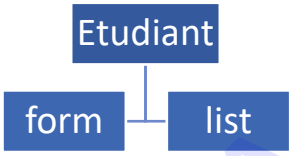
Angular version 17

Réalisé par : Adil ENAANAI

Partie I : Formulaire d'ajout

Etape1 : Créer la liste des étudiants

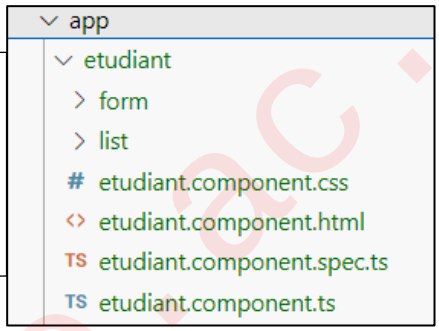
Créer la structure des composants suivants :



Procédure :

Exécuter les commandes suivantes :

```
cd myApp
ng g c Etudiant
cd src/app/Etudiant
ng g c list
ng g c form
```



Accéder au fichier « `app.routes.ts` »

Et ajouter la route suivante :

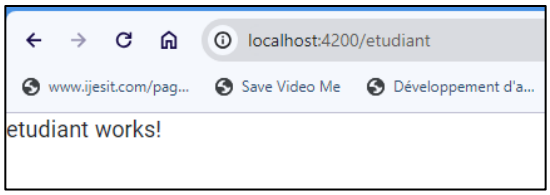
```
import { Routes } from '@angular/router';
import { EtudiantComponent } from './etudiant/etudiant.component';

export const routes: Routes = [
    {path:"etudiant",component:EtudiantComponent}
];
```

Ensuite, ajouter la balise `<router-outlet></router-outlet>` dans le body du fichier « app.component.html »

Lancer l'application avec : **`ng s -o`**

Et tester



Création d'un service :

Pour partager les données entre les différents composants au sein de Etudiant, il faut créer un service interne avec la commande : `IWAProject\myApp\src\app\etudiant> ng g s etudiant`

Voilà le code de la classe du service :

```
export class EtudiantService {
  etudiants=[
    {"id":1,"nom":"MANSOURI","age":23},
    {"id":2,"nom":"HOUSSNI","age":22},
    {"id":3,"nom":"BAKKALI","age":24},
  ]
  constructor() {}
```

Adil ENAANAI
aenaanai@uae.ac.ma

```
  getEtudiants(){
    return this.etudiants;
  }
}
```

La méthode getEtudiants() permet de renvoyer tous les étudiants à partir du service.

Maintenant, affichons la liste des étudiants dans le composant « list ». Pour cela, nous avons besoin d'ajouter bootstrap avec la commande : **`npm install bootstrap`**

Ensuite, modifier la propriété « styles » du fichier « angular.json » en ajoutant « `"node_modules/bootstrap/dist/css/bootstrap.min.css",` »

avant `"src/styles.css"`

Arrêter et démarrer Angular pour que la nouvelle configuration prenne effet.

Lancer une recherche internet sur « bootstrap table » et accéder au lien qui présente les tableau bootstrap.

Prenez celui nommé : **Striped rows** et coller son code dans la vue du composant liste.

Changer la classe « ListeComponent » comme suit :

```
@Component({
  selector: 'app-list',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './list.component.html',
  styleUrl: './list.component.css'
})
export class ListComponent {
  etudiants:any
  constructor(private etudiantService: EtudiantService){}
  ngOnInit() {
  this.etudiants=this.etudiantService.getEtudiants()
  }
}
```

N'oublie pas d'ajouter ce module afin de pouvoir utiliser ngFor

Puis la vue du composant « **list** »

```
<table class="table table-striped">
    <thead>
      <tr>
        <th scope="col">Id</th><th scope="col">Nom</th><th scope="col">age</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let etudiant of etudiants">
      <td>{{etudiant.id}}</td><td>{{etudiant.nom}}</td><td>{{etudiant.age}}</td>
      </tr>
    </tbody>
  </table>
```

