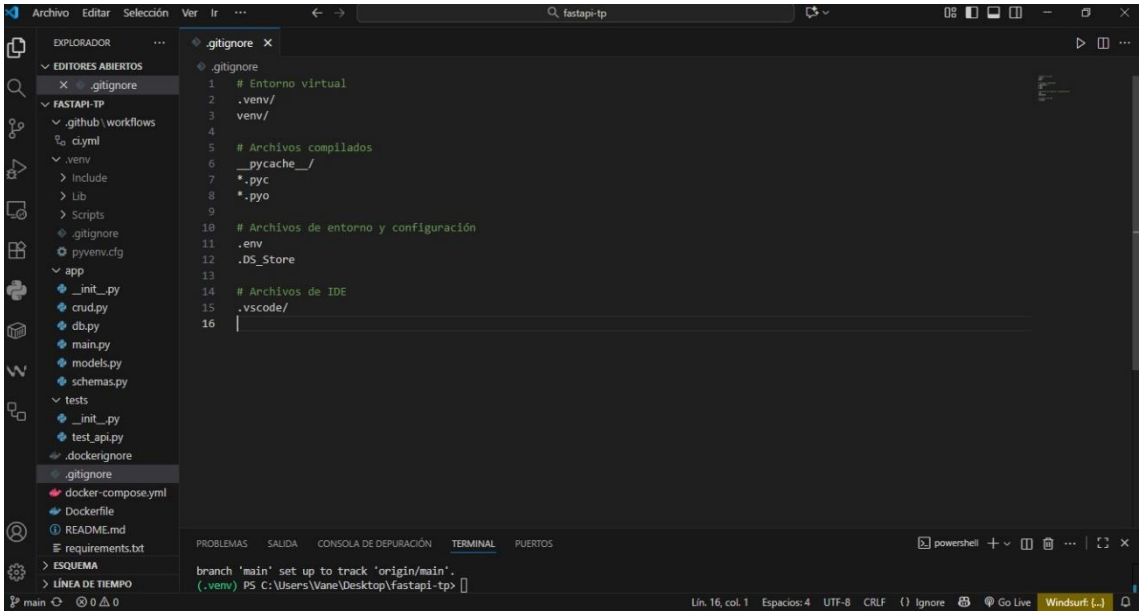
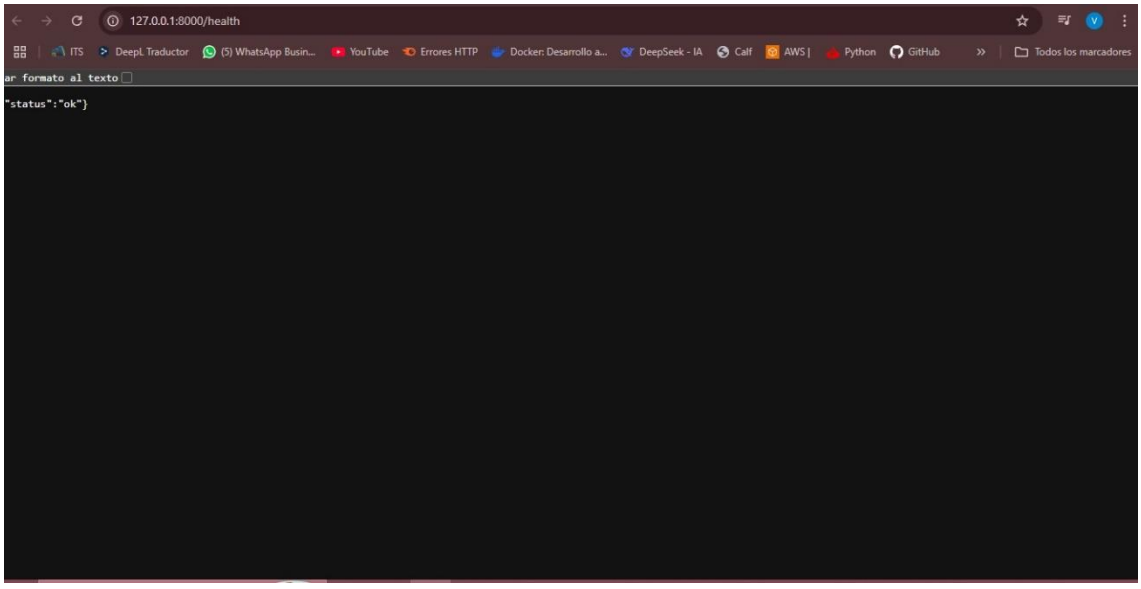
Trabajo Final Obligatorio

