Universidad Autónoma de Madrid

Escuela Politécnica Superior

23|24

Trabajo fin de grado

Plataforma de diseño y potenciación de atletas de alto rendimiento

Alejandro Monterrubio Navarro

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Plataforma de diseño y potenciación de atletas de alto rendimiento

Autor: Alejandro Monterrubio Navarro
Tutor: Pablo Cerro Cañizares

junio 2024

Alejandro Monterrubio Navarro
Plataforma de diseño y potenciación de atletas de alto rendimiento

Alejandro Monterrubio Navarro
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mis padres, mi hermano, mi novia y mis amigos por el amor y apoyo incondicional.
A mi tutor por su paciencia y motivación.
Y a todos ellos, por no dejar de confiar en mí.
Gracias

El poder viene en respuesta a una necesidad, no a un deseo. Tienes que crear esa necesidad.

Akira Toriyama

Agradecimientos

En primer lugar me gustaría agradecer a la Escuela Politécnica Superior por su apoyo para la creación de esta clase y que sea el formato básico para la creación de tesis, trabajos fin de grado y trabajos fin de master.

En particular quiero destacar el trabajo realizado por Fernando López-Colino por su apoyo en la comisión de imagen institucional y por sus comentarios para mejorar este estilo.

También quiero tener un recuerdo para Carmen Navarrete Navarrete dado que este estilo comencé a crearlo a partir de sus necesidades a la hora de escribir la tesis. Y por supuesto a no quiero olvidarme de mi esposa e hijos que han servido de conejillos de indias en sis correspondientes trabajos fin de master y de grado. No quiero olvidar a todos los estudiantes que me pidieron este estilo y lo han usado para presentar sus trabajos pero son muchos y podría olvidarme de alguno, por tanto, mi agradecimiento en general a todos ellos.

Resumen

La intersección entre el deporte y la nutrición es fundamental para mantener un estilo de vida saludable. Sin embargo, muchas personas enfrentan dificultades para integrar prácticas saludables en su rutina diaria debido a la falta de tiempo y conocimientos específicos. Este desafío a menudo conduce a la adopción de dietas genéricas o rutinas de ejercicio inadecuadas, lo que incrementa el riesgo de resultados negativos y lesiones. La principal barrera para adoptar un estilo de vida saludable radica en la falta de información relevante y accesible, así como en la limitación de tiempo para adquirir dichos conocimientos.

Reconociendo esta necesidad, mi Trabajo Final de Grado (TFG) se centra en simplificar el acceso a planes de nutrición y ejercicio personalizados. El objetivo es proporcionar a los usuarios, independientemente de su experiencia previa en deporte, las herramientas necesarias para mejorar su rendimiento o iniciar un camino hacia el bienestar físico, sin requerir una inversión significativa de tiempo en aprendizaje autodidacta o prácticas erróneas.

Como solución, he desarrollado FitFuelBalance, una plataforma integrada que ofrece servicios tanto en formato de aplicación web como móvil. Esta aplicación permite a los usuarios solicitar servicios personalizados de entrenadores y nutricionistas, quienes pueden utilizar un repositorio de ejercicios y dietas predefinidas para crear planes a medida. Los usuarios tienen la capacidad de revisar detalles y solicitar ajustes en sus rutinas a los profesionales a través de la aplicación, garantizando así una experiencia personalizada y eficiente. Además, se incorpora el uso de inteligencia artificial para facilitar recomendaciones instantáneas y adaptaciones, reforzando la personalización de los planes ofrecidos.

El proyecto ha despertado un gran interés entre distintos perfiles de usuarios, desde expertos deportivos hasta individuos sin experiencia previa, así como personas dedicadas al entrenamiento o la nutrición, demostrando el potencial de la tecnología para transformar la manera en que las personas acceden y gestionan su salud y bienestar. Este documento detallará el análisis, diseño, codificación, pruebas e implementación de la plataforma FitFuelBalance, dirigida individuos que buscan mejorar su salud y rendimiento físico de manera eficiente y personalizada.

Palabras clave

Aplicación Web, Django, React, Nutrición, Deporte, Usuario, Ordenador, Móvil, Back-End, Front-End, Base de Datos, PostgreSQL

Abstract

The intersection between sports and nutrition is fundamental to maintaining a healthy lifestyle. However, many individuals struggle to integrate healthy practices into their daily routines due to a lack of time and specific knowledge. This challenge often leads to the adoption of generic diets or inadequate exercise routines, increasing the risk of negative outcomes and injuries. The main barrier to adopting a healthy lifestyle lies in the lack of relevant and accessible information, as well as the limited time to acquire such knowledge.

Recognizing this need, my Final Degree Project (TFG) focuses on simplifying access to personalized nutrition and exercise plans. The aim is to provide users, regardless of their prior sports experience, with the necessary tools to improve their performance or embark on a journey towards physical well-being, without requiring significant time investment in self-learning or incorrect practices.

As a solution, I have developed FitFuelBalance, an integrated platform offering services both as a web and mobile application. This application allows users to request personalized services from trainers and nutritionists, who can use a repository of predefined exercises and diets to create tailored plans. Users have the ability to review details and request adjustments to their routines from professionals through the application, ensuring a personalized and efficient experience. Additionally, the use of artificial intelligence is incorporated to facilitate instant recommendations and adaptations, enhancing the personalization of the plans offered.

The project has garnered significant interest among various user profiles, from sports experts to individuals with no prior experience, as well as those dedicated to training or nutrition, demonstrating the potential of technology to transform how people access and manage their health and well-being. This document will detail the analysis, design, coding, testing, and implementation of the FitFuelBalance platform, aimed at businesses, institutions, and individuals seeking to improve their health and physical performance efficiently and personally.

Keywords

Web Aplication, Django, React, Nutrition, Sport, User, Computer, Smartphone, Back-End, Front-End, Database, PostgreSQL

Índice

Listas

Lista de códigos

Lista de figuras

Lista de tablas

1

Introducción

En el presente Trabajo Final de Grado (TFG) se desarrolla FitFuelBalance, una plataforma integrada que ofrece servicios personalizados de nutrición y entrenamiento a través de una aplicación web y móvil. La creciente demanda de soluciones tecnológicas que faciliten la adopción de estilos de vida saludables ha motivado la creación de esta plataforma, dirigida a usuarios con diversos niveles de experiencia en el ámbito deportivo y nutricional.

La idea del proyecto me fue propuesta por parte del Dr. Pablo Cerro, entonces se me explicó el gran alcance que podría tener esta aplicación y el avance que podría suponer en el mundo de la nutrición y el deporte. Al ser yo una persona que realiza mucho deporte y cuida su nutrición, el proyecto llamó mi atención convirtiéndose en mi trabajo de fin de grado.

1.1. Motivación

La intersección entre el deporte y la nutrición es esencial para mantener un estilo de vida saludable. No obstante, es difícil integrar prácticas saludables en una rutina diaria por falta de tiempo o conocimientos sobre el campo. Esto nos lleva a utilizar dietas o rutinas de ejercicio de internet o realizadas por gente no experta, incrementando el riesgo de resultados negativos.

La principal motivación para el desarrollo de FitFuelBalance radica en la necesidad de simplificar el acceso a planes personalizados de nutrición y ejercicio. La plataforma está diseñada para proporcionar a los usuarios las herramientas necesarias para mejorar su rendimiento físico o iniciar un camino hacia el bienestar, sin requerir una inversión significativa de tiempo en aprendizaje autodidacta o prácticas erróneas.

Estudios recientes han resaltado la importancia de una nutrición adecuada combinada con ejercicio regular para prevenir enfermedades crónicas y mejorar la calidad de vida. Según la Organización Mundial de la Salud (OMS), la inactividad física es uno de los principales factores de riesgo de mortalidad mundial, responsable de aproximadamente 3.2 millones de muertes cada año [1]. También una dieta equilibrada puede prevenir hasta un 80 % de las enfermedades cardíacas prematuras y derrames cerebrales [2].

Además, un estudio publicado en el British Journal of Sports Medicine muestra que la combinación de actividad física y una dieta saludable reduce significativamente el riesgo de mortalidad por todas las causas. Según este estudio, las personas que realizan ejercicio regular y siguen una dieta equilibrada tienen un 29 % porciento menos de probabilidades de morir prematuramente en comparación con aquellas que no mantienen estos hábitos saludables [3].

La plataforma FitFuelBalance no solo busca facilitar la adopción de hábitos saludables, sino también contribuir al bienestar general de la sociedad. Por ejemplo, un estudio realizado por el American College of Sports Medicine indica que las intervenciones tecnológicas en el ámbito de la salud pueden aumentar la adherencia a los programas de ejercicio y mejorar los resultados de salud [4].

1.2. Objetivos

El objetivo principal de este TFG es desarrollar una plataforma integrada, FitFuelBalance, que ofrezca servicios personalizados de nutrición y entrenamiento a través de una aplicación web y móvil. Para lograr esto, se han establecido los siguientes objetivos específicos:

- **Diseño e implementación:** Crear una interfaz intuitiva y fácil de usar que permita a los usuarios acceder a planes personalizados de nutrición y ejercicio y crear un back-end para gestionar de manera eficiente y segura los datos de los usuarios
- **Facilitar la personalización de planes:** Permitir que los entrenadores y nutricionistas creen y ajusten planes personalizados basados en las necesidades específicas de cada usuario.
- **Desplegar la plataforma en la nube:** Realizar un despliegue de la plataforma para ser utilizado como Software as a Service (SaaS).

1.3. Estructura del documento

Este documento se organiza de la siguiente manera:

- **Sección 1: Introducción** - Presenta la motivación, los objetivos y la estructura del documento.
- **Sección 2: Antecedentes** - Describe el contexto y la necesidad del proyecto, así como las tecnologías utilizadas.
- **Sección 3: Estado del Arte** - Proporciona una visión general de las tecnologías y aplicaciones existentes relacionadas con el proyecto.
- **Sección 4: Diseño** - Detalla el diseño del proyecto, incluyendo la definición, el alcance, la metodología, la arquitectura y los roles.
- **Sección 5: Desarrollo** - Describe el proceso de desarrollo del back-end y front-end de la plataforma.
- **Sección 6: Pruebas e Implementación** - Presenta las pruebas realizadas al sistema y el proceso de implementación.
- **Sección 7: Conclusiones y Trabajo Futuro** - Resume los logros del proyecto y sugiere posibles mejoras y direcciones futuras.

2

Antecedentes

2.1. Contexto y Necesidad

En esta sección se presenta el contexto y la necesidad de desarrollar una plataforma que integre nutrición y ejercicio de manera personalizada. Se abordan los desafíos comunes que enfrentan las personas al intentar adoptar un estilo de vida saludable y se destaca la importancia de proporcionar información relevante y accesible para facilitar la adopción de hábitos saludables.

Importancia de la Nutrición y el Ejercicio

La intersección entre la nutrición y el ejercicio es esencial para mantener un estilo de vida saludable. La nutrición adecuada proporciona los nutrientes necesarios para el funcionamiento óptimo del cuerpo, mejora la capacidad de recuperación y reduce el riesgo de enfermedades crónicas. Por otro lado, el ejercicio regular fortalece el sistema cardiovascular, mejora la salud mental y ayuda en el control del peso corporal. Numerosos estudios han demostrado que una combinación equilibrada de dieta y ejercicio puede prolongar la vida útil, mejorar la calidad de vida y aumentar la capacidad física y mental. Sin embargo, la implementación de estas prácticas de manera efectiva requiere un conocimiento adecuado y la habilidad para personalizar rutinas y dietas según las necesidades individuales.

Desafíos Comunes

A pesar de la clara importancia de la nutrición y el ejercicio, muchas personas enfrentan desafíos significativos para integrar estas prácticas en su vida diaria. Los principales desafíos incluyen la falta de tiempo, ya que la vida moderna y las responsabilidades laborales y familiares a menudo dejan poco tiempo para la planificación y la implementación de rutinas de salud y fitness, lo que dificulta dedicar tiempo suficiente para hacer ejercicio y preparar comidas saludables. Además, la falta de conocimientos específicos sin una orientación adecuada lleva a que las personas adopten dietas genéricas o rutinas de ejercicio que no están alineadas con sus objetivos o necesidades específicas, resultando en falta de progreso, desmotivación e incluso lesiones. Finalmente, la adopción de prácticas inadecuadas es común debido a la abundancia de información contradictoria y poco confiable en internet, lo que

puede llevar a la adopción de dietas extremas o regímenes de ejercicio poco saludables. Sin la guía adecuada, es fácil que las personas caigan en trampas de marketing que prometen resultados rápidos sin considerar los riesgos asociados.

Necesidad de Información Relevante y Accesible

La principal barrera para adoptar un estilo de vida saludable radica en la falta de información relevante y accesible. Muchas personas no tienen el tiempo ni los recursos para educarse adecuadamente sobre nutrición y ejercicio, y al intentar acceder a ayuda de profesionalista y especialista, entra en juego el alto coste de contratar uno de estos. Además, la personalización de estos planes es crucial, ya que cada individuo tiene necesidades, capacidades y objetivos únicos.

