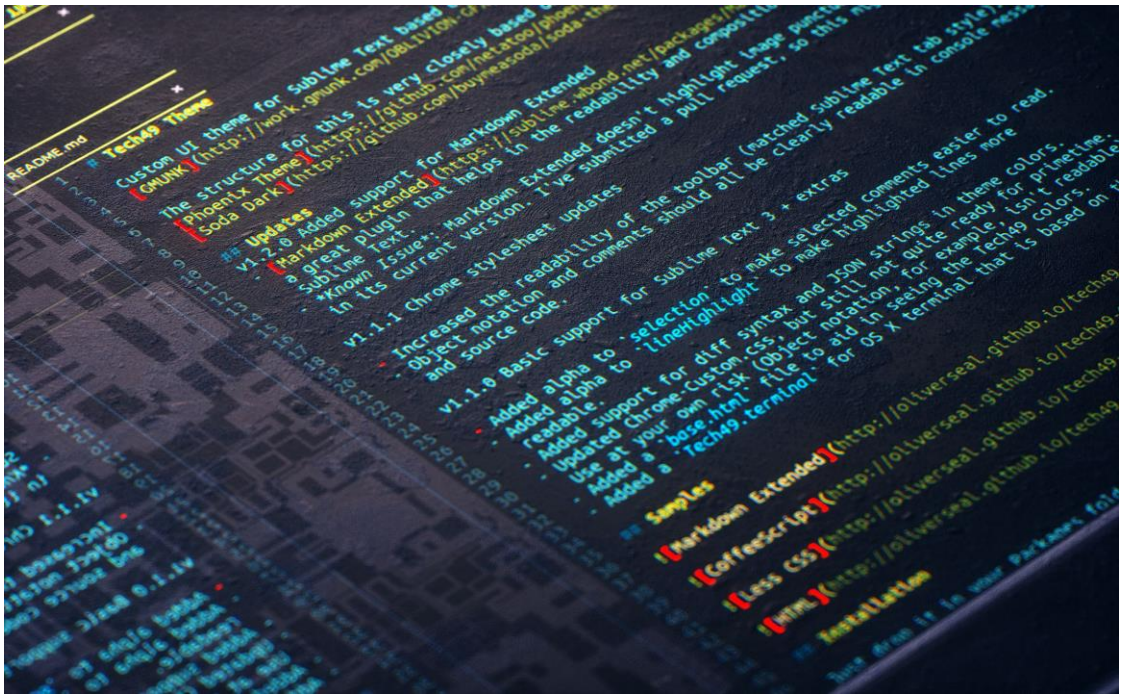
TRABAJO FINAL DE CICLO

DESARROLLO APLICACIONES WEB



Autor:

ALEJANDRO SANTOS CABRERA

Tutor:

FERNANDO PRADO

13/06/2025

ÍNDICE

1. Introducción (Idea del proyecto)

Este Trabajo de Fin de Ciclo del Grado Superior en Desarrollo de Aplicaciones Web (DAW) persigue el diseño y desarrollo de una plataforma web completa para usuarios que buscan gestionar sus entrenamientos y nutrición. La aplicación incluirá:

- Listado de ejercicios organizados por grupo muscular y tipo de equipo (barbell, dumbbell, bodyweight, etc.).
- Imágenes y vídeos demostrativos para mejorar la técnica.
- Creación de rutinas personalizadas con orden, repeticiones, series y descanso.
- Posibilidad de iniciar una rutina como sesión real, registrando peso real,repeticiones efectivas y comentarios.
- Registro de progreso físico con fotos, peso y comentarios.
- Visualización de evolución en gráficos.
- Consulta nutricional de alimentos vía API externa (OpenFoodFacts).
- Recálculo de macros según porción personalizada (por ejemplo, 150 g).
- Creación de dietas divididas en comidas (desayuno, comida, cena...) con alimentos importados.
- Estructura modular, escalable y diseño responsive.

2. Debilidades del proyecto

- Gestión de archivos multimedia (imágenes y vídeos), que puede requerir mucho almacenamiento y ancho de banda en servicios gratuitos.
- Riesgo de retrasos por una planificación poco detallada.
- Experiencia limitada en despliegue completo de proyectos individuales.
- Dependencia del ritmo de trabajo personal para cumplir con los plazos establecidos.

