Base de Datos I

Trabajo Obligatorio - Sistema para Gestión de Reserva de Salas de Estudio

Docentes:

Juan Kosut

Sofía Guerrico

Estudiantes:

Valentina Blanco

Andrea González

María Belén Kanas

Fecha:

23 de noviembre de 2025

Índice

Introducción

Descripción del Trabajo Obligatorio

El objetivo del trabajo obligatorio es diseñar e implementar un sistema de información para la gestión de las salas de estudio en la Universidad Católica del Uruguay. El sistema permite administrar reservas, registrar asistencias y generar reportes que apoyen tanto la gestión académica como la toma de decisiones institucionales.

Las salas de estudio son espacios utilizados por estudiantes y docentes para diversas actividades académicas, tales como reuniones grupales, videoconferencias o trabajos colaborativos. El problema surge en que, actualmente, la administración de estas reservas se realiza de forma manual mediante planillas de papel, gestionadas por funcionarios de biblioteca, secretaría y administración. Este procedimiento presenta limitaciones en términos de trazabilidad, control de disponibilidad, registros históricos y detección de incumplimientos.

Por lo tanto, con el propósito de mejorar este proceso, se desarrolló un sistema digital que permite administrar participantes, salas y reservas, así como aplicar reglas de uso y registrar sanciones ante incumplimientos. Entre las funcionalidades principales se incluyen la administración de participantes y sus roles, gestión de salas y edificios, creación y control de reservas por bloques horarios, registro de asistencia por participante, consultas y reportes sobre uso, cancelaciones y sanciones.

Reglas y restricciones consideradas

Para garantizar un uso ordenado y equitativo de las salas, se incorporaron al sistema las siguientes reglas establecidas en la propuesta del trabajo obligatorio:

- El horario disponible para reservar salas comienza a las 8:00 y finaliza a las 23:00.
- Las reservas se pueden realizar únicamente por bloques de una hora; para intervalos mayores deben solicitarse bloques consecutivos.
- Cada sala posee una capacidad máxima que no puede ser superada; esta incluye a todos los participantes (alumnos y docentes)
- Los usuarios se clasifican en estudiantes de grado, estudiantes de posgrado y docentes.
- Existen tres tipos de salas:
    - Uso libre (accesibles a estudiantes de grado, posgrado y docentes).
    - Exclusivas de posgrado
    - Exclusivas de docentes

Restricciones que pueden derivar a sanciones

- Un usuario no puede ocupar salas por más de dos horas diarias en cualquiera de los edificios.
- No puede tener más de tres reservas activas por semana.
- Estas restricciones no aplican a docentes ni a estudiantes de posgrados, únicamente cuando utilizan las salas exclusivas para su tipo.
- Si ninguno de los participantes registrados asiste a la reserva, se genera una sanción de dos meses sin poder realizar nuevas reservas.

Estructura del proyecto

Tecnologías Utilizadas

El sistema se compone de dos partes principales: backend y frontend, administradas mediante repositorios independientes en GitHub.

Link al repositorio de Backend: https://github.com/belenkanas/BDI-Obligatorio---V.Blanco-A.Gonzalez-B.Kanas.git

Link al repositorio de Frontend: https://github.com/andreagonzalezucu/FrontEnd-Obligatorio-BD.git

Backend

El backend emplea las siguientes tecnologías principales

- **MySQL,** para el modelado, administración y persistencia de los datos
- **Python,** como lenguaje de programación del servidor
- **Flask:** Framework utilizado para la construcción de la API REST, la definición de rutas y la gestión de peticiones HTTP.
- **Postman:** Fue utilizada como herramienta para la validación y prueba de los endpoints durante el proceso de desarrollo
    - Link al Workspace de Postman: https://mariabelenkanas.postman.co/workspace/ca3bb8cf-33a5-41fe-82f6-3788387c0468
    - Para poder observar con mayor detalle la estructura del Workspace, dirigirse a *Anexo 1.1* - Estructura de colecciones en Workspace de Postman

Frontend

El frontend fue desarrollado utilizando:

- **React Native,** para la construcción de la aplicación móvil (con posibilidad de ser implementado en la web)
- **JavaScript / Typescript** como lenguaje principal del frontend
- **Expo,** como plataforma de desarrollo para facilitar la ejecución, prueba y compilación del proyecto.
- **Expo Router** para la navegación.

Contenedorización con Docker

De forma adicional, para facilitar la ejecución y despliegue del sistema, se implementó un entorno completamente contenedorizado mediante **Docker**, utilizando la herramienta **Docker Desktop** como plataforma de gestión.

Se definieron contenedores independientes para el backend y para la base de datos MySQL, permitiendo un entorno de ejecución aislado, consistente y fácilmente desplegable mediante **docker-compose,** además de conseguir una independencia de la configuración local y un proceso de despliegue más controlado.

Descripción de la solución

La solución implementada corresponde a un sistema integral para la gestión de salas de estudio. Está organizado en módulos independientes desde la parte de frontend que se comunican a través de una API REST creada en backend, lo que favorece la escalabilidad y mantenibilidad del sistema.

Además, el sistema se encuentra gestionado en base a las tablas creadas desde la base de datos, con herramientas como MySQL, la cual luego se conecta con el resto de la API mediante el contenedor de Docker.

Módulos del sistema (Frontend)

La aplicación desarrollada implementa los principales flujos funcionales vinculados al proceso de reserva de salas. Si bien el backend es el componente que contempla las reglas importantes y entidades que conforman la base de datos, el frontend se enfoca en los módulos necesarios para la interacción directa del usuario final.

