Crear dos componentes en el proyecto suministrado en **Angular**, utilizando un **Backend** con **FastAPI** para la gestión de ambos **CRUD**.
Al utilizar **FASTAPI** se debe tener en cuenta la **validación del token** enviado.

Componente 1: CRUD Alumnos
Tabla con alumnos, los alumnos pertenecen a una entidad de tipo CENTRO EDUCATIVO. Un alumno **sólo** puede pertenecer a un centro, en cada fila un alumno tiene:

TABLA: sgi_alumnos (ejemplo)

- **id_alumno: 1 (PK) ***
- **nif_nie: 12345678X (UNIQUE) ***
- **Nombre: José ***
- **Apellidos: Fernández López ***
- **Fecha_nacimiento: 01/02/2003 (DATE) ***
- **Entidad: IES Playamar (FK) (Debe ser de tipo centro educativo) ***
- **Ciclo: DAM (FK) ***
- **Curso: 2 (Numero) ***
- **Vacante asignada: Accenture (Este valor se gestiona en la componente siguiente, sólo aparece en el grid)**
- Teléfono: 666 666 666 (Texto) *
- Dirección: Calle Goya 5 (Texto)
- CP: 2900 0(Texto)
- Localidad: Málaga (Texto)
- Provincia: Málaga (FK)
- Observaciones (Texto)

Componente 2: CRUD Vacantes Tabla con vacantes de las entidades para realizar las prácticas de los alumnos.
En cada fila de la tabla, una vacante tiene:

TABLA: sgi_vacantes (ejemplo)

- **id_vacante: 1 (PK) ***
- **Entidad: Accenture (FK) ***
- **Ciclo: DAM (FK) ***
- **Curso: 1 (Número) ***
- **Num_vacantes: 4 (Número) * (Indica la cantidad de vacantes disponibles)**
- **Num_alumnos: 3 (Número) * (campo calculado según el número de alumnos del listado)**
- Listado_alumnos: Luigi, Mario, Peach (Tabla auxiliar)
- Observaciones

UNIQUE(id_entidad, id_ciclo, curso), para no repetir vacantes.

Durante la edición tendremos que poder añadir o eliminar alumnos.
Añadir un alumno de entre los disponibles (no seleccionado aún en una vacante, con el mismo ciclo y curso que la vacante). Listado_alumnos en realidad no existe en la tabla. Se trata de una referencia de muchos a muchos con una tabla auxiliar, en la que, para el ejemplo tendríamos:

TABLA: sgi_vacantes_X_alumnos

- id_vacante_x_alumno
- id_vacante NOT NULL
- id_alumno NOT NULL
- UNIQUE(id_alumno) (no se puede repetir un alumno en ninguna vacante)

Campo con *: Obligatorio
Campo en **negrita**: aparece en el grid

Documentar y subir los ficheros implicados.
Exponer el proyecto: Se tendrá en cuenta durante la exposición el dominio del proyecto. El no control del código, no recordar, no saber se dará por sentado que el alumno no domina el proyecto y se dará por suspensa la materia.

NOTA: No mostrar id de las FK, sino el dato, ejemplo Accenture, no un 3

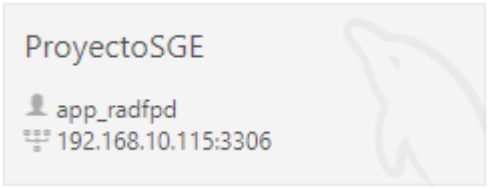
DOCUMENTACION PROYECTO SGE:

Día 1: Configuración, Base de Datos y Autenticación Backend

Objetivo: Tener el entorno listo, las tablas creadas y FastAPI validando los tokens del sistema antiguo.

