Informe de Práctica 9: Desarrollo de REST API con Flask y PostgreSQL

Asignatura: Administración de Bases de Datos (ADBD)

Autor: Guillermo Silva González

Fecha: 18 de diciembre de 2025

1. Introducción

El objetivo de esta práctica es el desarrollo de una aplicación web robusta utilizando el micro-framework **Flask** y el sistema de gestión de bases de datos **PostgreSQL**. La práctica se divide en dos bloques:

1. **Gestión de Libros:** Implementación de un sistema CRUD completo con énfasis en seguridad y manejo de excepciones.
2. **API de Telemetría:** Desarrollo de endpoints para la consulta de estadísticas de temperatura en diferentes estancias.

2. Configuración del Entorno

Para garantizar la reproducibilidad y el aislamiento de dependencias, se configuró un entorno virtual de Python:

Bash

```
# Preparación del entorno
python3 -m venv env
source env/bin/activate  # En Linux/Mac

# Instalación de dependencias críticas
pip install flask psycopg2
```

3. Actividad 1: Sistema de Gestión de Libros (CRUD)

3.1. Diseño de la Base de Datos e Inicialización

Se desarrolló el script init_db.py para automatizar la creación del esquema. Se insertaron **10 registros iniciales** que incluyen clásicos de la literatura, asegurando que la base de datos cuente con un volumen de datos suficiente para pruebas.

Manejo de Excepciones en el Script:
Se implementó una lógica de control de errores mediante bloques try-except-finally para capturar fallos de conexión (OperationalError) o errores sintácticos en SQL (DatabaseError), garantizando que los recursos se liberen siempre.

3.2. Operaciones Implementadas

A continuación se detallan las funcionalidades principales del sistema:

- **Lectura (READ):** La ruta raíz (/) recupera todos los libros utilizando consultas parametrizadas para evitar ataques de inyección.
- **Creación (CREATE):** Formulario en /create/ con validación estricta de tipos de datos (ej. asegurar que el número de páginas sea un entero) y limpieza de espacios con strip().
- **Actualización (UPDATE):** Permite editar registros existentes. Incluye una verificación previa de existencia (404 si el ID no existe) y precarga los datos en el formulario.
- **Borrado (DELETE):** Implementado exclusivamente mediante el método **POST** para prevenir ejecuciones accidentales a través de peticiones GET de navegadores o bots.

3.3. Seguridad y Robustez

Se han aplicado las siguientes capas de seguridad:

1. **Consultas Parametrizadas:** Uso sistemático de %s en psycopg2 en lugar de f-strings.
2. **Gestión de Transacciones:** Uso de conn.commit() tras operaciones exitosas y conn.rollback() en caso de excepción.
3. **Logging:** Registro de eventos y errores en la consola para facilitar la auditoría y el debugging.

4. Actividad 2: API de Consultas de Temperatura

Esta sección se centra en la exposición de datos estadísticos a través de endpoints JSON, simulando un sistema de monitorización de habitaciones.

4.1. Endpoints de la API

Método	Endpoint	Descripción
GET	/api/temperatures/average	Calcula la temperatura media de todos los registros globales.
GET	/api/temperatures/max	Devuelve el valor máximo de temperatura registrado.
GET	/api/rooms/<id>/name	Recupera únicamente el nombre de una estancia por su ID.
GET	/api/rooms/<id>/average	Procesa la media histórica específica de una habitación.
GET	/api/rooms/<id>/min-stats	Retorna un objeto JSON con el nombre y la temperatura mínima.

5. Conclusiones

La práctica demuestra la importancia de no solo hacer que el código funcione, sino de hacerlo **seguro y resiliente**. La separación de responsabilidades entre el script de inicialización y la aplicación Flask, junto con el manejo de excepciones de base de datos, proporciona una base sólida para aplicaciones de nivel de producción.

6. Capturas de Pantalla (Resumen)

Para las capturas de pantalla consultar books/README.md y home/README.md en el código.