DESARROLLO DEL PROCESO BACKEND SEGUNDA ENTREGA

1. Abrir visual studio code, File, open folder, seleccionar carpeta donde esta el backend (SIGUEU\_BACKEND\_API\_AGL).

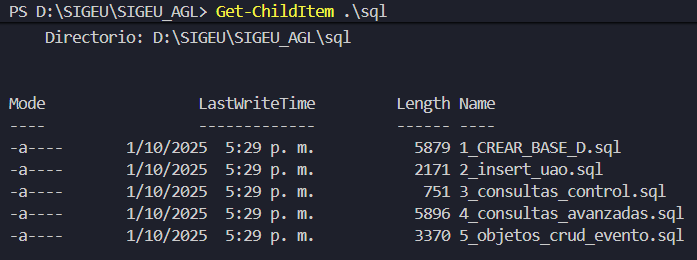
2. Abrir terminal en VSC (PS D:\SIGEU\SIGEU\_Backend\_Api\_AGL>\_ )

3. abrir MySQL Work brench, abrir y activar Xamp.

4. en terminar de VSC:

# Verifica que existe la carpeta 'sql'

Get-ChildItem .\sql



5. EJECUTA EL PROYECTO

5.1 Define la ruta del cliente MySQL de XAMPP

$MYSQL = "C:\xampp\mysql\bin\mysql.exe"



5.2 Crear BD y tablas ejecutando o invocando la consulta: CREAR\_DASE\_D\_ver\_2.sql

Get-Content .\sql\1\_CREAR\_BASE\_D.sql | & $MYSQL -h 127.0.0.1 -P 3306 -u root



5.3 Insertar datos a través de la consulta correspondiente

Get-Content .\sql\2\_insert\_uao.sql | & $MYSQL -h 127.0.0.1 -P 3306 -u root -D uao\_eventos



5.4 Crear OBJETOS SP/funciones/triggers

Get-Content .\sql\5\_objetos\_crud\_evento.sql | & $MYSQL -h 127.0.0.1 -P 3306 -u root -D uao\_eventos

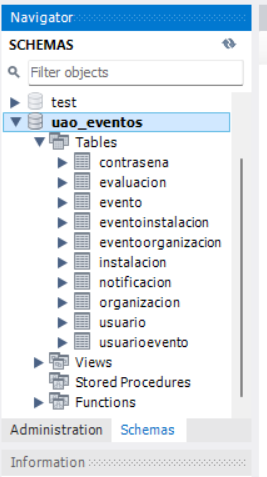


5.5 Verificacion de creación de la BD

Desde MySQL work brench, abrir la conexión que apunta al MySQL de XAMPP ( “Eventos”).

En el panel izquierdo (Navigator, SCHEMAS), botón derecho, Refresh All.

Debe verse el esquema uao\_eventos. Si no aparece, se da clic en el ícono de recargar otra vez.

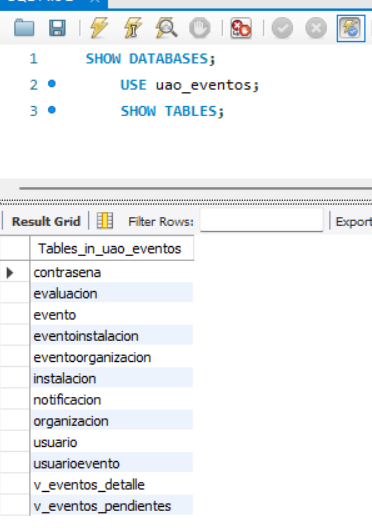


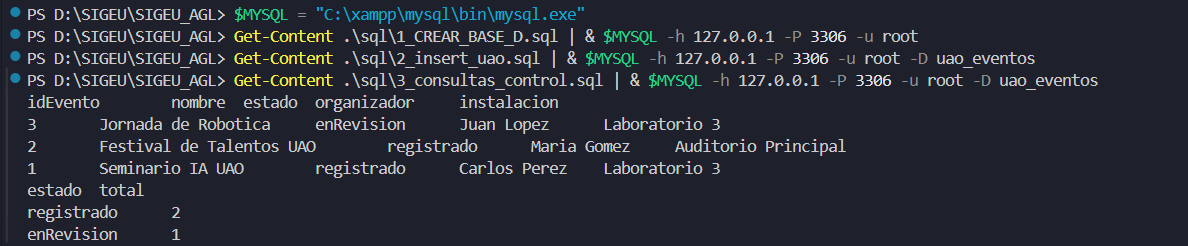
7. Abre un SQL Editor (crear nueva sql - botón del rayo) y ejecutar:

SHOW DATABASES;

USE uao\_eventos;

SHOW TABLES;





Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

8. en la terminal de Visual Studio Code, entrar a la carpeta del backend

cd .\backend\

9. Crea venv con 3.11 y reinstala dependencias

pip3 install virtualenv

10. Usar Python 3.11 para el backend, Es la ruta sencilla porque los drivers async de MySQL tienen soporte estable en 3.11/3.12. luego se crea el entorno con (nombre del entorno : sigeu\_venv):

py -3.11 -m venv sigeu\_venv

10. Se activa el entorno virtual (sigeu\_venv)

sigeu\_venv\Scripts\actívate

# debe salir : (sigeu\_venv) PS D:\SIGEU\SIGEU\_Backend\_Api\_AGL\backend>

11. Instalar seaborn

pip3 install seaborn

12. Instalar pandas

pip3 install pandas --only-binary=:all:

13. Instalar uvicorn

pip3 install uvicorn

14. instalar fastapi

pip3 install fastapi

15. Instalar archivo de requerimientos (librerías , dependencias y demás)

pip3 install -r requirements.txt

NOTA: LOS PASOS 16, 17 Y 18, NO HAY NECESIDAD DE EJECUTARLOS, PUES YA SE TIENE COPIADO Y EDITADO EL ARCHIVO ".env", ES DECIR, PASEN AL PUNTO 19 (YA ARREGLE TODO).

16. Crea tu .env con la cadena de conexión En la carpeta raíz del backend (donde está la carpeta app/):

* # Es un archivo de variables de entorno que NO se versiona (secrets, rutas, puertos).
* # Nos permite cambiar fácil la cadena de conexión (dev/stage/prod) sin tocar el código.
* # En nuestro backend, app/db/\_\_init\_\_.py lee DATABASE\_URL con os.getenv(...).
* # Al lanzar Uvicorn con --env-file .env, esas variables se cargan antes de arrancar la app.
* # se crea desde la raíz del backend: la carpeta que contiene app/ (donde están app/main.py, app/api, etc.).
* # Según la ruta, es: D:\SIGEU\SIGEU\_Backend\_Api\_AGL\backend> (es el lugar correcto).
* # se verifica esto con estos comandos:

pwd

dir app

dir app\main.py

* # dentro de la carpeta "backend" hay un archivo llamado "env.example", su contenido es:

DATABASE\_URL=mysql+asyncmy://user:password@localhost:3306/uao\_eventos

# es de este archivo del cual se habla

17. Copiar "env.example a ".env " Desde la carpeta backend (D:\SIGEU\SIGEU\_Backend\_Api\_AGL\backend>):

Copy-Item .env.example .env

* #debe aparecer el archivo .env con un icono en forma de engranaje.

18. Editar .env, para esto se abre y se cambia la línea DATABASE\_URL=... por la que sirva.

* # Como estás en Windows con MySQL de XAMPP y Python 3.11, se recomienda usar aiomysql:
* #Root SIN contraseña (usado normalmente por XAMPP):

DATABASE\_URL=mysql+aiomysql://root@127.0.0.1:3306/uao\_eventos

* # Root CON contraseña:
* #

DATABASE\_URL=mysql+aiomysql://root:TU\_PASSWORD@127.0.0.1:3306/uao\_eventos

* # se debe tener en cuenta usar 127.0.0.1 (no localhost) para forzar TCP y evitar lío de sockets/named pipes.
* # Si el MySQL que se tiene instalado, está en otro puerto (a veces 3307), se cambia :3306 por el correcto, y listo.

# ¿Y por qué no asyncmy (como venía el ejemplo)?

# Pues porque en el entorno activado, aiomysql es más estable. asyncmy funciona en 3.11, pero si llega a dar error, simplemente se debe mantener aiomysql. (El resto del código no cambia.)