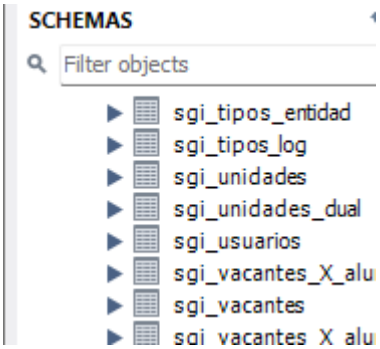
Base de Datos (SQL)

1. Conéctate a la Base de Datos con tu cliente SQL favorito (DBeaver, Workbench) usando las credenciales que vimos en conn.php.

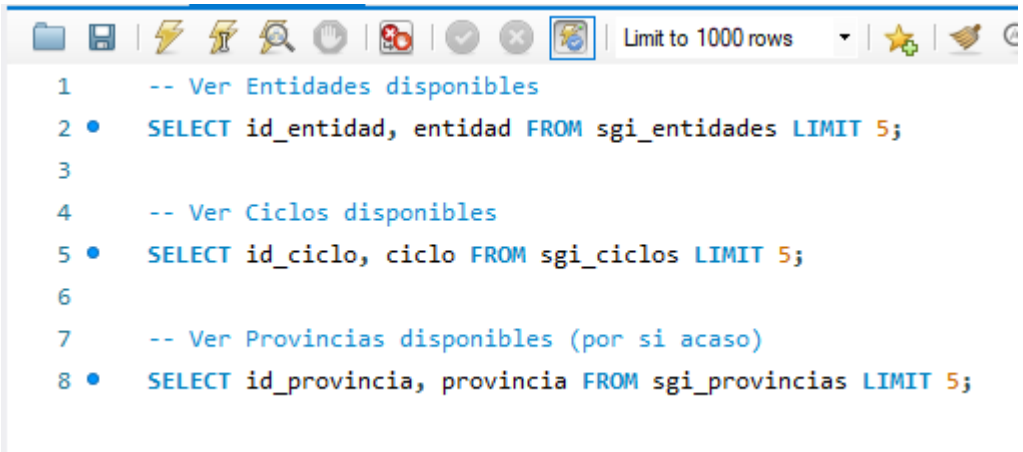


   Usuario: app_radfpd
   Contraseña: 0dTtnIG!-WDZRC8k

2. Ejecuta el script de creación de tablas (sgi_alumnos, sgi_vacantes, sgi_vacantes_X_alumnos).
   - El script esta adjuntado en el zip del proyecto.



3. Añade algunos datos de prueba en entidades y ciclos si no existen, ya que los necesitamos para las claves foráneas (FK).
   - Hago selects para comprobar datos de las bases de datos:



- Ahora inserto datos de prueba para comprobar que todo funciona correctamente:

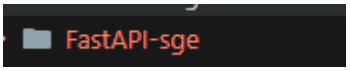
```sql
SELECT
    a.nombre,
    a.apellidos,
    v.num_vacantes,
    e.entidad as vacante_en_entidad
FROM sgi_alumnos a
JOIN sgi_vacantes_X_alumnos va ON a.id_alumno = va.id_alumno
JOIN sgi_vacantes v ON va.id_vacante = v.id_vacante
JOIN sgi_entidades e ON v.id_entidad = e.id_entidad
WHERE a.nif_nie = 'TEST0001X';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

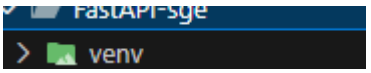
nombre	apellidos	num_vacantes	vacante_en_entidad
Alumno1	Prueba Asignado	5	IES Playamar

Inicializar Proyecto FastAPI

1. Crea la carpeta backend-sge.



2. Crea un entorno virtual: python -m venv venv y actívalo.

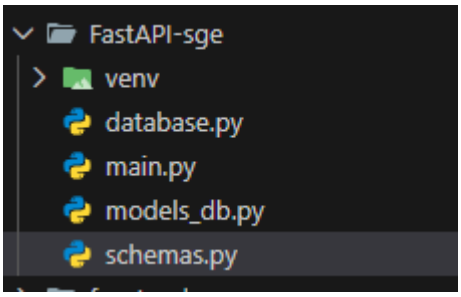


3. Instala librerías: pip install fastapi uvicorn sqlalchemy mysql-connector-python pydantic requests.

