

## PROYECTO SISTEMA DE GESTIÓN DE EVENTOS SIGEU

RODRIGO ANDRÉS GÓMEZ LÓPEZ. Código: 2247014

DAVID HERNÁNDEZ PAZ. Código: 2247003

DANIEL ALEXANDER BRAND GARCÍA. Código: 2246133

SEBASTIÁN MANRIQUE MEJIA. 2246988

MICHAEL MACOWLI CARDONA RODRIGUEZ. Código: 2246268

### ALMACENAMIENTO DE DATOS

Presentado a:

JHON EDER MASSO DAZA



UNIVERSIDAD AUTÓNOMA DE OCCIDENTE

INGENIERÍA DE DATOS E INTELIGENCIA ARTIFICIAL

SANTIAGO DE CALI

2025

## CONTENIDO

1. OBJETIVOS.....	5
1.1 OBJETIVO GENERAL .....	5
1.2 OBJETIVOS ESPECIFICOS .....	5
2. METODOLOGÍA .....	6
2.1 SCRUM – Trabajo en equipo .....	6
2.2 CRISP-DM – Desarrollo del sistema .....	7
2.3. TECNOLOGÍAS UTILIZADAS. ....	8
3. ANALISIS DE REQUERIMIENTOS .....	11
3.1 ANALIS DEL PROYECTO Y DEFINICION DE REQUERIMIENTOS .....	11
4. MODELADO DE LA BASE DE DATOS .....	20
4.1 MODELO DE DATOS RELACIONAL (MER/MR) .....	20
4.1.2 MR .....	22
4.1.3 TABLAS.....	23
4.1.4 Inserción de datos.....	38
4.1.5 CONSULTAS DE VERIFICACIÓN Y CONTROL .....	42
5. PRUEBAS BACKEND EN SWAGGER (FastAPI – SIGEU).....	45
5.1 CREA UN EVENTO (POST /api/v1/) .....	48
5.1.1 Resultados de creación .....	49
5.1.2 Resultados de Listar Eventos. ....	55
5.1.3 Resultados de consulta para Obtener evento por id.....	59
5.1.4 Resultados de actualizar evento /api/v1/{id_evento}.....	63
5.1.5 Resultados de eliminar evento /api/v1/{id_evento} .....	68
6. PRUEBAS REALIZADAS EN POSTMAN.....	73
6.1 EJECUCION DE PRUEBAS EN POSTMAN .....	73
6.1.1 Ejecución De Creación De Evento .....	79
6.1.2 Evidencias de creación de listar eventos EN POSTMAN .....	83
6.1.3 Evidencias de obtener evento por id EN POSTMAN .....	88

6.1.4 Evidencias de actualizar evento por id en POSTMAN .....	91
6.1.5 Evidencias de eliminar evento por id en POSTMAN .....	94
6.1.6 Evidencias de confirmación de obtener evento eliminado en POSTMAN.	98
6.1.7 Enlace con acceso público a las colecciones creadas.....	102
<b>7. CONSULTAS AVANZADAS CON SQL .....</b>	<b>110</b>
<b>7.1 CONSULTA 1 – PRODUCTIVIDAD POR ORGANIZADOR.....</b>	<b>112</b>
<b>7.2 CONSULTA 2: HORAS RESERVADAS POR INSTALACIÓN .....</b>	<b>113</b>
<b>7.3 CONSULTA TASA DE APROBACIÓN GLOBAL Y BACKLOG .....</b>	<b>115</b>
.....	116
<b>7.4 CONSULTA TIEMPO PROMEDIO A DECISIÓN (HORAS) POR ESTADO .....</b>	<b>116</b>
.....	117
<b>7.5 CONSULTA TASA DE APROBACIÓN POR ORGANIZADOR .....</b>	<b>117</b>
<b>7.6 CONSULTA TASA DE APROBACIÓN POR CATEGORÍA .....</b>	<b>119</b>
<b>7.7 CONSULTA BACKLOG POR ANTIGÜEDAD (SLA) PARA REVISIÓN .....</b>	<b>120</b>
<b>7.8 CONSULTA SOLAPAMIENTOS POR INSTALACIÓN (CONTEO DE CONFLICTOS)</b>	<b>122</b>
.....	122
<b>7.9 CONSULTA EVENTOS PRÓXIMOS (7 DÍAS) CON RIESGO DE CONFLICTO.....</b>	<b>124</b>
.....	125
<b>7.10 CONSULTA DURACIÓN PROMEDIO POR CATEGORÍA (MINUTOS).....</b>	<b>125</b>
<b>7.11 CONSULTA ORGANIZADORES CON MÁS DE 3 EVENTOS ACTIVOS (FECHAS FUTURAS) .....</b>	<b>126</b>
<b>7.12 CONSULTA EVENTOS POR MES Y CATEGORÍA (ÚLTIMOS 12 MESES).....</b>	<b>128</b>
<b>7.13 CONSULTA NOTIFICACIONES POR TIPO Y RECEPTOR (ÚLTIMOS 30 DÍAS) .</b>	<b>129</b>
<b>7.14 CONSULTA CALIDAD DE DATOS: EVENTOS SIN EXACTAMENTE UN ORGANIZADOR PRINCIPAL .....</b>	<b>131</b>
<b>7.15 CONSULTA PENDIENTES VENCIDOS (FECHAFIN PASADA SIN DECISIÓN FINAL).....</b>	<b>132</b>
<b>7.16 CONSULTA ALIANZAS EXTERNAS POR CATEGORÍA (TOP ORGANIZACIONES).</b>	<b>134</b>
.....	134
<b>7.17 CONCLUSIONES EJECUCION DE CONSULTAS.....</b>	<b>135</b>
<b>8. OBJETOS ALMACENADOS EN MYSQL .....</b>	<b>135</b>

8.1 FUNCIONES.....	136
8.1.1 Función fn_dias_restantes_evento(p_id).....	136
8.1.2 Función fn_evento_duracion_horas(p_id) .....	136
8.1.3 Función fn_horas_instalacion_30d .....	137
8.2 VISTAS.....	138
8.2.1 Vista v_eventos_pendientes.....	138
8.2.2 Vista v_evento_x_instalacion.....	139
8.3.3 Vista v_conflictos_instalacion.....	139
8.3 TRIGGERS.....	140
8.3.1 Trigger trg_ue_check_rol (BEFORE INSERT) .....	140
8.3.2 Trigger trg_ue_unico_principal_ins (BEFORE INSERT).....	141
8.3.3 Trigger trg_ue_unico_principal_upd (BEFORE UPDATE) .....	142
8.3.4 Trigger trg_eval_after_insert (AFTER INSERT) .....	142
8.3.5 Trigger trg_eval_after_update (AFTER UPDATE).....	143
ANEXO 1. EJECUCION PASO A PASO DEL PROYECTO.....	144
ANEXO 2. ACCESO GITHUB – BACKEND .....	144
ANEXO 3. ACCESO PRUEBAS POSTMAN.....	144

## **1. OBJETIVOS**

### **1.1 OBJETIVO GENERAL**

Implementar dos soluciones de almacenamiento, una relacional (SQL) y otra documental (NoSQL), para el Sistema de Gestión de Eventos Universitarios (SIGEU), con el fin de respaldar su backend inicial y comparar su eficiencia, escalabilidad y adaptabilidad.

### **1.2 OBJETIVOS ESPECIFICOS**

- 1.2.1 Modelar e implementar la solución de almacenamiento relacional (SQL)**
- 1.2.2 Modelar e implementar la solución de almacenamiento documental (NoSQL)**
- 1.2.3 Desarrollar un backend mínimo con API REST**
- 1.2.4 Definir, ejecutar y documentar pruebas comparativas entre SQL y NoSQL**

## 2. METODOLOGÍA

Para el desarrollo de este proyecto se abordarán dos metodologías complementarias. La primera enfocada al trabajo en equipo, y la otra aplicada al desarrollo técnico del sistema. Adicionalmente, se definen las tecnologías que se utilizaran para el desarrollo.

En este sentido, las metodologías elegidas son:

1. SCRUM para el trabajo en equipo, ya que permite organizar el proyecto en sprints cortos, asignar responsabilidades claras y fomentar la colaboración continua entre los integrantes. Esta metodología garantiza que los avances se entreguen de manera progresiva y que los ajustes se realicen de forma ágil.
2. CRISP-DM para la metodología de desarrollo, considerando que, aunque fue creada para minería de datos, se adapta muy bien a un proyecto como SIGEU. Esto porque plantea un ciclo claro: comprender el problema, preparar los datos, modelar, evaluar y desplegar, lo cual se ajusta al flujo de trabajo que implica construir y comparar soluciones de almacenamiento SQL y NoSQL.

### 2.1 SCRUM – Trabajo en equipo

SCRUM es una metodología ágil de gestión de proyectos que divide el trabajo en ciclos cortos llamados *sprints* y promueve la comunicación constante entre los miembros del equipo.

Aplicación en SIGEU:

- Roles:

- *Product Owner (PO)*: líder del equipo (define prioridades y comunicación con el docente).
  - *Scrum Master*: miembro que guía el cumplimiento de la metodología y resuelve bloqueos.
  - *Equipo de desarrollo*: todos los integrantes, responsables de las tareas técnicas.
- Artefactos: backlog de tareas (lista completa de actividades), sprint backlog (tareas de cada ciclo) e incrementos (resultados parciales).
- Eventos:
  - *Sprint planning*: planificación semanal de tareas.
  - *Daily stand-up*: reunión breve para revisar avances y bloqueos.
  - *Sprint review*: muestra de resultados en cada entrega.
  - *Retrospectiva*: análisis de mejoras para el siguiente ciclo.

## 2.2 CRISP-DM – Desarrollo del sistema

CRISP-DM (Cross Industry Standard Process for Data Mining) es una metodología estructurada para proyectos de datos, que organiza el desarrollo en seis fases iterativas: comprensión, preparación, modelado, evaluación y despliegue.

Aplicación en SIGEU:

- Comprensión del negocio: análisis del enunciado y definición de requerimientos.
- Comprensión de los datos: identificación de entidades clave (eventos, usuarios, organizaciones, documentos).

- Preparación de datos: construcción del MER/MR para SQL y definición del modelo documental en NoSQL.
- Modelado: implementación de la base relacional en MySQL y de colecciones en MongoDB.
- Evaluación: pruebas de consultas, validación de la API y comparación SQL vs NoSQL.
- Despliegue académico: documentación, sustentación y entrega de resultados.

### 2.3. TECNOLOGÍAS UTILIZADAS.

Dentro del desarrollo del presente proyecto se hace uso de las siguientes tecnologías:

- XAMPP
  - Uso: Paquete que incluye Apache, MySQL, PHPMyAdmin y otros servicios.
  - Aporte: Provee un entorno local fácil de instalar para correr el motor MySQL y gestionar la base de datos desde PHPMyAdmin.
- MySQL Workbench
  - Uso: Herramienta oficial de diseño y administración de MySQL.
  - Aporte: Permite modelar gráficamente el MER/MR, ejecutar scripts SQL, crear consultas simples y anidadas, y administrar la base relacional.

- MySQL (SQL relacional)
  - Uso: Motor de base de datos relacional en el que se implementará el MER/MR con tablas, relaciones, restricciones, vistas y procedimientos.
  - Aporte: Ofrece integridad referencial, soporte a consultas complejas y robustez para el backend del sistema.
- MongoDB (NoSQL documental)
  - Uso: Motor NoSQL basado en documentos JSON para representar información de eventos y usuarios con flexibilidad.
  - Aporte: Escalabilidad y adaptación para consultas rápidas y estructuras de datos no rígidas.
- Swagger / OpenAPI
  - Uso: Documentación interactiva de la API REST.
  - Aporte: Prueba y validación de endpoints por parte del equipo.
- Postman
  - Uso: Ejecución de pruebas sobre la API y validación de respuestas.
  - Aporte: Facilita la verificación de casos de uso funcionales y pruebas de rendimiento.
- GitHub
  - Uso: Repositorio de código y documentación colaborativa.
  - Aporte: Control de versiones, registro de aportes y Wiki del proyecto.
- Trello / Jira
  - Uso: Organización de tareas bajo metodología ágil (SCRUM).

- Aporte: Seguimiento de backlog, roles y sprints.
- Python (FastAPI)
  - Uso: Framework backend para exponer la API mínima conectada a SQL y NoSQL.
  - Aporte: Comunicación entre bases de datos y pruebas de endpoints.
- Visual Studio Code
  - Uso: Entorno de desarrollo de código.
  - Aporte: Integración con GitHub, ejecución de scripts, soporte para SQL y MongoDB.

## 11. PlantUML / StarUML

- Uso: Creación de diagramas UML (clases, procesos, secuencia).
- Aporte: Documentación gráfica del análisis y diseño.

### 3. ANALISIS DE REQUERIMIENTOS

En un proyecto de desarrollo de software, los requerimientos son la base para guiar el diseño, la implementación y las pruebas. En este sentido, dentro de la metodología SCRUM (trabajo en equipo), los requerimientos se reflejan en el Product Backlog, es decir, en la lista priorizada de funcionalidades a construir.

Por otra parte, en la metodología CRISP-DM (desarrollo), este punto corresponde a la fase de Comprensión del Negocio y de los Datos, donde se identifican las necesidades del sistema y se expresan en requerimientos claros.

Por lo anterior, en el proyecto SIGEU, los requerimientos se dividen en:

1. Requerimientos Funcionales (RF): describen qué debe hacer el sistema.
2. Requerimientos No Funcionales (RNF): describen cómo debe comportarse el sistema (rendimiento, seguridad, escalabilidad, etc.).
3. Reglas de Negocio (RN): condiciones que siempre deben cumplirse según la lógica institucional.
4. Actores y Roles: usuarios que interactúan con el sistema, con sus responsabilidades y permisos.

#### 3.1 ANALIS DEL PROYECTO Y DEFINICION DE REQUERIMIENTOS

Para este ejercicio, es necesario hacer lectura del enunciado del proyecto y poder definir los requerimientos:

Párrafo 1:

*“Este proyecto tiene como objetivo principal el modelado e implementación de dos soluciones de almacenamiento destinadas a respaldar el desarrollo inicial del backend de una aplicación web para la gestión de eventos en una institución universitaria. Una de las soluciones empleará bases de datos relacionales y la otra base de datos NoSQL, lo que permitirá comparar y evaluar diferentes enfoques de almacenamiento orientados a garantizar una gestión eficiente de los datos y un funcionamiento óptimo de la futura aplicación.”*

De acuerdo al texto anterior, el proyecto busca construir dos soluciones de almacenamiento de datos para el sistema SIGEU (Gestión de Eventos Universitarios):

- Una solución con base de datos relacional (SQL).
- Otra solución con base de datos NoSQL (documental).

El propósito es respaldar el backend inicial de la aplicación y, al mismo tiempo, comparar ambos enfoques, evaluando su capacidad de:

- Manejar los datos con eficiencia.
- Garantizar escalabilidad y adaptación.
- Asegurar un rendimiento óptimo de la aplicación en el futuro.

En este sentido los requerimientos iniciales son los siguientes:

Tabla 1. Requerimientos parciales 1.

REQUERIMIENTOS FUNCIONALES (RF)	
RF-01	Implementar almacenamiento relacional (SQL) para SIGEU.
RF-02	Implementar almacenamiento NoSQL (documental) para SIGEU.
RF-03	Desarrollar un backend mínimo de prueba que conecte con ambas bases.
RF-04	Permitir la ejecución de consultas y operaciones básicas sobre ambos almacenamientos.
REQUERIMIENTOS NO FUNCIONALES (RNF)	
RNF-01	El sistema debe ofrecer eficiencia en las operaciones (consultas rápidas).
RNF-02	Debe ser escalable, soportando crecimiento en usuarios y datos.
RNF-03	Debe ser adaptable, permitiendo cambios futuros sin perder rendimiento.
RNF-04	Se debe comparar y documentar SQL vs NoSQL en métricas de latencia, facilidad y escalabilidad.
MODELO DE DATOS (ALTO NIVEL, INICIAL)	
Entidades centrales	Evento, Usuario/Rol, Organización externa, Documentos (PDFs).
REGLAS DE NEGOCIO	
RN-01	El proyecto debe entregar dos soluciones de almacenamiento (SQL y NoSQL).
RN-02	La entrega debe incluir un informe comparativo entre ambos enfoques.

<b>PROCESOS PRINCIPALES</b>	
P1	Implementar almacenamiento SQL.
P2	Implementar almacenamiento NoSQL.
P3	Ejecutar pruebas de validación.
P4	Comparar resultados y elaborar informe.
<b>ACTORES (MARCADORES INICIALES)</b>	
Rol-1	Equipo de desarrollo
Rol-2	Docente
Rol-3	Los roles de usuario final (estudiante, docente, secretario académico)
<b>API MÍNIMA ESPERADA</b>	
API-1	Endpoint de salud: GET /health.
API-2	Diagnóstico SQL: GET /diagnostico/sql.
API-3	Diagnóstico NoSQL: GET /diagnostico/nosql.
<b>PRUEBAS</b>	
PRUEBA - 1	Verificar que se crean las tablas en SQL y las colecciones en NoSQL.
PRUEBA – 2	Validar consultas básicas en ambos entornos.
PRUEBA - 3	Medir tiempo de respuesta en operaciones clave.
<b>PENDIENTES / SUPUESTOS</b>	
P/S -1	Confirmar tecnologías exactas: (ej. MySQL y MongoDB)
P/S - 2	Definir framework backend para pruebas (Node.js, Python FastAPI, etc.).
<b>RIESGOS INICIALES</b>	

RIESG - 1	Subestimar el trabajo al implementar dos modelos de datos paralelos.
RIESG - 2	No definir criterios claros de comparación.

Párrafo 2.

*“Es importante destacar que el proyecto no contempla el desarrollo completo del backend, sino que se centra exclusivamente en la implementación de las soluciones de almacenamiento, garantizando que los modelos y bases de datos diseñados sean consistentes, escalables y cumplan con las necesidades de la aplicación en su etapa inicial de desarrollo.”*

Ahora bien, este parte del enunciado aclara que el proyecto no incluye la construcción del backend completo de la aplicación. Básicamente, el foco está en la implementación de las soluciones de almacenamiento (SQL y NoSQL), asegurando que:

- Los modelos y bases de datos sean consistentes (esquemas correctos y sin contradicciones).
- Sean escalables (puedan crecer en datos y usuarios).
- Respondan adecuadamente a las necesidades de la aplicación en su fase inicial (backend mínimo para pruebas, no despliegue completo).

Tabla 2. Requerimientos parciales 2.

REQUERIMIENTOS FUNCIONALES (RF)	
RF-05	Implementar únicamente las soluciones de almacenamiento (SQL y NoSQL) sin desarrollar todo el backend.
RF-06	Asegurar que las bases de datos creadas soporten las operaciones necesarias del backend mínimo.
REQUERIMIENTOS NO FUNCIONALES (RNF)	
RNF-05	Los modelos deben ser consistentes (sin ambigüedades en esquemas ni contradicciones en relaciones).
RNF-06	Las soluciones deben ser escalables, soportando crecimiento progresivo de datos.
RNF-07	Las soluciones deben estar alineadas a las necesidades iniciales de la aplicación (no se pide sistema productivo, solo pruebas académicas).
MODELO DE DATOS (ALTO NIVEL, INICIAL)	
Entidades centrales	<p>Evento, Usuario/Rol, Organización externa, Documentos (PDFs).</p> <p>En esta fase se reafirma que los modelos deben ser consistentes y escalables; en este sentido, se refuerza la identificación de las entidades mencionadas</p>
REGLAS DE NEGOCIO	
RN-03	El sistema debe excluir el desarrollo del backend completo.
RN-04	Los modelos deben cumplir con las necesidades iniciales de la aplicación (fase temprana).
PROCESOS PRINCIPALES (REFORZANDO PÁRRAFO 1)	

P5	Limitar el alcance al almacenamiento, excluyendo backend completo
P6	Validar consistencia y escalabilidad de los modelos antes de pruebas.
<b>ACTORES (MARCADORES INICIALES)</b>	
Rol-1	Equipo de desarrollo (responsable de asegurar consistencia/escalabilidad)
Rol-2	Docente (validador del cumplimiento del alcance limitado)
Rol-3	Los roles de usuario final (estudiante, docente, secretario académico)
<b>API MÍNIMA ESPERADA</b>	
	Nota: Se mantiene como API de prueba básica (health y diagnósticos).
	Nota 2: Se confirma que no se debe extender hacia backend completo.
<b>PRUEBAS</b>	
PRUEBA – 4	Validar consistencia del modelo (integridad referencial en SQL, esquema válido en NoSQL).
PRUEBA – 5	Simular escalabilidad con dataset de prueba mayor a lo mínimo esperado.
<b>PENDIENTES / SUPUESTOS</b>	
P/S -3	Confirmar tamaño de dataset para validar escalabilidad.
P/S – 4	Definir criterios de consistencia para comparar SQL vs NoSQL.
<b>RIESGOS INICIALES</b>	

RIESG – 3	Intentar desarrollar backend más allá del alcance definido (riesgo de dispersión).
RIESG – 4	Diseñar modelos poco escalables que no resistan pruebas de crecimiento.

### Párrafo 3

*“En una institución universitaria se desea construir una aplicación que permita llevar a cabo el registro, aprobación y publicación de los eventos organizados por la comunidad académica (docentes y/o estudiantes) de los distintos programas adscritos a las unidades académicas de la universidad.”*

Tabla 3. Requerimientos parciales 3.

REQUERIMIENTOS FUNCIONALES (RF)	
RF-07	Permitir el registro de eventos por parte de docentes y estudiantes.
RF-08	Permitir la aprobación de eventos por parte de la secretaría académica.
RF-09	Permitir la publicación de eventos aprobados
REQUERIMIENTOS NO FUNCIONALES (RNF)	
RNF-08	El sistema debe asegurar la trazabilidad del evento desde su registro hasta su publicación.
MODELO DE DATOS (ALTO NIVEL, INICIAL)	
Entidades centrales	Entidad Evento (ya identificada en P1).
	Relación con Usuario (Docente/Estudiante como organizadores).
	Relación con UnidadAcadémica (programa/unidad adscrita).

REGLAS DE NEGOCIO	
RN-05	Todo evento debe pasar por las fases de registro, aprobación, publicación.
PROCESOS PRINCIPALES (REFORZANDO PÁRRAFO 1)	
P7	Registrar evento.
P8	Aprobar evento
P9	Publicar evento
ACTORES (MARCADORES INICIALES)	
Rol-4	Estudiante (registrar eventos).
Rol-5	Docente (registrar eventos).
Rol-6	Secretaría académica (aprobar/publicar eventos).
API MÍNIMA ESPERADA	
API-4	POST /eventos : registrar evento.
API-5	PATCH /eventos/{id}/aprobar : aprobar evento.
API-6	PATCH /eventos/{id}/publicar : publicar evento.
PRUEBAS	
PRUEBA – 6	Validar que un estudiante pueda registrar un evento.
PRUEBA – 7	Verificar que un evento no pueda publicarse sin aprobación previa.
PENDIENTES / SUPUESTOS	
P/S – 5	Confirmar qué información mínima requiere un evento (ej. título, tipo, fecha, lugar).
RIESGOS INICIALES	
RIESG – 5	Posibilidad de que eventos sean publicados sin pasar por aprobación (inconsistencia).

## 4. MODELADO DE LA BASE DE DATOS

### 4.1 MODELO DE DATOS RELACIONAL (MER/MR)

En esta etapa modelamos el “QUE” debe persistir y cómo se relaciona, para soportar los procesos de registro, evaluación, publicación y participación externa.

Ahora bien, en cuanto a la metodología de desarrollo CRISP-DM, este punto hace referencia a la etapa de Modeling, usando los hallazgos de Business Understanding y Data Understanding (requerimientos consolidados) para construir estructuras que garanticen consistencia y trazabilidad.

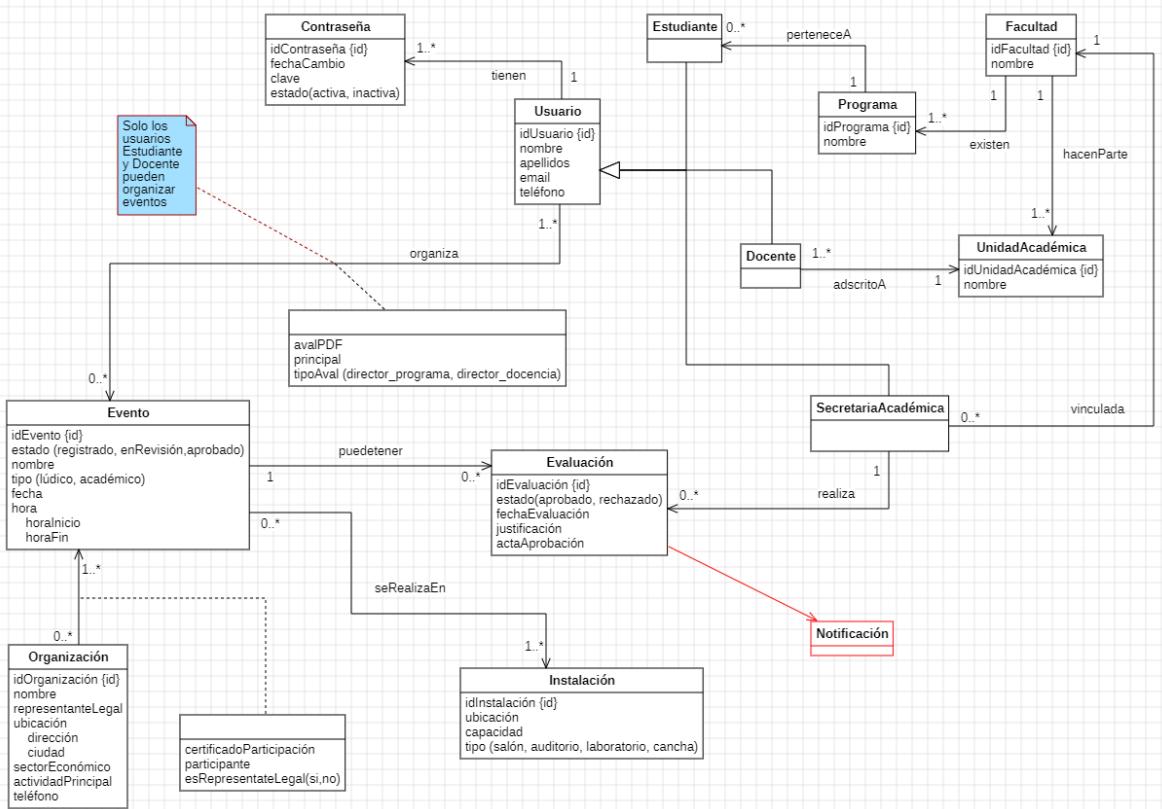
Por otra parte, en cuanto a la metodología de trabajo en equipo (SCRUM): este entregable hace parte del Sprint 2 como base del desarrollo técnico y de las pruebas.

En este punto, se estructura el modelado teniendo en cuenta las entidades principales. Dentro de los pasos relevantes están:

1. MER Y MR
2. Crea el esquema, tablas, PK/FK y restricciones
3. Llena tablas con datos de la UAO (correos, instalaciones, eventos, etc.).
4. agrega vistas, triggers y demás objetos para el flujo de negocio (aprobaciones, notificaciones, etc.).