19. Levantar el backend usando ese .env que se acaba de editar, y con el venv activado y desde la misma carpeta backend.

uvicorn --env-file .env app.main:app --reload

# (si uvicorn no se reconoce)

python -m uvicorn --env-file .env app.main:app –reload

20. abrir en el navegador, la siguiente direccion:

http://127.0.0.1:8000/docs

* # debe aparecer swagger UI, correctamente
* # Lo que vemos en http://127.0.0.1:8000/docs es Swagger UI, la interfaz que FastAPI genera automáticamente a partir del OpenAPI de la app nuestra. Eso significa:
  + #El backend está corriendo bien (Uvicorn + conexión a BD OK).
  + # Swagger cargó la documentación de los endpoints que definimos.
  + # ¿Qué aparece en pantalla?
    - Arriba: SIGEU 0.1.0 OAS 3.1 → título/versión de tu API y el estándar OpenAPI usado.

Sección Eventos con 5 endpoints:

POST /api/v1/eventos/ : Crear evento

GET /api/v1/eventos/ : Listar eventos (con filtros)

GET /api/v1/eventos/{id\_evento} : Obtener por id

PUT /api/v1/eventos/{id\_evento} : Actualizar

DELETE /api/v1/eventos/{id\_evento} : Eliminar

Sección Schemas:

EventoCrear (payload de POST),

EventoActualizar (payload de PUT),

EventoOut (respuesta estándar),

ValidationError (errores 422 de Pydantic).

|  |
| --- |
| ¿Qué es la sección “Eventos” y por qué hay 5 endpoints?   * “Eventos” es el recurso REST que expone la API (los registros de la tabla evento). * Cada endpoint es una operación sobre ese recurso:   + POST: crear un evento nuevo.   + GET (colección): listar varios eventos, con filtros.   + GET (detalle): obtener un evento por su id.   + PUT: actualizar un evento existente.   + DELETE: eliminar un evento existente.   REST en simple: trabajas con recursos (eventos) vía rutas predecibles y verbos HTTP (POST/GET/PUT/DELETE).  La API está versionada (/api/v1/...), lo que permite cambios futuros sin romper clientes. |
| Endpoints del recurso “Eventos”  1) POST /api/v1/eventos/ — Crear evento   * Qué hace: inserta un nuevo registro en BD. * Cuerpo (JSON): debe cumplir el esquema EventoCrear (abajo te lo explico). * Respuestas:   + 201 Created + EventoOut (incluye id\_evento y estado="pendiente").   + 422 si el JSON no cumple el esquema (tipos/obligatorios).   + 400 si falla una regla de negocio (p. ej., fecha\_fin < fecha\_inicio). * Idempotencia: no, cada POST crea un nuevo registro.   Ejemplo body:  {  "nombre": "Seminario IA UAO",  "descripcion": "Charlas sobre ML",  "id\_organizador": 1,  "id\_instalacion": 2,  "fecha\_inicio": "2025-11-05T08:00:00",  "fecha\_fin": "2025-11-05T12:00:00",  "categoria": "academico",  "ruta\_aval\_pdf": "/avales/aval1.pdf"  } |
| 2) GET /api/v1/eventos/ — Listar eventos (con filtros)   * Qué hace: devuelve una lista de eventos; puedes filtrar/paginar. * Parámetros de consulta (query):   + q (texto): busca en nombre/descripcion.   + categoria: academico | ludico.   + estado: pendiente | aprobado | rechazado.   + fecha\_ini y fecha\_fin (YYYY-MM-DD) para acotar por fechas.   + limit (1..200) y offset (>=0) para paginar. * Respuestas:   + 200 OK + list[EventoOut]. * Idempotencia/Safety: seguro (no modifica datos) e idempotente.   Ejemplos:   * /api/v1/eventos/?categoria=academico&limit=10 * /api/v1/eventos/?q=IA&fecha\_ini=2025-11-01&fecha\_fin=2025-11-30 |
| 3) GET /api/v1/eventos/{id\_evento} — Obtener por id   * Qué hace: trae el detalle de un evento específico. * Parámetro de ruta: id\_evento (entero). * Respuestas:   + 200 OK + EventoOut.   + 404 si no existe. * Idempotencia/Safety: seguro e idempotente. |
| 4) PUT /api/v1/eventos/{id\_evento} — Actualizar evento   * Qué hace: modifica campos de un evento ya existente. * Cuerpo (JSON): EventoActualizar (todos los campos son opcionales, se actualiza solo lo enviado). * Respuestas:   + 200 OK + EventoOut actualizado.   + 404 si no existe.   + 400 si viola reglas (p. ej., fechas incoherentes).   + 422 si el JSON tiene tipos inválidos. * Idempotencia: sí (mismo body aplicado varias veces deja el mismo estado).   Ejemplo body:  {  "nombre": "Seminario IA UAO - Actualizado",  "fecha\_fin": "2025-11-05T13:00:00"  } |
| 5) DELETE /api/v1/eventos/{id\_evento} — Eliminar evento   * Qué hace: borra un evento por id. * Respuestas:   + 204 No Content si se eliminó.   + 404 si no existe. * Idempotencia: sí (borrar dos veces sigue dejando “no existe”). |