Los módulos implementados son los siguientes:

- **Módulo de Autenticación:** Permite a los usuarios ingresar al sistema utilizando sus credenciales registradas. Incluye una pantalla de inicio de sesión (página principal) y el envío de credenciales al backend para la validación.
- **Módulos de Edificios y Salas:** Presentan una visión de la estructura de la universidad y permite navegar entre un listado de edificios y luego un listado de salas por edificio, en el que más adelante se presentan los horarios disponibles y las personas habilitadas para poder crear una reserva.
  Las salas presentadas por edificio son las habilitadas por el tipo del usuario que esté visitando la página.
- **Módulo de Reservas:** Está dentro del módulo de sala y es ahí donde yace la lógica creada en backend con respecto a la creación de reservas, además de notificar los supuestos casos de error.
- **Módulo "Mis Reservas":** Muestra al usuario sus reservas activas, pasadas o canceladas según lo brindado por la API desarrollada en backend. Además, permite cancelar reservas dentro de los límites establecidos por el sistema.
- **Módulo "Reservas Generales":** Mientras el módulo anterior es solo visible para los docentes y estudiantes, los administradores por otro lado pueden visualizar todas las reservas hechas por todos los usuarios, como también eliminarlas o cancelarlas.
- **Módulo "Estadísticas":** Exclusivo para aquellos participantes del rol "*admin*". Permite visualizar los reportes generales del sistema de reservas, como por ejemplo la sala más solicitada, los horarios más concurridos, las sanciones activas, entre otros.
- **Módulo "Panel de Control":** Exclusivo para participantes administradores. Permite la implementación de funciones CRUD con respecto a los edificios, salas, facultades, programas académicos y participantes. Con estos últimos, el administrador mediante este módulo se encarga de registrar a nuevos usuarios, con contraseñas creadas por el mismo.

Toda la lógica de verificación de reglas (límites, disponibilidad, etc.) es controlada por el backend; el frontend actúa como interfaz de consumo.

Estructura del Backend

El backend del sistema fue desarrollado utilizando Python junto con el framework Flask, adoptando una arquitectura modular basada en la separación de responsabilidades, siguiendo los principios recomendados en la ingeniería de software. Esta estructura permite mantener un código ordenado, escalable y fácilmente mantenible. A continuación, se detalla cada uno de los componentes principales del proyecto.

1. **Carpeta app/**
   Es el núcleo del backend, donde se encuentran todos los módulos necesarios para la ejecución de la API.
   1.1. **__init__.py:** Es el archivo responsable de la configuración e inicialización de la aplicación Flask. Aquí se definen los parámetros generales y se registran los distintos blueprints que conforman la API
   1.2. **__main__.py:** Es el punto de entrada principal del backend. Desde este archivo es que se ejecuta toda la aplicación.

2. **Carpeta database/**
   Incluye los módulos relacionados con la conexión a MySQL
   2.1. **conexion_db.py:** Define la función de conexión reutilizable para todos los servicios, encapsulando los parámetros de acceso y garantizando consistencia en las operaciones sobre la base de datos.

3. **Carpeta endpoints/**
   Incluye todos los blueprints que agrupan las rutas por funcionalidad.

   Ejemplos de blueprints incluidos:
   - login_bp.py (autenticación)
   - edificio_bp.py (gestión de edificios)
   - facultad_bp.py (gestión de facultades)
   - sala_bp.py (gestión de salas)
   - reserva_bp.py (operaciones sobre reservas)
   - turno_bp.py (administración de turnos)
   - sancion_participante_bp.py (manejo de sanciones)
   - participante_bp.py
   - participante_programa_academico_bp.py
   - programa_academico_bp.py
   - reserva_reportes_bp.py (Almacena las consultas solicitadas en la consigna del proyecto)
   - reserva_participante_bp.py

   Esta organización permite mantener separadas las rutas según su funcionalidad, facilitando pruebas y futuras ampliaciones.

4. **Carpeta services/**
   Contiene la lógica de negocio asociada a cada entidad del sistema.
   Cada archivo ***<entidad>_service.py*** contiene las operaciones necesarias para consultar, insertar, actualizar o eliminar información en la base de datos.

Ejemplos:

- reserva_service.py: lógica de creación, listado y actualización de reservas.
- participante_service.py: gestión de participantes.
- programa_academico_service.py: administración de programas académicos.
- sala_service.py: gestión de salas y tipos de salas.

Esta separación entre *endpoints* (controladores) y *services* (lógica de negocio) aporta claridad al flujo de la aplicación y sigue buenas prácticas de desarrollo.

5. **Archivos principales del backend**
   5.1. **Dockerfile:** Define la imagen del backend en Docker, incluyendo:
      - Instalación del entorno Python
      - Instalación de dependencias
      - Copia de los archivos de la aplicación
      - Comando de ejecución del backend

   5.2. **requirements.txt:** Especifica las dependencias del proyecto. Se usa durante la construcción del contenedor.

6. **Carpeta sql/**
   Contiene los archivos necesarios para la creación e inicialización de la base de datos.
   - 1 - Creación de Base de Datos y Tablas.sql
     - Estructura completa del esquema relacional.
   - 2 - Inserción Tablas.sql
     - Carga inicial de datos relevantes para el funcionamiento.
   - 3 - Consultas.sql
     - Conjunto de consultas útiles durante la fase de verificación y pruebas.
   - schema.sql
     - Versión consolidada del esquema final.

7. **Archivo docker-compose-obligatorio.yml**
   Es el motor responsable de la ejecución del backend y servicio de la base de datos MySQL mediante contenedores Docker.
   Facilita la ejecución del entorno completo con un solo comando:
   docker compose -f docker-compose-obligatorio.yml up --build

8. **Archivos adicionales**
   - .env
     - Variables de entorno (credenciales, configuraciones dinámicas del sistema).
   - config_local.json
     - Configuración local opcional para desarrollo.
   - .gitignore
     - Define archivos y carpetas excluidos del control de versiones.
   - README.md
     - Información general sobre el proyecto, instrucciones de ejecución y notas importantes.

Para mayor detalle de la estructura, ver *Anexo 2.1 – Estructura Backend*

Estructura de la Base de Datos

La base de datos del sistema fue diseñada siguiendo un modelo relacional normalizado, incorporando claves primarias, claves foráneas, restricciones de integridad y relaciones acordes al dominio del problema. Se empleó MySQL por su madurez, soporte de integridad referencial y facilidad de uso en entornos académicos.

