BITÁCORA DEL PROYECTO — Obligatorio BDD

1. Definición del modelo y decisiones iniciales

Se analizaron los requerimientos de la consigna, identificando entidades principales como Facultad, Programa, Participante, Sala, Reserva, Turnos y Edificio. Se seleccionó Python con Flask como backend por su flexibilidad, MySQL como base de datos y HTML + Jinja2 + Flask-WTF como frontend. La consigna prohibía el uso de ORM, por lo que se decidió implementar acceso directo mediante mysql.connector.

2. Arquitectura general, dtypes y ObjectCreator

Se diseñó el diccionario control.dtypes que contiene todas las clases, atributos y dominios limitados y no limitados. Esto permitió crear ObjectCreator, la función central del proyecto, encargada de ordenar argumentos, validar dominios, normalizar strings y crear objetos antes de insertarlos en la base. Se eligió centralizar las inserciones en los métodos save() de cada clase, evitando duplicación y mejorando la seguridad. Se definió también un usuario restringido appuser para evitar operaciones peligrosas, dejando a root solo la creación de estructura.

3. Base de datos, inserciones maestras y validaciones

Se construyó todo el script SQL a mano, con claves foráneas, restricciones de unicidad y columnas obligatorias. Se cargaron datos maestros suficientes para la defensa. Se implementaron validaciones en todas las capas: WTForms, control.dtypes, lógica en control.py y restricciones SQL. La normalización evitó problemas como espacios finales, cédulas inválidas, correos mal formateados o valores fuera de dominio.

4. Endpoints, lógica de negocio y frontend

Se estructuraron los endpoints en Blueprints para mantener legibilidad. Todos los POST, excepto las inserciones, se manejan en endpoints; las inserciones se ejecutan exclusivamente desde las clases. Se implementaron formularios dinámicos con WTForms y plantillas HTML con Jinja2, evitando exponer información sensible en errores y reforzando la seguridad mediante CSRF Token.

5. Procesos de prueba, debugging y seguridad

Se probaron todas las inserciones con scripts y se controlaron errores provenientes de dominios, normalización y dependencias entre entidades. Se reemplazaron todas las concatenaciones de strings por parámetros %s para evitar SQL injection. También se ajustó el manejo de excepciones para no revelar estructuras internas del backend.

6. Documentación, GitHub y Dockerización

Se elaboró el README con las instrucciones completas para correr la aplicación localmente. Se preparó la bitácora y bibliografía. Se realizaron pruebas de dockerización mediante Dockerfile y docker-compose, incluyendo servicios de app, base de datos y, opcionalmente, phpMyAdmin. Se

organizaron las rutas, volúmenes y variables de entorno para permitir la ejecución del proyecto en contenedores.