Lazy Module

Exercice 1

Créez le module playlist avec l'option suivante :

```
ng g m playlist --routing
```

Maintenant créez le compoment music-list dans ce module :

```
ng g c playlist/music-list
```

Ajoutez la route "playlist" dans le menu principal, celle-ci permettra de charger le module playlist. Dans le routing de l'application vous allez ajouter la route suivante :

```
// app-routing.module.ts
 {
    path: 'playlist',
    loadChildren: './playlist/playlist.module#PlaylistModule'
  },
```

Dans le module playlist vous n'avez pas à exporter le compoment MusicList-Component pour le rendre accessible dans l'application. Le module chargera le compoment de manière "lazy" seulement si vous cliquez sur la route playlist définie dans l'application.

```
// dans le module routing de module playlist
const routes: Routes = [
  {
    path : '',
    component : MusicListComponent
  }
];
```

Si vous cliquez sur le lien playlist vous devriez voir maintenant s'afficher la page suivante :

Exercice 2 service

Vous allez maintenant créer un service que nous utiliserons que dans le module PlaylistModule :

```
ng g s playlist/music
```

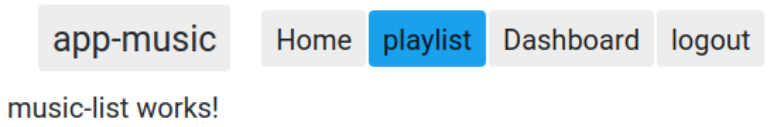

Figure 1: Lazy module

Nous allons faire en sorte que le service soit disponible uniquement dans le component MusicListComponent.

Définissez le service MusicService, sans utiliser la propriété providedIn, nous préciserons son utilisation dans le component lui-même.

Voici le service que nous souhaitons définir, il possèdera une méthode musics qui retournera sous forme d'un Observable la liste des chansons de la playlist (utilisez l'obersvable of) ainsi qu'une méthode setColor qui permettra de changer la couleur au clique dans le component.

```
import { Injectable } from '@angular/core';
import { PlaylistModule } from './playlist.module'
import { Observable, of } from 'rxjs';

// définir un type avec enum
export enum Color {
  Violet = "violet",
  Orange = "orange",
}

export interface Music { id: number; name: string; color: Color }

@Injectable()
export class MusicService {

  private _musics: Music[] = [
    { id: 11, name: "Future Legend", color: Color.Violet },
    { id: 12, name: "Diamond Dogs", color: Color.Violet },
```

```
    { id: 13, name: "Sweet Thing", color: Color.Violet },
    { id: 14, name: "Candidate", color: Color.Violet },
    { id: 15, name: "Rock 'n' Roll with Me", color: Color.Violet },
    { id: 16, name: "1984", color: Color.Violet },
    { id: 17, name: "Big Brother", color: Color.Violet },
  ];

  // todo implémentez les méthodes demandées
}
```

Dans le component déclarez le service ci-dessus comme suit :

```
import { Component, OnInit } from '@angular/core';
import { MusicService, Music, Color } from '../music.service';
import { Observable } from 'rxjs';

@Component({
  selector: 'app-music-list',
  templateUrl: './music-list.component.html',
  styleUrls: ['./music-list.component.scss'],
  providers : [MusicService]
})
export class MusicListComponent implements OnInit {
  // todo injectez le service dans le component
}
```

Exercice 3 création d'une directive

Une directive permet d'agir sur le template lui-même, la directive que nous voulons développer ici est une directive qui permet de changer la couleur du survol des chansons dans la playlist.

Vous allez maintenant créer cette directive spécifique à notre nouveau module PlaylistModule. Pensez à vérifier que cette directive est bien déclarée dans ce module.

```
ng g d playlist/music-tag
```

Créez une méthode tag dans la liste des musiques de la playlist, partie component, une fois que l'on aura cliqué sur cette méthode changez la couleur de survol de la chanson.

Pour faire cela vous vous aiderez de la documentation officielle suivante d'Angular : directive sur la création d'une directive.

Voici ci-dessous pour vous aidez une idée du rendu que l'on souhaite obtenir :

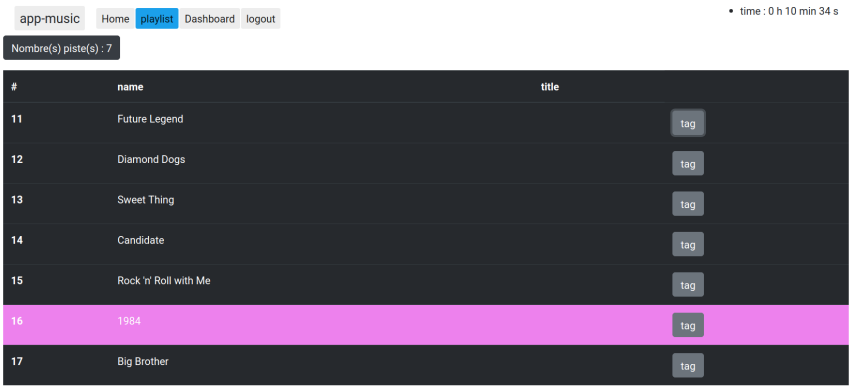

Figure 2: Lazy module

Pipe

Un pipe Angular est une fonction que l'on applique à une valeur dans le template, notez qu'un pipe peut avoir ou non des paramètres, les pipes peuvent également se chaîner :

```
<p>Date du dernier album {{ publication | date:"MM/dd/yy" }} </p>

<p>Date du dernier album {{ publication | date | uppercase }} </p>
```

Vous pouvez également créer une méthode dans le component qui s'appliquera sur le pipe :

```
get format()   { return this.isChangedFormat ? 'shortDate' : 'fullDate'; }

toggleFormat(){
  this.isChangedFormat != this.isChangedFormat;
}

```

La méthode format peut alors être passer en argument du pipe :

```html
```

```
<p>Date du dernier album {{ publication | date:format }} </p>
```

Exercice 4

Créez un pipe pretty avec un paramètre color (pretty:super) qui permet de mettre des CSS sur un titre ou un texte.

Les paramètres de votre pipe seront les valeurs suivantes possibles, vous ajouterez une couleur différente par mot, si le nombre de mot est supérieur au nombre de couleurs vous répéterez les mêmes couleurs de la liste en recommençant du début de la liste de couleurs :

```
// Dans votre pipe

const styles = ['super', 'normal', 'star'];
const colors =
{
  "super": ['#39CCCC', '#AAAAAA', '#FFDC00'],
  "normal": ['#FF851B', '#FF4136', '#85144b'],
  "star": ['#B10DC9', '#AAAAAA', '#FFDC00']
};
```

Vous testerez avant d'appliquer le pipe sur l'élément que ce dernier est bien une chaîne de caractères, le pipe ne s'appliquera pas dans le cas contraire.