3. Fortalezas del Proyecto

- Alto valor práctico para usuarios interesados en fitness y gimnasio.
- Uso de tecnologías modernas con amplia documentación y soporte comunitario (React, FastAPI, PostgreSQL).
- Arquitectura escalable que permite futuras integraciones con funcionalidades como comunidades, inteligencia artificial o dispositivos wearables.

4. Objetivos

- Desarrollar una aplicación web responsive, accesible tanto desde escritorio como desde dispositivos móviles.
- Diseñar y conectar una base de datos relacional bien estructurada, utilizando PostgreSQL.
- Implementar el backend mediante FastAPI y Uvicorn, siguiendo una arquitectura modular y escalable.
- Gestionar el código fuente con Git y alojar el proyecto en GitHub, fomentando el control de versiones.
- Realizar el despliegue del backend en Railway o, alternativamente, mediante contenedores Docker.
- Elaborar la documentación técnica y los prototipos visuales en Figma, con el fin de facilitar la defensa del Trabajo de Fin de Ciclo.

5. Fases y subfases del proyecto

- Fase 1 – Planificación y Diseño (marzo – abril)
    - Análisis inicial de requisitos y definición de funcionalidades principales.
    - Elaboración de wireframes y prototipos en Figma.
    - Estructuración del proyecto y creación del repositorio Git.
- Fase 2 – Preparación de Entornos
    - Instalación de entornos virtuales (Python 3.11, Node.js 18).
    - Configuración de Docker y contenedor PostgreSQL para el entorno local.

    - Inicialización del repositorio en GitHub y organización de carpetas backend/frontend.
- Fase 3 – Desarrollo Backend
    - Implementación de FastAPI con SQLModel como ORM.
    - Modelado completo de la base de datos relacional.
    - Creación de endpoints RESTful (CRUD) para todas las entidades.
    - Desarrollo de la lógica para sesiones generadas a partir de rutinas (copia dinámica de ejercicios)
- Fase 4 – Desarrollo Frontend
    - Fase aún no desarrollada en el momento de redacción del presente documento).
- Fase 5 – Integración y Despliegue
    - Pruebas automáticas con pytest y validación manual vía Swagger.
    - Despliegue del backend en Railway (entorno cloud temporal).
    - Elaboración de la documentación técnica y anexos para la defensa del TFC.

6. Temporalización

Fase	**Duración**	**Periodo**
Planificación y Diseño	3 semanas	Marzo – 1ª quincena Abril
Preparación de entornos	2 semanas	1ª – 2ª quincena Abril
Desarrollo Backend	4 semanas	Finales Abril – Mayo
Desarrollo Frontend	3 semanas	Mayo
Integración, Pruebas y Despliegue	4 semanas	Finales Mayo – 2ª quincena Junio

7. Medios a emplear

Hardware:

- Ordenador con al menos 16 GB de RAM y disco SSD.
- Conexión a Internet estable.

Software:

- Visual Studio Code
- Docker Desktop
- PostgreSQL
- FastAPI, React, Tailwind CSS
- Figma (diseño de interfaz)
- Git + GitHub

Servicios externos:

- Railway (backend)
- Vercel (frontend)
- Buscador OpenFoodFacts (API de alimentos)