|  |
| --- |
| Sección “Schemas” (modelos de datos)  Los Schemas son clases Pydantic que definen la forma de los JSON que entran/salen. Garantizan validación automática y documentación en Swagger. |
| EventoCrear (payload de POST)   * Para qué: datos obligatorios para crear. * Estructura:   + nombre: str (3..180 chars)   + descripcion: str | null (opcional)   + id\_organizador: int   + id\_instalacion: int   + fecha\_inicio: datetime (ISO 8601, ej. 2025-11-05T08:00:00)   + fecha\_fin: datetime (debe ser ≥ fecha\_inicio)   + categoria: "academico" | "ludico"   + ruta\_aval\_pdf: str (ruta/URL al aval) * Validación: tipos, longitudes; negocio: coherencia de fechas (se valida en la capa services). |
| EventoActualizar (payload de PUT)   * Para qué: actualizaciones parciales (campos opcionales). * Estructura (todos opcionales):   + nombre, descripcion, id\_instalacion, fecha\_inicio, fecha\_fin, categoria, ruta\_aval\_pdf, estado * Regla: si cambias fechas, se revalida fecha\_fin ≥ fecha\_inicio. |
| EventoOut (respuesta estándar)   * Para qué: salida uniformizada en todos los endpoints. * Estructura:   + Todos los campos de EventoCrear más:     - id\_evento: int     - estado: "pendiente" | "aprobado" | "rechazado" * Nota: from\_attributes=True permite construir el schema desde el modelo SQLAlchemy. |
| ValidationError (errores 422 de Pydantic)   * Para qué: describe qué campo falló y por qué. * Estructura típica:   {  "detail": [  { "loc": ["body", "nombre"], "msg": "String should have at least 3 characters", "type": "string\_too\_short" },  { "loc": ["body", "fecha\_fin"], "msg": "Input should be a valid datetime", "type": "datetime\_parsing" }  ]  }  Si aparece: corrige el JSON (tipos/formato), ejecuta de nuevo. |

¿Cómo se conecta esto con el código?

* routes/eventos.py: define las rutas y declara response\_model= (lo que ves como Schemas).
* schemas/evento.py: define EventoCrear, EventoActualizar, EventoOut.
* services/evento.py: valida reglas (fechas), maneja transacciones (commit, refresh) y respuestas 404/400.
* crud/evento.py: hace las queries ORM (SELECT/INSERT/UPDATE/DELETE).
* models/evento.py: mapea la tabla evento.

