

Estoy programando con Python.

- Quiero crear una API REST utilizando el framework FastAPI que cuente con los end points necesarios para las tareas CRUD contra una base de datos.

El contexto es la creación de una utilidad para la escuela de música 'Armonía'.

Esta escuela quiere almacenar toda su información de alumnos, disciplinas y niveles de progreso

de las mismas, profesores, tarifas, descuentos, matrículas, etc. en una base de datos SQL.

La API gestionará el acceso y manejo de esos datos.

He desarrollado la base de datos con el siguiente código SQL:

```
-- Crear la tabla de profesores CREATE TABLE IF NOT EXISTS profesores (  
id SERIAL PRIMARY KEY,  
nombre VARCHAR(50) NOT NULL  
);
```

```
CREATE TABLE IF NOT EXISTS alumnos (  
id SERIAL PRIMARY KEY,  
    nombre VARCHAR(50) NOT NULL,  
    apellidos VARCHAR(50) NOT NULL,  
    edad INT NOT NULL,  
    telefono VARCHAR(20),  
    correo VARCHAR(100),  
    familiar_id INT,  
    CONSTRAINT fk_familiar  
        FOREIGN KEY(familiar_id)  
            REFERENCES alumnos(id)  
);
```

```
ALTER TABLE alumnos  
DROP COLUMN IF EXISTS correo,  
DROP COLUMN IF EXISTS telefono;
```

```
CREATE TABLE IF NOT EXISTS datos_sensibles (  
    alumno_id INT PRIMARY KEY,  
    email VARCHAR(100),  
    telefono VARCHAR(20),  
    FOREIGN KEY (alumno_id) REFERENCES alumnos(id)  
);
```

```
CREATE OR REPLACE FUNCTION insert_alumno_with_sensitive_data(  
    p_nombre VARCHAR(50),  
    p_apellidos VARCHAR(50),  
    p_edad INT,  
    p_email VARCHAR(100),  
    p_telefono VARCHAR(20),  
    p_familiar_id INT DEFAULT NULL  
) RETURNS INT AS $$  
DECLARE  
    new_alumno_id INT;  
BEGIN
```

```

-- Insert into alumnos
INSERT INTO alumnos (nombre, apellidos, edad, familiar_id)
VALUES (p_nombre, p_apellidos, p_edad, p_familiar_id)
RETURNING id INTO new_alumno_id;

-- Insert into datos_sensibles
INSERT INTO datos_sensibles (alumno_id, email, telefono)
VALUES (new_alumno_id, p_email, p_telefono);

RETURN new_alumno_id;
END;
$$ LANGUAGE plpgsql;

-- Crear la tabla de clases
CREATE TABLE IF NOT EXISTS clases (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    profesor_id INT NOT NULL,
    precio_base DECIMAL(5,2) NOT NULL,
    tipo_pack VARCHAR(20) NOT NULL, -- pack1 para clases de 35€, pack2 para clases de
40€
    CONSTRAINT fk_profesor
        FOREIGN KEY(profesor_id)
            REFERENCES profesores(id)
);

-- Crear la tabla de niveles
CREATE TABLE IF NOT EXISTS niveles (
    id SERIAL PRIMARY KEY,
    clase_id INT NOT NULL,
    nivel VARCHAR(20) NOT NULL,
    CONSTRAINT fk_clase
        FOREIGN KEY(clase_id)
            REFERENCES clases(id)
);

CREATE TABLE IF NOT EXISTS clase_profesor (
    clase_id INT REFERENCES clases(id) ON DELETE CASCADE,
    profesor_id INT REFERENCES profesores(id) ON DELETE CASCADE,
    PRIMARY KEY (clase_id, profesor_id)
);

-- Crear la tabla de alumnos_clases
CREATE TABLE IF NOT EXISTS alumnos_clases (
    id SERIAL PRIMARY KEY,
    alumno_id INT NOT NULL,
    clase_id INT NOT NULL,
    nivel_id INT NOT NULL,
    CONSTRAINT fk_alumno