Ensuite, modifier la vue du composant Etudiant en ajoutant un bouton et un textField bootstrap comme suit :

Adil ENAANAI
aenaanai@uae.ac.ma

```
<div>
    <button (click)="openModal()" id="openModal" type="button" class="btn btn-
primary">Ajouter</button>
</div>
<app-list></app-list>
```

Avec un peu de CSS 😊

```
#searchByName{
width: 20%;
}
#openModal{
    float: right;
}
```

Tester pour avoir le résultat suivant :

Chercher par nom

Ajouter

Id	Nom	age
1	MANSOURI	23
2	HOUSSNI	22
3	BAKKALI	24

Etape2 : Créer le formulaire d'ajout

Pour créer le formulaire d'ajout, faites une recherche internet sur « bootstrap form ». Puis, prenez « **Form grid** »

Coller le code dans la vue du composant « form » et faire quelques modifications comme suit :

```
<form>
    <div class="col">
      <div class="col">
        <input type="text" class="form-control" placeholder="Id">
      </div>
      <div class="col">
        <input type="text" class="form-control" placeholder="Nom">
      </div>
      <div class="col">
        <input type="text" class="form-control" placeholder="Age">
      </div>
      <div class="col">
        <button type="button" class="btn btn-primary">Ajouter</button>
      </div>
    </div>
  </form>
```

Etape3 : Ouvrir le composant « form » en tant que Modal

Pour ouvrir une fenêtre de type Modal, ajouter d'abord l'action (click) au bouton « ajouter » du composant « Etudiant ». Mais avant tous, il faut installer le package « @ng-bootstrap/ng-bootstrap » avec la commande :

```
ng add @ng-bootstrap/ng-bootstrap
```

Adil ENAANAI
aenaanai@uae.ac.ma

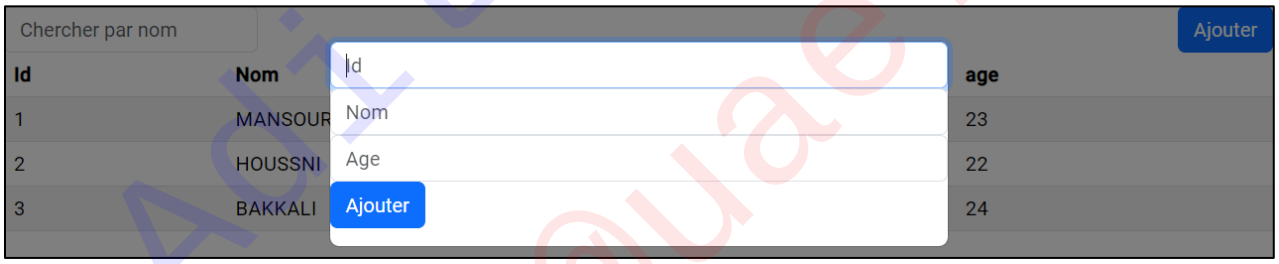
```
<input (click)="openModal()" id="searchByName" type="text" class="form-control"
placeholder="Chercher par nom">
```

Puis, créer la méthode « openModal » dans la classe « EtudiantComponent »

```
import { Component } from '@angular/core';
import { ListComponent } from "./list/list.component";
import { NgbModal } from '@ng-bootstrap/ng-bootstrap';
import { FormComponent } from './form/form.component';

@Component({
    selector: 'app-etudiant',
    standalone: true,
    templateUrl: './etudiant.component.html',
    styleUrl: './etudiant.component.css',
    imports: [ListComponent]
})
export class EtudiantComponent {
    constructor(private modal:NgbModal){}
    openModal(){
        this.modal.open(FormComponent)
    }
}
```

Tester pour voir l'interface suivante :



Un peu de CSS est nécessaire 😊

```
button{
    float: right;
}
form{
    padding: 10%;
    margin-left: 10%;
    width: 80%;
}
input{
    margin-top: 5%;
    margin-bottom: 5%;
}
h2{
    text-align: center;
}
#fermer{
    float: right;
}
```