A continuación, se detalla la estructura implementada:

- **login** (correo, contraseña)
  Tabla destinada a almacenar credenciales de acceso.
  Se utiliza como base para autenticar participantes.
- **participante** (ci, nombre, apellido, email)
  Representa a cada usuario del sistema: estudiantes y docentes.
  La relación con *login* mediante la clave foránea ***email*** garantiza consistencia entre identidad y credenciales.
- **programa_academico** (id_programa, nombre_programa, id_facultad, tipo [grado, posgrado])
  Incluye una restricción **UNIQUE(nombre_programa, id_facultad)** para evitar duplicados dentro de la misma facultad.
- **participante_programa_academico** (id_alumno_programa, ci_participante, id_programa, rol [alumno, docente, admin])
  Tabla que modela la relación **muchos-a-muchos** entre participantes y programas.
- **facultad** (id_facultad, nombre)
- **sala** (id_sala, nombre_sala, id_edificio, capacidad, tipo_sala [libre, posgrado, docente])
- **edificio** (id_edificio, nombre_edificio, dirección, departamento)
  Se incluye la restricción: **UNIQUE(nombre_sala, id_edificio)**, la cual evita nombres duplicados dentro de un mismo edificio.
- **turno** (id_turno, hora_inicio, hora_fin)
  Define los bloques horarios sobre los cuales se solicitan reservas.
- **reserva** (id_reserva, id_sala, fecha, id_turno, estado [activa, cancelada, sin asistencia, finalizada])
  Representa cada reserva realizada por un usuario.
  Se incluye la restricción **UNIQUE(id_sala, fecha, id_turno, estado)** para así poder evitar que se realice el mismo tipo de reserva para la misma fecha y horario en la misma sala.
- **reserva_participante** (ci_participante, id_reserva, fecha_solicitud_reserva, asistencia [true, false])
  Modela la relación **muchos-a-muchos** entre reservas y participantes, permitiendo registrar asistencia individual.
- **sancion_participante** (id_sancion, ci_participante, fecha_inicio, fecha_fin)
  Permite registrar múltiples sanciones por participante, cada una con su propio período.

Para mayor detalle, se puede observar la estructura de las tablas en el archivo ***./sql/schema.sql*.**

Para observar el tipo de datos almacenados en las entidades, ir a *Anexo 2.2 – Tipos de datos en Tablas SQL -Ejemplos-*

Estructura del Frontend – Conexión con Backend

La aplicación frontend se desarrolla utilizando React Native con Expo y se organiza mediante Expo Router, lo que permite manejar la navegación a través de estructuras de carpetas. La comunicación entre frontend y backend se realiza mediante solicitudes HTTP (fetch) hacia los endpoints REST implementados en Flask.

Principales componentes de la estructura:

- **Pantalla de Login**: Valida credenciales y habilita el acceso a los módulos principales.
- **Navegación principal (app/principal/)**: Contiene el flujo general del sistema tras iniciar sesión. En él se puede decidir entre las opciones de reservas, ver reservas propias, o estadísticas y panel de control (estas últimas solamente habilitadas para los administradores)
- **Gestión de Edificios y Salas**: Permite seleccionar el edificio y luego navegar hacia salas y turnos. Dentro de ellas se permite la gestión de creación de reservas.
- **Pantalla "Mis Reservas"**: Muestra información actualizada obtenida desde el backend, con respecto a reservas activas, canceladas, finalizadas, etc.
- **Pantalla "Reservas Generales"**: Similar al módulo anterior, con la leve diferencia de que es visible sólo para administradores y permite gestionar todas las reservas del sistema.
- **Pantalla "Estadísticas"**: Visible solo para administradores, muestra reportes generales sobre el sistema de reservas.
- **Pantalla "Panel de Control"**: Visible solo para administradores también, permite crear, modificar o eliminar salas, edificios y registrar nuevos participantes.

La comunicación se realiza mediante JSON, respetando los contratos definidos en la API del backend. Todas las validaciones, reglas de negocio y restricciones son aplicadas en el servidor, mientras que el frontend se limita a solicitar, mostrar y enviar información según la interacción del usuario.

Para mayor detalle de la estructura, ver *Anexo 2.3 – Estructura Frontend*

Instrucciones para correr el proyecto

Dado que se implementó la contenedorización en Docker, para poder construir y levantar el mismo proyecto, se deben seguir los siguientes pasos:

PRIMER PASO: Desde el repositorio del ambiente backend:

1. **Construcción y levantamiento del proyecto con Docker**
   Desde la raíz del repositorio (guardado en una carpeta local), ejecutar el siguiente comando:

   ```
   docker compose -f docker-compose-obligatorio.yml up –-build
   ```

   Este comando se encarga de levantar los contenedores que refieren al backend de Flask y el modelo de datos relacional escrito en MySQL.

2. **Establecer conexión en DataGrip**
   Siguiendo los datos brindados en la variable de entorno ***.env*** , desde la ruta ***(+)/ Data Source / MySQL,*** completar:

   - **name:** obligatorio
   - **host:** localhost
   - **port:** 3307
   - **user:** admin
   - **password:** admin123

   Para mayor detalle, ver *Anexo 2.4 – Conexión al modelo de base de datos en DataGrip*

3. **Cargar Base de Datos en el sistema**
   Para poder manejar los datos del sistema, se debe ejecutar manualmente el archivo schema.sql desde la consola de DataGrip con conexión al entorno del obligatorio.

SEGUNDO PASO: Desde el repositorio del ambiente frontend:

Desde la ruta .\FrontEnd-Obligatorio-BD\front-obligatorio, en el cmd ejecutar:

1. npm install → Para instalar las dependencias
2. npx expo start → Para ejecutar la aplicación

En caso de error, instalar las siguientes librerías utilizadas en el desarrollo:

- npx expo install react-native-calendars
- npx expo install @react-native-async-storage/async-storage
- npm install @react-native-picker/picker

Dicha aplicación se podrá ejecutar desde Expo Go (tanto en Android o iOS), emulador de Android (desde Android Studio por ejemplo) o desde el navegador web.