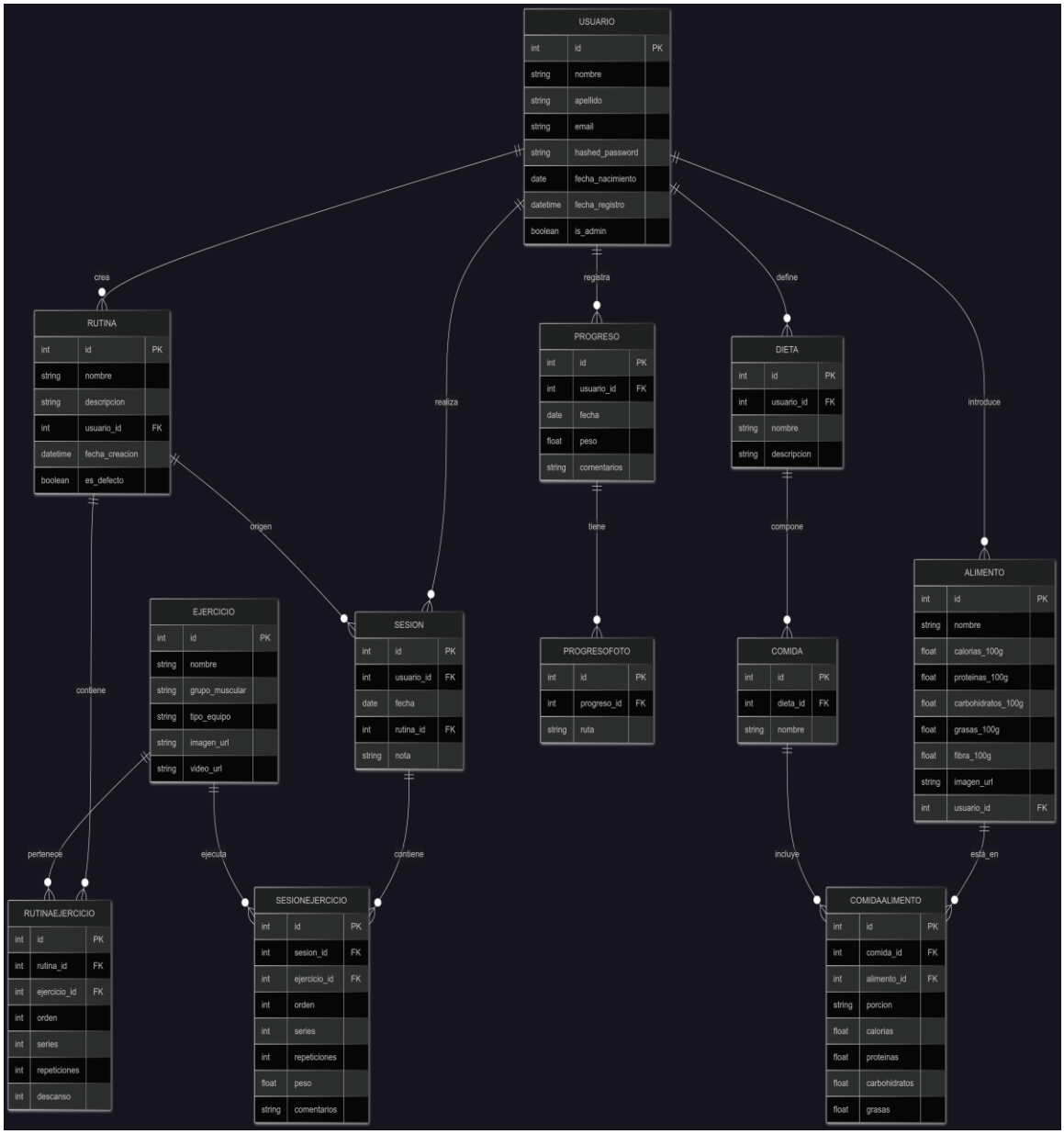
8. Parte desarrollada

Durante esta fase se ha llevado a cabo:

