Obligatorio BD1

Integrantes: Diego De Oliveira, Agustin García y Facundo Piriz.

Introducción:

En el marco del proyecto obligatorio de cierre para Base de Datos 1, se nos encomendó hacer un gestor de salas de estudio para los diferentes edificios de la UCU . El gestor de salas posee funcionalidades diferentes para 3 tipos de usuarios:

- Participante: Alumno/Docente de la UCU que quiere utilizar la sala en grupo, a través de su página puede hacer una reserva para una sala y mandar invitaciones a otros participantes para que se unan a la misma. Además puede reseñar las salas una vez terminada la reserva.
- Administrador: El administrador puede ver acceder a todas las funcionalidades de alta, baja y modificación de salas, reservas, sanciones, reseñas y usuarios. Además pueden acceder a diferentes estadísticas con respecto a las salas y sus reservas.
- Funcionario: El funcionario accede a prácticamente las mismas funciones que el administrador, exceptuando las salas y usuarios.

A continuación relataremos más a fondo sobre su implementación.

Decisiones de implementación:

Backend:

Lenguaje de Programación: Python
Framework: Flask
Conexión SQL: MySQL-connector-python

Para la gestión de la API en backend optamos por Flask, nos pareció el framework de python más eficiente y práctico para llevar a cabo la implementación de la API. También consideramos otras opciones como FastAPI o Django, pero debido a la vasta información que conseguimos de primera mano sobre Flask, por lo tanto, nos resultó la opción idónea para comenzar el proyecto.

Para la conexión con la base de datos utilizamos la librería MySQL-connector-python dado que fue la que usamos anteriormente en prácticos de la asignatura y resultó de gran practicidad para utilizar.

Se decidió usar CORS usando la biblioteca de Flask-CORS para permitir la comunicación entre el frontend y el backend del proyecto. Dado que el frontend desarrollado en React corre en http://localhost:5173 y el backend en Flask corre en http://localhost:5000, el navegador aplica por defecto la política de Same-Origin Policy, la cual bloquea solicitudes entre diferentes orígenes por razones de seguridad. Para evitar este bloqueo y permitir que la aplicación funcione correctamente, configuramos CORS especificando únicamente los orígenes permitidos, garantizando un intercambio seguro entre ambos servicios. Además,

se habilitó el manejo automático de solicitudes OPTIONS (preflight), necesarias para operaciones con encabezados personalizados y autenticación con JWT.

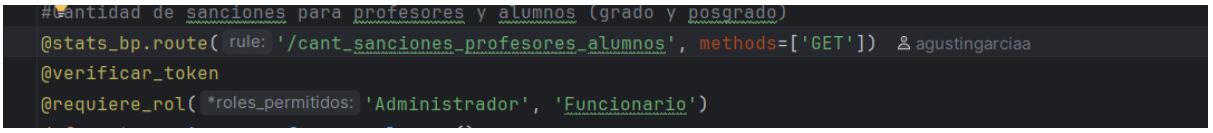
Seguridad y Autenticación:

Se incluyeron grants dentro de la base de datos para darle mayor seguridad a la misma, se establecieron 4 usuarios diferentes para regular a que tabla de la base podía acceder cada uno. Esto fue para tener una capa extra a nivel de la base de datos, independiente del backend.

Se utilizó JWT que significa JSON web token para poder manejar el tiempo de sesión que tiene el usuario sin necesidad de almacenar esa información en el servidor, dado que los datos esenciales del usuario (Como CI, email y rol) se incluyen directamente en el token que es enviado al cliente. Cada vez que el frontend realiza una solicitud éste adjunta el token en los encabezados permitiendo que el backend verifique su validez, su firma y su fecha de expiración. Estos token tienen un tiempo de expiración el cual asegura que el usuario no pueda permanecer conectado indefinidamente sin renovar su token.

