CONFIGURACION DE VISUAL CODE Y VISUAL STUDIO PARA CONECTAR BACKEND CON FRONTEND Y AGREGAR SEGURIDAD DE USUARIOS JWT

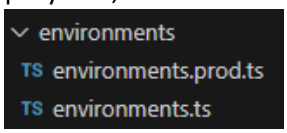
PARTE FRONTEND

1. Crear archivo de servicio para poder llamar las peticiones html.
   En la terminal digitamos: ng g s estudiantes/estudiantes –skip-tests=true.

2. Para poder trabajar con el servicio tenemos que usar el modulo http, el cual lo importamos en el archivo app.modulo.ts

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

3. También tenemos que configurar los archivos environment de desarrollo y de producción, sino aparecen creamos un archivo nuevo dando click derecho sobre la carpeta inicial del proyecto, creamos una carpeta environment y dentro dos archivos.



4. Editamos los archivos y colocamos la configuración adecuada. No colocamos la url de estudiantes para dejarlo genérico por si en un futuro agregamos más entidades y servicio webApi.

```
export const environment ={
    production : true,
    apiURL : 'https://localhost:44383/api/'
}
```

5. En la carpeta que maneja la lógica de la entidad damos click derecho y creamos un archivo estudiante.ts para interpretar la entidad que viene en la petición http. Van a ser los mismos DTO que hacemos en visual studio.

```
export interface  estudiantesDTO{
    id: number;
    nombre: string;
    fechaNacimiento: Date;
    sexo: string;
    cedula: string;
    carrera: string;
}

export interface estudiantesCreacionDTO{
    nombre: string;
    fechaNacimiento: Date;
    sexo: string;
    cedula: string;
    carrera: string;
}
```

6. Vamos al archivo de estudiantes.service.ts para programar todas las conexiones e importamos environment y httpclient.

```
import { Injectable } from '@angular/core';
import { estudiantesCreacionDTO, estudiantesDTO } from './estudiante';
import { HttpClient } from '@angular/common/http';
import { environment } from 'src/environments/environments';
import { Observable } from 'rxjs';

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

  constructor(private http: HttpClient) { }

  private apiURL = environment.apiURL + 'estudiantes';

  public obtenerTodos(): Observable<estudiantesDTO[]>{
    return this.http.get<estudiantesDTO[]>(this.apiURL);
  }

  public obtenerPorId(id:number): Observable<estudiantesDTO>{
    return this.http.get<estudiantesDTO>(`${this.apiURL}/${id}`);
  }

  public crear(estudiante: estudiantesCreacionDTO){
    return this.http.post(this.apiURL, estudiante);
  }

  public editar(id: number, estudiante:estudiantesCreacionDTO){
    return this.http.put(`${this.apiURL}/${id}`, estudiante);
  }

  public borrar(id: number){
    return this.http.delete(`${this.apiURL}/${id}`);
  }
}
```

7. En el archivo .ts del grupo que contiene la pagina html añadimos el servicio de estudiantes e implementamos los métodos CRUD usando las funciones creadas en estudiantes.service.ts

```ts
import { Component , OnInit} from '@angular/core';
import { EstudianteService } from '../estudiante.service';
import { FormBuilder, FormGroup } from '@angular/forms';
import { estudiantesCreacionDTO, estudiantesDTO } from '../estudiante';

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

  constructor(private estudianteService: EstudianteService, private formBuilder: FormBuilder){}
  form : FormGroup;

  ngOnInit(): void {
    this.form = this.formBuilder.group({
      nombre:'',
      carrera:'',
      cedula:'',
      fechaNacimiento:'',
      sexo:''
    })
  }

  submit()
  {
    this.guardarCambios(this.form.value);
  }

  guardarCambios(estudiante: estudiantesCreacionDTO){
    this.estudianteService.crear(estudiante).subscribe(()=>{

    })
  }

}
```

8. Ejemplo de código hmtl para que función Guardar Nuevo estudiante.

```
<h2>CREAR ESTUDIANTE</h2>
<form [formGroup]="form">

    <mat-form-field appearance="outline">
      <mat-label>Nombre</mat-label>
      <input formControlName="nombre" matInput />
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>Fecha Nacimiento</mat-label>
      <input formControlName="fechaNacimiento" matInput [matDatepicker]="picker" />
      <mat-datepicker-toggle matSuffix [for]="picker"></mat-datepicker-toggle>
      <mat-datepicker #picker></mat-datepicker>
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>Sexo</mat-label>
      <input formControlName="sexo" matInput />
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>Carrera</mat-label>
      <input formControlName="carrera" matInput />
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>Cedula</mat-label>
      <input formControlName="cedula" matInput />
    </mat-form-field>
    <div>
      <button mat-flat-button color="primary" (click)="submit()">
        Guardar Cambios
      </button>
      <a mat-stroked-button routerLink="/estudiantes">
        Cancelar
      </a>
    </div>

</form>
```

PARTE BACKEND

1. Vamos al archivo startup.cs o programs.cs y agregamos el servicio CORS que es una validación para que se conecten app backend y frontend con diferente url.

```
var frontEndURL = builder.Configuration.GetValue<string>("frontEnd_url");

builder.Services.AddCors(options =>
{
    options.AddDefaultPolicy(builder =>
    {
        builder.WithOrigins(frontEndURL).AllowAnyMethod().AllowAnyHeader();
    });
});
```

```
app.UseCors();
```
IMPORTANTE!!

2. En el archivo appsettings.json agregamos la cadena de conexión.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "frontEnd_url" : "http://localhost:4200"
}
```

JWT (JSON Web Token) compuesto de tres partes: Cabezera(Algoritmo a utilizar), datos(información del usuario) y la firma(Llave secreta).