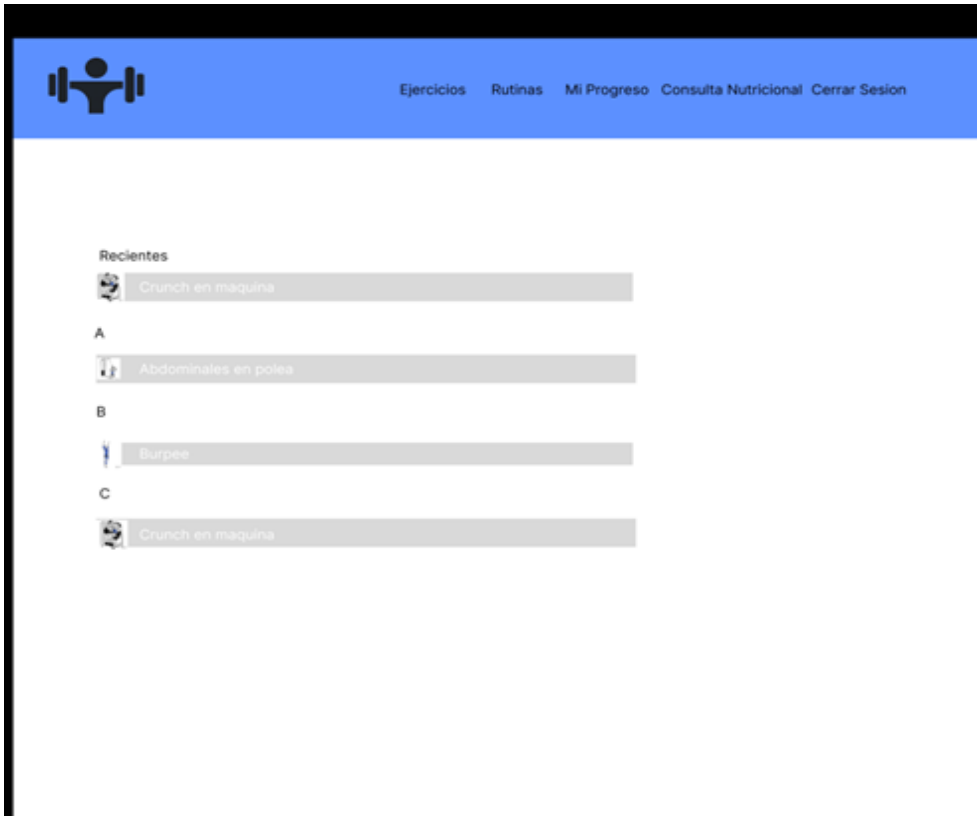
- Configuración del entorno virtual con Python 3.11.
- Instalación de FastAPI, Uvicorn, SQLModel, psycopg2, httpx, python-multipart y pytest.
- Configuración de Docker Compose para PostgreSQL.
- Modelado y creación de las relaciones entre las entidades: Usuario, Ejercicio, Rutina, RutinaEjercicio, Sesion, SesionEjercicio, Progreso, ProgresoFoto y Alimento.
- Desarrollo de endpoints CRUD completos para:
    - Usuarios con autenticación JWT.
    - Ejercicios con filtros por grupo muscular y tipo de equipo.
    - Rutinas con asociación dinámica de ejercicios.
    - Sesiones generadas a partir de rutinas, permitiendo editar repeticiones reales, peso y comentarios.
    - Progresos físicos, con subida de hasta 10 fotos por fecha, edición y eliminación.
    - Alimentos, con integración para búsqueda externa desde OpenFoodFacts.
- Implementación de tests automáticos con pytest, incluyendo pruebas completas para usuarios, alimentos y progresos con fotos.
- Preparación de rutas seguras con dependencias y verificación de permisos.
- Organización de una carpeta de servicios para las integraciones externas (por ejemplo, openfood.py).