```
(venv) PS C:\Users\dam2\Desktop\DAM\SGE\T5.1\FastAPI-sge> pip install fastapi "uvicorn[sta
dard]" sqlalchemy mysql-connector-python pydantic python-multipart requests
Collecting fastapi
  Downloading fastapi-0.128.2-py3-none-any.whl.metadata (30 kB)
Collecting sqlalchemy
  Downloading sqlalchemy-2.0.46-cp313-cp313-win_amd64.whl.metadata (9.8 kB)
```

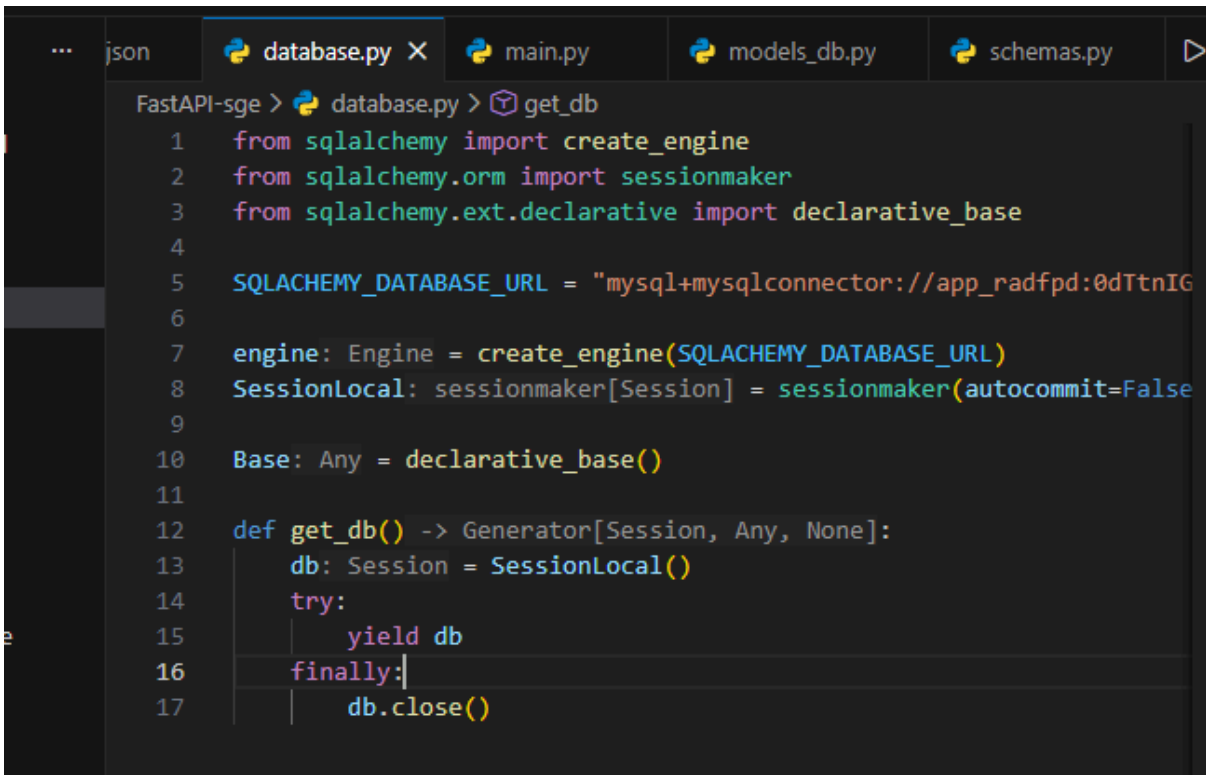
   - fastapi: El framework web.
   - uvicorn: El servidor para ejecutarlo.
   - sqlalchemy: Para manejar la Base de Datos (ORM).
   - mysql-connector-python: El driver para conectar a MySQL.
   - pydantic: Para validar datos (schemas).
   - python-multipart: Necesario a veces para forms (por si acaso).

4. Crea la estructura básica: main.py, database.py, models_db.py, schemas.py.



Conexión y Autenticación (Crucial)

1. En database.py: Configura la conexión a los mismos datos que conn.php (DB app_radfpd).
    - Aqui hago la conexion a la base de datos con usando mysqlconnector y sqlalchemy:



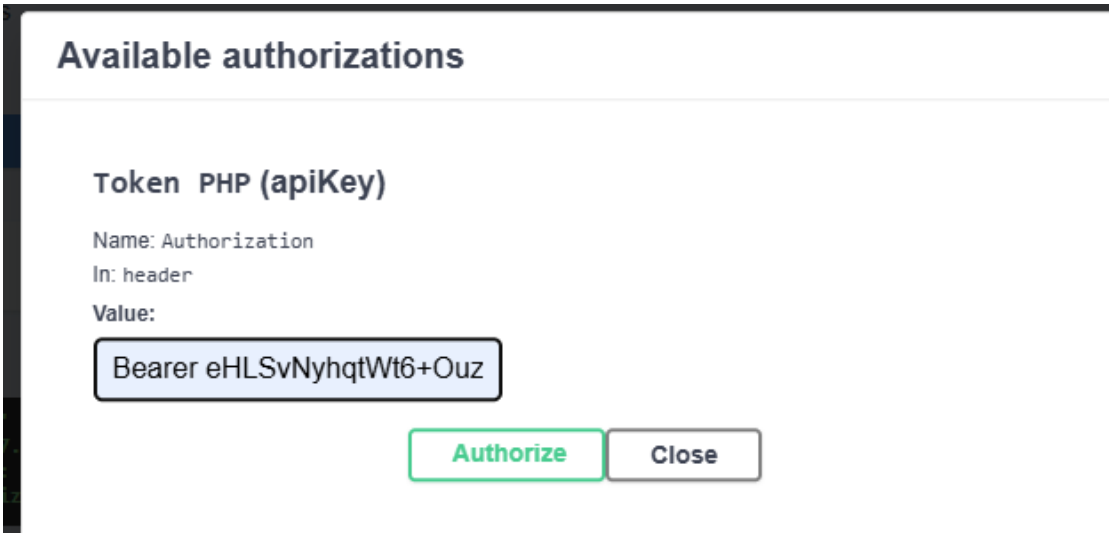
2. En auth.py: Crea una función get_current_user(token: str).
    - Lógica: Esta función debe hacer una SELECT id_usuario FROM sgi_usuarios WHERE token_sesion = :token.