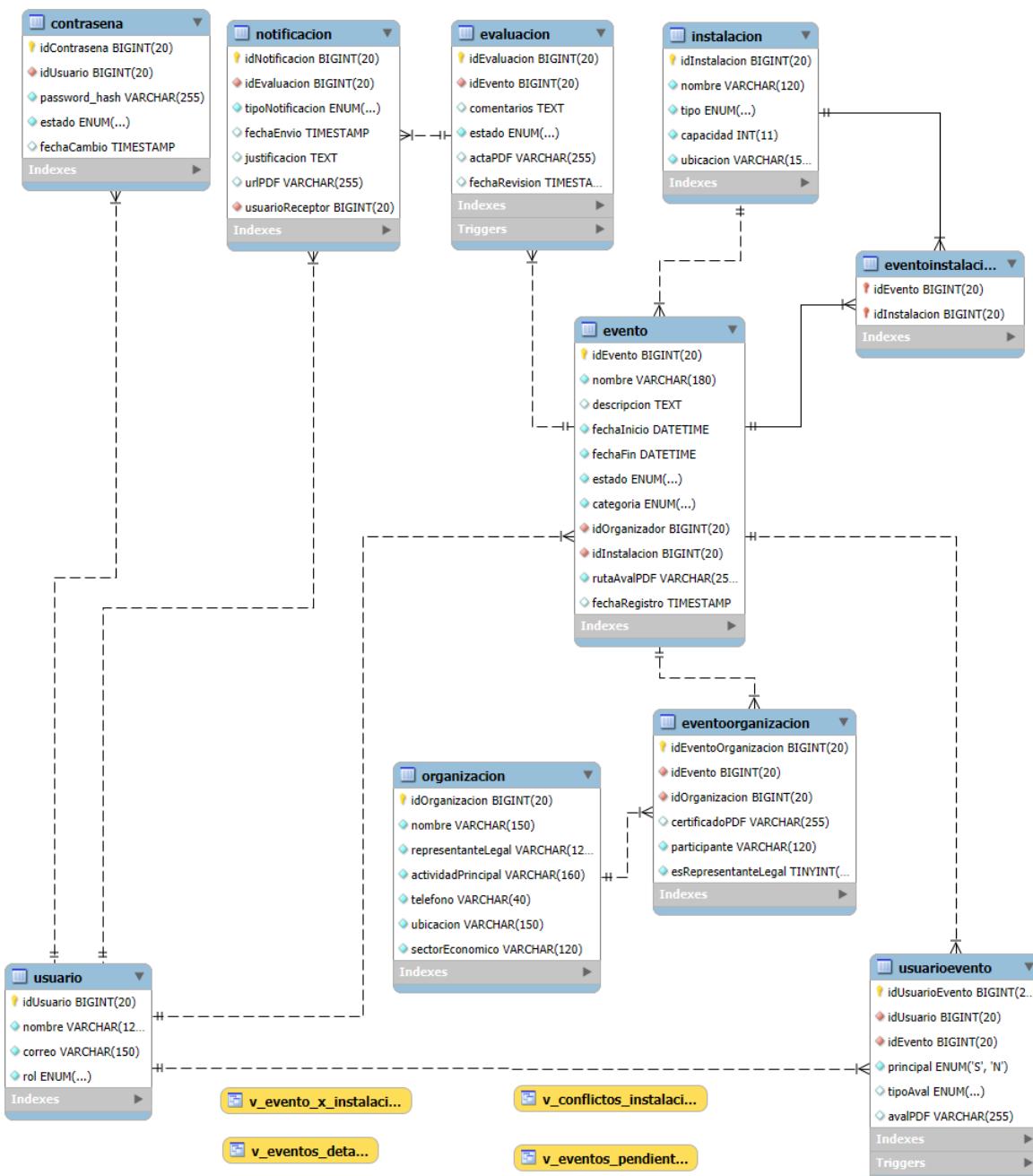
#### 4.1.1 MER

A continuación se muestra el MER ajustado según análisis de requerimientos y sugerencias del docente.



#### 4.1.2 MR

A continuación se muestra el MR elaborado a partir del MER



#### 4.1.3 TABLAS

En primer lugar, se creo una consulta con el nombre 1\_CREAR\_BASE\_D.sql. Esta consulta define el esquema uao\_eventos, tablas, claves foráneas, tipos ENUM y restricciones. Esta consulta es la base del sistema; y asegura que la estructura refleje el MR/MER final y que la integridad referencial esté garantizada desde el inicio.

Para este punto es necesario primero crear la base de datos a partir de la siguiente orden:

```
DROP DATABASE IF EXISTS uao_eventos;  
CREATE DATABASE uao_eventos CHARACTER SET utf8mb4 COLLATE  
utf8mb4_unicode_ci;  
CLEAR
```

Luego se procede a crear las tablas como se indica a continuación.

##### 4.1.3.1 Tabla usuario

Propósito: Personas de la plataforma (docente, estudiante, secretaria).

PK: idUsuario (BIGINT, AI)

Campos principales

- nombre (VARCHAR(120), NOT NULL)
- correo (VARCHAR(150), NOT NULL, UNIQUE)
- rol (ENUM('docente','estudiante','secretariaAcademica'), NOT NULL)

Relaciones

- 1–N con evento (como organizador).
- 1–N con contrasena.
- N–N con evento a través de usuarioevento (organizadores/aval).
- 1–N como receptor de notificacion.

Importancia: saber quién organiza, quién recibe notificaciones, y segmentar métricas por rol (p.ej., ¿cuántos eventos organizan los docentes vs. estudiantes?).

```
CREATE TABLE usuario (
    idUsuario BIGINT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(120) NOT NULL,
    correo VARCHAR(150) NOT NULL UNIQUE,
    rol ENUM('docente','estudiante','secretarioAcademico') NOT NULL
);
```

Campo	Tipo	Nulo	Clave	Descripción
idUsuario	BIGINT AI	No	PK	Identificador del usuario
nombre	VARCHAR(120)	No		Nombre completo
correo	VARCHAR(150)	No	UNIQUE	Correo institucional
rol	ENUM('docente','estudiante','secretariaAcademica')	No		Perfil dentro del sistema
Utilidad: segmentar métricas por rol; identificar organizadores y destinatarios de notificaciones				

#### 4.1.3.2 TABLA contraseña

Propósito: Estado y hash de la clave del usuario (auditoría básica).

PK: idContrasena

FK: idUsuario: usuario(idUsuario)

Campos

- estado (ENUM('activa','inactiva'), DEFAULT 'activa')
- clave/password\_hash (VARCHAR(255))

- fechaCambio (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP)

Importancia: controles de seguridad y auditoría ( por ejemplo claves caducadas, usuarios inactivos).

```
CREATE TABLE contrasena (
    idContrasena BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    idUsuario BIGINT UNSIGNED NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    estado ENUM('activa','inactiva') NOT NULL,
    fechaCambio TIMESTAMP NULL DEFAULT NULL ON UPDATE
    CURRENT_TIMESTAMP,
    CONSTRAINT fk_contra_usuario
    FOREIGN KEY (idUsuario) REFERENCES usuario(idUsuario)
    ON UPDATE CASCADE ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Campo	Tipo	Nulo	Clave	Descripción
idContrasena	BIGINT AI	No	PK	Identificador del registro de clave
idUsuario	BIGINT	No	FK:usuario	Usuario dueño de la clave
clave / password_has h	VARCHAR(255)	No		Hash de la contraseña
estado	ENUM('activa','inactiva')	No		Estado de la clave
fechaCambio	TIMESTAMP	Sí		Último cambio
Utilidad: auditoría de seguridad básica.				

#### 4.1.3.3 TABLA instalacion

Propósito: Espacios físicos para eventos.

PK: idInstalacion

Campos

- nombre (VARCHAR(120))
- tipo (ENUM('salon','laboratorio','auditorio','otro'))
- capacidad (INT, CHECK (capacidad>0))
- ubicacion (VARCHAR(150))

Relaciones

- N–N con evento por eventoinstalacion.

Importancia: planeación de espacios (ocupación, top de salones usados, topes de capacidad).

```
CREATE TABLE instalacion (
    idInstalacion BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre        VARCHAR(120)      NOT NULL,
    tipo          ENUM('salon','laboratorio','auditorio','otro') NOT NULL,
    capacidad     INT              NOT NULL,
    ubicacion     VARCHAR(150)      NOT NULL,
    CONSTRAINT chk_inst_capacidad CHECK (capacidad > 0)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Campo	Tipo	Nulo	Clave	Descripción
idInstalacion	BIGINT AI	No	PK	Identificador del espacio
nombre	VARCHAR(120)	No		Nombre del espacio
tipo	ENUM('salon','laboratorio','auditorio','otro')	No		Clasificación

capacidad	INT (CHECK > 0)	No	Aforo
ubicacion	VARCHAR(150)	No	Ubicación física
Utilidad: planificación de espacios y análisis de ocupación.			

#### 4.1.3.4 TABLA organización

Propósito: Organizaciones externas que participan.

PK: idOrganizacion

Campos

- nombre, representanteLegal, actividadPrincipal, telefono, ubicacion, sectorEconomico

Relación

- N–N con evento por eventoorganizacion.

Importancia: alianzas más activas, sectores con mayor participación.

```
CREATE TABLE organizacion (
    idOrganizacion BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(150) NOT NULL,
    representanteLegal VARCHAR(120) NOT NULL,
    actividadPrincipal VARCHAR(160) NOT NULL,
    telefono VARCHAR(40) NOT NULL,
    ubicacion VARCHAR(150) NOT NULL,
    sectorEconomico VARCHAR(120) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Campo	Tipo	Nulo	Clave	Descripción
idOrganizacion	BIGINT AI	No	PK	Identificador
nombre	VARCHAR(150)	No		Razón social
representanteLegal	VARCHAR(120)	No	RL	

actividadPrincipal	VARCHAR(160)	No		Objeto social
telefono	VARCHAR(40)	No		Contacto
ubicacion	VARCHAR(150)	No		Dirección
sectorEconomico	VARCHAR(120)	No		Sector
Utilidad: seguimiento de alianzas externas.				

#### 4.1.3.5 TABLA evento

Propósito: Actividades de la gestión (unidad central del dominio).

PK: idEvento

Campos

- nombre (VARCHAR(180))
- descripcion (TEXT, NULL)
- fechalinicio, fechaFin (DATETIME, CHECK fechaFin>=fechalinicio)
- estado (ENUM según versión:  
'registrado','enRevision','aprobado','rechazado' o  
'pendiente','aprobado','rechazado')
- categoria (ENUM('academico','ludico'))
- idOrganizador (FK a usuario)
- rutaAvalPDF (VARCHAR(255))
- fechaRegistro (TIMESTAMP, default now)

Relaciones

- 1–N con usuario (un creador).
- N–N con instalacion (eventoinstalacion).
- N–N con organizacion (eventoorganizacion).

- 1–N con usuarioevento (organizadores/aval); la regla de negocio asegura un principal por evento.
- 1–N con evaluacion (historial de revisiones).

Importancia: pipeline de eventos, % aprobación/rechazo, tiempos de aprobación, uso por categoría.

```
CREATE TABLE evento (
    idEvento      BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre        VARCHAR(180)      NOT NULL,
    descripcion   TEXT            NULL,
    fechalinicio DATETIME        NOT NULL,
    fechaFin     DATETIME        NOT NULL,
    estado        ENUM('registrado','enRevision','aprobado','rechazado') NOT NULL,
    categoria     ENUM('academico','ludico') NOT NULL,
    idOrganizador BIGINT UNSIGNED    NOT NULL, -- FK usuario
    idInstalacion BIGINT UNSIGNED    NOT NULL, -- FK instalacion
    rutaAvalPDF   VARCHAR(255)       NOT NULL,
    fechaRegistro TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_evento_organizador
        FOREIGN KEY (idOrganizador) REFERENCES usuario(idUsuario)
        ON UPDATE CASCADE ON DELETE RESTRICT,
    CONSTRAINT fk_evento_instalacion
        FOREIGN KEY (idInstalacion) REFERENCES instalacion(idInstalacion)
        ON UPDATE CASCADE ON DELETE RESTRICT,
    CONSTRAINT chk_evento_fechas CHECK (fechaFin >= fechalinicio)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Campo	Tipo	Nulo	Clave	Descripción
idEvento	BIGINT AI	No	PK	Identificador del evento
nombre	VARCHAR(180)	No		Título

descripción	TEXT	Sí		Descripción
fechalinicio	DATETIME	No		Inicio
fechaFin	DATETIME	No		Fin (CHECK >= fechalinicio)
estado	ENUM('registrado','enRevision','aprobado','rechazado')	No		Estado del flujo
categoria	ENUM('academico','ludico')	No		Clasificación
idOrganizador	BIGINT	No	FK:usuario	Organizado r principal
idInstalacion	BIGINT	No	FK:instalacion	Instalación por defecto
rutaAvalPDF	VARCHAR(255)	No		Evidencia de aval
fechaRegistro	TIMESTAMP	Sí		Creación
Utilidad: núcleo del dominio; soporta KPIs (tasa de aprobación, tiempos, categorías).				

#### 4.1.3.6 TABLA usuarioevento

Propósito: vincula usuarios con eventos indicando rol dentro del evento.

PK: idUsuarioEvento

FK: idUsuario → usuario, idEvento → evento

Campos

- principal (ENUM('S','N'), default 'N')
- tipoAval (ENUM('director\_programa','director\_docencia'), NULL)
- avalPDF (VARCHAR(255), NULL)

Reglas (implementadas con constraint/trigger)

- Solo roles docente/estudiante pueden organizar.
- Un principal='S' por cada evento.

Importancia: gobernanza (qué perfil impulsa más eventos), seguimiento de avales.

```
CREATE TABLE usuarioEvento (
    idUsuarioEvento BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    idUsuario      BIGINT UNSIGNED      NOT NULL,
    idEvento       BIGINT UNSIGNED      NOT NULL,
    principal     ENUM('S','N')        NOT NULL DEFAULT 'N',
    tipoAval      ENUM('director_programa','director_docencia') NULL,
    avalPDF       VARCHAR(255)         NULL,
    CONSTRAINT fk_ue_usuario FOREIGN KEY (idUsuario) REFERENCES
    usuario(idUsuario)
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT fk_ue_evento FOREIGN KEY (idEvento) REFERENCES
    evento(idEvento)
        ON UPDATE CASCADE ON DELETE CASCADE
    -- Reglas de negocio (pendiente con triggers):
    -- 1) Solo roles docente/estudiante.
    -- 2) Un 'principal'='S' por idEvento.
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Campo	Tipo	Nulo	Clave	Descripción
idUsuarioEvento	BIGINT AI	No	PK	Identificador
idUsuario	BIGINT	No	FK:usuario	Usuario vinculado
idEvento	BIGINT	No	FK:evento	Evento vinculado
principal	ENUM('S','N')	No (default 'N')		Si es organizador principal
tipoAval	ENUM('director_programa','director_docencia')	Sí		Tipo de aval
avalPDF	VARCHAR(255)	Sí		Archivo de aval
Reglas:				
Solo docente/estudiante pueden estar aquí (trigger de validación).				
Un principal='S' por cada idEvento (trigger/validación).				

#### 4.1.3.7 TABLA eventoInstalacion

Propósito: asignación de instalaciones a eventos (N–N).

PK: (depende del script: PK compuesto o id propio).

FK: idEvento: evento, idInstalacion: instalacion

importancia: agenda de ocupación y detección de solapamientos.

CREATE TABLE eventoInstalacion (

    idEvento BIGINT UNSIGNED NOT NULL,

    idInstalacion BIGINT UNSIGNED NOT NULL,

    PRIMARY KEY (idEvento, idInstalacion),

```

CONSTRAINT fk_ei_evento FOREIGN KEY (idEvento) REFERENCES
evento(idEvento)
    ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT fk_ei_instalacion FOREIGN KEY (idInstalacion) REFERENCES
instalacion(idInstalacion)
    ON UPDATE CASCADE ON DELETE RESTRICT
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

Campo	Tipo	Nulo	Clave	Descripción
idEventoInstalacion (o PK compuesta)	BIGINT/PK compuesta	No	PK	Identificador o (idEvento,idInstalacion )
idEvento	BIGINT	No	FK:evento	Evento
idInstalacion	BIGINT	No	FK:instalació n	Instalación
Utilidad: asignar múltiples espacios a un evento y detectar solapamientos.				

#### 4.1.3.8 TABLA eventoorganizacion

Propósito: organizaciones participantes por evento (N-N).

PK: idEventoOrganizacion

FK: idEvento: evento, idOrganizacion: organizacion

Campos

- certificadoPDF (opcional), participante, esRepresentanteLegal  
(BOOLEAN)

Importancia: valor de alianzas y trazabilidad legal.

CREATE TABLE eventoOrganizacion (

```

idEventoOrganizacion BIGINT UNSIGNED PRIMARY KEY
AUTO_INCREMENT,
idEvento      BIGINT UNSIGNED      NOT NULL,
idOrganizacion  BIGINT UNSIGNED     NOT NULL,
certificadoPDF VARCHAR(255)          NULL,
participante   VARCHAR(120)          NOT NULL,
esRepresentanteLegal BOOLEAN        NOT NULL DEFAULT TRUE,
CONSTRAINT fk_eo_evento FOREIGN KEY (idEvento) REFERENCES
evento(idEvento)
    ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT fk_eo_organizacion FOREIGN KEY (idOrganizacion)
REFERENCES organizacion(idOrganizacion)
    ON UPDATE CASCADE ON DELETE RESTRICT
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

Campo	Tipo	Nulo	Clave	Descripción
idEventoOrganizacion	BIGINT AI	No	PK	Identificador
idEvento	BIGINT	No	FK:evento	Evento
idOrganizacion	BIGINT	No	FK:organizacion	Organización
certificadoPDF	VARCHAR(255)	Sí		Evidencia
participante	VARCHAR(120)	No		Nombre del participante
esRepresentanteLegal	BOOLEAN	No (default TRUE)		Indicador
Utilidad: trazabilidad con terceros.				

#### 4.1.3.9 TABLA evaluación

Propósito: registro de revisión/decisión académica sobre el evento.

PK: idEvaluacion

FK: idEvento → evento (*y si tu script lo incluye, también idSecretariaAcademica*)

Campos

- estado (ENUM('aprobado','rechazado'))
- comentarios (TEXT), actaPDF (VARCHAR(255)), fechaRevision (TIMESTAMP)

Regla de negocio típica

- AFTER INSERT / AFTER UPDATE: sincroniza evento.estado y genera una notificación.

Importancia: auditoría de decisiones; medir tiempos, motivos de rechazo.

```
CREATE TABLE evaluacion (
    idEvaluacion BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    idEvento      BIGINT UNSIGNED      NOT NULL,
    comentarios   TEXT                 NULL,
    estado        ENUM('aprobado','rechazado') NOT NULL,
    actaPDF       VARCHAR(255)        NULL,
    fechaRevision TIMESTAMP NULL DEFAULT NULL,
    CONSTRAINT fk_eval_evento FOREIGN KEY (idEvento) REFERENCES
    evento(idEvento)
    ON UPDATE CASCADE ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Campo	Tipo	Nulo	Clave	Descripción
idEvaluacion	BIGINT AI	No	PK	Identificador
idEvento	BIGINT	No	FK:evento	Evento evaluado
comentarios	TEXT	Sí		Observaciones
estado	ENUM('aprobado','rechazado')	No		Decisión

actaPDF	VARCHAR(255)	Sí		Acta
fechaRevision	TIMESTAMP	Sí		Fecha de decisión
Regla: AFTER INSERT/UPDATE actualiza evento.estado y genera notificacion.				

#### 4.1.3.10 TABLA notificación

Propósito: evidencia de la comunicación generada por una evaluación.

PK: idNotificacion

FK: idEvaluacion → evaluacion, usuarioReceptor → usuario

Campos

- tipoNotificacion (ENUM('aprobado','rechazado')), fechaEnvio, justificacion, urlPDF

Importancia: cumplimiento de avisos; SLA de comunicación, quién recibe qué y cuándo.

```
CREATE TABLE notificacion (
    idNotificacion BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    idEvaluacion BIGINT UNSIGNED NOT NULL,
    tipoNotificacion ENUM('aprobado','rechazado') NOT NULL,
    fechaEnvio TIMESTAMP NULL DEFAULT NULL,
    justificacion TEXT NULL,
    urlPDF VARCHAR(255) NULL,
    usuarioReceptor BIGINT UNSIGNED NOT NULL,
    CONSTRAINT fk_not_eval FOREIGN KEY (idEvaluacion) REFERENCES evaluacion(idEvaluacion)
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT fk_not_usuario FOREIGN KEY (usuarioReceptor)
        REFERENCES usuario(idUsuario)
        ON UPDATE CASCADE ON DELETE RESTRICT
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Campo	Tipo	Nulo	Clave	Descripción
idNotificacion	BIGINT AI	No	PK	Identificador
idEvaluacion	BIGINT	No	FK:evaluacion	Origen
tipoNotificacion	ENUM('aprobado','rechazado')	No		Tipo
fechaEnvio	TIMESTAMP	Sí		Fecha
justificacion	TEXT	Sí		Texto enviado
urlPDF	VARCHAR(255)	Sí		Enlace a documento
usuarioReceptor	BIGINT	No	FK:usuario	Destinatario
Utilidad: evidencia de comunicación y verificación de SLA internos.				

#### 4.1.3.11 Vistas básicas v\_eventos\_detalle

##### Vistas básicas

Propósito: “todo en una sola vista” para reportes: evento + organizador + instalación (normalmente agregada) + estado/categoría.  
 Columnas: idEvento, nombre, categoria, estado, organizador, instalaciones\_listadas, fechas, etc.  
 Utilidad: alimenta paneles (BI) y reportes sin hacer múltiples JOIN manuales.

```

CREATE OR REPLACE VIEW v_eventos_detalle AS
SELECT e.idEvento, e.nombre, e.fechalnicio, e.fechaFin, e.categoria, e.estado,
       u.idUsuario AS idOrganizador, u.nombre AS organizador, i.nombre AS
       instalacion
FROM evento e
JOIN usuario u  ON u.idUsuario = e.idOrganizador
  
```

```
JOIN instalacion i ON i.idInstalacion = e.idInstalacion;
```

#### 4.1.3.12 Vista básica v\_eventos\_pendientes

Propósito: lista de eventos aún pendientes/en revisión.

Utilidad: tablero operativo para la secretaría; prioriza qué evaluar.

```
CREATE OR REPLACE VIEW v_eventos_pendientes AS
```

```
SELECT * FROM evento WHERE estado IN ('registrado','enRevision');
```

#### 4.1.3.13 Relaciones importantes

usuario (1) —< (N) evento (*creador*)

usuario (N) —< (N) usuarioevento >— (N) evento (*organizadores/aval – con “un principal”*)

evento (N) —< (N) eventoinstalacion >— (N) instalacion

evento (N) —< (N) eventoorganizacion >— (N) organizacion

evento (1) —< (N) evaluacion —< (N) notificacion

usuario (1) —< (N) notificacion (*receptor*)

#### 4.1.4 Inserción de datos.

En este punto, se crean la consulta “2\_insert\_uao.sql”. Esta consulta permite cargar usuarios @uao.edu.co, instalaciones, organizaciones y eventos iniciales. Además, permite probar el backend end-to-end con datos reales, reproducir escenarios y evidenciar resultados en Swagger/Postman y Workbench.

##### Tabla usuarios

```
INSERT INTO usuario (nombre, correo, rol) VALUES  
('Carlos Perez','carlos.perez@uao.edu.co','docente'),  
('Maria Gomez','maria.gomez@uao.edu.co','estudiante'),
```

```

('Juan Lopez','juan.lopez@uao.edu.co','docente'),
('Laura Moreno','laura.moreno@uao.edu.co','estudiante'),
('Andres Torres','andres.torres@uao.edu.co','docente'),
('Paula Ramirez','paula.ramirez@uao.edu.co','estudiante'),
('Diego Martinez','diego.martinez@uao.edu.co','docente'),
('Ana Rodriguez','ana.rodriguez@uao.edu.co','estudiante'),
('Felipe Sosa','felipe.sosa@uao.edu.co','docente'),
('Natalia Rios','natalia.rios@uao.edu.co','estudiante'),
('Sergio Vargas','sergio.vargas@uao.edu.co','docente'),
('Luisa Arias','luisa.arias@uao.edu.co','estudiante'),
('David Castaño','david.castano@uao.edu.co','docente'),
('Juliana Silva','juliana.silva@uao.edu.co','estudiante'),
('Camilo Ortiz','camilo.ortiz@uao.edu.co','docente'),
('Valentina Paz','valentina.paz@uao.edu.co','estudiante'),
('Mateo Sanchez','mateo.sanchez@uao.edu.co','docente'),
('Sofia Herrera','sofia.herrera@uao.edu.co','estudiante'),
('Javier Nino','javier.nino@uao.edu.co','docente'),
('Daniela Calle','daniela.calle@uao.edu.co','secretariaAcademica');

```

TABLA contrasena
-- CONTRASENAS ( <i>hash simbólico</i> )
-- Campo correcto: <i>password_hash + estado</i>
<i>INSERT INTO contrasena (idUsuario, password_hash, estado) VALUES</i>
(1, '\$2b\$12\$hash1', 'activa'), (2, '\$2b\$12\$hash2', 'activa'),
(3, '\$2b\$12\$hash3', 'activa'), (4, '\$2b\$12\$hash4', 'activa'),
(5, '\$2b\$12\$hash5', 'activa'), (6, '\$2b\$12\$hash6', 'activa'),
(7, '\$2b\$12\$hash7', 'activa'), (8, '\$2b\$12\$hash8', 'activa'),
(9, '\$2b\$12\$hash9', 'activa'), (10, '\$2b\$12\$hash10', 'activa'),
(11, '\$2b\$12\$hash11', 'activa'), (12, '\$2b\$12\$hash12', 'activa'),

(13, '\$2b\$12\$hash13', 'activa'),	(14, '\$2b\$12\$hash14', 'activa'),
(15, '\$2b\$12\$hash15', 'activa'),	(16, '\$2b\$12\$hash16', 'activa'),
(17, '\$2b\$12\$hash17', 'activa'),	(18, '\$2b\$12\$hash18', 'activa'),
(19, '\$2b\$12\$hash19', 'activa'),	(20, '\$2b\$12\$hash20', 'activa');

#### TABLA instalacion

```
INSERT INTO instalacion (nombre, tipo, capacidad, ubicacion) VALUES
('Auditorio Principal','auditorio',300,'Bloque A - Piso 1'),
('Laboratorio 3','laboratorio',35,'Bloque C - Piso 2'),
('Salón 201','salon',50,'Bloque B - Piso 2');
```

#### TABLA organización

```
INSERT INTO instalacion (nombre, tipo, capacidad, ubicacion) VALUES
('Auditorio Principal','auditorio',300,'Bloque A - Piso 1'),
('Laboratorio 3','laboratorio',35,'Bloque C - Piso 2'),
('Salon 201','salon',50,'Bloque B - Piso 2');
```

#### TABLA evento

-- *Eventos (3 de muestra)*

```
INSERT INTO evento (nombre, descripcion, idOrganizador, idInstalacion,
fechalinicio, fechaFin, categoria, estado, rutaAvalPDF)
VALUES
('Seminario IA UAO','Charlas sobre ML',1,2,'2025-11-05 08:00:00','2025-11-05
12:00:00','academico','registrado','/avales/aval1.pdf'),
('Festival de Talentos UAO','Musica y artes escenicas',2,1,'2025-11-10
18:00:00','2025-11-10 21:30:00','ludico','registrado','/avales/aval_evento2.pdf'),
('Jornada de Robotica','Exhibicion interfacultades',3,2,'2025-11-15
09:00:00','2025-11-15 17:00:00','academico','enRevision','/avales/aval_rob1.pdf');
```

TABLA usuarioEvento

-- USUARIOEVENTO (*organizador principal y apoyos*)

-- =====

-- Organizadores principales (*uno por evento*)

*INSERT INTO usuarioEvento (idUsuario, idEvento, principal) VALUES*

*(1,1,'S'),(2,2,'S'),(3,3,'S');*

-- Apoyos (*principal = 'N'*)

*INSERT INTO usuarioEvento (idUsuario, idEvento, principal) VALUES*

*(4,1,'N'),(6,1,'N'),(5,3,'N');*

TABLA eventoInstalacion

*INSERT INTO eventoInstalacion (idEvento, idInstalacion) VALUES*

*(1,1),(1,3);*

TABLA eventoOrganizacion

-- EVENTO-ORGANIZACION (*ajustado a columnas reales*)

-- =====

*INSERT INTO eventoOrganizacion (idEvento, idOrganizacion, certificadoPDF, participante, esRepresentanteLegal) VALUES*

*(1,1,'cert/aliado1.pdf','Laura Ruiz',0),*

*(2,2,'cert/patro2.pdf','Miguel Hoyos',1);*

#### 4.1.5 CONSULTAS DE VERIFICACIÓN Y CONTROL

Se creó un grupo de consultas en el archivo “3\_consultas\_control.sql”. Estas consultas incluyen chequeos de calidad, tales como conteos por estado, detección de huérfanos, un “principal” por evento, duplicados y solapamientos de instalaciones. En cierta manera, es el “semáforo” de salud del dato; pues si aquí aparece una alerta, se corrige antes de continuar con pruebas funcionales.

Ahora bien, cuando se habla de “huérfanos”, nos referimos a filas “hijas” que apuntan (por FK) a un registro “padre” que no existe. Por ejemplo,

- Una notificación que apunta a una evaluación inexistente.
- Una evaluación que apunta a un evento que ya fue borrado.
- Un usuarioEvento que referencia un usuario o evento que no están.

Esto es importante porque, los registros huérfanos pueden causar lo siguiente:

- Rompen la coherencia del dato (lo que ves no es confiable).
- Pueden causar errores en el backend (joins que fallan) y en análisis (indicadores inflados o vacíos).
- indican problemas en el flujo de borrado o en la carga de datos.

Por otra parte, los solapamientos se presentan cuando dos (o más) eventos reservados en la misma instalación con rangos de tiempo que se cruzan.

Criterio clásico de solapamiento entre A y B:

- $A.\text{inicio} < B.\text{fin}$  AND  $B.\text{inicio} < A.\text{fin}$

Este tipo de consulta son importantes porque un solapamiento puede:

- Genera conflictos logísticos (dos actividades en el mismo salón/auditorio).
- Impacta la experiencia de asistentes y la operación del evento
- Afecta indicadores de ocupación real vs. planificada.

Al detectar esto, se puede:

- Reprogramar el evento en otro horario o mudar de instalación.
- Detectar patrones de saturación (qué salas chocan más y cuándo), lo cual sirve como insumo para planeación de infraestructura, distribución de eventos y políticas de reserva.