Reconociendo esta necesidad, el desarrollo de herramientas y plataformas que proporcionen información precisa, relevante y personalizada es fundamental. Estas herramientas deben ser accesibles, fáciles de usar y capaces de adaptarse a las circunstancias individuales de cada usuario.

Este contexto subraya la importancia de crear soluciones como FitFuelBalance, que integren nutrición y ejercicio en una plataforma única y personalizada, abordando los desafíos mencionados y facilitando el acceso a planes de salud efectivos y adaptados.

2.2. Tecnologías Utilizadas.

En esta sección, se explican los principales frameworks, bibliotecas y tecnologías utilizadas para el desarrollo del proyecto *FitFuelBalance*.

Django

Django es un framework de alto nivel para el desarrollo web en Python que fomenta el desarrollo rápido y un diseño limpio y pragmático. Fue creado en 2005 por Adrian Holovaty y Simon Willison mientras trabajaban en el periódico *Lawrence Journal-World*. Django se basa en la idea de DRY (Don't Repeat Yourself) y ofrece herramientas poderosas como un ORM (Object-Relational Mapping), un sistema de plantillas y una interfaz administrativa automáticamente generada [5].

Las ventajas de Django incluyen la administración automática de la base de datos, un sistema de plantillas potente y flexible, y una gran cantidad de bibliotecas y paquetes disponibles. Sin embargo, puede ser más complejo para proyectos pequeños y requiere familiaridad con Python y el ecosistema de Django.

React

React es una biblioteca de JavaScript para crear interfaces de usuario para sistemas distribuidos, especialmente aquellos que trabajan con datos y actualizaciones en tiempo real. Utiliza JSX, una sintaxis muy similar a HTML, lo que facilita su implementación. El sistema se basa en componentes que contienen su propia lógica y estado [6].

Entre las ventajas de React se encuentran la facilidad para crear interfaces de usuario dinámicas y responsivas, una gran comunidad y un ecosistema de bibliotecas y herramientas, así como la modularización y reutilización de componentes. Las limitaciones incluyen un proceso de configuración inicial y la complejidad en la gestión del estado en aplicaciones grandes sin herramientas adicionales.

React Native

React Native es un marco de desarrollo de aplicaciones móviles creado por Facebook en 2015. Permite a los desarrolladores construir aplicaciones móviles utilizando JavaScript y React, compilando a código nativo para iOS y Android [7].

Las ventajas de React Native incluyen la reutilización de código entre plataformas iOS y Android, un ecosistema robusto y activo, y componentes nativos que garantizan un rendimiento óptimo. No obstante, algunas funcionalidades nativas pueden requerir desarrollo adicional y la actualización de versiones puede requerir ajustes significativos.

PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional y objeto-relacional, conocido por su estabilidad, extensibilidad y cumplimiento de estándares. Es una elección popular para aplicaciones web y sistemas de información que requieren una base de datos robusta y escalable [8].

Las ventajas de PostgreSQL incluyen su alta extensibilidad y soporte para tipos de datos avanzados, garantía de transacciones ACID completas para la integridad de los datos, y una amplia comunidad y soporte de herramientas. Sin embargo, puede requerir configuración y ajuste para un rendimiento óptimo y la administración puede ser compleja para usuarios sin experiencia previa en bases de datos.

Python

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general, conocido por su sintaxis legible y facilidad de uso. En este proyecto, Python se utiliza tanto para el desarrollo de Django como para otras funciones del backend [9].

Las ventajas de Python son su sintaxis sencilla y clara, ideal para desarrollo rápido, una gran cantidad de bibliotecas y frameworks disponibles, y una comunidad activa y extensa documentación.

No obstante, puede no ser tan rápido como lenguajes compilados en ciertas tareas y requiere un buen manejo de entornos virtuales para la gestión de dependencias.

JavaScript

JavaScript es un lenguaje de programación interpretado que se utiliza principalmente para el desarrollo de aplicaciones web dinámicas. Es el lenguaje más común para el desarrollo frontend y se utiliza en este proyecto junto con React [10].

Las ventajas de JavaScript incluyen su amplio soporte por todos los navegadores web, una gran cantidad de frameworks y bibliotecas para desarrollo web, y su idoneidad para la creación de aplicaciones web interactivas y dinámicas. Las limitaciones incluyen problemas de compatibilidad entre diferentes navegadores y la necesidad de una gestión adecuada del código para evitar problemas de rendimiento.

Bootstrap y CSS

Bootstrap es un framework de front-end de código abierto que permite diseñar sitios y aplicaciones web de forma rápida y sencilla. CSS (Cascading Style Sheets) es el lenguaje utilizado para describir la presentación de un documento escrito en HTML o XML [11, 12].

Las ventajas de Bootstrap y CSS incluyen la facilidad para crear interfaces responsivas y atractivas, una gran cantidad de componentes predefinidos, y la compatibilidad con todos los navegadores modernos. Sin embargo, pueden resultar en sitios web que se ven similares si no se personaliza adecuadamente y la dependencia de archivos externos puede afectar el rendimiento si no se gestiona bien.

3

Estado del arte

En esta sección, se presenta un análisis detallado de las soluciones actuales en el ámbito de la nutrición y el ejercicio, con el objetivo de identificar las limitaciones y oportunidades para mejorar la integración de estas prácticas en la vida cotidiana de las personas. La revisión de las tecnologías y plataformas existentes proporcionará una base sólida para entender cómo la plataforma FitFuelBalance puede ofrecer soluciones innovadoras y personalizadas que aborden las necesidades de diversos perfiles de usuarios, desde deportistas profesionales hasta individuos sin experiencia previa en salud y fitness.

3.1. Soluciones Existentes

En esta sección, se examinan las soluciones y plataformas actuales en el mercado que abordan la integración de la nutrición y el ejercicio. Se identifican sus características, ventajas y limitaciones.

3.1.1. Aplicaciones de Nutrición

Existen numerosas aplicaciones que ayudan a los usuarios a gestionar su nutrición. Entre las más populares se encuentran MyFitnessPal [13] y Yazio [14], cuyas características, ventajas y limitaciones se resumen en la Tabla 3.1.

3.1.2. Aplicaciones de Ejercicio

Las aplicaciones de ejercicio proporcionan rutinas de entrenamiento y seguimiento del progreso físico. Entre las más destacadas están Nike Training Club [15], Fitbod [16] y Calisteniapp [17], cuyas características, ventajas y limitaciones se detallan en la Tabla 3.2.

Aplicación	Ventajas	Limitaciones
MyFitnessPal	MyFitnessPal ofrece una amplia base de datos de alimentos y facilita el registro de alimentos mediante escaneo de códigos de barras. También permite un seguimiento detallado de macronutrientes.	MyFitnessPal presenta planes de nutrición genéricos y requiere una versión premium para acceder a funciones avanzadas.
Yazio	Yazio proporciona planes de nutrición personalizados, recetas y planes de comidas saludables, y una interfaz amigable y fácil de usar.	Algunas funciones de Yazio requieren suscripción premium y tiene una menor base de datos de alimentos en comparación con MyFitnessPal.

Tabla 3.1: Comparación de aplicaciones de nutrición

Aplicación	Ventajas	Limitaciones
Nike Training Club	Nike Training Club ofrece una variedad de entrenamientos para diferentes niveles y objetivos, con instrucciones detalladas y videos de alta calidad, y es gratis para la mayoría de las funciones.	Carece de personalización avanzada basada en datos del usuario y requiere conexión a internet para acceder a los videos.
Fitbod	Fitbod ofrece personalización avanzada de los entrenamientos, adaptación de ejercicios según el progreso y el equipo disponible, y seguimiento detallado del rendimiento.	Necesita una suscripción para acceder a todas las funciones y puede ser complejo para principiantes.
Calisteniapp	Calisteniapp se especializa en calistenia, con rutinas personalizables basadas en el nivel y los objetivos del usuario, y una comunidad activa con funciones de seguimiento del progreso.	Puede no ser adecuada para usuarios que prefieren entrenamientos con pesas o equipos de gimnasio, y algunas funciones avanzadas requieren suscripción premium.

Tabla 3.2: Comparación de aplicaciones de ejercicio

3.1.3. Plataformas Combinadas

Algunas plataformas intentan integrar tanto la nutrición como el ejercicio, proporcionando una solución más holística para la gestión de la salud. Ejemplos de estas plataformas son Fitbit [18] y Samsung Health [19], cuyas características, ventajas y limitaciones se resumen en la Tabla 3.3.

Plataforma	**Ventajas**	**Limitaciones**
Fitbit	Fitbit integra datos de actividad física y nutrición, ofrece una amplia gama de dispositivos wearables, y cuenta con una comunidad activa y funciones sociales.	La personalización en los planes de ejercicio y nutrición es limitada y depende de dispositivos adicionales para obtener el máximo beneficio.
Samsung Health	Samsung Health ofrece una amplia gama de funciones de salud y bienestar, integración con dispositivos Samsung, y funciones de monitoreo del sueño y la salud en general.	La personalización es limitada en comparación con plataformas especializadas y tiene menor integración con dispositivos de otras marcas.

Tabla 3.3: Comparación de plataformas combinadas

3.1.4. Limitaciones de las Soluciones Actuales

A pesar de los avances, las soluciones actuales enfrentan varias limitaciones. La falta de personalización no considera completamente las necesidades individuales de cada usuario. El acceso a asesoramiento profesional es limitado, ya que la mayoría de las plataformas no proporcionan acceso directo a profesionales de la salud y el fitness, lo que restringe la capacidad de obtener asesoramiento personalizado. Además, algunas aplicaciones y servicios personalizados pueden ser costosos. Finalmente, la integración de datos es insuficiente, ya que la falta de integración entre diferentes dispositivos puede dificultar una visión holística.

3.2. Plataformas Personalizadas para la Nutrición y el Ejercicio

3.2.1. Aplicaciones de Nutrición Personalizada

Existen diversas plataformas que ofrecen planes de nutrición personalizados adaptados a las necesidades individuales de los usuarios. Entre las más populares se encuentran Nutrifix [20] y PlateJoy [21]. Las características, ventajas y limitaciones de algunas de estas plataformas se resumen en la Tabla 3.4.

Plataforma	**Ventajas**	**Limitaciones**
Nutrifix	Nutrifix ofrece personalización basada en datos de actividad física y considera restricciones dietéticas y preferencias.	Necesita sincronizar con otras aplicaciones para obtener datos precisos y algunas funciones avanzadas requieren una suscripción.
PlateJoy	PlateJoy proporciona listas de compras y recetas personalizadas, además de adaptarse a diversas necesidades dietéticas (vegetariano, sin gluten, etc.).	Requiere una suscripción para acceder a todas las funciones y depende de la precisión de los datos ingresados por el usuario.

Tabla 3.4: Comparación de plataformas de nutrición personalizada

3.2.2. Aplicaciones de Ejercicio Personalizado

Las aplicaciones de ejercicio proporcionan rutinas de entrenamiento y seguimiento del progreso físico. Entre las más destacadas se encuentran Freeletics [22] y JEFIT [23]. Las características, ventajas y limitaciones de algunas de estas aplicaciones se detallan en la Tabla 3.5.

3.2.3. Plataformas Combinadas

Algunas plataformas buscan integrar tanto la nutrición como el ejercicio para ofrecer una solución completa. Entre las más populares se encuentran 8fit [24] y Noom [25]. Las características, ventajas y limitaciones de algunas de estas plataformas se resumen en la Tabla 3.6.

3.2.4. Limitaciones y Oportunidades de Mejora

A pesar de los avances en la personalización de planes de nutrición y ejercicio, las soluciones actuales todavía enfrentan desafíos. Las principales limitaciones y oportunidades de mejora se resumen

Plataforma	Ventajas	Limitaciones
Freeletics	Freeletics ofrece planes de entrenamiento personalizados y adaptativos, además de videos instructivos y seguimiento del rendimiento.	Requiere una suscripción para acceder a funciones premium y puede no ser adecuado para usuarios que prefieren entrenamiento con equipos específicos.
JEFIT	JEFIT cuenta con una base de datos extensa de ejercicios y una comunidad activa para compartir y comparar rutinas.	Requiere suscripción para acceder a algunas funciones avanzadas y la personalización es limitada sin la versión premium.

Tabla 3.5: Comparación de plataformas de ejercicio personalizado

Plataforma	Ventajas	Limitaciones
8fit	8fit integra planes de nutrición y ejercicio, ofrece videos instructivos y listas de compras.	Requiere una suscripción para acceder a todas las funciones y necesita consistencia en el ingreso de datos por parte del usuario.
Noom	Noom se enfoca en la psicología del comportamiento para cambios duraderos e integra dieta y ejercicio en un solo plan.	Requiere suscripción para acceder a las funciones completas y puede ser más caro en comparación con otras aplicaciones.

