1. Introducción

Nombre del proyecto: Proyecto SGE

Objetivo: Conectar FastAPI al proyecto de SGE

Tecnologías usadas:

- Frontend: Angular
- Backend: FastAPI
- Base de datos: MySQL
- Autenticación: JWT (JSON Web Token)

2. Arquitectura del Proyecto

Backend (FastAPI)

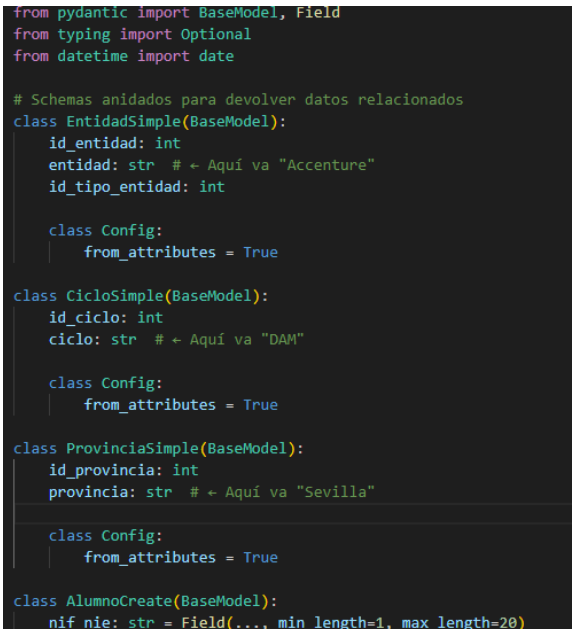
Uso de APIRouter y separación por routers:

1. Básicamente esto sirve para modularizar y organizar aplicaciones web, permitiendo agrupar múltiples rutas (endpoints) en archivos separados

```
router = APIRouter(
    prefix="/alumnos",
    tags=["alumnos"],
    dependencies=[Depends(auth.get_current_user)]
)
```

Validación con Pydantic (schemas):

Antes de pasar por la base de datos tienes que pasar por aquí y validan los datos



```python
from pydantic import BaseModel, Field
from typing import Optional
from datetime import date

# Schemas anidados para devolver datos relacionados
class EntidadSimple(BaseModel):
    id_entidad: int
    entidad: str  # ← Aquí va "Accenture"
    id_tipo_entidad: int

    class Config:
        from_attributes = True

class CicloSimple(BaseModel):
    id_ciclo: int
    ciclo: str  # ← Aquí va "DAM"

    class Config:
        from_attributes = True

class ProvinciaSimple(BaseModel):
    id_provincia: int
    provincia: str  # ← Aquí va "Sevilla"

    class Config:
        from_attributes = True

class AlumnoCreate(BaseModel):
    nif_nie: str = Field(..., min_length=1, max_length=20)
```

El class Config que ves ahí permite que Pydantic convierta directamente objetos SQLAlchemy en JSON.

ORM con SQLAlchemy
Aquí creo modelos para evitar escribir SQL:

```python
class Usuario(Base):
    __tablename__: str = "sgi_usuarios"
    id_usuario: Column[int] = Column(Integer, primary_key=True)
    usuario: Column[str] = Column(String(20))
    token_sesion: Column[str] = Column(String(255))

class SgiAlumno(Base):
    # nombre tabla
    __tablename__: str = "sgi_alumnos"

    # columnas
    id_alumno: Column[int] = Column(Integer, primary_key=True)
    nif_nie: Column[str] = Column(String(20))
    nombre: Column[str] = Column(String(100))
    apellidos: Column[str] = Column(String(150))
    fecha_nacimiento: Column[date] = Column(Date)

    # claves foraneas
    id_entidad: Column[int] = Column(Integer, ForeignKey("sgi_entidades.id_entidad"))
    id_ciclo: Column[int] = Column(Integer, ForeignKey("sgi_ciclos.id_ciclo"))

    curso: Column[int] = Column(Integer)
    telefono: Column[str] = Column(String(20))
    direccion: Column[str] = Column(String(255))
    localidad: Column[str] = Column(String(100))
    id_provincia: Column[int] = Column(Integer, ForeignKey("sgi_provincias.id_provincia"))
    observaciones: Column[str] = Column(Text)

    # relaciones
    entidad: RelationshipDeclared[Any] = relationship("SgiEntidades", back_populates="alumnos")
```

