

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:\SIGUEU\SIGUEU_Backend_Api_AGL> _)
3. abrir MySQL Work bench, abrir y activar Xamp.
4. en terminar de VSC:

Verifica que existe la carpeta 'sql'

Get-ChildItem .\sql

```
PS D:\SIGUEU\SIGUEU_AGL> Get-ChildItem .\sql
Directorio: D:\SIGUEU\SIGUEU_AGL\sql

Mode                LastWriteTime         Length Name
----                -
-a----          1/10/2025  5:29 p. m.         5879 1_CREAR_BASE_D.sql
-a----          1/10/2025  5:29 p. m.         2171 2_insert_uao.sql
-a----          1/10/2025  5:29 p. m.          751 3_consultas_control.sql
-a----          1/10/2025  5:29 p. m.         5896 4_consultas_avanzadas.sql
-a----          1/10/2025  5:29 p. m.         3370 5_objetos_crud_evento.sql
```

5. EJECUTA EL PROYECTO

5.1 Define la ruta del cliente MySQL de XAMPP

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

```
● PS D:\SIGUEU\SIGUEU_AGL> $MYSQL = "C:\xampp\mysql\bin\mysql.exe"
○ PS D:\SIGUEU\SIGUEU_AGL> █
```

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

```
PS D:\SIGEU\SIGEU_AGL> Get-Content .\sql\1_CREAR_BASE_D.sql | & $MYSQL -h 127.0.0.1 -P 3306 -u root
PS D:\SIGEU\SIGEU_AGL>
```

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

```
PS D:\SIGEU\SIGEU_AGL> Get-Content .\sql\2_insert_uao.sql | & $MYSQL -h 127.0.0.1 -P 3306 -u root -D uao_eventos
PS D:\SIGEU\SIGEU_AGL>
```

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

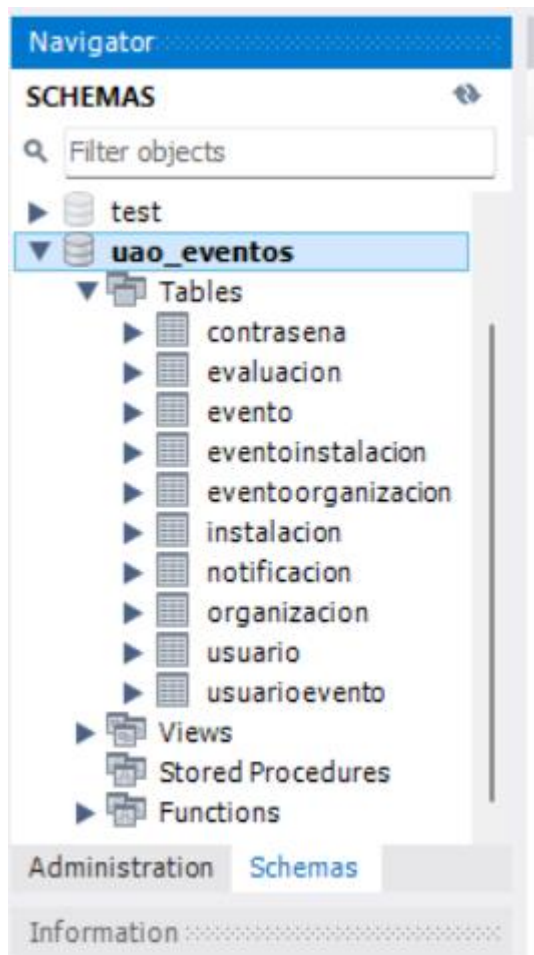
```
PS D:\SIGEU\SIGEU_AGL> Get-Content .\sql\5_objetos_crud_evento.sql | & $MYSQL -h 127.0.0.1 -P 3306 -u root -D uao_eventos
PS D:\SIGEU\SIGEU_AGL>
```

5.5 Verificación de creación de la BD

Desde MySQL work bench, 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.

