**TRABAJO FINAL DE CICLO**

**DESARROLLO APLICACIONES WEB**



Autor: **ALEJANDRO SANTOS CABRERA**

Tutor:

**FERNANDO PRADO**

**13/06/2025**

Contenido

[1. Introducción (Idea del proyecto) 2](#_Toc200963771)

[2. Debilidades del proyecto 2](#_Toc200963772)

[3. Fortalezas del Proyecto 3](#_Toc200963773)

[4. Objetivos 3](#_Toc200963774)

[5. Fases y subfases del proyecto 3](#_Toc200963775)

[6. Temporalización 4](#_Toc200963776)

[7. Medios a emplear 5](#_Toc200963777)

[8. Parte desarrollada 6](#_Toc200963778)

[9. Despliegue 10](#_Toc200963779)

[10. Anexos 13](#_Toc200963780)

[11. Presupuestos 21](#_Toc200963781)

[12. Bibliografía 24](#_Toc200963782)

# 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.

# Debilidades del proyecto

* Gestión de archivos multimedia (imágenes), 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.

# 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.

# 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.

# 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.

# 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 |

# 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)

# Parte desarrollada

**Desarrollo Backend**

La API se ha desarrollado utilizando **FastAPI** con **Python 3.11** y el ORM **SQLModel**, aprovechando su integración con Pydantic y SQLAlchemy. El backend sigue una arquitectura modular, escalable y protegida mediante autenticación JWT. Todo el entorno se levanta en local con **Docker Compose** y una base de datos **PostgreSQL**.

**Configuración inicial**

* Entorno virtual (venv) activado con Python 3.11.
* Instalación de dependencias principales:  
  fastapi, uvicorn, sqlmodel, psycopg2-binary, httpx, python-multipart, pytest.
* Contenedor de PostgreSQL configurado vía docker-compose.yml.
* Inicialización del proyecto con estructura profesional (app/, routers/, models/, schemas/, validators/, etc.).

**Autenticación JWT**

* Implementación de sistema completo de login con JWT (/login) y verificación de token (/verify-token).
* Protección de rutas mediante dependencias (get\_current\_user).
* Recuperación de contraseña con **token temporal por correo** (/usuarios/recuperar, /usuarios/restablecer).
* Hashing de contraseñas con passlib.

**Gestión de usuarios**

* Registro, login, lectura, edición y eliminación.
* Restricción de acceso a datos personales (sólo el usuario puede editar su perfil).
* Los administradores pueden ver y eliminar a otros usuarios.
* Validaciones avanzadas: campos vacíos, correo válido, contraseña segura, edad mínima.

**Ejercicios**

* Endpoints para crear, listar, filtrar y eliminar ejercicios.
* Filtros por grupo muscular, tipo de equipo y nombre (usados por el frontend).
* Soporte para múltiples fotos por ejercicio (EjercicioFoto).
* Control de permisos: los ejercicios públicos no pueden editarse ni eliminarse.

**Rutinas**

* Creación de rutinas personalizadas o copia de rutinas predefinidas.
* Asociación dinámica de ejercicios a rutinas (RutinaEjercicio) con:
  + Orden
  + Comentarios
  + Series personalizadas (RutinaSerie)
* Edición y eliminación de ejercicios o series dentro de la rutina.
* Cambio del orden visual con validaciones (sin duplicados, sin ceros).

**Sesiones**

* Generación automática de una **sesión de entrenamiento** a partir de una rutina.
* Copia completa de los ejercicios, series, repeticiones y comentarios.
* Registro de peso real, repeticiones efectivas y reordenamiento por drag-and-drop.
* Función para **actualizar la rutina base** con los datos reales de una sesión completada.

**Progreso físico**

* Registro de peso corporal, comentarios y fotos (hasta 10 por fecha).
* Gestión segura de archivos (formato, límite, ruta física).
* Edición o eliminación de registros.
* Eliminación automática de fotos físicas al borrar un progreso.

**Consulta de alimentos**

* Búsqueda de alimentos reales en la API externa de **OpenFoodFacts**.
* Almacenamiento de alimentos seleccionados, incluyendo macros, porción e imagen.
* Posibilidad de crear alimentos personalizados desde el frontend.

**Tests automáticos**

* Pruebas unitarias con **pytest** para:
  + Usuarios (registro, login, verificación)
  + Alimentos (búsqueda, creación, permisos)
  + Progresos físicos (subida de fotos, validaciones)
* Se ejecutan localmente antes del despliegue.

**Desarrollo Frontend**

Para la parte cliente se ha utilizado **React.js con TypeScript** y **Tailwind CSS** para los estilos. El proyecto se ha creado con **Vite** por su velocidad y simplicidad en el entorno de desarrollo.

Se han implementado las siguientes vistas y funcionalidades:

**Autenticación de usuarios**

* Componente Registro.tsx: formulario con validación para crear una cuenta (nombre, apellido, email, fecha de nacimiento, contraseña, confirmación).
* Componente Login.tsx: formulario para iniciar sesión con JWT. Si el usuario ya tiene sesión activa, se redirige automáticamente al Dashboard.

Ambos formularios muestran errores y estados de carga para mejorar la experiencia de usuario.

**Dashboard principal**

* Dashboard.tsx: vista de bienvenida tras el login. Contiene accesos rápidos mediante tarjetas a:
  + Ejercicios
  + Rutinas
  + Historial de sesiones
  + Mi progreso
  + Consulta nutricional

**Gestión de ejercicios (Ejercicios.tsx)**

* Visualización de todos los ejercicios disponibles (públicos y propios del usuario).
* Filtros dinámicos por grupo muscular, tipo de equipo y nombre.
* Miniaturas con imagen representativa si está disponible.
* Posibilidad de añadir ejercicios personalizados desde la interfaz (pendiente de implementación).

**Gestión de rutinas (Rutinas.tsx)**

* Vista con rutinas propias y ejemplos predefinidos.
* Posibilidad de crear una nueva rutina personalizada.
* Visualización del nombre, descripción y tipo (ejemplo o personal).

**Sesión de entrenamiento (SesionDetalle.tsx)**

* Permite **iniciar una sesión a partir de una rutina**.
* Se copian los ejercicios y se registran:
  + Peso utilizado
  + Repeticiones realizadas
  + Series (editables)
  + Comentarios y estado de completado
* El usuario puede reordenar los ejercicios con **drag-and-drop** (@hello-pangea/dnd).
* Al finalizar, se puede **actualizar la rutina original** con los datos reales.

**Mi Progreso (Progreso.tsx)**

* Permite añadir registros de peso, comentarios y subir fotos (hasta 10 por fecha).
* Visualización por fecha, miniaturas de imágenes y acceso a los detalles.

**Consulta Nutricional (ConsultaNutricional.tsx)**

* Búsqueda de alimentos reales usando la **API de OpenFoodFacts**.
* Muestra:
  + Nombre y marca del producto
  + Imagen
  + Calorías y macronutrientes por 100 g
* Preparado para integrar la selección del alimento dentro de una dieta personalizada.