Adil ENAANAI
aenaanai@uae.ac.ma

Et enfin 😊

Etape4 : Ajouter un nouvel étudiant à la liste

Pour pouvoir ajouter un nouvel étudiant à la liste, nous allons utiliser d'abord le module « ReactiveFormModule » qu'on doit importer dans la classe « formComponent ». Puis ajouter la méthode « ajouterEtudiant() » comme suit :

```
import { Component } from '@angular/core';
import { FormBuilder, ReactiveFormsModule, Validators } from '@angular/forms';
import { EtudiantService } from '../etudiant.service';
import { NgbActiveModal } from '@ng-bootstrap/ng-bootstrap';
@Component({
  selector: 'app-form',
  standalone: true,
  imports: [ReactiveFormsModule],
  templateUrl: './form.component.html',
  styleUrl: './form.component.css'
})
export class FormComponent {
  formEtudiant=this.fb.group({
    "id":['',Validators.required],
    "nom":['',Validators.required],
    "age":['',Validators.required]
  })
constructor(private fb:FormBuilder,private etudianService:EtudiantService, private
activeModal:NgbActiveModal){}
ajouterEtudiant(){
  this.etudianService.addEtudiant(this.formEtudiant.value)
  this.activeModal.close()
}
fermer() {
  this.activeModal.close()
  }
}
```

Il faut ajouter la méthode « addEtudiant » au service

Ajouter la méthode addEtudiant au service

```
  addEtudiant(etudiant:any){
    this.etudiants.push(etudiant)
  }
```

Adil ENAANAI
aenaanai@uae.ac.ma

Puis, modifier la vue du composant « form » comme suit :

```
<button id="fermer" (click)="fermer()" type="button" class="btn btn-danger"
>X</button>
<form [formGroup]="formEtudiant">
    <H2>Ajouter un étudiant</H2>
    <div class="col">
      <div class="col">
        <input formControlName="id" type="text" class="form-control"
placeholder="Id">
      </div>
      <div class="col">
        <input formControlName="nom" type="text" class="form-control"
placeholder="Nom">
      </div>
      <div class="col">
        <input formControlName="age" type="text" class="form-control"
placeholder="Age">
      </div>
      <div class="col">
        <button (click)="ajouterEtudiant()" type="button" class="btn btn-primary"
[disabled]="!formEtudiant.valid">Ajouter</button>
      </div>
    </div>
  </form>
```

C'est le formGroup qu'on a déclaré en haut

C'est la propriété id du formGroup qu'on a déclaré en haut

C'est la propriété nom du formGroup qu'on a déclaré en haut

C'est la propriété age du formGroup qu'on a déclaré en haut

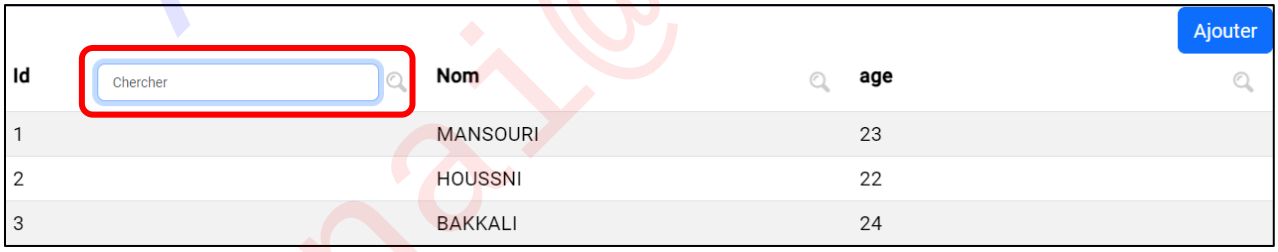
Pour désactiver le bouton si le formulaire n'est pas

Tester votre application

Partie II : Filtrage des données

Etape1 : Créer le composant de recherche

Avant de commencer, on présente l'interface graphique souhaitée



L'élément encadré en rouge est un nouveau composant que nous allons créer. On lui donne le nom « loupe »