4. **Ingresar al sistema**
   Una vez que se ejecuta el programa podrá observar una pantalla de “login” para poder ingresar a las funcionalidades de la aplicación. Debido a que existen distintos roles y cada uno accede a diferente información, a continuación, dejamos una lista con los usuarios a ingresar para ver las diferentes interfaces según el usuario utilizado para el ingreso:

   - Ingresar con un estudiante de grado
     - Mail: mariabelen.kanas@correo.ucu.edu.uy
     - Contraseña:12345678765

   - Ingresar con un estudiante de posgrado
     - Mail: paz.garcia@correo.ucu.edu.uy
     - Contraseña: 6734vbibgiw

   - Ingresar con un profesor de grado
     - Mail: pedro.silva@correo.ucu.edu.uy
     - Contraseña: ucu8932ohwe

- Ingresar con un estudiante de grado con suspensión vigente
  - Mail: andrea.gonzalez@correo.ucu.edu.uy
  - Contraseña: irry834dj

- Ingresar con el administrador (acceso completo)
  - Mail: admin@correo.ucu.edu.uy
  - Contraseña: admin123

Se podrá observar que dependiendo el usuario con el que se ingrese varía entre salas disponibles, usuarios a los cuales invitar o estadísticas a las que se tiene acceso. En el caso de entrar con el usuario de administrador se puede observar que al entrar al panel de control tiene la posibilidad de crear nuevas salas, edificios y usuarios; en el caso de querer poner a prueba esa funcionalidad, simplemente ingrese los datos requeridos en el formulario.

Consideraciones

Organización del proyecto entre los participantes

La organización del proyecto entre las integrantes del equipo se llevó a cabo mediante una planificación estructurada y el empleo de herramientas colaborativas que permitieron distribuir responsabilidades y asegurar un flujo de trabajo ordenado. Para la gestión de tareas se utilizó Trello, donde se definieron actividades, estados de avance y asignaciones individuales, lo que facilitó la coordinación y el seguimiento progresivo del desarrollo. Paralelamente, se adoptó un modelo de trabajo basado en ramas dentro del sistema de control de versiones, generando ramas específicas para cada funcionalidad o corrección. Esta práctica permitió evitar conflictos de código y asegurar un proceso de integración más seguro y controlado.

Los distintos repositorios del proyecto fueron alojados en GitHub, lo que proporcionó un entorno confiable para el almacenamiento del código fuente, la revisión de cambios, el control de versiones y la colaboración asincrónica entre las participantes. Asimismo, durante el desarrollo del backend, se empleó Postman como herramienta de apoyo para realizar pruebas sobre los endpoints expuestos por la API.

Justificación de cambios y elecciones de programas

La elección de las tecnologías utilizadas en el proyecto respondió a criterios de eficiencia, adecuación al alcance del trabajo y simplicidad en la implementación. Para el backend se optó por Python junto con el microframework Flask, dado que permite desarrollar APIs REST de forma ágil y flexible, ofreciendo una estructura liviana pero suficientemente robusta para un sistema modular como el requerido . Además, su amplia documentación y su uso frecuente en entornos académicos constituyen una ventaja significativa. Esta elección evitó la complejidad innecesaria de frameworks más pesados como Django, cuya arquitectura no resultaba necesaria para el tipo de aplicación a construir.

En comparación con otros frameworks, Flask Python cuenta con un mayor alcance (es rápido de instalar y utilizar), flexibilidad, y un fácil aprendizaje de sus herramientas. (IONOS, 2023). Para su instalación y entendimiento se utilizaron tutoriales especificados en la bibliografía.

En cuanto al frontend, se seleccionó React Native, la cual es una “biblioteca de JavaScript de primera clase para crear interfaces de usuario” según su documentación oficial (Meta Platforms, Inc, 2025), debido a que posibilita el desarrollo de una única base de código compatible con dispositivos iOS, Android e incluso con la web. El framework cuenta con una documentación madura y variada, lo que facilitó el desarrollo y la resolución de problemas durante todo el proceso del proyecto.

Finalmente, se incorporó Docker como herramienta de contenedorización para garantizar la reproducibilidad del entorno de ejecución. Su uso permite que el sistema funcione de manera consistente independientemente del sistema operativo del usuario, evita conflictos de dependencias y facilita el despliegue conjunto del backend y la base de datos MySQL mediante archivos de configuración simples y portables (Weiss, 2024).

Adicionalmente, para poder probar progresivamente el avance del desarrollo en backend, se utilizó Postman. La elección de esta herramienta se debe a que funciona enviando

solicitudes HTTP a un servidor a través de una URL de API (el backend en nuestro caso) y mostrando las respuestas que se recibe. Además, según fue investigado en documentaciones de esta aplicación, “permite trabajar cómodamente con todos los métodos del HTTP, como GET, POST, PUT, PATCH, DELETE” (DesarrolloWeb, 2021), como también permite la creación de colecciones para mayor organización de métodos.

Por otro lado, en relación con el desarrollo de la base de datos del sistema de gestión de reservas de salas de estudio, fue necesario realizar diversas modificaciones respecto a la estructura inicial propuesta en la consigna.

En la propuesta original, algunas tablas utilizaban campos de texto (como nombres de programas, edificios o salas) como claves primarias. Esta práctica no es recomendable dado que los valores textuales pueden presentar duplicaciones, variaciones ortográficas o modificaciones a lo largo del tiempo, afectando así la integridad del modelo. Por este motivo, se añadieron identificadores numéricos autoincrementales como claves primarias en las tablas ***programa_academico***, ***edificio*** y ***sala***. De este modo, las relaciones entre entidades se establecen mediante identificadores únicos y estables.

Asimismo, se revisaron y ajustaron diversas claves foráneas con el fin de asegurar que apuntaran a los identificadores correctos. Por ejemplo, en la tabla ***participante_programa_academico***, la relación fue modificada para referenciar id_programa en lugar de *nombre_programa*, lo cual fortalece la consistencia referencial y evita problemas en caso de cambios en los nombres de los programas académicos.