# Despliegue

El despliegue de la plataforma web se ha realizado de forma separada para el frontend y el backend, empleando servicios en la nube que permiten una publicación sencilla, gratuita y funcional para proyectos académicos. Se han utilizado tecnologías modernas compatibles con entornos de desarrollo y producción.

**Frontend (React + Vite)**

El frontend, desarrollado con React, TypeScript, Tailwind CSS y Vite, se ha desplegado en el servicio Vercel. Este proveedor ofrece integración directa con GitHub, dominio seguro con HTTPS, CDN global y actualizaciones automáticas tras cada cambio en el repositorio.

* Proveedor: Vercel (<https://vercel.com>)
* Repositorio: web-fitness-front
* URL pública del frontend:  
  https://web-fitness-ashen.vercel.app
* Variables de entorno configuradas mediante VITE\_API\_URL para apuntar al backend.

El sistema incluye protección de rutas mediante JWT, control de acceso a páginas privadas y carga de datos desde el backend desplegado. La interfaz es responsive y se adapta a dispositivos móviles y de escritorio.

**Backend (FastAPI + PostgreSQL)**

El backend se ha implementado en Python 3.11 utilizando el framework FastAPI y el ORM SQLModel. Para su despliegue se ha utilizado el servicio Railway, que permite ejecutar contenedores web y base de datos en la nube con soporte para entornos virtuales.

* Proveedor: Railway (<https://railway.app>)
* Repositorio: web-fitness-back
* URL pública del backend:  
  https://web-fitness-production.up.railway.app
* Base de datos PostgreSQL alojada también en Railway.

Se han configurado las rutas del backend con autenticación JWT, dependencias de seguridad, endpoints CRUD y rutas protegidas. El entorno cuenta con un archivo requirements.txt que contiene todas las dependencias necesarias.

Además, se ha montado la carpeta progreso\_fotos como ruta estática mediante StaticFiles para permitir el acceso a las imágenes de progreso en desarrollo local. Sin embargo, se ha detectado que Railway no permite persistencia de archivos entre reinicios, lo que impide mantener las imágenes en producción. Como solución a largo plazo, se recomienda utilizar un sistema externo como Supabase Storage, Cloudinary o Firebase Storage para la gestión de archivos.

**Integración entre frontend y backend**

El frontend se comunica con el backend mediante peticiones HTTP protegidas con tokens JWT. Tras el inicio de sesión, se almacena el token en el contexto de autenticación y se incluye en todas las peticiones protegidas mediante la cabecera Authorization. La URL del backend se define en una variable de entorno (VITE\_API\_URL) y se adapta según el entorno (local o producción).

**Verificación del despliegue**

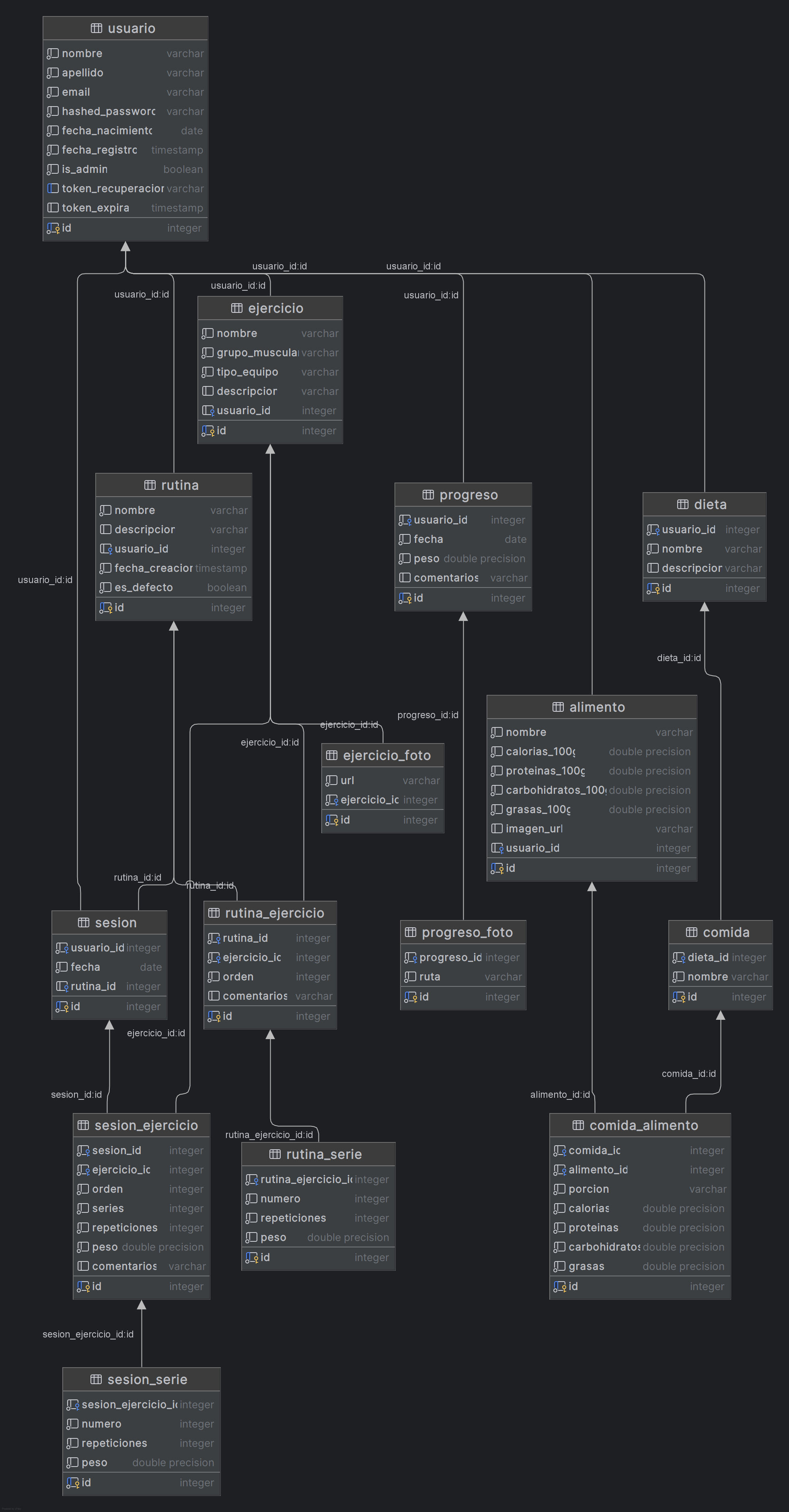
Se han realizado pruebas manuales para verificar el correcto funcionamiento del sistema en producción. Entre ellas se incluyen:

* Registro e inicio de sesión
* Navegación entre las vistas del usuario
* Consulta de ejercicios, creación de rutinas y gestión de sesiones
* Registro de progreso físico y comentarios
* Consulta de alimentos reales desde OpenFoodFacts

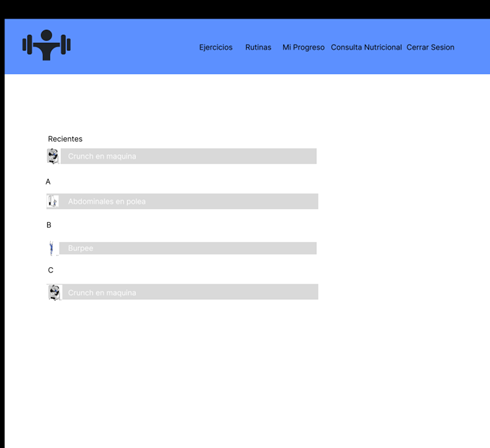
El resultado ha sido satisfactorio en todos los módulos, a excepción del sistema de imágenes, cuya persistencia queda limitada al entorno local hasta que se integre un almacenamiento externo.

