Universidad De Las Fuerzas Armadas - ESPE

Departamento De Ciencias De La Computación

Aplicaciones Basadas en el Conocimiento - 28492

Documentación

Docente:

Ing. David Esteban Galarza García

Estudiantes:

Asmal Haro Kevin David

Tipan Muñoz Fernando Rene

Chiriboga Zurita Daniel Isai

28 de enero de 2026

Documentación de la Pseudoestructura del Sistema de Turnos

Este documento describe la pseudoestructura (pseudocódigo estructural) del módulo Turno, basado en el sistema desarrollado en Java (Swing). La pseudoestructura representa la lógica, flujo y organización del sistema de forma abstracta, independiente del lenguaje de programación.

1. Descripción general

La API de Turnos permite gestionar la creación, consulta y actualización de turnos dentro del sistema. Proporciona endpoints para insertar nuevos turnos, listar todos los turnos, buscar turnos por nombre y actualizar turnos específicos.

Todas las solicitudes y respuestas usan el formato application/json.

Los métodos HTTP utilizados son: GET, POST, PUT.

Las respuestas exitosas devuelven código 200 OK.

El módulo Turno permite gestionar horarios de trabajo mediante una interfaz gráfica.
Las funciones principales incluyen:
- Registro de turnos
- Edición de turnos
- Búsqueda por nombre
- Cálculo automático de horas
- Listado de turnos

2. Arquitectura

La solución sigue una arquitectura cliente-servidor:
- Servidor: API REST en Python (FastAPI)
- Comunicación: HTTP + JSON
- Base de datos: MySQL

Estructura del Proyecto

proyecto

database

connection.py

interface

init.py

turno_ i. py -> read_all(), read_by_name(), insert(), update()

repositorio

init.py

turno_r. py -> driver de conx

routers

init.py

turno_ro. py -> url/api/read_all

venv

env.

app. py

package.json

package-lock.json

turnos.db

database

connection.py: Este código define una función para establecer una conexión a una base de datos PostgreSQL utilizando la librería psycopg2; al importar RealDictCursor, se configura la conexión para que los resultados de las consultas se devuelvan como diccionarios en lugar de tuplas, facilitando su conversión a JSON. La función get_connection() encapsula los parámetros de conexión (host, nombre de la base de datos, usuario, contraseña y puerto) y devuelve una conexión lista para ser usada por la capa de repositorio de la aplicación.

interface

turno_ i. py -> read_all(), read_by_name(),, insert(), update(): Este código corresponde a la capa de repositorio o acceso a datos del módulo de turnos y se encarga de comunicarse con la base de datos SQLite: utiliza get_connection() para abrir la conexión y define funciones para leer todos los turnos (read_all), buscar turnos por nombre usando una coincidencia parcial con LIKE (read_by_name), insertar un nuevo turno (insert) y actualizar uno existente por su identificador (update); en cada caso, los registros obtenidos se transforman en diccionarios para que la API pueda devolverlos fácilmente en formato JSON, y la conexión se cierra correctamente tras cada operación.

repository

turno_r. py -> driver de conx: Este código implementa el repositorio de datos para la entidad turnos usando PostgreSQL, encargándose de realizar operaciones CRUD sobre la tabla turnos: define funciones para obtener todos los turnos (get_all) y buscarlos por nombre de forma insensible a mayúsculas usando ILIKE (get_by_name), formateando los campos de hora con TO_CHAR para devolverlos en formato HH:MM; además, permite crear (create) y actualizar (update) turnos, utilizando RETURNING para devolver inmediatamente el registro insertado o modificado, cerrando correctamente el cursor y la conexión en cada operación para garantizar una gestión adecuada de recursos.