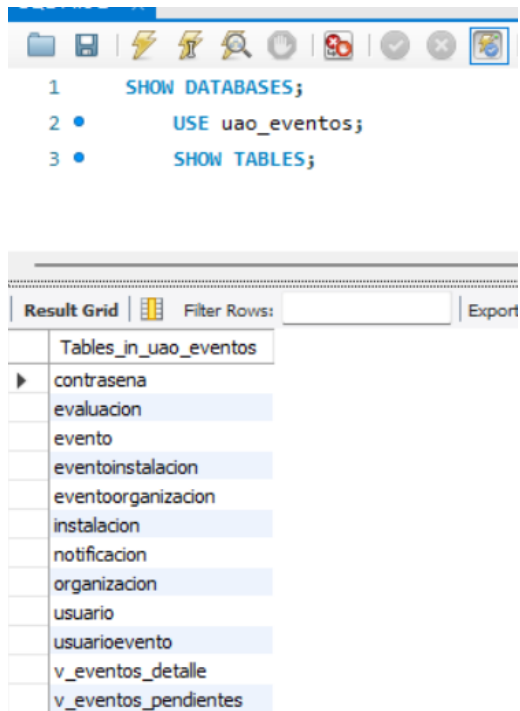


7. Abre un SQL Editor (crear nueva sql - botón del rayo) y ejecutar:

```
SHOW DATABASES;
```

```
USE uao_eventos;
```

```
SHOW TABLES;
```



```

PS D:\SIGEU\SIGEU_AGL> $MYSQL = "C:\xampp\mysql\bin\mysql.exe"
PS D:\SIGEU\SIGEU_AGL> Get-Content .\sql\1_CREAR_BASE_D.sql | & $MYSQL -h 127.0.0.1 -P 3306 -u root
PS D:\SIGEU\SIGEU_AGL> Get-Content .\sql\2_insert_uao.sql | & $MYSQL -h 127.0.0.1 -P 3306 -u root -D uao_eventos
PS D:\SIGEU\SIGEU_AGL> Get-Content .\sql\3_consultas_control.sql | & $MYSQL -h 127.0.0.1 -P 3306 -u root -D uao_eventos
idEvento nombre estado organizador instalacion
3 Jornada de Robotica enRevision Juan Lopez Laboratorio 3
2 Festival de Talentos UAO registrado Maria Gomez Auditorio Principal
1 Seminario IA UAO registrado Carlos Perez Laboratorio 3
estado total
registrado 2
enRevision 1

```

```

PS D:\SIGEU\SIGEU_AGL> Get-Content .\sql\4_consultas_avanzadas.sql | & $MYSQL -h 127.0.0.1 -P 3306 -u root -D uao_eventos
idUsuario nombre eventos_creados
3 Juan Lopez 1
2 Maria Gomez 1
1 Carlos Perez 1
idInstalacion nombre horas_reservadas_30d
2 Laboratorio 3 12
1 Auditorio Principal 7
3 Salon 201 4
aprobados evaluados tasa_aprobacion_pct pendientes
0 0 NULL 3
estado horas_promedio
aprobado 355.0000
rechazado 378.0000
idUsuario nombre aprobados evaluados tasa_aprobacion_pct
3 Juan Lopez 1 1 100.00
2 Maria Gomez 0 1 0.00

```

```

categoria      aprobados      evaluados      tasa_aprobacion_pct
academico      1              1              100.00
ludico 0       1              0.00
pendientes_0a2d pendientes_3a7d pendientes_8a14d pendientes_mas14d
3              0              0
categoria      duracion_promedio_min
academico      360.0000
ludico 210.0000
anio  mes      categoria      total_eventos
2025  11      academico      2
2025  11      ludico 1
organizacion categoria      total_eventos
Comunidad AI Cali      academico      1
Fundacion Talentos      ludico 1
PS D:\SIGEU\SIGEU_AGL>

