Informe Tercer Laboratorio – API RESTful Con PostreSQL Y Flask

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Integrantes:

Arley Santiago Álvarez Ortiz - 20241020008

Juan Esteban Rodriguez Camacho – 202401020029

Samuel Garzón Gonzales – 202401020042

Fecha:

24 de junio de 2025

1. Introducción:

El desarrollo de la práctica tuvo como principal objetivo apropiar los conocimientos adquiridos previamente en sesiones anteriores de clase, conceptos como API RESTful, BD relacionales gestionadas a partir del SGBD PostgreSQL, la aplicación del ORM SqlAlchemy y el uso del framework Flask para la gestión de sockets en el desarrollo de aplicaciones orientadas al uso Web y su conexión con la gestión de bases de datos.

A partir de ello, se buscó el fortalecimiento de competencias en el desarrollo de sistemas backend modernos que ofrezcan aplicacion al desarrollo de tareas CRUD (Create, Read, Update, Delete) en sistemas de gestión enfocados, en este caso, a la gestión de tareas de manera sencilla, conectando la información provista por el usuario con una base de datos que almacene y gestione la información.

2. Descripción de la solución:

Partiendo de lo solicitado en el laboratorio, podemos englobar el desarrollo de la API de la siguiente manera:

1. Crear carpeta tareasdb desde el cmd y abrir VSCode para el trabajo en cuestión.
2. Creación y activación de un venv, posterior a ello, instalar todas las dependencias necesarias para el laboratorio, finalmente, en terminal realizar un pip freeze para almacenar las librerías trabajadas en un archivo requirements.txt.
3. Abrir psql (Gestión de BD PostgreSQL) y crear la base de datos a partir del comando CREATE DATABASE tareasdb.
4. Implementación de código base provisto por el profesor en guía de laboratorio en un archivo python app.py, aplicando modificaciones necesarias para hacer funcional la conexión a la bd.
5. Correr app.py y abrir el servidor local provisto por Flask.
6. Abrir Postman y ejecutar pruebas necesarias para verificar correcto funcionamiento de la API.
7. Modificación de app.py para incluir las 3 routes solicitadas como actividad adicional en el laboratorio.
8. Ejecutar pruebas en Postman de la API y verificación a partir de las routes en el servidor local.

A partir de ello, la API que gestionará las tareas se compondrá de los siguientes componentes:

- Estructura del código: El código está contenido en un único archivo `app.py`. Se realiza la configuración de la base de datos PostgreSQL mediante SQLAlchemy.
- Se define un modelo de datos llamado `Tarea` que representa cada tarea con un `id`, `nombre` y un booleano `hecha`.
- Diseño de rutas: Se implementaron las siguientes rutas para la aplicación de CRUD basica:
  - Método: Ninguno, Ruta: / --> Retorna un mensaje al cliente “API de Tareas”.
  - Método: GET, Ruta: /tareas --> Consulta todas las tareas encontradas y las retorna en formato JSON.
  - Método: POST, Ruta: /tareas --> Crea una nueva tarea y la almacena en la BD.
  - Método: GET, Ruta: /tareas/<id> --> Consulta una tarea por su ID y la retorna en formato JSON.