Evidencia de ejecución

- Estructura inicial del proyecto

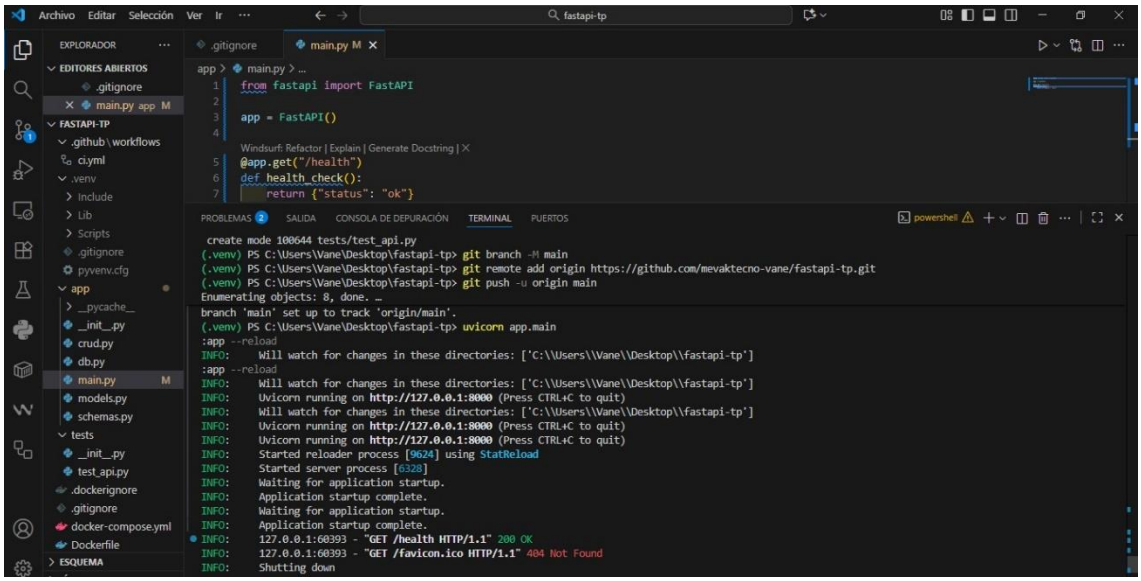


- La API corriendo en http://localhost:8000/health

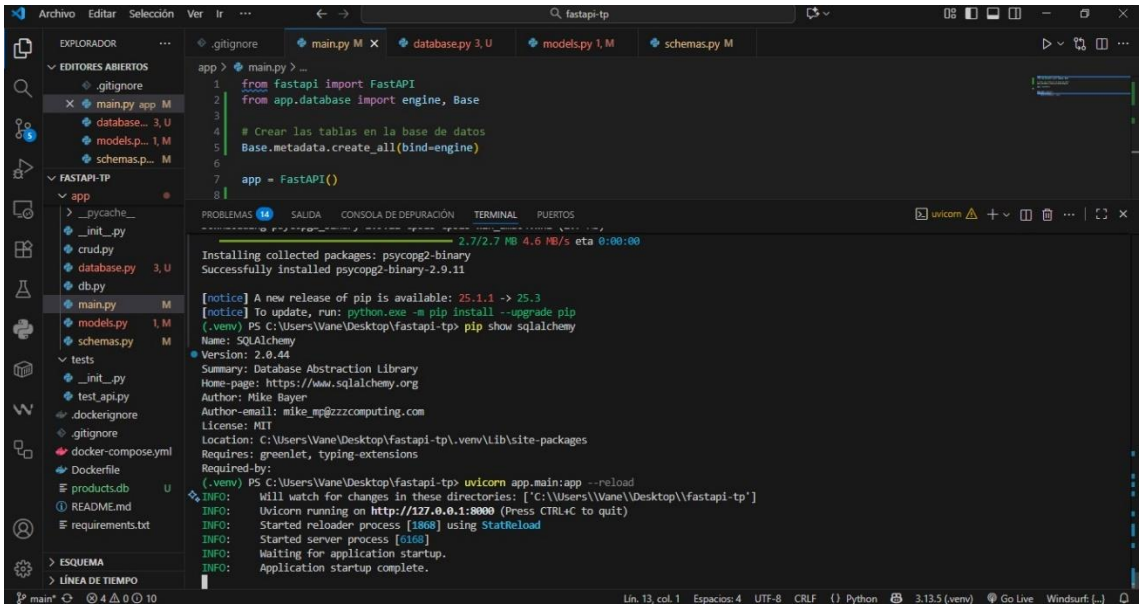


Trabajo Final Obligatorio

Y en la terminal:



- Se crea database y el archivo products.db

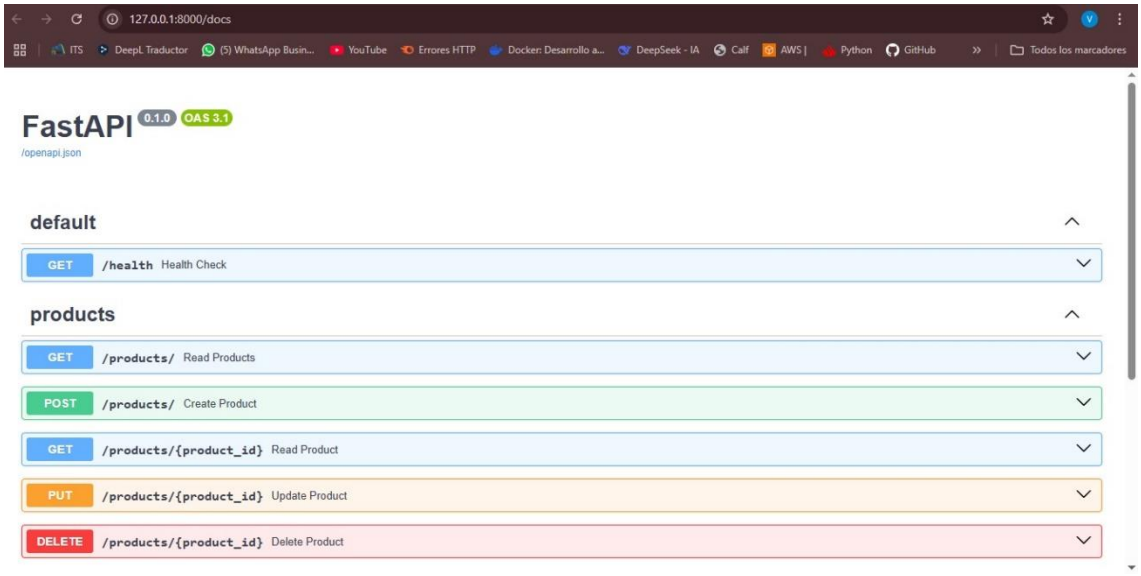


- Endpoints CRUD de Productos

Trabajo Final Obligatorio

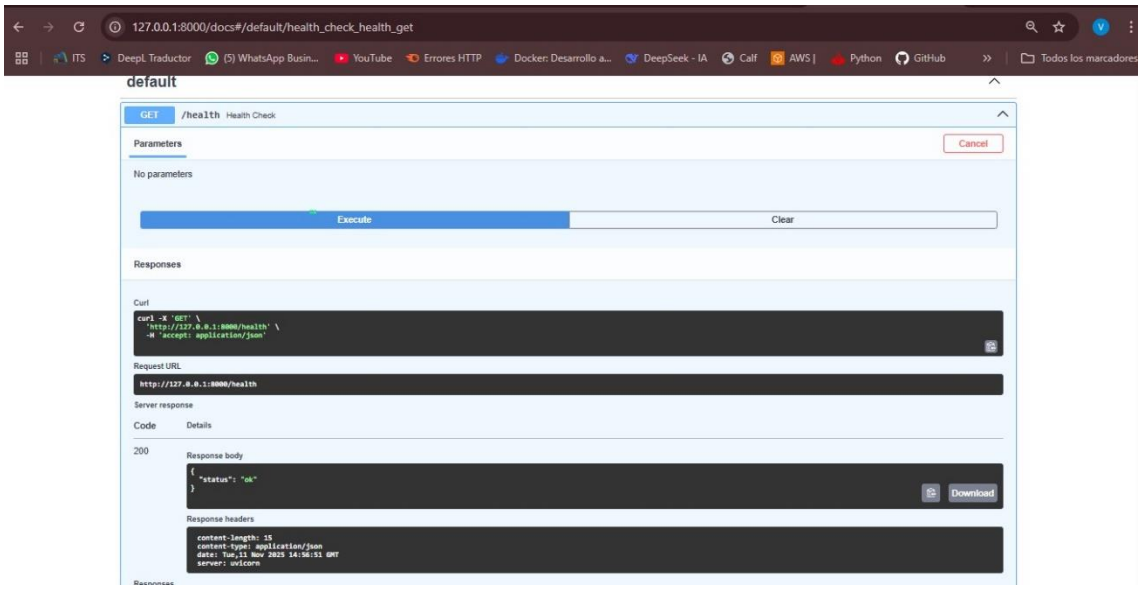
- Registro de las rutas creadas CRUD en app/main.py

En http://127.0.0.1:8000/docs



Pruebas del CRUD de productos: la API funciona correctamente desde el navegador o Swagger UI. http://127.0.0.1:8000/docs