```
● 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 Robótica    enRevisión   Juan Lopez     Laboratorio 3
2      Festival de Talentos UAO    registrado   María Gómez    Auditorio Principal
1      Seminario IA UAO    registrado   Carlos Pérez   Laboratorio 3
estado total
registrado    2
enRevisión   1
```

#### 4.1.5.1 Últimos eventos con organizador e instalación

```
SELECT e.idEvento, e.nombre, e.estado, u.nombre AS organizador, i.nombre AS instalacion
FROM evento e
JOIN usuario u ON u.idUsuario = e.idOrganizador
JOIN instalacion i ON i.idInstalacion = e.idInstalacion
ORDER BY e.idEvento DESC
LIMIT 10;
```

#### 4.1.5.2 Verificar que solo Docente/Estudiante estén en usuarioEvento

```
SELECT ue.idEvento, ue.idUsuario, u.rol
FROM usuarioEvento ue
JOIN usuario u ON u.idUsuario = ue.idUsuario
WHERE u.rol NOT IN ('docente','estudiante');
```

#### 4.1.5.3 Validar unicidad de principal por evento

```
SELECT idEvento, SUM(principal='S') AS cant_principales  
FROM usuarioEvento  
GROUP BY idEvento  
HAVING cant_principales > 1;
```

#### 4.1.5.4 Conteo de eventos por estado

```
SELECT estado, COUNT(*) AS total  
FROM evento  
GROUP BY estado  
ORDER BY total DESC;
```

#### 4.1.5.5 Últimas notificaciones

```
-- Incluye idEvento mediante JOIN con evaluacion  
SELECT n.idNotificacion,  
       ev.idEvento,  
       n.tipoNotificacion,  
       n.fechaEnvio,  
       n.justificacion AS mensaje,  
       n.urlPDF,  
       n.usuarioReceptor  
  FROM notificacion n  
  JOIN evaluacion ev ON ev.idEvaluacion = n.idEvaluacion  
 ORDER BY n.idNotificacion DESC  
 LIMIT 20;
```

## 5. PRUEBAS BACKEND EN SWAGGER (FastAPI – SIGEU)

Para verificar el correcto funcionamiento del backend desarrollado en FastAPI, se utilizaron las herramientas integradas en Swagger UI, generadas automáticamente por el framework a partir de los modelos, rutas y esquemas definidos en el código fuente.

Swagger permitió probar directamente cada endpoint de la API REST sin necesidad de utilizar código externo, validando así la comunicación entre el backend y la base de datos MySQL, la estructura de los objetos enviados y recibidos (JSON), y las respuestas esperadas de cada operación del ciclo CRUD (Create, Read, Update, Delete) del módulo de eventos.

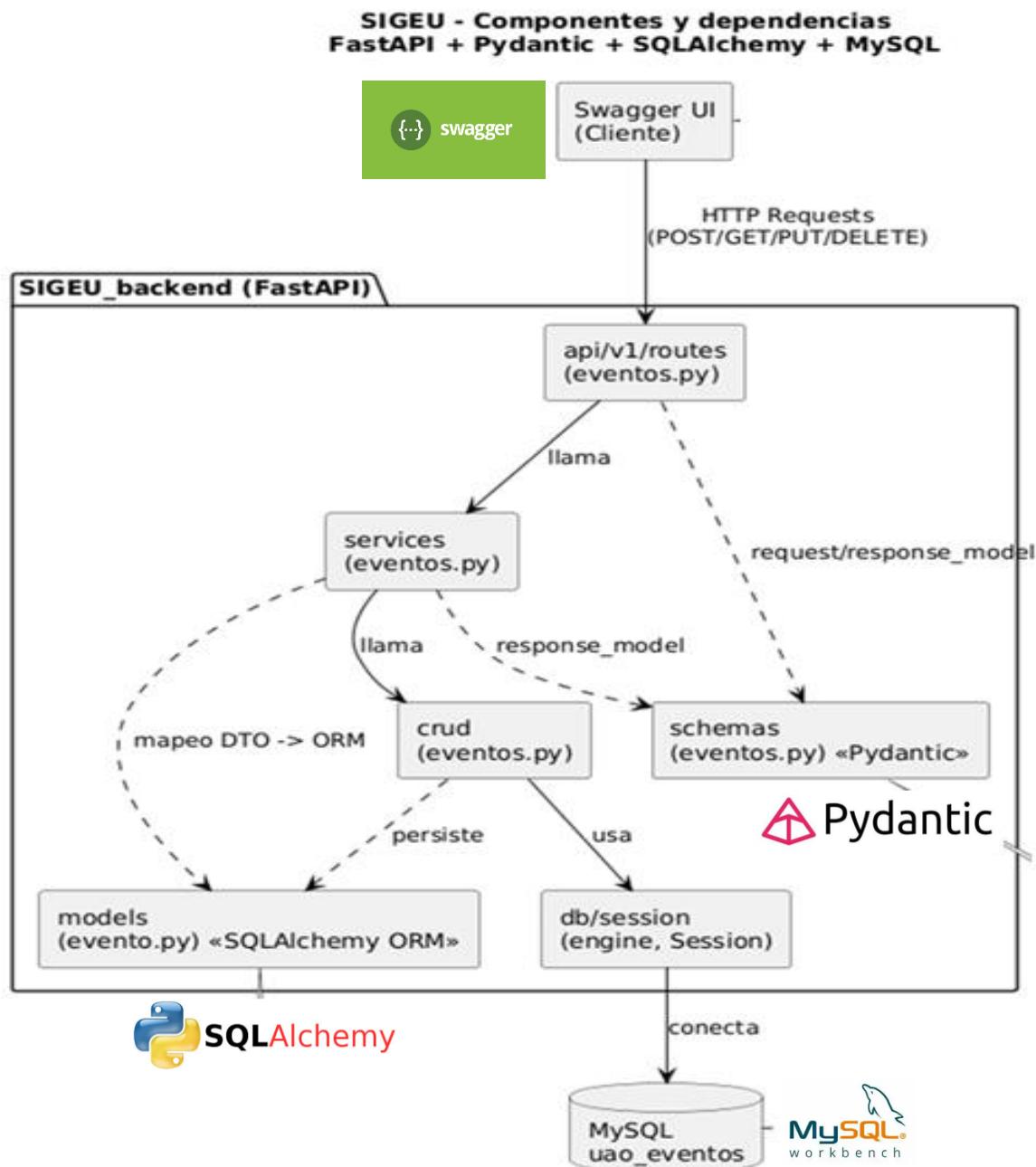
Durante las pruebas se comprobó que:

- El endpoint POST permitió crear registros de eventos con sus respectivos atributos, generando correctamente el idEvento y el estado inicial del proceso.
- El endpoint GET devolvió la lista completa de eventos existentes, así como la consulta puntual por idEvento.
- El endpoint PUT actualizó satisfactoriamente los datos de un evento existente, reflejando los cambios en la base de datos.
- El endpoint DELETE eliminó los registros seleccionados, devolviendo la respuesta 204 No Content según las buenas prácticas de REST.

Asimismo, se verificó la correcta validación de errores y la generación de respuestas con los códigos de estado HTTP esperados (201 Created, 200 OK, 404 Not Found, 422 Unprocessable Entity).

Estas pruebas evidencian que el backend cumple con las especificaciones de diseño, que las rutas y dependencias están correctamente configuradas, y que existe coherencia entre los modelos de datos (schemas, models) y la persistencia en la base de datos uao\_eventos.

Diagrama de vida:



## Estructura de directorios

```
SIGEU_AGL
├── backend
│   ├── .env
│   ├── app
│   ├── api
│   │   └── v1
│   │       └── routes
│   │           └── eventos.py
│   ├── crud
│   │   └── evento.py
│   ├── db
│   │   ├── __init__.py
│   │   └── session.py
│   ├── main.py
│   ├── models
│   │   ├── __init__.py
│   │   ├── base.py
│   │   ├── evaluacion.py
│   │   ├── evento.py
│   │   ├── evento_instalacion.py
│   │   ├── evento_organizacion.py
│   │   ├── instalacion.py
│   │   ├── notificacion.py
│   │   ├── organizacion.py
│   │   ├── usuario.py
│   │   └── usuario_evento.py
│   ├── schemas
│   │   └── evento.py
│   ├── services
│   │   └── evento.py
│   └── requirements.txt
└── docs
    ├── ARCHIVO_PASO_A_PASO_EJECUCION_SIGEU.pdf
    ├── DIAGRAMA_DE_VIDA_VER_1.png
    ├── DOCUMENTO_FINAL_SEGUNDA_ENTREGA.pdf
    ├── MER.png
    ├── MR_SIGEU.mwb
    ├── MR_SIGEU.png
    ├── SIGEU_local_environment.json
    └── SIGEU_postman_collection.json
└── sql
    ├── 1_CREAR_BASE_D.sql
    ├── 2_insert_uao.sql
    ├── 3_consultas_control.sql
    ├── 4_consultas_avanzadas.sql
    ├── 5_objetos_crud_evento.sql
    └── 6. vista_reporte_eventos.sql
└── README_SIGEU
└── TREE.txt
```

## 5.1 CREA UN EVENTO (POST /api/v1/)

```
[notice] to update, run: python -m pip install --upgrade pip
(sigeu_venv) PS D:\SIGEU\SIGEU_AGL\backend> uvicorn --env-file .env app.main:app --reload
INFO:     Will watch for changes in these directories: ['D:\\SIGEU\\SIGEU_AGL\\backend']
INFO:     Loading environment from '.env'
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [20000] using WatchFiles
INFO:     Started server process [10400]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO: 127.0.0.1:52746 - "GET / HTTP/1.1" 307 Temporary Redirect
INFO: 127.0.0.1:52746 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:52746 - "GET /openapi.json HTTP/1.1" 200 OK
```

[http://127.0.0.1:8000/docs#/default/crear\\_evento\\_api\\_v1\\_post](http://127.0.0.1:8000/docs#/default/crear_evento_api_v1_post)

The screenshot shows the Postman interface for a POST request to `/api/v1/Crear Evento`. The request body is defined as follows:

```
{
  "nombre": "Seminario SIGEU",
  "descripcion": "Charlas técnicas de backend",
  "fechaInicio": "2025-11-12T08:00:00",
  "fechaFin": "2025-11-12T12:00:00",
  "estado": "registrado",
  "categoria": "academico",
  "idOrganizador": 1,
  "idInstalacion": 2,
  "rutaAvalPDF": "/avales/aval_seminario.pdf"
}
```

The JSON fields are underlined in red, indicating validation errors or highlighting specific fields.

Payload de ejemplo

```
{  
    "nombre": "Seminario SIGEU",  
    "descripcion": "Charlas técnicas de backend",  
    "fechalinicio": "2025-11-12T08:00:00",  
    "fechaFin": "2025-11-12T12:00:00",  
    "estado": "registrado",  
    "categoria": "academico",  
    "idOrganizador": 1,  
    "idInstalacion": 2,  
    "rutaAvalPDF": "/avales/aval_seminario.pdf"  
}
```

Esperado: 201 + objeto EventoOut con idEvento generado.

Si sale 422: Hay que revisar enums (estado, categoria) y que fechaFin >= fechalinicio.

Si sale 500: Hay que verificar DATABASE\_URL y que exista el idOrganizador/idInstalacion en la BD

#### 5.1.1 Resultados de creación

Durante la prueba del endpoint POST /api/v1/ se obtuvo una respuesta 201 Created, lo que confirma que el flujo de validación (Pydantic), el mapeo hacia el modelo ORM (SQLAlchemy) y la persistencia en MySQL funcionaron correctamente. El cuerpo de la respuesta devuelve el objeto EventoOut con los campos normalizados y el idEvento asignado por la base de datos (en la captura, idEvento = 5).

Previo a esta confirmación, los errores 422 detectados correspondían a discrepancias entre nombres de campos y valores ENUM; tras la incorporación de

alias y validadores en los esquemas, el backend acepta cuerpos camelCase y snake\_case, valida el rango de fechas (fechaFin ≥ fechalinicio) y restringe estado/categoría a valores permitidos.

En conclusión, la creación de eventos está operativa, consistente con el esquema uao\_eventos; lo anterior, se evidencia en las siguientes capturas:

```
(sigeu_venv) PS D:\SIGEU\SIGEU_AGL\backend> uvicorn --env-file .env app.main:app --reload
INFO: Will watch for changes in these directories: ['D:\\SIGEU\\SIGEU_AGL\\backend']
INFO: Loading environment from '.env'
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [13880] using WatchFiles
INFO: Started server process [19812]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:63759 - "GET / HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:63760 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:63761 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:63761 - "GET /openapi.json HTTP/1.1" 200 OK
INFO: 127.0.0.1:63789 - "POST /api/v1/ HTTP/1.1" 201 Created
```

Responses

Curl	Server response
Code	Details
<pre>curl -X 'POST' \   'http://127.0.0.1:8000/api/v1/' \   -H 'accept: application/json' \   -H 'Content-Type: application/json' \   -d '{     "nombre": "Seminario SIGEU",     "descripcion": "Charlas técnicas de backend",     "fechalinicio": "2025-11-12T08:00:00",     "fechaFin": "2025-11-12T12:00:00",     "estado": "registrado",     "categoria": "academico",     "idOrganizador": 1,     "idInstalacion": 2,     "rutaAvalPDF": "/avales/aval_seminario.pdf"   }'</pre>	<p>201</p> <p>Response body</p> <pre>{   "nombre": "Seminario SIGEU",   "descripcion": "Charlas técnicas de backend",   "categoria": "academico",   "idOrganizador": 1,   "idInstalacion": 2,   "fechalinicio": "2025-11-12T08:00:00",   "fechaFin": "2025-11-12T12:00:00",   "rutaAvalPDF": "/avales/aval_seminario.pdf",   "idEvento": 5,   "estado": "registrado" }</pre> <p>Response headers</p> <pre>content-length: 280 content-type: application/json date: Mon, 06 Oct 2025 00:18:43 GMT server: uvicorn</pre>
Request URL	<a href="http://127.0.0.1:8000/api/v1/">http://127.0.0.1:8000/api/v1/</a>

The screenshot shows a REST API documentation interface. On the left, under the 'Responses' tab, there's a table for a successful response (201). It includes a dropdown for 'Media type' set to 'application/json'. Below it, there's an example value and its schema. On the right, under the 'Validation Error' tab, there's a dropdown for 'Media type' set to 'application/json'. It shows a JSON schema for validation errors:

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

Nota: camelCase y snake\_case

Son estilos para escribir nombres de campos/variables:

- camelCase: la primera palabra va en minúscula y cada palabra siguiente empieza con mayúscula, sin guiones bajos.  
Ejemplos: fechalinicio, idOrganizador, rutaAvalPDF.
- snake\_case: todas las palabras en minúscula y separadas por guiones bajos.  
Ejemplos: fecha\_inicio, id\_organizador, ruta\_aval\_pdf.

Y cuando se dice en el texto anterior, que “el backend acepta cuerpos camelCase y snake\_case”, se hace referencia a que la API que tenemos puede recibir cualquiera de los dos estilos en el JSON del request y los entiende igual. Por ejemplo, estos dos cuerpos funcionan y son equivalentes:

```
// camelCase
{
  "idOrganizador": 1,
  "idInstalacion": 2,
```

```
"fechalinicio": "2025-11-12T08:00:00",
"rutaAvalPDF": "/avales/aval.pdf"
}
```

```
// snake_case
{
    "id_organizador": 1,
    "id_instalacion": 2,
    "fecha_inicio": "2025-11-12T08:00:00",
    "ruta_aval_pdf": "/avales/aval.pdf"
}
```

En este proyecto, lo anterior se logró con Pydantic v2 usando `validation_alias` (y `AliasChoices`) en los *schemas*, para que un mismo campo pueda mapearse desde `idOrganizador` o `id_organizador`. Además, al responder usamos `serialization_alias` para normalizar la salida (por ejemplo, devolver siempre `idEvento`).

Todo esto es útil por tres razones importantes:

- Interoperabilidad: clientes JS/TS suelen usar `camelCase`; clientes Python suelen usar `snake_case`.
- Menos fricción: evitas errores 422 por “nombre de campo incorrecto”.
- Consistencia: internamente trabajas con un solo estilo (`snake`) y hacia fuera respondes en el formato que acordaste (`camel`), sin obligar al cliente.

#### 5.1.1.1 Verificación en Mysql Work Brench

- Seleccionar el esquema

- USE uao\_eventos;
- Confirma que existe el registro recién creado
  - SELECT \*
  - FROM evento
  - WHERE idEvento = 5;

The screenshot shows a MySQL Workbench interface. At the top, there is a toolbar with various icons. Below the toolbar, a text area contains the following SQL code:

```

1 • USE uao_eventos;
2 • SELECT *
3   FROM evento
4   WHERE idEvento = 5;
5

```

Below the code, the results are displayed in a grid. The grid has the following columns: idEvento, nombre, descripcion, fechaInicio, fechaFin, estado, categoria, idOrganizador, idInstalacion, rutaAvailPDF. There is one row of data:

	idEvento	nombre	descripcion	fechaInicio	fechaFin	estado	categoria	idOrganizador	idInstalacion	rutaAvailPDF
▶	5	Seminario SIGEU	Charlas técnicas de backend	2025-11-12 08:00:00	2025-11-12 12:00:00	registered	academico	1	2	/avales/aval_seminario.pdf
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Verificamos integridad con FKs (organizador e instalación)

-- Organizador

```

SELECT e.idEvento, e.idOrganizador, u.nombre AS organizador
FROM evento e
JOIN usuario u ON u.idUsuario = e.idOrganizador
WHERE e.idEvento = 5;

```

-- Instalación

```

SELECT e.idEvento, e.idInstalacion, i.nombre AS instalacion
FROM evento e
JOIN instalacion i ON i.idInstalacion = e.idInstalacion
WHERE e.idEvento = 5;

```

The screenshot shows a MySQL Workbench interface. At the top is a toolbar with various icons. Below it is a code editor window containing the following SQL query:

```

4     JOIN usuario u ON u.idUsuario = e.idOrganizador
5     WHERE e.idEvento = 5;
6
7     -- Instalación
8 •  SELECT e.idEvento, e.idInstalacion, i.nombre AS instalacion
9     FROM evento e
10    JOIN instalacion i ON i.idInstalacion = e.idInstalacion
11   WHERE e.idEvento = 5;
12

```

Below the code editor is a results grid titled "Result Grid". It has columns for "idEvento", "idInstalacion", and "instalacion". A single row is shown with values 5, 2, and "Laboratorio 3". There are buttons for "Filter Rows", "Export", and "Wrap Cell Content" below the grid.

- Chequeo rápido de consistencia de fechas

```

SELECT idEvento, fechaInicio, fechaFin,
       (fechaFin >= fechaInicio) AS rango_valido
  FROM evento
 WHERE idEvento = 5;

```

The screenshot shows a MySQL Workbench interface. At the top is a toolbar with various icons. Below it is a code editor window containing the following SQL query:

```

1 •  SELECT idEvento, fechaInicio, fechaFin,
2       (fechaFin >= fechaInicio) AS rango_valido
3     FROM evento
4   WHERE idEvento = 5;
5

```

Below the code editor is a results grid titled "Result Grid". It has columns for "idEvento", "fechaInicio", "fechaFin", and "rango\_valido". A single row is shown with values 5, 2025-11-12 08:00:00, 2025-11-12 12:00:00, and 1. There are buttons for "Filter Rows", "Export", and "Wrap Cell Content" below the grid.

- Vista de control

```
SELECT *
FROM vi_eventos_base
WHERE idEvento = 5;
```

### 5.1.2 Resultados de Listar Eventos.

Al ejecutar el GET, se listan los eventos ordenados por idEvento de forma descendente.

Swagger: GET /api/v1/

Esperado: 200 con el arreglo que incluye el evento recién creado (orden desc por idEvento).

```
INFO:    127.0.0.1:63789 - "POST /api/v1/ HTTP/1.1" 201 Created
INFO:    127.0.0.1:62527 - "GET /api/v1/ HTTP/1.1" 200 OK
INFO:    127.0.0.1:62531 - "GET /api/v1/ HTTP/1.1" 200 OK
INFO:    127.0.0.1:62531 - "GET /api/v1/ HTTP/1.1" 200 OK
```

**GET** /api/v1/ Listar Eventos

Lista los eventos ordenados por **idEvento** descendente.

#### Parameters

No parameters

**Execute**

#### Responses

##### Curl

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

##### Request URL

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

200

##### Response body

```
[  
  {  
    "nombre": "Seminario SIGEU",  
    "descripcion": "Charlas técnicas de backend",  
    "categoria": "academico",  
    "idOrganizador": 1,  
    "idInstalacion": 2,  
    "fechaInicio": "2025-11-12T08:00:00",  
    "fechaFin": "2025-11-12T12:00:00",  
    "rutaAvalPDF": "/avales/aval_seminario.pdf",  
    "idEvento": 5,  
    "estado": "registrado"  
  },  
  {  
    "nombre": "Seminario SIGEU",  
    "descripcion": "Charlas técnicas de backend",  
    "categoria": "academico",  
    "idOrganizador": 1,  
    "idInstalacion": 2,  
    "fechaInicio": "2025-11-12T08:00:00",  
    "fechaFin": "2025-11-12T12:00:00",  
    "rutaAvalPDF": "/avales/aval_seminario.pdf",  
    "idEvento": 4,  
    "estado": "registrado"  
  },  
]
```

200

##### Response body

```
{  
  {  
    "nombre": "Jornada de Robotica",  
    "descripcion": "Exhibicion interfacultades",  
    "categoria": "academico",  
    "idOrganizador": 3,  
    "idInstalacion": 2,  
    "fechaInicio": "2025-11-15T09:00:00",  
    "fechaFin": "2025-11-15T17:00:00",  
    "rutaAvalPDF": "/avales/aval_rob1.pdf",  
    "idEvento": 3,  
    "estado": "enRevision"  
  },  
  {  
    "nombre": "Festival de Talentos UAO",  
    "descripcion": "Musica y artes escenicas",  
    "categoria": "ludico",  
    "idOrganizador": 2,  
    "idInstalacion": 1,  
    "fechaInicio": "2025-11-10T18:00:00",  
    "fechaFin": "2025-11-10T21:30:00",  
    "rutaAvalPDF": "/avales/aval_evento2.pdf",  
    "idEvento": 2,  
    "estado": "registrado"  
  },  
]
```

200

Response body

```
[{"idEvento": 3, "estado": "enRevision"}, {"idEvento": 2, "nombre": "Festival de Talentos UAO", "descripcion": "Musica y artes escenicas", "categoria": "ludico", "idOrganizador": 2, "idInstalacion": 1, "fechaInicio": "2025-11-10T18:00:00", "fechaFin": "2025-11-10T21:30:00", "rutaAvalPDF": "/avales/aval_evento2.pdf", "idEvento": 2, "estado": "registrado"}, {"idEvento": 1, "nombre": "Seminario IA UAO", "descripcion": "Charlas sobre ML", "categoria": "academico", "idOrganizador": 1, "idInstalacion": 2, "fechaInicio": "2025-11-05T08:00:00", "fechaFin": "2025-11-05T12:00:00", "rutaAvalPDF": "/avales/aval1.pdf", "idEvento": 1, "estado": "registrado"}]
```

Response headers

```
content-length: 1383  
content-type: application/json  
date: Mon, 06 Oct 2025 01:00:08 GMT  
server: uvicorn
```

Responses

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

200 Successful Response

Media type

application/json

Controls Accept header.

[Example Value](#) | [Schema](#)

```
[{"idEvento": 3, "nombre": "string", "descripcion": "string", "categoria": "academico", "idOrganizador": 0, "idInstalacion": 0, "fechaInicio": "2025-10-06T01:00:08.408Z", "fechaFin": "2025-10-06T01:00:08.408Z", "rutaAvalPDF": "string", "idEvento": 0, "estado": "registrado"}]
```

SQL de verificación:

```
USE uao_eventos;  
SELECT idEvento, nombre, estado, categoria, fechalinicio, fechaFin  
FROM evento  
ORDER BY idEvento DESC  
LIMIT 10;
```

The screenshot shows the MySQL Workbench interface. At the top, there is a toolbar with various icons. Below the toolbar, a code editor window displays the following SQL query:

```

1 • USE uao_eventos;
2 • SELECT idEvento, nombre, estado, categoria, fechaInicio, fechaFin
3   FROM evento
4 ORDER BY idEvento DESC
5 LIMIT 10;
6

```

Below the code editor is a result grid. The grid has the following columns: idEvento, nombre, estado, categoria, fechaInicio, and fechaFin. The data is as follows:

	idEvento	nombre	estado	categoria	fechaInicio	fechaFin
▶	5	Seminario SIGEU	registrado	academico	2025-11-12 08:00:00	2025-11-12 12:00:00
	4	Seminario SIGEU	registrado	academico	2025-11-12 08:00:00	2025-11-12 12:00:00
	3	Jornada de Robotica	enRevision	academico	2025-11-15 09:00:00	2025-11-15 17:00:00
	2	Festival de Talentos UAO	registrado	ludico	2025-11-10 18:00:00	2025-11-10 21:30:00
*	1	Seminario IA UAO	registrado	academico	2025-11-05 08:00:00	2025-11-05 12:00:00
	NULL	NULL	NULL	NULL	NULL	NULL

Al ejecutar el método GET /api/v1/ desde la interfaz de Swagger UI, el servidor respondió con un código 200 OK, indicando que la operación se realizó exitosamente y que la API logró establecer conexión con la base de datos MySQL para recuperar los registros de eventos.

En el cuerpo de la respuesta se visualiza un arreglo JSON con múltiples objetos, donde cada uno corresponde a un evento almacenado en la tabla evento. Cada registro muestra los campos principales: nombre, descripcion, categoria, idOrganizador, idInstalacion, fechalinicio, fechaFin, rutaAvalPDF, idEvento y estado. Ademas, la lista está ordenada de forma descendente por idEvento, tal como se definió en el código de la capa CRUD. Esto permite que el evento más reciente aparezca primero en la respuesta.

Igualmente, los resultados se validaron en MySQL Workbench, ejecutando la consulta descrita previamente y evidenciada en captura. Los resultados de esta consulta, mostraron exactamente los mismos eventos que fueron devueltos por la

API, con coincidencia perfecta entre los campos idEvento, nombre, estado, categoria, fechalinicio y fechaFin. Entre ellos se destacan los eventos “*Seminario SIGEU*”, “*Jornada de Robótica*”, “*Festival de Talentos UAO*” y “*Seminario IA UAO*”, que evidencian que los datos se están persistiendo correctamente en la base de datos y se recuperan de forma estructurada desde el backend.

En síntesis, los resultados mostrados confirman que:

- El método GET /api/v1/ está completamente funcional.
- La integración FastAPI - SQLAlchemy - MySQL funciona sin errores.
- Los registros se devuelven en el formato JSON definido por el esquema EventoOut.
- La información en Swagger coincide con la información almacenada en MySQL, garantizando consistencia entre backend y base de datos.

#### 5.1.3 Resultados de consulta para Obtener evento por id

En este GET, se ingresa el Id de algún evento existente, para poder ver sus detalles. En este caso, consultados el evento con Id\_evento = 3.

**GET /api/v1/{id\_evento}** Obtener Evento

Obtiene un evento por su identificador.

**Parameters**

Name	Description
<b>id_evento</b> * required integer (path)	3

**Execute**

INFO: 127.0.0.1:62531 - "GET /api/v1/ HTTP/1.1" 200 OK  
 ● INFO: 127.0.0.1:62561 - "GET /api/v1/3 HTTP/1.1" 200 OK

**Responses**

Curl		Responses	
Code	Description	Code	Description
curl -X 'GET' \       'http://127.0.0.1:8000/api/v1/3' \       -H 'accept: application/json'		200	Successful Response
Request URL		Media type	application/json
http://127.0.0.1:8000/api/v1/3		Controls Accept header.	
Server response		Example Value   Schema	
Code Details			
200 Response body			
{ "nombre": "Jornada de Robotica", "descripcion": "Exhibicion interfacultades", "categoria": "academico", "idOrganizador": 3, "idInstalacion": 2, "fechaInicio": "2025-11-15T09:00:00", "fechaFin": "2025-11-15T17:00:00", "rutaAvalPDF": "/avales/aval_rob1.pdf", "idEvento": 3, "estado": "enRevision" }			
Response headers			
content-length: 277 content-type: application/json date: Mon, 06 Oct 2025 01:12:50 GMT server: uvicorn			

422 Validation Error

Media type

application/json

Example Value | Schema

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

En esta prueba se ejecutó la solicitud GET /api/v1/3, cuyo propósito es recuperar los datos de un evento específico mediante su identificador único id\_evento. La ejecución arrojó un código de estado 200 OK, lo que confirma que la API REST logró acceder a la base de datos, localizar el registro correspondiente y devolverlo en formato JSON conforme al esquema EventoOut definido en los modelos de salida.

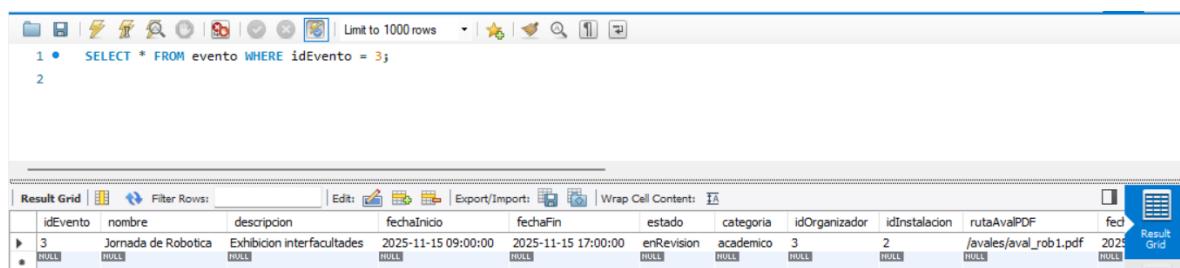
El cuerpo de la respuesta mostró la información del evento con idEvento = 3, identificado como “Jornada de Robótica”, incluyendo sus principales atributos:

- Descripción: “Exhibición inter facultades”.
- Categoría: “Académico”.
- Organizador: idOrganizador = 3.
- Instalación: idInstalacion = 2.
- Fechas: desde el *15 de noviembre de 2025 a las 09:00 hasta las 17:00 del mismo día*.
- Ruta del aval PDF: /avales/aval\_robo1.pdf.
- Estado actual: “enRevisión”.

El encabezado de respuesta confirmó el tipo de contenido (application/json) y la longitud de los datos entregados, lo que demuestra que el backend está serializando correctamente los objetos del modelo SQLAlchemy a JSON.

Ahora bien, la validación cruzada en MySQL Workbench mediante la consulta:

```
SELECT * FROM evento WHERE idEvento = 3;
```



The screenshot shows the MySQL Workbench interface. At the top, there is a toolbar with various icons. Below the toolbar, a query editor window displays the following SQL code:

```
1 •  SELECT * FROM evento WHERE idEvento = 3;
2
```

Below the query editor is a result grid. The grid has columns labeled: idEvento, nombre, descripcion, fechaInicio, fechaFin, estado, categoria, idOrganizador, idInstalacion, rutaAvalPDF, and fed. There is one row of data shown:

idEvento	nombre	descripcion	fechaInicio	fechaFin	estado	categoria	idOrganizador	idInstalacion	rutaAvalPDF	fed
3	Jornada de Robotica	Exhibicion inter facultades	2025-11-15 09:00:00	2025-11-15 17:00:00	enRevision	academico	3	2	/avales/aval_rob1.pdf	2025

Producieron exactamente los mismos valores devueltos por la API, confirmando que el backend accede correctamente al registro solicitado y que existe consistencia total entre el modelo de datos y la respuesta expuesta por la API.

Finalmente, se puede decir que:

- El endpoint GET /api/v1/{id\_evento} funciona correctamente.
- La API valida el parámetro de ruta (path parameter) y devuelve la información exacta del evento solicitado.
- Los datos coinciden con los registros de la base de datos uao\_eventos.
- El formato JSON cumple el contrato del esquema EventoOut.

Esto garantiza que los consumidores de la API (por ejemplo, clientes web o aplicaciones móviles) pueden obtener información precisa de un evento individual de manera confiable, rápida y segura.

#### 5.1.4 Resultados de actualizar evento /api/v1/{id\_evento}

A través de este PUT, se actualiza campos del evento. Por otra parte, este PUT acepta o permite actualización parcial (parches parciales); es decir, solo se aplican los campos enviados.

Evento: 4

Payload:

```
{  
    "nombre": "Seminario SIGEU – edición 2",  
    "descripcion": "Actualizado",  
    "fechainicio": "2025-11-12T09:00:00",  
    "fechaFin": "2025-11-12T13:00:00",  
    "estado": "enRevision",  
    "rutaAvalPDF": "/avales/aval_seminario_v2.pdf"  
}
```

**PUT**

/api/v1/{id\_evento} Actualizar Evento

Actualiza campos del evento. Acepta parches parciales; solo se aplican los campos enviados.

### Parameters

Name	Description
------	-------------

<b>id_evento</b> * required integer (path)	4
--	---

### Request body required

Edit Value | Schema

```
{  
    "nombre": "Seminario SIGEU - edición 2",  
    "descripcion": "Actualizado",  
    "fechaInicio": "2025-11-12T09:00:00",  
    "fechaFin": "2025-11-12T13:00:00",  
    "estado": "enRevision",  
    "rutaAvalPDF": "/avales/aval_seminario_v2.pdf"  
}
```

Resultado esperado: 200 con los cambios

```
● INFO: 127.0.0.1:62561 - "GET /api/v1/3 HTTP/1.1" 200 OK  
INFO: 127.0.0.1:53082 - "PUT /api/v1/4 HTTP/1.1" 200 OK
```

## Responses

### Curl

```
curl -X 'PUT' \
  'http://127.0.0.1:8000/api/v1/4' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "nombre": "Seminario SIGEU - edición 2",
    "descripcion": "Actualizado",
    "fechaInicio": "2025-11-12T09:00:00",
    "fechaFin": "2025-11-12T13:00:00",
    "estado": "enRevision",
    "rutaAvalPDF": "/avales/aval_seminario_v2.pdf"
  }'
```

### Request URL

<http://127.0.0.1:8000/api/v1/4>

### Server response

#### Code Details

200

#### Response body

```
{
  "nombre": "Seminario SIGEU - edición 2",
  "descripcion": "Actualizado",
  "categoria": "academico",
  "idOrganizador": 1,
  "idInstalacion": 2,
  "fechaInicio": "2025-11-12T09:00:00",
  "fechaFin": "2025-11-12T13:00:00",
  "rutaAvalPDF": "/avales/aval_seminario_v2.pdf",
  "idEvento": 4,
  "estado": "enRevision"
}
```

#### Response headers

```
content-length: 281
content-type: application/json
date: Mon, 06 Oct 2025 01:32:30 GMT
server: uvicorn
```

## Responses

### Code Description

200 Successful Response

#### Media type

application/json

Controls Accept header.

#### Example Value | Schema

```
{
  "nombre": "string",
  "descripcion": "string",
  "categoria": "academico",
  "idOrganizador": 0,
  "idInstalacion": 0,
  "fechaInicio": "2025-10-06T01:32:30.869Z",
  "fechaFin": "2025-10-06T01:32:30.869Z",
  "rutaAvalPDF": "string",
  "idEvento": 0,
  "estado": "registrado"
}
```

422

## Validation Error

#### Media type

application/json

#### Example Value | Schema

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

Sql para verificación:

```
SELECT nombre, descripcion, fechalinicio, fechaFin, estado, rutaAvalPDF
FROM evento
WHERE idEvento = 4;
```

The screenshot shows a MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it is a SQL editor window containing the following code:

```

1 •  SELECT nombre, descripcion, fechaInicio, fechaFin, estado, rutaAvalPDF
2   FROM evento
3   WHERE idEvento = 4;
4

```

Below the code is a results grid titled "Result Grid". It has columns for nombre, descripcion, fechaInicio, fechaFin, estado, and rutaAvalPDF. A single row is displayed:

	nombre	descripcion	fechaInicio	fechaFin	estado	rutaAvalPDF
▶	Seminario SIGEU – edición 2	Actualizado	2025-11-12 09:00:00	2025-11-12 13:00:00	enRevisión	/avales/aval_seminario_v2.pdf

En esta verificación del proceso, se probó la operación PUT /api/v1/4, cuyo propósito es actualizar los campos de un evento existente dentro del sistema de gestión SIGEU.

El cuerpo de la solicitud (request body) incluyó valores modificados (payload) en los campos nombre, descripcion, fechalinicio, fechaFin, estado y rutaAvalPDF, para reflejar la actualización del evento “Seminario SIGEU – edición 2”.

El servidor respondió con un código HTTP 200 OK, lo que indica que la solicitud fue procesada correctamente y que el registro correspondiente al idEvento = 4 fue actualizado en la base de datos.

El cuerpo de la respuesta confirmó los nuevos valores persistidos, mostrando la información actualizada:

- nombre: “Seminario SIGEU – edición 2”
- descripción: “Actualizado”
- fechalinicio: 2025-11-12 09:00:00
- fechaFin: 2025-11-12 13:00:00
- estado: “enRevisión”

- rutaAvalPDF: /avales/aval\_seminario\_v2.pdf

Ademas, el encabezado HTTP también mostró un content-type de application/json, validando que el backend está serializando la respuesta conforme al esquema EventoOut.

Adicioanlmente, para verificar la persistencia de los cambios, se ejecutó en MySQL Workbench, con la consulta descrita anteriormente, y el resultado coincidió exactamente con los valores devueltos por la API, confirmando que las modificaciones se aplicaron correctamente en la base de datos uao\_eventos.

Todo lo anterior, evidencia que la capa CRUD implementa adecuadamente la lógica de actualización mediante SQLAlchemy, aplicando parches o cambios parciales sobre los campos del modelo sin sobrescribir información no enviada.

Finalmente, se puede decir que:

- El método PUT /api/v1/{id\_evento} funciona de forma correcta y segura.
- Se valida la actualización selectiva de campos (patch parcial).
- La información modificada se refleja de inmediato tanto en la respuesta JSON como en la base de datos.
- Existe coherencia total entre el backend, el modelo de datos y la vista en Swagger UI.

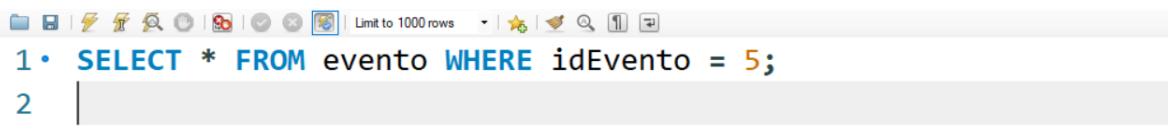
Por tanto, estos resultados garantizan que los administradores del sistema (por ejemplo la secretaria de la universidad o quien haga sus veces de administrador de la BD) pueden actualizar los datos de eventos de manera confiable, manteniendo trazabilidad y consistencia en todo el flujo de la aplicación.

### 5.1.5 Resultados de eliminar evento /api/v1/{id\_evento}

Este DELETE, permite eliminar un evento a partir de su ID (id\_evento). Ahora bien, si la operación es exitosa, devuelve o muestra el código 204 No Content.

En este caso, primero verificamos que existe un evento, por ejemplo el identificado con el número 5.

```
SELECT * FROM evento WHERE idEvento = 5;
```



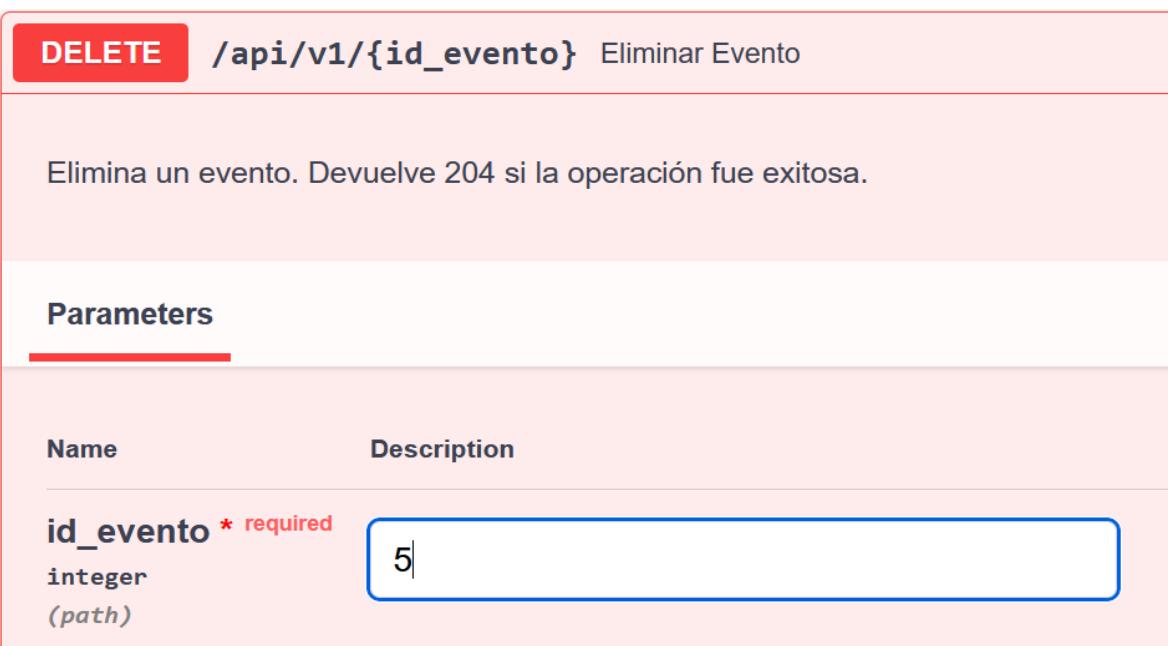
The screenshot shows the MySQL Workbench interface. A query window contains the following code:

```
1 • SELECT * FROM evento WHERE idEvento = 5;
2 |
```

Below the query window is a result grid displaying the results of the executed query. The grid has columns: idEvento, nombre, descripcion, fechaInicio, fechaFin, estado, categoria, idOrganizador, idInstalacion, rutaAvalPDF. One row is shown, corresponding to the event with id 5.

	idEvento	nombre	descripcion	fechaInicio	fechaFin	estado	categoria	idOrganizador	idInstalacion	rutaAvalPDF
▶	5	Seminario SIGEU	Charlas técnicas de backend	2025-11-12 08:00:00	2025-11-12 12:00:00	registrado	academico	1	2	/avalses/aval_seminario.pdf

Luego procedemos a eliminar el registro del evento (5):



The screenshot shows the API documentation for the DELETE /api/v1/{id\_evento} endpoint. The endpoint is described as "Eliminar Evento" (Delete Event) and its purpose is "Elimina un evento. Devuelve 204 si la operación fue exitosa." (Deletes an event. Returns 204 if the operation was successful.).

**Parameters**

Name	Description
<b>id_evento</b> * required integer (path)	5

```
INFO: 127.0.0.1:53082 - "PUT /api/v1/4 HTTP/1.1" 200 OK
INFO: 127.0.0.1:53153 - "DELETE /api/v1/5 HTTP/1.1" 204 No Content
```

## Responses

Curl

```
curl -X 'DELETE' \
'http://127.0.0.1:8000/api/v1/5' \
-H 'accept: */*'
```

Request URL

```
http://127.0.0.1:8000/api/v1/5
```

Server response

Code	Details
204	

Response headers

```
date: Mon, 06 Oct 2025 01:50:40 GMT
server: uvicorn
```

Responses

Code	Description
204	Successful Response
422	Validation Error

Media type

application/json

▼

Example Value | Schema

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

Finalmente, verificamos si todavía existe el evento identificado con el numero 5.

```
SELECT * FROM evento WHERE idEvento = 5;
```

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it, a query editor window contains the following text:

```
1 • SELECT * FROM evento WHERE idEvento = 5;  
2
```

Below the query editor is a results grid titled "Result Grid". It has a header row with columns: idEvento, nombre, descripcion, fechaInicio, fechaFin, estado, categoria, idOrganizador, idInstalacion, rutaAvalPDF, and fechaRegistro. There is one data row, indicated by an asterisk (\*), which contains all values as "NULL".

En esta prueba se ejecutó la operación DELETE /api/v1/5, cuyo objetivo es eliminar de forma definitiva el evento identificado con idEvento = 5 de la base de datos uao\_eventos.

Antes de la eliminación, se validó en MySQL Workbench que dicho registro existía y correspondía al evento “Seminario SIGEU”, con estado “registrado”, categoría “académico” y fechas programadas entre el 12 de noviembre de 2025 a las 08:00 y 12:00. Posteriormente, al ejecutar la solicitud desde Swagger, el servidor respondió con el código HTTP 204 No Content, lo que indica que la operación fue completada correctamente y que no se devuelve contenido en el cuerpo de la respuesta. El encabezado de respuesta mostró el tipo de servidor (unicorn) y la fecha de ejecución, confirmando que FastAPI procesó la solicitud sin errores de validación.

Despues, se realizó la verificación en MySQL Workbench mediante la instrucción:

```
SELECT * FROM evento WHERE idEvento = 5;
```

Lo cual, mostro un resultado sin registros, lo que confirma que la eliminación fue efectiva y el evento ya no se encuentra en la tabla evento. Lo anterior, demuestra que la capa de persistencia (CRUD) está correctamente conectada al motor MySQL, aplicando la operación de borrado sobre el registro correspondiente y actualizando el estado del sistema en tiempo real.

Finalmente, también podemos decir que:

- El endpoint `DELETE /api/v1/{id_evento}` ejecutó exitosamente la eliminación del evento.
- El código 204 No Content valida el cumplimiento del estándar HTTP en respuestas sin cuerpo.
- La API garantizó integridad referencial y sincronización inmediata entre FastAPI, SQLAlchemy y MySQL.
- La verificación en Workbench evidenció la eliminación definitiva del evento con `idEvento = 5`.

En conclusión, después de todas las pruebas, los resultados confirmaron que el backend SIGEU implementa correctamente todo el flujo CRUD (Crear, Listar, Consultar, Actualizar y Eliminar), garantizando consistencia, trazabilidad y control total de los registros dentro del sistema de gestión de eventos.

Durante las pruebas realizadas en Swagger UI, cada una de las operaciones fue validada de forma independiente, confirmando la comunicación efectiva entre la API REST y la base de datos relacional.

La operación POST (/api/v1/) permitió registrar nuevos eventos con validación de datos y asignación automática de identificadores primarios, asegurando la integridad de las inserciones.

El método GET (/api/v1/) evidenció la capacidad de listar todos los eventos en formato JSON, garantizando consistencia con los registros almacenados en la base de datos.

Asimismo, GET (/api/v1/{id\_evento}) recuperó información específica de un evento individual, validando el correcto funcionamiento del parámetro de ruta y la conversión de objetos SQLAlchemy a Pydantic.

Por su parte, el PUT (/api/v1/{id\_evento}) permitió la actualización parcial de los datos, reflejando inmediatamente los cambios en MySQL. Finalmente, el DELETE (/api/v1/{id\_evento}) eliminó de manera definitiva los registros seleccionados, devolviendo el código estándar 204 No Content, en cumplimiento con las buenas prácticas de diseño RESTful.

Las verificaciones cruzadas en MySQL Workbench confirmaron que los resultados obtenidos desde Swagger coincidían plenamente con el estado real de la base de datos, lo que valida la robustez de la capa ORM y la integridad del flujo de datos.

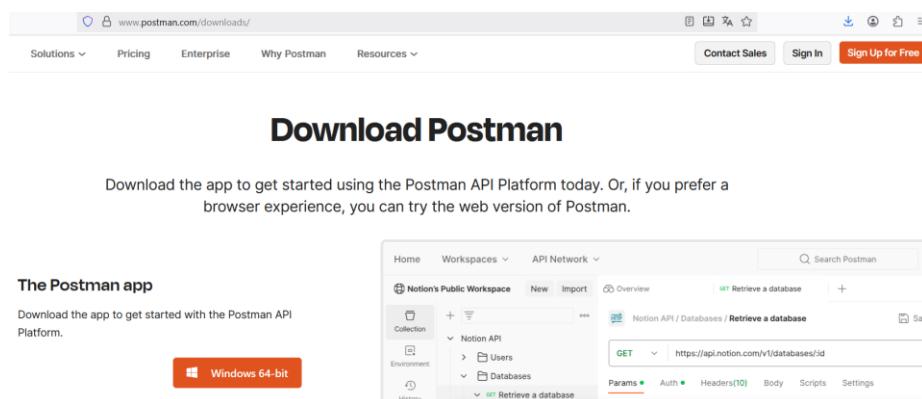
## 6. PRUEBAS REALIZADAS EN POSTMAN.

Para validar el correcto funcionamiento de los endpoints del backend SIGEU, se utilizó la herramienta Postman, la cual permite ejecutar peticiones HTTP y visualizar las respuestas del servidor. Para ello, se creó una colección de pruebas automatizadas que incluye las operaciones CRUD (crear, listar, consultar, actualizar y eliminar eventos) implementadas en la API REST. Ahora bien, cada solicitud fue configurada con su método, URL, cuerpo y pruebas automáticas (tests) que validan los códigos de estado y la coherencia de los datos.

Las ejecuciones se realizaron sobre el entorno local (<http://127.0.0.1:8000>), y los resultados obtenidos confirmaron que el backend cumple con los estándares RESTful y mantiene la integridad de los datos entre FastAPI y MySQL.

### 6.1 EJECUCION DE PRUEBAS EN POSTMAN

- Paso 1: Abrir Postman
  - Descargar e instalar Postman desde:  
<https://www.postman.com/downloads>.



## Working with APIs simplified with Postman

Your work email makes it easy for you to collaborate with your teammates.

[Create Free Account](#)

Already have an account? [Log In](#)

[Continue with Google](#)

[Continue with Github](#)

[Single Sign On \(SSO\)](#)

[Continue without an account](#)



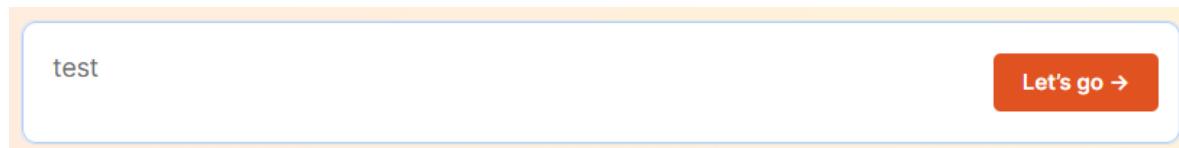
**It's great to have you aboard,  
andres gomez**

Redirecting you to the Desktop App

If you aren't redirected automatically, use [authorization token to sign in](#).

Damos clic en: use authorization token to sign in.

Paso 2: escribe test y luego clic en lest go



Escribe: Building and debugging your own APIs



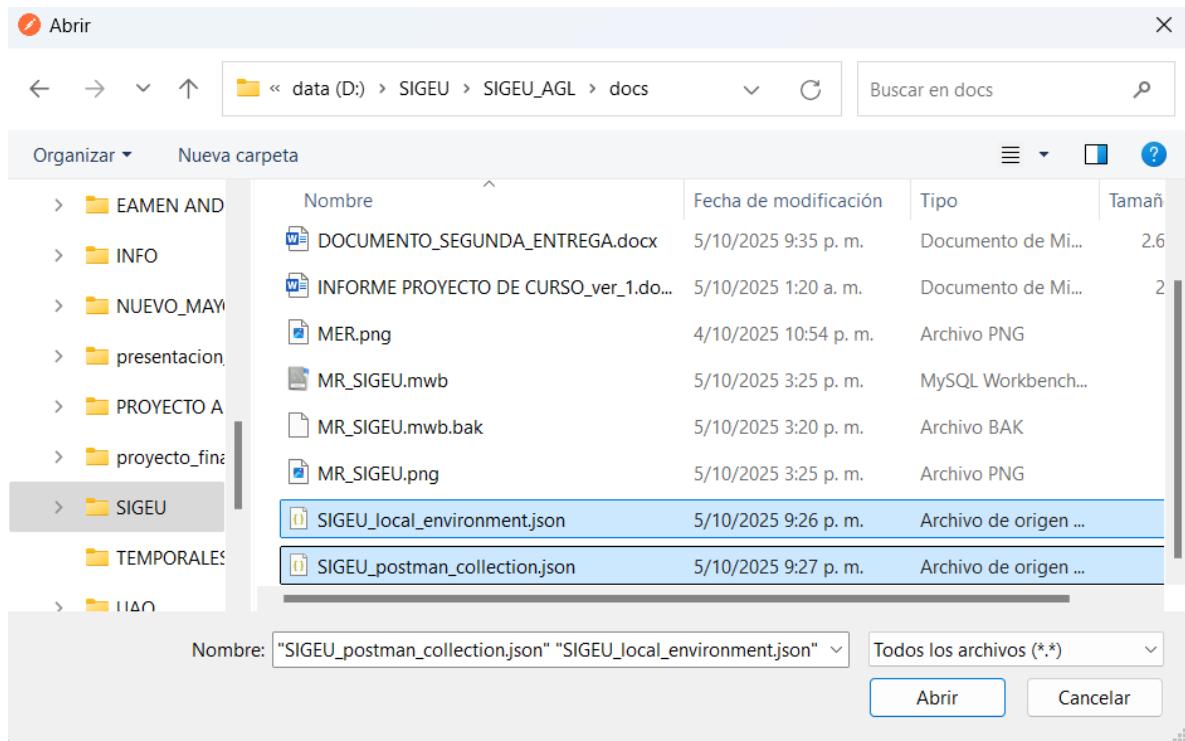
Aparece la siguiente interfaz:

The screenshot shows the Postman application interface. On the left, there's a sidebar with options like 'Collections', 'Environments', 'Flows', and 'History'. The main area has two tabs: 'My Collection' (selected) and 'Onboarding API Agent'. The 'My Collection' tab contains a 'Get data' request with a 'Send' button. The 'Onboarding API Agent' tab has a welcome message: '#Onboarding-Agent: Building and debugging your own APIs', followed by a list of questions and a 'Start building...' button.

clic en el botón “Import” y aparece lo siguiente:

This screenshot shows the 'Import' dialog box in Postman. It features a large text input field at the top with placeholder text 'Paste cURL, Raw text or URL...'. Below it is a dashed rectangular area with a circular icon containing a downward arrow, labeled 'Drop anywhere to import' and 'Or select files or folders'. At the bottom, there are three links: 'Migrate to Postman', 'Other Sources', and 'Learn more about importing data'.

Clic en “files” y carga los dos archivos .json



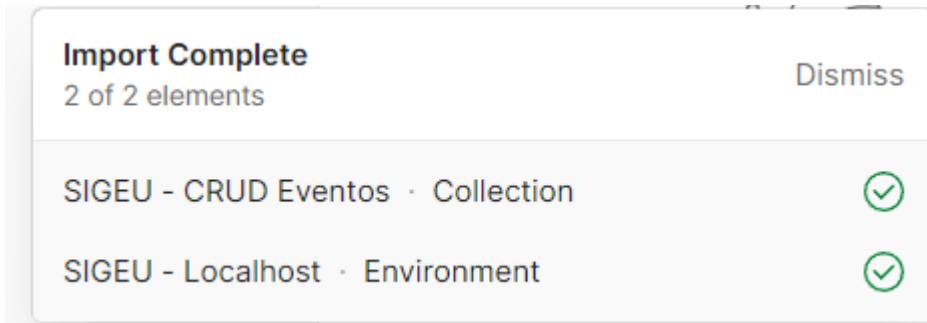
## Import Elements (2)

A screenshot of the Postman "Import Elements" dialog. It shows a table with two items selected:

<input checked="" type="checkbox"/>	Name	Import As
<input checked="" type="checkbox"/>	SIGEU - CRUD Eventos Postman Collection v2.1	Collection
<input checked="" type="checkbox"/>	SIGEU - Localhost Postman Environment	Environment

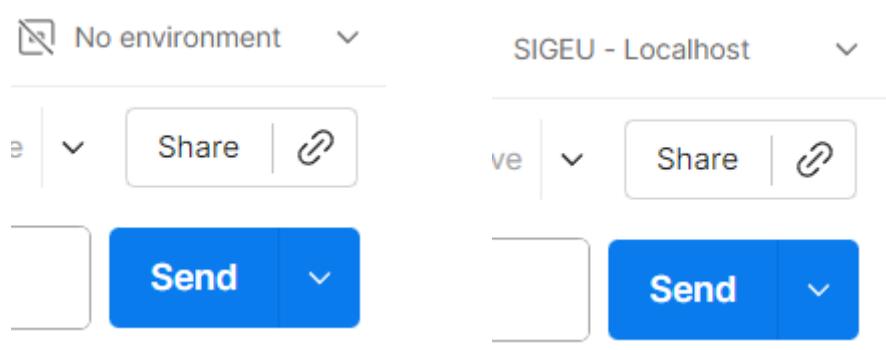
At the bottom are "Back" and "Import" buttons.

Clic en "Import" y sale la siguiente ventana:



Paso 3: Seleccionar el entorno “SIGEU - Localhost”.

En la parte superior derecha el menú desplegable que dice “No environment”, se cambia a SIGEU - Localhost.



Nota: Asegurar de tener el servidor en ejecución, para ello en VS Code o PowerShell ejecutar :

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

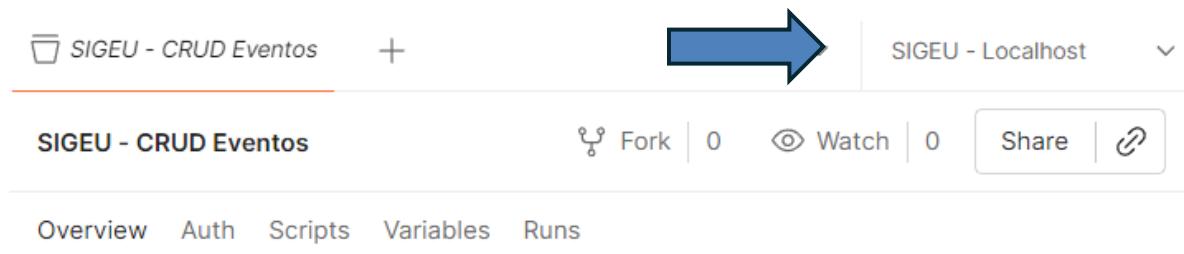
Paso 4. Ejecutar las peticiones en orden (CRUD).

The screenshot shows the Postman application interface. On the left, the sidebar includes sections for Collections, Environments, Flows, and History. The main workspace displays a collection named "My Collection" which contains two requests: "GET Get data" and "POST Post data". Below the collection list, there is a section titled "SIGEU - CRUD Eventos". A modal window titled "Onboarding API Agent" is open on the right, containing a welcome message and a list of questions to guide the user.

En la barra izquierda, ubicamos “SIGEU - CRUD Eventos”, clic en el triángulo para expandirla

The screenshot shows the Postman interface with the "SIGEU - CRUD Eventos" section expanded in the left sidebar. This section lists six API endpoints: "POST 01 - Crear Evento", "GET 02 - Listar Eventos", "GET 03 - Obtener Evento por ID", "PUT 04 - Actualizar Evento", "DEL 05 - Eliminar Evento", and "GET 06 - Obtener Eliminado (404 esper...)".

Verificamos que estamo usando el entorno correcto, para ello, en la parte superior derecha (arriba del botón Send) seleccionamos: SIGEU – Localhost



The screenshot shows the Postman interface. At the top, there's a header with a folder icon labeled "SIGEU - CRUD Eventos" and a "+" button. To the right of the header is a dropdown menu set to "SIGEU - Localhost" with a blue arrow pointing to it. Below the header, there's a navigation bar with tabs: "SIGEU - CRUD Eventos" (which is active and underlined), "Auth", "Scripts", "Variables", and "Runs". On the far right of the navigation bar are buttons for "Fork" (0), "Watch" (0), "Share", and a link icon.

## SIGEU - CRUD Eventos

Make things easier for your teammates with a complete collection description.

De esta manera, Postman sabe que debe apuntar a <http://127.0.0.1:8000>.

### 6.1.1 Ejecución De Creación De Evento

PASO 5. Ejecutar las pruebas una por una.

1. POST: Crear Evento

1. Haz clic en POST Crear Evento

▼ SIGEU - CRUD Eventos
<b>POST</b> 01 - Crear Evento
<b>GET</b> 02 - Listar Eventos
<b>GET</b> 03 - Obtener Evento por ID
<b>PUT</b> 04 - Actualizar Evento
<b>DEL</b> 05 - Eliminar Evento
<b>GET</b> 06 - Obtener Eliminado (404 esper...)

2. Abajo, entra a la pestaña Body, selecciona raw, tipo JSON
3. Pega este ejemplo:

```
{  
  "nombre": "Feria de Emprendimiento UAO",  
  "descripcion": "Exposición de proyectos innovadores de estudiantes",  
  "idOrganizador": 1,  
  "idInstalacion": 2,  
  "fechaInicio": "2025-11-22T09:00:00",  
  "fechaFin": "2025-11-22T13:00:00",  
  "categoria": "academico",  
  "rutaAvalPDF": "/avales/aval_emprendimiento.pdf"  
}
```

HTTP SIGEU - CRUD Eventos / 01 - Crear Evento

Save Share

POST {{baseUrl}} /api/v1/

Send

Params Auth Headers (9) Body Scripts Settings Cookies

raw JSON Schema Beautify

```
1 {
2   "nombre": "Feria de Emprendimiento UAO",
3   "descripcion": "Exposición de proyectos innovadores de estudiantes",
4   "idOrganizador": 1,
5   "idInstalacion": 2,
6   "fechaInicio": "2025-11-22T09:00:00",
7   "fechaFin": "2025-11-22T13:00:00",
8   "categoria": "academico",
9   "rutaAvalPDF": "/avales/aval_emprendimiento.pdf"
10 }
```

Response History

4. Damos clic en SEND

POST {{baseUrl}} /api/v1/

Send

5. Se debe obtener:

- Código 201 Created
- Un JSON con el nuevo evento creado.

### 6.1.1. 1 Evidencias de creacion de evento en POSTMAN

The screenshot shows the Postman interface for a POST request to {{baseUrl}}/api/v1/. The request body is a JSON object representing an event. The response status is 201 Created, indicating success.

```
1 {  
2   "nombre": "Feria de Emprendimiento UAO",  
3   "descripcion": "Exposición de proyectos innovadores de estudiantes",  
4   "idOrganizador": 1,  
5   "idInstalacion": 2,  
6   "fechaInicio": "2025-11-22T09:00:00",  
7   "fechaFin": "2025-11-22T13:00:00",  
8   "categoria": "academico",  
9   "rutaAvalPDF": "/avales/aval_emprendimiento.pdf"  
10 }  
  
Body 201 Created 18 ms 451 B Save Response ...  
{ } JSON Preview Visualize  
1 {  
2   "nombre": "Feria de Emprendimiento UAO",  
3   "descripcion": "Exposición de proyectos innovadores de estudiantes",  
4   "categoria": "academico",  
5   "idOrganizador": 1,  
6   "idInstalacion": 2,  
7   "fechaInicio": "2025-11-22T09:00:00",  
8   "fechaFin": "2025-11-22T13:00:00",  
9   "rutaAvalPDF": "/avales/aval_emprendimiento.pdf",  
10  "idEvento": 6,  
11  "estado": "registrado"  
12 }
```

En esta prueba se verificó la correcta funcionalidad del método POST de la API REST desarrollada en FastAPI para el sistema SIGEU. Para ello, se envió desde Postman una solicitud con los datos de un nuevo evento titulado “Feria de Emprendimiento UAO”, incluyendo su descripción, organizador, instalación, fechas, categoría y ruta del aval.

El servidor respondió con el código 201 Created, confirmando que el evento fue creado exitosamente en la base de datos. Ahora bien, el cuerpo de la respuesta (Response Body) devuelve el objeto completo del evento recién insertado, incluyendo el campo idEvento generado automáticamente (6) y el estado inicial “registrado”.

Este resultado demuestra que el backend procesa correctamente las solicitudes de creación (POST), valida los datos recibidos y guarda la información de los eventos en la tabla evento del sistema.

#### 6.1.2 Evidencias de creación de listar eventos EN POSTMAN

- Clic en GET Listar Eventos
- Presionar Send
- Se vera una lista JSON con todos los eventos existentes

##### ✓ SIGEU - CRUD Eventos

**POST** 01 - Crear Evento

**GET** 02 - Listar Eventos

...

**GET** 03 - Obtener Evento por ID

**PUT** 04 - Actualizar Evento

**DEL** 05 - Eliminar Evento

**GET** 06 - Obtener Eliminado (404 esper...)

HTTP SIGEU - CRUD Eventos / 02 - Listar Eventos

Save Share

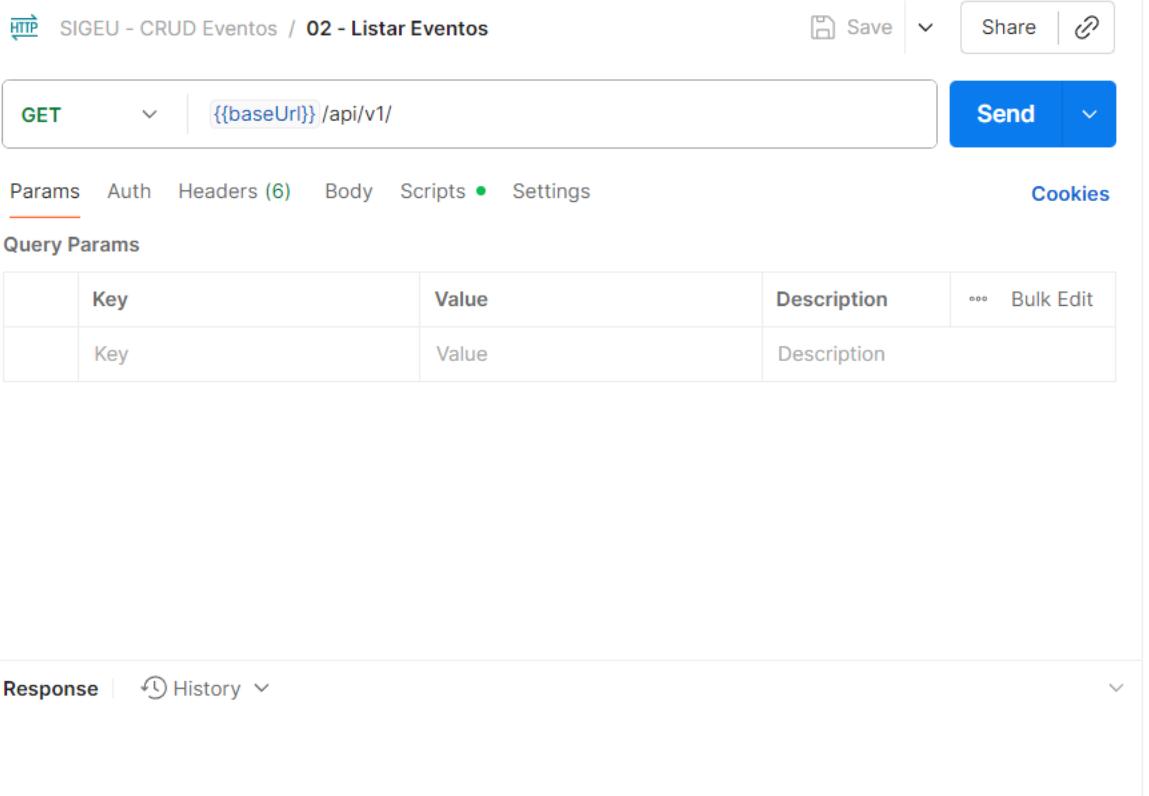
GET {{baseUrl}} /api/v1/ Send

Params Auth Headers (6) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Response History



Llista JSON con todos los eventos existentes:

HTTP SIGEU - CRUD Eventos / 02 - Listar Eventos

Save Share

GET {{baseUrl}} /api/v1/ Send

Params Auth Headers (6) Body Scripts Settings Cookies

Query Params

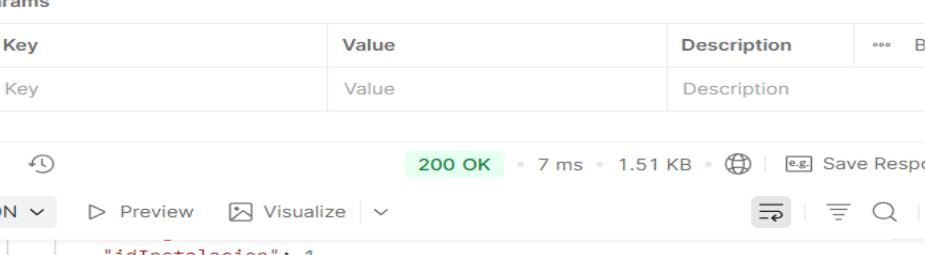
	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body 200 OK 7 ms 1.51 KB Save Response

[{} JSON ▾] Preview Visualize

```
1 [ 
2 { 
3   "nombre": "Feria de Emprendimiento UAO",
4   "descripcion": "Exposición de proyectos innovadores de estudiantes",
5   "categoria": "academico",
6   "idOrganizador": 1,
7   "idInstalacion": 2,
8   "fechaInicio": "2025-11-22T09:00:00",
9   "fechaFin": "2025-11-22T13:00:00",
10  "rutaAvalPDF": "/avales/aval_emprendimiento.pdf",
11  "idEvento": 6,
12  "estado": "registrado"
13 },
14 { 
15   "nombre": "Seminario SIGEU - edición 2",
16   "descripcion": "Actualizado",
17   "categoria": "academico",
18   "idOrganizador": 1,
19   "idInstalacion": 2,
20   "fechaInicio": "2025-11-12T09:00:00",
```

```
Body ▾ ⏱ 200 OK • 7 ms • 1.51 KB • 🌐 | e.g. Save Response ⋮
{ } JSON ▾ Preview Visualize ▾
26 { "nombre": "Jornada de Robotica",
27   "descripcion": "Exhibicion interfacultades",
28   "categoria": "academico",
29   "idOrganizador": 3,
30   "idInstalacion": 2,
31   "fechaInicio": "2025-11-15T09:00:00",
32   "fechaFin": "2025-11-15T17:00:00",
33   "rutaAvalPDF": "/aval/aval_rob1.pdf",
34   "idEvento": 3,
35   "estado": "enRevision"
36 },
37 {
38   "nombre": "Festival de Talentos UAO",
39   "descripcion": "Musica y artes escenicas",
40   "categoria": "ludico",
41   "idOrganizador": 2,
42   "idInstalacion": 1,
43   "fechaInicio": "2025-11-10T18:00:00",
44   "fechaFin": "2025-11-10T21:30:00",
45 }
```



GET {{baseUrl}} /api/v1/ Send

Params Auth Headers (6) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body 200 OK • 7 ms • 1.51 KB | Save Response ...

{ } JSON ▾ ▶ Preview Visualize |

```
43     "idInstalacion": 1,
44     "fechaInicio": "2025-11-10T18:00:00",
45     "fechaFin": "2025-11-10T21:30:00",
46     "rutaAvalPDF": "/avales/aval_evento2.pdf",
47     "idEvento": 2,
48     "estado": "registrado"
49   },
50   {
51     "nombre": "Seminario IA UAO",
52     "descripcion": "Charlas sobre ML",
53     "categoria": "academico",
54     "idOrganizador": 1,
55     "idInstalacion": 2,
56     "fechaInicio": "2025-11-05T08:00:00",
57     "fechaFin": "2025-11-05T12:00:00",
58     "rutaAvalPDF": "/avales/aval1.pdf",
59     "idEvento": 1,
60     "estado": "registrado"
61   }
62 ]
```

En esta prueba se evaluó la operación GET del endpoint principal /api/v1/, correspondiente a la funcionalidad de listar todos los eventos registrados en la base de datos del sistema SIGEU. Para lo anterior, la solicitud se realizó desde Postman utilizando el entorno local configurado (SIGEU - Localhost) y el resultado arrojó un código de respuesta 200 OK, indicando que la petición fue procesada de manera exitosa.

En el cuerpo de la respuesta (Response Body) se recibió un arreglo JSON con todos los eventos actualmente almacenados en la base de datos, incluyendo tanto los creados previamente mediante Swagger como los insertados durante las pruebas de Postman.

Cada objeto del arreglo contiene la información completa del evento:

- nombre,
- descripción,
- categoría,
- identificadores del organizador e instalación,
- fechas de inicio y fin,
- ruta del aval en PDF,
- estado actual y
- idEvento asignado automáticamente por el sistema.

Este resultado confirma que el backend recupera correctamente los registros desde la base de datos MySQL, mostrando consistencia entre los datos almacenados y los devueltos por la API REST. Además, la respuesta también evidencia que la consulta está ordenada por idEvento descendente, lo que facilita la visualización de los eventos más recientes en primer lugar.

```

INFO: 127.0.0.1:63761 - "GET /openapi.json HTTP/1.1" 200 OK
INFO: 127.0.0.1:63789 - "POST /api/v1/ HTTP/1.1" 201 Created
INFO: 127.0.0.1:62527 - "GET /api/v1/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:62531 - "GET /api/v1/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:62531 - "GET /api/v1/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:62561 - "GET /api/v1/3 HTTP/1.1" 200 OK
INFO: 127.0.0.1:53082 - "PUT /api/v1/4 HTTP/1.1" 200 OK
INFO: 127.0.0.1:53153 - "DELETE /api/v1/5 HTTP/1.1" 204 No Content
INFO: 127.0.0.1:51521 - "POST /api/v1/ HTTP/1.1" 201 Created
INFO: 127.0.0.1:51572 - "GET /api/v1/ HTTP/1.1" 200 OK

```

### 6.1.3 Evidencias de obtener evento por id EN POSTMAN

- clic en GET Obtener Evento por ID

SIGEU - CRUD Eventos

- POST** 01 - Crear Evento
- GET** 02 - Listar Eventos
- GET** 03 - Obtener Evento por ID
- PUT** 04 - Actualizar Evento
- DEL** 05 - Eliminar Evento
- GET** 06 - Obtener Eliminado (404 esper...)

- En la URL cambiar el final por el ID del evento que acabamos de crear, por ejemplo: <http://127.0.0.1:8000/api/v1/6>.

En este caso, la opción mas sencilla es cambiar el texto “{{createdId}}”, por el numero “6” (evento que creamos de ultimo), de modo que la línea del GEt quede así: {{baseUrl}}/api/v1/6



- Presionar Send
- La respuesta 200 OK mostrando ese evento específico.

```
INFO: 127.0.0.1:51572 - "GET /api/v1/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:61544 - "GET /api/v1/6 HTTP/1.1" 200 OK
```

The screenshot shows the Postman application interface. At the top, there is a search bar with the URL `{}{{baseUrl}}/api/v1/6` and a blue "Send" button. Below the search bar, there are tabs for Params, Auth, Headers (6), Body, Scripts, Settings, and Cookies. The Headers tab is currently selected. Under "Query Params", there is a table with columns: Key, Value, Description, and Bulk Edit. The table is empty. In the main body area, the status is shown as "200 OK" with a response time of 6 ms and a size of 446 B. Below the status, there are buttons for Body (JSON, Preview, Visualize), and various icons for copy, save, and refresh. The JSON response body is displayed below:

```

1 {
2   "nombre": "Feria de Emprendimiento UAO",
3   "descripcion": "Exposición de proyectos innovadores de estudiantes",
4   "categoria": "academico",
5   "idOrganizador": 1,
6   "idInstalacion": 2,
7   "fechaInicio": "2025-11-22T09:00:00",
8   "fechaFin": "2025-11-22T13:00:00",
9   "rutaAvalPDF": "/avales/aval_emprendimiento.pdf",
10  "idEvento": 6,
11  "estado": "registrado"
12 }
```

En esta etapa se validó la operación GET del endpoint parametrizado `/api/v1/{id_evento}`, cuya finalidad es recuperar un evento específico a partir de su identificador. Para la prueba se utilizó el ID 6, que corresponde al evento creado previamente ("Feria de Emprendimiento UAO").

La solicitud se ejecutó desde Postman con la URL: {{baseUrl}}/api/v1/6

y el backend respondió con código 200 OK, lo que confirma que:

1. el evento existe en la base de datos,
2. el parámetro de ruta fue interpretado correctamente, y
3. la capa de acceso a datos pudo consultar y serializar el registro sin errores.

El cuerpo de la respuesta devolvió el objeto JSON completo del evento, con los siguientes campos de negocio:

- nombre: “Feria de Emprendimiento UAO”
- descripcion: “Exposición de proyectos innovadores de estudiantes”
- categoria: “academico”
- idOrganizador: 1
- idInstalacion: 2
- fechalinicio: “2025-11-22T09:00:00”
- fechaFin: “2025-11-22T13:00:00”
- rutaAvalPDF: “/avales/aval\_emprendimiento.pdf”
- idEvento: 6
- estado: “registrado”

La API resuelve correctamente la recuperación puntual de entidades por su identificador. Esto permite a los usuarios/servicios consumidores consultar el detalle de un evento para tareas como verificación de información, prellenado de formularios de edición, validación de disponibilidad de instalaciones o generación de reportes. Además, la consistencia entre el ID solicitado y el objeto retornado

evidencia que el enrutamiento, la validación del parámetro y el mapeo ORM están implementados adecuadamente.

Nota: ORM: *Object–Relational Mapping* (Mapeo Objeto-Relacional). Es una técnica (y también las librerías que la implementan) para traducir objetos de un lenguaje de programación (clases, atributos, métodos) a filas y columnas de una base de datos relacional, y viceversa. Y sirve para trabajar con clases y objetos en lugar de escribir SQL a mano todo el tiempo. Además, el ORM genera las consultas SQL por ti (INSERT, SELECT, UPDATE, DELETE) y convierte los resultados en objetos. Finalmente, aporta validaciones, relaciones y migraciones.

En otras palabras, el ORM es como un intérprete: tú hablas “objetos de Python” y la base “SQL”; el ORM traduce automáticamente en ambas direcciones.

#### 6.1.4 Evidencias de actualizar evento por id en POSTMAN

- Clic en PUT Actualizar Evento

The screenshot shows the Postman interface with the SIGEU - CRUD Eventos collection expanded. Inside the collection, there are six items listed vertically: 01 - Crear Evento (POST), 02 - Listar Eventos (GET), 03 - Obtener Evento por ID (GET), 04 - Actualizar Evento (PUT, highlighted with a gray background), 05 - Eliminar Evento (DEL), and 06 - Obtener Eliminado (404 esper... (GET)).

- Cambia el ID al mismo del evento que creamos antes (por ejemplo /api/v1/6).



- En el body pegamos lo siguiente:

```
{  
    "nombre": "Feria de Emprendimiento UAO - Edición 2",  
    "descripcion": "Evento actualizado con más expositores",  
    "estado": "enRevision",  
    "rutaAvalPDF": "/avales/aval_emprendimiento_v2.pdf"  
}
```

The screenshot shows a REST client interface with a PUT request to {{baseUrl}}/api/v1/6. The 'Body' tab is selected, and the JSON content is displayed in the 'JSON' tab:

```
1 {  
2     "nombre": "Feria de Emprendimiento UAO - Edición 2",  
3     "descripcion": "Evento actualizado con más expositores",  
4     "estado": "enRevision",  
5     "rutaAvalPDF": "/avales/aval_emprendimiento_v2.pdf"  
6 }
```

- Clic en SEND.
- El body del request y el JSON de respuesta deben mostrar 200 OK

```
INFO: 127.0.0.1:61544 - "GET /api/v1/6 HTTP/1.1" 200 OK  
INFO: 127.0.0.1:61624 - "PUT /api/v1/6 HTTP/1.1" 200 OK
```

The screenshot shows a POSTMAN interface with a PUT request to `{{baseUrl}}/api/v1/6`. The request body is a JSON object:

```
1 {  
2   "nombre": "Feria de Emprendimiento UAO - Edición 2",  
3   "descripcion": "Evento actualizado con más expositores",  
4   "estado": "enRevision",  
5   "rutaAvalPDF": "/avales/aval_emprendimiento_v2.pdf"  
6 }
```

The response is `200 OK` with a response body:

```
1 {  
2   "nombre": "Feria de Emprendimiento UAO - Edición 2",  
3   "descripcion": "Evento actualizado con más expositores",  
4   "categoria": "academico",  
5   "idOrganizador": 1,  
6   "idInstalacion": 2,  
7   "fechaInicio": "2025-11-22T09:00:00",  
8   "fechaFin": "2025-11-22T13:00:00",  
9   "rutaAvalPDF": "/avales/aval_emprendimiento_v2.pdf",  
10  "idEvento": 6,  
11  "estado": "enRevision"  
12 }
```

Durante la prueba de la operación PUT sobre la ruta `{{baseUrl}}/api/v1/6`, se envió una solicitud de actualización al backend con el objetivo de modificar algunos campos del evento previamente registrado con el identificador `idEvento = 6`.

El cuerpo de la solicitud (Body) incluyó los siguientes datos en formato JSON:

```
{  
  "nombre": "Feria de Emprendimiento UAO - Edición 2",  
  "descripcion": "Evento actualizado con más expositores",  
  "estado": "enRevision",  
  "rutaAvalPDF": "/avales/aval_emprendimiento_v2.pdf"  
}
```

El servidor respondió con un código de estado 200 OK, indicando que la operación fue procesada correctamente. Además, en la respuesta del cuerpo (Response body), el sistema devolvió la representación actualizada del evento, donde se reflejan los cambios aplicados:

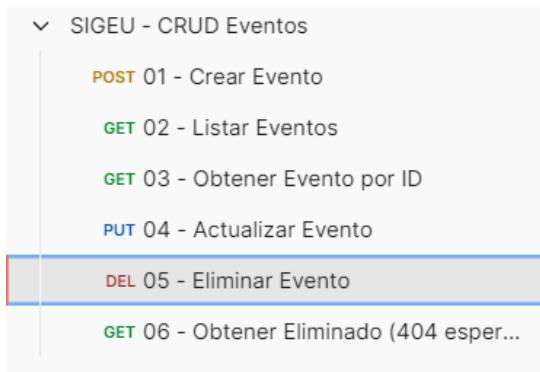
- nombre: se actualizó a “*Feria de Emprendimiento UAO - Edición 2*”
- descripcion: cambió a “*Evento actualizado con más expositores*”
- estado: pasó de “*registrado*” a “*enRevision*”, lo que refleja una modificación del flujo administrativo del evento.
- rutaAvalPDF: fue reemplazada por “*/avales/aval\_emprendimiento\_v2.pdf*”, apuntando al nuevo archivo aval.

El resto de los campos (como categoria, idOrganizador, idInstalacion, fechalinicio y fechaFin) se mantuvieron sin cambios, preservando la integridad del registro original.

En conclusión, la prueba confirmó que el endpoint PUT del sistema actualiza correctamente los campos específicos de un evento existente sin alterar los demás, cumpliendo con la funcionalidad esperada del CRUD.

#### 6.1.5 Evidencias de eliminar evento por id en POSTMAN

- Clic en DELETE Eliminar Evento



- Cambiar el ID a uno existente (por ejemplo /api/v1/6)



- Presionar Send
- El código 204 No Content, que confirma la eliminación.