Créer le composant loupe à l'intérieur du composant « list » avec la commande : **`ng g c loupe`**

Voilà le code de la vue :

```
<div (mouseover)="afficherInput()" (mouseout)="masquerInput()">
<img src="./assets/images/loupe.png" >
<input id="search" type="text" class="form-control" placeholder="Chercher"
[(ngModel)]="value" [style.opacity]="opacity">
</div>
```

ngModel est liée à l'attribut « value » de la classe LoupeComponent. Donc, la variable value et l'élément <input> ont toujours la même valeur

Adil EMMANAI
aemmanai@uae.ac.ma

Il faut chercher une petite image sur internet qui présente une loupe avec arrière-plan transparent. Puis l'enregistrer dans un nouveau dossier « images » dans le dossier « assets » existant dans le projet.

Ensuite, ajouter le code suivant au fichier CSS

```
img{
    width: 30px;
    height: 30px;
    float: right;
    opacity: 0.2;
}
img:hover{
    opacity: 1;
    cursor: pointer;
}

#search{
    width: 70%;
    float: right;
    font-size: 11px;
    }
```

En encore le fichier de la classe TypeScript :

```
import { Component } from '@angular/core';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-loupe',
  standalone: true,
  imports: [FormsModule],
  templateUrl: './loupe.component.html',
  styleUrl: './loupe.component.css'
})
export class LoupeComponent {
    opacity:string
    value:string
  constructor(private listComponent:ListComponent){
    this.opacity="0"
    this.value=""
  }

  afficherInput(){
    this.opacity="1"
  }
  masquerInput(){
    if ((this.value==undefined)||(this.value==""))
    this.opacity="0"
  }
}
```

Pour ajouter le composant « loupe » à la liste voilà ce qu'il faut modifier :

Adil ENAANAI
aenaanai@uae.ac.ma

```
<table class="table table-striped">
    <thead>
      <tr>
        <th scope="col">><app-loupe #idLoupe></app-loupe>Id</th>
        <th scope="col">><app-loupe #nomLoupe></app-loupe>Nom</th>
        <th scope="col">><app-loupe #ageLoupe></app-loupe>age</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let etudiant of etudiants">
      <td>{{etudiant.id}}</td><td>{{etudiant.nom}}</td><td>{{etudiant.age}}</td>
      </tr>
    </tbody>
  </table>
```

Maintenant, il faut ajouter une méthode de filtrage au service « **EtudiantService** »

```
filterEtudiantsByNom(nom:string){
    return this.etudiants.filter(etudiant =>
      etudiant.nom.toLowerCase().startsWith(nom.toLowerCase()));
  }
```

Puis, appeler cette méthode à chaque fois qu'on appuie sur une touche clavier à l'intérieur de la zone de recherche.

Dans LoupeComponent, injecter listComponent pour pouvoir utiliser sa méthode `ngAfterViewInit()`.

Pour cela, on ajoute le constructeur et la fonction reload() qui sera appelée à chaque fois que l'utilisateur tape un caractère dans la zone de recherche.

```
  constructor(private listComponent:ListComponent){
    this.opacity="0"
    this.value=""
  }
  reload(){
    this.listComponent.ngAfterViewInit()
  }
```

Voilà le code de la vue :

```
<div (mouseover)="afficherInput()" (mouseout)="masquerInput()">
<img src="./assets/images/loupe.png" >
<input id="search" type="text" class="form-control" placeholder="Chercher"
[(ngModel)]="value" [style.opacity]="opacity" (keyup)="reload()">
</div>
```

Lorsque vous tapez un texte à l'intérieur de la zone de recherche, on appelle la méthode `ngAfterViewInit()`
Cette méthode utilise la valeur de la zone de recherche de son composant fils **loupeComponent#nomLoupe**

Pour pouvoir utiliser les attributs et les méthodes d'un composant fils, on utilise l'annotation ***@ViewChild*** comme suit :