Tabla 3.6: Comparación de plataformas combinadas

Limitación	Descripción
Falta de Personalización	Muchas aplicaciones ofrecen planes genéricos que no consideran completamente las necesidades individuales de cada usuario.
Acceso a Asesoramiento Profesional	La mayoría de las plataformas no proporcionan acceso directo a profesionales de la salud y el fitness, lo que limita la capacidad de obtener asesoramiento personalizado.
Costo	Algunas aplicaciones y servicios personalizados pueden ser costosos, lo que restringe el acceso para muchos usuarios.
Integración de Datos	La falta de integración entre diferentes dispositivos y aplicaciones puede dificultar una visión holística de la salud del usuario.

Tabla 3.7: Limitaciones y oportunidades de mejora

en la Tabla 3.7.

Estas limitaciones abren oportunidades para el desarrollo de nuevas plataformas que puedan ofrecer un mayor nivel de personalización, accesibilidad y apoyo profesional.

4

Diseño

En esta sección se describirá el diseño del sistema FitFuelBalance, abordando las directrices y metodologías empleadas durante el ciclo de vida del proyecto, así como su arquitectura. El objetivo es proporcionar una visión detallada de cómo se estructuró y organizó el desarrollo para garantizar la eficiencia y funcionalidad del sistema.

4.1. Definición del proyecto y alcance

El proyecto FitFuelBalance tiene como objetivo desarrollar una plataforma integrada de nutrición y ejercicio, accesible tanto en formato web como móvil. Esta plataforma permite a los usuarios recibir planes personalizados de nutrición y ejercicio, creados por profesionales en base a sus necesidades individuales. El sistema no solo servirá como una herramienta para obtener estos planes, sino también por ejemplo, de realizar un entrenamiento y que la aplicación te guie por los ejercicios.

El proyecto abarca el desarrollo completo de una aplicación web y móvil, desde la fase de análisis y diseño, pasando por la implementación y pruebas, hasta su despliegue final en un entorno de producción. La plataforma está diseñada para ser accesible desde cualquier dispositivo con conexión a Internet, asegurando que los usuarios puedan interactuar con sus planes y profesionales en cualquier momento y lugar.

El ciclo de vida del proyecto está basado en un enfoque incremental, lo que permite la adición progresiva de funcionalidades y asegura que cada incremento deje el sistema en un estado funcional y robusto. Esto es especialmente beneficioso para adaptarse a nuevos requisitos y mejoras continuas basadas en la retroalimentación de los usuarios.

Con este proyecto, se busca no solo ofrecer una solución práctica a los usuarios, sino también impulsar el uso de tecnologías modernas como Django, React, React Native y PostgreSQL en la creación de sistemas distribuidos eficientes y escalables orientados hacia la nutrición y el deporte.

4.2. Metodología

Para el desarrollo de FitFuelBalance, se ha optado por utilizar un modelo incremental. Esta metodología permite el desarrollo del proyecto en pequeños incrementos funcionales, lo que facilita la incorporación de nuevas funcionalidades de manera progresiva, esto permite el desarollo de un sistema robusto y haciendo que los incrementos no afecten a otros subsistemas al desarollarse. Figura 4.1 muestra el ciclo de vida del proyecto basado en un enfoque incremental.

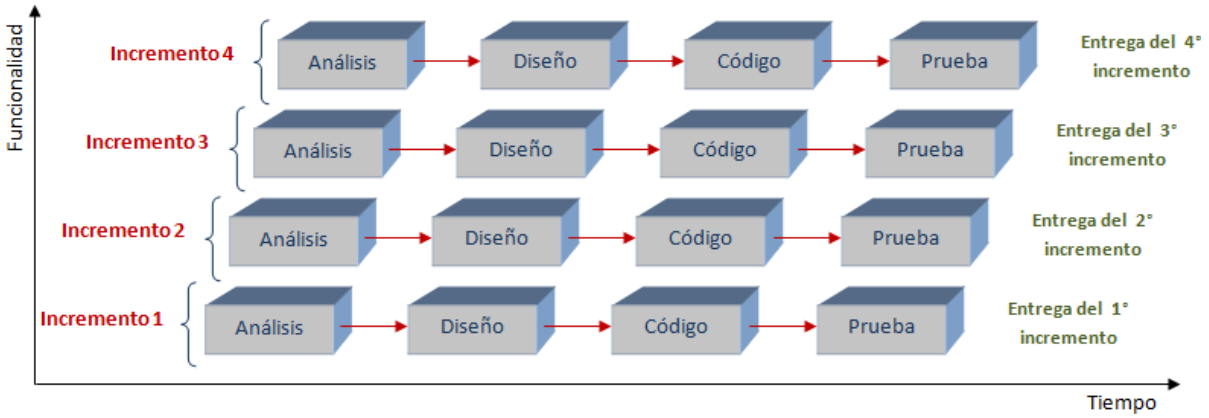

Figura 4.1: Modelo de desarrollo incremental [26]

Fases del proyecto

El proyecto se ha dividido en varias fases clave, cada una con objetivos específicos y entregables definidos:

- **Análisis de requisitos:** Se recopilaron los requisitos funcionales y no funcionales a través de la solicitud de opioniones a entrenadores y deportistas sobre que esperarían de una aplicación innovadora.
- **Diseño:** En esta fase se definió la arquitectura del sistema, incluyendo la estructura de la base de datos, la arquitectura de la aplicación web y móvil, y la integración de servicios externos. Se crearon diagramas UML para visualizar la arquitectura y las interacciones entre los componentes del sistema.
- **Desarrollo:** El desarrollo se realizó en incrementos, cada uno añadiendo nuevas funcionalidades al sistema. Las tecnologías utilizadas en esta fase incluyen Django para el backend, React para el frontend web, React Native para la aplicación móvil, y PostgreSQL gestionado a través de Neon para la base de datos.
- **Pruebas:** Se realizaron pruebas unitarias, de integración y funcionales para garantizar la calidad del código y el correcto funcionamiento de la aplicación.
- **Despliegue:** La aplicación se desplegó en entornos de prueba y producción utilizando Render para el alojamiento tanto del frontend como del backend.
- **Mantenimiento:** Se establecieron procedimientos para el mantenimiento y actualización continua de la aplicación, cada vez que se incluyen funcionalidades, estas son subidas a producción para mantener la aplicación actualizada.

Herramientas y técnicas

Durante el desarrollo del proyecto se utilizaron diversas herramientas y técnicas para apoyar el modelo incremental. Git y GitHub [27] se emplearon para el control de versiones y la colaboración en el código fuente, permitiendo un seguimiento detallado de los cambios y facilitando el trabajo en equipo. Visual Studio Code [28] fue el entorno de desarrollo integrado (IDE) elegido para la programación, debido a su flexibilidad y amplia gama de extensiones que mejoran la productividad. Figma [29] se utilizó para el diseño de interfaces de usuario y la creación de prototipos, proporcionando una visualización clara y precisa del diseño final antes de su implementación. React Native CLI [30] se empleó para la prueba y depuración de la aplicación móvil, asegurando que la funcionalidad y el rendimiento fueran óptimos en dispositivos reales. Neon [31] se utilizó para la gestión de la base de datos PostgreSQL, facilitando la administración de datos de manera eficiente y segura. Finalmente, Render [32] se encargó del despliegue y alojamiento del frontend y backend, garantizando que la aplicación estuviera disponible y funcionando correctamente en un entorno de producción.

4.3. Arquitectura

La arquitectura de FitFuelBalance se ha diseñado para ser modular, escalable y mantener una separación clara entre las distintas capas del sistema. A continuación, se describen los principales componentes de la arquitectura y sus interacciones.

El sistema está estructurado en tres capas principales. La Capa de Presentación se divide en dos componentes: el Frontend Web y la Aplicación Móvil. El Frontend Web, desarrollado utilizando React, se encarga de la interacción con el usuario a través de una interfaz intuitiva y responsive. Para el diseño y estilo visual, se utilizan Bootstrap y CSS. Por otro lado, la Aplicación Móvil, desarrollada con React Native, permite a los usuarios acceder a los servicios de FitFuelBalance desde dispositivos móviles iOS [33] y Android [34]. La interfaz móvil está diseñada para ser amigable y fácil de usar.

La Capa de Lógica de Negocio está representada por el Backend, desarrollado con Django. El backend maneja la lógica de negocio de la aplicación, las reglas de negocio y la gestión de datos. Además, implementa APIs RESTful [35] para la comunicación con el frontend y la aplicación móvil, asegurando una integración eficiente y segura entre las distintas partes del sistema.

Finalmente, la Capa de Datos se centra en la Base de Datos. Para almacenar y gestionar los datos de la aplicación se utiliza PostgreSQL. Neon gestiona esta base de datos, proporcionando alta disponibilidad y escalabilidad, lo que garantiza que los datos estén siempre accesibles y seguros, incluso a medida que la aplicación crece y se expande.

Diagrama de Arquitectura

A continuación, Figura 4.2 muestra un diagrama de la arquitectura general de FitFuelBalance, que ilustra la interacción entre los distintos componentes del sistema.

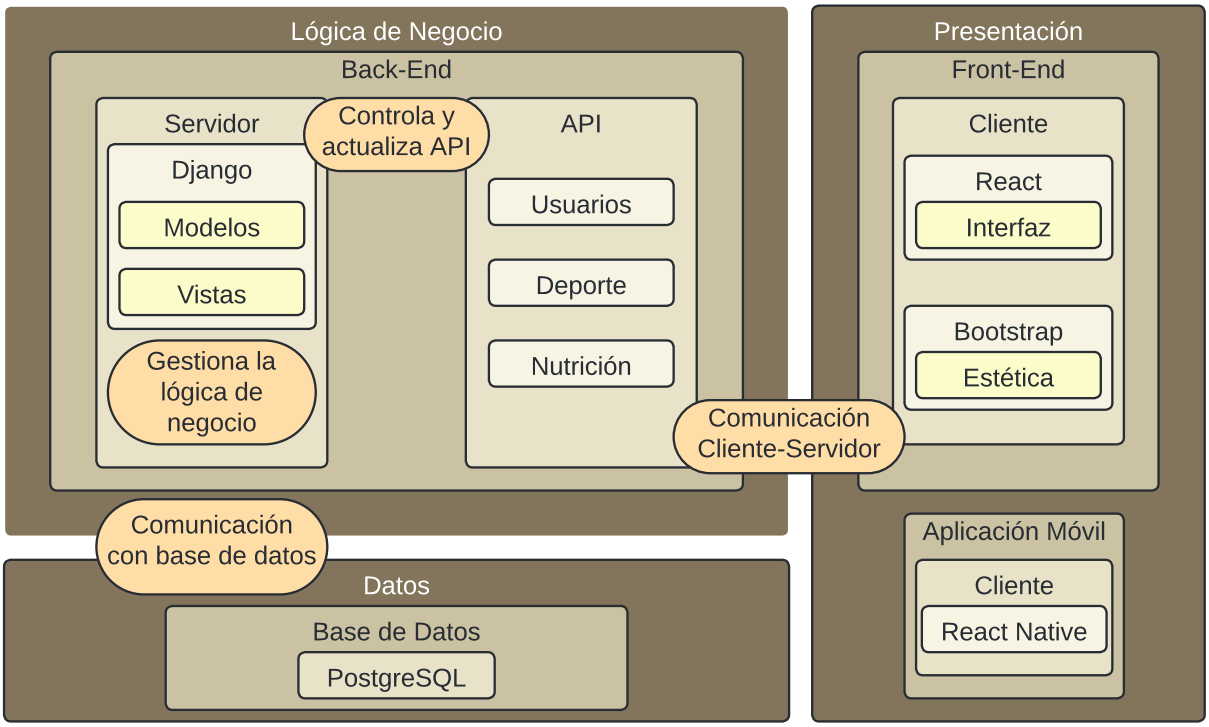

Figura 4.2: Subsistemas del sistema FitFuelBalance

Descripción de Componentes

El **Frontend Web** está construido utilizando **React**, lo que permite una interfaz de usuario dinámica y eficiente, facilitando la creación de componentes reutilizables y un manejo eficaz del estado de la aplicación. Para asegurar una presentación visual consistente y atractiva, se utilizan **Bootstrap** y **CSS** para estilizar la aplicación.

La **Aplicación Móvil** se desarrolla con **React Native**, lo que permite el desarrollo de aplicaciones móviles multiplataforma a partir de una única base de código. React Native proporciona una experiencia de usuario nativa y acceso a funcionalidades específicas del dispositivo, asegurando una interfaz amigable y efectiva en dispositivos iOS y Android.

El **Backend** se implementa utilizando **Django**, un framework de alto nivel que facilita el desarrollo rápido y limpio de aplicaciones web. Django proporciona una estructura robusta con una gran cantidad de funcionalidades integradas. Los principales módulos utilizados para controlar la lógica de la aplicación son los modelos y las vistas. La comunicación entre el frontend, la aplicación móvil y el backend se realiza a través de **APIs RESTful**, que permiten un intercambio de datos eficiente y seguro. La API se

divide en tres módulos principales: usuarios, nutrición y deporte, asegurando una gestión organizada y efectiva de cada área.

La **Base de Datos** utiliza **PostgreSQL**, una base de datos relacional que almacena datos estructurados de manera eficiente. La gestión de la base de datos se realiza a través de **Neon**, que proporciona capacidades avanzadas de administración y escalabilidad, garantizando que los datos estén siempre accesibles y seguros.