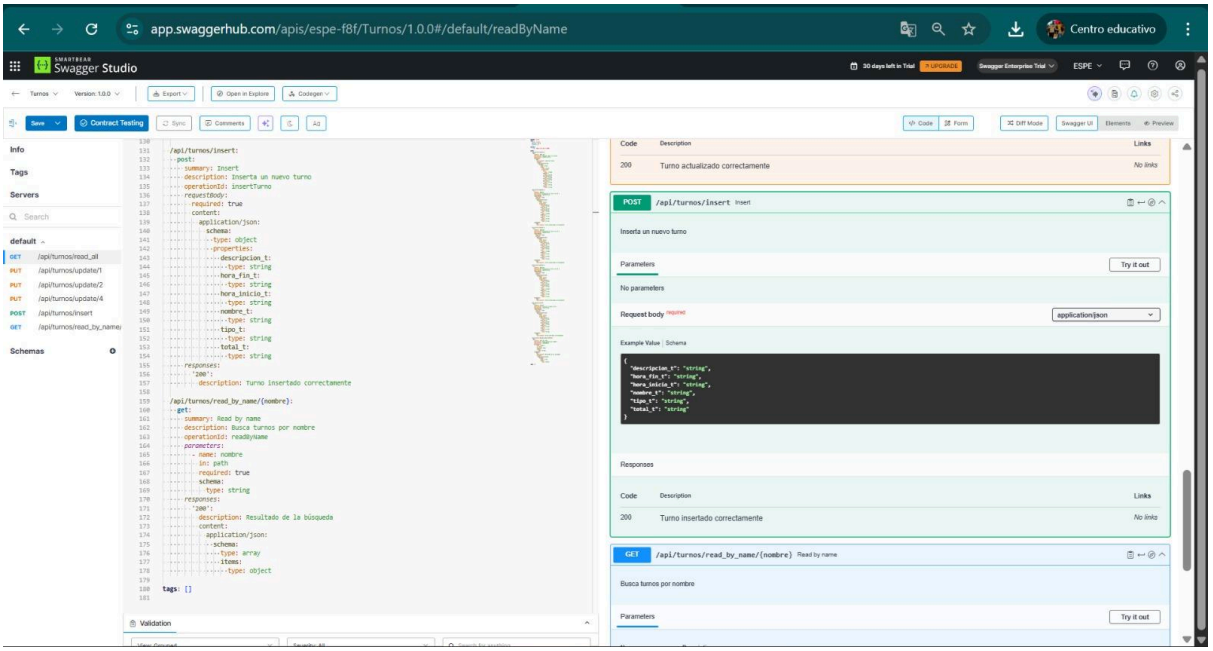
routers

turno_ro. py -> url/api/read_all: Este código define un Blueprint de Flask que expone la API REST del módulo de turnos bajo la ruta base /api/turnos: declara endpoints para obtener todos los turnos (GET /read_all), buscar turnos por nombre (GET/read_by_name/<nombre>), insertar un nuevo turno (POST /insert) y actualizar un turno existente por su identificador (PUT /update/<turno_id>); cada ruta recibe o envía datos en formato JSON, delega la lógica de negocio a la capa de interfaces (read_all, read_by_name, insert, update_turno) y devuelve códigos HTTP adecuados, manteniendo una separación clara entre controladores, lógica y acceso a datos.

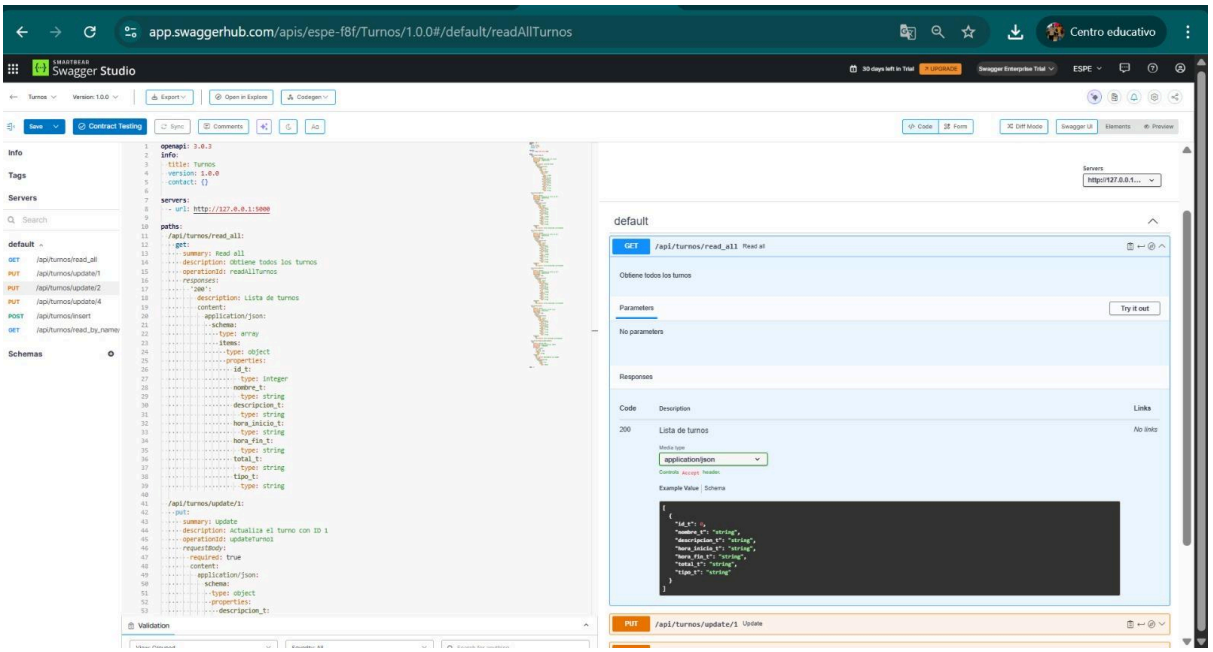
app. py: Este código crea y ejecuta una API web usando Flask, donde la aplicación principal se define de forma modular mediante una función (create_app) y se organizan las rutas a través de un Blueprint importado (turno_bp), que contiene los endpoints relacionados con los turnos; al registrarlo, esas rutas quedan disponibles en la aplicación. Finalmente, el servidor se inicia en modo desarrollo (debug=True), escuchando en localhost (127.0.0.1) y en el puerto 5000, permitiendo acceder a la API desde el navegador o herramientas como Postman.

3. Métodos

POST /api/turnos/insert: Inserta un nuevo turno en el sistema.

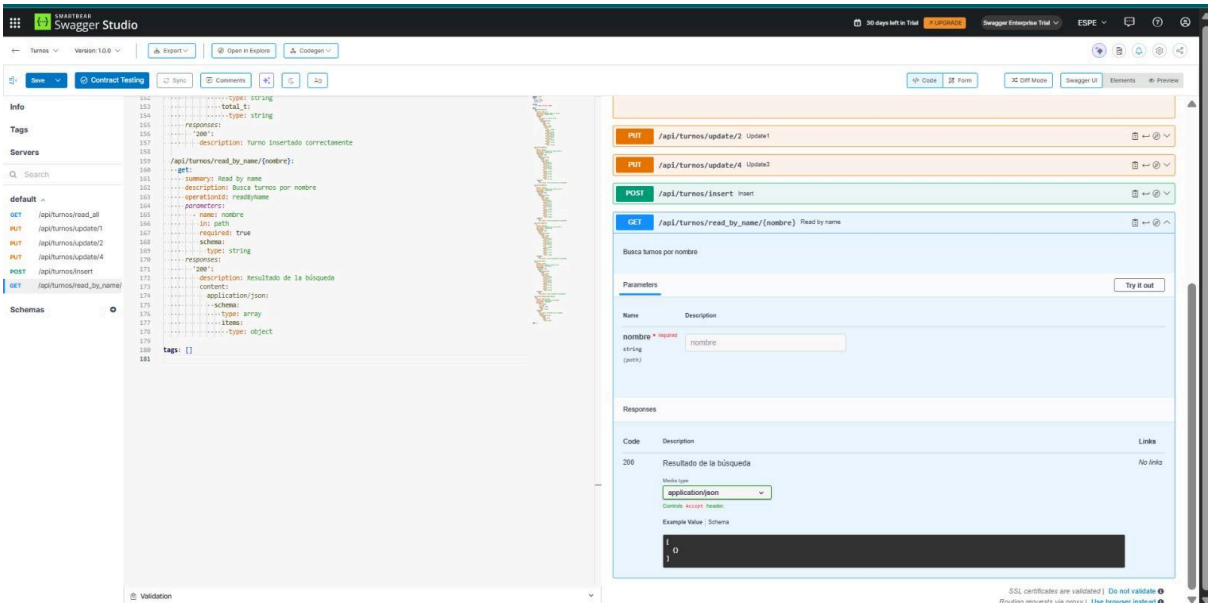


GET /api/turnos/read_all: Obtiene la lista completa de turnos registrados.