21. probar los procesos

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

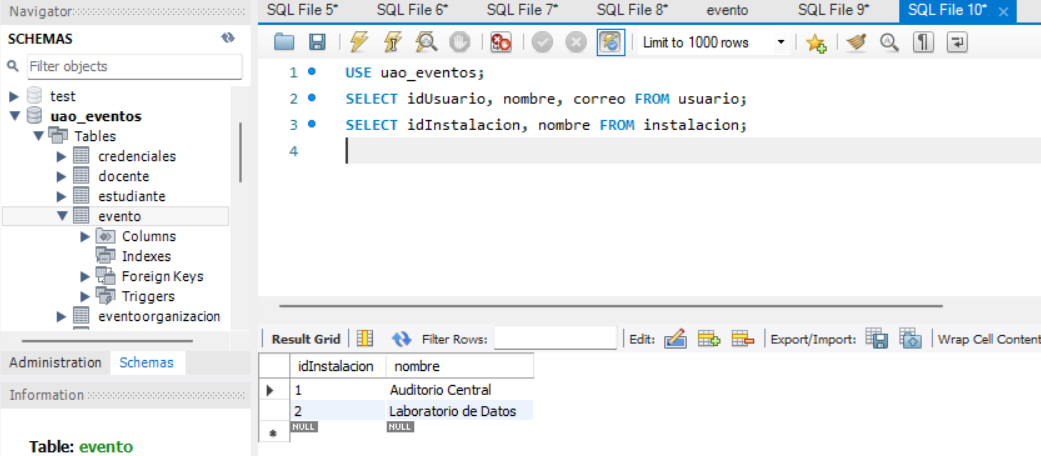
21.1 Antes de crear: validar FKs

El POST necesita IDs válidos para:

* id\_organizador : existe en tabla usuario
* id\_instalacion : existe en tabla instalacion

Si usaste los SQL que te pasé, normalmente hay:

* usuario.idUsuario: 1, 2, 3…
* instalacion.idInstalacion: 1, 2…



21.2 Crear evento — POST /api/v1/eventos/

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Respuestas después de dar clic en Execute:

* 201 Created
* JSON con id\_evento (guárdalo), estado: "pendiente" y todo lo que enviaste.
* Si sale 422: Hay que revisar los campos y formatos coinciden (fecha en ISO YYYY-MM-DDTHH:MM:SS, categoría académico : ludico).
* Si sale 400: fecha\_fin no puede ser menor que fecha\_inicio.
* Si sale 500: los IDs de id\_organizador o id\_instalacion no existen : hay que usar IDs reales (ver paso “VALIDAR FKS”)

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Forma, Patrón de fondo, Rectángulo

El contenido generado por IA puede ser incorrecto.

Forma, Rectángulo

El contenido generado por IA puede ser incorrecto.Forma, Rectángulo

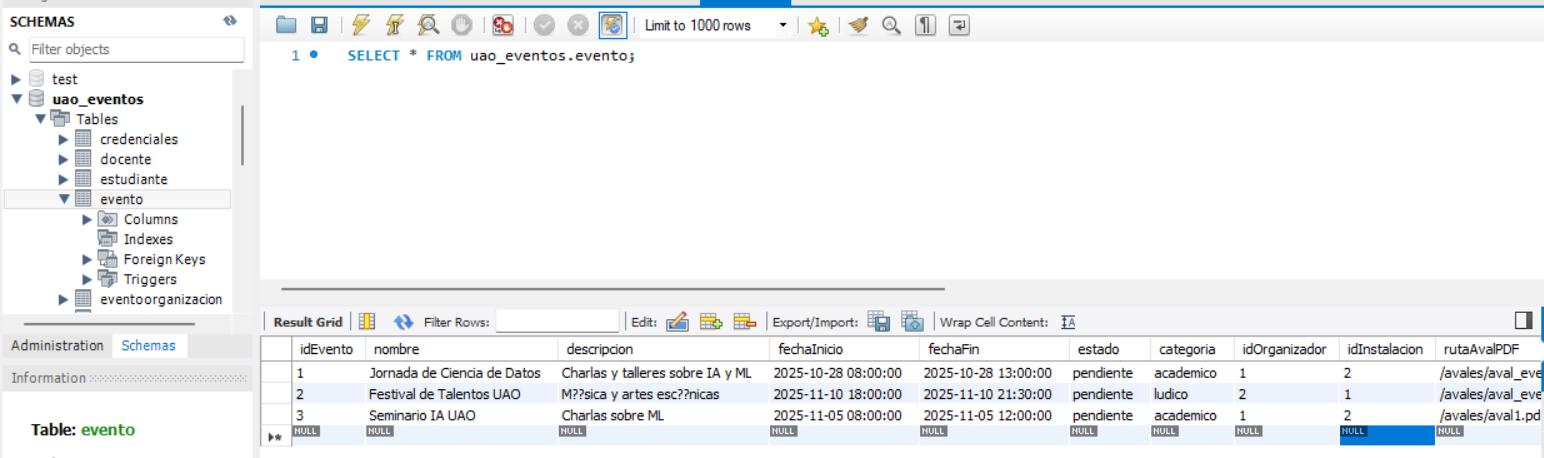
El contenido generado por IA puede ser incorrecto.