Despliegue y Infraestructura

El despliegue de FitFuelBalance se ha realizado utilizando **Render**, una plataforma que facilita la gestión y el despliegue continuo tanto del frontend como del backend. Render permite gestionar la aplicación de manera eficiente, asegurando una alta disponibilidad, escalabilidad y rendimiento, lo que permite manejar un alto volumen de usuarios y datos.

4.4. Roles

En la plataforma FitFuelBalance, se han definido tres roles principales que determinan las capacidades y permisos de los usuarios dentro del sistema: Usuarios Normales, Entrenadores y/o Nutricionistas, y Administradores. A continuación, se detallan las características y funciones de cada uno de estos roles.

Usuarios Regulares

Los Usuarios Regulares utilizan la plataforma para acceder a planes de entrenamiento y nutrición personalizados. Pueden registrarse, iniciar sesión y gestionar su perfil personal, solicitar planes de entrenamiento y nutrición según sus objetivos y necesidades, visualizar los planes proporcionados por entrenadores y nutricionistas, y seguir las recomendaciones. Además, pueden registrar su progreso y recibir retroalimentación basada en sus resultados y evolución.

Entrenadores y/o Nutricionistas

Los Entrenadores y/o Nutricionistas son profesionales que proporcionan planes de entrenamiento y nutrición a los usuarios. Pueden registrarse como entrenadores o nutricionistas, completar su perfil profesional y ofrecer sus servicios. Desarrollan planes personalizados utilizando el repositorio de ejercicios y dietas de la plataforma, asignan planes a los usuarios según sus requerimientos y objetivos, y monitorean el progreso de los usuarios, realizando ajustes necesarios para asegurar resultados óptimos.

Administradores

Los Administradores son responsables de la gestión y el mantenimiento general de la plataforma. Supervisan el registro de usuarios, entrenadores y nutricionistas, gestionan permisos y roles, moderan el contenido para asegurar que cumpla con los estándares de calidad y las políticas de la plataforma, realizan tareas de mantenimiento y actualización para asegurar el buen funcionamiento y la seguridad de los datos, y proporcionan soporte técnico a los usuarios para resolver problemas que puedan surgir.

4.5. Requisitos Funcionales y No Funcionales

Para definir las necesidades y servicios que debe cumplir la aplicación y las funciones que debe ofrecer, se han identificado los requisitos funcionales y no funcionales y se han recogido en una lista en el Apéndice C. Estos requisitos se han obtenido a partir de la información recopilada en la fase de análisis y diseño del proyecto, y se han organizado en categorías para facilitar su comprensión y seguimiento.

4.6. Incrementos

Los incrementos se dividen en Back-end, Front-end y Aplicación Móvil. Generalmente, un incremento dentro de una sección no comienza hasta que el incremento anterior de la misma sección se ha completado. Sin embargo, en algunos casos, debido a problemas de dependencia lógica o por razones de prueba, puede haber dos o más incrementos en desarrollo simultáneamente. Por la misma razón, a veces se han desarrollado incrementos de más de una sección al mismo tiempo.

Cada incremento tiene un pequeño ciclo de vida que consiste en un análisis, un diseño, desarrollo y finalmente pruebas. En esta sección, solo veremos la definición de cada incremento sin entrar en detalles sobre su ciclo de vida.

Back-end

- **IN1.1** Crear una base de datos PostgreSQL y configurar el servidor HTTP local con Django.
- **IN1.2** Definir modelos en Django para gestionar el módulo de deporte.
- **IN1.3** Definir modelos en Django para gestionar el módulo de nutrición.
- **IN1.4** Definir modelos en Django para gestionar los usuarios.
- **IN1.5** Definir vistas para controlar las operaciones de los modelos.
- **IN1.6** Crear controladores para manejar los inicios de sesión y registros de usuarios.
- **IN1.7** Crear APIs RESTful con Django REST framework para manejar las llamadas de los clientes.

- **IN1.8** Implementar la lógica de negocio y middleware necesarios para las operaciones del servidor.
- **IN1.9** Desplegar todo en Render.

Front-end

- **IN2.1** Crear una aplicación básica con una barra de navegación y definir las rutas en React Router.
- **IN2.2** Desarrollar la sección de inicio de sesión.
- **IN2.3** Desarrollar el componente de inicio.
- **IN2.4** Desarrollar la sección de dietas.
- **IN2.5** Desarrollar la sección de entrenamientos.
- **IN2.6** Desarrollar la logica de autenticación y manejo de sesiones.
- **IN2.7** Desarrollar estetica y diseño de la aplicación.
- **IN2.8** Desarrollar sistema de filtraje y busqueda de dietas y entrenamientos.
- **IN2.9** Desplegar todo en Render.

Aplicación Móvil

- **IN3.1** Configurar el entorno de desarrollo con React Native CLI.
- **IN3.2** Crear pantallas básicas para la navegación principal.
- **IN3.3** Implementar la autenticación y el manejo de sesiones.
- **IN3.4** Desarrollar la pantalla de inicio con acceso a las funcionalidades principales.
- **IN3.5** Implementar la funcionalidad de seguimiento de entrenamientos en tiempo real.
- **IN3.6** Desarrollar la funcionalidad de registro y seguimiento de dietas diarias.
- **IN3.7** Probar la aplicación en dispositivos iOS y Android.
- **IN3.8** Exportar la aplicación a una APK.

5

Desarrollo

Esta sección detalla el desarrollo del proyecto, explicando en detalle los procedimientos seguidos y las tecnologías utilizadas. El sistema se ha dividido en dos partes principales: Back-End y Front-End. Se proporcionará una descripción detallada de cada una de estas partes junto con las herramientas y tecnologías empleadas.

5.1. Back-End

El desarrollo del Back-End se ha llevado a cabo utilizando Django, un framework de desarrollo web de alto nivel en Python. Django permite la creación de aplicaciones web seguras y mantenibles de manera rápida y eficiente. A continuación se detalla la estructura y los componentes principales del servidor:

La configuración del proyecto se realiza mediante la gestión de configuraciones globales con el archivo `settings.py`. El servidor está dividido en varias aplicaciones Django, cada una responsable de funcionalidades específicas. Los modelos, que definen la estructura de la base de datos utilizando modelos de Django. Las vistas, que manejan las solicitudes HTTP y las respuestas con Django REST Framework. Los serializadores, los cuales convierten instancias de modelos Django a formatos JSON y viceversa. Las URLs, que se definen para mapear solicitudes HTTP a las vistas correspondientes. La API REST, que se construye para permitir la comunicación entre el servidor y los clientes. La autenticación y autorización, la cual se implementan mediante un sistema basado en tokens JWT y finalmente, el despliegue del servidor que se realiza en la plataforma Render.

En el Apéndice E, Figura E.1, se muestra un diagrama de componentes del Back-End.

5.1.1. Estructura del Servidor

El servidor se ha desarrollado utilizando el framework Django, que proporciona una estructura modular y reutilizable, facilitando el desarrollo y mantenimiento de la aplicación. A continuación, Figura 5.1 ilustra la estructura general del servidor y la interacción entre sus componentes, así como la conexión

con la API REST y los clientes (aplicaciones web y móvil):

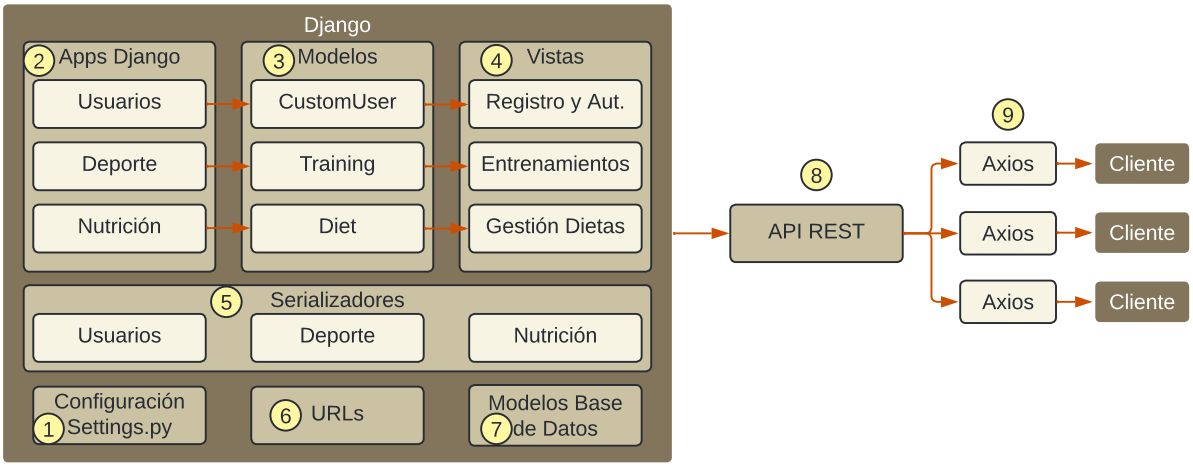

Figura 5.1: Estructura del servidor de FitFuelBalance

El proyecto está configurado con un archivo `settings.py` ① que gestiona las configuraciones globales, incluyendo la base de datos, aplicaciones instaladas, middleware y otros ajustes importantes. El servidor está dividido en varias aplicaciones Django, cada una responsable de una parte específica de la funcionalidad del sistema. Las aplicaciones principales ② incluyen: **Usuarios**, que maneja el registro, autenticación y gestión de perfiles de los usuarios; **Entrenamientos**, que gestiona la creación, actualización y seguimiento de los planes de entrenamiento; y **Nutrición**, que administra los planes de nutrición y el seguimiento de la dieta de los usuarios.

Los modelos de Django ③ definen la estructura de la base de datos. Cada modelo se traduce en una tabla de la base de datos y define los campos y las relaciones entre ellos. Algunos de los modelos clave incluyen: **CustomUser**, que define la información básica del usuario y deriva en `Trainer` y `RegularUser`, representando a los entrenadores y nutricionistas, y usuarios normales respectivamente; **Diet**, que contiene todas las comidas, alimentos y recetas que se pueden utilizar en los planes de nutrición; **Option**, que contiene un plan de 3 opciones de comidas distintas para cada día durante una semana; y **Training**, que se encarga de crear un entrenamiento con todos sus ejercicios correspondientes y sus series y repeticiones.

Las vistas en Django ④ son responsables de manejar las solicitudes HTTP y devolver las respuestas adecuadas. Utilizando Django REST Framework, las vistas se han estructurado como vistas basadas en clases (Class-Based Views), facilitando la reutilización y el manejo de lógica compleja. Las principales vistas incluyen: **Registro y Autenticación**, que maneja el registro de nuevos usuarios y la autenticación mediante JWT; **Gestión de Planes**, que permite a los entrenadores y nutricionistas crear, actualizar y eliminar planes de entrenamiento y nutrición; y **Seguimiento de Progreso**, que controla la actualización de comidas y entrenamientos creados por los usuarios.

Los serializadores ⑤ de Django REST Framework se utilizan para convertir instancias de mo-

delos Django a formatos JSON y viceversa. Esto es esencial para la comunicación entre el servidor y los clientes (aplicaciones web y móvil). Los serializadores principales incluyen: **Serializadores de Usuario**, que convierten las instancias de los modelos de usuario; **Serializadores de Deporte**, que convierten las instancias de los modelos de la parte deportiva de la aplicación; y **Serializadores de Nutrición**, que convierten las instancias de los modelos de la parte nutritiva de la aplicación.

Las rutas URL ⑥ definen cómo se mapean las solicitudes HTTP a las vistas correspondientes. En el archivo `urls.py`, se han definido las rutas principales de la API, organizadas por aplicación. La base de datos ⑦ utilizada es PostgreSQL, elegida por su robustez y capacidad para manejar grandes volúmenes de datos de manera eficiente. Las migraciones de base de datos se gestionan mediante las herramientas integradas de Django.

Se ha utilizado Django REST Framework para construir la API RESTful ⑧ que permite la comunicación entre el servidor y los clientes (aplicaciones web y móvil). Esta API maneja las solicitudes HTTP y proporciona respuestas en formato JSON. Las llamadas a la API ⑨ se autentican utilizando tokens JWT [36] y la librería Axios [37] en el frontend. Se ha implementado un sistema de autenticación basado en tokens utilizando JWT (JSON Web Tokens) para asegurar las comunicaciones y gestionar las sesiones de usuario de manera segura. El servidor se ha desplegado en la plataforma Render, que permite gestionar de manera sencilla la infraestructura necesaria para la aplicación. Render proporciona un entorno de despliegue robusto y escalable, adecuado para las necesidades del proyecto.

5.1.2. Base de Datos

Para la persistencia de datos en el proyecto FitFuelBalance, se ha utilizado PostgreSQL, un sistema de gestión de bases de datos relacional conocido por su robustez, flexibilidad y soporte para tipos de datos avanzados. A continuación, se describen los aspectos clave de la estructura de la base de datos y su integración con Django.

