INFORME – Sistema de Gestión de Reservas de Salas de Estudio

Base de Datos 1 – Universidad Católica del Uruguay (UCU)
Segundo Semestre 2025
Trabajo Obligatorio – Gestión de Salas de Estudio
Integrantes: Esteban Durán, Sebastián Martony y Joaquín Viola

1. Introducción

El presente informe documenta el desarrollo completo del sistema solicitado por la consigna del trabajo obligatorio “Sistema para Gestión de Reserva de Salas de Estudio” para la UCU. Este proyecto abarcó:

- Diseño y construcción de la base de datos MySQL.
- Implementación de un backend en Python (FastAPI) sin ORM.
- Creación de un frontend en React + Vite + Tailwind.
- Dockerización de la aplicación.
- Implementación de todas las reglas de negocio descritas en la consigna.
- Validaciones y controles de seguridad.

El objetivo fue desarrollar un sistema que permita administrar salas de estudio de manera centralizada, registrando reservas, participantes, asistencia y sanciones, y mantener trazabilidad completa del uso de las salas.

2. Descripción General del Sistema

El sistema implementa:

Gestión completa (ABM) de:

- Participantes
- Salas
- Turnos
- Reservas

- Sanciones

Procesos automáticos:

- Registro de asistencia
- Generación automática de sanciones cuando una reserva finaliza sin asistencia
- Validación de restricciones (horas permitidas, tipos de salas, límite de horas diarias, límite semanal, etc.)

Seguridad:

- Contraseñas hasheadas con bcrypt
- Validación de datos en las distintas capas
- Restricciones en la base de datos
- Dockerización evitando exponer puertos innecesarios

Consultas para reportes (BI):

- Salas más reservadas
- Turnos más usados
- Asistencias
- Reservas por facultad/programa
- Entre otras

Estas funcionalidades abarcan todos los puntos obligatorios definidos en la consigna del curso.

3. Arquitectura General

La solución se estructuró en **tres capas**:

Frontend (React + Vite + Tailwind)
|
v
Backend (FastAPI – Python – REST)
|

v
Base de Datos (MySQL 8 – Docker)

3.1. Tecnologías utilizadas

Frontend

- React 18
- Vite
- TailwindCSS
- Lucide Icons
- TypeScript
- Next (para compatibilidad de estructura)
- Vercel Analytics (removible)

Se incluyeron componentes reutilizables, formularios y validaciones complementarias antes de enviar los datos al backend.

Backend

- Python 3.12
- FastAPI
- bcrypt
- mysql-connector-python
- Modularización por routers, models y services
- Validadores propios para todas las reglas de negocio
- Responses estandarizadas (200, 400, 404, 422)

Base de Datos

- MySQL 8.0
- Estructura exacta a la solicitada en la letra:
    - participante
    - login
    - sala
    - reserva

- turno
- sancion_participante
- etc.

Infraestructura

- Docker
- Docker Compose (servicios: backend, base de datos)

4. Base de Datos (MySQL)

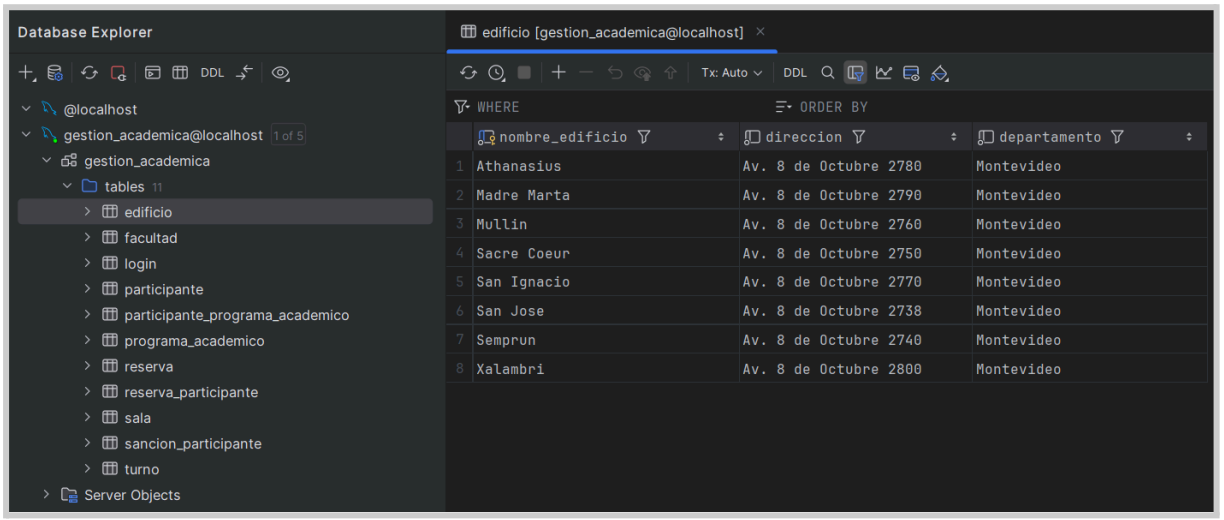
La estructura respeta estrictamente los lineamientos:

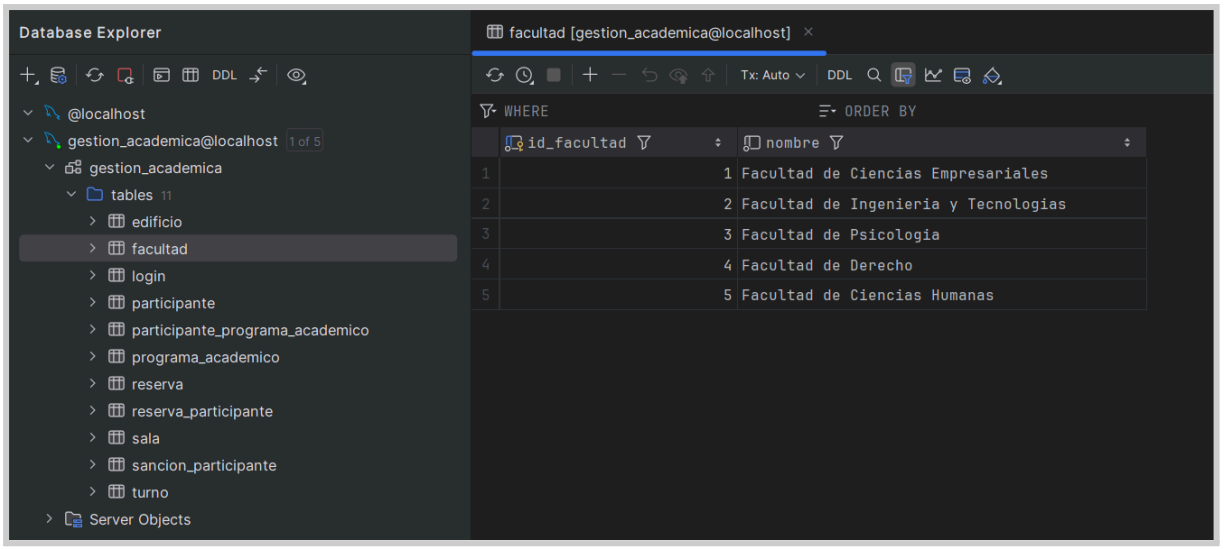
- Tablas y claves primarias
- Relaciones con claves foráneas y ON DELETE CASCADE cuando corresponde
- Validaciones con CHECK (por ejemplo: duración del turno = 1 hora exacta)
- Integridad referencial total

Incluye:

- Script completo SQL, que se ejecuta automáticamente al levantar el contendor, en /database/docker-entry
- Datos iniciales (salas, facultades, programas académicos, turnos, participantes)

Capturas a insertar aquí:





- Creación completa de tablas
- Datos insertados
- SHOW TABLES
- Ejecución en Docker

5. Backend (FastAPI)

La API se implementó modularmente con:

- routes/
- models/
- services/
- utils/helpers.py
- utils/validators.py

5.1. Endpoints incluidos

Sistema de Gestión de Salas UCU 1.0.0 OAS 3.1

/openapi.json

Backend del obligatorio de Bases de Datos 1 - UCU

Participantes

- GET /participantes/ Listar Todos
- POST /participantes/ Crear
- GET /participantes/{ci} Obtener
- PATCH /participantes/{ci} Actualizar
- DELETE /participantes/{ci} Borrar