```

```

        FOREIGN KEY(alumno_id)
            REFERENCES alumnos(id),
    CONSTRAINT fk_clase
        FOREIGN KEY(clase_id)
            REFERENCES clases(id),
    CONSTRAINT fk_nivel
        FOREIGN KEY(nivel_id)
            REFERENCES niveles(id)
);

-- Crear la tabla de precios y descuentos
CREATE TABLE IF NOT EXISTS precios (
    id SERIAL PRIMARY KEY,
    tipo_pack VARCHAR(20) NOT NULL,
    precio_base DECIMAL(5,2) NOT NULL,
    descuento_segunda DECIMAL(5,2) NOT NULL,
    descuento_tercera DECIMAL(5,2) NOT NULL
);

```

ChatGPT:

Para desarrollar una API REST usando FastAPI que gestione las operaciones CRUD para la base

de datos de la escuela de música "Armonía", podemos seguir los siguientes pasos:

- 1 Instalación de FastAPI y SQLAlchemy:** Primero, asegúrate de tener instalado FastAPI,

SQLAlchemy, pydantic, y asyncpg para la conexión con PostgreSQL.

```

pip install fastapi sqlalchemy databases asyncpg
pydantic

```

- 2 Configuración de la Base de Datos:** Configuramos la conexión a la base de datos PostgreSQL

usando SQLAlchemy y databases.

- 3 Definición de los Modelos:** Definimos los modelos ORM para SQLAlchemy.
- 4 Definición de los Esquemas:** Creamos los esquemas con Pydantic para la validación de datos.
- 5 Creación de los Endpoints CRUD:** Definimos los endpoints en FastAPI.

Paso 1: Configuración de la Base de Datos

```
from sqlalchemy import create_engine, MetaData
```

```
DATABASE_URL = "postgresql+asyncpg://  
user:password@localhost/dbname"
```

```
engine = create_engine(DATABASE_URL)  
metadata = MetaData()
```

Paso 2: Definición de los Modelos

```
from sqlalchemy import Table, Column, Integer, String,  
ForeignKey, DECIMAL  
from sqlalchemy.orm import relationship
```

```
profesores = Table(  
    "profesores", metadata,  
    Column("id", Integer, primary_key=True, index=True),  
    Column("nombre", String(50), nullable=False)  
)
```

```
alumnos = Table(  
    "alumnos", metadata,  
    Column("id", Integer, primary_key=True, index=True),  
    Column("nombre", String(50), nullable=False),  
    Column("apellidos", String(50), nullable=False),  
    Column("edad", Integer, nullable=False),  
    Column("familiar_id", Integer,  
ForeignKey("alumnos.id"))  
)
```

```
datos_sensibles = Table(  
    "datos_sensibles", metadata,  
    Column("alumno_id", Integer, primary_key=True),  
    Column("email", String(100)),  
    Column("telefono", String(20)),  
    ForeignKeyConstraint(["alumno_id"], ["alumnos.id"])  
)
```

```
clases = Table(  
    "clases", metadata,  
    Column("id", Integer, primary_key=True, index=True),
```

```

        Column("nombre", String(50), nullable=False),
        Column("profesor_id", Integer,
ForeignKey("profesores.id"), nullable=False),
        Column("precio_base", DECIMAL(5, 2), nullable=False),
        Column("tipo_pack", String(20), nullable=False)
    )

niveles = Table(
    "niveles", metadata,
    Column("id", Integer, primary_key=True, index=True),
    Column("clase_id", Integer, ForeignKey("clases.id"),
nullable=False),
    Column("nivel", String(20), nullable=False)
)

clase_profesor = Table(
    "clase_profesor", metadata,
    Column("clase_id", Integer, ForeignKey("clases.id",
ondelete="CASCADE")),
    Column("profesor_id", Integer,
ForeignKey("profesores.id", ondelete="CASCADE")),
    PrimaryKeyConstraint("clase_id", "profesor_id")
)

alumnos_clases = Table(
    "alumnos_clases", metadata,
    Column("id", Integer, primary_key=True, index=True),
    Column("alumno_id", Integer,
ForeignKey("alumnos.id"), nullable=False),
    Column("clase_id", Integer, ForeignKey("clases.id"),
nullable=False),
    Column("nivel_id", Integer, ForeignKey("niveles.id"),
nullable=False)
)

precios = Table(
    "precios", metadata,
    Column("id", Integer, primary_key=True, index=True),
    Column("tipo_pack", String(20), nullable=False),
    Column("precio_base", DECIMAL(5, 2), nullable=False),
    Column("descuento_segunda", DECIMAL(5, 2),
nullable=False),
    Column("descuento_tercera", DECIMAL(5, 2),
nullable=False)