Validación JWT con dependencia get_current_user
Aquí controlo que el token se esté vigilando en todo momento:

```python
# api_key_header sirve para sacar el token de la cabecera Authorization. auto_error=False hace que no se lance ur
api_key_header = APIKeyHeader(name="Authorization", auto_error=False, scheme_name="Token PHP")

def get_current_user(token_header: str = Depends(api_key_header), db: Session = Depends(get_db)) -> Usuario:

    print(f"DEBUG AUTH: Header received: '{token_header}'") # DEBUG PRINT

    # si no existe el token pues un error 401
    if not token_header:
        print("DEBUG AUTH: No token header provided")
        raise HTTPException(status_code=401, detail="Token no proporcionado")

    # angular de por si manda el token con el prefijo Bearer asi que lo borro
    token: str = token_header.replace("Bearer ", "")
    print(f"DEBUG AUTH: Token parsed: '{token}'") # DEBUG PRINT

    # select para saber si el token existe ne la base d datos
    user: Usuario | None = db.query(Usuario).filter(Usuario.token_sesion == token).first()

    # si no existe out
    if not user:
        print(f"DEBUG AUTH: User not found for token: '{token}'") # DEBUG PRINT
        raise HTTPException(status_code=401, detail="Token inválido")

    return user # devolvermos el usuario q paso los filtros
```

Validaciones de negocio:

Esto es muy buen ejemplo de las validaciones de negocio que implemente en el proyecto:

```python
# 1) Validar DNI/NIE único
existing_alumno: SgiAlumno | None = db.query(SgiAlumno).filter(SgiAlumno.nif_nie == alumno.nif_nie).first()
if existing_alumno:
    raise HTTPException(status_code=400, detail="El DNI/NIE ya existe")
```

DNI único:
Arriba se puede ver cómo controlo que el DNI sea único

Fecha nacimiento válida:

Aquí hay varios factores bien con Pydantic poniendo el valor Date o bien como validaciones de negocio que la fecha sea anterior a hoy, entre otras

```python
class AlumnoCreate(BaseModel):
    nif_nie: str = Field(..., min_length=1, max_length=20)
    nombre: str = Field(..., min_length=1, max_length=100)
    apellidos: str = Field(..., min_length=1, max_length=150)
    fecha_nacimiento: date
    id_entidad: int
    id_ciclo: int
    curso: int
```

```python
if alumno.fecha_nacimiento >= date.today():
    raise HTTPException(status_code=400, detail="La fecha de nacimiento debe ser anterior a hoy")
# 3) Validar que la entidad sea de tipo 'centro educativo'
```

Entidad tipo centro educativo:

Aquí es donde saco el dato de centro educativo que necesito de la base de datos para más adelante darle el uso que corresponde

```python
# Schemas anidados para devolver datos relacionados
class EntidadSimple(BaseModel):
    id_entidad: int
    entidad: str  # ← Aquí va "Accenture"
    id_tipo_entidad: int

    class Config:
        from_attributes = True
```

UNIQUE compuesto en vacantes:

Aqui es donde implemento una constraint única para que entidad ciclo y curso no tengan duplicados

```python
class SgiVacantes(Base):

    __tablename__: str = "sgi_vacantes"

    id_vacante: Column[int] = Column(Integer, primary_key=True)
    id_entidad: Column[int] = Column(Integer, ForeignKey("sgi_entidades.id_entidad"))
    id_ciclo: Column[int] = Column(Integer, ForeignKey("sgi_ciclos.id_ciclo"))
    curso: Column[int] = Column(Integer)
    num_vacantes: Column[int] = Column(Integer)
    observaciones: Column[str] = Column(Text)

    entidad: _RelationshipDeclared[Any] = relationship("SgiEntidades")
    ciclo: _RelationshipDeclared[Any] = relationship("SgiCiclos")
    alumnos: _RelationshipDeclared[Any] = relationship("SgiVacanteAlumno", back_populates="vacante")

    __table_args__: tuple[UniqueConstraint] = (
        UniqueConstraint('id_entidad', 'id_ciclo', 'curso', name='uq_vacante_entidad_ciclo_curso'),
    )
```