```

```

sql > 5_objetos_crud_evento.sql
1  -- =====
2  -- 5_objetos_crud_evento.sql
3  -- Esquema: uao_eventos
4  -- Contiene: 3 funciones, 3 vistas y 4 triggers (>=3) con propósito de gestión
5  -- =====
6
7  USE uao_eventos;
8
9  -- =====
10 -- =====  FUNCIONES  =====
11 -- =====
12
13 -- [función 1]
14 -- Nombre: fn_dias_restantes_evento
15 -- ¿Qué es?: Utilitaria de calendario para un evento puntual.
16 -- ¿Para qué sirve?: Medir cuántos días faltan para el inicio de un evento.
17 -- Propósito: Apoyar recordatorios, SLAs y priorizar logística.
18 -- Información que arroja: Entero con días (positivo=por iniciar, negativo=ya pasó)
19
PROBLEMS (473) OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB SPELL CHECKER (473) POLYGLOT NOTEBOOK
PS D:\SIGEU\SIGEU_AGL> Get-Content .\sql\5_objetos_crud_evento.sql | & $MYSQL -h 127.0.0.1 -P 3306 -u root -D uao_eventos
PS D:\SIGEU\SIGEU_AGL>

```

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\activate
```

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 \geq 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 $fecha_fin \geq fecha_inicio$.

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:

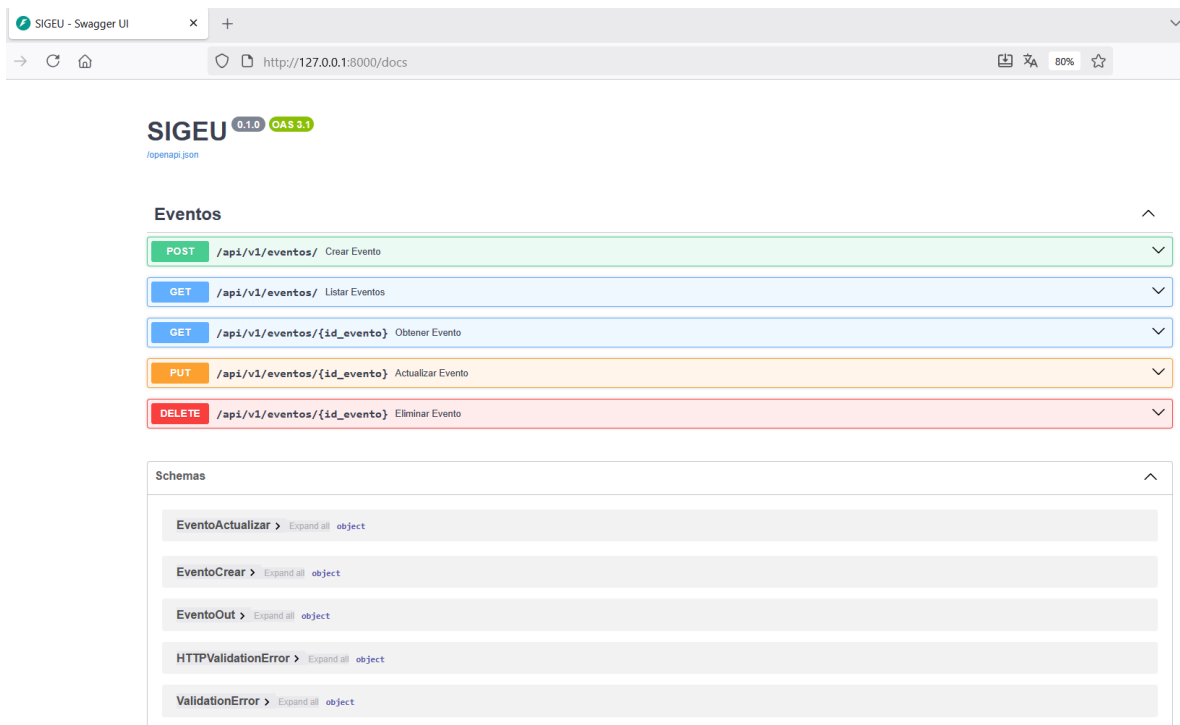
```
{
  "detail": [
    { "loc": ["body", "nombre"], "msg": "String should have at least 3
characters", "type": "string_too_short" },
    { "loc": ["body", "fecha_fin"], "msg": "Input should be a valid datetime",
"type": "datetime_parsing" }
  ]
}
```

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



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...

The screenshot shows a SQL client interface with a sidebar on the left displaying a database schema. The main window shows a SQL query and its results.

Schema:

- test
 - uao_eventos
 - Tables
 - credenciales
 - docente
 - estudiante
 - evento
 - Columns
 - Indexes
 - Foreign Keys
 - Triggers
 - eventoorganizacion

SQL Query:

```
1 • USE uao_eventos;
2 • SELECT idUsuario, nombre, correo FROM usuario;
3 • SELECT idInstalacion, nombre FROM instalacion;
4 •
```

Result Grid:

idInstalacion	nombre
1	Auditorio Central
2	Laboratorio de Datos
NULL	NULL

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

SIGEU 0.1.0 CAS 3.1
OpenAPI JSON

Eventos

POST `/api/v1/eventos/` Create Evento

Parameters

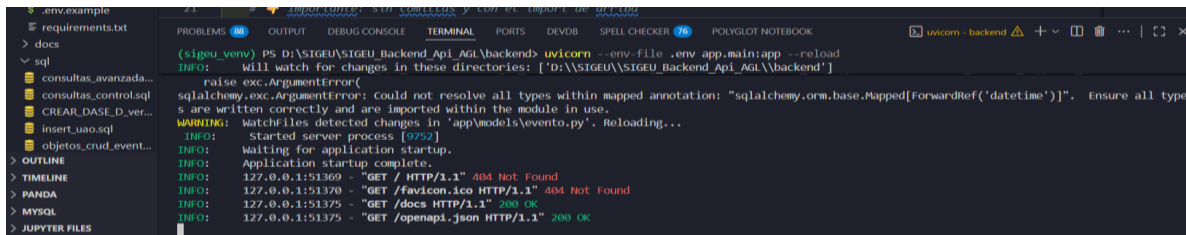
No parameters

Request body required

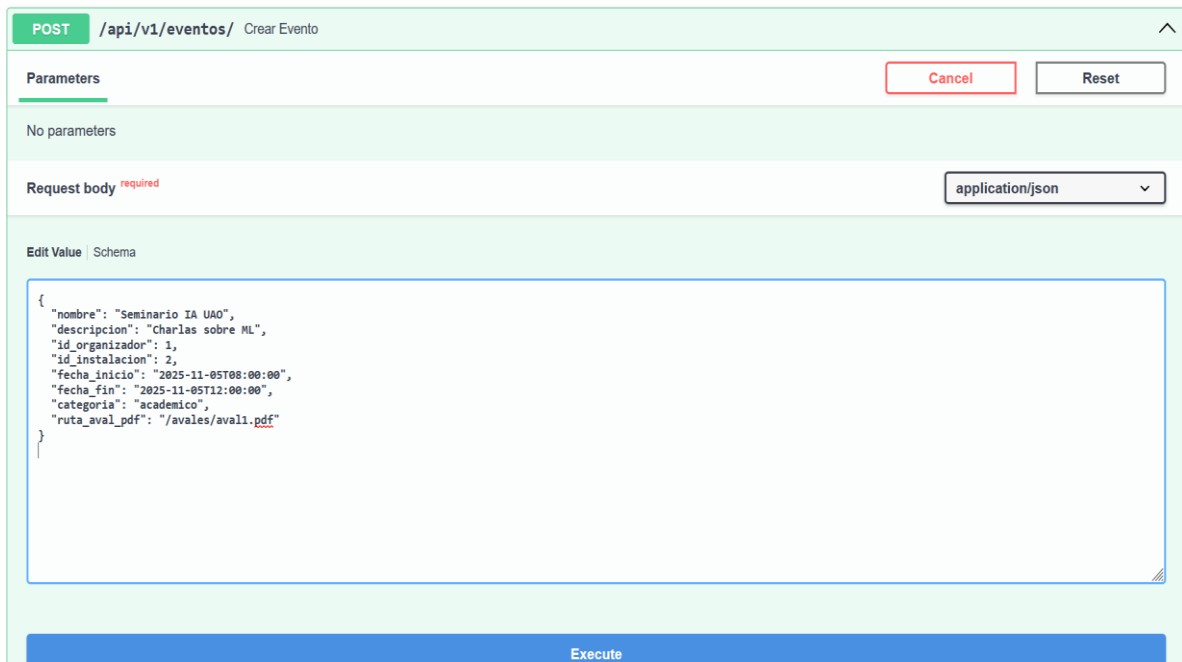
application/json

Example Value | Schema

