Tarea - FastAPI EventMaster

Contexto

Te han encargado desarrollar el motor de una plataforma de venta de entradas. Tu misión es construir una API robusta que gestione los recintos (lugares donde se hacen eventos) y los eventos programados, asegurando que nunca se supere el aforo permitido.

Requisitos del Proyecto

1. Arquitectura y Stack

- Framework: FastAPI.
- Base de Datos: PostgreSQL alojada en Aiven.
- ORM: SQLAlchemy o SQLModel.
- Despliegue: La API debe estar operativa en Railway.
- Variables de Entorno: No se deben subir credenciales al repo (usar `.env`).

2. Modelos de Datos

Debes implementar dos tablas relacionadas (1:N):

1. Recinto (Venue):
    - `id` (int, PK)
    - `nombre` (str)
    - `ciudad` (str)
    - `capacidad` (int)
2. Evento (Event):
    - `id` (int, PK)
    - `nombre` (str)
    - `fecha` (datetime/date)
    - `precio` (float)
    - `tickets_vendidos` (int, por defecto 0)
    - `recinto_id` (FK hacia Recinto)

3. Endpoints Obligatorios

- Recintos: CRUD completo (`POST`, `GET`, `PUT`, `DELETE`).
- Eventos:
    - `POST /eventos/`: Al crear un evento, el `precio` no puede ser negativo.
    - `GET /eventos/`: Debe permitir filtrar opcionalmente por `ciudad` (usando el nombre de la ciudad del recinto).
    - `PATCH /eventos/{id}/comprar`: Recibe un entero `cantidad`.
        - Lógica de negocio: Si `tickets_vendidos + cantidad > capacidad_del_recinto`, debe devolver un error 400 Bad Request indicando que no hay aforo suficiente.

Tarea - FastAPI EventMaster

¿Cómo entregarlo?

- Enlace al **Repositorio de GitHub**.
- Enlace a la **URL de Railway** (donde se vea el JSON de bienvenida o el /docs).

Ayuda

¿Cómo filtrar eventos por ciudad?

Para filtrar los eventos por el nombre de la ciudad, recuerda que la **ciudad** no está en la tabla Evento, sino en la tabla Recinto. Por tanto, necesitamos "unirlas" en nuestra consulta.

1. El Esquema de la Consulta (Query)

En tu endpoint de GET /eventos/, debes añadir un parámetro opcional para la ciudad.

```
@app.get("/eventos/")
def leer_eventos(ciudad: str = None, db: Session = Depends(get_db)):
    query = db.query(models.Evento).join(models.Recinto) # Unimos las tablas

    if ciudad:
        # Filtramos por el campo 'ciudad' de la tabla 'Recinto'
        query = query.filter(models.Recinto.ciudad.ilike(f"%{ciudad}%"))

    return query.all()
```

2. Conceptos clave para investigar:

- .join(models.Recinto): Esto le dice a SQLAlchemy: "Busca la relación que conecta Eventos con Recintos".
- .ilike(): Es mucho mejor que usar = porque no distingue entre mayúsculas y minúsculas (ej: buscar "madrid" encontrará "Madrid").
- Relación en los Modelos: Asegúrate de que en tu archivo models.py tengas definida la relación. Algo como:
  - En Evento: recinto = relationship("Recinto", back_populates="eventos")
  - En Recinto: eventos = relationship("Evento", back_populates="recinto")

Tarea - FastAPI EventMaster

Consejo para el Error de Aforo (El PATCH)

Para el endpoint de compra, recuerda seguir este orden lógico:

1. Busca el Evento por su ID.
2. Accede al Recinto asociado a ese evento (gracias a la relación del ORM).
3. Suma tickets_vendidos + cantidad_solicitada.
4. Compara esa suma con recinto.capacidad.
5. Si es mayor, lanza la excepción:

Python

```
raise HTTPException(status_code=400, detail="Aforo insuficiente en el recinto")
```

En un archivo requirements.txt básico para que sepamos qué librerías son imprescindibles figuran las siguientes:

- fastapi: El framework principal.
- uvicorn[standard]: El servidor que pondrá la API en marcha.
- sqlalchemy: El ORM para hablar con la base de datos sin escribir SQL puro.
- psycopg2-binary: El "driver" necesario para que Python pueda conectar con PostgreSQL (Aiven).
- pydantic
- pydantic-settings
- python-dotenv: Fundamental para leer el archivo .env y que Railway reconozca la DATABASE_URL.