Many to many:

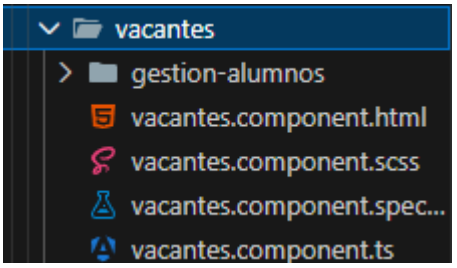
Este requisito lo gestiono con una tabla intermedia llamada vacante_X_alumno:

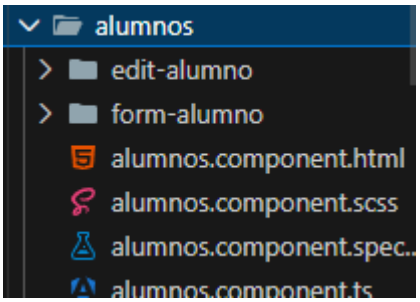
```python
class SgiVacanteAlumno(Base):

    __tablename__: str = "sgi_vacantes_X_alumnos"

    id_vacante_x_alumno: Column[int] = Column(Integer, primary_ke
    id_vacante: Column[int] = Column(Integer, ForeignKey("sgi_vac
    id_alumno: Column[int] = Column(Integer, ForeignKey("sgi_alum

    vacante: _RelationshipDeclared[Any] = relationship("SgiVacant
    alumno: _RelationshipDeclared[Any] = relationship("SgiAlumno"

    __table_args__: tuple[UniqueConstraint] = (
        UniqueConstraint('id_alumno', name='uq_vacante_alumno'),
    )
```

Frontend (Angular)

Así un poco en resumen dentro del front lo que he modificado/añadido ha sido lo siguiente:

- Creé dos componentes alumnos y vacantes aqui las puedes ver:





CRUD de alumnos:
C:

```
guardar() {
  if (this.alumnoForm.valid) {
    const nuevoAlumno = this.alumnoForm.value;
    this.alumnosService.crearAlumno(nuevoAlumno).subscribe(
      res => {
        alert('Alumno creado con éxito');
        this.alumnoForm.reset(); // Limpiar el formulario
        this.dialogRef.close(true); // Cerrar el diálogo y pasar true para indicar que se editó
      },
      err => {
        console.error(err);
        this.dialogRef.close(false); // Cerrar el diálogo y pasar false para indicar que hubo un error
        alert('Error al crear el alumno: ' + err.message);
      }
    );
  }
}
```

R:

```
cargarAlumnos() {
  this.alumnosService.getAlumnos().subscribe(
    res => {
      this.dataSource = new MatTableDataSource(res);
      this.dataSource.paginator = this.paginator; // Activar paginación
      this.dataSource.sort = this.sort;       // Activar ordenación
      this.alumnos = res;
      this.alumnos.forEach(alumno => {
        console.log(`ID: ${alumno.id_alumno} - Nombre: ${alumno.nombre}`); // Muestra id y nombre juntos
      });
    },
    err => console.error(err)
  );
}
```

U:

```
guardar() {
  if (this.alumnoForm.valid) {
    const nuevoAlumno = this.alumnoForm.value;
    this.alumnosService.actualizarAlumno(this.id, nuevoAlumno).subscribe(
      res => {
        alert('Alumno editado con éxito');
        this.alumnoForm.reset(); // Limpiar el formulario
        this.dialogRef.close(true); // Cerrar el diálogo y pasar true para indicar que se editó
      },
      err => {
        console.error(err);
        alert('Error al editar el alumno: ' + err.message);
        this.dialogRef.close(false); // Cerrar el diálogo y pasar false para indicar que no se editó
      }
    );
  }
}
```

D:

```
eliminarAlumno(id: number) {
  if (confirm('¿Estás seguro de que deseas eliminar este alumno?')) {
    this.alumnosService.eliminarAlumno(id).subscribe(
      res => {
        alert('Alumno eliminado correctamente');
        this.cargarAlumnos(); // Recargar la lista después de eliminar
      },
      err => console.error(err)
    );
  }
}
```