```
{
  "nombre": "string",
  "descripcion": "string",
  "id_organizador": 0,
  "id_instalacion": 0,
  "fecha_inicio": "2025-10-02T03:10:14.793Z",
  "fecha_fin": "2025-10-02T03:10:14.793Z",
  "categoria": "academico",
  "ruta_archivo": "string"
}
```

```
(sigeu_venv) PS D:\SIGEU\SIGEU_Backend_Api_AGI\backend> uvicorn --env-file .env app.main:app --reload
INFO: Will watch for changes in these directories: ['D:\\SIGEU\\SIGEU_Backend_Api_AGI\\backend']
raise exc.ArgumentError(
    sqlalchemy.exc.ArgumentError: could not resolve all types within mapped annotation: "sqlalchemy.orm.base.Mapped[forwardref('datetime')]". Ensure all type
s are written correctly and are imported within the module in use.
WARNING: Watchfiles detected changes in 'app\models\evento.py'. Reloading...
INFO: Started server process [9752]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:51369 - "GET / HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:51370 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:51375 - "GET /docs HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:51375 - "GET /openapi.json HTTP/1.1" 200 OK
```



POST /api/v1/eventos/ Crear Evento

Parameters Cancel Reset

No parameters

Request body required application/json

Edit Value Schema

```
{
  "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"
}
```

Execute

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”)

```
> siget_venv
  sql
  consultas_avanzada...
> OUTLINE
> TIMELINE
> PANDA
> MYSQL
> JUPYTER FILES
```

```
INFO: waiting for application shutdown.
INFO: Application shutdown complete.
INFO: Finished server process [14240]
INFO: Started server process [2296]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:52230 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:52230 - "GET /openapi.json HTTP/1.1" 200 OK
INFO: 127.0.0.1:52263 - "POST /api/v1/ HTTP/1.1" 201 Created
```

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/v1/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "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/avall.pdf"
  }'
```

Request URL

```
http://127.0.0.1:8000/api/v1/
```

Server response

Code Details

```
201
Response 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/avall.pdf",
  "id_evento": 3,
  "estado": "pendiente"
}
```

Response headers

Mostrar

422 Validation Error No links

Media type
application/json

Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

Response headers

```
content-length: 266
content-type: application/json
date: Thu, 02 Oct 2025 02:21:32 GMT
server: uvicorn
```