```python
FastAPI-sge > auth.py > ...
from fastapi import Depends, HTTPException
from fastapi.security import APIKeyHeader
from sqlalchemy.orm import Session
from database import get_db
from models import Usuario

# api_key_header sirve para sacar el token de la cabecera Authorization. auto_error=False hace que no se lance un error automáticamente si no se
api_key_header = APIKeyHeader(name="Authorization", auto_error=False)

def get_current_user(token_header: str = Depends(api_key_header), db: Session = Depends(get_db)) -> Usuario:
    # si no existe el token pues un error 401
    if not token_header:
        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 ", "")

    # 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:
        raise HTTPException(status_code=401, detail="Token inválido")

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

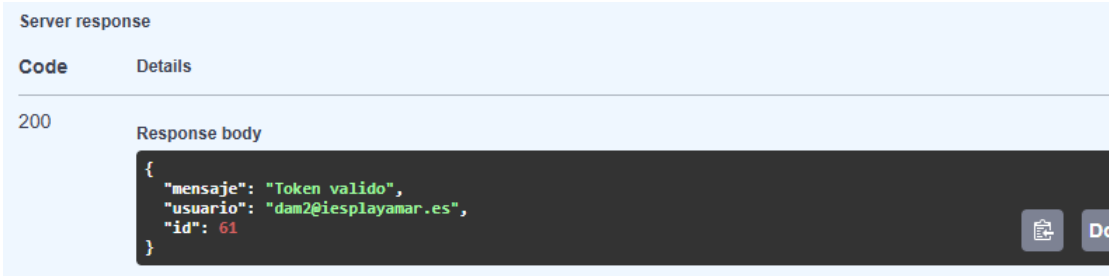
- Si devuelve un ID, el token es válido. Si no, lanza un error 401. Esto conecta la seguridad del PHP con tu Python.