En la tabla ***programa_academico*** se incorporó la restricción *UNIQUE (nombre_programa, id_facultad)* con el fin de evitar la existencia de dos programas con el mismo nombre dentro de una misma facultad, permitiendo simultáneamente la coexistencia de programas de misma designación en facultades distintas. Además, se creó el rol de admin para más adelante desde el frontend poner restricciones que permitan gestiones más generales y robustas con respecto a los datos cargados.

En el caso de la tabla ***edificio***, se añadió una restricción de unicidad sobre el nombre del edificio, garantizando que no existan dos edificios registrados bajo la misma denominación.

Para la tabla ***sala***, se corrigió la clave foránea correspondiente, asegurando que *id_edificio* referencie adecuadamente a ***edificio****(id_edificio)*. Además, de forma análoga al caso de edificios, se agregó la restricción *UNIQUE (nombre_sala, id_edificio)* a fin de evitar el mismo nombre de sala dentro del mismo edificio.

En lo que corresponde a la tabla ***reserva***, se eliminaron las referencias compuestas que no aportaban valor al modelo, manteniendo únicamente las relaciones directas con ***sala****(id_sala)* y ***turno****(id_turno)*. Originalmente, la tabla **reserva** incluía tanto *id_sala* como *id_edificio*, pero este último puede deducirse a través de la tabla ***sala***, ya que esta contiene la referencia correspondiente al edificio.

En las tablas de relación, se mantuvieron claves primarias compuestas para garantizar unicidad sin necesidad de identificadores adicionales. En ***reserva_participante***, cuya clave primaria está compuesta por *(ci_participante, id_reserva)* se asegura que un mismo participante no pueda tener dos registros iguales para la misma reserva; además, en el atributo *asistencia* se estableció DEFAULT FALSE para evitar nulos. En

sancion_participante se permite registrar múltiples sanciones asociadas a un mismo participante en distintos períodos mediante la clave primaria autoincremental *id_sancion*.

Como resultado de las modificaciones mencionadas, la estructura final del esquema quedó conformada por las siguientes tablas:

- login (correo, contraseña)
- participante (ci, nombre, apellido, email)
- programa_academico (id_programa, nombre_programa, id_facultad, tipo [grado, posgrado])
- participante_programa_academico (id_alumno_programa, ci_participante, id_programa, rol [alumno, docente, admin])
- facultad (id_facultad, nombre)
- sala (id_sala, nombre_sala, id_edificio, capacidad, tipo_sala [libre, posgrado, docente])
- edificio (id_edificio, nombre_edificio, dirección, departamento)
- turno (id_turno, hora_inicio, hora_fin)
- reserva (id_reserva, id_sala, fecha, id_turno, estado [activa, cancelada, sin asistencia, finalizada])
- reserva_participante (ci_participante, id_reserva, fecha_solicitud_reserva, asistencia [true, false])
- sancion_participante (id_sancion, ci_participante, fecha_inicio, fecha_fin)

Oportunidades de mejora

A partir del desarrollo realizado, se identificaron diversas oportunidades de mejora y ampliación que permitirían fortalecer la funcionalidad, usabilidad y escalabilidad del sistema. Estas oportunidades no llegaron a realizarse por falta de tiempo o porque no se consideraban esenciales para el funcionamiento correcto del sistema.

En primer lugar, sería recomendable implementar un sistema de autenticación más robusto basado en tokens (por ejemplo, JWT) que permita manejar sesiones de manera segura, así como roles mayormente extendidos y robustos (además de administradores, coordinadores académicos, entre otros).

Otra oportunidad relevante consiste en mejorar la experiencia de usuario mediante funcionalidades adicionales en el frontend, tales como notificaciones en tiempo real para alertar sobre confirmaciones de reserva, recordatorios de asistencia o avisos de sanción.

Adicionalmente, en términos del código de frontend, se tuvo en cuenta agregar una barra para las distintas pestañas para asegurar una mejor navegación, pero como no era esencial en términos de backend no fue implementado. Adicionalmente, notamos archivos que terminaron siendo demasiados extensos, por lo que quizás con mayor tiempo de desarrollo, se hubiese podido definir mejor los componentes principales para así poder dividirlos según sus dependencias.

A nivel de conexión entre los distintos entornos, hubiese sido una muy buena herramienta haber podido conectar ambos ambientes mediante un solo contenedor de Docker. Actualmente, pudimos realizar mediante un contenedor la ejecución de la app realizada en Flask en conjunto con el modelo de la base de datos. El frontend se ejecuta manualmente, pero haberlo incluido en el contenedor hubiese sido una mejor práctica. Como último

punto, otra práctica más efectiva hubiese sido la ejecución del script schema.sql automático, para así evitar que aquella persona ajena al código tenga que correrlo manualmente para poder usar la aplicación.

Estas oportunidades representan líneas de trabajo que ampliarían significativamente el alcance del sistema y lo acercarían a un entorno productivo real, generando mayor valor para usuarios y administradores.

Conclusiones

El desarrollo del sistema permitió digitalizar de forma integral la gestión de reservas de salas de estudio, mejorando significativamente la trazabilidad del proceso, el control de asistencias y la aplicación automática de las reglas del dominio. La separación clara entre backend, frontend y base de datos favoreció una arquitectura modular, escalable y mantenible, facilitando tanto el desarrollo como el trabajo colaborativo.

La utilización de Docker garantizó un entorno reproducible y consistente, eliminando problemas de compatibilidad y asegurando la portabilidad del proyecto. Por su parte, el modelo de datos final se diseñó siguiendo principios de normalización y buenas prácticas, resultando en un esquema sólido, coherente y alineado con los requisitos funcionales establecidos.

Si bien el sistema cumple con todos los objetivos principales planteados en la consigna, existen oportunidades de mejora futura, tales como la incorporación de notificaciones en tiempo real, ampliación de reportes administrativos o integración con servicios institucionales. No obstante, el resultado obtenido constituye una solución robusta y funcional que responde adecuadamente a las necesidades actuales del dominio.