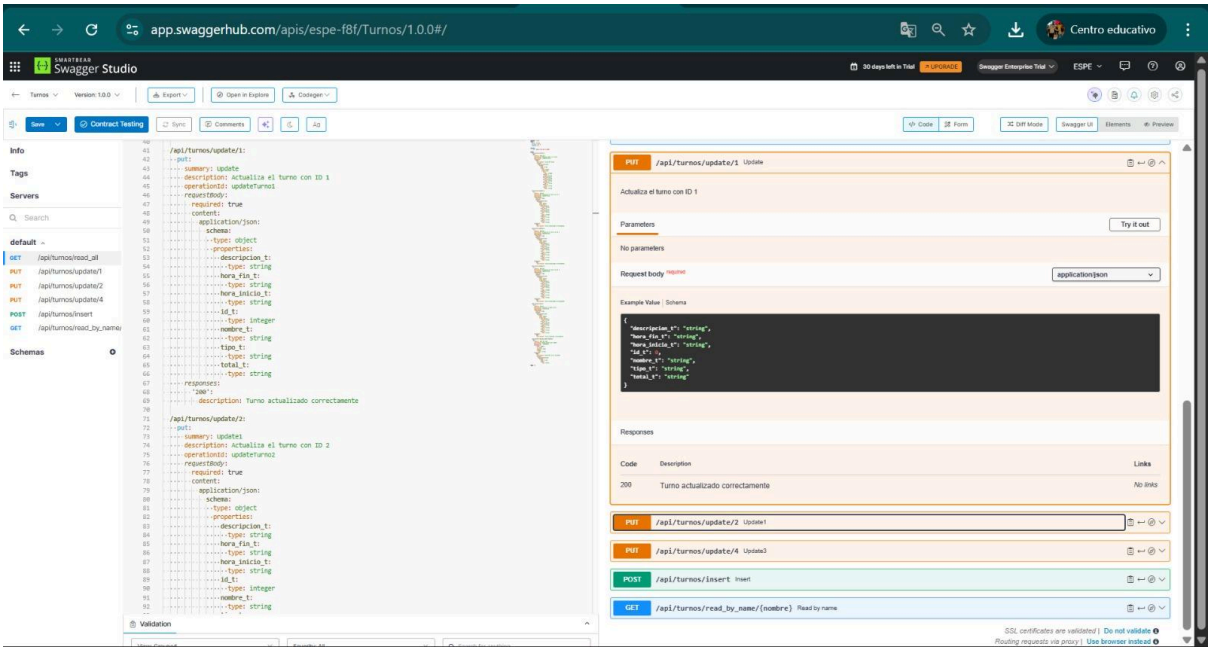


GET /api/turnos/read_by_name/{nombre}: Busca turnos cuyo nombre coincida con el parámetro enviado.

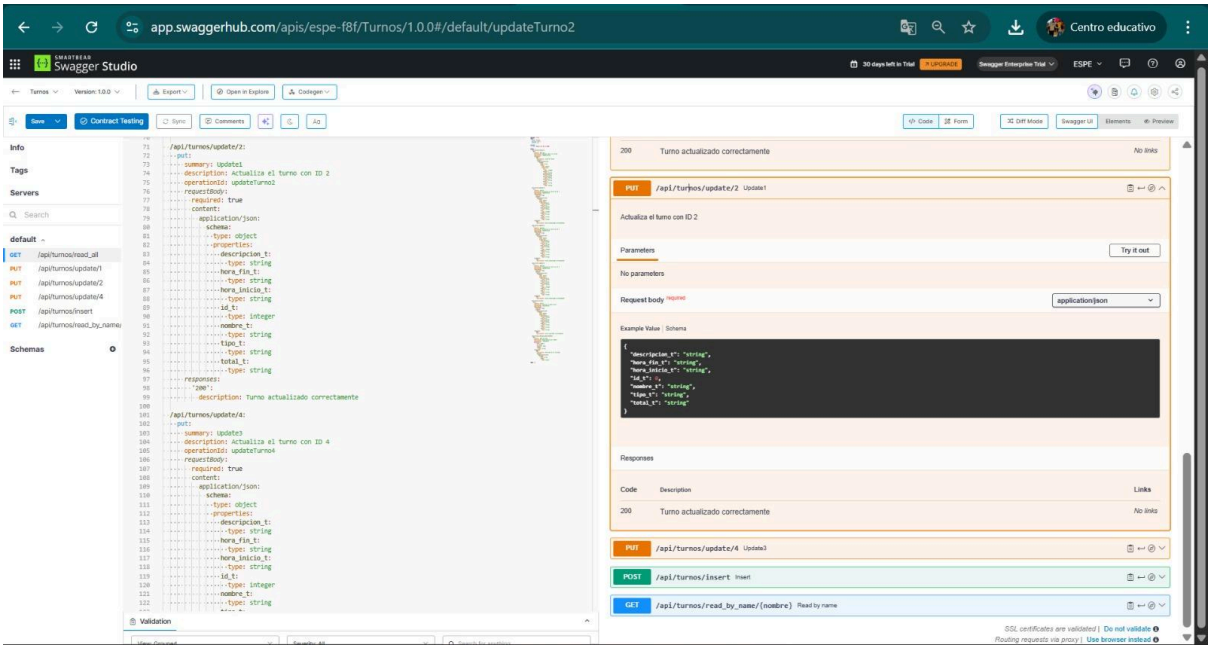
nombre (string): nombre del turno a buscar.