Para poder comprobarlo hago esto:

1. Primero pongo Bearer eHLSvNyhqtWt6+Ouz+qlqqfpl7XOnJSDZY6Krw== (que es el token de inicio de sesion de pepi) aqui:



2. Luego de esto le doy a try it out y se ve como el token es valido y nos devuelve el nombre y la id:



En realidad da igual si usas lo de bearer o no de todas formas lo dejo por si acaso.

Día 2: Desarrollo Backend - Módulo Alumnos

Objetivo: Tener la API de Alumnos funcionando y probada.

Modelos y Schemas

1. Define el modelo SQLAlchemy para la tabla sgi_alumnos en models_db.py.

```python
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)
    id_ciclo: Column[int] = Column(Integer)
    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)
    observaciones: Column[str] = Column(Text)
```

2. Define los schemas Pydantic en schemas.py (ej. AlumnoCreate, AlumnoResponse) para validar las entradas.

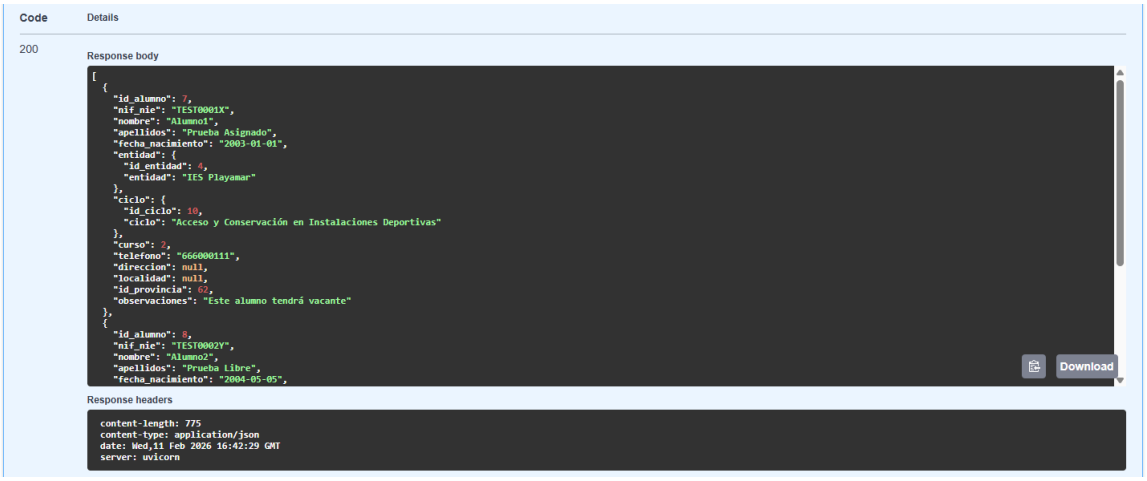
schemas.py > ...

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

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
    telefono: Optional[str] = Field(None, max_length=20)
    direccion: Optional[str] = Field(None, max_length=255)
    localidad: Optional[str] = Field(None, max_length=100)
    id_provincia: Optional[int] = None
    observaciones: Optional[str] = None

class AlumnoResponse(BaseModel):
    id_alumno: int
    nif_nie: str
    nombre: str
    apellidos: str
    fecha_nacimiento: date
    id_entidad: int
    id_ciclo: int
    curso: int
    telefono: Optional[str] = None
    direccion: Optional[str] = None
    localidad: Optional[str] = None
    id_provincia: Optional[int] = None
    observaciones: Optional[str] = None

    class Config:
        from_attributes = True
```

Endpoints CRUD Alumnos

1. Implementa en main.py (o en un router routers/alumnos.py):
    - GET /alumnos: Listado.