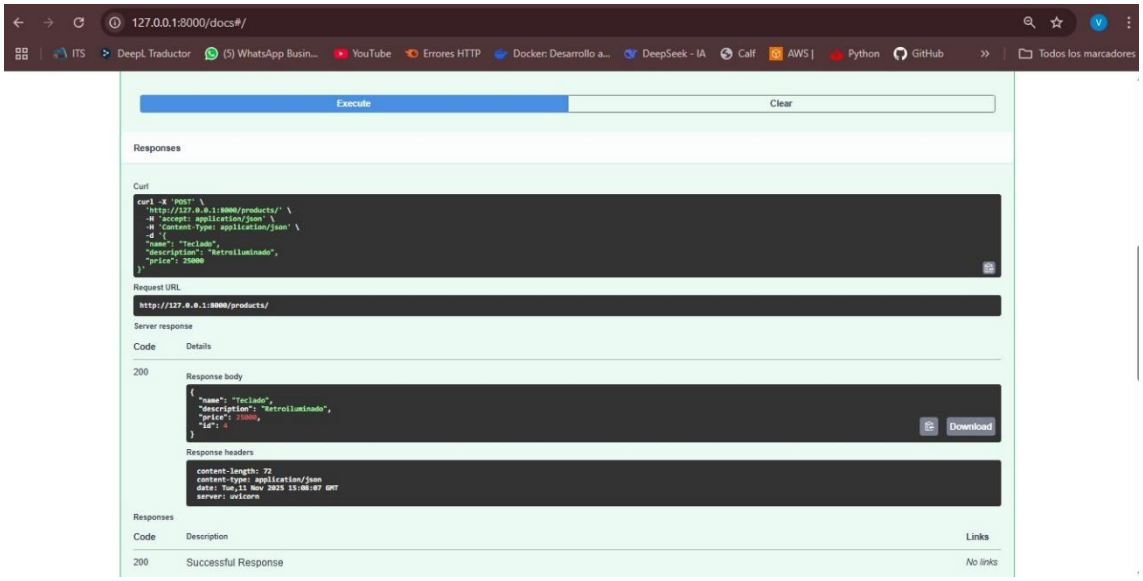
- GET /health — Health Check: devuelve un JSON simple que confirma que la API está viva



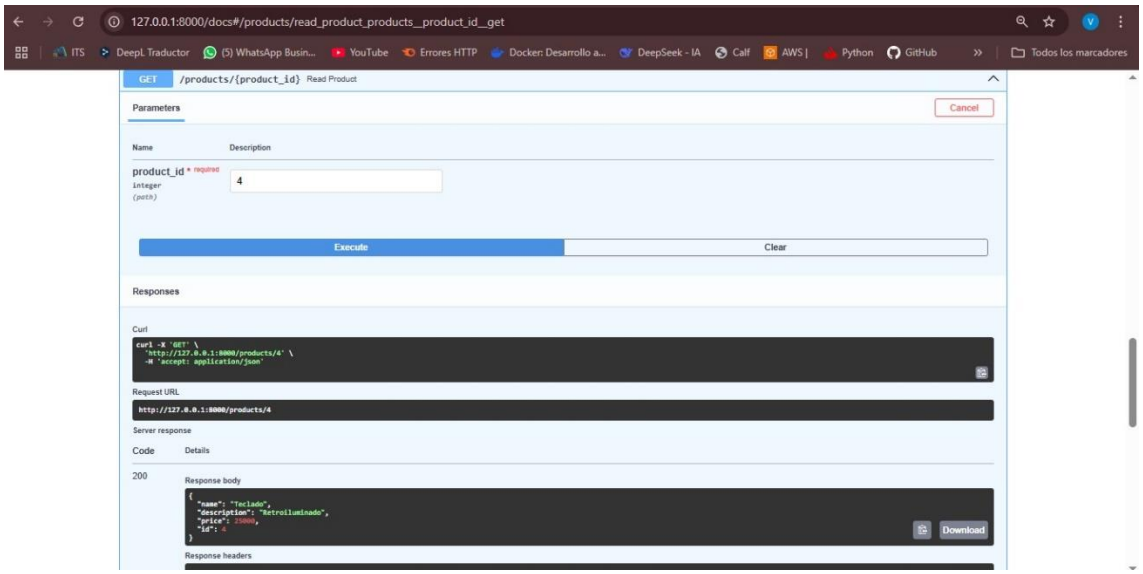
- POST /products/ — *Crear producto*: crea un nuevo registro en la base de datos.