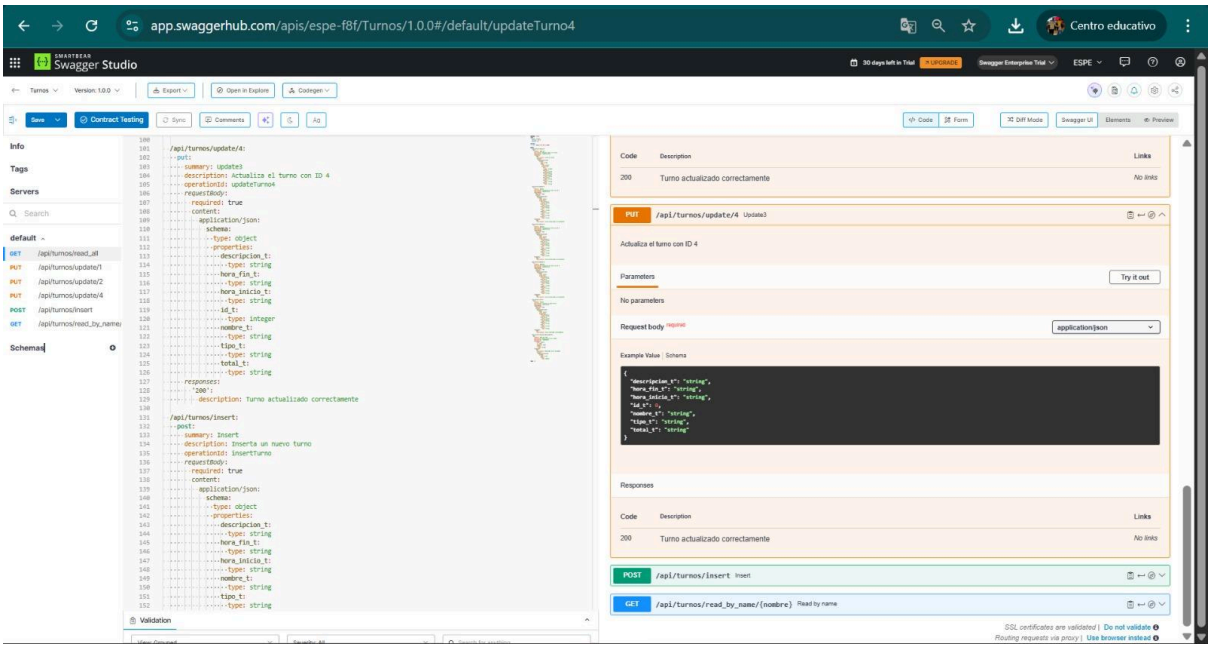
PUT /api/turnos/update/1: Actualiza la información del turno con ID 1.



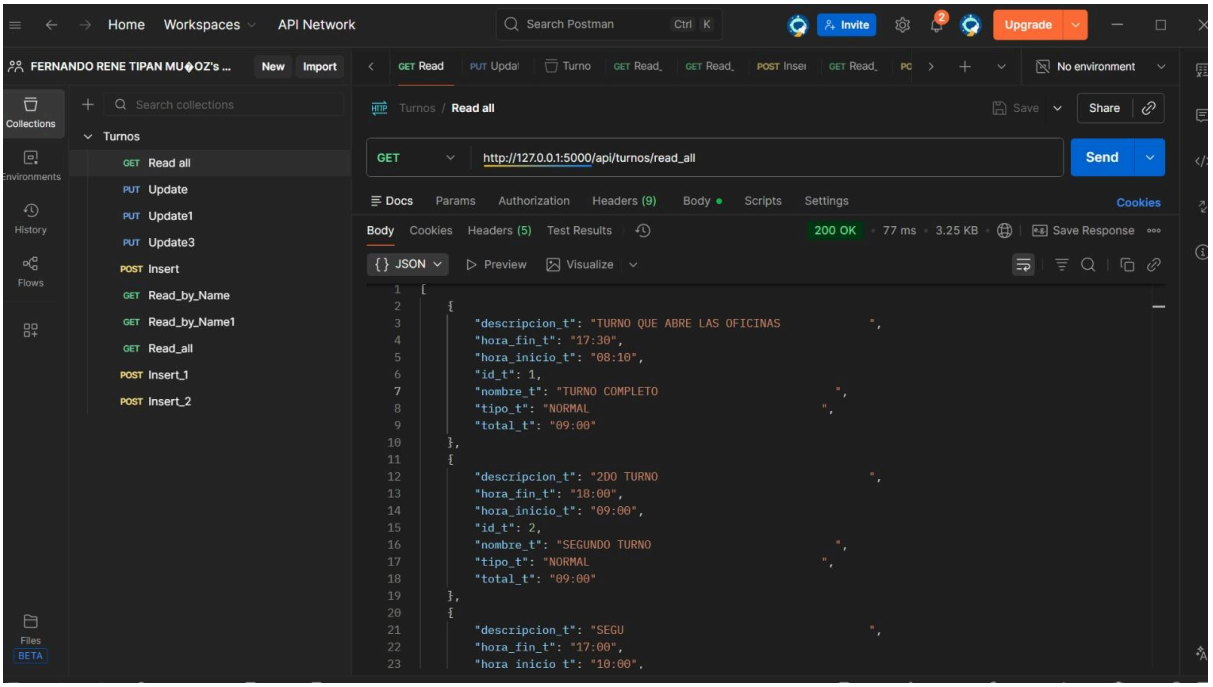
PUT /api/turnos/update/2: Actualiza la información del turno con ID 2.