- **Diseño de la Base de Datos:** El diseño de la base de datos ha sido cuidadosamente planificado para asegurar la eficiencia y escalabilidad. Los principales esquemas de la base de datos incluyen las siguientes tablas (no se incluyen todas las columnas para simplificar la descripción):
  - **CustomUser:** Almacena información de los usuarios. Sus propiedades son:
    - **id (Integer):** Identificador único del usuario.
    - **password (String):** Contraseña hasheada del usuario.
    - **last_login (Datetime):** Fecha y hora del último inicio de sesión.
    - **is_superuser (Boolean):** Indica si el usuario es un superusuario.
    - **username (String):** Nombre de usuario único.
    - **first_name (String):** Nombre del usuario.
    - **last_name (String):** Apellido del usuario.
    - **email (String):** Correo electrónico del usuario.

- ◇ **is_staff (Boolean):** Indica si el usuario es parte del staff.
- ◇ **is_active (Boolean):** Indica si la cuenta del usuario está activa.
- ◇ **date_joined (Datetime):** Fecha y hora de creación de la cuenta.

- **RegularUser:** Extiende `CustomUser` y almacena medidas corporales y datos de los usuarios normales. Sus propiedades son:
  - ◇ **customuser_ptr_id (Integer):** Identificador único del usuario regular.
  - ◇ **weight (Float):** Peso del usuario.
  - ◇ **height (Float):** Altura del usuario.
  - ◇ **personal_trainer_id (Integer):** Identificador del entrenador personal asignado.
  - ◇ **arm (Float):** Medida del brazo.
  - ◇ **chest (Float):** Medida del pecho.
- **Trainer:** Extiende `CustomUser` y almacena información adicional sobre los entrenadores. Sus propiedades son:
  - ◇ **customuser_ptr_id (Integer):** Identificador único del entrenador.
  - ◇ **trainer_type (String):** Tipo de entrenador.
- **Food:** Almacena información detallada sobre alimentos. Sus propiedades son:
  - ◇ **id (Integer):** Identificador único del alimento.
  - ◇ **name (String):** Nombre del alimento.
  - ◇ **unit_weight (Float):** Peso unitario del alimento.
  - ◇ **calories (Float):** Calorías por unidad.
  - ◇ **protein (Float):** Proteínas por unidad.
  - ◇ **carbohydrates (Float):** Carbohidratos por unidad.
  - ◇ **sugar (Float):** Azúcar por unidad.
  - ◇ **fiber (Float):** Fibra por unidad.
  - ◇ **fat (Float):** Grasa por unidad.
  - ◇ **saturated_fat (Float):** Grasa saturada por unidad.
  - ◇ **gluten_free (Boolean):** Indica si es libre de gluten.
  - ◇ **lactose_free (Boolean):** Indica si es libre de lactosa.
  - ◇ **contains_meat (Boolean):** Indica si contiene carne.
  - ◇ **contains_vegetables (Boolean):** Indica si contiene vegetales.
- **Diet:** Almacena los planes de dietas desarrollados por los nutricionistas. Sus propiedades son:
  - ◇ **id (Integer):** Identificador único del plan de dieta.
  - ◇ **name (String):** Nombre del plan de dieta.
  - ◇ **start_date (Datetime):** Fecha de inicio del plan de dieta.
  - ◇ **end_date (Datetime):** Fecha de finalización del plan de dieta.
  - ◇ **user_id (Integer):** Identificador del usuario al que pertenece el plan de dieta.
- **Option:** Registra las opciones de comidas para cada día de la semana. Sus propiedades son:
  - ◇ **id (Integer):** Identificador único de la opción.
  - ◇ **name (String):** Nombre de la opción.

- ◇ **trainer_id (Integer):** Identificador del entrenador que creó la opción.
- ◇ **week_option_one_id (Integer):** Identificador de la primera opción semanal.
- ◇ **week_option_two_id (Integer):** Identificador de la segunda opción semanal.
- ◇ **week_option_three_id (Integer):** Identificador de la tercera opción semanal.

- **Migraciones:** Django facilita la gestión de cambios en la estructura de la base de datos a través del sistema de migraciones. Cada vez que se realiza un cambio en los modelos, se crea una migración que, al aplicarse, sincroniza la base de datos con los nuevos cambios. Esto asegura que la base de datos esté siempre en línea con el código del proyecto.
- **ORM de Django:** Para interactuar con la base de datos, se utiliza el Object-Relational Mapping (ORM) de Django, que permite trabajar con la base de datos utilizando objetos Python en lugar de escribir consultas SQL directamente. Esto no solo simplifica el código sino que también mejora la portabilidad y mantenimiento del mismo. Código 5.1 muestra un ejemplo de consulta ORM para ver todos los entrenamientos.

Código 5.1: Ejemplo de consulta ORM en Django.

```
1  trainings = Training.objects.all()
```

- **Conexión a la Base de Datos:** La configuración de la conexión a la base de datos se encuentra en el archivo `settings.py` del proyecto Django, donde se especifican los detalles necesarios para conectarse a PostgreSQL. Código 5.2 muestra un ejemplo de configuración de la base de datos en Django.

Código 5.2: Configuración de la base de datos en Django.

```
5  DATABASES = {
6              'default': {
7                  'ENGINE': 'django.db.backends.postgresql',
8                  'NAME': 'fitfuelbalance_db',
9                  'USER': 'dbuser',
10                 'PASSWORD': 'password',
11                 'HOST': 'localhost',
12                 'PORT': '5432',
13             }
14 }
```

- **Gestión de Datos:** Para la administración y gestión de la base de datos, se ha utilizado Neon, una herramienta que facilita la visualización y manipulación de datos de PostgreSQL. Neon ofrece una interfaz gráfica intuitiva que permite realizar consultas, revisar tablas y gestionar esquemas de manera eficiente.
- **Seguridad y Copias de Seguridad:** Se han implementado medidas de seguridad para proteger los datos almacenados. Esto incluye la encriptación de contraseñas de usuarios utilizando el sistema de hash de Django y la realización periódica de copias de seguridad de la base de datos para prevenir pérdidas de datos.

Una imagen del diseño de la base de datos se puede ver en el Apéndice E en las figuras E.2 y E.3.

5.1.3. API REST

El desarrollo de la API RESTful en FitFuelBalance se ha realizado utilizando Django REST Framework (DRF). Esta API permite la comunicación entre el servidor y los clientes (tanto la aplicación web como la móvil) a través de solicitudes HTTP, proporcionando respuestas en formato JSON. A continuación, se detalla la estructura y los componentes principales de la API.

Endpoints y Rutas

Las rutas de la API están definidas en el archivo `urls.py` del proyecto. Estas rutas mapean las solicitudes HTTP a las vistas correspondientes, las cuales manejan la lógica de negocio. Código G.1 muestra un ejemplo de rutas en Django.

Vistas y Controladores

Las vistas en Django se encargan de manejar las solicitudes HTTP y devolver las respuestas correspondientes. En FitFuelBalance, las vistas se han implementado utilizando clases basadas en vistas (Class-Based Views) y ViewSets de DRF para facilitar la reutilización y el manejo de lógica compleja. Código G.2 muestra un fragmento del código la vistas del proyecto en Django.

Serializadores

Los serializadores de DRF se utilizan para convertir instancias de modelos de Django a formatos JSON y viceversa. Esto es esencial para la comunicación entre el servidor y los clientes. Código G.3 muestra un ejemplo de estos serializadores en Django.

Métodos HTTP

La API maneja los siguientes métodos HTTP:

- **GET:** Solicita una representación de un recurso específico del servidor.
- **POST:** Envía un nuevo recurso al servidor.
- **PUT:** Envía un recurso existente al servidor con modificaciones.
- **DELETE:** Elimina un recurso del servidor.

Ejemplo de Solicitud y Respuesta

Código 5.3 muestra un ejemplo de solicitud GET en Python utilizando la librería `requests`. En este caso, se solicita la lista de alimentos disponibles en la API de FitFuelBalance.

Código 5.4 muestra un ejemplo de lo que podría ser la respuesta a la solicitud GET anterior. En este caso, se devuelve una lista de alimentos en formato JSON con sus propiedades.

Código 5.3: Ejemplo de solicitud GET en Python.

```
response = requests.get('https://fitfuelbalance.onrender.com/nutrition/foods/')
```

Código 5.4: Ejemplo de respuesta GET en JSON.

```
[{
    "id": 1,
    "name": "Apple",
    "calories": 52,
    "protein": 0.3,
    "carbohydrates": 14,
    "sugar": 10,
    "fiber": 2.4,
    "fat": 0.2,
    "image": "url/media/foods/apple.jpg"
}]
```

5.1.4. Tokens JWT

Para garantizar la seguridad y autenticidad de los usuarios en la plataforma FitFuelBalance, se ha implementado un sistema de autenticación basado en JSON Web Tokens (JWT). Este estándar de seguridad permite la comunicación segura entre dos partes mediante el uso de tokens firmados digitalmente. La implementación de JWT en este proyecto es crucial para asegurar que solo los usuarios registrados por un administrador puedan acceder a los recursos protegidos del sistema [38].

El proceso de autenticación con JWT en FitFuelBalance sigue los siguientes pasos: primero, el usuario ingresa sus credenciales en el formulario de inicio de sesión del front-end. Al enviar el formulario, se genera una solicitud HTTP POST al login. El controlador de inicio de sesión en el backend verifica las credenciales del usuario. Si son correctas, se genera un token firmado que contiene la identificación del usuario y otros datos relevantes. Este token se envía de vuelta al cliente y se almacena en el almacenamiento local del navegador.

Para cada solicitud posterior realizada por el cliente a recursos protegidos, el token se incluye en el encabezado de la solicitud HTTP. El middleware `TokenAuthentication` en el backend intercepta la solicitud y verifica la validez del token. Si el token es válido, se permite el acceso al recurso solicitado; de lo contrario, se devuelve una respuesta HTTP 401 Unauthorized. Figura 5.2 muestra un ejemplo de cómo funciona el sistema de verificación de tokens.

Figura 5.3 muestra un ejemplo de token JWT y como funciona la codificación de los datos en el token.

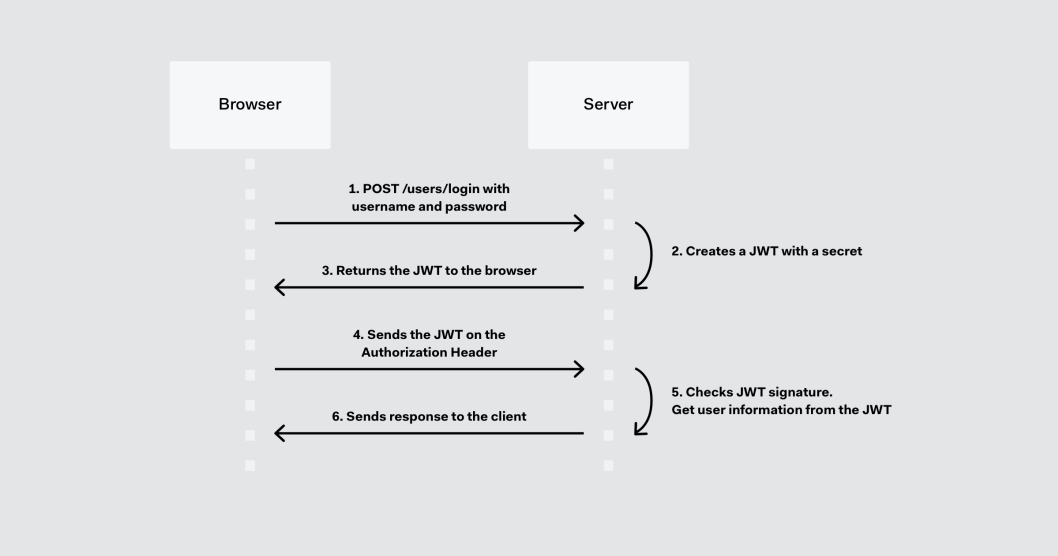

Figura 5.2: Diagrama del sistema de verificación de tokens [38]

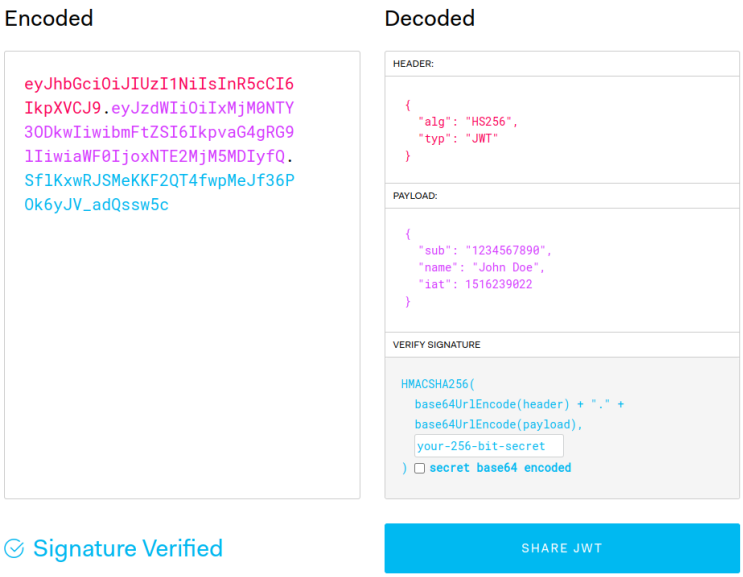

Figura 5.3: Ejemplo de token JWT [38]

Middleware y Configuración

El middleware `TokenAuthentication` es esencial para la autenticación de usuarios en cada solicitud. En el archivo `settings.py` se define la configuración necesaria para el uso de JWT en el proyecto Django, Código 5.5 muestra un ejemplo de configuración de JWT en Django.