Trabajo Final Obligatorio

Resultado de post:

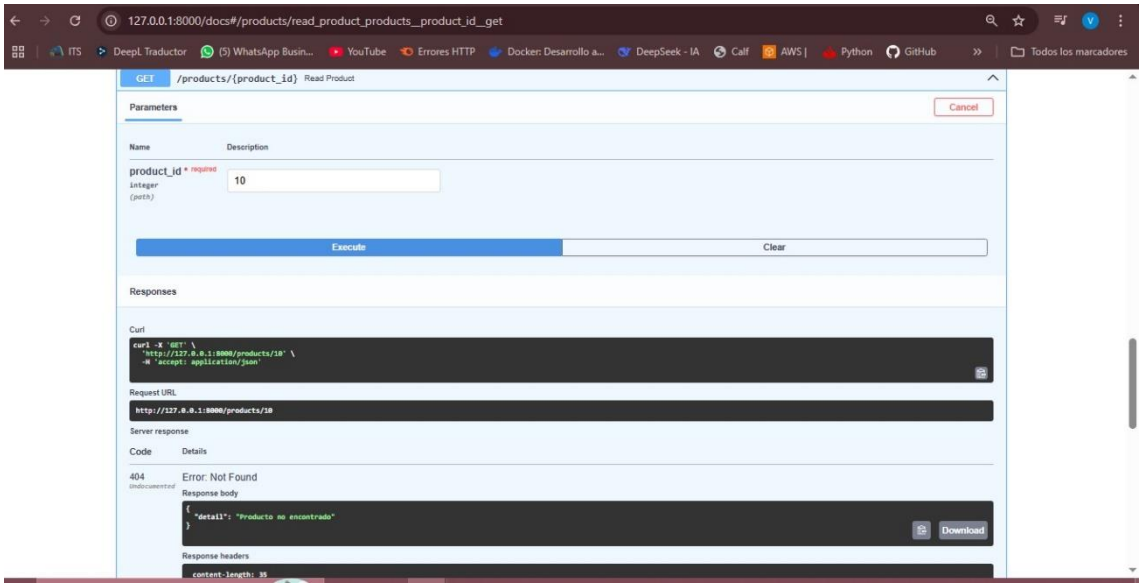


- GET /products/{product_id} — Leer producto por id: devuelve el producto con el id dado. (en este caso el id 4, el producto creado antes)

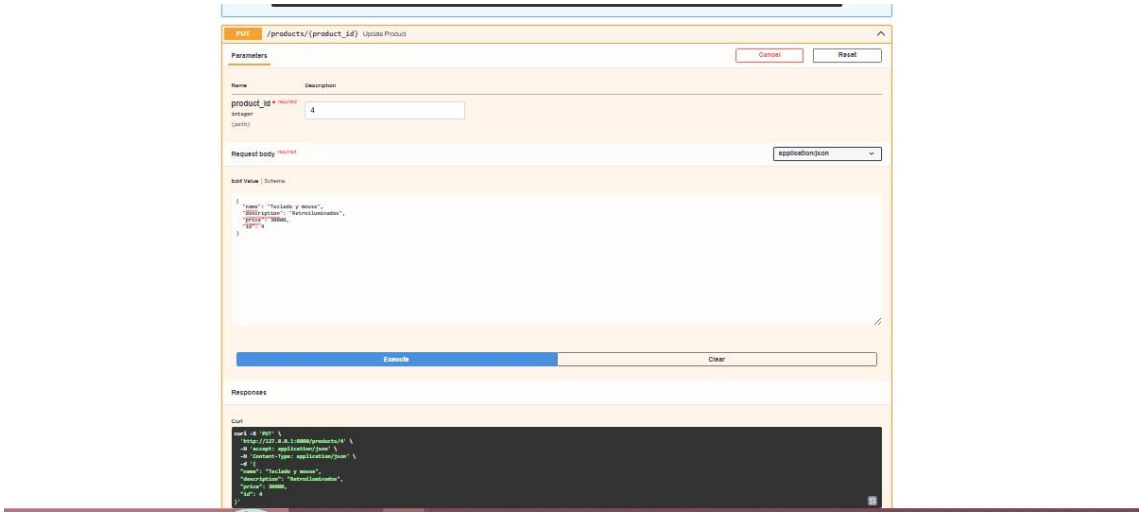


Trabajo Final Obligatorio

Ahora coloco un id de un producto inexistente para que devuelva error (id10, error 404)