Adil ENAANAI
aenaanai@uae.ac.ma

```
import { Component, ViewChild } from '@angular/core';
import { EtudiantService } from '../etudiant.service';
import { CommonModule } from '@angular/common';
import { LoupeComponent } from "./loupe/loupe.component";

@Component({
    selector: 'app-list',
    standalone: true,
    templateUrl: './list.component.html',
    styleUrl: './list.component.css',
    imports: [CommonModule, LoupeComponent]
})
export class ListComponent {
  etudiants:any
  @ViewChild("nomLoupe")  loupeComponent!: LoupeComponent;
  constructor(private etudiantService: EtudiantService){}
  ngAfterViewInit() {
    const nom=this.loupeComponent.value
    if ((nom!="")&&(nom!=undefined)){
      this.etudiants=this.etudiantService.filterEtudiantsByNom(nom)
    }else{
      this.etudiants=this.etudiantService.getEtudiants()
    }

  }
}
```

Pour référencier le composant fils de l'id=nomLoupe

On récupère la valeur de l'attribut « value » du composant fils

Enfin, tester l'application 😊

Partie III : Liaison avec Spring et filtrage de données

Etape1 : Récupération de données

Tout d'abord, on vérifie si le contrôleur mappé par /etudiants dispose d'un endpoint getAllEtudiant(). Sinon, voici le code du contrôleur :

```
@RestController
@CrossOrigin(origins = "*") //N'oublier pas cette ligne pour accepter les requêtes externes
public class StudentController {

      @Autowired StudentService studentService;

      @GetMapping("students")
      public List<Student> getAllStudent(){
            return studentService.getAllStudents();
      }

      @PostMapping("students")
      public Student addStudent(@RequestBody Student student) {
            return studentService.addStudent(student);
      }
      @DeleteMapping("students/{id}")
      public void deleteStudent(@PathVariable Long id) {
            studentService.deleteStudent(id);
      }
}
```

Adil ENAANAI
aenaanai@uae.ac.ma

Ensuite, au niveau d'Angular, on modifier le code du service EtudiantService comme suit :

```
export class EtudiantService {
  backEndURL="http://localhost:8080/students"
  etudiants:any
  constructor(private http:HttpClient) {}
  getEtudiants(){
    this.http.get(this.backEndURL).subscribe(data=>{
      this.etudiants=data
    })
    return this.etudiants
  }
//Autres méthodes
}
```

Le service ci-dessus utilise la méthode GET de l'objet HttpClient qui permet de se connecter à l'url de l'application backEnd. Comme la méthode GET renvoie un observable, il faut appeler la méthode ***subscribe*** afin de lire les données qui circulent à l'intérieur. Ces données (data) sont ensuite affectées à la liste this.etudiants. Cette liste sera renvoyée par la méthode **getEtudiants()**

Le service ci-dessus est injecté dans le constructeur du composant ListComponent pour bénéficier de la méthode **getEtudiants()**. Le résultat de cette dernière est affecté à la liste etudiants du composant ListComponent.

```
export class ListComponent {
  etudiants:any
  @ViewChild("nomLoupe")  loupeComponent!: LoupeComponent;
  constructor(private etudiantService: EtudiantService ){}
  ngAfterViewInit() {
      this.etudiants=this.etudiantService.getEtudiants()
  }}
```

Et enfin, les fichier HTML du composant ListComponent :

```
<table class="table table-striped">
    <thead>
      <tr>
        <th scope="col"><app-loupe #idLoupe></app-loupe>Id</th>
        <th scope="col"><app-loupe #nomLoupe></app-loupe>Nom</th>
        <th scope="col"><app-loupe #ageLoupe></app-loupe>age</th>
        <th></th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let etudiant of etudiants">
      <td>{{etudiant.id}}</td><td>{{etudiant.nom}}</td><td>{{etudiant.age}}</td>
      </tr>
    </tbody>
  </table>
```

Etape2 : Filtrer les données de la liste

Afin de filtrer les données de la liste, on utilise le composant **loupeComponent**. Dans cette étape, nous allons utiliser l'annotation @Output() pour transmettre la valeur de la propriété « valeur » du composant enfant « loupeComponent » vers le composant parent « listComponent ».