```
INFO: 127.0.0.1:61624 - "PUT /api/v1/6 HTTP/1.1" 200 OK
INFO: 127.0.0.1:61673 - "DELETE /api/v1/6 HTTP/1.1" 204 No Content
```

HTTP SIGEU - CRUD Eventos / 05 - Eliminar Evento

**DELETE** | {{baseUrl}}/api/v1/6 | **Send**

Params Auth Headers (6) Body Scripts • Settings Cookies

Query Params

	Key	Value	Description	Bulk Edit
	Key	Value	Description	

Body ▾ 204 No Content 8 ms 81 B Save Response

Raw ▾ Preview Visualize

1

Se probó la operación DELETE sobre el endpoint /api/v1/6, con el fin de eliminar definitivamente el evento cuyo identificador era idEvento = 6.

Resultado de la solicitud (Postman):

- Código HTTP: 204 No Content

- Tiempo de respuesta: ~8 ms
- Cuerpo: vacío (como corresponde a un 204)

El código 204 confirma que el backend procesó la eliminación correctamente y que el recurso ya no está disponible. Al no devolverse contenido en el cuerpo, la API se ajusta a las buenas prácticas REST para operaciones de borrado.

Funcionalmente, esto significa que:

- El registro fue eliminado de la tabla evento en MySQL.
- No hubo bloqueos por claves foráneas (si existieran referencias, la base habría rechazado el borrado o el backend habría respondido con error de conflicto).
- El sistema queda listo para que el ID 6 no aparezca en listados ni pueda consultarse.

Se verifica la eliminación de las siguientes formas

1. Reintentar GET por ID: GET /api/v1/6 y devuelve 404 Not Found.

HTTP SIGEU - CRUD Eventos / 05 - Eliminar Evento

Save | Share | [Link](#)

**DELETE** | [/{{baseUrl}} /api/v1/6]({{baseUrl}}/api/v1/6) | **Send** | [Send](#)

Params Auth Headers (6) Body Scripts • Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description	...	

Body [Preview](#) [Debug with AI](#) | [404 Not Found](#) • 5 ms • 165 B • [Save Response](#) • [Copy](#)

```

1  {
2   |   "detail": "Evento no encontrado"
3  }

```

INFO: 127.0.0.1:61673 - "DELETE /api/v1/6 HTTP/1.1" 204 No Content  
INFO: 127.0.0.1:64232 - "DELETE /api/v1/6 HTTP/1.1" 404 Not Found

2. Revisar el listado: GET /api/v1/ ya no debe aparecer el evento con idEvento = 6.

HTTP SIGEU - CRUD Eventos / 02 - Listar Eventos

Save | Share | [Link](#)

**GET** | [/{{baseUrl}} /api/v1/]({{baseUrl}}/api/v1/) | **Send** | [Send](#)

Params Auth Headers (6) Body Scripts • Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description	...	

Body [Preview](#) [Visualize](#) | [200 OK](#) • 5 ms • 1.2 KB • [Save Response](#) • [Copy](#)

```

1  [
2   {
3    |   "nombre": "Seminario SIGEU - edición 2",
4    |   "descripcion": "Actualizado",
5    |   "categoria": "academico",
6    |   "idOrganizador": 1,
7    |   "idInstalacion": 2,
8    |   "fechaInicio": "2025-11-12T09:00:00",
9    |   "fechaFin": "2025-11-12T13:00:00",
10   |   "rutaAvalPDF": "/avales/aval_seminario_v2.pdf",
11   |   "idEvento": 4,
12   |   "estado": "enRevision"
13  },
14 ]

```

### 3. Verificación en MySQL Workbench.

```
USE uao_eventos;  
SELECT * FROM evento WHERE idEvento = 6;
```

The screenshot shows the MySQL Workbench interface. In the top-left pane, there is a code editor with three numbered lines of SQL:

- 1 • USE uao\_eventos;
- 2 • SELECT \* FROM evento WHERE idEvento = 6;
- 3

In the bottom-right pane, there is a "Result Grid" table with the following schema and data:

	idEvento	nombre	descripcion	fechaInicio	fechaFin	estado	categoria	idOrganizador	idInstalacion	rutaAvalPDF	fechaRegistro
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Finalmente, podemos decir que la operación DELETE funciona según lo esperado: elimina el recurso, mantiene la integridad de datos y responde con 204. Esto garantiza que los consumidores del servicio puedan depurar registros obsoletos o erróneos y que la API preserve un comportamiento REST coherente. Si más adelante el proyecto exige historial/auditoría, se puede evaluar convertir este borrado en un soft delete (por ejemplo, un campo activo/estado), pero para el alcance actual el borrado físico cumple correctamente con el requisito de CRUD.

#### 6.1.6 Evidencias de confirmación de obtener evento eliminado en POSTMAN.

Con el fin de garantizar la consistencia del flujo CRUD y el manejo correcto de errores, se implementó una prueba negativa que verifica la respuesta del servicio al consultar un recurso inexistente.

El backend expone el endpoint GET /api/v1/{id\_evento} para obtener un evento por

ID. Tras eliminar un evento (`DELETE /api/v1/{id_evento}`), la misma ruta debe responder 404 Not Found si se intenta consultar nuevamente ese ID.

Ahora bien, en Swagger no aparece “otro” endpoint porque se trata del mismo GET ya documentado. En Postman, sin embargo, incluimos una solicitud dedicada llamada “06 – Obtener Eliminado (404 esperado)”, que reusa la ruta GET `/api/v1/{id_evento}` solo para evidenciar el 404 después de un borrado, facilitando la demostración y la automatización.

A nivel de código, la validación se realiza en `app/services/evento.py`: si el evento no existe al consultarlo, el servicio lanza `HTTPException(404, "Evento no encontrado")`. Esto permite asegurar que el API cumple su contrato y que no se devuelven datos inexistentes tras una operación de eliminación.

En síntesis, Obtener Eliminado (404 esperado), es una prueba negativa para confirmar que, después de borrar un evento, el backend responde 404 Not Found cuando intentas consultarla por su ID. No es un endpoint especial ni adicional, simplemente reutiliza el mismo endpoint de consulta por ID (GET `/api/v1/{id_evento}`), pero ejecutado después del DELETE. En este caso, Postman muestra “06 – Obtener Eliminado (404 esperado)” porque la colección incluye una solicitud prefabricada que usa el mismo GET `/api/v1/{id_evento}` y que sirve solo para documentar y automatizar la verificación del 404. Es decir, en Postman es un caso de prueba, no un endpoint nuevo.

El código del GET `/api/v1/{id_evento}`, está en la siguiente ruta:

- Ruta (endpoint): `app/api/v1/routes/eventos.py`
  - Allí se define la función que atiende GET `/api/v1/{id_evento}`.
- Servicio (reglas de negocio): `app/services/evento.py`
  - Aquí se consulta el evento y, si no existe, se hace:

```
from fastapi import HTTPException

async def obtener_por_id(session, id_evento: int):
    ev = await crud.obtener_por_id(session, id_evento)
    if not ev:
        raise HTTPException(status_code=404, detail="Evento no encontrado")
    return ev
```

- Capa de acceso a datos (CRUD): app/crud/evento.py
  - Implementa la lectura por ID (p.ej. await session.get(Evento, id\_evento)).



Hacemos un GET con el evento que eliminamos: 6

HTTP SIGEU - CRUD Eventos / 06 - Obtener Eliminado (404 esperado)

Save Share

GET {{baseUrl}} /api/v1/6 Send

Params Auth Headers (6) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description	...	

Response History

GET {{baseUrl}} /api/v1/6 Send

Params Auth Headers (6) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description	...	

Body 404 Not Found 5 ms 165 B Save Response

{ } JSON Preview Debug with AI

```
1 {
2   "detail": "Evento no encontrado"
3 }
```

INFO: 127.0.0.1:64239 - "GET /api/v1/ HTTP/1.1" 200 OK  
INFO: 127.0.0.1:54657 - "GET /api/v1/6 HTTP/1.1" 404 Not Found

### 6.1.7 Enlace con acceso público a las colecciones creadas

Para validar el correcto funcionamiento del backend del proyecto SIGEU (Sistema de Gestión de Eventos UAO), se implementó un conjunto de pruebas en la herramienta Postman, la cual permite enviar solicitudes HTTP a una API y verificar las respuestas del servidor de forma controlada y documentada.

En este caso, se construyó una colección denominada “SIGEU – CRUD Eventos”, que agrupa todas las operaciones implementadas en el backend bajo el modelo CRUD (Create, Read, Update, Delete). Estas pruebas se ejecutaron directamente sobre la API REST desarrollada en FastAPI, conectada a la base de datos MySQL, con el objetivo de comprobar la funcionalidad y la consistencia de los endpoints del sistema.

Cada una de las rutas fue probada de manera individual:

- POST /api/v1/ → para crear un nuevo evento.
- GET /api/v1/ → para listar todos los eventos registrados.
- GET /api/v1/{id\_evento} → para obtener un evento específico por su identificador.
- PUT /api/v1/{id\_evento} → para actualizar datos existentes del evento.
- DELETE /api/v1/{id\_evento} → para eliminar un evento determinado.
- Y una prueba adicional GET /api/v1/{id\_evento} (404 esperado) para verificar la eliminación efectiva.

Durante el proceso, se observó que cada petición obtuvo los códigos de respuesta esperados, confirmando el comportamiento correcto del servidor y la integridad de los datos en la base de datos. Las respuestas devueltas (201, 200, 204 y 404) evidencian que el backend procesa adecuadamente las operaciones de inserción, consulta, modificación y eliminación.

Asimismo, se configuró un entorno local de Postman llamado “SIGEU – Localhost”, que define las variables base para las pruebas, como la URL del servidor (<http://127.0.0.1:8000>). Esto permite reutilizar la colección en cualquier equipo sin modificar manualmente las direcciones de conexión.

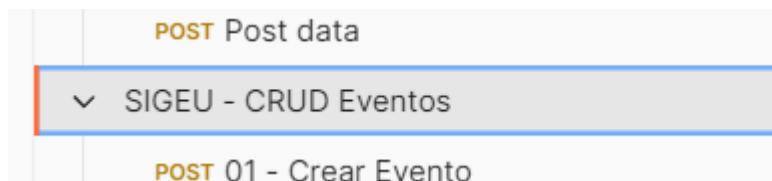
Finalmente, la colección fue publicada mediante un enlace de acceso público, de forma que el profesor o compañeros puedan importar y ejecutar las mismas pruebas directamente en Postman, sin necesidad de realizar configuraciones adicionales.

Este enlace representa la evidencia completa del ciclo de pruebas sobre la API REST del sistema, garantizando transparencia y reproducibilidad en la validación del proyecto.

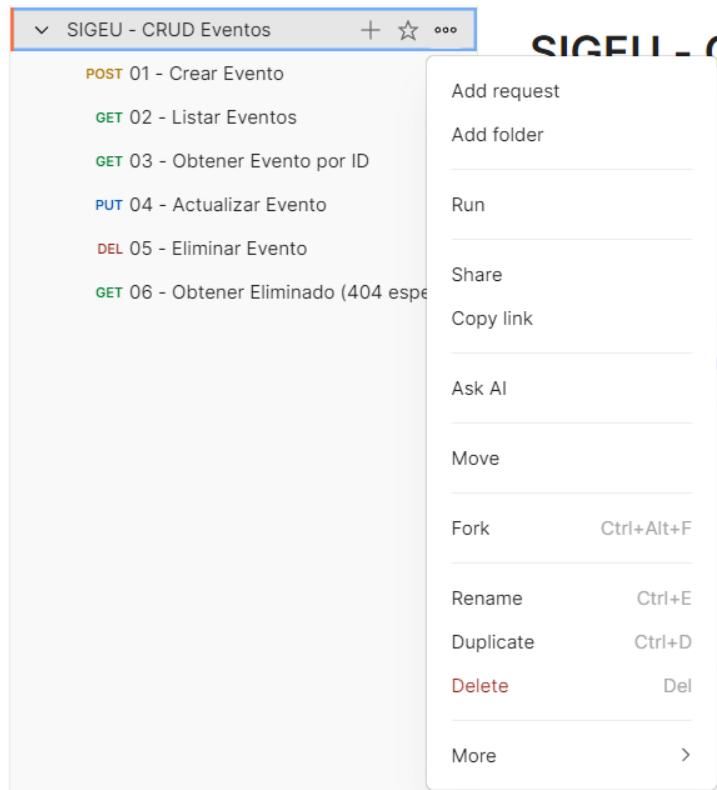
#### 6.1.8.1 Generar el enlace público en Postman

##### Paso 1. Abrir la colección en Postman

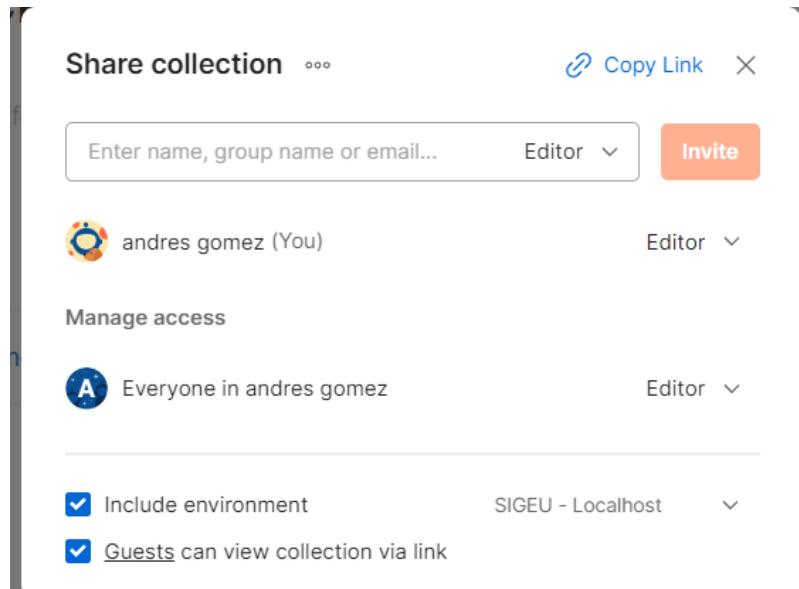
- En el panel izquierdo, ubica SIGEU – CRUD Eventos.



- Damos clic derecho sobre el nombre y seleccionamos “Share” o “Compartir colección”.



- Sale la siguiente ventana, donde podemos hacer pública la colección, permitiendo que cualquier persona la visualice y ejecute desde su navegador o app; así que marca la casilla que dice:
  - “Guests can view collection via link”. (esto permite que cualquier persona pueda acceder sin tener cuenta).
  - “Include environment: SIGEU - Localhost”. De esta manera, se incluye la configuración del entorno local).



- Luego, haz clic en “Copy Link” (arriba a la derecha). Esto, copiará el enlace público, algo como:

```
https://andresgomezdoctorado-8529723.postman.co/workspace/andres-gomez's-Workspace~25a1ed28-155c-4f0f-8243-ddf433ed08ec/collection/49033182-3c0f62d8-68e1-48f7-97e5-8c46d3e3771f?action=share&creator=49033182&active-environment=49033182-d9c4618b-a1d6-43fe-892b-1dbf047b8f7a
```

#### 6.1.8.1.1 Enlace público de la colección Postman

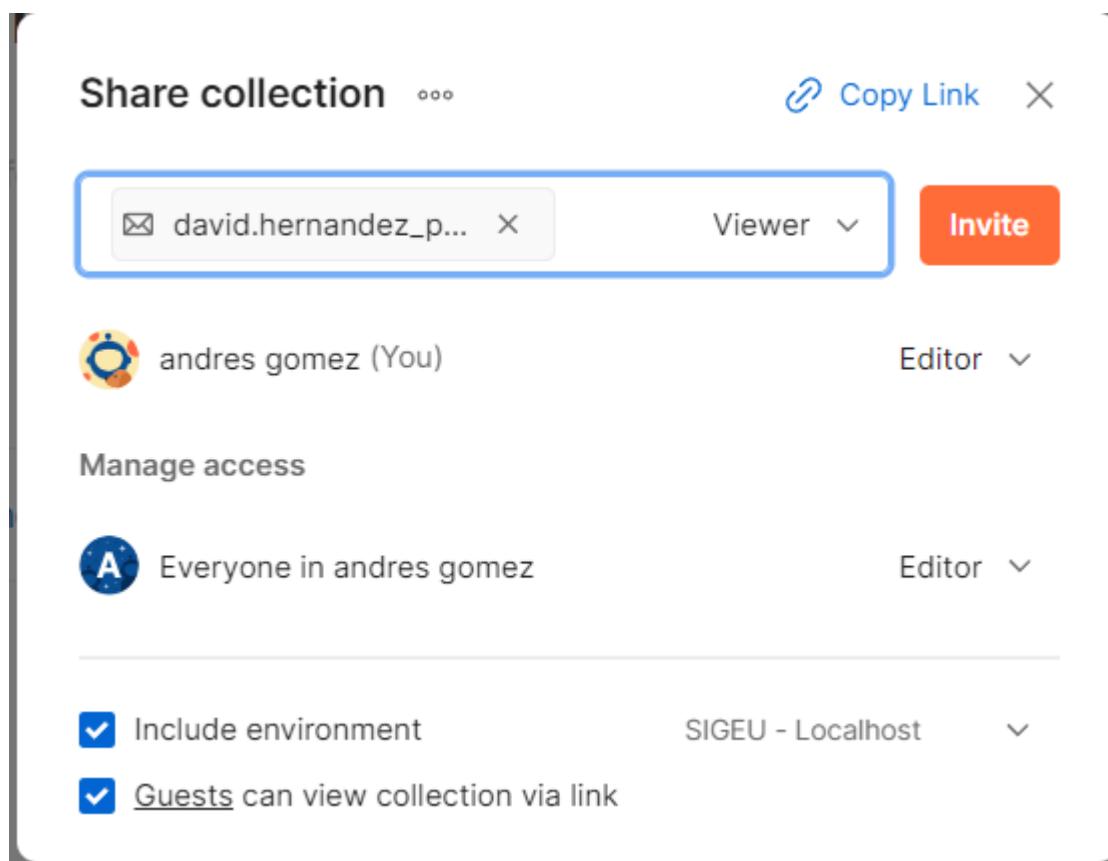
Para evidenciar las pruebas del backend desarrollado con FastAPI, se generó una colección de Postman llamada “SIGEU – CRUD Eventos”, que contiene todas las operaciones principales de la API REST: crear, listar, obtener por ID, actualizar y eliminar eventos.

Desde la opción *Share Collection* en Postman se activó el modo de acceso público, lo que permite que cualquier usuario con el enlace pueda importar y ejecutar las

pruebas en su propio entorno, verificando la correcta funcionalidad del sistema. Esta evidencia demuestra la comunicación efectiva entre el backend FastAPI y la base de datos MySQL, mediante el uso de endpoints probados en el entorno local SIGEU - Localhost.

**Enlace publico:**

```
https://andresgomezdoctorado-8529723.postman.co/workspace/andres-gomez's-Workspace~25a1ed28-155c-4f0f-8243-ddf433ed08ec/collection/49033182-3c0f62d8-68e1-48f7-97e5-8c46d3e3771f?action=share&creator=49033182&active-environment=49033182-d9c4618b-a1d6-43fe-892b-1dbf047b8f7a
```

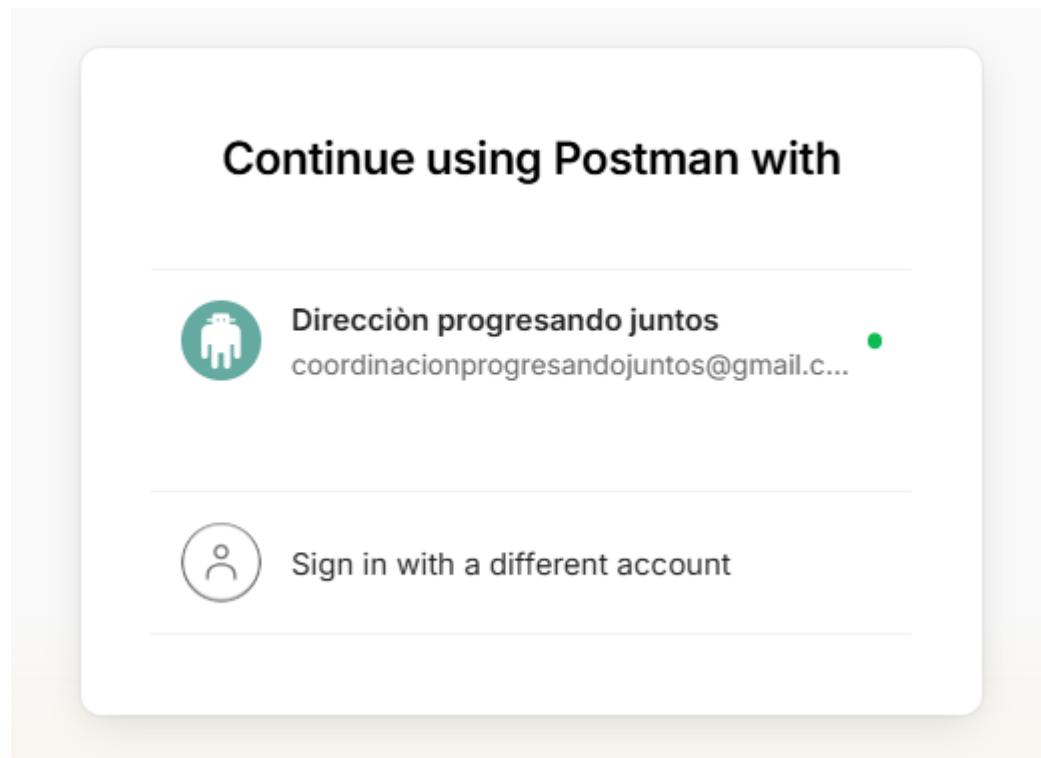


También se puede compartir, escribiendo el correo electrónico de la persona a la cual queremos permitir acceder a la colección.



Luego damos clic en “Invite”, para enviar la invitacion.

Para acceder desde algún usuario a quien comparto enlace, se debe hacer lo siguiente:



Dar clic en el usuario (correo)



## Team andresgomezdoctorado-8529723 is excited to work with you

Seamlessly collaborate with multiple teams

- ✓ Contribute to multiple teams without leaving your existing team.
  - ✓ Invite collaborators, even if they are from different teams.
  - ✓ Instantly switch teams without signing into a separate account.
- I understand that teams have complete ownership to manage all data, including personal workspaces and if I leave the team, I will not have access to them.

[Join Team](#)

[Cancel](#)

Clic en “Join Team”, para acceder a la interfaz de pruebas de Postman.

The screenshot shows the Postman application interface. At the top, there are navigation links for 'Home', 'Workspaces', and 'API Network'. A search bar on the right contains the placeholder 'Search Postman' with keyboard shortcuts 'Ctrl K'. Below the header, the left sidebar shows 'andres gomez's Workspace' with a 'Collections' section containing a list of API collections: 'SIGEU - CRUD Eventos' (selected), '01 - Crear Evento', '02 - Listar Eventos', '03 - Obtener Evento por ID', '04 - Actualizar Evento', '05 - Eliminar Evento', and '06 - Obtener Eliminado (404 esper...)'.

The main content area is titled 'SIGEU - CRUD Eventos'. It includes tabs for 'Overview', 'Auth', 'Scripts', 'Tests', and 'Variables'. A 'Request to Edit' button is located in the top right corner. The 'Overview' tab displays the message 'This collection doesn't have a description.' Below this, there is a link 'View complete documentation →'.

Al compartir la colección en Postman, se generó un enlace público que permite a otros usuarios unirse al equipo andresgomezdoctorado-8529723 y acceder a todas las pruebas del backend SIGEU.

Al ingresar, Postman solicita confirmar la colaboración mediante la opción Join Team, lo que permite ejecutar los endpoints directamente desde la nube, facilitando la revisión y validación del API por parte de los docentes o compañeros sin necesidad de configuración manual.

## 7. CONSULTAS AVANZADAS CON SQL

Estas consultas, generan indicadores operativos y estratégicos tales como: tasas de aprobación/rechazo, tiempos de revisión por secretaría, uso de instalaciones, horas pico, anticipación de solicitudes, top de organizadores/organizaciones, etc. Su importancia, radica en que estas métricas guían decisiones de planificación (calendario, asignación de espacios y personal), mejora de procesos (SLA) y gestión de aliados externos.

Para este ejercicio se construyo el archivo “4\_consultas\_avanzadas.sql”, del cual se tomo captura de su ejecución desde Visual Studio Code, como se muestra a continuación:

```
● 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
academicos    1      1      100.00
ludicos    0      1      0.00
pendientes_0a2d    pendientes_3a7d    pendientes_8a14d    pendientes_mas14d
3      0      0      0
categoria    duracion_promedio_min
academicos    360.0000
ludicos    210.0000
anio    mes    categoria    total_eventos
2025    11    academicos    2
2025    11    ludicos    1
organizacion    categoria    total_eventos
Comunidad AI Cali    academicos    1
Fundacion Talentos    ludicos    1
PS D:\SIGEU\SIGEU_AGL> []
```

Este archivo, contiene las consultas descritas a continuación, y que siguen una estructura muy parecida entre sí. Todas están pensadas para analizar los datos de la base UAO\_Eventos, y por eso se construyen de una forma lógica y ordenada para poder obtener resultados claros sobre los eventos, los organizadores, las instalaciones y las evaluaciones.

En la mayoría de las consultas se usan CTEs que sirven como pasos intermedios para preparar los datos antes de hacer el cálculo final. Por ejemplo, primero se seleccionan los eventos con cierta información y después se aplican filtros, comparaciones o agrupaciones sobre ese resultado. Esto hace que las consultas sean más fáciles de leer y mantener.

También se usan varios tipos de JOINs para conectar tablas que están relacionadas entre sí. En el archivo se ven uniones entre tablas como evento, usuario, evaluacion, eventoInstalacion, instalacion, usuarioEvento, notificacion y organizacion. Estas relaciones permiten combinar datos de diferentes partes del sistema, por ejemplo, unir un evento con su organizador, con las evaluaciones que recibió o con la instalación donde se realiza. Gracias a eso, las consultas pueden mostrar información completa sin necesidad de trabajar cada tabla por separado.

Se emplean muchas funciones que ayudan a calcular tiempos y diferencias entre fechas, como `TIMESTAMPDIFF`, `DATEDIFF`, `DATE_SUB`, `NOW()` y `CURRENT_DATE()`. Estas funciones son muy usadas para medir duraciones, detectar retrasos o calcular la antigüedad de ciertos registros.

Otro aspecto importante es el uso de agregaciones, que aparecen en casi todas las consultas. Se usan funciones como `COUNT`, `SUM`, `AVG` y `ROUND` para resumir la información en valores más representativos, como promedios, porcentajes o totales.

## 7.1 CONSULTA 1 – PRODUCTIVIDAD POR ORGANIZADOR.

Definición: Esta consulta muestra cuántos eventos ha creado cada organizador dentro del sistema. Básicamente cuenta los eventos registrados por cada usuario.

Propósito: Sirve para medir la productividad de los organizadores y saber quiénes están más activos. También ayuda a identificar si hay una persona con demasiados eventos asignados o si hay otros con poca participación.

Explicación: La consulta usa un JOIN entre las tablas evento y usuario, conectadas por el campo idOrganizador. Luego se aplica un COUNT(\*) para contar los eventos que tiene cada usuario. El GROUP BY agrupa por el identificador y nombre del organizador, y el ORDER BY ordena los resultados de mayor a menor según la cantidad de eventos. Es una consulta que organiza la información de productividad.