CRUD de vacantes:

C:

```
save() {
  if (this.form.invalid) return;

  const vacante = this.form.value;

  if (this.isEditMode) {
    // Update
    const id = this.data.id_vacante;
    this.vacantesService.actualizarVacante(id, vacante).subscribe(
      () => this.dialogRef.close(true),
      err => this.manejarError(err)
    );
  } else {
    // Create
    this.vacantesService.crearVacante(vacante).subscribe(
      () => this.dialogRef.close(true),
      err => this.manejarError(err)
    );
  }
}
```

R:

```
cargarVacantes() {
  this.vacantesService.getVacantes().subscribe(
    res => {
      this.dataSource.data = res;
      this.dataSource.paginator = this.paginator;
      this.dataSource.sort = this.sort;
    },
    err => console.error('Error cargando vacantes:', err)
  );
}
```

U:

```
save() {
  if (this.form.invalid) return;

  const vacante = this.form.value;

  if (this.isEditMode) {
    // Update
    const id = this.data.id_vacante;
    this.vacantesService.actualizarVacante(id, vacante).subscribe(
      () => this.dialogRef.close(true),
      err => this.manejarError(err)
    );
  } else {
    // Create
    this.vacantesService.crearVacante(vacante).subscribe(
      () => this.dialogRef.close(true),
      err => this.manejarError(err)
    );
  }
}
```

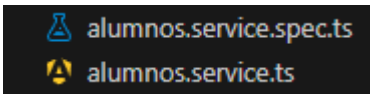
D:

```
eliminarVacante(id: number) {
  if (confirm('¿Seguro que quieres borrar esta vacante?')) {
    this.vacantesService.eliminarVacante(id).subscribe(
      () => {
        this.cargarVacantes();
      },
      err => console.error(err)
    );
  }
}
```

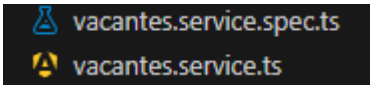
Bueno y también el CRUD de alumnos_X_vacantes:

```
getAlumnosDisponibles(idVacante: number): Observable<any[]> {
  return this.http.get<any[]>(`${this.apiUrl}/${idVacante}/alumnos-disponibles`);
}

getAlumnosAsignados(idVacante: number): Observable<any[]> {
  return this.http.get<any[]>(`${this.apiUrl}/${idVacante}/alumnos-asignados`);
}

asignarAlumno(idVacante: number, idAlumno: number): Observable<any> {
  return this.http.post<any>(`${this.apiUrl}/${idVacante}/asignar-alumno/${idAlumno}`, {});
}

quitarAlumno(idVacante: number, idAlumno: number): Observable<any> {
  return this.http.delete<any>(`${this.apiUrl}/${idVacante}/alumnos/${idAlumno}`);
}
```

Las conexiones al backend mediante services:




```
export class AlumnosService {

  private apiUrl = `${URL_FASTAPI}/alumnos`;

  constructor(private http: HttpClient) { }

  getAlumnos(): Observable<any[]> {
    return this.http.get<any[]>(this.apiUrl);
  }

  getAlumno(id: number): Observable<any> {
    return this.http.get<any>(`${this.apiUrl}/${id}`);
  }

  crearAlumno(alumno: any): Observable<any> {
    return this.http.post<any>(this.apiUrl, alumno);
  }

  actualizarAlumno(id: number, alumno: any): Observable<any> {
    return this.http.put<any>(`${this.apiUrl}/${id}`, alumno);
  }

  eliminarAlumno(id: number): Observable<any> {
    return this.http.delete<any>(`${this.apiUrl}/${id}`);
  }
```



```typescript
export class VacantesService {

  private apiUrl = `${URL_FASTAPI}/vacantes`;

  constructor(private http: HttpClient) { }

  getVacantes(): Observable<any[]> {
    return this.http.get<any[]>(this.apiUrl);
  }

  getVacante(id: number): Observable<any> {
    return this.http.get<any>(`${this.apiUrl}/${id}`);
  }

  crearVacante(vacante: any): Observable<any> {
    return this.http.post<any>(this.apiUrl, vacante);
  }

  actualizarVacante(id: number, vacante: any): Observable<any> {
    return this.http.put<any>(`${this.apiUrl}/${id}`, vacante);
  }

  eliminarVacante(id: number): Observable<any> {
    return this.http.delete<any>(`${this.apiUrl}/${id}`);
  }

  getAlumnosDisponibles(idVacante: number): Observable<any[]> {
    return this.http.get<any[]>(`${this.apiUrl}/${idVacante}/alumnos-disponibles`);
  }

  getAlumnosAsignados(idVacante: number): Observable<any[]> {
    return this.http.get<any[]>(`${this.apiUrl}/${idVacante}/alumnos-asignados`);
  }
```