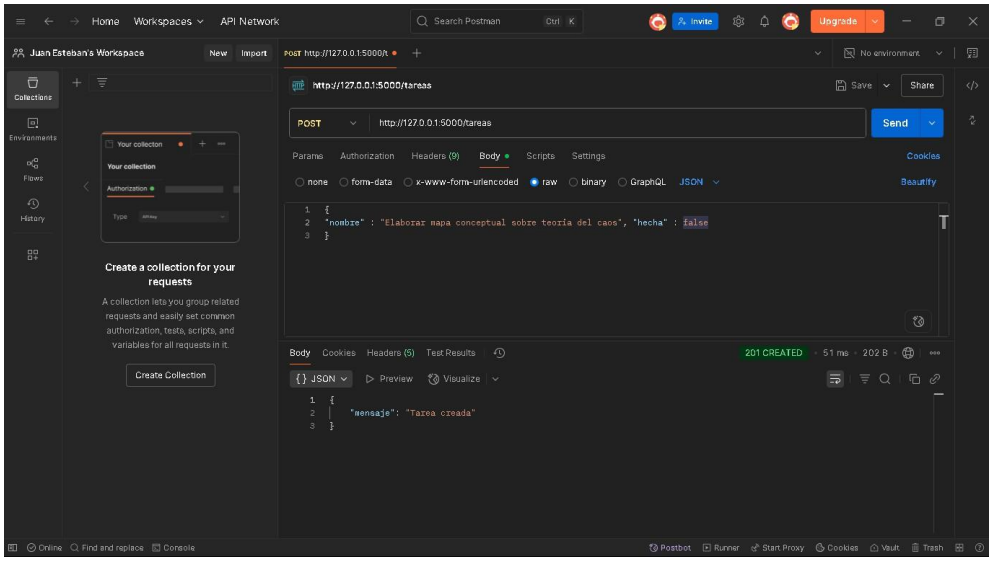
- Método: PUT, Ruta: /tareas/<id> --> Actualiza una tarea existente, genera una consulta a partir del ID y posterior a ello actualiza los datos de la tarea.
- Método: DELETE, Ruta: /tareas/<id> --> Elimina una tarea, genera una consulta a partir del ID y posterior a ello elimina la tarea en cuestión.
- Método: GET, Ruta: /tareas/completadas --> Lista tareas marcadas como hechas, consulta generada a partir de dato booleano 'hecha' y retorno de datos en formato JSON.
- Método: GET, Ruta: /tareas/pendientes --> Lista tareas marcadas como no hechas, consulta generada a partir de dato booleano 'hecha' y retorno de datos en formato JSON.
- Método: GET, Ruta: /tareas/buscar/<palabra> --> Filtra tareas por palabra clave, consulta generada a partir de coincidencia de palabra y retorno de datos en formato JSON.

- Configuración conexión a BD: app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://postgres:postgres@localhost:5432/tareasdb'

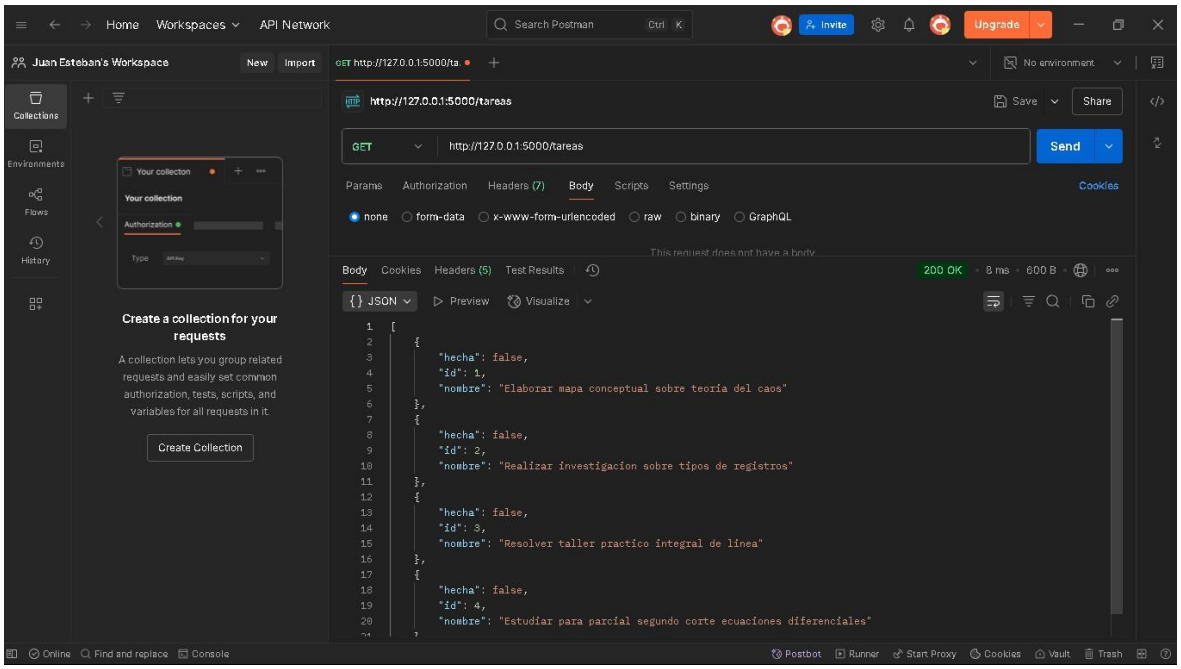
3. Resultados y pruebas:

En el siguiente espacio se registrarán las evidencias correspondientes al correcto funcionamiento de la API implementada con sus respectivas pruebas en postman:

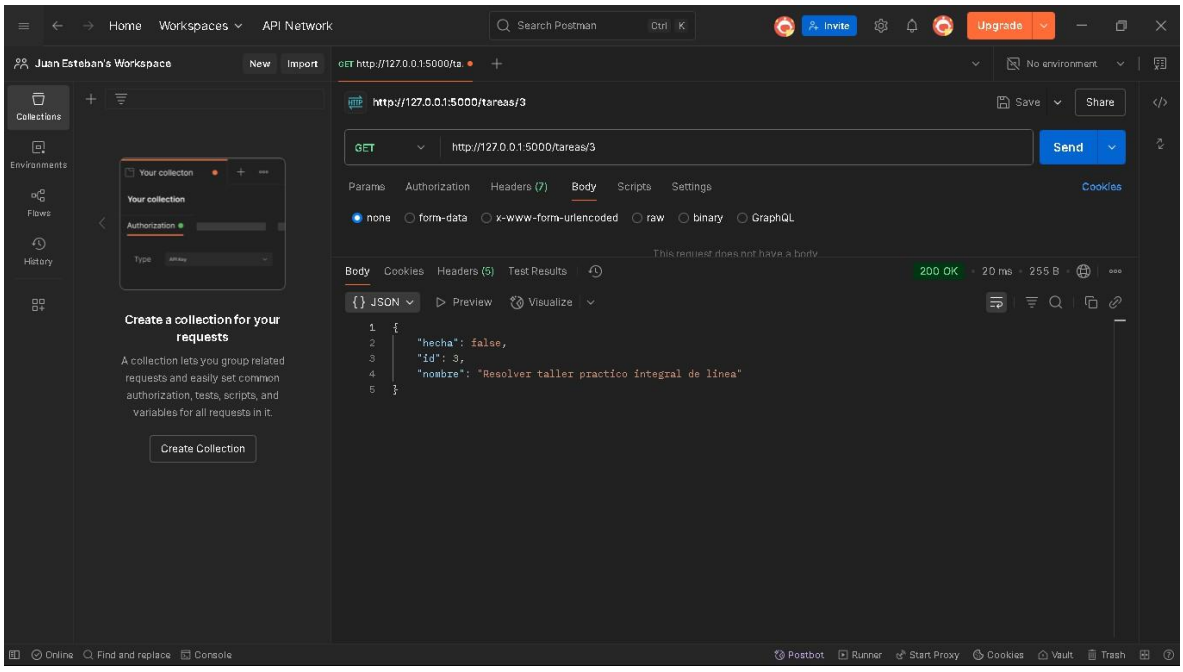
1. Crear tareas:



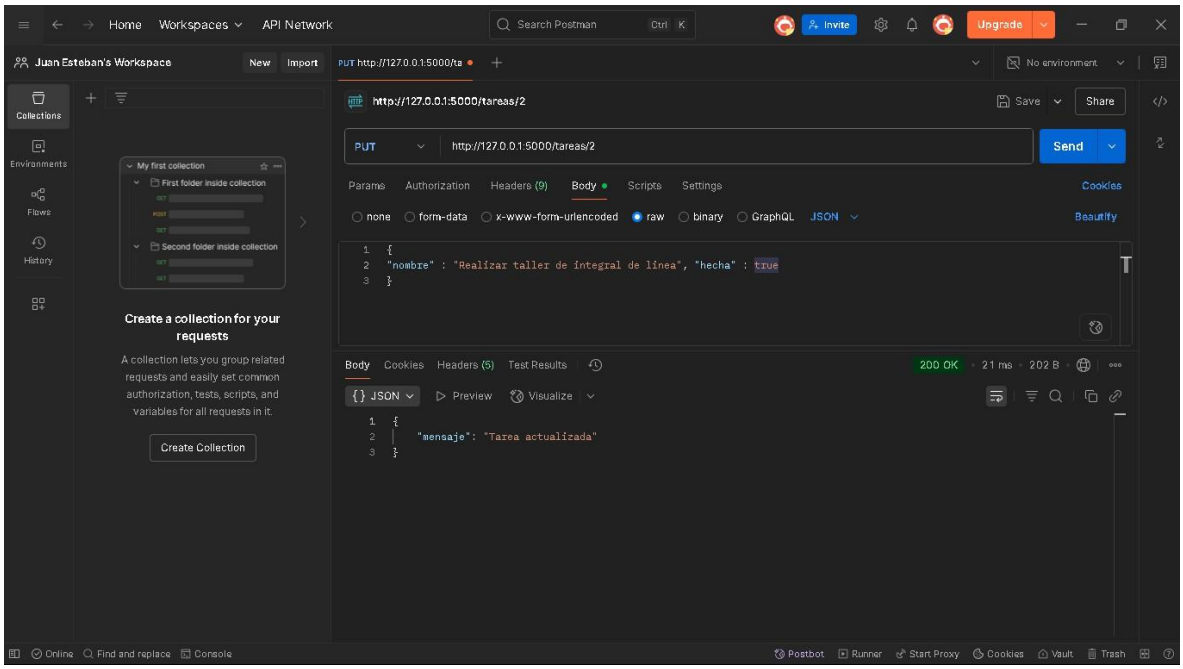
2. Visualizar todas las tareas:



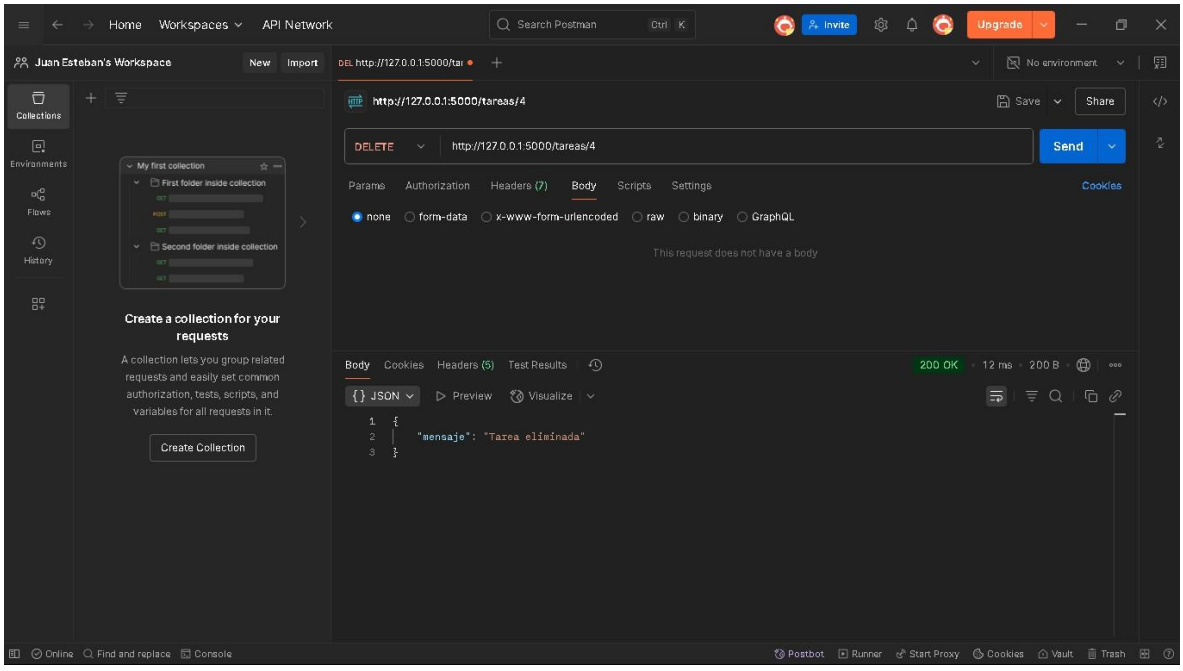
3. Visualizar tarea de acuerdo con su ID:



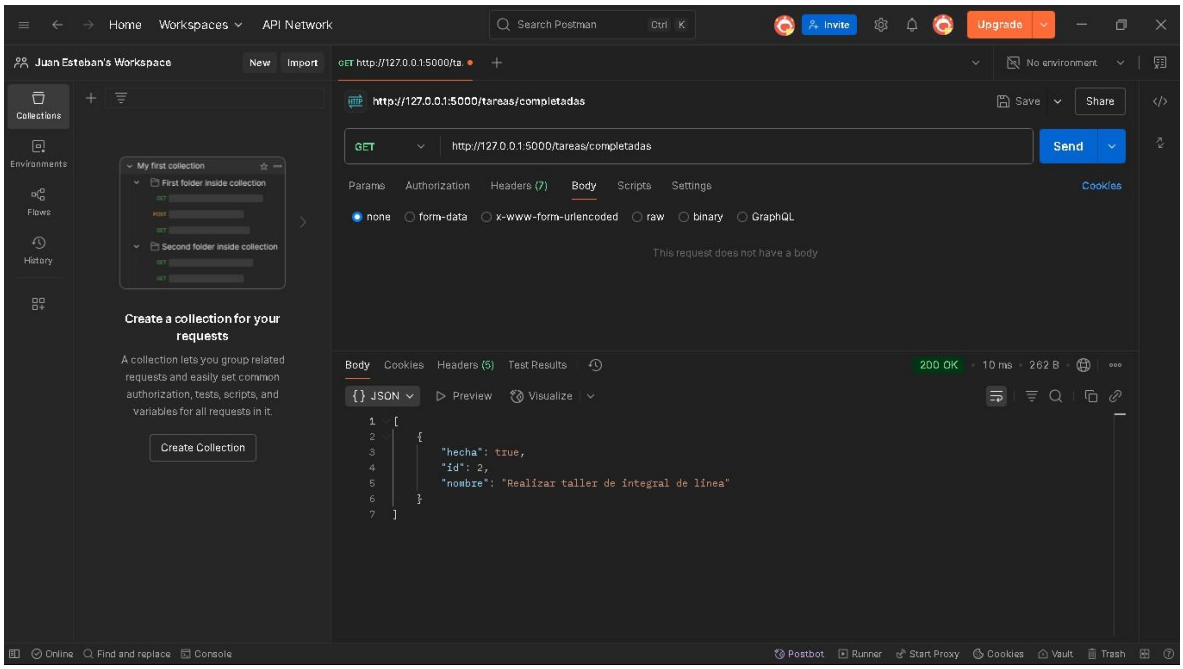
4. Actualizar una tarea:



