Informe Obligatorio de Bases de Datos

Introducción

En este obligatorio desarrollamos un sistema para gestionar las reservas de las salas de estudio de la UCU. El sistema permite manejar salas, participantes, reservas, asistencia y sanciones según diversas reglas que se plantean en la letra del obligatorio.

Para esto diseñamos una base de datos en MySQL y una aplicación en Python, las cuales se pueden ver funcionando en un frontend hecho con HTML.

Fundamentación de las decisiones de implementación

Fundamentación BACK: Para el backend usamos Python junto con el framework Flask. Elegimos esta combinación porque Flask es fácil de configurar y como no sabíamos cómo usarlo nos fue de mucha utilidad. Esto nos permitió conectar rápidamente el backend con el frontend y organizar funciones como manejo de reservas, validaciones y consultas a la base de datos de una forma clara y ordenada.

Fundamentación FRONT: trabajamos usando HTML como lenguaje para el frontend, complementado con CSS y JavaScript debido a que nos resultó un lenguaje el cual es fácil de actualizar y ver los cambios a medida que los vamos realizando. Además, hizo que el diseño y la organización de la interfaz del proyecto fueran mucho más fáciles de realizar en cuanto a estructura y funcionalidad.

Modelo de datos

El sistema se basa en un modelo relacional implementado en MySQL que incluye 11 tablas principales organizadas para gestionar todos los aspectos del sistema de reservas.

La tabla 'login' almacena las credenciales de acceso de los usuarios, mientras que 'participante' contiene los datos personales (CI, nombre, apellido y email). Las salas se registran en la tabla 'sala' con información sobre su ubicación, capacidad y tipo (libre, posgrado o docente).

Para la gestión temporal, implementamos la tabla 'turno' que define 15 bloques horarios desde las 8:00 AM hasta las 11:00 PM. Las reservas se almacenan en la tabla 'reserva', que registra la sala, edificio, fecha, turno y estado (activa, cancelada, sin asistencia o finalizada).

Dado que múltiples participantes pueden estar en una misma reserva, utilizamos la tabla intermedia 'reserva_participante' que implementa la relación muchos a muchos, registrando además la asistencia de cada participante. Las sanciones se gestionan en 'sancion_participante', almacenando las fechas de inicio y fin del período sancionado.

Para la estructura organizacional de la universidad, contamos con las tablas 'edificio' (con 2 edificios de prueba), 'facultad' (con 3 facultades) y 'programa_academico' que define las carreras de grado y posgrado. La relación entre usuarios y programas académicos se

establece mediante 'participante_programa_academico', que además especifica el rol de cada persona (alumno o docente).

Este diseño nos permitió cumplir con todos los requisitos de la letra del obligatorio y facilitó la implementación de las reglas de negocio establecidas

Sistema de Validaciones en Múltiples Capas

Implementamos un sistema de validación robusto que opera en tres niveles distintos, garantizando la integridad de los datos y el cumplimiento de las reglas de negocio establecidas en la letra del obligatorio.

En el frontend utilizamos JavaScript para validaciones inmediatas que mejoran la experiencia del usuario. Verificamos el formato correcto de emails institucionales, validamos que las fechas ingresadas sean coherentes, comprobamos que todos los campos obligatorios estén completos antes de enviar el formulario, verificamos que la cantidad de participantes no exceda la capacidad de la sala seleccionada, y mostramos mensajes de error descriptivos que guían al usuario para corregir sus datos.

El backend implementa las validaciones críticas de negocio usando Python. Antes de confirmar una reserva, verificamos que la sala esté disponible en el horario solicitado, controlamos que los usuarios de grado no excedan las 2 horas diarias de reserva, validamos que ningún participante tenga más de 3 reservas activas en la semana, verificamos que la cantidad de participantes no supere la capacidad física de la sala, comprobamos que el usuario tenga permisos para reservar el tipo de sala solicitado según su rol, y verificamos que ningún participante tenga sanciones activas. Además, implementamos un manejo robusto de excepciones que previene errores inesperados y protege información sensible.

A nivel de base de datos utilizamos los mecanismos propios de MySQL para garantizar la integridad referencial. Definimos PRIMARY KEYs en todas las tablas para asegurar la unicidad de registros, establecimos FOREIGN KEYs que mantienen la consistencia entre tablas relacionadas y previenen la inserción de datos huérfanos, implementamos constraints CHECK para validar que los campos como tipo_sala y estado solo acepten valores predefinidos, utilizamos constraints NOT NULL en campos esenciales, y aprovechamos la validación automática de tipos de datos que ofrece MySQL.

Esta estrategia de validación en múltiples capas nos dio seguridad en el funcionamiento del sistema y redujo significativamente la posibilidad de errores o inconsistencias en los datos.

Medidas de Seguridad Implementadas

Establecimos un sistema de validación de permisos basado en roles que verifica en cada operación si el usuario tiene autorización para realizarla. Por ejemplo, solo los usuarios con rol de docente pueden reservar salas de tipo "docente", y los estudiantes de grado están limitados a salas de tipo "libre".

Implementamos sanitización de inputs en el backend para todos los datos recibidos desde el frontend. Cada valor es validado contra patrones esperados antes de ser procesado, evitando la inyección de código malicioso.

