Prueba

SOBRE MATTILDA

En Mattilda trabajamos con colegios, estudiantes y facturación. Queremos ver cómo diseñas modelos de datos, APIs y lógica de negocio en un dominio similar al nuestro.

El objetivo de este reto no es que entregues algo “perfecto”, sino entender cómo piensas, cómo estructuras tu código y qué decisiones tomas bajo tiempo limitado.

Idealmente compartes tu avance en máximo 48 horas, aunque no esté totalmente terminado.

OBJETIVO DEL PROYECTO

Diseñar y desarrollar un pequeño sistema que modele:

- **Colegios (Schools)**
- **Estudiantes (Students)**
- **Facturas (Invoices)**

Estos modelos deben estar relacionados entre sí y permitir responder preguntas como:

- ¿Cuánto le debe un estudiante a un colegio?
- ¿Cuánto le deben todos los estudiantes a un colegio?
- ¿Cuántos alumnos tiene un colegio?
- ¿Cuál es el estado de cuenta de un colegio o de un estudiante?

Puedes agregar tablas y entidades adicionales si lo consideras necesario (por ejemplo, pagos, planes, status, etc.) siempre que sigan una lógica clara.

Alcance

MODELADO Y PERSISTENCIA

- Definir modelos para:
  - **School**
    Student
    Invoice
- Relacionarlos adecuadamente (por ejemplo: un Student pertenece a un School, una Invoice pertenece a un Student, etc.).
- Uso de **ORM** (por ejemplo, SQLAlchemy o similar).
- Migraciones opcionales, pero suman puntos.

API BACKEND

Implementar al menos los siguientes endpoints (REST o similar):

- **CRUD de Colegios**
- **CRUD de Estudiantes**
- **CRUD de Facturas**
- **Endpoint: Estado de cuenta del Colegio**
  - Ejemplo de respuesta: total facturado, total pagado (si modelas pagos), total pendiente, listado de facturas relevantes.
- **Endpoint: Estado de cuenta del Estudiante**
  - Similar al colegio, pero para un estudiante específico.

Idealmente, la API debe exponer documentación automática (OpenAPI/Swagger).

Requerimientos

REQUERIMIENTOS TÉCNICOS

- **Lenguaje:** Python.
- **Framework: FastAPI** (requerido).
- **Base de datos: PostgreSQL** (requerido).
- **Contenedores:** Proyecto dockerizado con **Docker Compose** que incluya:
    - Servicio de base de datos (Postgres).
    - Servicio Backend (FastAPI).
- El repositorio debe incluir:
    - Instrucciones de uso en el **README**.
    - Comandos para:
        - Levantar el proyecto.
        - Ejecutar pruebas.
        - (Opcional) Cargar datos de ejemplo.

EXTRAS (no obligatorios, pero suman puntos)

- **Cache** (Redis u otro) para algunos endpoints de lectura.
- **Paginación** en endpoints que listan información.
- **Front End simple** (Vue, React, Angular o similar) que consume la API.
- Manejo de **autenticación/autorización** básico.
- Estrategias sencillas de **observabilidad**: logs, métricas, health checks.

SECCIONES A EVALUAR

- Diseño de tablas y modelos ORM.
- Validaciones de datos y reglas de negocio.
- Documentación (README + docstring/comentarios donde aporte valor).
- Pruebas (unitarias e idealmente alguna integración).
- Legibilidad, estructura y complejidad del código.
  (Plus) Uso de cache, paginación, rendimiento y/o Front End.