Responses

Code	Description	Links
201	Successful Response	No links

Media type
application/json

Controls Accept header

Example Value | Schema

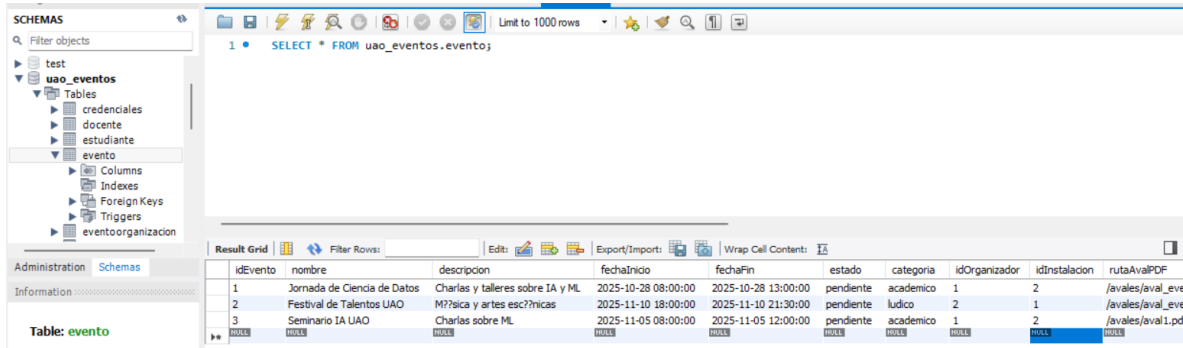
```
{
  "nombre": "string",
  "descripcion": "string",
  "id_organizador": 0,
  "id_instalacion": 0,
  "fecha_inicio": "2025-10-02T02:21:32.676Z",
  "fecha_fin": "2025-10-02T02:21:32.676Z",
  "categoria": "academico",
  "ruta_eval_pdf": "string",
  "id_evento": 0,
  "estado": "pendiente"
}
```

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"}
```



idEvento	nombre	descripcion	fechaInicio	fechaFin	estado	categoria	idOrganizador	idInstalacion	rutaAvalPDF
1	Jornada de Ciencia de Datos	Charlas y talleres sobre IA y ML	2025-10-28 08:00:00	2025-10-28 13:00:00	pendiente	academico	1	2	/avales/aval_eve
2	Festival de Talentos UAO	Música y artes escénicas	2025-11-10 18:00:00	2025-11-10 21:30:00	pendiente	ludico	2	1	/avales/aval_eve
3	Seminario IA UAO	Charlas sobre ML	2025-11-05 08:00:00	2025-11-05 12:00:00	pendiente	academico	1	2	/avales/aval1.pdf

También 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/



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

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/v1/?q=IA&categoria=academico&fecha_ini=2025-11-01&fecha_fin=2025-11-30&limit=10&offset=0' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/api/v1/?q=IA&categoria=academico&fecha_ini=2025-11-01&fecha_fin=2025-11-30&limit=10&offset=0
```

Server response

Code	Details
------	---------

200	
-----	--

Response 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/avall.pdf",
    "id_evento": 3,
    "estado": "pendiente"
  }
]
```

Response headers

```
content-length: 268
content-type: application/json
date: Thu, 02 Oct 2025 03:10:05 GMT
server: uvicorn
```

Responses

Code	Description
------	-------------

200	Successful Response
-----	---------------------

Media type

application/json



Controls Accept header.

Example Value | Schema

```
[
  {
    "nombre": "string",
    "descripcion": "string",
    "id_organizador": 0,
    "id_instalacion": 0,
    "fecha_inicio": "2025-10-02T03:10:06.557Z",
    "fecha_fin": "2025-10-02T03:10:06.557Z",
    "categoria": "academico",
    "ruta_aval_pdf": "string",
    "id_evento": 0,
    "estado": "pendiente"
  }
]
```

422

Validation Error

Media type

application/json



Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

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.

GET

/api/v1/{id_evento}

Obtener Evento

^

Parameters

Cancel

Name	Description
id_evento * required integer (path)	<input type="text" value="2"/>

Execute

```
INFO: 127.0.0.1:52230 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:52230 - "GET /openapi.json HTTP/1.1" 200 OK
INFO: 127.0.0.1:52263 - "POST /api/v1/ HTTP/1.1" 201 Created
INFO: 127.0.0.1:52650 - "GET /api/v1/?q=IA&categoria=academico&fec
INFO: 127.0.0.1:52760 - "GET /api/v1/2 HTTP/1.1" 200 OK
```

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/v1/2' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/api/v1/2
```