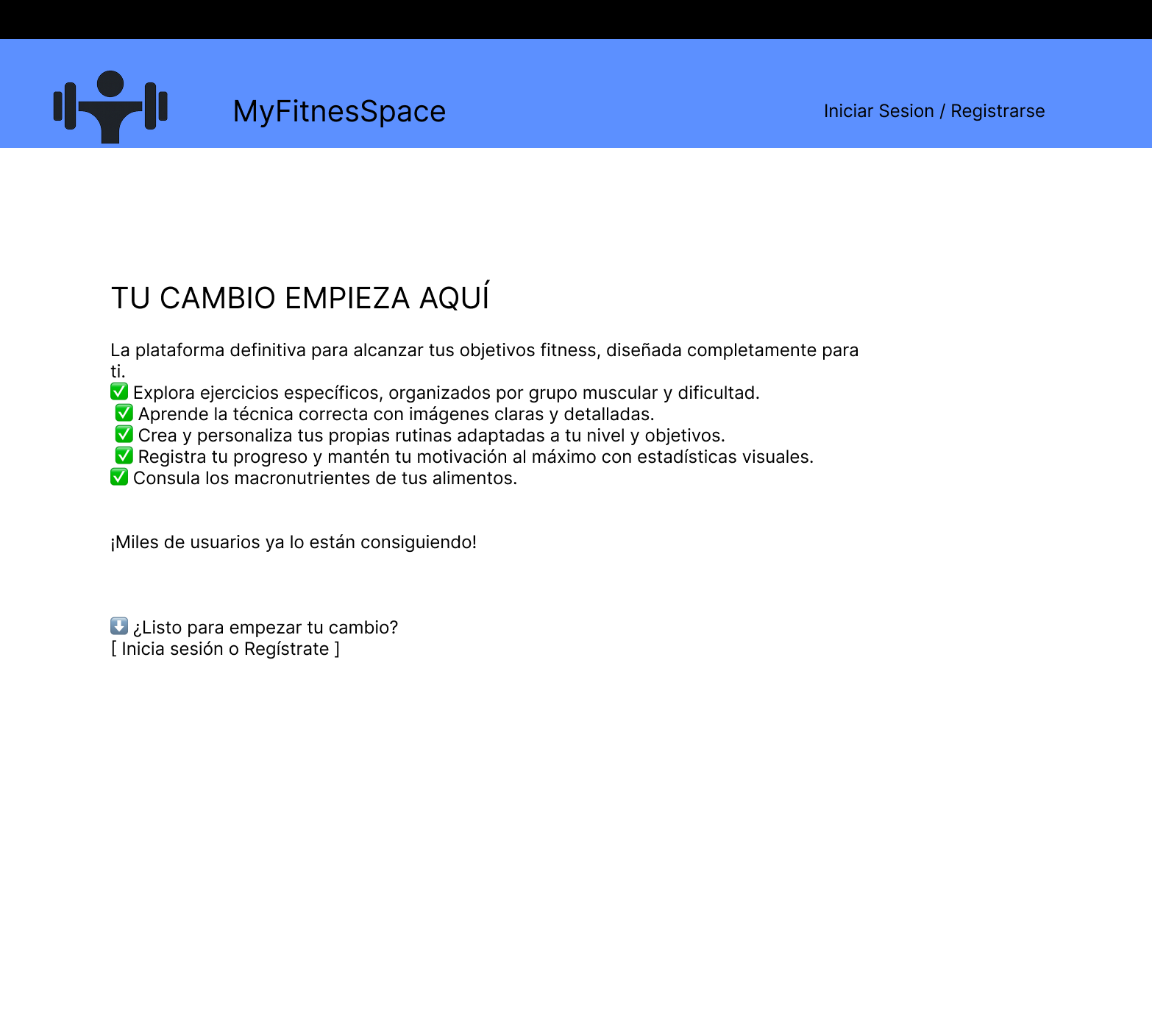
# Anexos

* Diagrama entidad-relación del backend



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

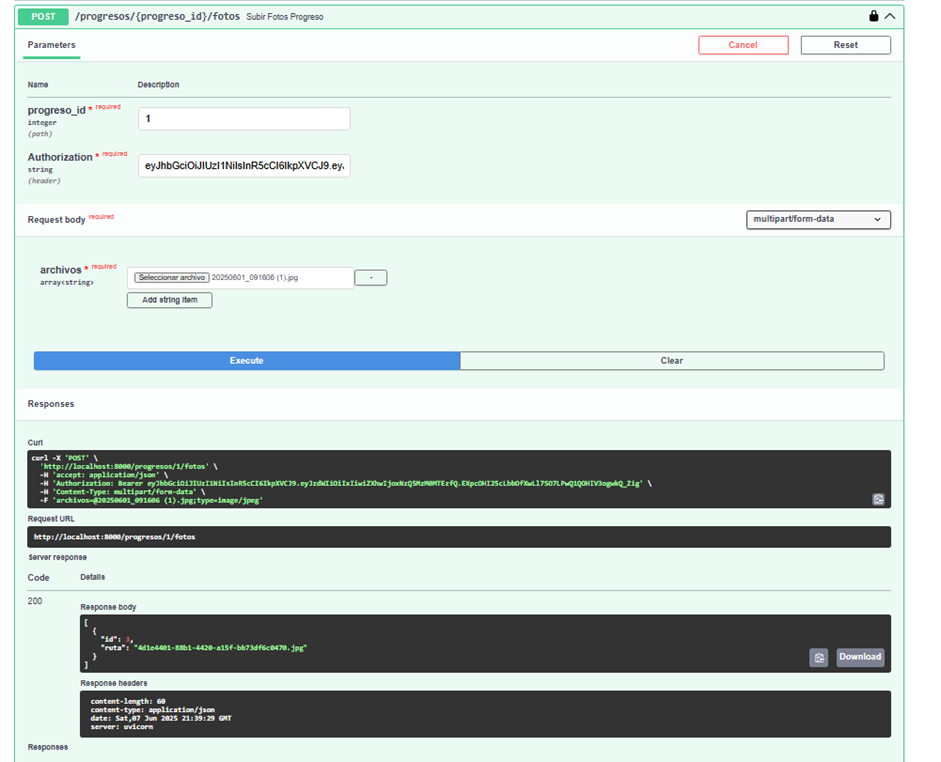




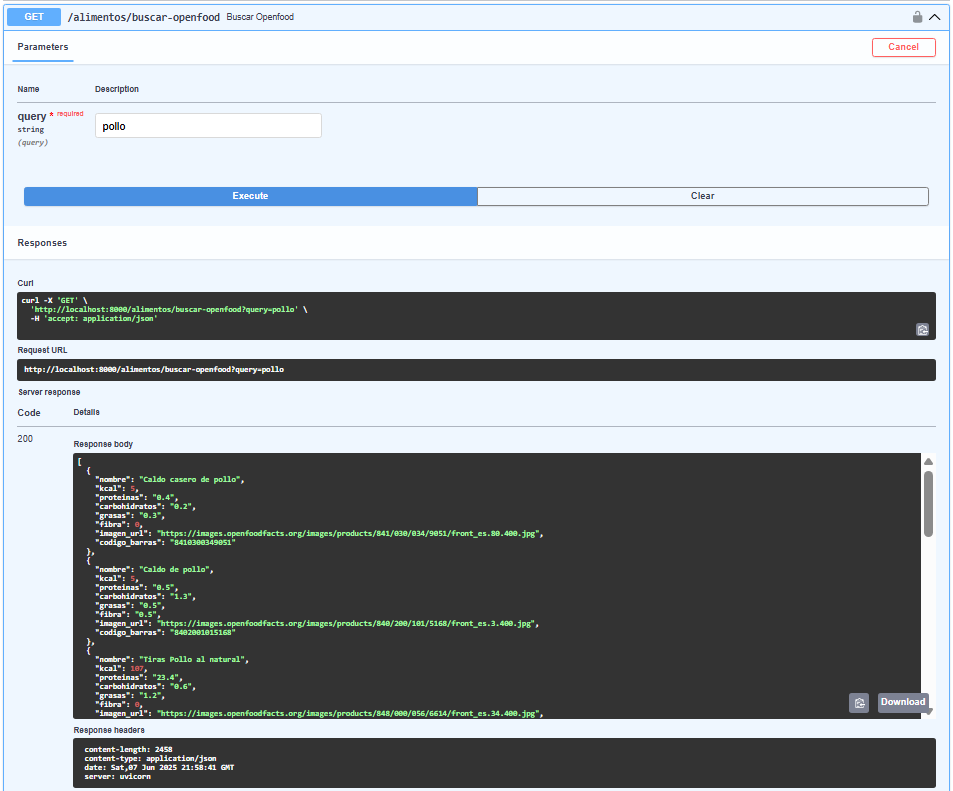
* Vista general de endpoints generados con Swagger