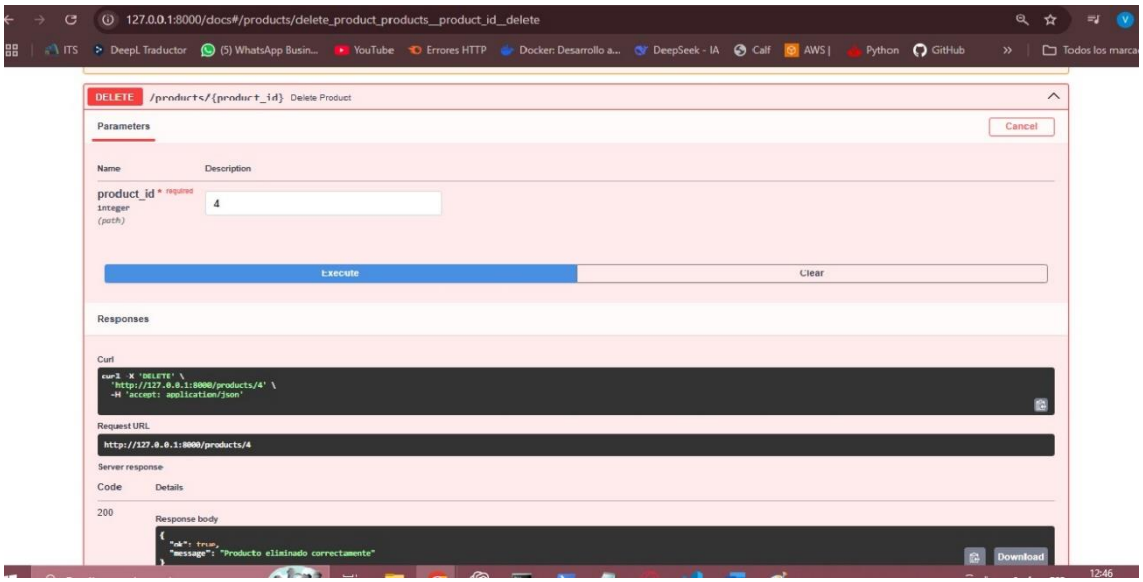


- PUT /products/{product_id} — Actualizar producto: actualiza los campos del producto indicado por product_id.



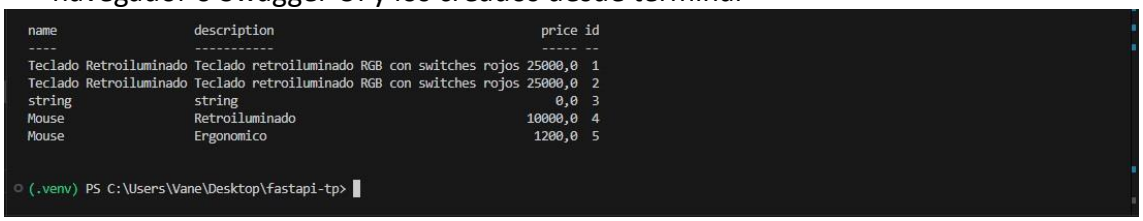
Trabajo Final Obligatorio

- DELETE /products/{product_id} — *Eliminar producto*: borra el producto con ese id



- Post desde terminal:

- Get desde terminal: muestra los distintos productos creados como prueba desde el navegador o Swagger UI y los creados desde terminal

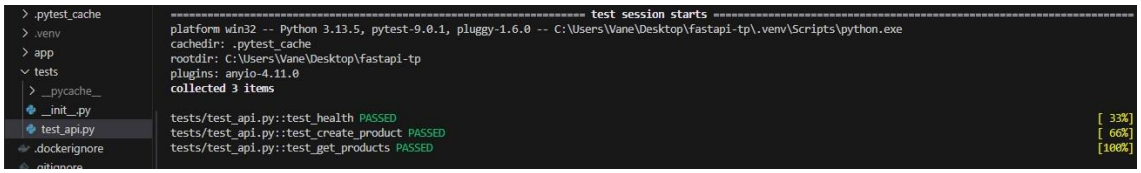


- PUT desde terminal:

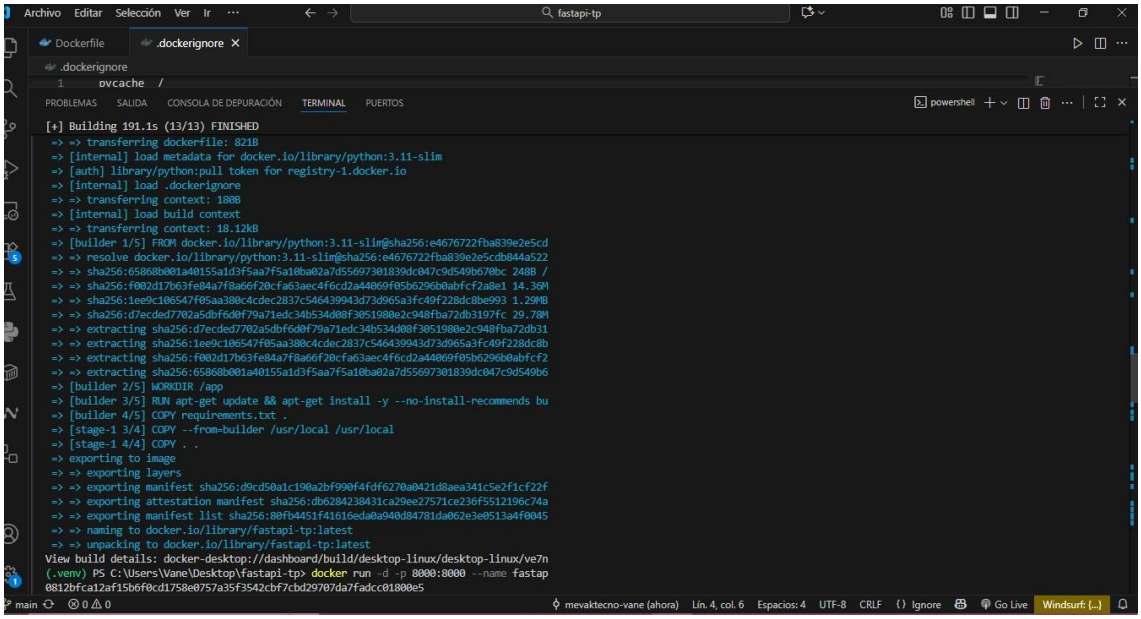
- DELETE desde terminal:

Trabajo Final Obligatorio

- Creo y pruebo tests



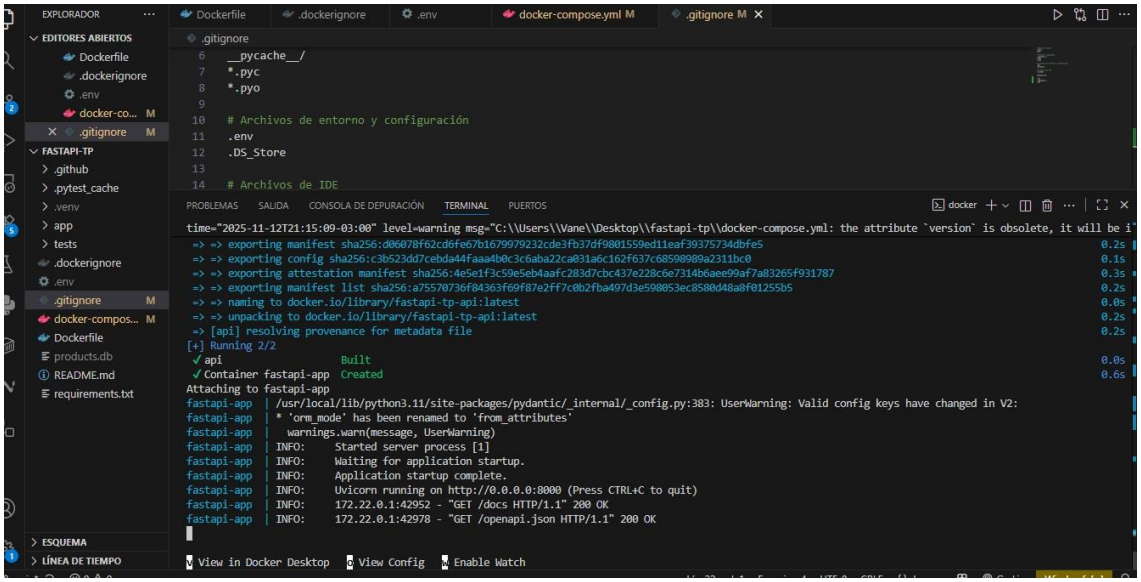
- Creación del contenedor



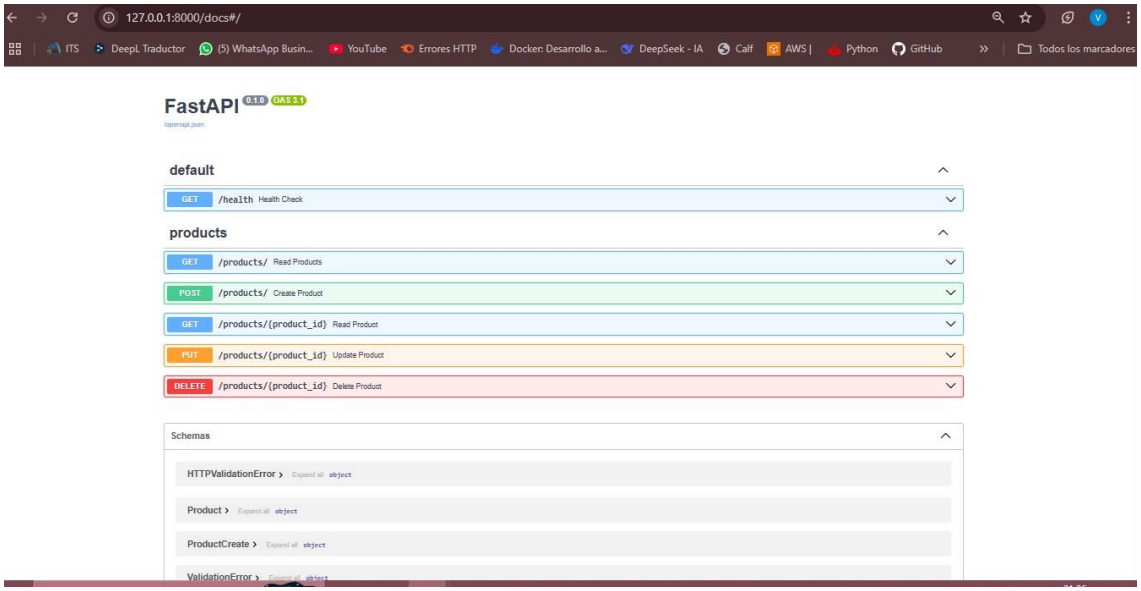
- Luego de resolver problemas de nombre (tenía otro contenedor con el mismo nombre) y de puerto (estaba siendo usado por Portainer) queda el contenedor corriendo