Server response

Code	Details
------	---------

200	
-----	--

Response body

```
{
  "nombre": "Festival de Talentos UAO",
  "descripcion": "Música y artes escénicas",
  "id_organizador": 2,
  "id_instalacion": 1,
  "fecha_inicio": "2025-11-10T18:00:00",
  "fecha_fin": "2025-11-10T21:30:00",
  "categoria": "ludico",
  "ruta_aval_pdf": "/avales/aval_evento2.pdf",
  "id_evento": 2,
  "estado": "pendiente"
}
```

Response headers

```
content-length: 288
content-type: application/json
date: Thu, 02 Oct 2025 03:28:48 GMT
server: uvicorn
```

Responses

Code	Description
------	-------------

200	Successful Response
-----	---------------------

Media type

application/json

Controls Accept header.

Example Value | Schema

```
{
  "nombre": "string",
  "descripcion": "string",
  "id_organizador": 0,
  "id_instalacion": 0,
  "fecha_inicio": "2025-10-02T03:28:49.378Z",
  "fecha_fin": "2025-10-02T03:28:49.378Z",
  "categoria": "academico",
  "ruta_aval_pdf": "string",
  "id_evento": 0,
  "estado": "pendiente"
}
```

422

Validation Error

Media type

application/json



Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

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 realmente 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}

Parameters

CancelReset

Name	Description
id_evento <small>* required</small> integer (path)	<input type="text" value="2"/>

Request body required

application/json ▼

Edit Value | Schema

```
{  "nombre": "Seminario IA UAO - Actualizado",  "fecha_fin": "2025-11-05T13:00:00"}  
```

```
{  "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:

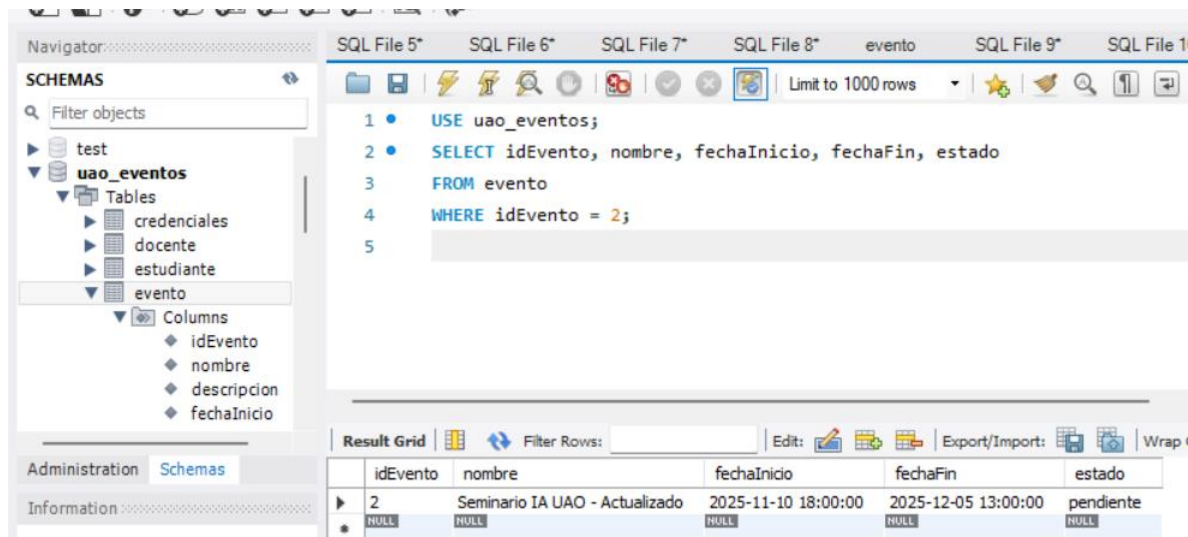
- nombre: "Seminario IA UAO - Actualizado"
- fecha_fin: "2025-12-05T13:00:00"
- id_evento: 2
- estado: "pendiente"

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.

DELETE /api/v1/{id_evento} Eliminar Evento

Parameters

Name	Description
id_evento * required	
integer (path)	

Execute