Bitácora

Tal como se mencionó previamente, durante el desarrollo del proyecto se mantuvo una bitácora semanal como herramienta de seguimiento y documentación continua del proceso. La utilización de una bitácora permitió registrar de manera sistemática las tareas realizadas, las dificultades técnicas encontradas, las decisiones de diseño adoptadas y los aprendizajes obtenidos en cada etapa del trabajo. Este registro resultó fundamental para garantizar la trazabilidad del proceso de desarrollo, facilitar la coordinación entre los integrantes del equipo y permitir la reflexión crítica sobre el avance del proyecto.

El uso de la bitácora también contribuyó a mejorar la planificación, dado que permitió identificar con claridad los bloqueos surgidos, reasignar tareas cuando fue necesario y documentar soluciones aplicadas que podrían resultar útiles en etapas posteriores o en futuras implementaciones. Desde una perspectiva académica, la bitácora constituyó un insumo valioso para evidenciar el proceso de trabajo y justificar decisiones técnicas tomadas a lo largo del proyecto.

El documento completo de la bitácora se encuentra disponible en el repositorio del backend, en la carpeta ***./documentacion/***, bajo el nombre *"Blanco, Gonzalez, Kanas - Bitácora.pdf"*.

Por otro lado, la organización del proyecto también se complementó con el uso de Trello, plataforma en la cual se administraron tareas, prioridades y estados de avance. El acceso al tablero utilizado puede consultarse en: Proyecto Trello - Obligatorio Base de Datos I

Bibliografía

Aris, S. (31 de Octubre de 2025). *Docker tutorial: A complete guide to running containers*. Obtenido de Hostinger: https://www.hostinger.com/tutorials/docker-tutorial?utm_campaign=Generic-Tutorials-DSA-t2|NT:Se|Lang:EN|LO:Other-LATAM-t1&utm_medium=ppc&gad_source=1&gad_campaignid=23205165764&gbraid=0AAAADMy-hYufYAdBBZBHdDhEOdWOrjGQ&gclid=CjwKCAiA24XJBhBXEiwAXElO38Mfll

DesarrolloWeb. (19 de Octubre de 2021). *Como usar Postman para probar nuestras APIs*. Obtenido de DesarrolloWeb: https://desarrolloweb.com/articulos/como-usar-postman-probar-api

Flask. (2010). *Flask QuickStart*. Obtenido de Flask: https://flask.palletsprojects.com/en/stable/quickstart/

IONOS. (03 de Enero de 2023). *¿Qué es Flask Python? Un breve tutorial sobre este microframework*. Obtenido de IONOS: https://www.ionos.com/es-us/digitalguide/paginas-web/desarrollo-web/flask/

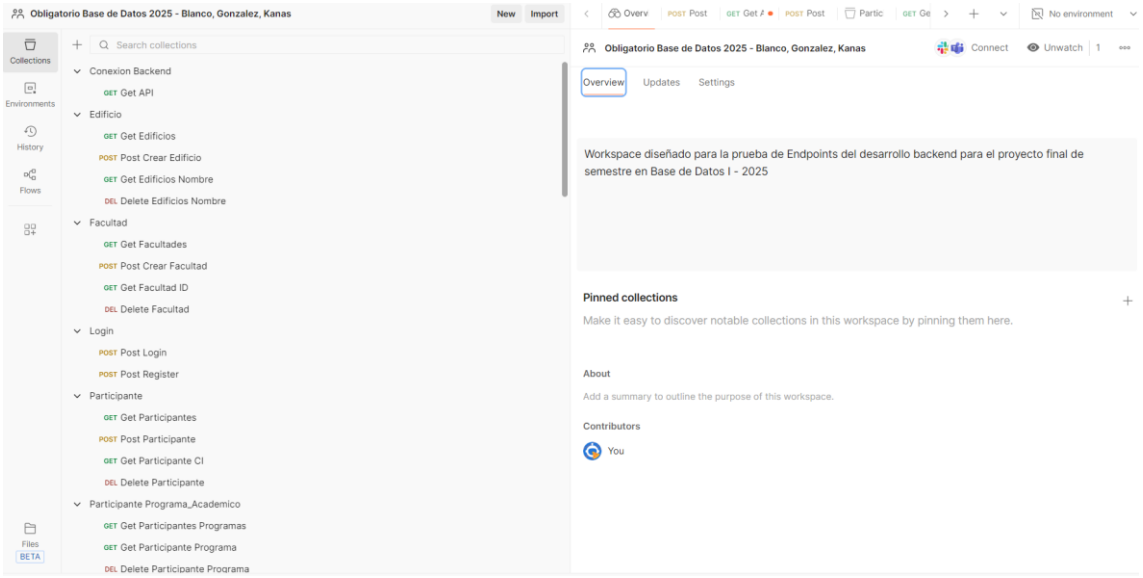
Meta Platforms, Inc. (2025). *React Native*. Obtenido de React Native: https://reactnative.dev/

Red Hat. (2023 de Enero de 2023). *¿Qué es Docker y cómo funciona?* Obtenido de Red Hat: https://www.redhat.com/es/topics/containers/what-is-docker

Sharir, J. (24 de Marzo de 2020). *What is Postman? How to Use Postman to Test APIs*. Obtenido de BlazeMeter: https://www.blazemeter.com/blog/how-use-postman-test-apis#what-is-postman

Weiss, T. R. (30 de Septiembre de 2024). *Explorando Docker para DevOps: qué es y cómo funciona*. Obtenido de Docker: https://www.docker.com/blog/docker-for-devops/