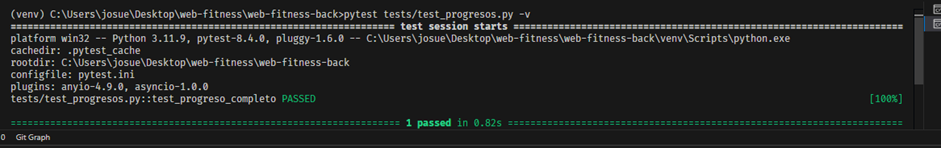
* Prueba de subida de fotos de progreso



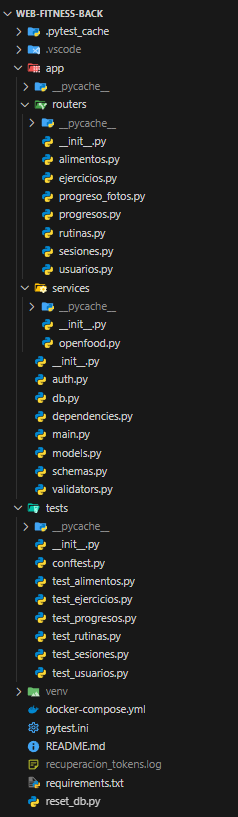
* Prueba del endpoint /alimentos/buscar-openfood



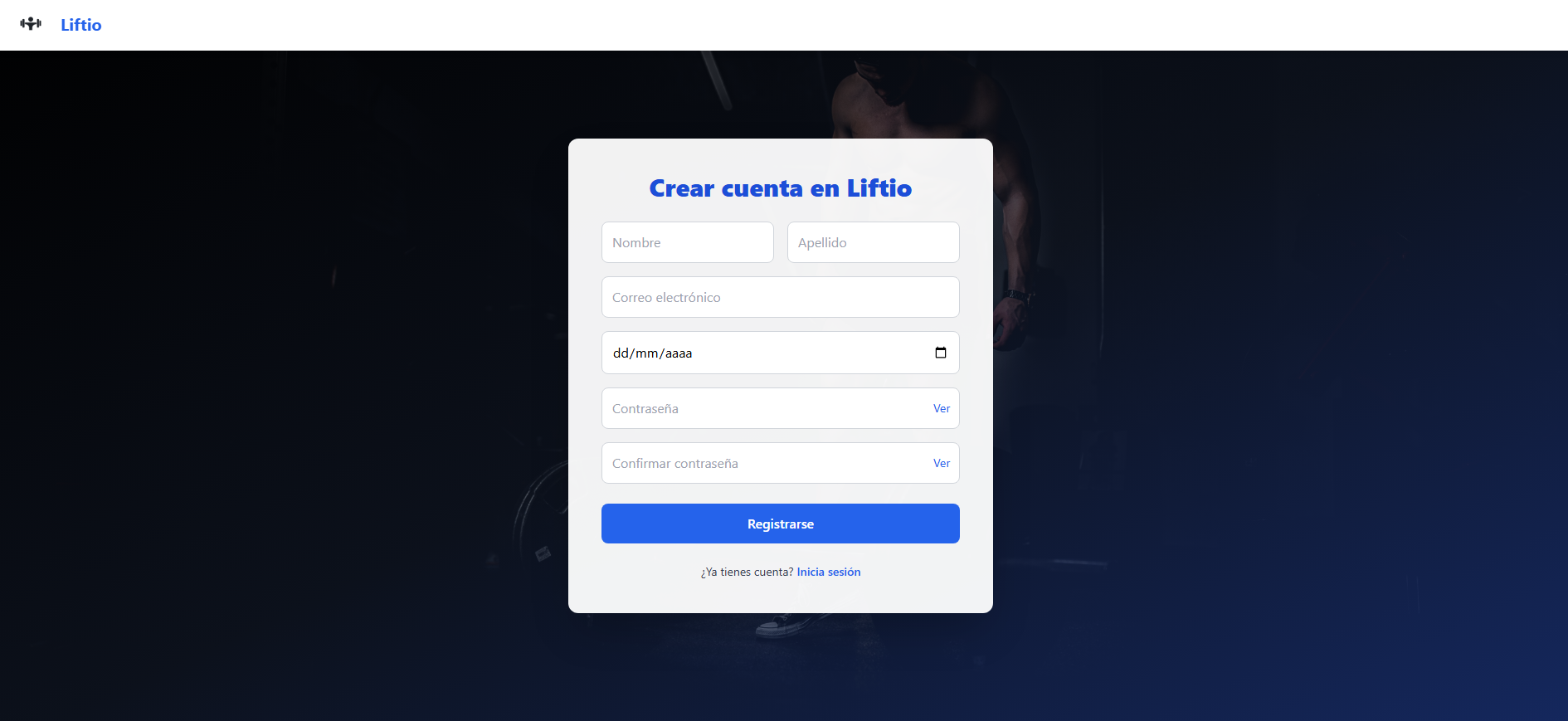
* Resultado del test automático con pytest