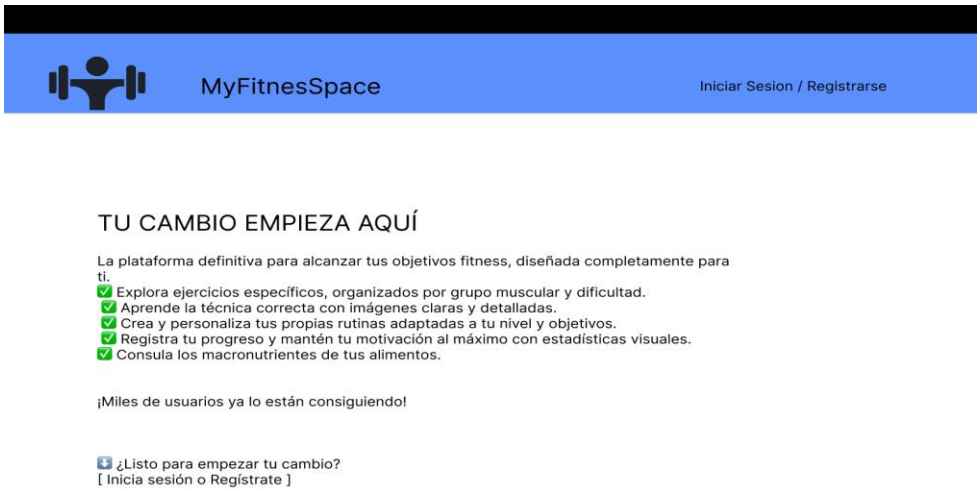
9. Anexos

- Diagrama entidad-relación del backend



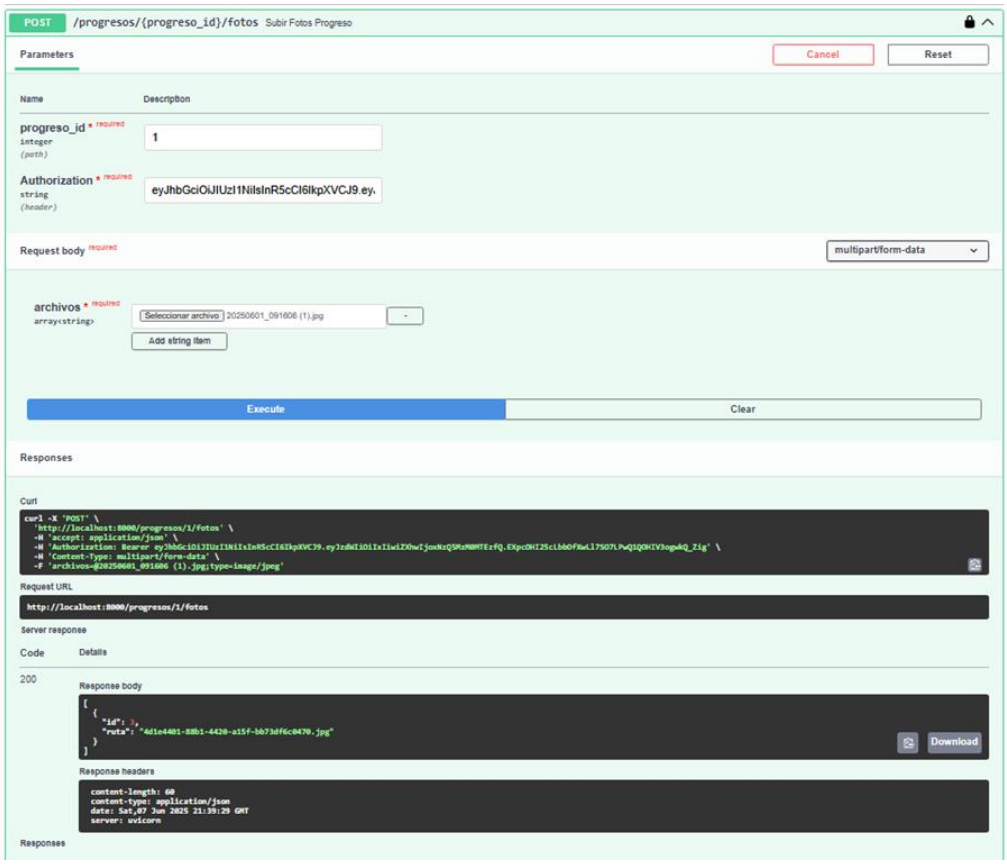
- Figma de la pestaña ‘Inicio’ y 'Ejercicios'