```python
@router.get("/", response_model=list[AlumnoResponse])
def listar_alumnos(db: Session = Depends(get_db)) -> List[SgiAlumno]:
    alumnos: List[SgiAlumno] = db.query(SgiAlumno).options(joinedload(SgiAlumno.entidad), joinedload(SgiAlumno.ciclo)).all()

    return alumnos
```



- POST /alumnos: Crear (Validar DNI/NIE único aquí).

```python
@router.post("/", response_model=AlumnoResponse, status_code=201)
def crear_alumno(alumno: AlumnoCreate, db: Session = Depends(get_db)) -> SgiAlumno:
    # 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")
    # 2) Validar fecha_nacimiento < hoy
    if alumno.fecha_nacimiento >= date.today():
        raise HTTPException(status_code=400, detail="La fecha de nacimiento debe ser anterior a hoy")
    # 3) Crear objeto SgiAlumno
    nuevo_alumno = SgiAlumno(
        nif_nie=alumno.nif_nie,
        nombre=alumno.nombre,
        apellidos=alumno.apellidos,
        fecha_nacimiento=alumno.fecha_nacimiento,
        id_entidad=alumno.id_entidad,
        id_ciclo=alumno.id_ciclo,
        curso=alumno.curso,
        telefono=alumno.telefono,
        direccion=alumno.direccion,
        localidad=alumno.localidad,
        id_provincia=alumno.id_provincia,
        observaciones=alumno.observaciones
    )
    # 4) db.add / db.commit / db.refresh
    db.add(nuevo_alumno)
    db.commit()
    db.refresh(nuevo_alumno)
    # 5) Devolver el alumno creado
    return nuevo_alumno
```

- PUT /alumnos/{id} y DELETE /alumnos/{id}.

```python
@router.put("/{alumno_id}", response_model=AlumnoResponse)
def actualizar_alumno(alumno_id: int, alumno: AlumnoCreate, db: Session = Depends(get_db)) -> SgiAlumno:
    alumno_db: SgiAlumno | None = db.query(SgiAlumno).filter(SgiAlumno.id_alumno == alumno_id).first()

    if not alumno_db:
        raise HTTPException(status_code=404, detail="Alumno no encontrado")

    # Validar DNI/NIE único (si se ha cambiado)
    if alumno.nif_nie != alumno_db.nif_nie:
        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")

    # Validar fecha_nacimiento < hoy
    if alumno.fecha_nacimiento >= date.today():
        raise HTTPException(status_code=400, detail="La fecha de nacimiento debe ser anterior a hoy")

    # Actualizar campos
    db.query(SgiAlumno).filter(SgiAlumno.id_alumno == alumno_id).update({
        SgiAlumno.nif_nie: alumno.nif_nie,
        SgiAlumno.nombre: alumno.nombre,
        SgiAlumno.apellidos: alumno.apellidos,
        SgiAlumno.fecha_nacimiento: alumno.fecha_nacimiento,
        SgiAlumno.id_entidad: alumno.id_entidad,
        SgiAlumno.id_ciclo: alumno.id_ciclo,
        SgiAlumno.curso: alumno.curso,
        SgiAlumno.telefono: alumno.telefono,
        SgiAlumno.direccion: alumno.direccion,
        SgiAlumno.localidad: alumno.localidad,
        SgiAlumno.id_provincia: alumno.id_provincia,
        SgiAlumno.observaciones: alumno.observaciones
    })

    db.commit()
    db.refresh(alumno_db)