Para prevenir ataques de SQL Injection, utilizamos prepared statements en todas las consultas a la base de datos. Esto significa que los parámetros de usuario nunca se concatenan directamente en las queries SQL, sino que se pasan como parámetros separados que MySQL sanitiza automáticamente.

Agregamos validación estricta de tipos de datos tanto en Python como en MySQL. Los campos numéricos solo aceptan números, los emails deben tener formato válido, las fechas son validadas antes de ser almacenadas, y las cédulas deben tener exactamente 7 u 8 dígitos.

Implementamos un manejo seguro de errores que previene la exposición de información sensible. Cuando ocurre un error, el sistema muestra al usuario un mensaje genérico amigable, pero internamente registra el error completo para que los administradores puedan diagnosticar problemas sin comprometer la seguridad.

Aunque por requisitos del obligatorio no podíamos usar un ORM, utilizamos la librería mysql-connector-python que implementa buenas prácticas de seguridad por defecto, como el manejo automático de escape de caracteres especiales en las consultas parametrizadas.

Mejoras implementadas o consideradas

Consideramos fuertemente usar React para el frontend porque se utiliza mucho para desarrollo web en la actualidad, pero debido a que no teníamos mucho dominio sobre cómo usarlo terminamos resolviendo que si lo incorporamos le estaríamos agregando una complejidad innecesaria al proyecto, cuando tranquilamente podríamos haber usado HTML. Para nosotros este último lenguaje es más conocido y seguro de incorporar en el proyecto, además de resultarnos de mayor fluidez para avanzar más rápido.

Al principio no teníamos claro cómo manejar las rutas del backend para que el frontend pudiera enviar y recibir datos correctamente. Lo solucionamos revisando la documentación de Flask y creando endpoints específicos para cada acción (crear reserva, listar salas, registrar asistencia, etc.).

Al principio no estábamos encriptando los datos sensibles. Implementamos hashing para las contraseñas usando funciones de Python para garantizar seguridad en el almacenamiento.

Algunas consultas no devolvían la información esperada porque nos faltaban joins. Revisamos la estructura relacional, hicimos pruebas por separado en MySQL y luego las integramos correctamente en el backend.

Incorporamos un sistema de login con perfiles diferenciados: docente, estudiante y posgrado. Esto permite controlar quién puede realizar reservas, administrar salas o gestionar asistencias y sanciones. Esto se refleja en filtrado de salas disponibles según tipo

de usuario, validación de restricciones (2 horas diarias para grado, sin límites para docentes en salas exclusivas) y un sistema de verificación de sanciones activas al momento del login.

Bitácora

Al empezar el proyecto nos planteamos nuestras ideas y las organizamos de manera que tuviéramos entendido cómo queríamos desarrollar el proyecto. Decidimos dividir las tareas según lo que cada uno estuviera más familiarizado trabajando, es decir, uno se encargaba de realizar las consultas, otro de programar el backend y por último, otro de realizar la interfaz en frontend.

Esta manera de organizarnos nos permitió tener el proyecto casi terminado en muy poco tiempo, cosa de que tuviéramos tiempo de aprender cómo conectar el backend con el frontend, y de cifrar los datos sensibles como las contraseñas.

Durante los primeros días del proyecto fuimos avanzando de manera paralela, cada uno trabajando por su parte y compartiendo nuestros avances, para luego actualizarnos en el trabajo de cada integrante y explicar qué hizo cada uno por su cuenta. A medida que progresamos empezamos a trabajar al simultáneo en una sola computadora o varias si es que estábamos en lugares distintos.

Intentamos que lo primero que estuviera realizado fueran las consultas, para así tener un mayor entendimiento de cómo realizar el backend respecto a la base de datos. Esto no dio mayores complicaciones. Al mismo tiempo, se estaba creando un frontend con lo que considerábamos necesario para simular cómo sería una web real de reserva de salas de la UCU, con todas las funciones y pestañas necesarias para que esta pareciera lo más cercano a la realidad posible.

Por último, conectamos el backend con el frontend e hicimos los últimos retoques para que todo funcionara como debía y no quedaran cosas sueltas que no funcionaran o que corrieran riesgo de romperse.

Conclusión

El desarrollo de este sistema nos permitió organizar de forma clara y unificada todo el proceso de reservas de salas en la UCU. Logramos implementar las funciones principales (reservas, salas, participantes, asistencia y sanciones) junto con las reglas necesarias.
En general, el obligatorio cumplió los objetivos planteados y nos ayudó a entender mejor cómo implementar herramientas como Flask.

Bibliografía

Pallets Projects. (s.f.). *Flask – Quickstart*. Flask Documentation. Recuperado el 23 de noviembre de 2025, de https://flask.palletsprojects.com/en/stable/quickstart/

Universidad Católica del Uruguay. (s.f.). Curso Base de Datos 1: Material de clase y prácticos.

MySQL. (s.f.). MySQL 8.0 reference manual. Oracle. Recuperado de https://dev.mysql.com/doc/refman/8.0/en/

MySQL. (s.f.). MySQL Connector/Python developer guide. Oracle. Recuperado de https://dev.mysql.com/doc/connector-python/en/

Docker. (s.f.). Docker documentation. Docker Inc. Recuperado de https://docs.docker.com/