También usamos middlewares (utilizando la librería functools de python) los cuales vienen implementados mediante decoraciones de Flask. Estos decoradores permiten interceptar la ejecución de cada endpoint antes de que se procese la solicitud, mejorando tanto la seguridad como la organización interna de la API.

A través de ellos verificamos que el token JWT no esté expirado y que el usuario tenga el rol adecuado para acceder a cada recurso. De esta forma se asegura que las restricciones por rol se apliquen de manera consistente y centralizada, evitando duplicación de código y fortaleciendo el control de acceso en toda la aplicación.



Ejemplo de middleware

Frontend:

Lenguaje de Programación: JavaScript
Framework: React
Librerías: Tailwind CSS, React Icons, React-Router-Dom, React-Toastify, React-Chart

Para la parte de frontend usamos React porque es el framework con el que estamos familiarizados ya que los 3 lo estamos usando en la materia de Desarrollo Web y Mobile.

En cuanto a las librerías usadas en React, decidimos utilizar tailwindcss para definir los estilos del frontend debido a su practicidad y efectividad al momento de diseñar con css. react-router-dom para poder definir las diferentes páginas y rutas dentro del frontend, React icons como un recurso para tener iconos extra para el proyecto, react-toastify para poder hacer alertas interactivas para indicar respuestas del backend y por último se utilizó react-charts para poder insertar gráficos en la parte de estadísticas.

Para hacer el Frontend usamos un service_API (llamado api.js que exporta una función llamada apiFetch que actúa como hook de react) el cual realiza un Fetch a los endpoints adecuados del backend para obtener los datos correspondientes a las acciones a realizar. Este se actualiza de manera dinámica, lo que nos facilitó la realización del trabajo, además se pudo.

Docker:

Decidimos usar Docker debido a la recomendación dada en la letra y también ya que vimos esa oportunidad como una buena manera de unificar y contener todo el proyecto, sumado a que ya lo conocíamos por retos previos de la carrera.

Utilizamos docker-compose, dado que era la manera más simple y eficaz de implementarlo, se definió un dockerfile tanto para frontend y backend estableciendo las dependencias a descargar para cada uno entre otras funciones.

Funcionalidad de la página:

Para realizar una reserva, el participante navega a través de distintas pestañas que muestran las opciones disponibles. En la pestaña de “hacer reserva”, se muestran los edificios de la UCU y, dentro de cada uno, todas sus salas. Al seleccionar una sala y hacer clic en “Reservar”, se despliega un modal donde el usuario puede elegir el día y hora en la que desea realizar la reserva, mostrando únicamente los espacios realmente disponibles.

Una vez confirmada la reserva, el sistema ofrece la posibilidad de invitar a otros usuarios, solicitando el correo electrónico de cada invitado. Si el participante decide omitir este paso, puede igualmente enviar invitaciones más tarde desde las secciones “Mis reservas” o “Invitar”, donde también se permite gestionar las invitaciones existentes. Un agregado que decidimos añadir es la pestaña de reseñar en la cual el usuario podrá reseñar y ponerle un puntaje a la sala, junto con un comentario para retroalimentación.

Mejoras implementadas o consideradas en el modelo de datos

Cambios de la base dada en la letra:

- Participante → Usuario:Cambiamos el nombre a Usuario por cuestiones de comodidad. También añadimos el atributo “activo” el cual muestra si el usuario sigue existiendo o no. Y por último añadimos el atributo “rol” el cual dependiendo de cual sea dicho rol, es que el usuario podrá realizar determinadas acciones adicionales. Los roles son:
    - Participante: Ninguna peculiaridad, un usuario que puede reservar sala como cualquier otro
    - Funcionario: Lo mismo que el anterior con el adicional de registrar quienes ingresan o no a las salas.
    - Administrador: Es quien tiene todos los roles de ABM (alta, baja y modificación) de varios campos, esto incluye tantos los pedidos por la letra como algunos extras que añadimos.