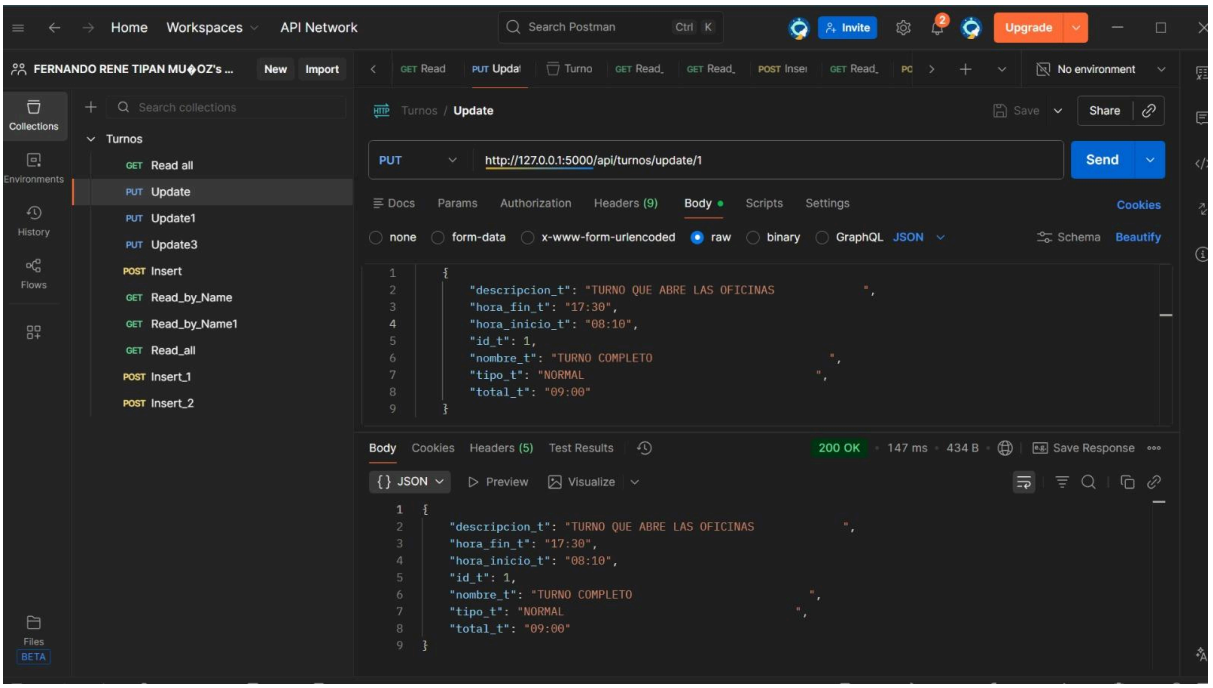
PUT /api/turnos/update/4: Actualiza la información del turno con ID 4.