Adil ENNAANAI
aennaanai@uae.ac.ma

```
export class LoupeComponent {
  @Output() childEvent = new EventEmitter(); //Emetteur d'événement vers le parent
  opacity:string
  valeur!: string;
  constructor(){
    this.opacity="0"
  }
  afficherInput(){
    this.opacity="1"
  }
  masquerInput(){
    if ((this.valeur==undefined)||(this.valeur==""))
    this.opacity="0"
  }
  filterData(){
    //méthode qui envoie this.valeur vers le composant parent listComponent
    this.childEvent.emit(this.valeur)
  }
}
```

Dans le code ci-dessous, et lors de l'événement **(keyup)**, l'utilisateur tape un nom dans la zone de recherche. Suite à cet événement, on appelle la méthode **filter()** qui envoie la **this.valeur** au composant parent.

```
<div (mouseover)="afficherInput()" (mouseout)="masquerInput()">
<img src="./assets/images/loupe.png" >
<input #search type="text" class="form-control" placeholder="Chercher"
[(ngModel)]="valeur" [style.opacity]="opacity" (keyup)="filterData()" >
</div>
```

Quand on utilise l'annotation **@Output()**, il faut considérer que l'objet annoté (ici childEvent) comme un nouvel événement à ajouter à l'objet enfant lorsqu'il est ajouter en HTML de l'objet parent. Donc, modifier l'appel au sélecteur du composant loupeComponent comme suit :

```
<th scope="col"><app-loupe (childEvent)="filtrer($event)" #nomLoupe></app-
loupe>Nom</th>
```

Explication :

Lorsque l'utilisateur tape un texte dans l'input du composant loupeComponent, on appelle la méthode filterData() suite à l'événement (keyup)

Dans la méthode appelée filterData(), on envoie this.valeur (valeur de l'input par data binding) au composant parent « listComponent » via l'EventEmitter childEbent « `this.childEvent.emit(this.valeur)` »

L'événement est déclenché dans le code

« `<app-loupe (childEvent)="filtrer($event)"` » pour appeler la méthode du composant parent « filtrer » en donnant le paramètre $**event** qui contient **this.valeur** qu'on transmit dans le code de filterData().

La méthode **filtrer** doit être définie dans le composant « **listComponent** »:

Adil ENAANAI
aenaanai@uae.ac.ma

```
filtrer(nom:string){
    this.etudiants=this.etudiantService.filterEtudiantsByNom(nom)
  }
```

Ici, on appelle la méthode « filterEtudiantByNom(nom) » du service « EtudiantService ». Donc, on change ou on ajoute cette méthode au service. Voilà son code :

```
  filterEtudiantsByNom(nom:string){
    if ((nom!="")&&(nom!=undefined)){
     return this.etudiants.filter((
      etudiant: { nom: string;}
      )=>etudiant.nom.toLowerCase().startsWith(nom.toLowerCase())
      )
     }
    return this.etudiants
}
```

Enfin, modifier la méthode `ngAfterViewInit()` comme suit :

```
  ngAfterViewInit() {
      this.etudiants=this.etudiantService.getEtudiants()
  }
```

Etape3 : Suppression de données

Pour supprimer un étudiant, on ajoute une nouvelle colonne à la liste qui contient une icone de suppression. Tout d'abord, il faut chercher sur Internet l'image de l'icône « delete » avec arrière-plan transparent.