- Login: Hicimos que el email referencia al email de “Usuario”, además de agregar un ON UPDATE CASCADE en el mismo de tal forma de que en caso de que se actualice el mail del usuario, la FK que contiene login no se “rompa”, ya que pasaría a estar apuntando a un objeto nulo, por eso vimos la mejor opción (aunque sabemos que es una opción con la que hay que tener cuidado) de usar Cascade para borrar los datos anteriores de login así no se genera conflictos y una vez terminado de actualizarse el usuario, el email de login pasaría a estar apuntando al nuevo mail del usuario.

- Facultad: Sin cambios
- programa_academico → planAcadémico: Hicimos el cambio de nombre porque nos pareció más acorde a lo que se pide.
- participantePrograma Academico: Sin cambios (lo del cascade de la cedula seguramente haya q sacarlo)
- Edificio: Cambiamos el nombre del atributo departamento → campus, porque nos pareció más acorde a como está organizada la UCU.
- sala → salasDeEstudio: El cambio de nombre fue por comodidad. Le añadimos el atributo de “disponible” que básicamente es un booleano que dice si la sala está disponible o no, lo vimos necesario para que quede bien registrado qué salas es posible reservar. También le añadimos el atributo “puntaje” el cual indica el puntaje promedio que tiene la sala, dado por el sistema de reseñas que añadimos.
- turno: Sin cambios
- reserva: Le añadimos el atributo de ci_usuario_principal, el cual es el CI de quien organizó la reserva, sin considerar el CI de los invitados.
- reservaParticipante: Le añadimos 2 atributos.
    - Confirmación: Para los usuarios invitados, hicimos que estos reciban un correo de confirmación ya que si por ejemplo: yo quiero invitar a Facundo, no sería justo que con solo enviar, ya quede registrado que Facundo tenga dicha reserva, entonces para ahorrarnos eso, implementamos este sistema. También nos sirve para diferenciar invitados de no invitados, ya que esta confirmación es un ENUM el cual tiene 3 opciones: Pendiente, el cual es el default, e indica qué el usuario aún no aceptó o rechazó la invitación, Confirmado el cual indica que aceptó la invitación y rechazado que indica que rechazó la invitación. Tanto si la invitación está en estado “pendiente” como “rechazado” el usuario no quedará como perteneciente a la reserva como tal, simplemente quedará en el sistema pero no será considerado para sanciones.
    - resenado (reseñado si se pudiera poner ñ): Un booleano que nos permite identificar si el participante ya hizo una reseña a la sala, ya que cada participante solo puede reseñar cada sala una vez.

    - id_reserva tiene ON UPDATE CASCADE, porque en caso de que la reserva sea eliminada, va a ser eliminada de la tabla de reserva cosa que ocasionará que este id_reserva apunte a un objeto nulo, con el cascade se borra la referencia de esta tabla también y no genera conflicto.
- sancion_participante: Añadimos 2 atributos:

    - id_sancion: Lo vimos más ordenado y lógico a que poner como PK a 3 atributos, de esta manera se puede referenciar más fácil cada sanción
    - motivo: Lo usamos como extra, dependiendo de cual sea el motivo, el participante tendrá más o menos tiempo de sanción dependiendo de la gravedad de la misma.

    - El mínimo de duración de una sanción es de 1 día ya que no consideramos que ningún motivo de la sanción amerite menos de dicha cantidad.

Nota: No hicimos registro de usuarios, porque dado que los administradores ya tienen el ABM (alta, baja, modificación) de usuarios, no vimos necesario que cada usuario tenga que registrarse, sino que simplemente se le da una contraseña default a cada uno y luego pueden cambiarla, además no tiene sentido que un usuario pueda seleccionar información importante tales como sus planes académicos o roles ya que este se podría aprovechar del sistema u obtener prioridad al momento de realizar reservas.