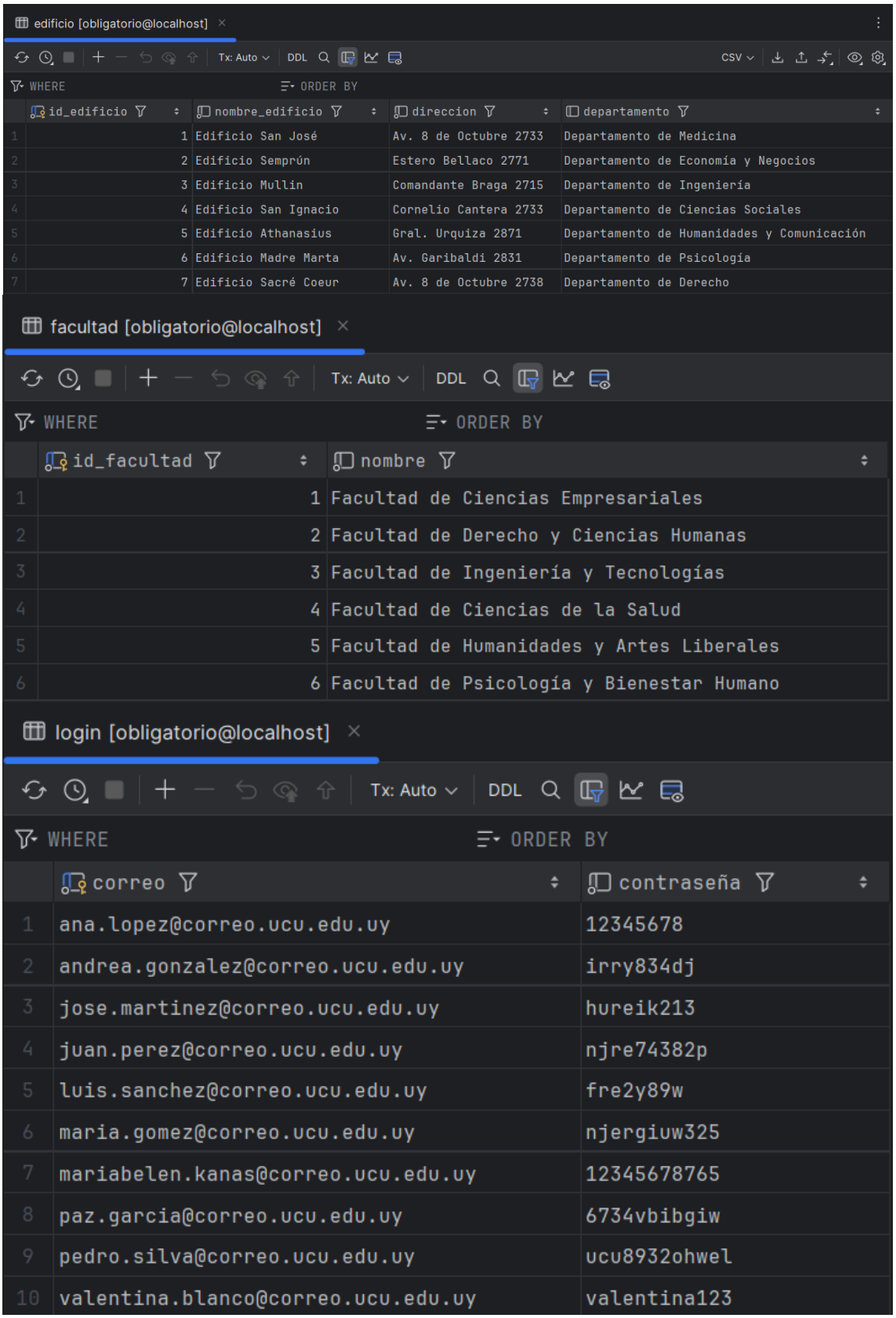
Anexos



1.1 Estructura de colecciones en Workspace de Postman

```
backend_flask/
│
├── app/
│   │
│   ├── database/
│   │   ├── __init__.py
│   │   └── conexion_db.py
│   │
│   ├── endpoints/
│   │   ├── __init__.py
│   │   ├── edificio_bp.py
│   │   ├── facultad_bp.py
│   │   ├── login_bp.py
│   │   ├── participante_bp.py
│   │   ├── participante_programa_academico_bp.py
│   │   ├── programa_academico_bp.py
│   │   ├── reserva_bp.py
│   │   ├── reserva_participante_bp.py
│   │   ├── reserva_reportes_bp.py
│   │   ├── sala_bp.py
│   │   ├── sancion_participante_bp.py
│   │   └── turno_bp.py
│   │
│   ├── services/
│   │   ├── __init__.py
│   │   ├── edificio_service.py
│   │   ├── facultad_service.py
│   │   ├── login_service.py
│   │   ├── participante_service.py
│   │   ├── participante_programa_academico_service.py
│   │   ├── programa_academico_service.py
│   │   ├── reserva_service.py
│   │   ├── reserva_participante_service.py
│   │   ├── reserva_reportes_service.py
│   │   ├── sala_service.py
│   │   ├── sancion_participante_service.py
│   │   └── turno_service.py
│   │
│   ├── __init__.py           # configuración de la aplicación Flask
│   └── __main__.py           # punto de entrada principal: `python -m app`
│
├── Dockerfile
├── requirements.txt
│
├── documentacion/           # archivos del informe y material de apoyo
│   ├── TrabajoObligatorio-V.Blanco,A.Gonzalez,B.Kanas.pdf
│   └── Bitacora.pdf
│
├── sql/
│   ├── 1- Creación de Base de Datos y Tablas.sql
│   ├── 2- Inserción Tablas.sql
│   ├── 3- Consultas.sql
│   └── schema.sql
│
├── .env
├── .gitignore
├── config_local.json
├── docker-compose-obligatorio.yml
├── package-lock.json
└── README.md
```

2.1 Estructura Backend



edificio [obligatorio@localhost]

Tx: Auto | DDL | CSV

WHERE | ORDER BY

	id_edificio	nombre_edificio	direccion	departamento
1	1	Edificio San José	Av. 8 de Octubre 2733	Departamento de Medicina
2	2	Edificio Semprún	Estero Bellaco 2771	Departamento de Economía y Negocios
3	3	Edificio Mullin	Comandante Braga 2715	Departamento de Ingeniería
4	4	Edificio San Ignacio	Cornelio Cantera 2733	Departamento de Ciencias Sociales
5	5	Edificio Athanasius	Gral. Urquiza 2871	Departamento de Humanidades y Comunicación
6	6	Edificio Madre Marta	Av. Garibaldi 2831	Departamento de Psicología
7	7	Edificio Sacré Coeur	Av. 8 de Octubre 2738	Departamento de Derecho

facultad [obligatorio@localhost]