Voici un lien utile :

https://cdn3.iconfinder.com/data/icons/softwaredemo/PNG/256x256/DeleteRed.png

Ensuite, Copier/Coller cette icone dans le dossier « asserts/images » de votre projet et l'intégrer au code HTML du composant « listComponent » comme suit :

```
<table class="table table-striped">
    <thead>
      <tr>
        <th scope="col"><app-loupe #idLoupe></app-loupe>Id</th>
        <th scope="col"><app-loupe (childEvent)="filtrer($event)" #nomLoupe></app-
loupe>Nom</th>
        <th scope="col"><app-loupe #ageLoupe></app-loupe>age</th>
        <th></th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let etudiant of etudiants">
      <td>{{etudiant.id}}</td><td>{{etudiant.nom}}</td><td>{{etudiant.age}}</td>
        <td><img class="delete" src="./assets/images/delete.png"
(click)="deleteEtudiant(etudiant)"></td>
      </tr>
    </tbody>
  </table>
```

N'oubliez pas ceci

Adil ENAANAI
aenaanai@uae.ac.ma

Si l'image est trop grande, vous pouvez régler ses dimensions en CSS du présent composant

```
.delete{
    width: 30px;
    height: 30px;
    cursor: pointer;
}
```

Ajouter la méthode `deleteEtudiant` au composant listComponent

```
deleteEtudiant(etudiant:any){
    this.etudiantService.deleteEtudiant(etudiant.id)
    const indexToRemove=this.etudiants.indexOf(etudiant)
    this.etudiants.splice(indexToRemove, 1);
  }
```

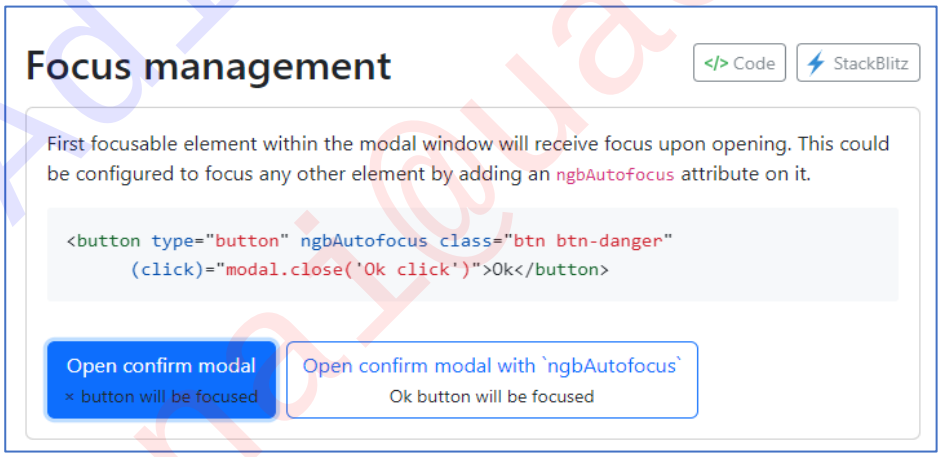
Et bien sûr, modifier EtudiantService en ajoutant la méthode

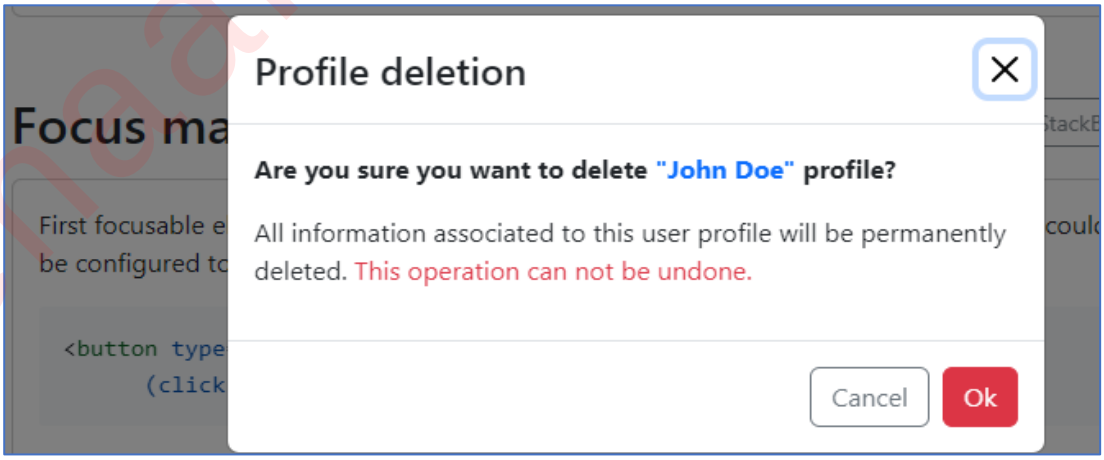
```
  deleteEtudiant(idEtudiant:string){
    this.http.delete(this.backEndURL+"/"+idEtudiant).subscribe()
  }
```