Ese 201 Created confirma que el backend insertó el evento en la base con estado: "pendiente"

21.2.1 verifico en MySQL Workbench

* Abrir tu conexión (la de XAMPP).
* En el panel izquierdo SCHEMAS , botón derecho , Refresh All.
* Expandir uao\_eventos , Tables , evento.
* Clic derecho en evento , Select Rows – Limit 1000.
* Se debe ver la fila con idEvento = 3 (o el id que devolvió Swagger), con los campos que enviamos.

|  |
| --- |
| { "nombre": "Seminario IA UAO", "descripcion": "Charlas sobre ML", "id\_organizador": 1, "id\_instalacion": 2, "fecha\_inicio": "2025-11-05T08:00:00", "fecha\_fin": "2025-11-05T12:00:00", "categoria": "academico", "ruta\_aval\_pdf": "/avales/aval1.pdf"} |



Tambien se puede probar con una consulta dentro del mysql work brench

|  |
| --- |
| USE uao\_eventos;  -- 1) Ver los últimos insertados  SELECT idEvento, nombre, fechaInicio, fechaFin, categoria, estado  FROM evento  ORDER BY idEvento DESC  LIMIT 10;  -- 2) Ver el detalle del creado (ajusta el id)  SELECT e.\*  FROM evento e  WHERE e.idEvento = 3;  -- 3) Validar que las FKs apuntan a registros reales  SELECT e.idEvento, e.nombre, u.nombre AS organizador, i.nombre AS instalación  FROM evento e  JOIN usuario u ON u.idUsuario = e.idOrganizador  JOIN instalacion i ON i.idInstalacion = e.idInstalacion  WHERE e.idEvento = 3; |