Código 5.5: Configuración de JWT en Django.

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ],
}
```

En el archivo `views.py`, se define la vista para el manejo de la autenticación utilizando tokens JWT, Código G.4 muestra un ejemplo de vista de autenticación en Django.

Uso de Tokens en el Front-End

En el archivo `App.js` del Front-end, se maneja el almacenamiento del token y su uso en las solicitudes a la API, Código G.6 muestra un ejemplo de uso de tokens en el Front-end.

5.2. Front-End

El Front-End del proyecto FitFuelBalance ha sido desarrollado utilizando React.js para la aplicación web y React Native para la aplicación móvil. Estas tecnologías han sido elegidas por su flexibilidad, capacidad de reutilización de componentes y su amplia adopción en la industria, lo cual asegura una comunidad activa y numerosos recursos disponibles. En esta sección se detallarán los aspectos principales del desarrollo del Front-End, incluyendo la estructura de la aplicación web y la implementación de la aplicación móvil. A continuación Figura 5.4 se muestra la estructura del Front-End de la aplicación FitFuelBalance.

5.2.1. Aplicación Web

La aplicación web de FitFuelBalance ① ha sido construida utilizando React.js, una biblioteca de JavaScript para construir interfaces de usuario. React permite crear aplicaciones web rápidas y eficientes, gracias a su enfoque en componentes reutilizables y su capacidad de manejar cambios de estado

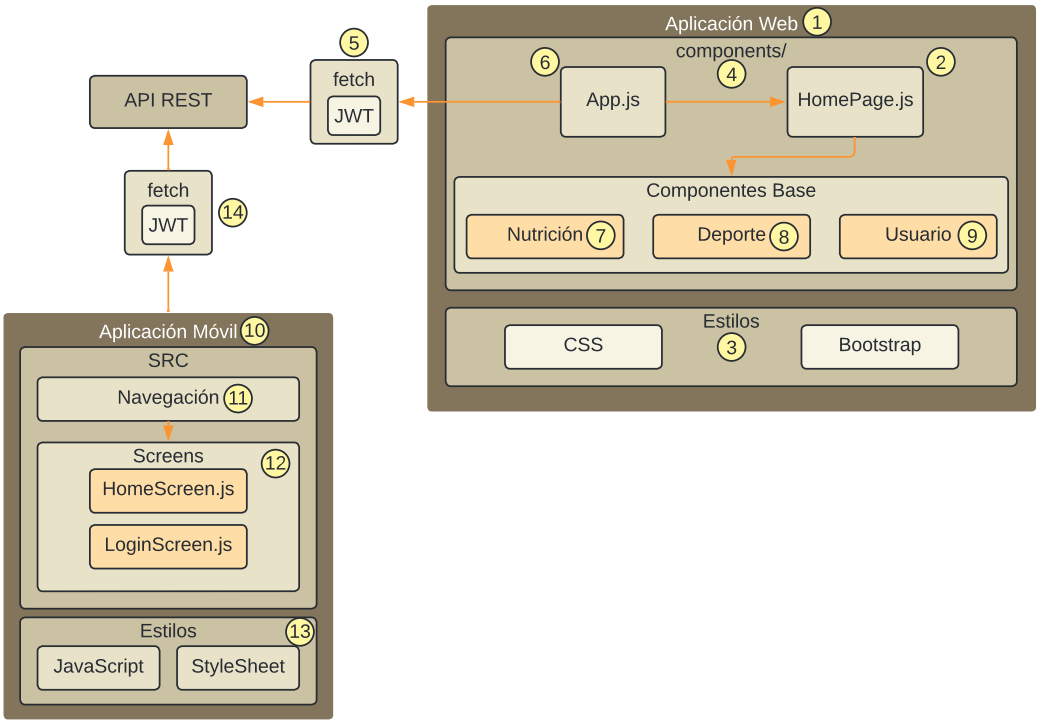

Figura 5.4: Estructura del Front-End de FitFuelBalance

de manera eficiente. Los componentes principales definidos en la aplicación incluyen el componente de encabezado (Header), que incluye la navegación principal; el componente HomePage.js ②, que se encarga de la navegación a todos los componentes principales; y el componente App.js, que define las rutas y la estructura general de la aplicación. Para los estilos ③, se ha utilizado Bootstrap junto con CSS personalizado para asegurar una apariencia moderna y responsiva en toda la aplicación. Bootstrap proporciona una base sólida de componentes y utilidades que aceleran el desarrollo y aseguran la consistencia visual. La integración con el backend de Django se realiza a través de una serie de servicios definidos en la carpeta `components/` ④, los cuales utilizan `fetch` ⑤ para realizar solicitudes HTTP y manejar las respuestas. El manejo del estado global de la aplicación se ha implementado utilizando `Context API` de React, lo que permite compartir datos y estados entre componentes sin necesidad de pasar props manualmente en múltiples niveles de la jerarquía de componentes. La autenticación de usuarios se maneja mediante JWT (JSON Web Tokens); al iniciar sesión, el backend proporciona un token que se almacena en el almacenamiento local del navegador y se adjunta a las cabeceras de las solicitudes HTTP subsecuentes para validar la identidad del usuario.

5.2.2. Componentes

En esta sección se detallarán los principales componentes, estos son los elementos que componen la interfaz de usuario de la aplicación web de FitFuelBalance. Cada componente tiene una función específica y contribuye a la funcionalidad general de la aplicación. A continuación se describen los

componentes más importantes y su función en la aplicación.

App.js

El componente principal `App.js` ⑥ es el núcleo de la aplicación web. En este archivo se define la estructura general de la aplicación, incluyendo la barra de navegación, las rutas y la gestión del estado de autenticación del usuario. Utiliza `useState` para gestionar el estado de autenticación del usuario, almacenando el token de autenticación en el `localStorage`. Utiliza `useEffect` para obtener el perfil del usuario desde el backend al cargar el componente si existe un token de autenticación válido. Maneja el inicio y cierre de sesión, almacenando y eliminando el token de autenticación en el `localStorage`. Además, permite buscar entrenadores basados en especialidades y tipo de entrenador, enviando una solicitud HTTP al backend y actualizando el estado con los resultados. También envía solicitudes a entrenadores específicos mediante una solicitud HTTP POST al backend.

El componente `App` se organiza en varias secciones clave: la barra de navegación, implementada utilizando componentes de `react-bootstrap`, permite al usuario acceder rápidamente a las diferentes secciones de la aplicación, como Nutrición y Deporte. Utiliza `react-router-dom` para definir las rutas de la aplicación. Cada ruta está asociada a un componente específico que se renderiza cuando el usuario navega a esa ruta. Las principales vistas y componentes incluyen `HomePage`, `Login`, `Profile`, entre otros.

Componentes de Nutrición

Los componentes de la sección de nutrición ⑦ en la aplicación web de FitFuelBalance son fundamentales para gestionar y administrar los planes de nutrición. Los componentes clave incluyen `CreateDiet.js`, que permite a los nutricionistas crear planes de dieta completos; `CreateDish.js`, utilizado para crear platos individuales que se pueden incluir en los planes de dieta; `EditDiet.js`, que facilita la edición de planes de dieta existentes; y `ListFood.js`, que presenta una lista completa de todos los alimentos en la base de datos, con funcionalidades de búsqueda y filtrado. Estos componentes permiten a los nutricionistas definir comidas específicas, ajustar porciones y modificar ingredientes según las necesidades del usuario.

Componentes de Deporte

Los componentes de deporte ⑧ en la aplicación web están diseñados para gestionar y visualizar los diferentes aspectos relacionados con los ejercicios y entrenamientos. Los componentes clave incluyen `CreateExercise.js`, que permite a los entrenadores crear nuevos ejercicios; `CreateTraining.js`, utilizado para crear planes de entrenamiento completos; `EditExercise.js`, que permite a los entrenadores editar los detalles de un ejercicio existente; y `ListExercise.js`, que proporciona una lista de todos los ejercicios disponibles en la base de datos. Estos componentes permiten a los entre-

nadores definir, ajustar y visualizar ejercicios y planes de entrenamiento.

Componentes de Usuario

Los componentes relacionados con la gestión de usuarios (9) en FitFuelBalance son esenciales para el manejo de la autenticación y los perfiles de usuario. El componente `Login.js` maneja el inicio de sesión de los usuarios, verificando las credenciales y gestionando los tokens JWT. `ManageClients.js` proporciona una interfaz para que los entrenadores gestionen la lista de sus clientes, actualizando información y asignando planes de entrenamiento o dietas. `Profile.js` permite a los usuarios ver y actualizar su perfil personal. Otros componentes importantes incluyen `RegularSignUp.js`, para el registro de nuevos usuarios normales, y `SearchTrainer.js`, que permite a los usuarios buscar entrenadores y nutricionistas en la plataforma.

5.2.3. Aplicación Móvil

La aplicación móvil (10) de FitFuelBalance ha sido desarrollada utilizando React Native, un framework que permite crear aplicaciones móviles nativas para iOS y Android utilizando JavaScript y React. Esta tecnología ha sido seleccionada por su capacidad de compartir código entre plataformas y su excelente rendimiento en dispositivos móviles. La Figura E.5 del Apéndice E muestra la estructura general de la aplicación móvil.

La navegación (11) en la aplicación móvil se ha implementado utilizando React Navigation. Esta biblioteca facilita la gestión de la navegación entre pantallas y la organización de la estructura de navegación de la aplicación. Código G.11 muestra un ejemplo de configuración de navegación.

Las pantallas de la aplicación móvil se encuentran en la carpeta `screens/` (12) y contienen los componentes principales de la aplicación. Algunas de las pantallas más importantes son: `HomeScreen.js`, `CalendarScreen.js` y `LoginScreen.js`. Estas pantallas se encargan de mostrar la información y permitir la interacción del usuario con la aplicación.

Para los estilos (13) de la aplicación móvil se ha utilizado una combinación de hojas de estilo en JavaScript y StyleSheet de React Native. Esto permite definir estilos que se aplican de manera consistente en toda la aplicación y aprovechar características específicas de la plataforma.

Similar a la aplicación web, la aplicación móvil se comunica con el backend de Django a través de servicios definidos en la carpeta `api/`. Estos servicios utilizan `fetch` (14) para realizar solicitudes HTTP y manejar las respuestas. Código G.10 muestra un ejemplo de un servicio para obtener planes de entrenamiento.

5.3. Herramientas y Entorno de Desarrollo

El desarrollo de FitFuelBalance ha requerido el uso de diversas herramientas y entornos para facilitar el proceso de programación y despliegue. A continuación, se describen las principales herramientas y entornos utilizados:

Control de Versiones

Para el control de versiones se ha utilizado Git junto con la plataforma GitHub. Git es un sistema de control de versiones distribuido que permite a los desarrolladores realizar un seguimiento de los cambios en el código fuente y colaborar de manera eficiente. GitHub, por su parte, proporciona un entorno de alojamiento de repositorios Git con características adicionales como la gestión de issues, pull requests y la integración continua. El código fuente del proyecto se encuentra alojado en un repositorio privado en GitHub, lo que permite realizar un seguimiento detallado de los cambios, gestionar versiones y colaborar con otros desarrolladores de manera eficiente. Se han utilizado ramas (branches) para organizar el desarrollo de nuevas funcionalidades y la corrección de errores. La rama `main` contiene la versión estable del proyecto, mientras que las ramas que contienen la aplicación en producción (`primer-deploy/`), versiones anteriores estables (`beforeRemake/`) y versiones para probar funcionalidades nuevas (`modelos-unicos/`) se utilizan para trabajar en paralelo sin afectar la versión principal.

Entorno de Desarrollo

Para el desarrollo del proyecto se ha utilizado el editor de código Visual Studio Code, una herramienta potente y versátil que ofrece una amplia gama de extensiones y características para facilitar la programación.

Diseño de la Interfaz de Usuario

Para el diseño de la interfaz de usuario se ha utilizado Figma, una herramienta de diseño colaborativa que permite crear prototipos y diseños de alta fidelidad.

Base de Datos

Para la gestión de la base de datos se ha utilizado Neon, una plataforma basada en PostgreSQL que facilita la creación y administración de bases de datos en la nube.

Despliegue

El despliegue de las aplicaciones web y backend se ha realizado utilizando Render, una plataforma de despliegue en la nube que facilita el proceso de publicación de aplicaciones. Esta plataforma ha sido seleccionada por su simplicidad y eficiencia en el despliegue de aplicaciones. Render permite automatizar el despliegue a partir del código fuente alojado en GitHub, asegurando que las últimas versiones del proyecto estén siempre disponibles en producción.

Testing y Depuración

Durante el desarrollo del proyecto, se han utilizado diversas herramientas y técnicas para realizar pruebas y depuración de código. React Native Debugger [39] se ha utilizado para depurar la aplicación móvil desarrollada en React Native. Postman [40] ha sido la herramienta utilizada para probar las API desarrolladas en Django y asegurarse de que respondan correctamente a las solicitudes. Pytest [41] es el framework de pruebas para Python, utilizado para crear y ejecutar pruebas automatizadas del backend. Google Chrome DevTools [42] ha sido la herramienta de desarrollo integrada en el navegador Chrome.