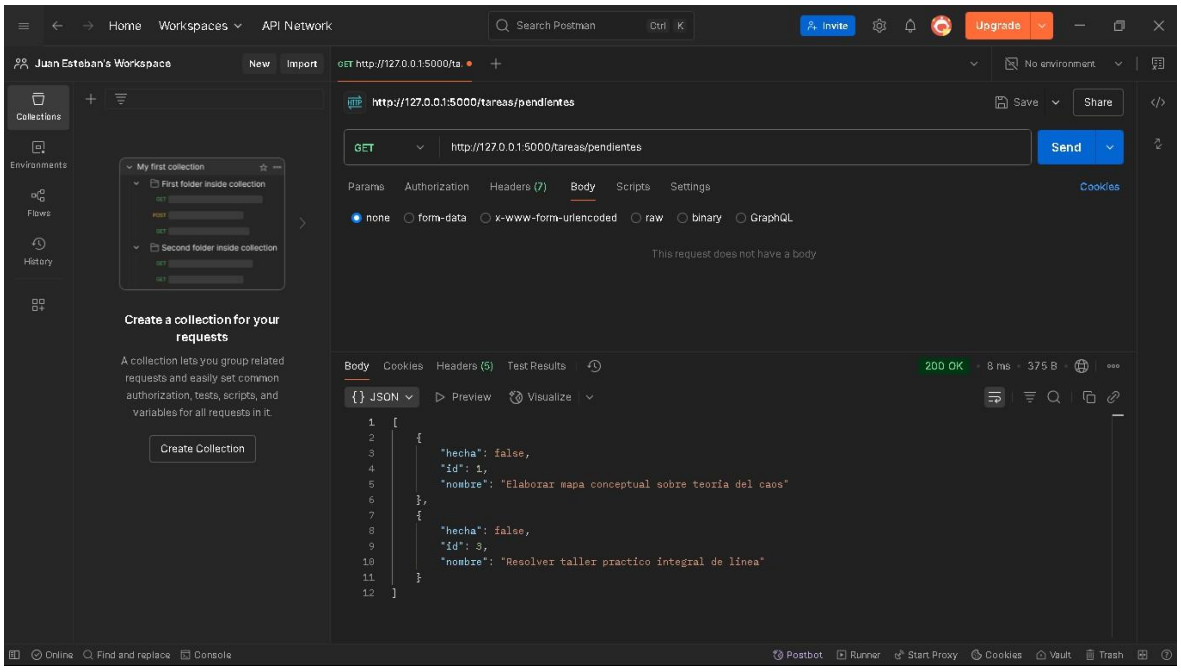
5. Eliminar una tarea:



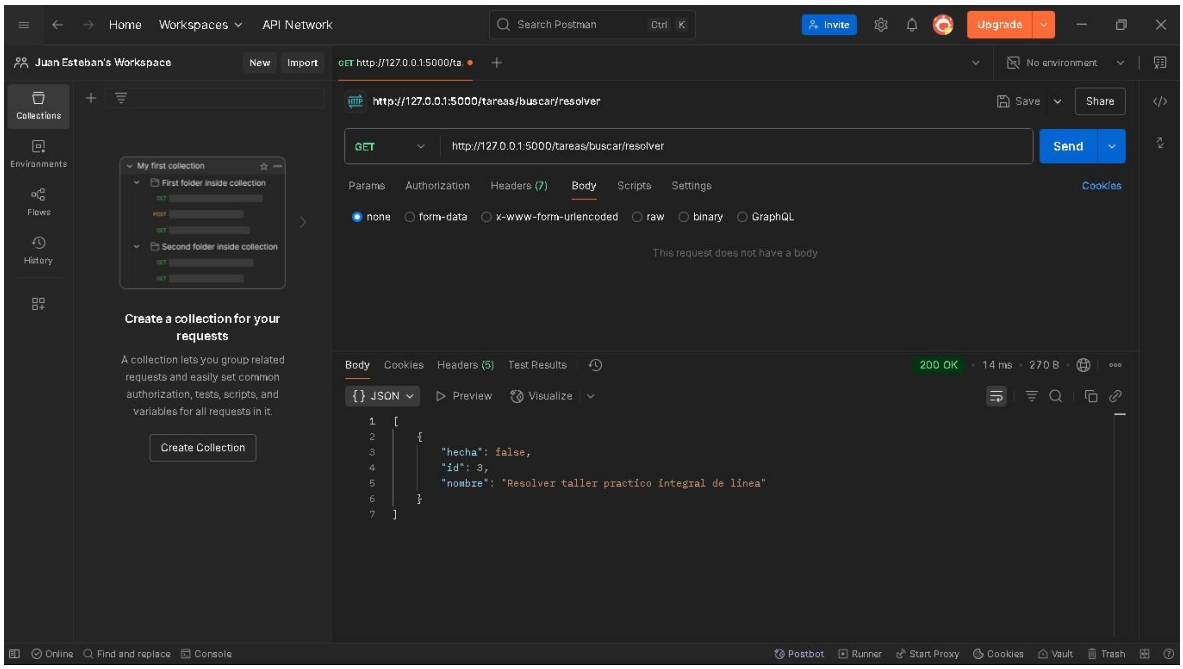
6. Visualizar tareas completadas:



7. Visualizar tareas pendientes:



8. Visualizar tareas por coincidencia de palabra de búsqueda:



4. **Dificultades y soluciones:**

- Error en la creación de tablas: Inicialmente se intentó usar @app.before_first_request, lo cual no funcionó correctamente.
  La solución fue ejecutar db.create_all() dentro de app.app_context().

- Pruebas fallidas con JSON: En algunos casos se olvidó enviar el Content-Type como application/json en Postman.
  Se corrigió configurando correctamente las cabeceras.

- Persistencia de datos: Se requirió reiniciar la conexión a la base de datos cuando PostgreSQL no estaba activo.

5. Conclusiones:

- El desarrollo de esta práctica de laboratorio permitió consolidar los conceptos fundamentales relacionados con la creación de una API RESTful utilizando Flask y PostgreSQL. A través de la implementación de un sistema CRUD para la gestión de tareas, se logró aplicar conocimientos clave como el uso de SQLAlchemy como ORM para interactuar con la base de datos, la configuración de rutas en Flask para manejar solicitudes HTTP, y la serialización de datos en formato JSON para facilitar la comunicación entre el cliente y el servidor.
- La elección de PostgreSQL como sistema de gestión de bases de datos relacionales aseguró la persistencia y consistencia de los datos, mientras que SQLAlchemy simplificó

las operaciones de consulta y manipulación de la base de datos mediante su abstracción de alto nivel. Por otro lado, Flask demostró ser un framework eficiente y flexible para el desarrollo de APIs, permitiendo la definición de rutas y métodos HTTP de manera intuitiva y escalable. Las pruebas realizadas en Postman validaron el correcto funcionamiento de cada endpoint, confirmando que la API cumple con los requisitos de crear, leer, actualizar y eliminar tareas, así como con las funcionalidades adicionales de filtrar tareas por estado (completadas o pendientes) y por palabras clave. Estas pruebas también destacaron la importancia de configurar adecuadamente las cabeceras HTTP, como el Content-Type: application/json, para garantizar una comunicación efectiva entre el cliente y el servidor.

- Entre las dificultades enfrentadas, destacan la configuración inicial de la base de datos y la necesidad de reiniciar conexiones, lo cual reforzó la comprensión sobre la gestión de sesiones y contextos en Flask. Estas experiencias subrayaron la importancia de trabajar dentro de un app_context para operaciones que requieren acceso a la base de datos.

6. Enlace a carpeta:

Finalmente, se incluirá la URL activa del contenido completo del laboratorio:
https://github.com/Ajax296/Laboratorio3_API_Flask.git