```

)

Paso 3: Definición de los Esquemas

```
from pydantic import BaseModel
from typing import Optional, List

class ProfesorBase(BaseModel):
    nombre: str

class ProfesorCreate(ProfesorBase):
    pass

class Profesor(ProfesorBase):
    id: int

    class Config:
        orm_mode: True

class AlumnoBase(BaseModel):
    nombre: str
    apellidos: str
    edad: int
    familiar_id: Optional[int] = None

class AlumnoCreate(AlumnoBase):
    email: str
    telefono: str

class Alumno(AlumnoBase):
    id: int

    class Config:
        orm_mode: True

class DatosSensiblesBase(BaseModel):
    email: str
    telefono: str

class DatosSensiblesCreate(DatosSensiblesBase):
    pass

class DatosSensibles(DatosSensiblesBase):
    alumno_id: int
```

```

class Config:
    orm_mode: True

# Definimos esquemas adicionales para clases, niveles,
etc.

```

Paso 4: Creación de los Endpoints CRUD

```

from fastapi import FastAPI, HTTPException, Depends
from sqlalchemy.orm import Session
from databases import Database

app = FastAPI()

database = Database(DATABASE_URL)

@app.on_event("startup")
async def startup():
    await database.connect()

@app.on_event("shutdown")
async def shutdown():
    await database.disconnect()

# Dependencia para obtener la sesión de la base de datos
def get_db():
    db = database
    try:
        yield db
    finally:
        db.close()

@app.post("/profesores/", response_model=Profesor)
async def create_profesor(profesor: ProfesorCreate, db: Session = Depends(get_db)):
    query =
profesores.insert().values(nombre=profesor.nombre)
    last_record_id = await db.execute(query)
    return {**profesor.dict(), "id": last_record_id}

@app.get("/profesores/{profesor_id}",
response_model=Profesor)
async def read_profesor(profesor_id: int, db: Session =

```

```

Depends(get_db)):
    query = profesores.select().where(profesores.c.id ==
profesor_id)
    profesor = await db.fetch_one(query)
    if profesor is None:
        raise HTTPException(status_code=404,
detail="Profesor not found")
    return profesor

@app.put("/profesores/{profesor_id}",
response_model=Profesor)
async def update_profesor(profesor_id: int, profesor:
ProfesorCreate, db: Session = Depends(get_db)):
    query = profesores.update().where(profesores.c.id ==
profesor_id).values(nombre=profesor.nombre)
    await db.execute(query)
    return {**profesor.dict(), "id": profesor_id}

@app.delete("/profesores/{profesor_id}")
async def delete_profesor(profesor_id: int, db: Session =
Depends(get_db)):
    query = profesores.delete().where(profesores.c.id ==
profesor_id)
    await db.execute(query)
    return {"message": "Profesor deleted successfully"}

```

CRUD para alumnos, clases, niveles, etc.

En este ejemplo, hemos configurado la base de datos, definido modelos SQLAlchemy y esquemas Pydantic, y creado endpoints CRUD para la entidad Profesor. De manera similar, puedes definir los endpoints CRUD para las demás entidades como Alumno, Clase, Nivel, etc.