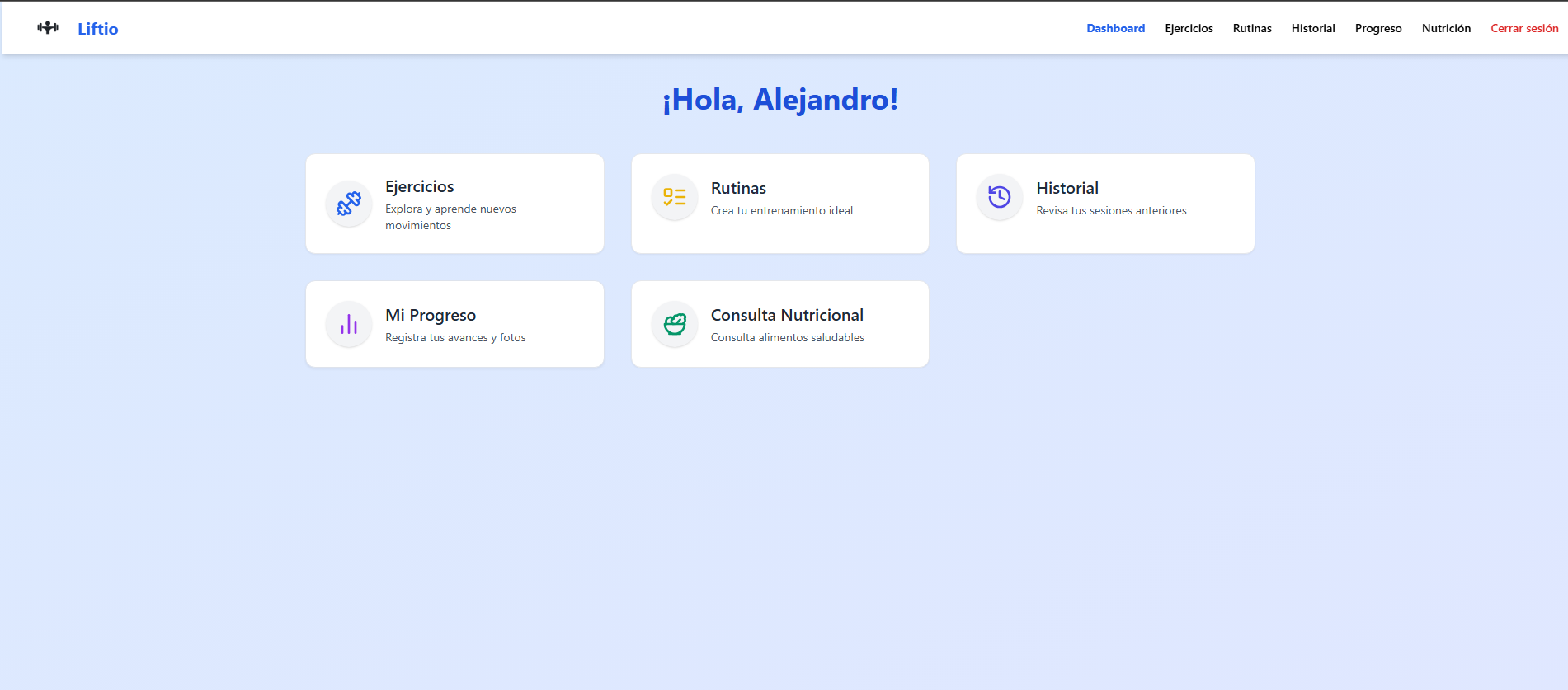
* Estructura de carpetas del backend



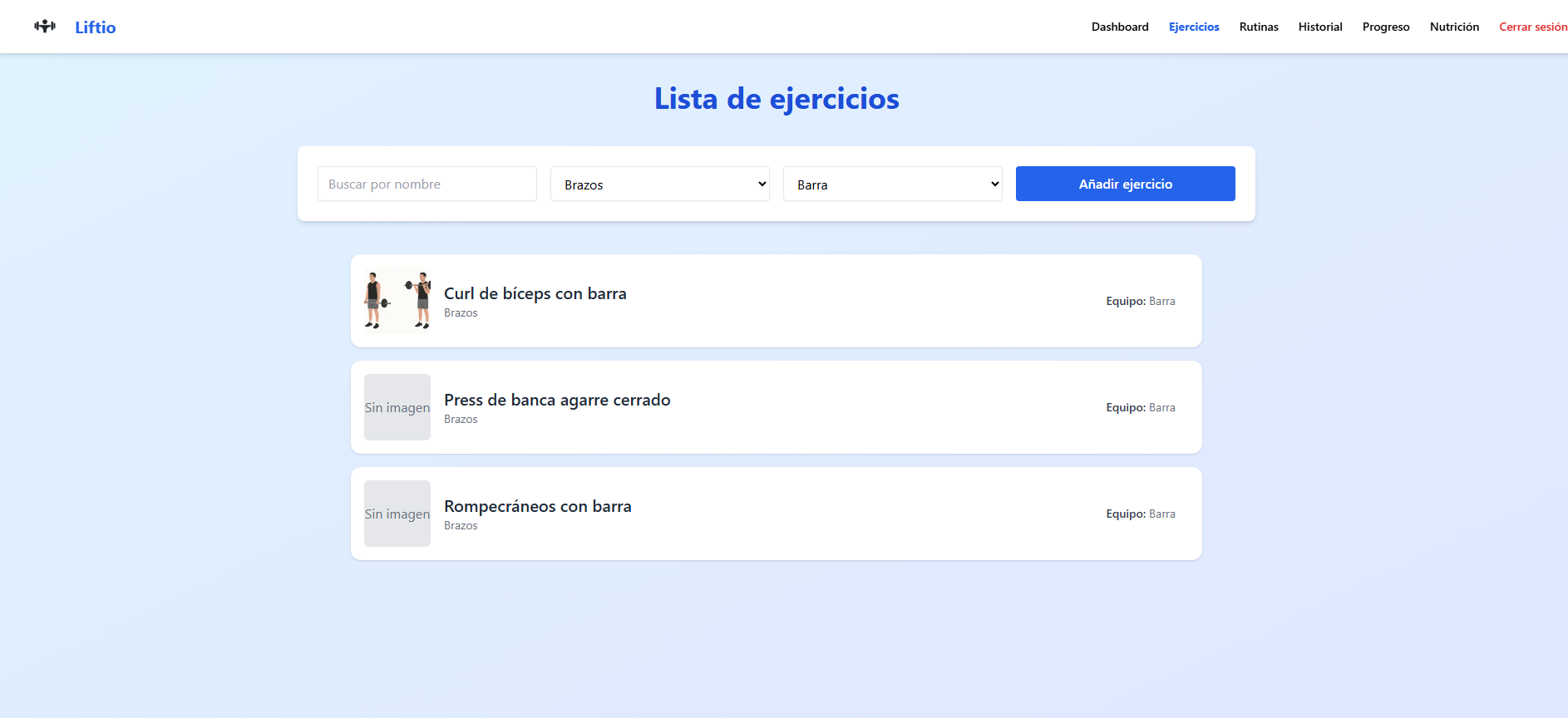
* Formulario de registro de usuario



