Obligatorio

Decisiones de implementación

Para la gestión de la API en backend usamos flask porque 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, pero no nos llegó a convencer del todo y además había más información de Flask a la que pudiéramos acceder.

Para la parte de frontend usamos React porque es el framework que mejor sabemos usar, 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.

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

Mejoras implementadas o consideradas en el modelo de datos

Cambios de la base dada en la letra:

- Login: No realizamos cambios

- Participante → Usuario: Cambiamos el nombre a Usuario por cuestiones de comodidad y 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.

- 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
- edificio: Cambiamos el nombre del atributo departamento → campus, porque nos pareció más acorde a la 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.
- turno: Sin cambios
- reserva: Le añadimos el atributo de ci_usuario_principal, el cual nos permite diferenciar el ci de quien hizo la reserva 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 inicia en false (menos para quien realiza la reserva), de esta manera se puede traer con una consulta, todos los mails de los invitados.
    - 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.
- 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.

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.

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

Bibliografía:

Delimiter

Stack Overflow. (2009). *What does delimiter do in a trigger?* Stack Overflow. https://stackoverflow.com/questions/1346637/what-does-delimiter-do-in-a-trigger

GeeksforGeeks. (s. f.). *Delimiters in SQL.* GeeksforGeeks. https://www.geeksforgeeks.org/dbms/delimiters-in-sql/

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/