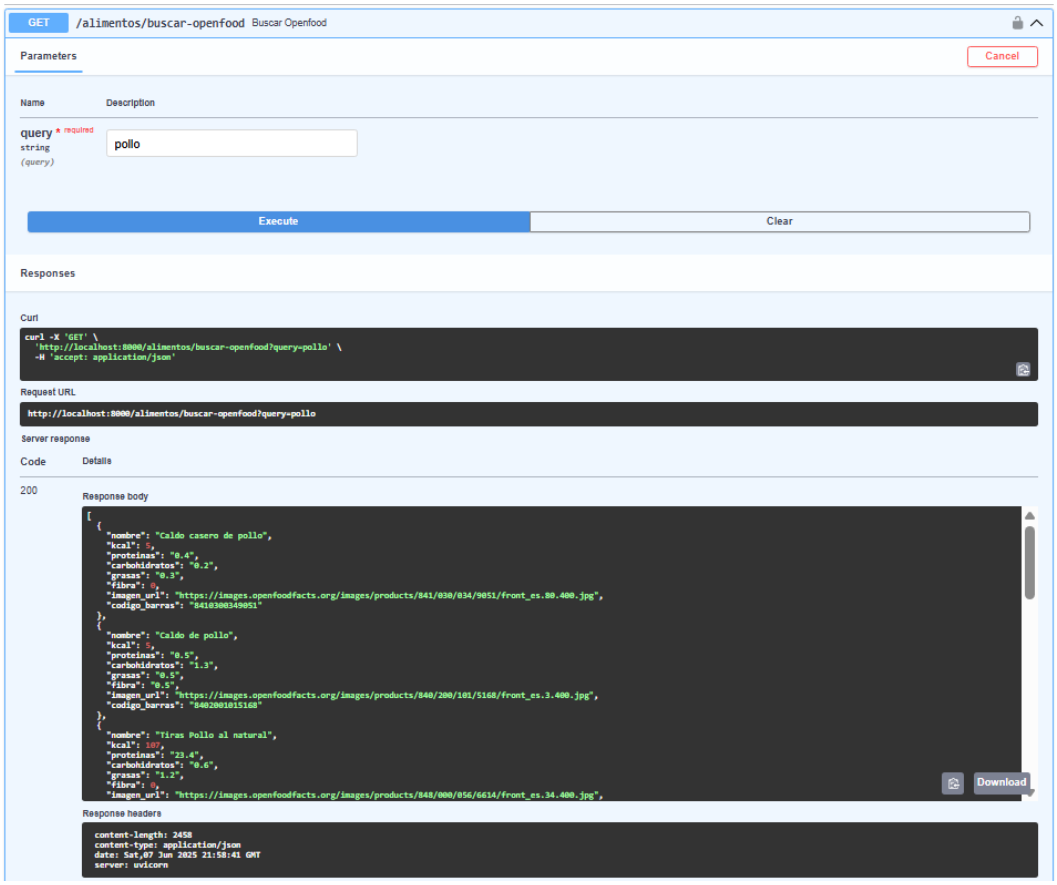


- Vista general de endpoints generados con Swagger

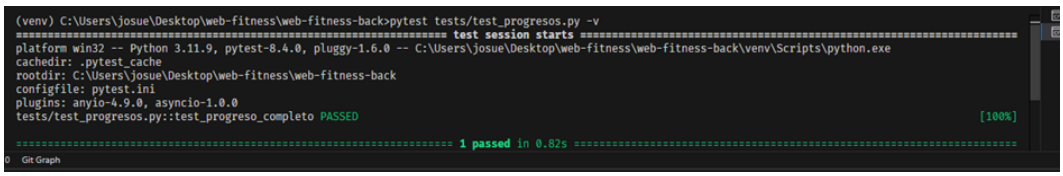
- Prueba de subida de fotos de progreso



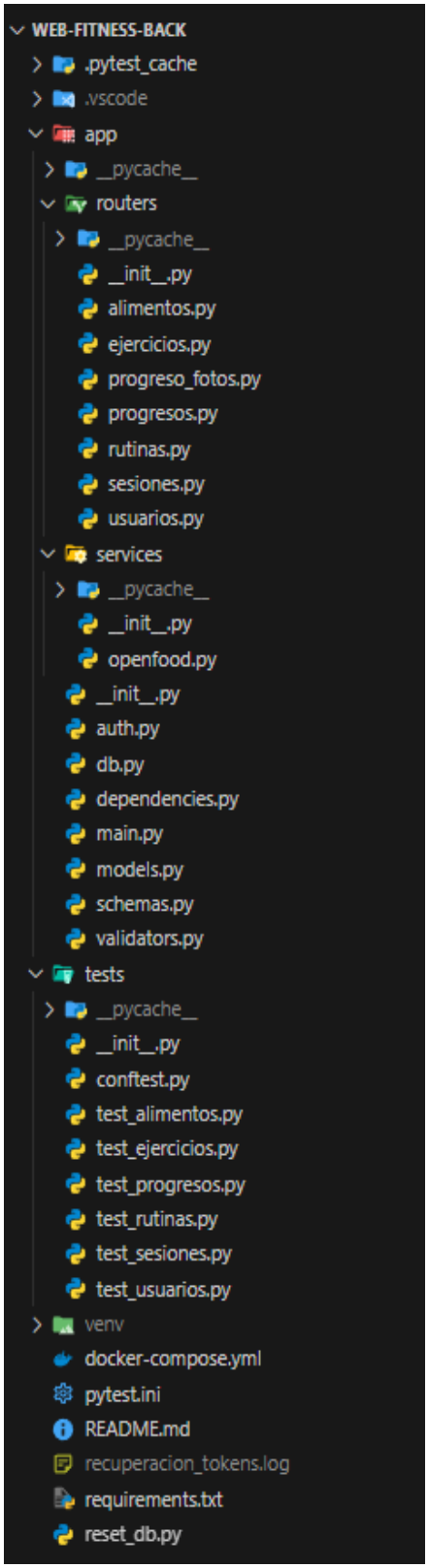
- Prueba del endpoint /alimentos/buscar-openfood



- Resultado del test automático con pytest



- Estructura de carpetas del backend



10. Presupuestos

- Hardware utilizado:

Componente	**Precio estimado**	**Unidades**	**Total**
Procesador Intel Core i5-12400F	159,99 €	1	159,99 €
Placa base ASUS Prime B760-PLUS D4	124,99 €	1	124,99 €
RAM Corsair Vengeance LPX 16GB (2x8GB) DDR4	39,99 €	1	39,99 €
SSD Samsung 970 EVO Plus 1TB NVMe M.2	72,99 €	1	72,99 €
Fuente Corsair RM750e 750W 80 Plus Gold	136,98 €	1	136,98 €
Torre Tempest Umbra RGB ATX	50,99 €	1	50,99 €
Refrigeración Noctua NH-U9S	69,90 €	1	69,90 €
GPU MSI AMD Radeon RX 6650 XT 8GB	229,90 €	1	229,90 €
Monitor BenQ RL2455	220 €	1	220 €
Monitor AOC 24G2W1G8	60 €	1	60 €

- Total: 1.165,73 €

- Software

Herramienta	**Tipo de licencia**	**Precio estimado**
Visual Studio Code	Libre (Open Source)	0 €
Docker Desktop	Libre para uso personal	0 €
PostgreSQL	Libre	0 €
FastAPI, SQLModel	Libre	0 €