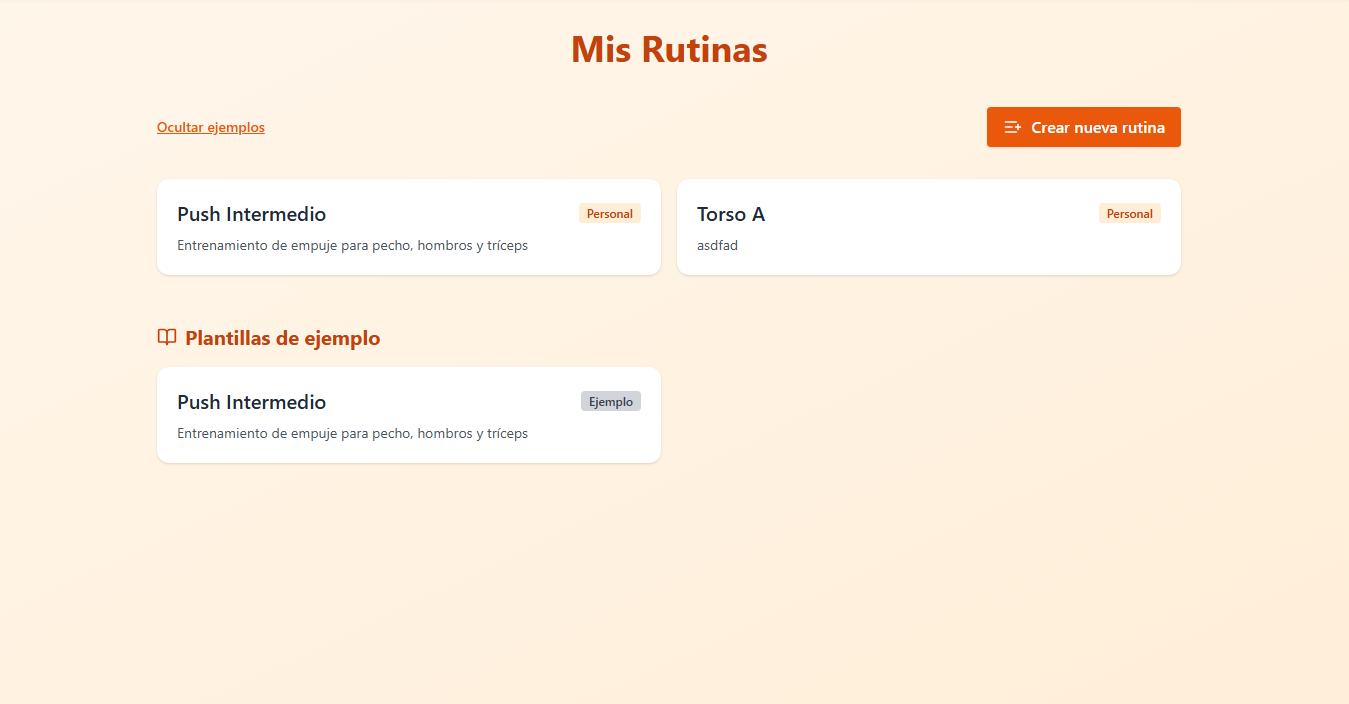
* Dashboard principal tras el login



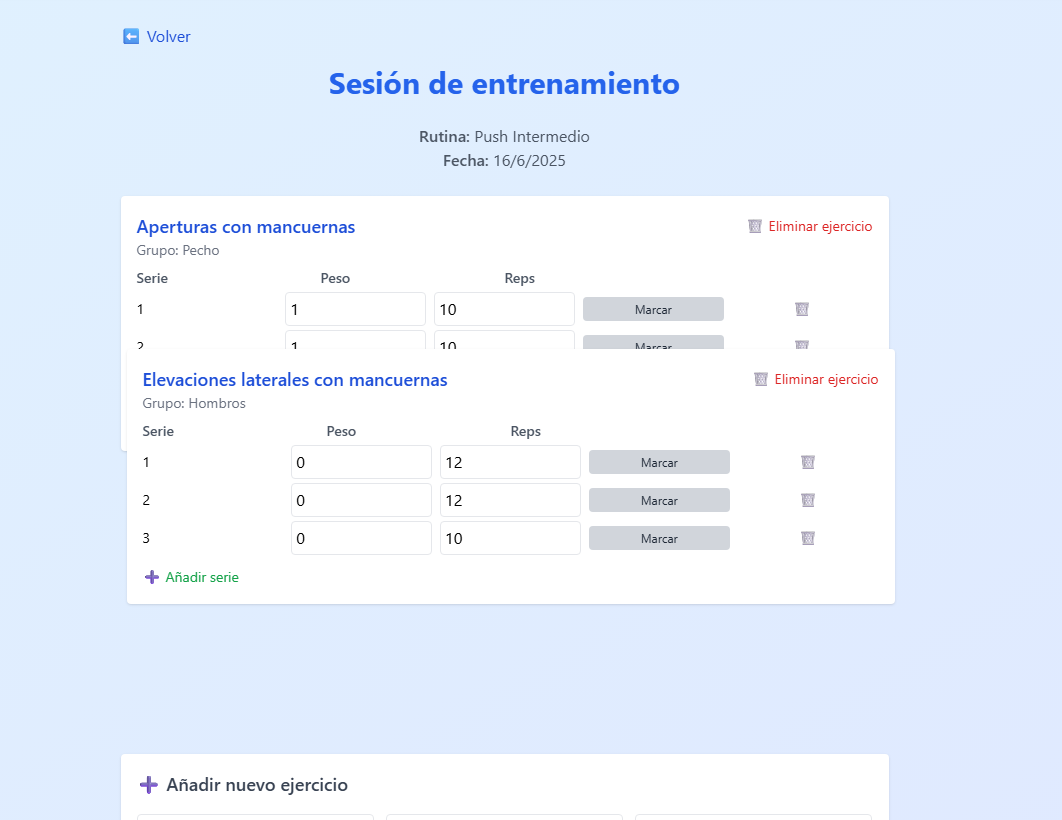
* Listado de ejercicios con filtros activos