```

INFO: 127.0.0.1:52650 - "GET /api/v1/?q=IA&categoria=academico&fecha_in
INFO: 127.0.0.1:52760 - "GET /api/v1/2 HTTP/1.1" 200 OK
INFO: 127.0.0.1:52931 - "PUT /api/v1/2 HTTP/1.1" 400 Bad Request
INFO: 127.0.0.1:52935 - "PUT /api/v1/2 HTTP/1.1" 200 OK
INFO: 127.0.0.1:53014 - "DELETE /api/v1/2 HTTP/1.1" 204 No Content

```

Responses

Curl

```
curl -X 'DELETE' \
'http://127.0.0.1:8000/api/v1/2' \
-H 'accept: */*'
```

Request URL

```
http://127.0.0.1:8000/api/v1/2
```

Server response

Server response

Code	Details
204	<p>Response headers</p> <pre>content-type: application/json date: Thu,02 Oct 2025 04:13:49 GMT server: uvicorn</pre>

Responses

Code	Description
204	Successful Response
422	Validation Error

Media type

application/json ▼

Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

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.

GET

/api/v1/{id_evento} Obtener Evento

Parameters

Name	Description
<div><div>id_evento</div><div><div>* required</div></div><div>integer</div><div>(path)</div></div>	<div>2</div>

Execute

```
INFO: 127.0.0.1:52935 - "PUT /api/v1/2 HTTP/1.1" 200 OK
INFO: 127.0.0.1:53014 - "DELETE /api/v1/2 HTTP/1.1" 204 No Content
INFO: 127.0.0.1:53041 - "GET /api/v1/2 HTTP/1.1" 404 Not Found
```

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/v1/2' \
  -H 'accept: application/json'
```

Request URL

http://127.0.0.1:8000/api/v1/2

Server response

Code	Details
404	Error: Not Found

Undocumented

Response body

```
{
  "detail": "Evento no encontrado"
}
```

Response headers

```
content-length: 33
content-type: application/json
date: Thu, 02 Oct 2025 04:19:21 GMT
server: uvicorn
```

Responses

Code	Description
200	Successful Response

Media type

application/json

Controls Accept header.

Example Value

Schema

```
{
  "nombre": "string",
  "descripcion": "string",
  "id_organizador": 0,
  "id_instalacion": 0,
  "fecha_inicio": "2025-10-02T04:19:22.457Z",
  "fecha_fin": "2025-10-02T04:19:22.457Z",
  "categoria": "academico",
  "ruta_aval_pdf": "string",
  "id_evento": 0,
  "estado": "pendiente"
}
```

422

Validation Error

Media type

application/json



Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

404t 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;

Navigator: SCHEMAS

Filter objects

- test
- uao_eventos
 - Tables
 - credenciales
 - docente
 - estudiante
 - evento
 - Columns
 - idEvento
 - nombre
 - descripcion
 - fechaInicio

Administration Schemas

Information

Table: evento

SQL File 5* SQL File 6* SQL File 7* SQL File 8* evento

Limit to 1000 rows

```
5 FROM evento
6 WHERE idEvento = 2;
7
8 -- Muestra los últimos eventos que quedan
9 • SELECT idEvento, nombre, estado
10 FROM evento
11 ORDER BY idEvento DESC
12 LIMIT 10;
13
```

Result Grid

	idEvento	nombre	estado
▶	3	Seminario IA UAO	pendiente
	1	Jornada de Ciencia de Datos	pendiente
•	NULL	NULL	NULL