Login

- POST /login/ Crear Login Endpoint
- GET /login/{correo} Obtener Login Endpoint
- PATCH /login/{correo} Actualizar Login Endpoint
- DELETE /login/{correo} Eliminar Login Endpoint
- POST /login/authenticate Autenticar Login Endpoint

Salas

- GET /salas/ Obtener Salas
- POST /salas/ Agregar Sala
- DELETE /salas/{nombre_sala}/{edificio} Eliminar Sala
- PUT /salas/{nombre_sala}/{edificio} Actualizar Sala

Reservas

- GET /reservas/ Obtener Reservas
- POST /reservas/ Crear Nueva Reserva
- GET /reservas/{id_reserva} Obtener Reserva
- DELETE /reservas/{id_reserva} Eliminar Reserva
- PATCH /reservas/{id_reserva}/cancelar Cancelar Reserva Endpoint
- PATCH /reservas/{id_reserva}/finalizar Finalizar Reserva Endpoint
- POST /reservas/{id_reserva}/participantes Agregar Participante
- GET /reservas/{id_reserva}/participantes Obtener Participantes Reserva
- PATCH /reservas/{id_reserva}/asistencia Actualizar Asistencia Reserva

Sanciones

- GET /sanciones/ Listar Todas
- POST /sanciones/ Crear
- GET /sanciones/participante/{ci} Listar Por Ci
- PATCH /sanciones/{id_sancion} Actualizar
- DELETE /sanciones/{id_sancion} Eliminar

5.2. Reglas de negocio implementadas

Se implementaron restricciones como:

- *Reservas solo en bloques de 1 hora*
- *Máximo 2 horas diarias (para grado)*
- *Máximo 3 reservas activas por semana (para grado)*
- *Salas posgrado disponibles solo para posgrado/docentes*
- *Salas docentes exclusivas para docentes*
- *Capacidad de sala no superada*
- *Asistencia obligatoria*
- *Sanción automática de 2 meses* si nadie asiste

Estas reglas están implementadas tanto en forma de *triggers* en la DB MySQL, como en *validators* en los servicios del backend Python.

Capturas para insertar aquí

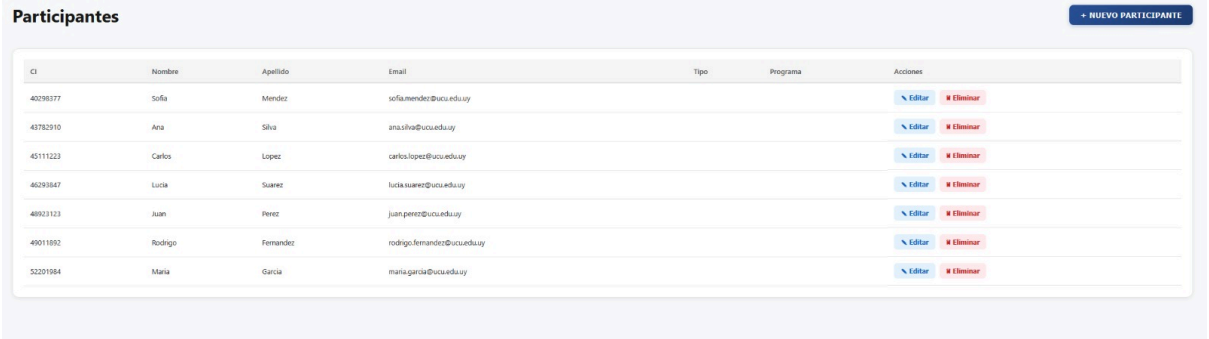
- Swagger mostrando todos los endpoints
- Ejemplos reales de POST /reservas
- Ejemplos reales de PATCH asistencia
- Mensajes de error de validación