Figma (plan gratuito)	Gratuito	0 €
Git y GitHub	Libre / plan estudiante	0 €
Pytest	Libre	0 €
DataGrip (licencia EDU)	Gratuito (educacional)	0 €

- Horas de desarrollo

Fase	**Horas estimadas**
Backend	
Planificación y diseño	10 h
Preparación de entornos	8 h
Desarrollo backend	45 h
Pruebas automáticas	10 h
Documentación técnica	7 h

Subtotal backend	**80 h**
Frontend (estimado)	
Creación de wireframes (Figma)	3 h
Maquetación con Tailwind	10 h
Desarrollo de componentes	12 h
Integración con backend	10 h
Pruebas e interfaz responsive	5 h
Subtotal frontend (previsto)	40 h
Total horas	120 h

- Total: 120 h × 12 €/h = 1.440 €

TOTAL: 2.605,73 €

11. Biliografía

- SQLModel – Documentación oficial: https://sqlmodel.tiangolo.com/
- FastAPI – Guía de uso: https://fastapi.tiangolo.com/
- PostgreSQL – Manual oficial: https://www.postgresql.org/docs/
- Pytest – Pruebas automatizadas: https://docs.pytest.org/en/stable/
- OpenFoodFacts – API y datasets: https://es.openfoodfacts.org/data
- Figma – Manual de usuario: https://help.figma.com/hc/es-es
- Stack Overflow, GitHub Discussions y foros técnicos como referencias puntuales de resolución de errores.