Personalmente yo añadi maestros que son getters de entidades, ciclos y provincias:

```typescript
export class MaestrosService {

  private apiUrl = `${URL_FASTAPI}/maestros`;

  constructor(private http: HttpClient) { }

  // Obtener lista completa de entidades (para desplegables)
  getEntidades(): Observable<any[]> {
    return this.http.get<any[]>(`${this.apiUrl}/entidades`);
  }

  // Obtener lista completa de ciclos
  getCiclos(): Observable<any[]> {
    return this.http.get<any[]>(`${this.apiUrl}/ciclos`);
  }

  // Obtener lista completa de provincias
  getProvincias(): Observable<any[]> {
    return this.http.get<any[]>(`${this.apiUrl}/provincias`);
  }
}
```

Modelado de datos:

En el proyecto tienes adjuntado el script que use para la creación de la tabla aquí te voy a poner las tablas en si según pedía el enunciado:

```sql
-- Script base de datos

-- 1. Tabla sgi_alumnos
CREATE TABLE sgi_alumnos (
    id_alumno INT AUTO_INCREMENT PRIMARY KEY,
    nif_nie VARCHAR(20) UNIQUE NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    apellidos VARCHAR(150) NOT NULL,
    fecha_nacimiento DATE NOT NULL,
    id_entidad INT NOT NULL,
    id_ciclo INT NOT NULL,
    curso INT NOT NULL,
    telefono VARCHAR(20) NOT NULL,
    direccion VARCHAR(255),
    cp VARCHAR(10),
    localidad VARCHAR(100),
    id_provincia INT,
    observaciones TEXT,

    -- Claves foráneas apuntando a las tablas sgi_* existentes
    CONSTRAINT fk_alumnos_entidad FOREIGN KEY (id_entidad) REFERENCES sgi_entidades(id_entidad),
    CONSTRAINT fk_alumnos_ciclo FOREIGN KEY (id_ciclo) REFERENCES sgi_ciclos(id_ciclo),
    CONSTRAINT fk_alumnos_provincia FOREIGN KEY (id_provincia) REFERENCES sgi_provincias(id_provincia)
);

-- 2. Tabla sgi_vacantes
CREATE TABLE sgi_vacantes (
    id_vacante INT AUTO_INCREMENT PRIMARY KEY,
    id_entidad INT NOT NULL,
    id_ciclo INT NOT NULL,
    curso INT NOT NULL,
    num_vacantes INT NOT NULL,
    observaciones TEXT,

    UNIQUE(id_entidad, id_ciclo, curso),

    CONSTRAINT fk_vacantes_entidad FOREIGN KEY (id_entidad) REFERENCES sgi_entidades(id_entidad),
    CONSTRAINT fk_vacantes_ciclo FOREIGN KEY (id_ciclo) REFERENCES sgi_ciclos(id_ciclo)
);

-- 3. Tabla auxiliar
CREATE TABLE sgi_vacantes_X_alumnos (
    id_vacante_x_alumno INT AUTO_INCREMENT PRIMARY KEY,
    id_vacante INT NOT NULL,
    id_alumno INT NOT NULL UNIQUE,

    CONSTRAINT fk_va_vacante FOREIGN KEY (id_vacante) REFERENCES sgi_vacantes(id_vacante) ON DELETE CASCADE,
    CONSTRAINT fk_va_alumno FOREIGN KEY (id_alumno) REFERENCES sgi_alumnos(id_alumno) ON DELETE CASCADE
);
```

Y yo creo que eso es todo o por lo menos lo más remarcable del proyecto.

Además de todo esto tienes acceso a la documentación del FastAPI aqui