Gestión de Dependencias

Para gestionar las dependencias del proyecto se han utilizado herramientas específicas para cada entorno de desarrollo. `pip` [43] se ha utilizado para gestionar las dependencias del backend desarrollado en Python y Django. `npm` [44] se ha utilizado para gestionar las dependencias del frontend desarrollado en React y React Native.

6

Pruebas e Implementación

La siguiente sección detalla las pruebas realizadas en el proyecto FitFuelBalance, incluyendo pruebas del back-end y pruebas del front-end. Todas las pruebas han sido realizadas en entornos controlados aparte del entorno de producción, para asegurar que el sistema funcione correctamente y sin errores. También aclarar que para la realización de las pruebas se ha desactivado el uso tokens de autenticación para facilitar el acceso a los endpoints.

6.1. Pruebas del Back-End

Las pruebas del back-end se han centrado en verificar la funcionalidad de la API y asegurar que todas las operaciones del servidor se realicen correctamente. Para esto, se han utilizado herramientas como *pytest* para ejecutar pruebas automatizadas y *Postman* para realizar pruebas manuales de los distintos endpoints de la API. Se ha creado un testsettings.py para configurar las pruebas de Django. Todas las pruebas se encuentran en la sección G.4 del Apéndice G.

Configuración de Pruebas

Para realizar las pruebas, se configuró un entorno de pruebas separado en el que se llevaron a cabo pruebas. Como se ha comentado anteriormente, para no causar problemas con el proyecto en producción todas las pruebas han sido realizadas en un entorno y una base de datos separada. Se ha creado un archivo de configuración de pruebas testsettings.py para configurar las pruebas de Django. En este archivo se han configurado las credenciales de la base de datos de pruebas y se han desactivado las migraciones automáticas para evitar conflictos con la base de datos de producción. La configuración de este archivo se muestra en Código G.13. La configuración de los tests de usuario se representan en Código G.14. La de deporte en la Código G.16 y la de nutrición en Código G.18.

Pruebas de Endpoints

Se han realizado pruebas a todos los endpoints críticos del back-end, asegurando que cada uno de ellos responda adecuadamente y maneje los errores de manera eficiente. Todas las aplicaciones del proyecto han sido probadas, incluyendo las aplicaciones de usuario, deporte y nutrición. Código G.15 muestra un ejemplo de pruebas realizadas a la aplicación de usuario. Código G.17 muestra un ejemplo de pruebas realizadas a la aplicación de deporte y Código G.19 muestra un ejemplo de pruebas realizadas a la aplicación de nutrición.

6.2. Pruebas del Front-End

Las pruebas del front-end son esenciales para asegurar que la interfaz de usuario sea intuitiva, funcional y libre de errores. Para esto, se han utilizado diferentes técnicas y herramientas para probar tanto la aplicación web como la aplicación móvil.

Pruebas de la Aplicación Web

Para la aplicación web, se ha utilizado *Jest* y *React Testing Library* para realizar las pruebas de sus componentes. Estas herramientas permiten simular eventos y verificar que los componentes reaccionen adecuadamente a diferentes interacciones del usuario.

Prueba de Componentes

Se ha realizado una serie de pruebas a los componentes principales de la aplicación web para asegurarse de que se rendericen correctamente y manejen los eventos de usuario como se espera. Los tests se encuentran en la sección G.5 del Apéndice G.

Pruebas de la Aplicación Móvil

Para la aplicación móvil desarrollada con *React Native*, se han utilizado herramientas como *Jest* y *React Native Testing Library* para realizar pruebas unitarias. Además, se ha utilizado *React Native CLI* para probar la aplicación en diferentes dispositivos y asegurar la compatibilidad.

Prueba de Componentes Móviles

Las pruebas de componentes móviles aseguran que los elementos de la interfaz de usuario se rendericen correctamente en diferentes tamaños de pantalla y manejen las interacciones del usuario de manera adecuada. Las pruebas se encuentran en la subsección G.5.4 de la sección G.5 del Apéndice G.

6.3. Implementación

La implementación de la plataforma ha sido un proceso que ha involucrado varias fases, desde el desarrollo inicial hasta el despliegue en un entorno de producción. Esta sección describe las principales etapas del proceso de implementación, incluyendo la configuración de los servidores, el despliegue de la base de datos y las aplicaciones, y la integración continua.

Despliegue del Back-End

El back-end de FitFuelBalance, desarrollado en Django, ha sido desplegado en la plataforma Render, que proporciona un entorno escalable y fácil de gestionar. Los pasos principales seguidos durante el despliegue incluyen la configuración del entorno, donde se configuraron las variables de entorno necesarias, incluyendo las credenciales de la base de datos y las claves secretas de la aplicación. El despliegue del código se manejó utilizando Git para el control de versiones y despliegue continuo, con el código desplegado desde el repositorio de GitHub a Render. Las migraciones de la base de datos se ejecutaron para configurar la estructura de la base de datos en PostgreSQL. Finalmente, se realizaron pruebas post-despliegue para asegurar que el servidor estuviera funcionando correctamente y todas las API estuvieran accesibles.

Despliegue del Front-End

El front-end de la plataforma, desarrollado en React, también fue desplegado en la plataforma Render. Los pasos seguidos incluyen la configuración del proyecto, donde se configuraron las variables de entorno necesarias para la comunicación con el back-end. El despliegue del código se manejó similar al back-end, utilizando GitHub para el despliegue continuo. Cada push al repositorio de producción desencadena un nuevo despliegue en Render. Se realizaron optimizaciones de compilación para mejorar el rendimiento y reducir el tiempo de carga de la aplicación. Finalmente, se verificó que la aplicación web estuviera funcionando correctamente y se realizaron pruebas de rendimiento.

7 Conclusiones y Trabajo Futuro

7.1. Conclusiones

El desarrollo de FitFuelBalance ha sido un proceso desafiante y gratificante que ha culminado en la creación de una plataforma robusta y accesible para la gestión de planes de nutrición y ejercicio. A través de la combinación de tecnologías modernas y un enfoque centrado en el usuario, se ha logrado proporcionar una herramienta que facilita el acceso a planes personalizados, mejorando así la salud y el bienestar de los usuarios. Entre los logros más destacados del proyecto se encuentran el desarrollo integral, que incluye la implementación de un back-end sólido utilizando Django y una base de datos PostgreSQL, permitiendo una gestión eficiente y segura de los datos; la interfaz de usuario intuitiva, lograda mediante el uso de React para el desarrollo del front-end y React Native para la aplicación móvil, resultando en interfaces de usuario intuitivas y responsivas; el despliegue eficiente, facilitado por la utilización de plataformas como Render y Neon para el despliegue continuo y la gestión de la infraestructura; y la personalización de planes, permitiendo a los entrenadores y nutricionistas crear y ajustar planes personalizados de manera efectiva, atendiendo a las necesidades específicas de cada usuario. Este proyecto demuestra cómo la tecnología puede ser utilizada para mejorar significativamente la accesibilidad y eficacia de los servicios de nutrición y entrenamiento, promoviendo estilos de vida más saludables.

7.2. Trabajo Futuro

Aunque se han alcanzado los objetivos iniciales, hay margen para la mejora y expansión. A continuación, se destacan algunas áreas clave para el trabajo futuro:

7.2.1. Mejoras en el Front-End

Se planea continuar mejorando la experiencia del usuario mediante la optimización de la interfaz y la inclusión de nuevas funcionalidades interactivas, así como mejorar la estética y el diseño de la

aplicación web y móvil para hacerla más atractiva y moderna.

7.2.2. Mejoras en la Aplicación Móvil

Se busca asegurar compatibilidad con cualquier tipo de dispositivos móviles y optimizar el rendimiento para una experiencia de usuario fluida, además de integrar notificaciones push para mantener a los usuarios informados sobre sus planes y recordatorios de actividades.

7.2.3. Nuevas Funcionalidades

Se propone desarrollar un sistema de recomendaciones personalizadas utilizando inteligencia artificial (IA), que pueda sugerir ajustes en los planes de nutrición y ejercicio basados en el progreso y las preferencias del usuario. También se busca implementar funcionalidades de seguimiento en tiempo real que permitan a los usuarios y entrenadores monitorizar el progreso de las actividades en directo.

7.2.4. Integración de Inteligencia Artificial

Una de las mejoras más prometedoras para el futuro de FitFuelBalance es la integración de IA, que podría incluir la generación automática de planes utilizando algoritmos de IA adaptados a las características individuales de los usuarios como peso, altura, alergias y objetivos específicos; la adaptación de planes existentes por parte de los entrenadores utilizando IA para adaptar planes a nuevos clientes con características diferentes; y el desarrollo de un asistente virtual que pueda interactuar con los usuarios, responder preguntas comunes y proporcionar soporte en tiempo real.

Bibliografía

[1] "Physical activity." Accessed: 2024-06-07.

[2] "Centers for disease control and prevention." Accessed: 2024-06-07.

[3] "British journal of sports medicine." Accessed: 2024-06-07.

[4] "Rebrandx." Accessed: 2024-06-07.

[5] "Django: The web framework for perfectionists with deadlines." Accessed: 2024-06-07.

[6] "React – a javascript library for building user interfaces." Accessed: 2024-06-07.

[7] "React native – create native apps for android and ios using react." Accessed: 2024-06-07.

[8] "Postgresql: The world's most advanced open source database." Accessed: 2024-06-07.

[9] "Welcome to python.org." Accessed: 2024-06-07.

[10] "Javascript.com." Accessed: 2024-06-07.

[11] "Bootstrap · the most popular html, css, and js library in the world.." Accessed: 2024-06-07.

[12] "Css: Cascading style sheets." Accessed: 2024-06-07.

[13] "Myfitnesspal." Accessed: 2024-06-07.

[14] "Yazio - your calorie counter." Accessed: 2024-06-07.

[15] "Nike training club." Accessed: 2024-06-07.

[16] "Fitbod - personal workout and fitness plans." Accessed: 2024-06-07.

[17] "Calisteniapp - calisthenics and street workout." Accessed: 2024-06-07.

[18] "Fitbit official site for activity trackers and more." Accessed: 2024-06-07.

[19] "Samsung health." Accessed: 2024-06-07.

[20] "Nutrifix - your personalized nutrition guide." Accessed: 2024-06-07.

[21] "Platejoy - healthy eating, simplified." Accessed: 2024-06-07.

[22] "Freeletics - workouts and training plans." Accessed: 2024-06-07.

[23] "Jefit - workout, fitness, bodybuilding app." Accessed: 2024-06-07.

[24] "8fit - workouts, meal planner and personal trainer." Accessed: 2024-06-07.

[25] "Noom - health and wellness programs." Accessed: 2024-06-07.

[26] "Metodología incremental," 2017. Accessed: 2024-06-07.

[27] "Github - where the world builds software." Accessed: 2024-06-07.

[28] "Visual studio code - code editing. redefined." Accessed: 2024-06-07.

