Ng Projet Angular 2 – Départements et communes

1 – Création de son environnement

On lance la commande **ng new geofrance** pour créer notre projet
Angular nous demande si on veut ajouter le routing dans notre projet dans notre cas on sélectionne avec Y pour valider
Ensuite on choisit notre format de stylesheet, on prend CSS.
On se rend ensuite dans le dossier geofrance grâce à ***cd geofrance***

À installer :

https://github.com/scttcper/ngx-toastr/blob/HEAD/src/lib/toastr.css

- npm install ngx-toastr

- ng add @ng-bootstrap/ng-bootstrap
- Npm i ngx-bootstrap
- ng add @fortawesome/angular-fontawesome

https://swimlane.gitbook.io/ngx-charts/installing

- npm install @swimlane/ngx-charts

On va créer un dossier components pour stocker tous nos composants et un dossier models pour y stocker nos modèles

Attention : Ces dossiers doivent être créés à l'intérieur de src/app

Mkdir components et **mkdir models** (ils sont au même niveau)
Ensuite il faut naviguer à l'intérieur de components (**cd components**)

2 – Création des composants et modules

On va créer un composant qui va se nommer commune
Ng g c commune
Une fois générer il faut ajouter le routing dans le fichier **app-routing.ts** dans la **const routes : Routes = [{path : 'commune', component : CommuneComponent}] ;**
Une fois que cela est fait nous allons commencer par créer votre navbar comme lors du précédent TP pour que l'utilisateur puisse naviguer dans l'application

3 – Récupérer les API du gouvernement

Nous allons utilisés deux web services qui sont :
https://geo.api.gouv.fr/departements : Pour lister tous les départements en France
https://geo.api.gouv.fr/departements/{code}/communes : Pour lister les communes d'un département.

Attention {code} doit être remplacé par le numéro du département
Nous allons nous rendre dans notre dossier models pour créer un nouveau model
Etant donné que nous préférons utiliser des objets flexibles nous allons utiliser des interfaces et non des types
Ng g interface commune –type=model et **ng g interface departement –type=model**

Grâce aux web services, nous allons pouvoir déterminer notre model pour les départements et les communes
Dans le fichier **departement.model.ts**

```
export interface Departement {
    nom: string;
    code: string;
    codeRegion: string;

}
```

Et dans commune.model.ts

```
export interface Commune {
    nom: string;
    code: string;
    codeDepartement: string;
    codeRegion: string;
    codePostaux: string[];
    population: number

}
```

Dans le composant **commune.component.ts** on va faire appel au module HttpClient dans le constructor tout en mettant sa visibilité en private

ATTENTION NE PAS OUBLIER D'IMPORTER HttpClientModule dans le fichier app.module.ts

En dehors du constructor nous allons créer une méthode qui s'appelle loadDepartements() : void{} qui va nous permettre de récupérer nos départements

```
export class CommuneComponent implements OnInit {

  departements: Departement[] = [];
  departementIsLoading: boolean = false;
  departementIsLoaded: boolean = false;


  constructor(private HttpClient: HttpClient) { }

  loadDepartements(): void{
    this.departementIsLoading = true;
    this.HttpClient.get<Departement[]>('https://geo.api.gouv.fr/departement').
subscribe(
      data => {
        this.departements = data;
        this.departementIsLoaded = true;
        this.departementIsLoading = false;
      }
```

```
    )
  }

  ngOnInit(): void {
  }

}
```

Une fois que nous avons codé le composant pour le chargement des départements, il faut bien à un moment donné que nous créons notre vue

4 – Création de la vue pour les départements.

Créer un nouveau component **departement-table.component.ts**
Ce component doit prendre en compte la liste des départements et les deux boolean de chargement
En sortie il doit prendre également **loadDepartements**

```
import { Component, OnInit, Input,Output, EventEmitter } from '@angular/core';
import { Departement } from 'src/app/models/departement.model';

@Component({
  selector: 'app-departement-table',
  templateUrl: './departement-table.component.html',
  styleUrls: ['./departement-table.component.css']
})
export class DepartementTableComponent implements OnInit {

  @Input() departements: Departement[] = [];
  @Input() departementsIsLoading: boolean = false;
  @Input() departementsIsLoaded: boolean = false;
  @Output() loadDepartements: EventEmitter<{}> = new EventEmitter();

  ngOnInit(){

  }

  getDepartements(): Departement[]{
    return this.departements

  }

}
```

On va se rendre dans la vue (**departement-table.component.html**)

```
<div *ngIf="departementsIsLoading">
    <span class="fas fa-circle-notch fa-spin fa-3x"></span>

</div>
```

```
<button *ngIf="!departementsIsLoaded" (click)="loadDepartements.emit()"
class="btn btn-primary" type="action">
   Charger les départements
</button>

<ng-container *ngIf="departementsIsLoaded">

<table class="table">

    <thead>
        <tr>
            <th scope="col">Nom</th>
            <th scope="col">Code</th>
            <th scope="col">Code région</th>
            <th scope="col">#</th>
        </tr>
    </thead>
    <tbody>
        <tr *ngFor="let departement of getDepartements()">
            <td> {{ departement.nom }} </td>
            <td> {{ departement.code}} </td>
            <td> {{ departement.codeRegion }}</td>
            <td>#</td>
        </tr>

    </tbody>

</table>

</ng-container>
```

6 – Contrôler le scroll avec une pagination

Pour pouvoir contrôler la navigation de notre site, on va utiliser la pagination que nous avons via *ngx-bootstrap*
Nous allons utiliser le component pagination pour ajouter une pagination sur notre tableau de départements. Par défaut la pagination se fait de 10 en 10. Il prend en paramètre le nombre total du tableau à paginer **totalItems** ainsi qu'une variable **currentPage** qui nous permettra de connaître la page actuelle.

Nous allons également utiliser la fonction JS **slice()** qui permet de découper un tableau. Cette fonction prend deux paramètres : la position du début et la position de fin.
Fichier **departement-table.component.ts**

```
  getLength(): number{
    return this.departements.length;
  }
```

```
  getDepartements(): Departement[]{
    return this.departements.slice((this.currentPage - 1 )   * 10,
this.currentPage * 10 )

  }
```

7 – Ajouter une barre de recherche

Pour filtrer un tableau en JS on utilise **filter()**
On va ajouter un input dans le fichier **departement-table.component.html**

```
<div>
    <label for="search" class="form-label">Rechercher un département</label>
    <input type="text" class="form-control" name="search"
placeholder="Bouches-du-Rhône" [(ngModel)]="search" autocomplete="off" >
</div>
```

Ajouter un champ input de recherche au-dessus du tableau des départements
Lier une nouvelle propriété search du component **departement-table.component.ts** au formulaire de recherche
Ajouter un filtre sur le tableau des départements en fonction de la valeur de search

```
  getLength(): number {
    return this.departements
        .filter(departement =>
departement.nom.toLowerCase().includes(this.search.toLowerCase()))
        .length;
}

getDepartements(): Departement[] {
    return this.departements
        .filter(departement =>
departement.nom.toLowerCase().includes(this.search.toLowerCase()))
        .slice((this.currentPage - 1) * 10, this.currentPage * 10)
}
```