Extras:

- resena(reseña): Añadimos un sistema de reseñas, en donde cada sala pueda ser valorada, cada participante puede valorar cada sala 1 vez. Atributos:
    - id_resena, id_reserva, ci_participante: Datos necesarios para que quede registrado este sistema, id_reserva y ci_participante referencian las tablas anteriores con FK
    - fecha_publicacion: No muy necesario, pero para que quede registrado cuando se hizo la reseña.
    - puntaje_general y descripcion: atributos de cada reseña para que funcione correctamente.
- Stats: No es una tabla de la base de datos, pero si se agregó como ruta en el Backend, este contiene varios endpoints para obtener estadísticas de las salas más usadas, los usuarios que más reservaron, etc.

El resto fueron modificaciones ya mencionadas sobre las tablas dadas para que cumplan ciertas funciones pedidas por la letra.

Instrucciones para usar la API:

Una vez obtenido el link al repo de Github, se tienen dos opciones: clonar el repositorio, o descargarlo como ZIP, esta segunda opción posiblemente sea la mejor ya que al clonar el repositorio de git se instalan más paquetes, los cuales son necesarios para poder subir posibles cambios al repositorio, sin embargo, si no se van a realizar cambios esto último es innecesario. Una vez teniendo el proyecto descargado, es necesario tener instalada la aplicación Docker Desktop (aunque también es posible correr la API sin este, lo mejor es usarlo para mayor eficiencia). Una vez instalado el Docker, se tiene que hacer lo siguiente:

Pasos para ejecutar el proyecto
1. Ejecutar la aplicación
2. Abrir la carpeta con el proyecto en la terminal
3. Ejecutar el comando `docker compose up`
4. Abrir en el navegador (cualquiera): http://localhost:5173/

Acceso a la página

Una vez estando dentro de la API, se necesita ingresar correo y contraseña. Para esto es necesario usar alguno de los ingresados en los *inserts* de la base de datos.
Todos tienen como contraseña: **contraseña_segura**
Y el mail varía dependiendo del usuario. A continuación, ejemplos para cada rol:

Rol	Correo
Administrador	`facundo.piriz@correo.ucu.edu.uy`
Funcionario	`diego.deoliveira@correo.ucu.edu.uy`
Participante común	`agustin.garciab@correo.ucu.edu.uy`

Inicio y cierre de sesión

Una vez iniciado sesión, es posible usar todas las funcionalidades. Para cerrar sesión se puede presionar el botón de “cerrar sesión”

Bibliografía:

Constraint

GeeksforGeeks. (s. f.). *MySQL CHECK constraint.* GeeksforGeeks. https://www.geeksforgeeks.org/plsql/mysql-check-constraint/

Flask

GeeksforGeeks. (s. f.). *Python Flask environment specific configurations.* GeeksforGeeks. https://www.geeksforgeeks.org/python/flask-environment-specific-configurations/

.Env y Docker

Traversy Media. (2023, 6 de abril). *Docker Secrets and Environment Variables Explained* [Video]. YouTube. https://www.youtube.com/watch?v=DM65_JyGxCo

Docker Inc. (s. f.). *Use Docker Compose to manage your secrets.* Docker Docs. https://docs.docker.com/compose/how-tos/use-secrets/

PyJWT

GeeksforGeeks. (s. f.). *Using JWT for user authentication in Flask.* GeeksforGeeks. https://www.geeksforgeeks.org/python/using-jwt-for-user-authentication-in-flask/

freeCodeCamp. (2023, 12 de junio). *JWT Authentication in Flask.* freeCodeCamp. https://www.freecodecamp.org/news/jwt-authentication-in-flask/

Flask y Cors:
https://medium.com/@ernestocullen/cors-7b3243577593
https://flask-cors.readthedocs.io/en/v1.1/

Decorators (middlewares)

Pallets Projects. (s. f.). *View decorators*. Flask Documentation. https://flask.palletsprojects.com/en/stable/patterns/viewdecorators/