```
SELECT u.idUsuario, u.nombre, COUNT(*) AS eventos_creados
FROM evento e
JOIN usuario u ON u.idUsuario = e.idOrganizador
GROUP BY u.idUsuario, u.nombre
ORDER BY eventos_creados DESC;
```

```

1 •   SELECT u.idUsuario, u.nombre, COUNT(*) AS eventos_creados
2     FROM evento e
3       JOIN usuario u ON u.idUsuario = e.idOrganizador
4   GROUP BY u.idUsuario, u.nombre
5 ORDER BY eventos_creados DESC;
6

```

Result Grid			
	idUsuario	nombre	eventos_creados
▶	1	Carlos Perez	2
	3	Juan Lopez	1
	2	Maria Gomez	1

## 7.2 CONSULTA 2: HORAS RESERVADAS POR INSTALACIÓN

Definición: Esta consulta calcula cuántas horas han sido reservadas en cada instalación durante los últimos 30 días.

Propósito: Sirve para conocer qué espacios se usan más y cuáles tienen menos actividad. Con eso se pueden tomar decisiones sobre mantenimiento, disponibilidad o incluso crear nuevas áreas si algunas están muy saturadas.

Explicación: Primero se crea una CTE llamada ev\_inst que une dos fuentes: las instalaciones principales guardadas directamente en la tabla evento y las adicionales de eventoInstalacion. Luego, esa lista se combina con las tablas instalacion y evento usando JOINs para obtener los nombres y las fechas.

La función TIMESTAMPDIFF(HOUR, fechalinicio, fechaFin) calcula la duración de cada evento en horas, y SUM suma todas esas horas por instalación. Se filtra solo

lo ocurrido en los últimos 30 días con DATE\_SUB(CURRENT\_DATE(), INTERVAL 30 DAY), y finalmente se agrupa por instalación y se ordena de mayor a menor cantidad de horas reservadas.

```
WITH ev_inst AS (
    SELECT idEvento, idInstalacion FROM evento
    UNION
    SELECT idEvento, idInstalacion FROM eventolInstalacion
)
SELECT i.idInstalacion, i.nombre,
    SUM(TIMESTAMPDIFF(HOUR, e.fechaInicio, e.fechaFin)) AS horas_reservadas_30d
FROM ev_inst ei
JOIN instalacion i ON i.idInstalacion = ei.idInstalacion
JOIN evento e ON e.idEvento = ei.idEvento
WHERE e.fechaInicio >= DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
GROUP BY i.idInstalacion, i.nombre
ORDER BY horas_reservadas_30d DESC;
```

	idInstalacion	nombre	horas_reservadas_30d
▶	2	Laboratorio 3	16
	1	Auditorio Principal	7
	3	Salon 201	4

## 7.3 CONSULTA TASA DE APROBACIÓN GLOBAL Y BACKLOG

Definición: Esta consulta muestra cuántos eventos han sido aprobados, cuántos ya fueron evaluados, el porcentaje de aprobación y cuántos siguen pendientes.

Propósito: Sirve para entender cómo va el proceso de revisión de eventos. Con esta información se puede ver si el comité de evaluación está al día o si hay muchos eventos atrasados o sin revisar.

Explicación: La consulta trabaja solo con la tabla evento. Usa SUM junto con expresiones booleanas para contar los registros según su estado. Por ejemplo, SUM(estado='aprobado') cuenta los eventos aprobados, y SUM(estado IN ('aprobado','rechazado')) cuenta todos los que ya se evaluaron.

Después calcula la tasa de aprobación usando una división entre aprobados y evaluados, multiplicando por 100 y redondeando con ROUND(...,2) para mostrar dos decimales. Se usa NULLIF para evitar error si no hay eventos evaluados.

Por último, también suma los que están en estado 'registrado' o 'enRevision' como pendientes. En conjunto, muestra una visión global del flujo de aprobación de eventos.

```
SELECT
    SUM(estado='aprobado') AS aprobados,
    SUM(estado IN ('aprobado','rechazado')) AS evaluados,
    ROUND(100*SUM(estado='aprobado')/
        NULLIF(SUM(estado IN ('aprobado','rechazado')),0),2) AS tasa_aprobacion_pct,
    SUM(estado IN ('registrado','enRevision')) AS pendientes
FROM evento;
```

```

SQL File 14* SQL File 15* SQL File 16* SQL File 17* SQL File 18* SQL File 19* SQL File 20*
[ ] [ ] [ ] [ ] [ ] [ ] [ ] Limit to 1000 rows [ ] [ ] [ ] [ ] [ ] [ ]
1 • SELECT
2     SUM(estado='aprobado') AS aprobados,
3     SUM(estado IN ('aprobado', 'rechazado')) AS evaluados,
4     ROUND(100*SUM(estado='aprobado')/
5           NULLIF(SUM(estado IN ('aprobado', 'rechazado')),0),2) AS tasa_aprobacion_pct,
6     SUM(estado IN ('registrado', 'enRevision')) AS pendientes
7     FROM evento;
8
9

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

	aprobados	evaluados	tasa_aprobacion_pct	pendientes
▶	0	0	NULL	4

#### 7.4 CONSULTA TIEMPO PROMEDIO A DECISIÓN (HORAS) POR ESTADO

Definición: Esta consulta calcula cuántas horas, en promedio, tarda un evento desde que se registra hasta que recibe una decisión final, ya sea aprobado o rechazado.

Propósito: Ayuda a medir la eficiencia del comité evaluador. Permite saber si las revisiones están tomando demasiado tiempo o si el proceso está funcionando dentro del plazo esperado.

Explicación: La consulta une las tablas evaluacion y evento usando un JOIN por el campo idEvento. Luego usa la función TIMESTAMPDIFF(HOUR, e.fechaRegistro, ev.fechaRevision) para medir el número de horas entre el registro del evento y la fecha en que fue revisado.

Después, aplica AVG para obtener el promedio de esas horas según el estado de la evaluación (por ejemplo, aprobado o rechazado). Finalmente, el GROUP BY ev.estado agrupa los resultados por cada tipo de estado.

El resultado muestra un promedio de tiempo por estado que refleja qué tan rápido se están tomando las decisiones en el sistema.

```
SELECT ev.estado,
       AVG(TIMESTAMPDIFF(HOUR, e.fechaRegistro, ev.fechaRevision)) AS
       horas_promedio
  FROM evaluacion ev
 JOIN evento e ON e.idEvento = ev.idEvento
 GROUP BY ev.estado;
```

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it is a query editor window containing the SQL code. The code is numbered from 1 to 6. The result grid below shows two rows of data: 'aprobado' with a 'horas\_promedio' value of 355.0000 and 'rechazado' with a value of 378.0000. The result grid has columns labeled 'estado' and 'horas\_promedio'.

estado	horas_promedio
aprobado	355.0000
rechazado	378.0000

## 7.5 CONSULTA TASA DE APROBACIÓN POR ORGANIZADOR

Definición: Esta consulta muestra el porcentaje de eventos aprobados por cada organizador, junto con la cantidad total de evaluaciones que ha tenido.

Propósito: Sirve para medir la calidad de las propuestas que presenta cada organizador. Con esta información se puede reconocer a quienes logran más aprobaciones y detectar a quienes podrían necesitar más apoyo o guía.

Explicación: La consulta une tres tablas: evento, usuario y evaluacion. El JOIN entre evento y usuario conecta los organizadores con sus eventos, y el otro JOIN con evaluacion trae el resultado de las revisiones.

Usa SUM(ev.estado='aprobado') para contar cuántos eventos fueron aprobados y COUNT(ev.idEvaluacion) para saber cuántos fueron evaluados en total. Luego calcula la tasa de aprobación con una división multiplicada por 100 y redondeada a dos decimales usando ROUND. Se incluye NULLIF para evitar errores si un organizador no tiene evaluaciones. Se agrupa por organizador y se ordena primero por la tasa de aprobación más alta y luego por cantidad de evaluaciones.

```
SELECT u.idUsuario, u.nombre,
       SUM(ev.estado='aprobado') AS aprobados,
       COUNT(ev.idEvaluacion) AS evaluados,
       ROUND(100*SUM(ev.estado='aprobado')/NULLIF(COUNT(ev.idEvaluacion),0),2)
) AS tasa_aprobacion_pct
FROM evento e
JOIN usuario u ON u.idUsuario = e.idOrganizador
JOIN evaluacion ev ON ev.idEvento = e.idEvento
GROUP BY u.idUsuario, u.nombre
ORDER BY tasa_aprobacion_pct DESC, evaluados DESC;
```

```

2     SUM(ev.estado='aprobado') AS aprobados,
3     COUNT(ev.idEvaluacion)   AS evaluados,
4     ROUND(100*SUM(ev.estado='aprobado')/NULLIF(COUNT(ev.idEvaluacion),0),2) AS tasa_aprobacion_pct
5   FROM evento e
6   JOIN usuario u  ON u.idUsuario = e.idOrganizador
7   JOIN evaluacion ev ON ev.idEvento = e.idEvento
8   GROUP BY u.idUsuario, u.nombre
9   ORDER BY tasa_aprobacion_pct DESC, evaluados DESC;
10

```

The screenshot shows a MySQL query editor interface. At the top, there are various toolbar icons. Below the toolbar, the SQL query is displayed. The result grid shows two rows of data:

	idUsuario	nombre	aprobados	evaluados	tasa_aprobacion_pct
▶	3	Juan Lopez	1	1	100.00
	2	Maria Gomez	0	1	0.00

## 7.6 CONSULTA TASA DE APROBACIÓN POR CATEGORÍA

**Definición:** Esta consulta calcula el porcentaje de eventos aprobados según su categoría, como “académico” o “lúdico”.

**Propósito:** Ayuda a comparar el rendimiento entre tipos de eventos y ver cuáles tienen mejores resultados. Con eso se pueden tomar decisiones sobre qué categorías fortalecer o equilibrar dentro del portafolio.

**Explicación:** La consulta une las tablas evaluacion y evento con un JOIN por el campo idEvento. Después, cuenta los eventos aprobados usando `SUM(ev.estado='aprobado')` y el total de evaluados con `COUNT(*)`.

Luego calcula la tasa de aprobación multiplicando por 100 y redondeando con `ROUND(...,2)`, usando `NULIF` para evitar divisiones por cero. Finalmente, agrupa por categoría y ordena los resultados de mayor a menor tasa de aprobación. Así se obtiene una comparación clara de qué tipo de evento logra más aprobaciones en el sistema.

```

SELECT e.categoría,
       SUM(ev.estado='aprobado') AS aprobados,
       COUNT(*)           AS evaluados,
       ROUND(100*SUM(ev.estado='aprobado')/NULLIF(COUNT(*),0),2)      AS
tasa_aprobacion_pct
FROM evaluacion ev
JOIN evento e ON e.idEvento = ev.idEvento
GROUP BY e.categoría
ORDER BY tasa_aprobacion_pct DESC;

```

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it is a query editor window containing the SQL code provided in the previous block. The code is numbered from 1 to 9. The result grid below shows the output of the query:

	categoria	aprobados	evaluados	tasa_aprobacion_pct
▶	academico	1	1	100.00
	ludico	0	1	0.00

## 7.7 CONSULTA BACKLOG POR ANTIGÜEDAD (SLA) PARA REVISIÓN

Definición: Esta consulta muestra cuántos eventos siguen pendientes de revisión, agrupados por la cantidad de días que han pasado desde que fueron registrados.

Propósito: Sirve para detectar si hay acumulación de solicitudes sin atender y en qué rango de antigüedad se concentran. Esto ayuda a identificar cuellos de botella y tomar decisiones para mejorar la velocidad de revisión.

Explicación: La consulta trabaja únicamente con la tabla evento. Usa la función DATEDIFF(CURRENT\_DATE(), fechaRegistro) para calcular los días que han pasado desde que se registró cada evento.

Luego, con SUM y condiciones dentro, clasifica los pendientes en cuatro grupos: de 0 a 2 días, de 3 a 7, de 8 a 14 y más de 14 días. Solo cuenta los eventos con estado 'registrado' o 'enRevision', ya que son los que todavía no tienen decisión.

El resultado final muestra la cantidad de eventos atrasados según su antigüedad, lo que permite visualizar fácilmente el tamaño y la edad del backlog.

```
SELECT
    SUM(DATEDIFF(CURRENT_DATE(), fechaRegistro) BETWEEN 0 AND 2
        AND estado IN ('registrado','enRevision')) AS pendientes_0a2d,
    SUM(DATEDIFF(CURRENT_DATE(), fechaRegistro) BETWEEN 3 AND 7
        AND estado IN ('registrado','enRevision')) AS pendientes_3a7d,
    SUM(DATEDIFF(CURRENT_DATE(), fechaRegistro) BETWEEN 8 AND 14
        AND estado IN ('registrado','enRevision')) AS pendientes_8a14d,
    SUM(DATEDIFF(CURRENT_DATE(), fechaRegistro) > 14
        AND estado IN ('registrado','enRevision')) AS pendientes_mas14d
FROM evento;
```

The screenshot shows a SQL query editor interface. At the top, there are tabs labeled 'SQL File 14\*' through 'SQL File 19\*'. Below the tabs is a toolbar with various icons for file operations, search, and navigation. The main area contains a multi-line text input field with the following SQL code:

```

3      AND estado IN ('registrado','enRevision')) AS pendientes_0a2d,
4      SUM(DATEDIFF(CURRENT_DATE(), fechaRegistro) BETWEEN 3 AND 7
5          AND estado IN ('registrado','enRevision')) AS pendientes_3a7d,
6      SUM(DATEDIFF(CURRENT_DATE(), fechaRegistro) BETWEEN 8 AND 14
7          AND estado IN ('registrado','enRevision')) AS pendientes_8a14d,
8      SUM(DATEDIFF(CURRENT_DATE(), fechaRegistro) > 14
9          AND estado IN ('registrado','enRevision')) AS pendientes_mas14d
10     FROM evento;
11

```

Below the code is a horizontal line separator. Underneath it, there is a 'Result Grid' section with a toolbar containing 'Result Grid', 'Filter Rows:', 'Export:', and 'Wrap Cell Content:' buttons. The result grid displays four columns with the following data:

	pendientes_0a2d	pendientes_3a7d	pendientes_8a14d	pendientes_mas14d
▶	3	0	0	0

## 7.8 CONSULTA SOLAPAMIENTOS POR INSTALACIÓN (CONTEO DE CONFLICTOS)

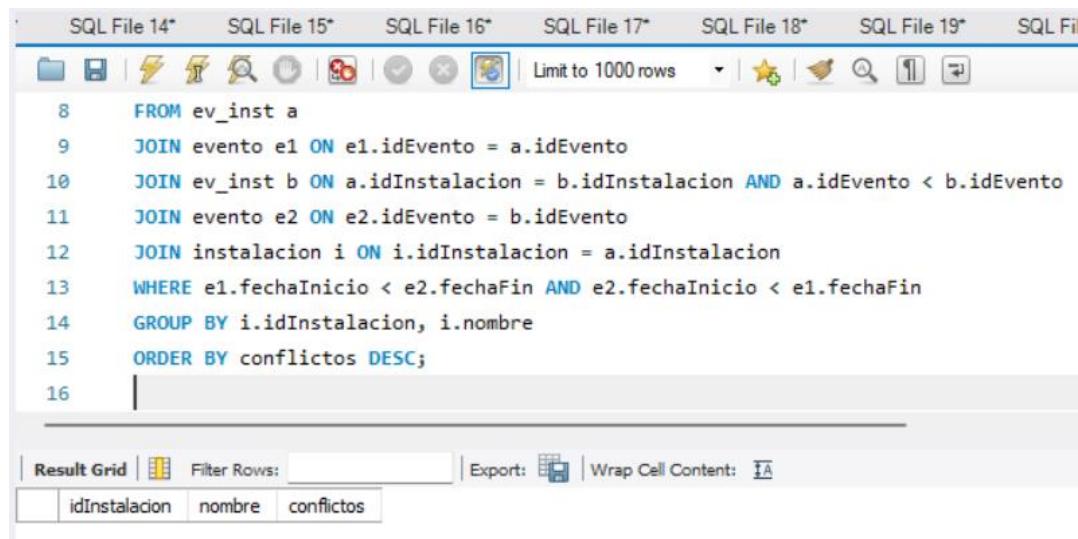
**Definición:** Esta consulta cuenta cuántos eventos se cruzan en el tiempo dentro de la misma instalación, es decir, los casos donde hay dos reservas que se traslanan.

**Propósito:** Sirve para detectar conflictos de agenda y saber qué espacios están teniendo choques de horario. Esto ayuda a prevenir problemas operativos y planificar mejor las reservas.

**Explicación:** Primero se crea una CTE llamada ev\_inst que combina las instalaciones principales de la tabla evento con las adicionales de eventoInstalacion usando un UNION. Luego, se hace un auto-join de esa CTE: se comparan dos eventos diferentes (a y b) que usan la misma instalación, asegurando que no se repitan pares con la condición a.idEvento < b.idEvento. Después, se unen ambas versiones con la tabla evento para obtener las fechas de inicio y fin, y con la tabla instalacion para el nombre del espacio. En el WHERE se colocan las condiciones

que detectan solapamiento de horarios: cuando el inicio de uno ocurre antes de que el otro termine, y viceversa. Se agrupa por instalación y se cuenta cuántos conflictos hay en total, ordenando de mayor a menor cantidad.

```
WITH ev_inst AS (
    SELECT idEvento, idInstalacion FROM evento
    UNION
    SELECT idEvento, idInstalacion FROM eventoInstalacion
)
SELECT i.idInstalacion, i.nombre,
    COUNT(*) AS conflictos
FROM ev_inst a
JOIN evento e1 ON e1.idEvento = a.idEvento
JOIN ev_inst b ON a.idInstalacion = b.idInstalacion AND a.idEvento < b.idEvento
JOIN evento e2 ON e2.idEvento = b.idEvento
JOIN instalacion i ON i.idInstalacion = a.idInstalacion
WHERE e1.fechaInicio < e2.fechaFin AND e2.fechaInicio < e1.fechaFin
GROUP BY i.idInstalacion, i.nombre
ORDER BY conflictos DESC;
```



The screenshot shows a SQL editor interface with the following details:

- Toolbar: Includes icons for file operations, search, and export.
- Text Area: Displays the SQL query with numbered lines 8 through 16.
- Result Grid: A table with three columns: idInstalacion, nombre, and conflictos. The table is currently empty.
- Bottom Bar: Includes buttons for Result Grid, Filter Rows, Export, and Wrap Cell Content.

## 7.9 CONSULTA EVENTOS PRÓXIMOS (7 DÍAS) CON RIESGO DE CONFLICTO

Definición: Esta consulta identifica los eventos programados para la próxima semana que se cruzan en horario con otros dentro de la misma instalación.

Propósito: Permite anticipar conflictos de agenda antes de que ocurran. Con esta información, se pueden notificar a los organizadores y hacer cambios a tiempo para evitar choques de espacio o de horario.

Explicación: Primero se define una CTE llamada ev\_inst que reúne todas las relaciones entre eventos e instalaciones, tanto las principales (evento) como las adicionales (eventoinstalacion). Luego, se hace un auto-join de esa CTE para comparar cada evento con otros que usan la misma instalación. Se usa a.idEvento <> b.idEvento para evitar comparar un evento consigo mismo. Despues, se unen los resultados con las tablas evento y instalación para obtener los nombres, fechas y lugares.

En el WHERE, la condición e1.fechainicio BETWEEN CURRENT\_DATE() AND DATE\_ADD(CURRENT\_DATE(), INTERVAL 7 DAY) filtra solo los eventos que ocurrirán en los próximos siete días. También se aplican condiciones de traslape entre fechas para detectar los conflictos reales. Finalmente, se agrupa por evento y se cuenta cuántos choques tiene cada uno, mostrando los que podrían necesitar atención inmediata.

```
WITH ev_inst AS (
    SELECT idEvento, idInstalacion FROM evento
    UNION
    SELECT idEvento, idInstalacion FROM eventoinstalacion
)
```

```

SELECT e1.idEvento, e1.nombre, e1.fechaInicio, i.nombre AS instalacion,
       COUNT(*) AS conflictos_en_instalacion
  FROM ev_inst a
 JOIN evento e1 ON e1.idEvento = a.idEvento
 JOIN ev_inst b ON a.idInstalacion = b.idInstalacion AND a.idEvento <> b.idEvento
 JOIN evento e2 ON e2.idEvento = b.idEvento
 JOIN instalacion i ON i.idInstalacion = a.idInstalacion
 WHERE e1.fechaInicio BETWEEN CURRENT_DATE() AND
 DATE_ADD(CURRENT_DATE(), INTERVAL 7 DAY)
      AND e1.fechaInicio < e2.fechaFin AND e2.fechaInicio < e1.fechaFin
 GROUP BY e1.idEvento, e1.nombre, e1.fechaInicio, i.nombre
 ORDER BY e1.fechaInicio ASC, conflictos_en_instalacion DESC;

```

	idEvento	nombre	fechaInicio	instalacion	conflictos_en_instalacion
9					
10					
11					
12					
13					
14					
15					
16					
17					

## 7.10 CONSULTA DURACIÓN PROMEDIO POR CATEGORÍA (MINUTOS)

Definicion: Esta consulta calcula cuánto dura en promedio cada tipo de evento, medido en minutos, según su categoría.

Propósito: Sirve para planificar mejor los horarios y la logística. Saber cuánto dura cada tipo de evento ayuda a organizar la agenda, definir tiempos de cambio entre actividades o ajustar la disponibilidad de las salas.

Explicación: La consulta usa solo la tabla evento. Con la función `TIMESTAMPDIFF(MINUTE, fechalinicio, fechaFin)` calcula la duración de cada evento en minutos. Luego aplica `AVG` para obtener el promedio de esas duraciones agrupadas por categoría.

El `GROUP BY` categoria separa los resultados por tipo de evento, como “académico” o “lúdico”. El resultado final muestra una duración promedio que sirve como referencia para la planeación de futuros eventos.

```
SELECT categoria, AVG(TIMESTAMPDIFF(MINUTE, fechalinicio, fechaFin)) AS  
duracion_promedio_min  
FROM evento  
GROUP BY categoria;
```

The screenshot shows the MySQL Workbench interface. At the top, there is a toolbar with various icons. Below the toolbar, the SQL editor window displays the following code:

```
1 •  SELECT categoria, AVG(TIMESTAMPDIFF(MINUTE, fechalinicio, fechaFin)) AS duracion_promedio_min  
2   FROM evento  
3   GROUP BY categoria;  
4  
5
```

Below the SQL editor is the Result Grid window, which displays the query results:

categoria	duracion_promedio_min
academic	320.0000
ludic	210.0000

## 7.11 CONSULTA ORGANIZADORES CON MÁS DE 3 EVENTOS ACTIVOS (FECHAS FUTURAS)

Definición: Esta consulta muestra qué organizadores tienen más de tres eventos programados que aún no han terminado.

Propósito: Ayuda a identificar posibles casos de sobrecarga de trabajo. Con esta información se puede distribuir mejor las tareas o asignar apoyo a los organizadores con más responsabilidad en los próximos eventos.

Explicación: La consulta une las tablas evento y usuario mediante un JOIN para relacionar cada evento con su organizador.

Luego usa un WHERE e.fechaFin >= NOW() para filtrar solo los eventos que todavía no han finalizado, es decir, los activos o futuros. Después agrupa por organizador con GROUP BY u.idUsuario, u.nombre y cuenta la cantidad de eventos con COUNT(\*). El HAVING COUNT(\*) > 3 selecciona solo a los que tienen más de tres eventos activos, y finalmente se ordenan de mayor a menor según esa cantidad.

El resultado deja ver fácilmente quiénes tienen más carga de trabajo próxima.

```
SELECT u.idUsuario, u.nombre, COUNT(*) AS eventos_activos
FROM evento e
JOIN usuario u ON u.idUsuario = e.idOrganizador
WHERE e.fechaFin >= NOW()
GROUP BY u.idUsuario, u.nombre
HAVING COUNT(*) > 3
ORDER BY eventos_activos DESC;
```

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it, the SQL editor contains the following query:

```
1 •   SELECT u.idUsuario, u.nombre, COUNT(*) AS eventos_activos
2     FROM evento e
3   JOIN usuario u ON u.idUsuario = e.idOrganizador
4   WHERE e.fechaFin >= NOW()
5   GROUP BY u.idUsuario, u.nombre
6   HAVING COUNT(*) > 3
7   ORDER BY eventos_activos DESC;
8
```

Below the SQL editor is a results grid with the following columns: idUsuario, nombre, and eventos\_activos. The grid currently has no data.

## 7.12 CONSULTA EVENTOS POR MES Y CATEGORÍA (ÚLTIMOS 12 MESES)

Definición: Esta consulta muestra cuántos eventos se realizaron cada mes durante el último año, separados por categoría.

Propósito: Sirve para analizar la tendencia de los eventos a lo largo del tiempo. Con esta información se pueden planear mejor los calendarios anuales, ajustar presupuestos o preparar campañas en los meses con más actividad.