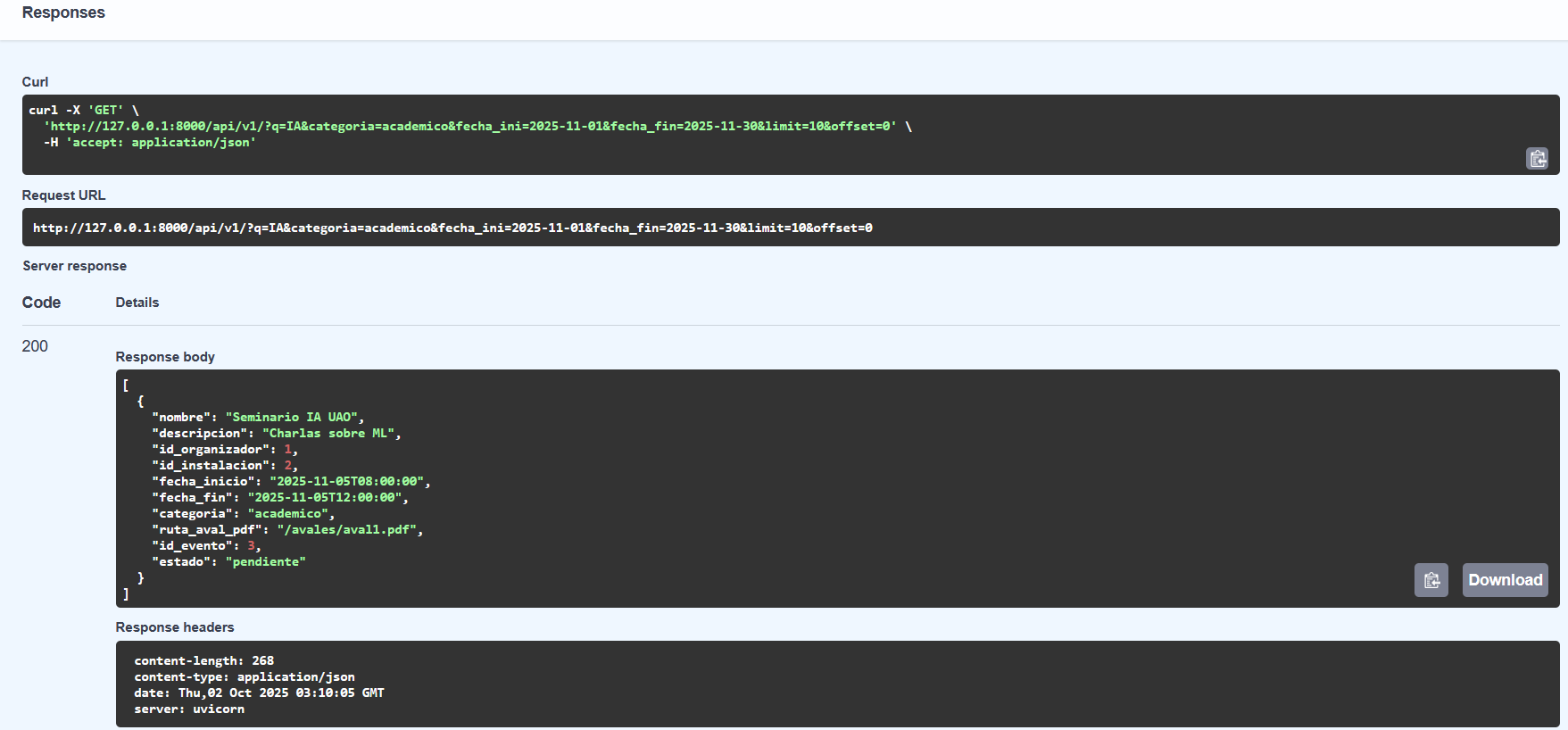
21.3 Listar — GET /api/v1/eventos/

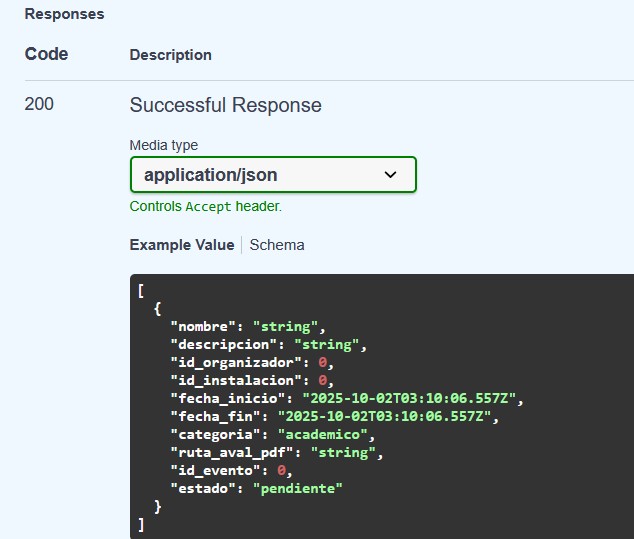
Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Try it out, Execute.

* Respuesta 200 con una lista que incluye el evento recién creado.
* probar filtros opcionales:
  + q=IA (busca en nombre/descripcion)
  + categoria=academico
  + fecha\_ini=2025-11-01
  + fecha\_fin=2025-11-30
  + limit=10
  + offset=0





Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Este GET /api/v1/eventos/ devolvió 200 OK con el evento que se creo y además aplicó bien los filtros (q=IA, categoria=academico, fecha\_ini/fin, limit/offset).

Nota: en Swagger en la aprte de abajo se ve “Example Value” para 200 y 422; estos, son solo ejemplos de la documentación, no el resultado real. El resultado real está en “Response body” .

21.4 Obtener por id : GET /api/v1/eventos/{id\_evento}

Ponemos el id\_evento devuelto en el POST, Execute.

* Respuesta 200 con el objeto completo.
* 404 si pones un ID que no existe.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

El GET por id esta bien: genera un mensaje 200 OK con el evento (en la captura es el id 2, “Festival de Talentos UAO”). Eso confirma que la ruta de detalle funciona y que el objeto en BD coincide con lo que realemtne se esta esperando del proceso.