Trabajo Final Obligatorio

- Dockercompose.yml

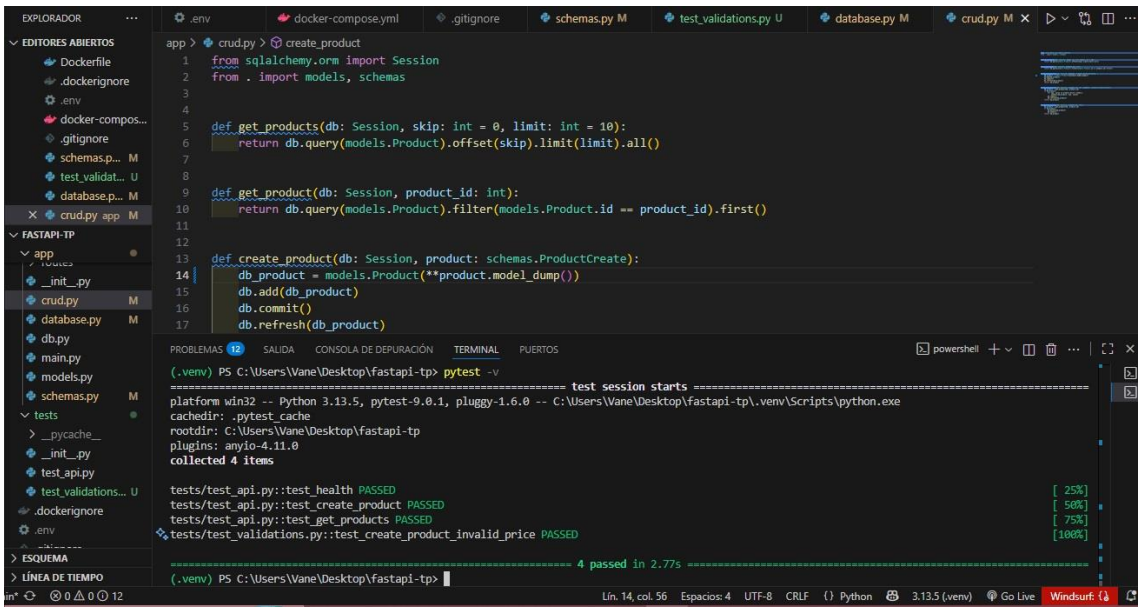


- http://127.0.0.1:8000/docs funcionando



Trabajo Final Obligatorio

- Agrego validaciones, manejo de errores claros y pruebas automáticas con pytest: Validaciones en schemas.py, Creo un nuevo archivo de test: tests/test_validations.py. Luego hice modificaciones en app/database.py y en app/crud.py por warnings que se veían a la hora de ejecutar los tests:



Conclusión

El desarrollo de esta API sirvió para integrar conceptos fundamentales de FastAPI, manejo de base de datos con SQLAlchemy, validaciones con Pydantic y ejecución de pruebas automáticas con pytest.
Además, se trabajó con contenedores Docker para asegurar un entorno de ejecución consistente y portable, y se incorporó un flujo de Integración Continua con GitHub Actions.

El resultado final es una API funcional, testeada, contenedorizada y lista para desplegarse.
El repositorio incluye todo el código fuente, el archivo docker-compose.yml, el Dockerfile, los tests, y este documento con las capturas de los pasos más importantes del proceso.

Integración Continua:
Los tests automáticos se ejecutan en cada commit mediante GitHub Actions.
Link: https://github.com/mevaktecno-vane/fastapi-tp/actions