Explicación: La consulta trabaja únicamente con la tabla evento. Usa las funciones YEAR(fechainicio) y MONTH(fechainicio) para dividir los eventos por año y mes. Además, aplica LPAD(MONTH(...), 2, '0') para mostrar los meses con dos dígitos (por ejemplo, 01, 02...). El WHERE filtra los registros de los últimos 12 meses usando DATE\_SUB(CURRENT\_DATE(), INTERVAL 12 MONTH). Después, con GROUP BY se agrupan los resultados por año, mes y categoría, y se cuentan los eventos con COUNT(\*) .

Finalmente, se ordena por año, mes y categoría para mostrar una línea de tiempo organizada que refleja el comportamiento mensual de cada tipo de evento.

```
SELECT YEAR(fechainicio) AS anio, LPAD(MONTH(fechainicio),2,'0') AS mes,
       categoria, COUNT(*) AS total_eventos
  FROM evento
 WHERE fechainicio >= DATE_SUB(CURRENT_DATE(), INTERVAL 12 MONTH)
 GROUP BY YEAR(fechainicio), MONTH(fechainicio), categoria
 ORDER BY anio, mes, categoria;
```

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons for file operations, search, and navigation. Below the toolbar, the SQL editor window displays the query steps numbered 1 through 7. Step 1 is the SELECT statement. Step 7 is a blank line. The result grid below shows the data returned by the query:

	anio	mes	categoria	total_eventos
▶	2025	11	academico	3
	2025	11	ludico	1

## 7.13 CONSULTA NOTIFICACIONES POR TIPO Y RECEPTOR (ÚLTIMOS 30 DÍAS)

Definicion: Esta consulta muestra cuántas notificaciones ha recibido cada usuario en el último mes, clasificadas por tipo de mensaje.

Propósito: Sirve para analizar la comunicación dentro del sistema, ver qué usuarios reciben más notificaciones y revisar si la frecuencia o el tipo de mensajes son adecuados.

Explicación: La consulta une tres tablas: notificacion, usuario y evaluacion. El JOIN con usuario permite identificar a quién se le envió la notificación, y el JOIN con evaluacion relaciona el mensaje con el proceso evaluativo correspondiente. El WHERE limita los resultados a los últimos 30 días usando DATE\_SUB(CURRENT\_DATE(), INTERVAL 30 DAY). Luego se agrupa por usuario y tipo de notificación con GROUP BY, y se cuenta el total de mensajes con COUNT(\*) .

Finalmente, se ordena de mayor a menor según la cantidad total, mostrando quiénes recibieron más notificaciones y de qué tipo.

```
SELECT u.idUsuario, u.nombre, n.tipoNotificacion, COUNT(*) AS total
FROM notificacion n
JOIN usuario u ON u.idUsuario = n.usuarioReceptor
JOIN evaluacion ev ON ev.idEvaluacion = n.idEvaluacion
WHERE n.fechaEnvio >= DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
GROUP BY u.idUsuario, u.nombre, n.tipoNotificacion
ORDER BY total DESC;
```

The screenshot shows a MySQL Workbench interface. At the top, there are tabs labeled 'SQL FILE 14' through 'SQL FILE 17'. Below the tabs is a toolbar with various icons. A dropdown menu says 'Limit to 1000 rows'. The main area contains a numbered SQL query:

```

1 •  SELECT u.idUsuario, u.nombre, n.tipoNotificacion, COUNT(*) AS total
2   FROM notificacion n
3   JOIN usuario u ON u.idUsuario = n.usuarioReceptor
4   JOIN evaluacion ev ON ev.idEvaluacion = n.idEvaluacion
5   WHERE n.fechaEnvio >= DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
6   GROUP BY u.idUsuario, u.nombre, n.tipoNotificacion
7   ORDER BY total DESC;
8

```

Below the query is a result grid with the following columns: idUsuario, nombre, tipoNotificacion, and total. The data is as follows:

	idUsuario	nombre	tipoNotificacion	total
1	1	John Doe	Notification Type	10
2	2	Jane Doe	Notification Type	5
3	3	Bob Smith	Notification Type	3
4	4	Emily Davis	Notification Type	2
5	5	David Johnson	Notification Type	1

## 7.14 CONSULTA CALIDAD DE DATOS: EVENTOS SIN EXACTAMENTE UN ORGANIZADOR PRINCIPAL

Definición: Esta consulta identifica los eventos que no tienen exactamente un organizador principal asignado, es decir, aquellos con cero o más de un responsable marcado como principal.

Propósito: Sirve para revisar la calidad de los datos y garantizar que cada evento tenga un único organizador principal antes de pasar al proceso de evaluación.

Explicación: La consulta usa un LEFT JOIN entre las tablas evento y usuarioEvento para incluir todos los eventos, incluso los que no tienen registros de organizadores.

Luego aplica `SUM(ue.principal='S')` para contar cuántos usuarios están marcados como principales por evento. Con `GROUP BY e.idEvento` agrupa los resultados por cada evento y, en la parte del HAVING, filtra solo los casos donde el número de principales sea diferente de uno o sea nulo.

El resultado muestra los eventos con datos inconsistentes, lo que ayuda a detectar errores y mantener la base de datos limpia y confiable.

```
SELECT e.idEvento,
       SUM(ue.principal='S') AS principales
  FROM evento e
 LEFT JOIN usuarioEvento ue ON ue.idEvento = e.idEvento
 GROUP BY e.idEvento
 HAVING principales <> 1 OR principales IS NULL;
```

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it is the SQL editor window containing the query code. The code is numbered from 1 to 7. The result grid below shows one row of data.

	idEvento	principales
▶	4	NULL

## 7.15 CONSULTA PENDIENTES VENCIDOS (FECHAFIN PASADA SIN DECISIÓN FINAL)

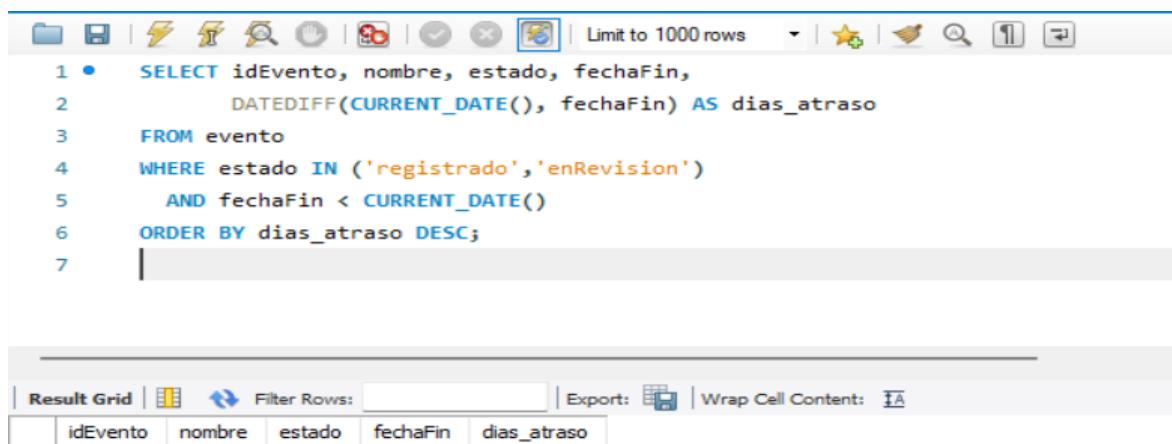
Definición: Esta consulta muestra los eventos cuya fecha de finalización ya pasó, pero que aún no tienen una decisión final, es decir, siguen en estado “registrado” o “en revisión”.

Propósito: Sirve para detectar casos atrasados que necesitan atención inmediata. Así se evita que eventos se realicen o queden abiertos sin haber sido aprobados o rechazados oficialmente.

Explicación: La consulta usa solo la tabla evento. En el WHERE, filtra los registros con estado 'registrado' o 'enRevision' y además con fechaFin < CURRENT\_DATE(), lo que significa que ya deberían haber terminado. Luego calcula cuántos días de atraso tiene cada evento usando DATEDIFF(CURRENT\_DATE(), fechaFin), que resta la fecha actual menos la fecha de finalización.

Esta consulta, ordena los resultados de mayor a menor según los días de atraso, para que los casos más urgentes aparezcan primero. De esta forma, la consulta ayuda a priorizar los eventos vencidos y tomar decisiones rápidas sobre su cierre o reprogramación.

```
SELECT idEvento, nombre, estado, fechaFin,
       DATEDIFF(CURRENT_DATE(), fechaFin) AS dias_atraso
  FROM evento
 WHERE estado IN ('registrado','enRevision')
   AND fechaFin < CURRENT_DATE()
 ORDER BY dias_atraso DESC;
```



The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it is the SQL editor window containing the query code. The code is numbered from 1 to 7. The SQL statement selects columns idEvento, nombre, estado, and fechaFin, calculates the difference between CURRENT\_DATE() and fechaFin as dias\_atraso, and orders the results by dias\_atraso in descending order. The WHERE clause filters for rows where estado is either 'registrado' or 'enRevision' and fechaFin is less than CURRENT\_DATE().

	idEvento	nombre	estado	fechaFin	dias_atraso
1					

## 7.16 CONSULTA ALIANZAS EXTERNAS POR CATEGORÍA (TOP ORGANIZACIONES).

Definición: Esta consulta muestra las organizaciones externas que más han participado en eventos, indicando también la categoría de esos eventos.

Propósito: Sirve para analizar qué alianzas o convenios tienen más actividad y en qué tipo de eventos colaboran. Esto ayuda a tomar decisiones sobre renovar o crear nuevos acuerdos con las instituciones más activas.

Explicación: La consulta une tres tablas: eventoOrganizacion, organización y evento. El JOIN con organización permite obtener el nombre de cada aliado, y el JOIN con evento trae la categoría del evento asociado. Después, agrupa los datos por nombre de organización y categoría usando GROUP BY, y cuenta la cantidad de eventos con COUNT(\*). Luego, ordena los resultados de mayor a menor con ORDER BY total\_eventos DESC y limita la salida a los 10 primeros usando LIMIT 10, para mostrar solo los socios más activos.

El resultado final permite identificar rápidamente qué organizaciones aportan más a cada categoría de eventos.

```
SELECT o.nombre AS organizacion, e.categoría, COUNT(*) AS total_eventos
FROM eventoOrganizacion eo
JOIN organización o ON o.idOrganización = eo.idOrganización
JOIN evento e ON e.idEvento = eo.idEvento
GROUP BY o.nombre, e.categoría
ORDER BY total_eventos DESC
LIMIT 10;
```

The screenshot shows a MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it is a SQL editor window containing the following code:

```
1 •   SELECT o.nombre AS organizacion, e.categoría, COUNT(*) AS total_eventos
2     FROM eventoOrganizacion eo
3       JOIN organización o ON o.idOrganización = eo.idOrganización
4       JOIN evento e ON e.idEvento = eo.idEvento
5   GROUP BY o.nombre, e.categoría
6   ORDER BY total_eventos DESC
7   LIMIT 10;
8
```

Below the SQL editor is a results grid. The title bar of the grid says "Result Grid". It contains three columns: "organización", "categoría", and "total\_eventos". The data is as follows:

	organización	categoría	total_eventos
▶	Fundación Talentos	ludico	1
	Comunidad AI Cali	academico	1

There are also "Export" and "Wrap Cell Content" buttons at the top of the results grid.

## 7.17 CONCLUSIONES EJECUCION DE CONSULTAS

Cada consulta cumple una función importante que ayuda a entender mejor cómo se están gestionando los eventos dentro del sistema.

Por ejemplo, hay consultas que sirven para medir productividad, otras que muestran la carga de trabajo o los tiempos de respuesta y también las que ayudan a detectar errores o conflictos de programación. Todo eso hace que los procesos sean más claros y que la información esté más organizada.

Gracias a estas consultas se pueden tomar decisiones más rápidas y con fundamento porque ya no se depende solo de la intuición, sino de datos reales.

## 8. OBJETOS ALMACENADOS EN MYSQL

Dentro de la base de datos “uao\_eventos” se implementaron diferentes objetos almacenados con el fin de automatizar procesos, validar información y facilitar la

generación de reportes. Estos objetos se dividen en tres categorías: funciones, vistas y triggers, los cuales complementan el funcionamiento del sistema SIGEU.

## 8.1 FUNCIONES

### 8.1.1 Función fn\_dias\_restantes\_evento(p\_id)

Esta función calcula la cantidad de días que faltan para que comience un evento. Utiliza la diferencia entre la fecha de inicio (fechalinicio) y la fecha actual (CURRENT\_DATE()) para devolver un número entero positivo (si el evento está por iniciar), cero (si inicia hoy) o negativo (si ya pasó).

Apoyar recordatorios y priorización de logística, ayudando a determinar qué eventos requieren atención inmediata.

```
-- [Función 1]
-- Nombre: fn_dias_restantes_evento
-- ¿Qué es?: Utilitaria de calendario para un evento puntual.
-- ¿Para qué sirve?: Medir cuántos días faltan para el inicio de un evento.
-- Propósito: Apoyar recordatorios, SLAs y priorizar logística.
-- Información que arroja: Entero con días (positivo=por iniciar; negativo=ya pasó).
-- Decisiones: Acelerar avales, reasignar salas, avisos automáticos si < X días.
-- Tablas/relaciones: evento (idEvento -> fechas).
DROP FUNCTION IF EXISTS fn_dias_restantes_evento;
DELIMITER $$
CREATE FUNCTION fn_dias_restantes_evento(p_id BIGINT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE d INT;
    SELECT DATEDIFF(fechaInicio, CURRENT_DATE()) INTO d
    FROM evento
    WHERE idEvento = p_id;
    RETURN d;
END$$
DELIMITER ;
```

### 8.1.2 Función fn\_evento\_duracion\_horas(p\_id)

Permite conocer la duración total de un evento en horas, calculando la diferencia entre fechalinicio y fechaFin mediante la función TIMESTAMPDIFF.

Sirve para generar reportes de uso de instalaciones y medir la ocupación horaria de cada evento.

```
-- [Función 2]
-- Nombre: fn_evento_duracion_horas
-- ¿Qué es?: Utilitaria de esfuerzo operativo.
-- ¿Para qué sirve?: Calcular horas de ocupación por evento.
-- Propósito: Insumo para KPIs de utilización/agenda y costos.
-- Información: Entero con duración en horas (redondeo natural de TIMESTAMPDIFF).
-- Decisiones: Bloques de agenda, personal, limpieza entre eventos.
-- Tablas/relaciones: evento (fechas inicio/fin).
DROP FUNCTION IF EXISTS fn_evento_duracion_horas;
DELIMITER $$
CREATE FUNCTION fn_evento_duracion_horas(p_id BIGINT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE horas INT;
    SELECT TIMESTAMPDIFF(HOUR, fechalinicio, fechaFin)
    | INTO horas
    FROM evento
    WHERE idEvento = p_id;
    RETURN horas;
END$$
DELIMITER ;
```

### 8.1.3 Función fn\_horas\_instalacion\_30d

Calcula la cantidad total de horas que una instalación ha sido utilizada en los últimos 30 días, considerando tanto los eventos principales como los adicionales.

Facilita la gestión del uso de espacios, ayudando a detectar saturación o necesidad de mantenimiento en las instalaciones.

```

-- [Función 3]
-- Nombre: fn_horas_instalacion_30d
-- ¿Qué es?: Agregación de uso de una instalación (últimos 30 días).
-- ¿Para qué sirve?: Medir saturación real por espacio.
-- Propósito: Insumo para priorizar ampliaciones/reparaciones o reasignar reservas.
-- Información: Entero con horas sumadas de todos los eventos (incluye instalación por defecto y adicional).
-- Decisiones: Abrir nuevas salas, limitar cupos, ajustar cronogramas.
-- Tablas/relaciones: evento, eventoInstalacion, instalacion.
DROP FUNCTION IF EXISTS fn_horas_instalacion_30d;
DELIMITER $$ 
CREATE FUNCTION fn_horas_instalacion_30d(p_inst BIGINT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE total_horas INT;
    SELECT IFNULL(SUM(TIMESTAMPDIFF(HOUR, e.fechaInicio, e.fechaFin)),0) INTO total_horas
    FROM (
        SELECT idEvento, idInstalacion FROM evento
        UNION ALL
        SELECT idEvento, idInstalacion FROM eventoInstalacion
    ) ei
    JOIN evento e ON e.idEvento = ei.idEvento
    WHERE ei.idInstalacion = p_inst
    AND e.fechaInicio >= DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY);
    RETURN total_horas;
END$$
DELIMITER ;

```

## 8.2 VISTAS

### 8.2.1 Vista v\_eventos\_pendientes

Muestra todos los eventos que se encuentran en estado “registrado” o “en revisión”.

Permite al comité de evaluación visualizar rápidamente los eventos pendientes de decisión para priorizar su revisión.

```

-- [Vista 1]
-- Nombre: v_eventos_pendientes
-- ¿Qué es?: Lista de eventos sin decisión final.
-- ¿Para qué sirve?: Controlar backlog y priorizar revisiones.
-- Propósito: Ver rápidamente los casos en 'registrado' o 'enRevision'.
-- Información: Todas las columnas de evento filtradas por estado.
-- Decisiones: Aumentar capacidad de comité, fijar fechas de revisión.
-- Tablas/relaciones: evento (solo).
CREATE OR REPLACE VIEW v_eventos_pendientes AS
SELECT *
FROM evento
WHERE estado IN ('registrado', 'enRevision');

```

### 8.2.2 Vista v\_evento\_x\_instalacion

Unifica la información de los eventos con sus respectivas instalaciones, tanto principales como adicionales.

Facilita el control del uso de espacios y la organización de la agenda institucional.

```

-- [Vista 2]
-- Nombre: v_evento_x_instalacion
-- ¿Qué es?: Mapa evento + instalación unificado.
-- ¿Para qué sirve?: Cálculos de ocupación, solapamientos y reportes por sala.
-- Propósito: Consolidar instalación por defecto + instalaciones adicionales.
-- Información: idEvento, idInstalacion, nombre de instalación.
-- Decisiones: Reprogramar, dividir eventos, asignar personal por sala.
-- Tablas/relaciones: evento + eventoInstalacion + instalacion.
CREATE OR REPLACE VIEW v_evento_x_instalacion AS
SELECT e.idEvento, e.idInstalacion AS idInstalacion, i.nombre AS instalacion
FROM evento e
JOIN instalacion i ON i.idInstalacion = e.idInstalacion
UNION
SELECT ei.idEvento, ei.idInstalacion, i2.nombre AS instalacion
FROM eventoInstalacion ei
JOIN instalacion i2 ON i2.idInstalacion = ei.idInstalacion;

```

### 8.3.3 Vista v\_conflictos\_instalacion

Identifica pares de eventos que se cruzan en horario dentro de la misma instalación.

Permite detectar conflictos de programación para evitar sobreaforos o choques de reservas.

```
-- [Vista 3]
-- Nombre: v_conflictos_instalacion
-- ¿Qué es?: Pares de eventos que se solapan en la misma instalación.
-- ¿Para qué sirve?: Anticipar choques y evitar sobreaforo.
-- Propósito: Listar conflictos con detalle (eventos y sala).
-- Información: idInstalacion, instalacion, idEvento1, idEvento2, ventana de choque.
-- Decisiones: Cambiar horario/sala, avisos, políticas de booking.
-- Tablas/relaciones: v_evento_x_instalacion, evento (self-join por sala/tiempo).
CREATE OR REPLACE VIEW v_conflictos_instalacion AS
SELECT
    a.idInstalacion,
    a.instalacion,
    e1.idEvento AS idEvento1,
    e2.idEvento AS idEvento2,
    GREATEST(e1.fechaInicio, e2.fechaInicio) AS choque_inicio,
    LEAST(e1.fechaFin, e2.fechaFin) AS choque_fin
FROM v_evento_x_instalacion a
JOIN evento e1 ON e1.idEvento = a.idEvento
JOIN v_evento_x_instalacion b
    ON a.idInstalacion = b.idInstalacion
    AND a.idEvento < b.idEvento
JOIN evento e2 ON e2.idEvento = b.idEvento
WHERE e1.fechaInicio < e2.fechaFin
    AND e2.fechaInicio < e1.fechaFin;
```

## 8.3 TRIGGERS

### 8.3.1 Trigger trg\_ue\_check\_rol (BEFORE INSERT)

Valida que solo los usuarios con rol de “docente” o “estudiante” puedan asociarse a eventos dentro de la tabla usuarioEvento.

Mantiene la integridad lógica del sistema al restringir la participación a roles válidos.

```
-- [Trigger 1]
-- Nombre: trg_ue_check_rol (BEFORE INSERT)
-- ¿Qué es?: Validación de negocio sobre roles permitidos.
-- ¿Para qué sirve?: Impedir que perfiles no válidos se asocien a eventos.
-- Propósito: Garantizar que solo 'docente'/'estudiante' estén en usuarioEvento.
-- Información: Falla con SIGNAL si el rol no es permitido (control de calidad de datos).
-- Decisiones: Mantener integridad del proceso de organización.
-- Tablas/relaciones: usuarioEvento (NEW.idUsuario) → usuario.rol
DROP TRIGGER IF EXISTS trg_ue_check_rol;
DELIMITER $$

CREATE TRIGGER trg_ue_check_rol
BEFORE INSERT ON usuarioEvento
FOR EACH ROW
BEGIN
    DECLARE vrol VARCHAR(30);
    SELECT rol INTO vrol FROM usuario WHERE idUsuario = NEW.idUsuario;
    IF vrol NOT IN ('docente','estudiante') THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'usuarioEvento: solo roles docente/estudiante';
    END IF;
END$$
DELIMITER ;
```

### 8.3.2 Trigger trg\_ue\_unico\_principal\_ins (BEFORE INSERT)

Evita que existan dos organizadores principales (principal = 'S') para un mismo evento al momento de la inserción.

Garantiza que cada evento tenga un único responsable principal.

```
-- [Trigger 2]
-- Nombre: trg_ue_unico_principal_ins (BEFORE INSERT)
-- ¿Qué es?: Regla de unicidad del organizador principal por evento (en altas).
-- ¿Para qué sirve?: Evitar ambigüedad de responsabilidad.
-- Propósito: Permitir solo un principal='S' por idEvento.
-- Información: Falla con SIGNAL si ya existe un principal.
-- Decisiones: Claridad de liderazgo y trazabilidad.
-- Tablas/relaciones: usuarioEvento (conteo por idEvento).
DROP TRIGGER IF EXISTS trg_ue_unico_principal_ins;
DELIMITER $$

CREATE TRIGGER trg_ue_unico_principal_ins
BEFORE INSERT ON usuarioEvento
FOR EACH ROW
BEGIN
    IF NEW.principal = 'S' THEN
        IF (SELECT COUNT(*) FROM usuarioEvento WHERE idEvento = NEW.idEvento AND principal='S') > 0 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'usuarioEvento: ya existe un principal= S para este evento';
        END IF;
    END IF;
END$$
DELIMITER ;
```

### 8.3.3 Trigger trg\_ue\_unico\_principal\_upd (BEFORE UPDATE)

Aplica la misma restricción anterior, pero al momento de actualizar registros existentes. Preserva la unicidad del organizador principal en modificaciones posteriores.

```
-- [Trigger 3]
-- Nombre: trg_ue_unico_principal_upd (BEFORE UPDATE)
-- ¿Qué es?: Regla de unicidad del organizador principal por evento (en cambios).
-- ¿Para qué sirve?: Evitar que una actualización genere doble principal.
-- Propósito: Mantener la unicidad en updates.
-- Información: Falla con SIGNAL si ya hay un principal distinto al actual.
-- Decisiones: Consistencia del flujo organizativo.
-- Tablas/relaciones: usuarioEvento (conteo por idEvento, excluyendo el propio).
DROP TRIGGER IF EXISTS trg_ue_unico_principal_upd;
DELIMITER $$
CREATE TRIGGER trg_ue_unico_principal_upd
BEFORE UPDATE ON usuarioEvento
FOR EACH ROW
BEGIN
    IF NEW.principal = 'S' AND (OLD.principal <> 'S') THEN
        IF (SELECT COUNT(*) FROM usuarioEvento
            WHERE idEvento = NEW.idEvento
            AND principal='S'
            AND idUsuarioEvento <> OLD.idUsuarioEvento) > 0 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'usuarioEvento: ya existe un principal= S para este evento';
        END IF;
    END IF;
END$$
DELIMITER ;
```

### 8.3.4 Trigger trg\_eval\_after\_insert (AFTER INSERT)

Una vez se registra una evaluación, actualiza automáticamente el estado del evento y crea una notificación para el organizador principal.

Automatiza el flujo de comunicación tras la evaluación de un evento.

```

-- [Trigger 4]
-- Nombre: trg_eval_after_insert (AFTER INSERT)
-- ¿Qué es?: Automatización post-evaluación.
-- ¿Para qué sirve?: Sincronizar estado del evento y generar notificación al organizador.
-- Propósito: Al registrar una evaluación, actualizar evento.estado y crear notificación formal.
-- Información: Inserta en notificación con datos coherentes al esquema (tipoNotificación, justificación, urlPDF, receptor).
-- Decisiones: Trazabilidad y comunicación inmediata del veredicto.
-- Tablas/relaciones: evaluacion NEW → evento (actualiza) → notificación (INSERT con idOrganizador como receptor).
DROP TRIGGER IF EXISTS trg_eval_after_insert;
DELIMITER $$
CREATE TRIGGER trg_eval_after_insert
AFTER INSERT ON evaluacion
FOR EACH ROW
BEGIN
    -- 1) Actualiza estado del evento
    UPDATE evento
    | SET estado = NEW.estado
    WHERE idEvento = NEW.idEvento;

    -- 2) Crea notificación al organizador principal (usamos idOrganizador del evento)
    INSERT INTO notificacion (idEvaluacion, tipoNotificación, fechaEnvio, justificación, urlPDF, usuarioReceptor)
    SELECT NEW.idEvaluacion, NEW.estado, NOW(), NEW.comentarios, NEW.actaPDF, e.idOrganizador
    FROM evento e
    WHERE e.idEvento = NEW.idEvento;
END$$
DELIMITER ;

```

### 8.3.5 Trigger trg\_eval\_after\_update (AFTER UPDATE)

Actúa de forma similar al anterior, pero cuando se actualiza una evaluación existente.

Propósito: Mantiene sincronizados los cambios de estado y asegura la trazabilidad de las notificaciones.

```

-- (Opcional pero recomendable) Trigger homólogo para updates
DROP TRIGGER IF EXISTS trg_eval_after_update;
DELIMITER $$
CREATE TRIGGER trg_eval_after_update
AFTER UPDATE ON evaluacion
FOR EACH ROW
BEGIN
    UPDATE evento
    | SET estado = NEW.estado
    WHERE idEvento = NEW.idEvento;

    INSERT INTO notificacion (idEvaluacion, tipoNotificación, fechaEnvio, justificación, urlPDF, usuarioReceptor)
    SELECT NEW.idEvaluacion, NEW.estado, NOW(), NEW.comentarios, NEW.actaPDF, e.idOrganizador
    FROM evento e
    WHERE e.idEvento = NEW.idEvento;
END$$
DELIMITER ;

```

## ANEXO 1. EJECUCION PASO A PASO DEL PROYECTO

VER ARCHIVO ADJUNTO :

/docs/ARCHIVO\_PASO\_A\_PASO\_EJECUCION\_SIGEU.pdf

## ANEXO 2. ACCESO GITHUB – BACKEND

<https://github.com/davidhernandezp-dotcom/Proyecto-sigeu>

## ANEXO 3. ACCESO PRUEBAS POSTMAN

<https://andresgomezdoctorado-8529723.postman.co/workspace/andres-gomez's-Workspace~25a1ed28-155c-4f0f-8243-ddf433ed08ec/collection/49033182-3c0f62d8-68e1-48f7-97e5-8c46d3e3771f?action=share&creator=49033182&active-environment=49033182-d9c4618b-a1d6-43fe-892b-1dbf047b8f7a>