Lo que se ve en el “Response body” es el EventoOut:

* id\_evento: 2
* nombre, descripcion
* id\_organizador, id\_instalacion
* fecha\_inicio, fecha\_fin (ISO 8601)
* categoria
* estado (por defecto, pendiente)
* ruta\_aval\_pdf

21.5 ACTUALIZAR — PUT /api/v1/eventos/{id\_evento}

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

|  |
| --- |
| {  "nombre": "Seminario IA UAO - Actualizado",  "fecha\_fin": "2025-11-05T13:00:00"  } |

En la terminal se ve 400 Bad Request en un primer intento (coloque una fecha errada para verificar respuesta) y luego 200 OK (segundo intento). El 400 sale cuando el primer body tenía algo inválido (fecha con formato errado, o fecha\_fin menor a la de fecha\_inicio, o con el body vacío). El segundo intento ya pasó todas las validaciones y se guardó.

En la respuesta del 200 ya aparece:

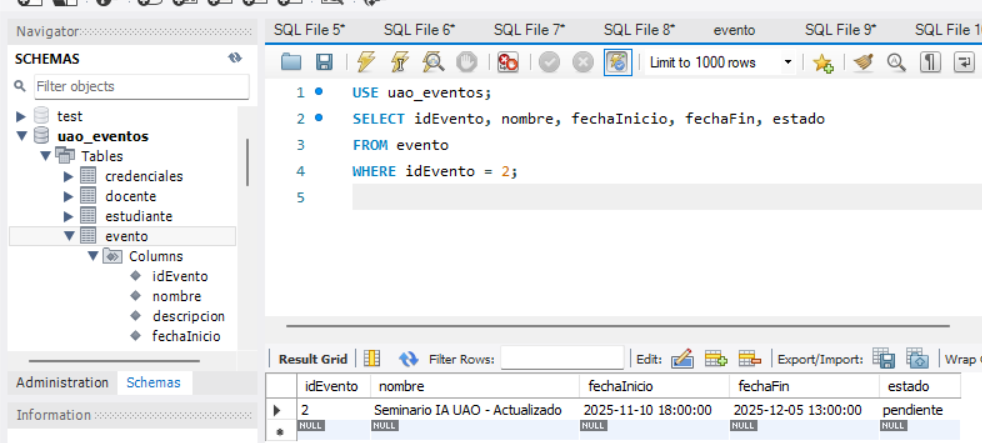
* nombre: "Seminario IA UAO - Actualizado"
* fecha\_fin: "2025-12-05T13:00:00"
* id\_evento: 2
* estado: "pendiente"

21.5.1 Verificar en MySQL Workbench

Se crea una consulta

|  |
| --- |
| USE uao\_eventos;  SELECT idEvento, nombre, fechaInicio, fechaFin, estado  FROM evento  WHERE idEvento = 2; |

Y se debe ver el nombre actualizado y la nueva fechaFin.



21.6 ELIMINAR — DELETE /api/v1/eventos/{id\_evento}

21.6.1 Eliminar evento

1. En Swagger abrir DELETE → Try it out.
2. Colocar id\_evento = 2 , Execute.
3. Esperado: 204 No Content.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Imagen que contiene teclado, cerca, computadora, puesto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto. Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

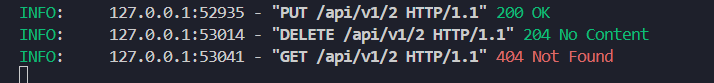
204 No Content: el servidor confirma que borró el recurso y no devuelve cuerpo.

21.6.2 Validar luego que ya no exista:

* Swagger: GET /api/v1/eventos/2 → 404 Not Found.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.



Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto. Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

* 1. t Found: el evento ya no existe, por eso responde “Evento no encontrado”.

21.6.3 Confirmar eliminación de evento en Workbench.

Se crea la siguiente consulta

|  |
| --- |
| USE uao\_eventos;  -- Debe devolver 0  SELECT COUNT(\*) AS existe  FROM evento  WHERE idEvento = 2;  -- Muestra los últimos eventos que quedan  SELECT idEvento, nombre, estado  FROM evento  ORDER BY idEvento DESC  LIMIT 10; |