Tx: Auto | DDL

WHERE | ORDER BY

	id_facultad	nombre
1	1	Facultad de Ciencias Empresariales
2	2	Facultad de Derecho y Ciencias Humanas
3	3	Facultad de Ingeniería y Tecnologías
4	4	Facultad de Ciencias de la Salud
5	5	Facultad de Humanidades y Artes Liberales
6	6	Facultad de Psicología y Bienestar Humano

login [obligatorio@localhost]

Tx: Auto | DDL

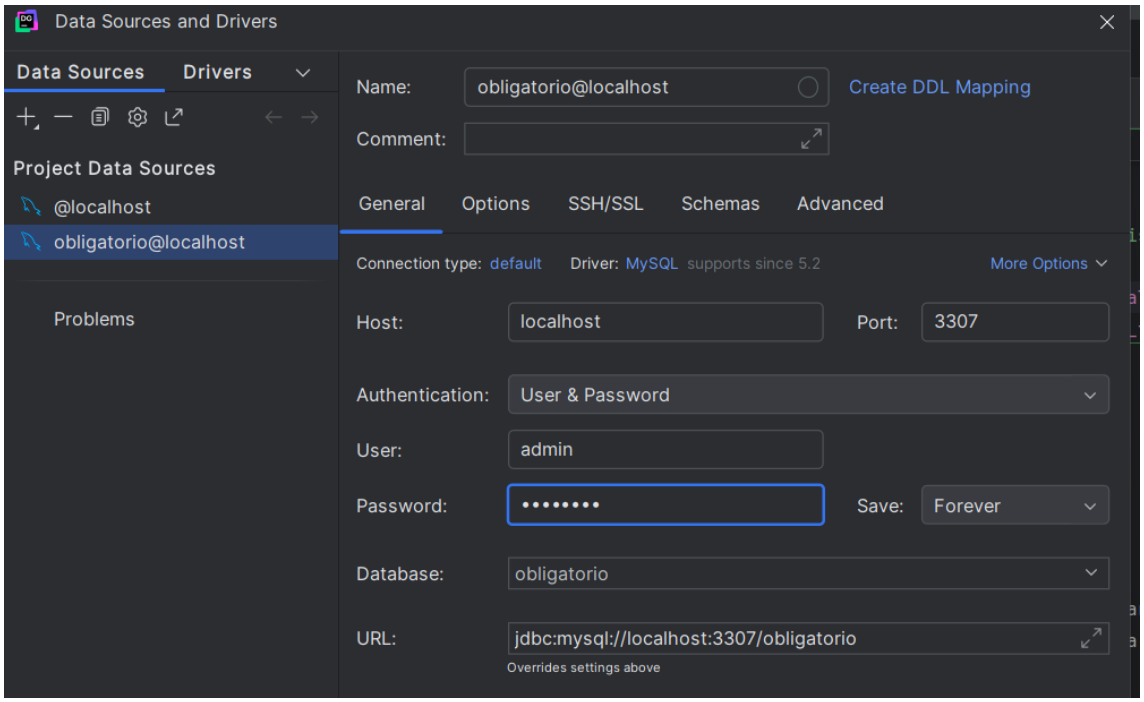
WHERE | ORDER BY

	correo	contraseña
1	ana.lopez@correo.ucu.edu.uy	12345678
2	andrea.gonzalez@correo.ucu.edu.uy	irry834dj
3	jose.martinez@correo.ucu.edu.uy	hureik213
4	juan.perez@correo.ucu.edu.uy	njre74382p
5	luis.sanchez@correo.ucu.edu.uy	fre2y89w
6	maria.gomez@correo.ucu.edu.uy	njergiuw325
7	mariabelen.kanas@correo.ucu.edu.uy	12345678765
8	paz.garcia@correo.ucu.edu.uy	6734vbibgiw
9	pedro.silva@correo.ucu.edu.uy	ucu8932ohwel
10	valentina.blanco@correo.ucu.edu.uy	valentina123

2.2 Tipos de datos en Tablas SQL -Ejemplos-

```
FRONTEND-OBLIGATORIO-BD/
└── front-obligatorio/
    ├── .expo/
    ├── .vscode/
    ├── app/
    │   ├── (tabs)/
    │   │   ├── _layout.jsx
    │   │   ├── estadisticas.jsx
    │   │   ├── index.jsx
    │   │   ├── login.tsx
    │   │   ├── misReservas.jsx
    │   │   ├── panelDeControl.jsx
    │   │   └── reservasGenerales.jsx
    │   │
    │   ├── components/
    │   │   ├── Accordion.jsx
    │   │   ├── ModalConfirmar.tsx
    │   │   ├── ModalConfirmarReserva.tsx
    │   │   └── ModalResultado.tsx
    │   │
    │   ├── principal/
    │   │   ├── edificio/
    │   │   │   ├── sala/
    │   │   │   │   ├── _layout.tsx
    │   │   │   │   └── [sal].tsx
    │   │   │   ├── [edi].tsx
    │   │   │   └── index.tsx
    │   │   │
    │   │   ├── _layout.tsx
    │   │   ├── estadisticas.tsx
    │   │   ├── index.tsx
    │   │   ├── misReservas.tsx
    │   │   ├── panelDeControl.tsx
    │   │   └── reservasGenerales.tsx
    │   │
    │   ├── types/
    │   │
    │   ├── _layout.tsx
    │   ├── index.tsx
    │   └── login.jsx
    │
    ├── assets/
    ├── node_modules/
    │
    ├── .gitignore
    ├── app.json
    ├── eslint.config.js
    ├── expo-env.d.ts
    ├── package.json
    └── package-lock.json
```

2.3 Estructura Frontend



2.4 Conexión al modelo de base de datos en DataGrip