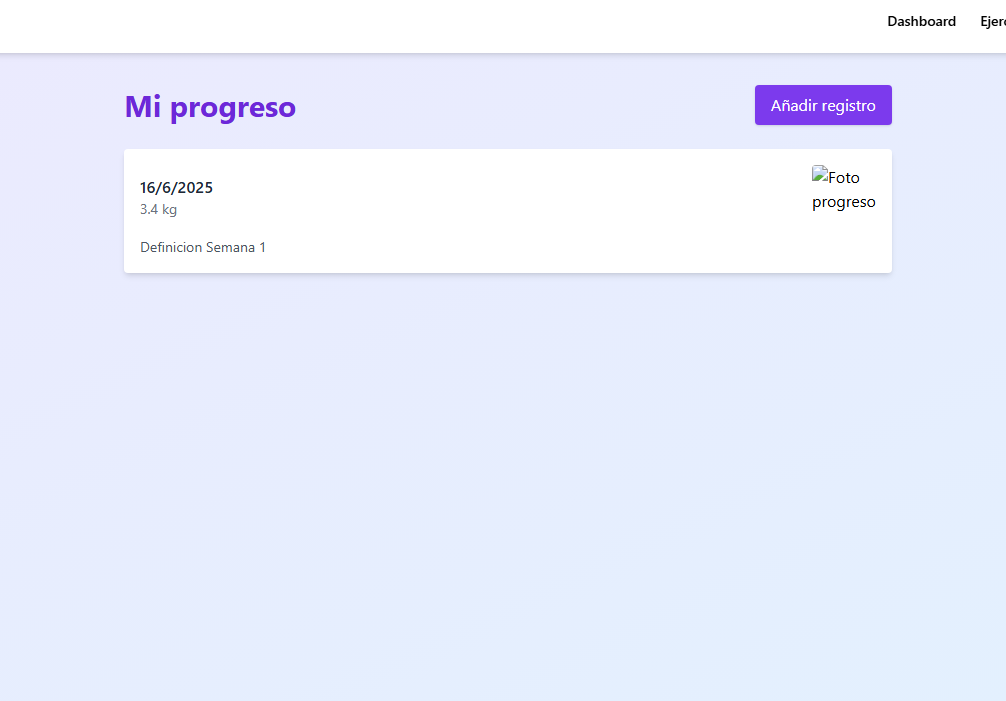
* Listado de rutinas personales



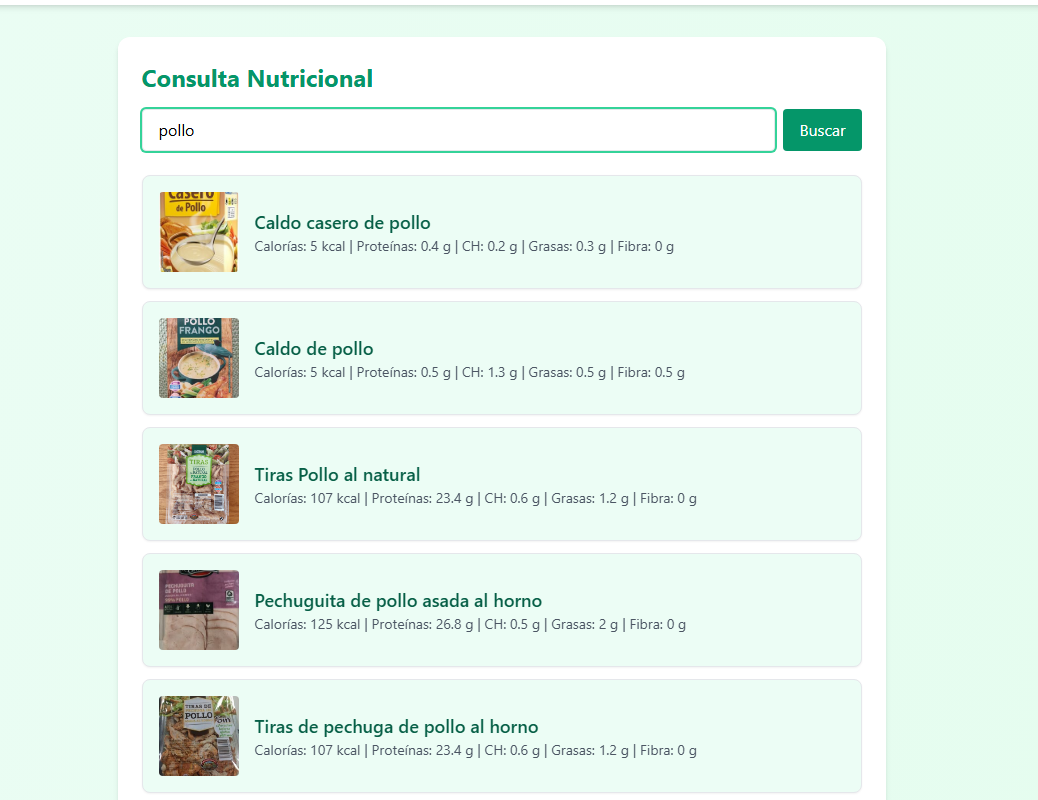
* Vista de sesión en curso con drag and drop