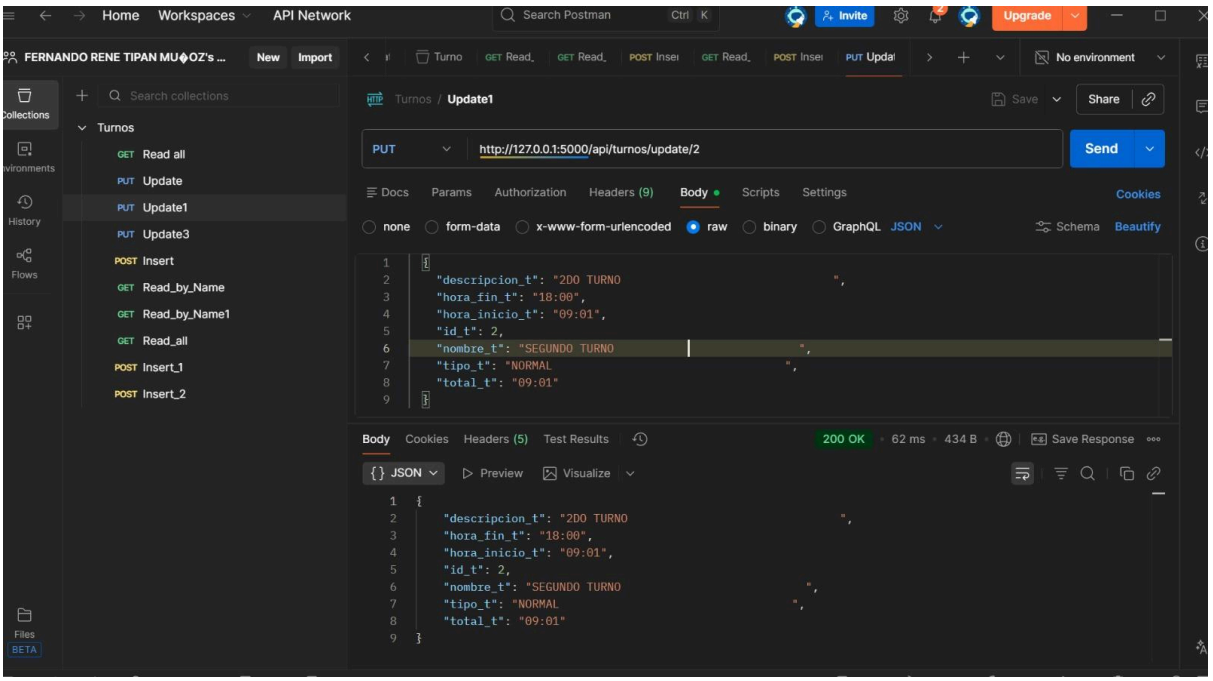
GET Read all



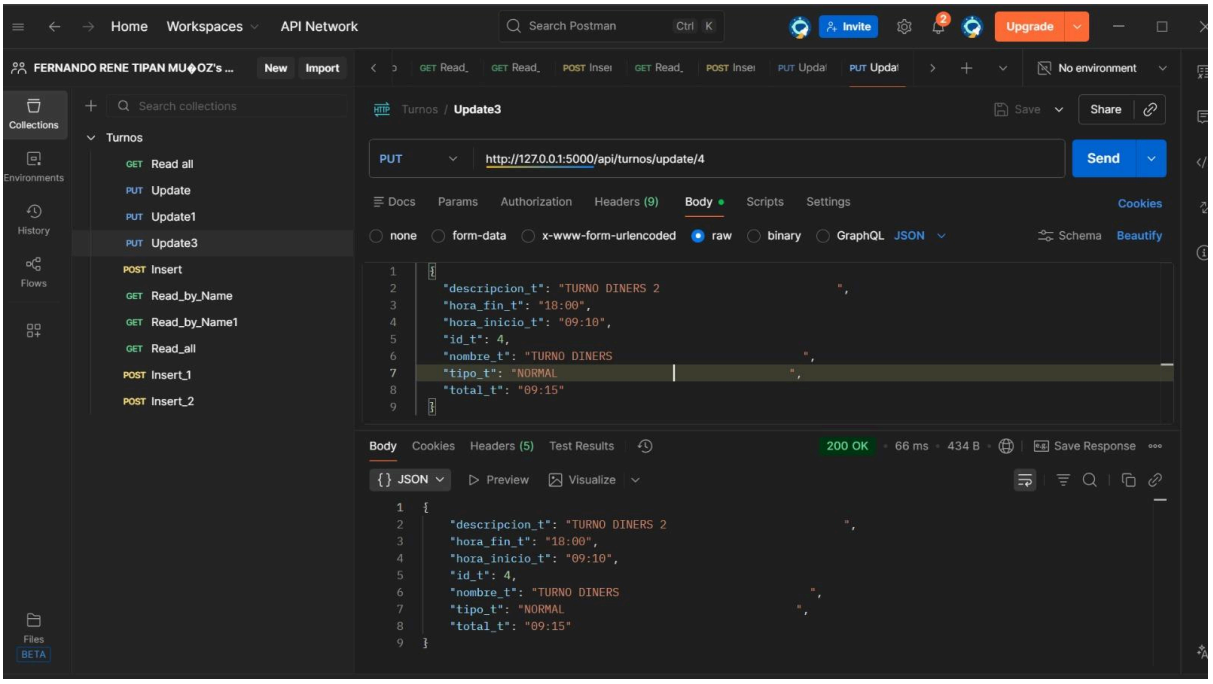
PUT Update



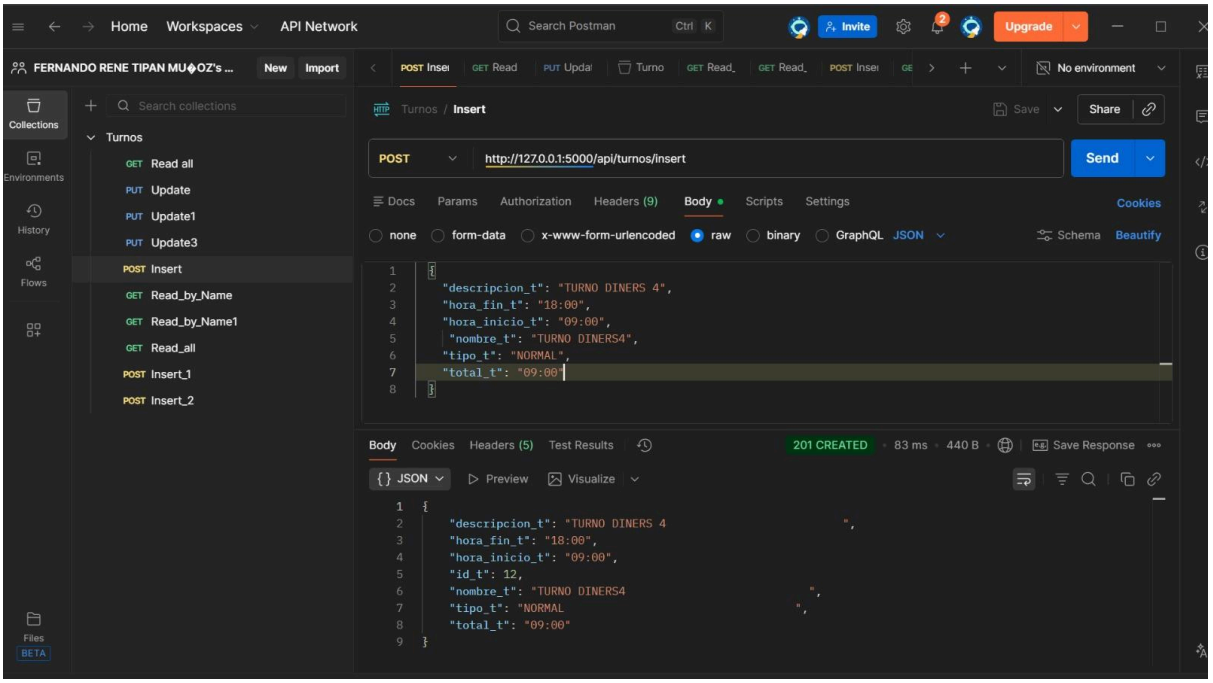
PUT Updatel



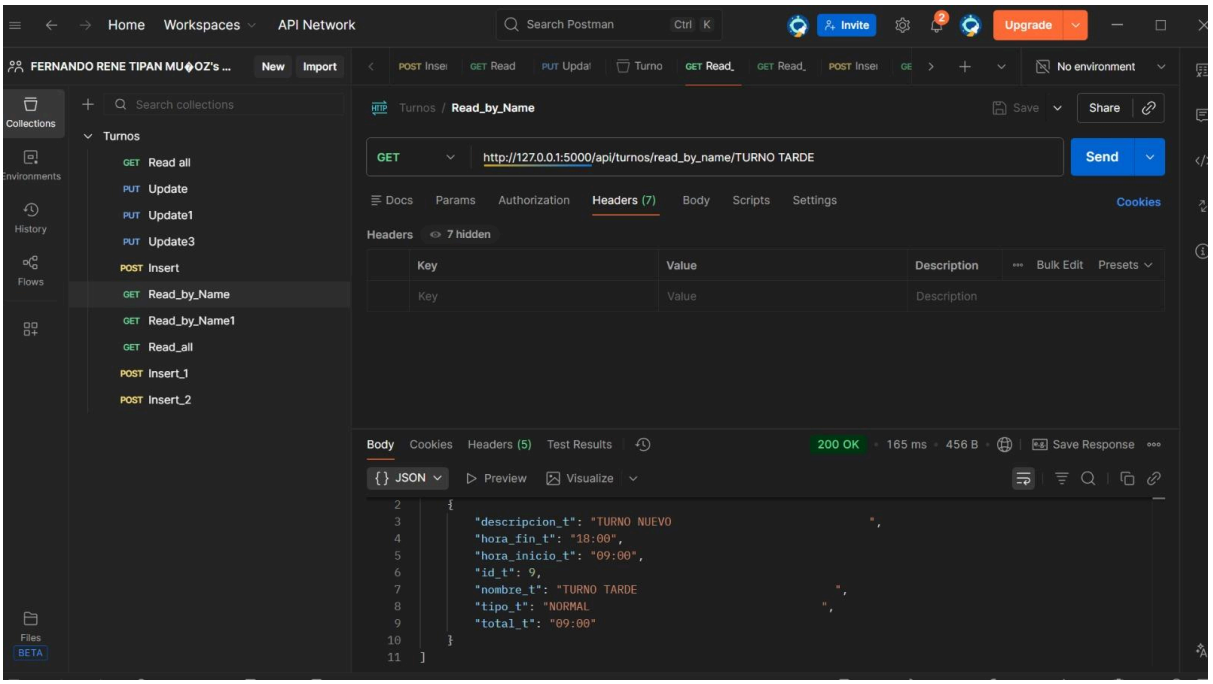
PUT Update3



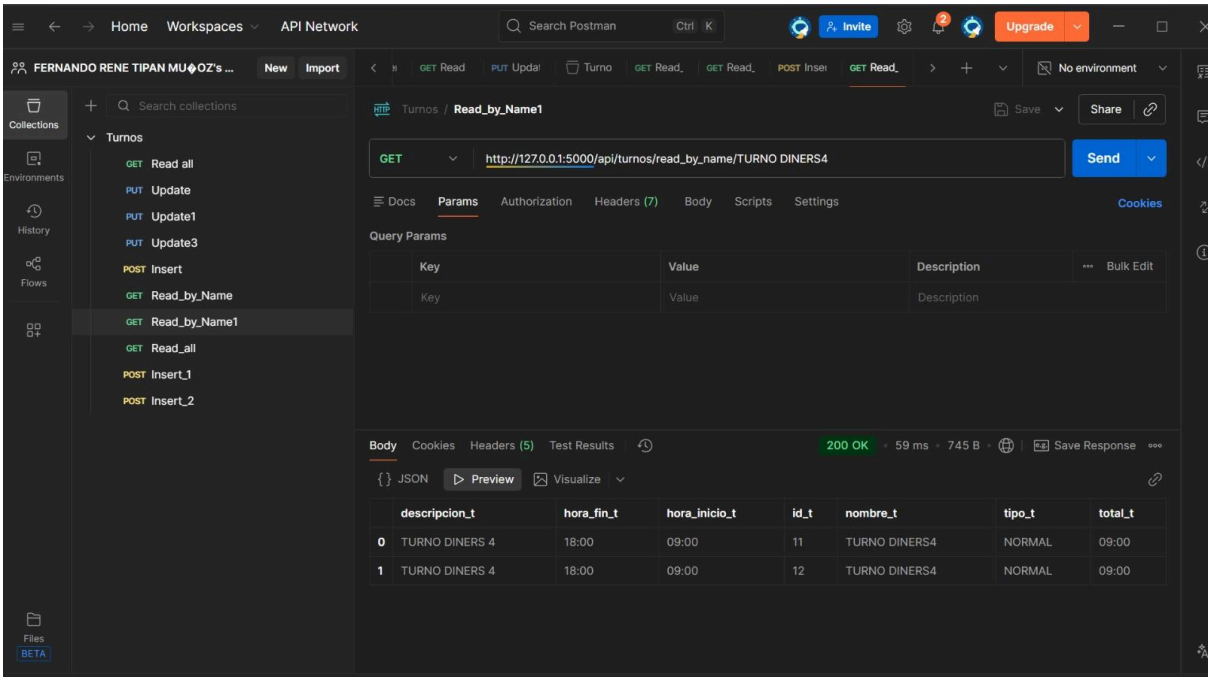
POST Insert



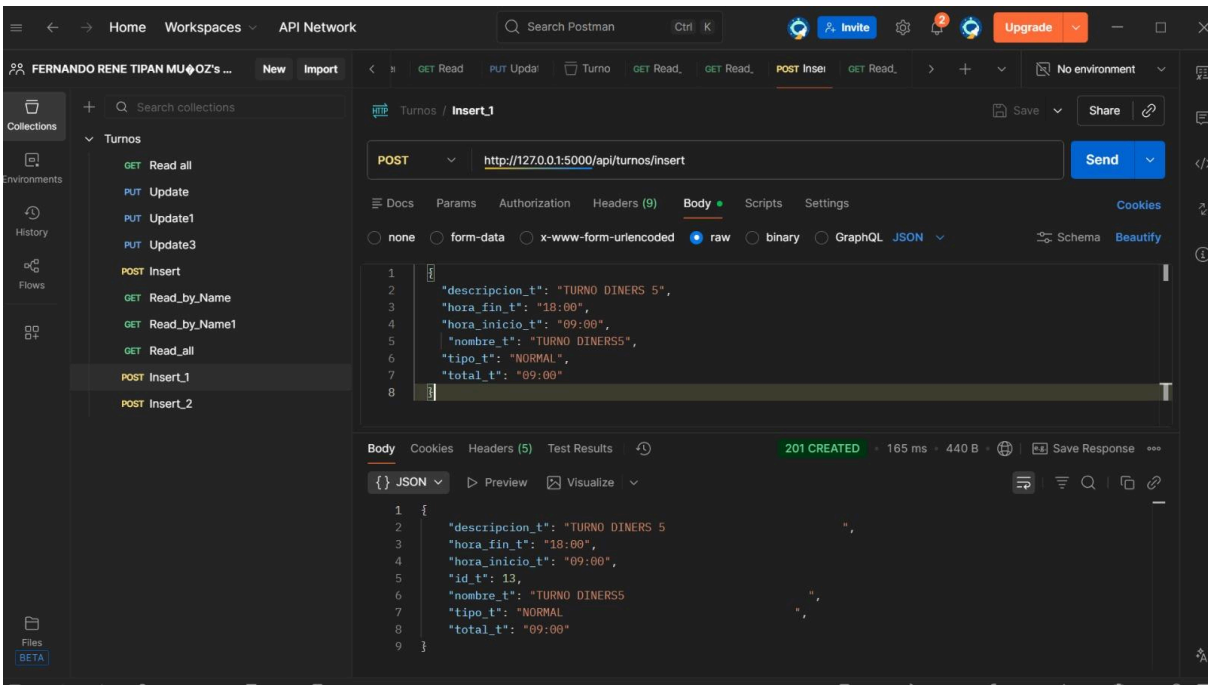
GET Read_by_Name



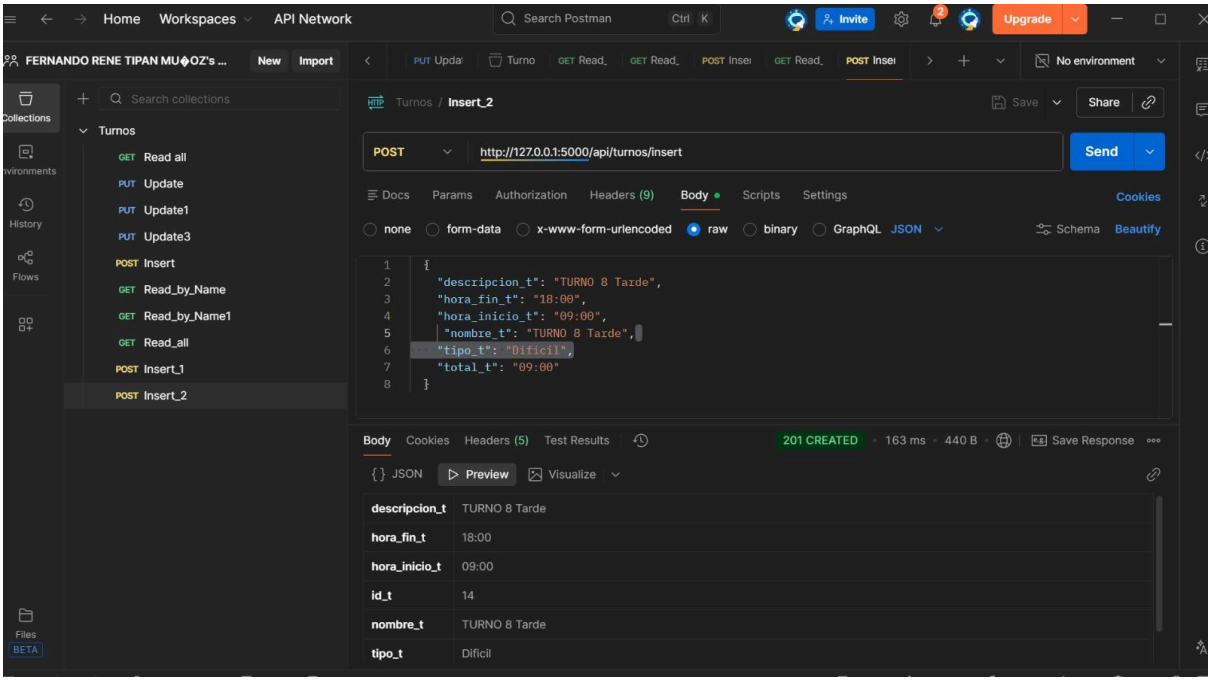
GET Read_by_Name1



POST Insert_1



POST Insert_2



Campo	Tipo	Descripción
id_t	Int	Identificador único del turno
nombre_t	String	Nombre del turno
descripcion_t	String	Descripción del turno
hora_inicio_t	String	Hora de inicio
hora_fin_t	String	Hora de fin
total_t	String	Total de horas del turno
tipo_t	String	Tipo de turno

4. Conclusión

La pseudoestructura permite comprender la lógica interna del sistema de turnos sin depender del código fuente, facilitando el análisis, documentación y posterior implementación como API.

En conjunto, todos estos códigos implementan una API REST completa para la gestión de turnos, estructurada por capas y conectada correctamente a una base de datos: la aplicación principal en Flask crea y arranca el servidor, registrando un Blueprint que define los endpoints HTTP; el Blueprint actúa como controlador, recibe las solicitudes (GET, POST y PUT), maneja los datos JSON y delega la lógica a la capa de interfaces; dicha capa se apoya en el repositorio, que es el encargado de ejecutar operaciones CRUD (consultar, insertar y

actualizar turnos) directamente sobre la base de datos; finalmente, la capa de conexión gestiona el acceso a SQLite o PostgreSQL, asegurando que la base exista, que las consultas se ejecuten correctamente y que los resultados se devuelvan en un formato adecuado (diccionarios/JSON), logrando así que la API funcione de forma modular, mantenible y con persistencia real de datos.