Etape3 : Ajouter une boite de dialogue pour confirmer la suppression

Pour ajouter une boite de dialogue pour confirmer la suppression, nous allons utiliser un composant modal déjà prêt comme celui-ci sur le lien :

https://ng-bootstrap.github.io/#/components/modal/examples





Adil ENAANAI
aenaanai@uae.ac.ma

Pour ce faire, on récupère le code du composant à partir du site pour créer un nouveau composant. Donc, on crée un nouveau composant nommé « confirm » à l'intérieur du composant « listComponent » et on lui adapte le code récupéré du site

```
PS C:\Users\Adil\Desktop\IWAProject\myapp\src\app\etudiant\list> ng g c confirm
CREATE src/app/etudiant/list/confirm/confirm.component.html (22 bytes)
CREATE src/app/etudiant/list/confirm/confirm.component.spec.ts (603 bytes)
CREATE src/app/etudiant/list/confirm/confirm.component.ts (238 bytes)
CREATE src/app/etudiant/list/confirm/confirm.component.css (0 bytes)
```

Voilà le code du composant « confirmComponent »

```
import { Component, Input, inject } from '@angular/core';
import { NgbActiveModal } from '@ng-bootstrap/ng-bootstrap';

@Component({
  selector: 'app-confirm',
  standalone: true,
  imports: [],
  templateUrl: './confirm.component.html',
  styleUrl: './confirm.component.css'
})
export class ConfirmComponent {
  modal = inject(NgbActiveModal)
  @Input() nom:any
}
```

Dans le code ci-dessus, `modal = inject(NgbActiveModal)` est une autre méthode d'injection. On injecte l'objet ngbActiveModal pour utiliser la méthode close qui renvoie une valeur de retour au composant parent. La valeur de retour est : soit 'ok' soit 'cancel'.

L'annotation `@Input() nom:any` est utilisée pour récupérer le paramètre envoyé par le parent lors de l'ouverture du modal. Il contient le nom de l'étudiant à supprimer.

Voilà le code HTML du nouveau composant « confirmComponent »

```
<div class="modal-header">
    <h4 class="modal-title" id="modal-title">Supprimer étudiant</h4>
    <button
        type="button"
        class="btn-close"
        aria-describedby="modal-title"
        (click)="modal.close('fermer')"
    ></button>
</div>
<div class="modal-body">
    <p>
        <strong>Êtes vous sûr de vouloir supprimer l'étudiant <span class="text-
primary">{{nom}}</span></strong>
    </p>
</div>
<div class="modal-footer">
    <button type="button" class="btn btn-outline-secondary"
(click)="modal.close('cancel')">Annuler</button>
```

Adil ENAANAI
aenaanai@uae.ac.ma

```
    <button type="button" class="btn btn-danger"
(click)="modal.close('ok')">Valider</button>
</div>
```

Ensuite, on ajoute la méthode « openModal » au composant « listComponent » comme suit :

```
  openModal(etudiant:any){
   const dialogRef = this.modal.open(ConfirmComponent)
   dialogRef.componentInstance.nom=etudiant.nom
   dialogRef.result.then(data=>{
    if (data=='ok'){
      this.deleteEtudiant(etudiant)
    }
   })
  }
```

Et enfin, on ouvre la fenêtre « modal » en cliquant sur l'icône « delete » du composant « listComponent »

```
<img class="delete" src="./assets/images/delete.png"
(click)="openModal(etudiant)">
```

Adil ENAANAI
aenaanai@uae.ac.ma