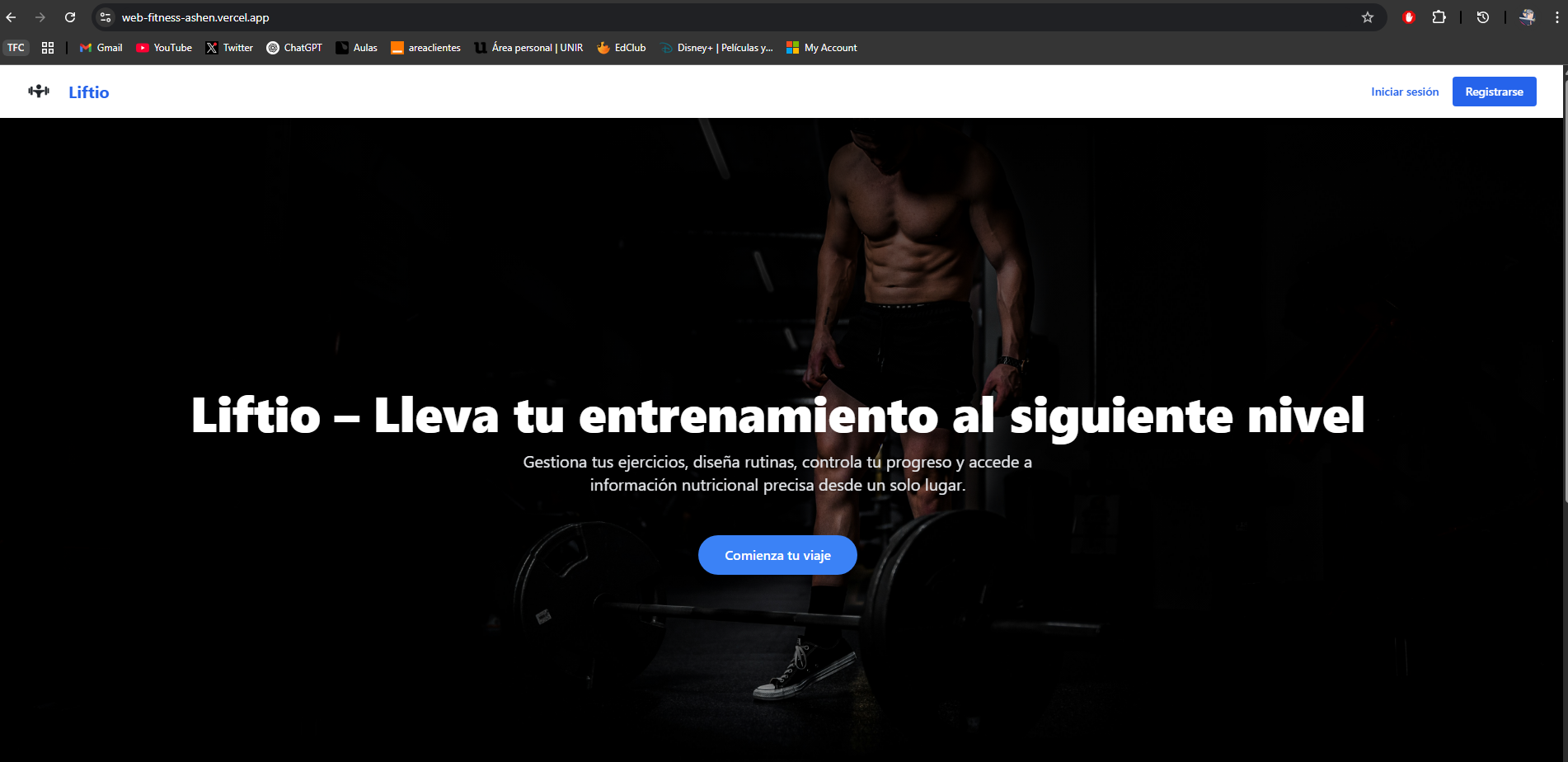
* Listado de registros de progreso físico



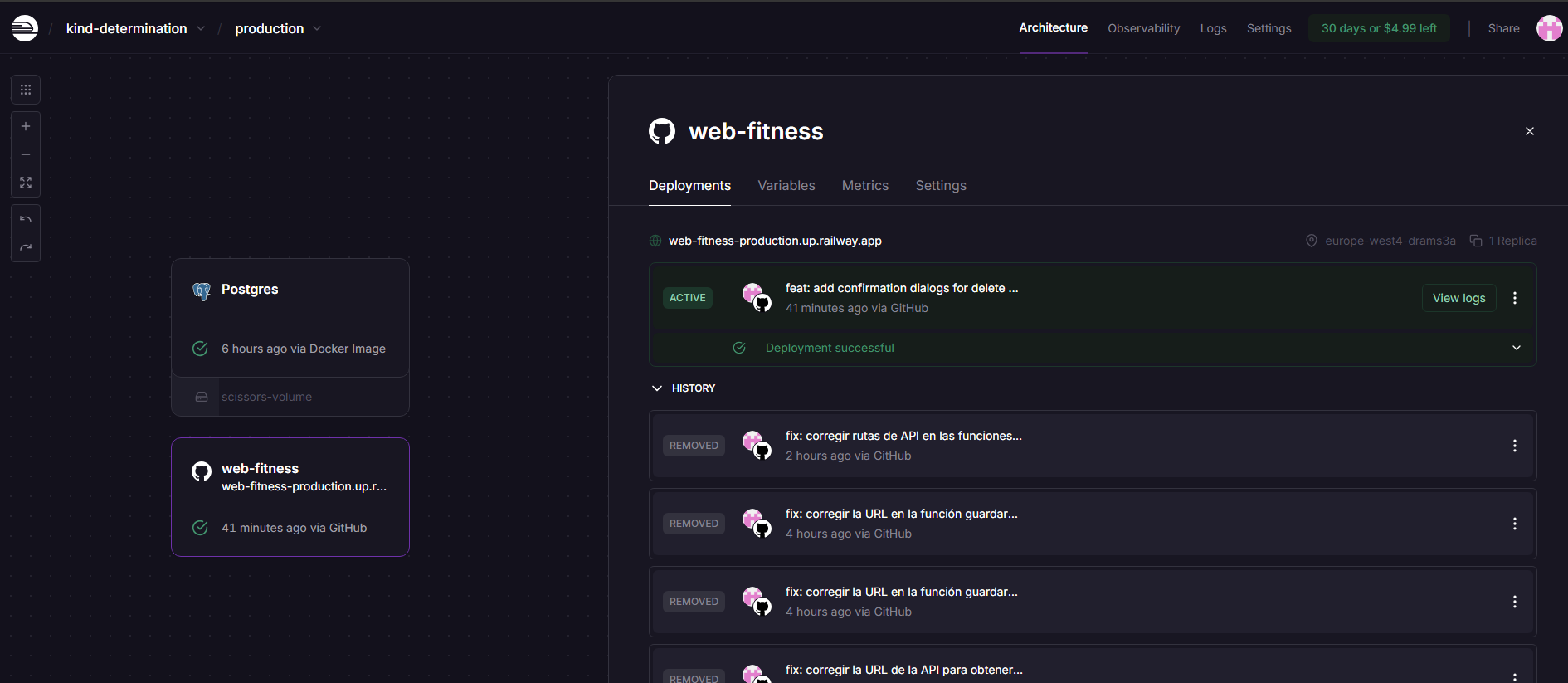
* Resultado de consulta a OpenFoodFacts



* Frontend desplegado en Vercel



* Panel del backend en Railway



# 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** |  |
| 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 €**

# Bibliografía

* 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.
* React – Documentación oficial: <https://react.dev/>
* TypeScript – Manual oficial: <https://www.typescriptlang.org/docs/>
* Vercel – Manual de despliegue: <https://vercel.com/docs>
* Railway – Documentación oficial: <https://docs.railway.app/>
* Git – Referencia rápida: <https://git-scm.com/docs>
* GitHub – Guía de uso y control de versiones: <https://docs.github.com/es>