6. Frontend (React + Vite + Tailwind)

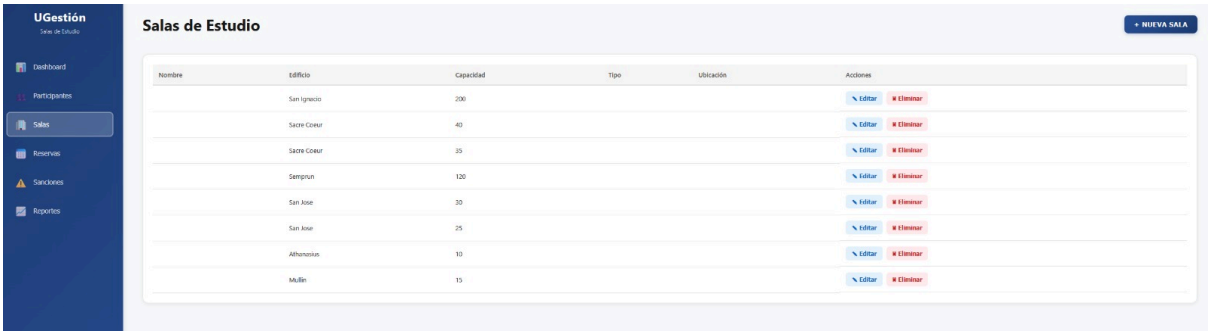
El frontend implementa:

- Pantalla de login

- Pantalla de reservas actuales



- Formulario de creación de reservas
- Gestión de salas



- Gestión de participantes
- Vista de sanciones

El objetivo fue entregar una interfaz clara que facilite a los funcionarios de UCU el uso del sistema.

Dependencias principales (package.json)

(Usa las que pasaste exactamente)

Capturas para insertar aquí

- Vista inicial
- ABM participantes
- ABM salas
- Reservas
- Sanciones
- Errores de validación del front

7. Validaciones y Seguridad

Seguridad

- Contraseñas hasheadas con bcrypt en el backend
- No se guardan contraseñas en texto plano
- No se usa ORM → consultas SQL controladas manualmente
- Sanitización de inputs
- Validaciones Pydantic en todos los modelos
- Validaciones adicionales en base de datos con CHECK + FK

Validaciones de reglas del negocio

Ejecutadas en utils/validators.py:

- Bloques estrictos de 1 hora
- No superposición de horarios
- Restricción de 2 horas por día
- Restricción de 3 reservas por semana
- Restricciones según tipo de sala
- Capacidad de sala
- Sanciones activas impiden reservar

8. Bitácora del Trabajo

Resumen realista del proceso:

1. Instalación de entornos, Docker y estructura del proyecto.
2. Diseño inicial del modelo relacional según la letra.
3. Creación del script SQL en MySQL y pruebas en Docker.
4. Implementación del backend base (routers, services, db connection).
5. Implementación progresiva de reglas de negocio.

6. Creación de endpoints adicionales (asistencia, sanciones, reportes).
7. Construcción del frontend con Vite + React + Tailwind.
8. Conexión front–back y pruebas de integración.
9. Corrección de errores detectados en swagger.
10. Generación del informe, instructivo y capturas de pantalla.

9. Decisiones de Diseño

- Elegimos FastAPI por su velocidad, claridad, tipado y compatibilidad con Pydantic.
- Se evitó el uso de ORM para respetar la letra del curso.
- Se modularizó por servicios para mantener un código limpio y testeable.
- Todas las reglas de negocio se centralizaron en un módulo de validadores para evitar duplicación y mejorar el mantenimiento.
- Se optó por React + Vite por eficiencia y facilidad para construir SPAs modernas.
- Se dockerizó todo el entorno para tener una reproducibilidad total del sistema mucho más fácil.

10. Bibliografía

- Documentación oficial de FastAPI
- Documentación de MySQL 8
- Pydantic – documentación oficial
- Docker & Docker Compose docs
- TailwindCSS docs
- bcrypt documentation
- Consigna oficial del curso (UCU, Base de Datos 1)

11. Conclusión

Se intentó realizar un sistema robusto y consistente, que use las tecnologías más apropiadas y que funcione acorde a lo pedido. Tanto el uso de Swagger, MySQL y el frontend fueron fundamentales, y

las múltiples verificaciones en las distintas capas fueron capaces de generar seguridad y un sistema más sólido .

Las capturas a ser agregadas demostrarán:

- Creación correcta de la base
- Uso real de todos los endpoints
- Validaciones funcionando
- Sanciones generadas automáticamente
- Frontend operativo