    return alumno_db
```

2. Nota: Recuerda que en el GET debes devolver el nombre de la Entidad y Ciclo, no solo sus IDs (puedes usar .join en SQLAlchemy).
   Ya esta hecho en los Esquemas les creo dos simples para pasarles los nombres correspondientes

Pruebas con Swagger

1. Arranca el servidor: uvicorn main:app --reload.
2. Entra a http://localhost:8000/docs.
3. Prueba a insertar y listar alumnos usando un token real que cojas de tu navegador (haz login en la web actual y copia el token del LocalStorage para probar).

Día 3: Desarrollo Backend - Módulo Vacantes

Objetivo: Finalizar toda la lógica del servidor, incluyendo la asignación compleja de alumnos.

CRUD Vacantes Básico

1. Modelos y endpoints para creación y edición básica de la tabla sgi_vacantes.
2. Validar la restricción UNIQUE(id_entidad, id_ciclo, curso).

Lógica Compleja: Asignación de Alumnos

1. Crea el endpoint GET /alumnos/disponibles:

    - Debe recibir ciclo_id y curso.
    - Consulta: "Select alumnos WHERE ciclo=X AND curso=Y AND id NOT IN (SELECT id_alumno FROM sgi_vacantes_x_alumnos)".
2. Endpoint para asignar: POST /vacantes/{id}/alumnos. Recibe una lista de IDs de alumnos y los inserta en la tabla intermedia.
3. Endpoint para desasignar: DELETE /vacantes/{id}/alumnos/{id_alumno}.

Cálculos

1. Asegúrate de que al pedir una Vacante, calculas el campo num_alumnos contando cuántos registros tiene asociados en la tabla intermedia.

Día 4: Frontend Angular - Integros y Alumnos

Objetivo: Integrar Angular con Python y terminar la pantalla de Alumnos.

Configuración Servicio

1. Edita environment.ts: Añade URL_API_FASTAPI: 'http://localhost:8000'.
2. Genera AlumnosService: Métodos que hagan http.get, post, etc. atacando a esa nueva URL. Asegúrate de que el CommonService inyecte el token en las cabeceras también para estas peticiones (o configúralo manualmente en este servicio).

Componentes Alumnos

1. ng g m ciclos/alumnos --routing (Crea un módulo estanco para alumnos).
2. Listado: Usa MatTable. Columnas: Nombre, Apellidos, Centro (texto), Ciclo, etc.
3. Formulario (Add/Edit):
    - Usa MatSelect para elegir Entidad y Ciclo (deberás cargar estas listas desde los servicios existentes de PHP EntidadesService, CiclosService).
    - Validador de DNI.

Día 5: Frontend Angular - Vacantes y Remates

Objetivo: Pantalla de Vacantes, tests finales y documentación.

Componentes Vacantes

1. Listado: Muestra el campo calculado num_alumnos vs num_vacantes.
2. Edición (Parte difícil):

- El formulario de edición debe tener la info de la vacante arriba.
- Abajo: Dos paneles o una tabla con acciones.
- Panel 1 "Asignados": Lista los alumnos que ya tiene la vacante. Botón "Quitar".
- Panel 2 "Añadir": Un MatSelect que carga los resultados de GET /alumnos/disponibles (filtrados por el ciclo/curso de esta vacante). Botón "Asignar".

Revisión y Documentación

1. Verifica que no puedes asignar estudiantes si num_alumnos >= num_vacantes (opcional, pero recomendado).
2. Genera un PDF o README explicando:
    - Cómo levantar el FastAPI.
    - Estructura de la BBDD.
    - Capturas de pantalla del funcionamiento.

COSAS A PARTE:
Añado las opciones al menu:

```
async getMenu() {
  const RESPONSE = await this.menuService.getMenu().toPromise();
  this.menu = RESPONSE.data;
  const menuPracticas = {
    grupo: 'Prácticas',
    opciones: [
      {
        opcion: 'alumnos',
        accion: 'alumnos',
        texto_tooltop: 'Gestion de alumnos FCT'
      },
      {
        opcion: 'vacantes',
        accion: 'vacantes',
        texto_tooltop: 'Gestion de vacantes FCT'
      }
    ]
  };

  this.menu.push(menuPracticas);
}
```