient et déplacement sur celle-ci
git checkout -b httplcient
```

Httpclient

Ce module basé sur RxJS permet d'interroger des serveurs distants à l'aide du protocole HTTP. Il utilise l'interface XMLHttpRequest exposée par le navigateur.

HttpClient est une couche d'abstraction pour la consommation de requêtes HTTP : testabilité, gestion des erreurs, objets Request et Response.

Importez le module dans AppModule de l'application :

```
import { HttpClientModule } from '@angular/common/http';

// N'oubliez pas de le définir dans les imports

 imports: [
    BrowserModule,
    FormsModule,
    RouterModule.forRoot(albumsRoutes),
    HttpClientModule, // module HttpClient
  ],
```

Création de la base de données dans Firebase

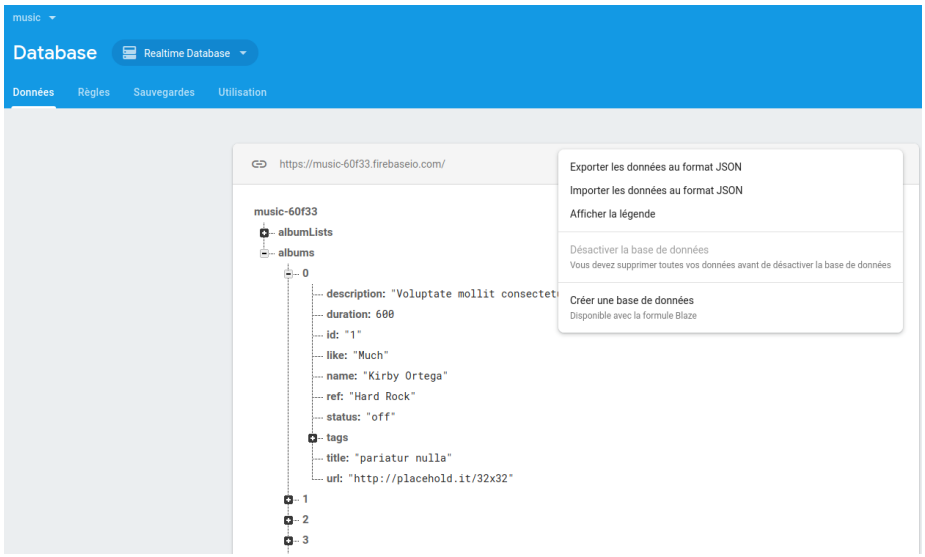
Créez une base de données Firebase "music", définissez les règles suivantes pour la base de données music de lecture et d'écriture (règles) :

```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

Importez les données dans Firebase

Vous allez maintenant importer les données dans Firebase et récupérer dans ce chapitre le dossier source data et le fichier albums.json. Ces données sont formatées aux données que nous utilisons déjà dans notre application.

Importez ces données dans Firebase : onglet “Données” :



Service HttpClient

Il faut maintenant faire une injection de dépendance dans le service AlbumService :

Imports

```
// Service et classe utile
import { HttpClient, HttpHeaders } from '@angular/common/http';
// Opérateurs de RxJS
import { map } from 'rxjs/operators';
// libraire utile pour le traitement de données
import * as _ from 'lodash';
```

```
// définition des headers
const httpOptions = {
  headers: new HttpHeaders({
    'Content-Type': 'application/json',
  })
};
```

L'injection de dépendance dans le service se fera de manière classique.

```
constructor(private http: HttpClient) { }
```

Définition des routes

Vous devez également définir les routes permettant d'accéder aux données sur le serveur de base de données Firebase dans votre service AlbumService.

```
// Service AlbumService

@Injectable({
  providedIn: 'root'
})
export class AlbumService {

  // convention dans l'API ajoutez votre identifant de base de données
  private albumsUrl = 'https://music-[VOTRE_IDENTIFIANT].firebaseio.com/albums';
  private albumListsUrl = 'https://music-[VOTRE_IDENTIFIANT].firebaseio.com/albumLists';
```

Exercice 1

Voici comment vous allez refactoriser la récupération des données, nous vous donnons deux exemples vous ferez le reste de la factorisation pour toute l'application. Les méthodes switchOn et switchOff seront faites dans l'exercice numéro 2.

Voyez les exemples sur les méthodes getAlbums et getAlbum suivantes et refactorisez les autres méthodes de l'application pour que tout refonctionne comme avant. Attention nous utilisons lodash qui est une libraire JS; elle nous permet ici de préparer les données récupérées depuis Firebase pour les exploiter correctement dans notre application.

```
@Injectable({
  providedIn: 'root'
})
export class AlbumService {
```

```
  // ...

  getAlbums(): Observable<Album[]> {

    return this.http.get<Album[]>(this.albumsUrl + '/.json', httpOptions).pipe(
      // Préparation des données avec _.values pour avoir un format exploitable dans l'appli
      map(albums => _.values(albums)),
      // Ordonnez les albums par ordre de durées décroissantes
      map(albums => {
        return this._albums.sort(
          (a, b) => { return b.duration - a.duration }
        );
      })
    )
  }

  getAlbum(id: string): Observable<Album> {

    // URL/ID/.json pour récupérer un album
    return this.http.get<Album>(this.albumsUrl + `/${id}/.json`).pipe(
      map(album => album) // JSON
    );
  }

  // ...
}
```

Dans le component AlbumsComponent vous devez maintenant souscrire pour récupérer les données :

```
// ...
export class AlbumsComponent implements OnInit {

  titlePage: string = "Page princiaple Albums Music";
  albums: Album[] = ALBUMS;
  selectedAlbum : Album;
  status: string = null;
  perPage : number = 5;

  constructor(private ablumService: AlbumService) {

    // récupération des données depuis Firebase avec la méthode HttpClient
    console.log(this.ablumService.getAlbums().subscribe(
      albums => console.log(albums)
    ))
  }
```

```
  // ...

}
```

Terminez cet exercice en refactorisant les autres méthodes. Considérez les remarques pour les méthodes suivantes.

```
  map(albums => {
      if (word.length > 2) {
      let response = [];
      // lodash
      _.forEach(album => {
        if (album.title.includes(word)) response.push(album);
      });

      return response;
    })
```

Pour le component PaginateComponent, vous devrez refactoriser la méthode init comme suit :

```
init(page: number = 1) {
    // lorsqu'on a à disposition le nombre d'albums depuis la base de données :
    this.aS.count().subscribe(count => {
      this.perPage = this.perPageElement;
      this.setParameters(count, page);
      this.pages = [];

      for (let i = 1; i < this.numberPages + 1; i++) {
        this.pages.push(i);
      }
    })
  }
```

Exercice 2

Nous allons maintenant refactoriser les méthodes switchOn et switchOff.

Nous devons mettre à jour dans Firebase le paramètre “status” de l’album lorsqu’on a cliqué sur le bouton player. Pour faire cela nous utiliserons la méthode put du service HttpClient d’Angular. Dans Firebase on utilise la convention d’url suivante : url.json/{id}. La méthode put prend en paramètre l’url et l’objet à modifier.

```
 switchOn(album: Album): void {
    album.status = 'on';
    this.http.put<void>(this.albumsUrl + `/${album.id}/.json`, album).subscribe(
```

```
      e => e,
      error => console.warn(error),
      () => {
        this.subjectAlbum.next(album);
      }
    );
  }
```

Terminez l'exercice en refactorisant la méthode switchOff également.