[29] "Figma - the collaborative interface design tool." Accessed: 2024-06-07.

[30] "React native cli - command line interface for react native." Accessed: 2024-06-07.

[31] "Neon - serverless, fault-tolerant, and developer-friendly postgres." Accessed: 2024-06-07.

[32] "Render - the easiest cloud for all your apps and websites." Accessed: 2024-06-07.

[33] "ios - apple." Accessed: 2024-06-07.

[34] "Android." Accessed: 2024-06-07.

[35] “Restful api - an architectural style for networked applications.” Accessed: 2024-06-07.

[36] “Json web tokens (jwt).” Accessed: 2024-06-07.

[37] “Axios - promise based http client for the browser and node.js.” Accessed: 2024-06-07.

[38] “Introduction to jwt.” Accessed: 2024-06-07.

[39] “React native debugger.” Accessed: 2024-06-07.

[40] “Postman - the collaboration platform for api development.” Accessed: 2024-06-07.

[41] “pytest - a mature full-featured python testing tool.” Accessed: 2024-06-07.

[42] “Google chrome devtools.” Accessed: 2024-06-07.

[43] “pip - the python package installer.” Accessed: 2024-06-07.

[44] “npm - node package manager.” Accessed: 2024-06-07.

[45] “Lucidchart - online diagram software and visual solution.” Accessed: 2024-06-07.

Apéndices

A

Dependecias y librerías

A.1. Acceso a FitFuelBalance

Para acceder a FitfuelBalance en su versión de producción, simplemente hay que acceder al link `https://fitfuelbalance.onrender.com` y registrarse (El Back-End se encuentra en (https://tfg-691w.onrender.com), pero no es necesario entrar para el uso de la aplicación). Una vez registrado, se puede acceder a la aplicación web como usuario o entrenador, ya que ambos roles están abiertos a cualquier usuario. También se proporcionan usuarios para acceder a la aplicación en la siguiente tabla:

Rol	**Username**	**Password**
Entrenador	trainer	Tfg123@
Usuario	user	Tfg123@

Tabla A.1: Usuarios para acceder a FitFuelBalance

La aplicación funciona en cualquier tipo de navegador, pero se recomienda no abrir varias sesiones con usuarios distintos en un mismo navegador para evitar errores.

A.2. Acceso al código fuente

El código fuente del Front-End, Back-End y Aplicación Móvil de FitFuelBalance se encuentra en los siguientes repositorios de GitHub:

- **Front-End:** `https://github.com/Monte-10/TFG/tree/main/fitfuel-app`
- **Back-End:** `https://github.com/Monte-10/TFG/tree/main/fitfuelbalance`
- **Aplicación Móvil:** `https://github.com/Monte-10/TFG/tree/main/fitfuelmobile`

B

Dependencias

Para la realización de este proyecto se ha utilizado una serie de herramientas y tecnologías que han permitido el desarrollo de la aplicación FitFuelBalance. A continuación, se detallan las dependencias necesarias para la instalación y ejecución del proyecto tanto en el Back-End, Front-End como en la aplicación móvil.

B.0.1. Dependencias Back-End

Nombre	Descripción	Versión
asgiref	Implementación ASGI de referencia.	3.7.2
Django	Framework web de alto nivel para Python.	4.2.6
django-cors-headers	Manejo de encabezados CORS en Django.	4.3.1
django-debug-toolbar	Herramientas de depuración para Django.	4.3.0
django-filter	Filtros para vistas basadas en clases.	23.5
djangorestframework	Framework para construir APIs web.	3.14.0
psycopg2-binary	Adaptador de base de datos PostgreSQL.	2.9.9
pandas	Análisis y manipulación de datos.	2.2.0
numpy	Paquete fundamental para computación científica.	1.26.2
django-bootstrap-datepicker-plus	Selector de fechas para Django con Bootstrap.	5.0.4
django-dynamic-formsets	Formsets dinámicos en Django.	0.0.8
django-widget-tweaks	Personalización de widgets en Django.	1.5.0
async-timeout	Administrador de contexto para timeouts asíncronos.	4.0.1
python-decouple	Configuración desacoplada de código.	3.6
dj-database-url	Utilidad para configurar bases de datos mediante URL.	0.5.0
gunicorn	Servidor WSGI para aplicaciones Python.	-
reportlab	Generación de documentos en PDF.	3.6.8

Tabla B.1: Dependencias Back-End

B.0.2. Dependencias Front-End

Nombre	Descripción	Versión
@testing-library/jest-dom	Utilidades personalizadas de DOM para pruebas con Jest.	5.17.0
@testing-library/react	Pruebas para componentes React.	13.4.0
@testing-library/user-event	Simulación de eventos del usuario para pruebas.	13.5.0
axios	Cliente HTTP basado en promesas.	1.7.2
bindings	Vinculación de módulos nativos en Node.js.	1.5.0
bootstrap	Framework CSS para desarrollo web responsivo.	5.3.3
chart.js	Biblioteca de gráficos en JavaScript.	4.4.3
file-uri-to-path	Conversión de URIs de archivos a rutas de archivos.	1.0.0
moment	Manipulación de fechas y horas en JavaScript.	2.30.1
nan	Utilidades para complementos nativos de Node.js.	2.19.0
papaparse	Parseo de archivos CSV en JavaScript.	5.4.1
react-big-calendar	Calendario para aplicaciones React.	1.12.2
react-bootstrap	Componentes Bootstrap para React.	2.10.2
react-chartjs-2	Componentes React para Chart.js.	5.2.0
react-dom	Enlace de React para el DOM.	18.3.1
react-router-bootstrap	Enlace de React Router para Bootstrap.	0.26.2
react-router-dom	Enrutador para aplicaciones web React.	6.23.1
react-scripts	Scripts de configuración para crear aplicaciones React.	3.4.4
react	Biblioteca para construir interfaces de usuario.	18.3.1
serve	Servidor estático simple para archivos.	14.2.3
web-vitals	Medición de métricas de rendimiento web.	2.1.4

Tabla B.2: Dependencias Front-End

B.0.3. Dependencias de la Aplicación Móvil

Nombre	Descripción	Versión
@babel/core	Núcleo de Babel para la compilación de JavaScript.	7.23.9
@babel/preset-env	Preset de Babel para compilar ECMAScript.	7.23.9
@babel/runtime	Dependencias de tiempo de ejecución de Babel.	7.23.9
@react-native-async-storage/async-storage	Almacenamiento asíncrono persistente para React Native.	1.22.2
@react-native/babel-preset	Preset de Babel para React Native.	0.73.21
@react-native/eslint-config	Configuración de ESLint para React Native.	0.73.2
@react-native/metro-config	Configuración de Metro para React Native.	0.73.5
@react-native/typescript-config	Configuración de TypeScript para React Native.	0.73.1
@react-navigation/bottom-tabs	Navegación por pestañas inferiores.	6.5.14
@react-navigation/native-stack	Navegación por pilas nativas.	6.9.20
@react-navigation/native	Navegación nativa para React Navigation.	6.1.12
jest	Marco de pruebas para JavaScript.	29.7.0
react-native-calendars	Componentes de calendario para React Native.	1.1303.0
react-native-elements	Biblioteca de componentes UI para React Native.	3.4.3
react-native-gesture-handler	Manejo de gestos para React Native.	2.15.0
react-native-safe-area-context	Manejo de áreas seguras en dispositivos.	4.9.0
react-native-screens	Manejo de pantallas para React Native.	3.29.0
react-native-sound	Reproducción de sonidos en React Native.	0.11.2
react-native	Framework para construir aplicaciones móviles nativas.	0.73.4
react-test-renderer	Renderizador para pruebas de componentes React.	18.2.0
react	Biblioteca para construir interfaces de usuario.	18.2.0
typescript	Lenguaje de programación con tipos estáticos opcionales.	5.0.4

Tabla B.3: Dependencias de la Aplicación Móvil

C

Requisitos

C.1. Requisitos Funcionales

C.1.1. Subsistema de Servidor y Comunicación con la Base de Datos

- **RF1** Será posible crear, modificar y eliminar modelos en el servidor que se almacenarán en la base de datos PostgreSQL.
- **RF2** Será posible crear y eliminar cualquier tabla en la base de datos PostgreSQL.
- **RF3** Será posible consultar, crear, editar y eliminar cualquier registro de las diferentes tablas en la base de datos mediante llamadas a través de la API REST.

C.1.2. Subsistema de Gestión de Usuarios

- **RF4** Un administrador podrá registrar, modificar y eliminar cualquier usuario.
- **RF5** Un administrador podrá registrar, modificar y eliminar a otros administradores.
- **RF6** Un usuario/administrador podrá acceder a un panel de administración de Django.
- **RF7** Un usuario podrá ver su información personal en un panel de usuario.
- **RF8** Un administrador podrá ver la información de cualquier usuario en su panel de administración.

C.1.3. Subsistema de Gestión de Entrenamientos y Nutrición

- **RF9** Un usuario podrá solicitar planes de entrenamiento y nutrición personalizados.
- **RF10** Un entrenador/nutricionista podrá crear, asignar y modificar planes de entrenamiento y nutrición.
- **RF11** Un usuario podrá visualizar y seguir los planes asignados por su entrenador/nutricionista.
- **RF12** Un entrenador/nutricionista podrá hacer seguimiento y ajustes en los planes de los usuarios.

C.2. Requisitos No Funcionales

C.2.1. Generales

- **RNF1** La aplicación tendrá una versión de desarrollo en la red local y otra de producción accesible desde cualquier navegador y dispositivo.

C.2.2. Errores

- **RNF2** En caso de error en cualquier transacción con la base de datos, se devolverá un error específico que será manejado de manera consistente.
- **RNF3** La API gestionará y transmitirá los errores de transacciones de la base de datos al cliente.
- **RNF4** La lógica del front-end manejará estos errores y los mostrará apropiadamente al usuario.

C.2.3. Seguridad

- **RNF5** Se implementará un sistema seguro basado en JSON Web Tokens, donde se asignará un token con cierta información a cada usuario.
- **RNF6** Se garantizará la autenticación basada en tokens para las operaciones de la base de datos solicitadas por la API.
- **RNF7** Las variables susceptibles se almacenarán de forma segura en variables de entorno.
- **RNF8** Las contraseñas de los usuarios se cifrarán mediante hashing con SHA256 antes de ser almacenadas en la base de datos.
- **RNF9** Las contraseñas cifradas nunca se devolverán en ninguna consulta.
- **RNF10** Ningún error debe conducir a la caída del sistema.

C.2.4. Interfaz y Usabilidad

- **RNF11** La aplicación tendrá un alto grado de usabilidad en dispositivos más pequeños como móviles o tabletas.
- **RNF12** Todas las vistas y componentes se adaptarán al tamaño del dispositivo que los reproduzca.
- **RNF13** Todas las vistas y componentes seguirán un código de color y forma coherente.
- **RNF14** Se utilizarán iconos e imágenes de uso libre junto con creaciones propias para proporcionar un mejor diseño.
- **RNF15** Para generar una notificación de alarma, se abrirá un panel de confirmación basado en texto para evitar çlics erróneos".
- **RNF16** Todas las acciones críticas como eliminaciones abrirán un panel de confirmación para evitar çlics erróneos".
- **RNF17** Se mostrarán mensajes de error en texto cuando no se encuentre una página o el acceso esté prohibido/restringido.

C.2.5. Soporte y Documentación

- **RNF18** La facilidad de acceso debe ser máxima simplemente ingresando la URL de la aplicación.
- **RNF19** No se requerirá instalación en la versión de ordenador (excepto para casos de prueba y simulación).

C.2.6. Regulatorio y Legal

- **RNF20** Todos los recursos externos utilizados serán de uso libre y estarán debidamente acreditados en la documentación del proyecto.

D

Diagramas de Flujo

En este apéndice se presentan los diagramas de los procesos más importantes del sistema. Estos diagramas se han realizado con la herramienta *Lucidchart* [45] y se han exportado a formato *pdf* para su inclusión en este documento.

D.0.1. Creación de un nuevo usuario

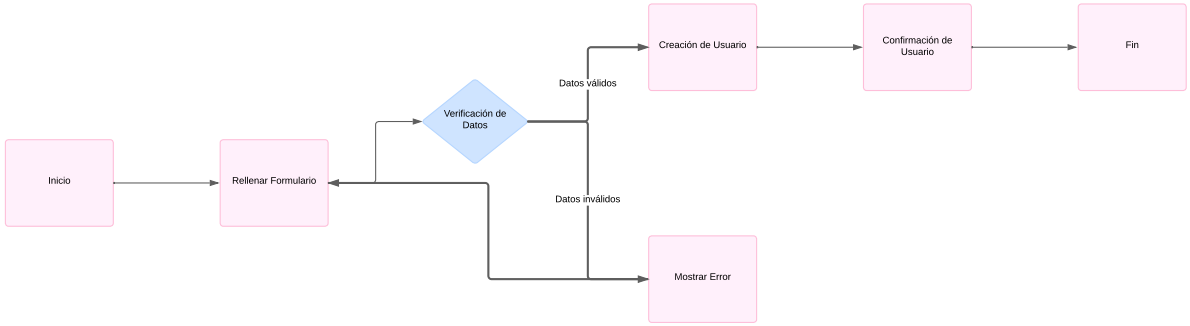

Figura D.1: Diagrama de creación de un nuevo usuario en FitFuelBalance

D.0.2. Creación de un nuevo entrenamiento

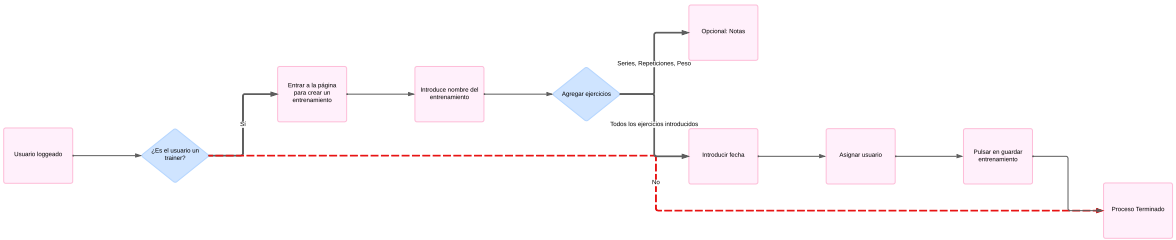

Figura D.2: Diagrama de creación de un nuevo entrenamiento en FitFuelBalance

D.0.3. Creación y asignación de una Opción

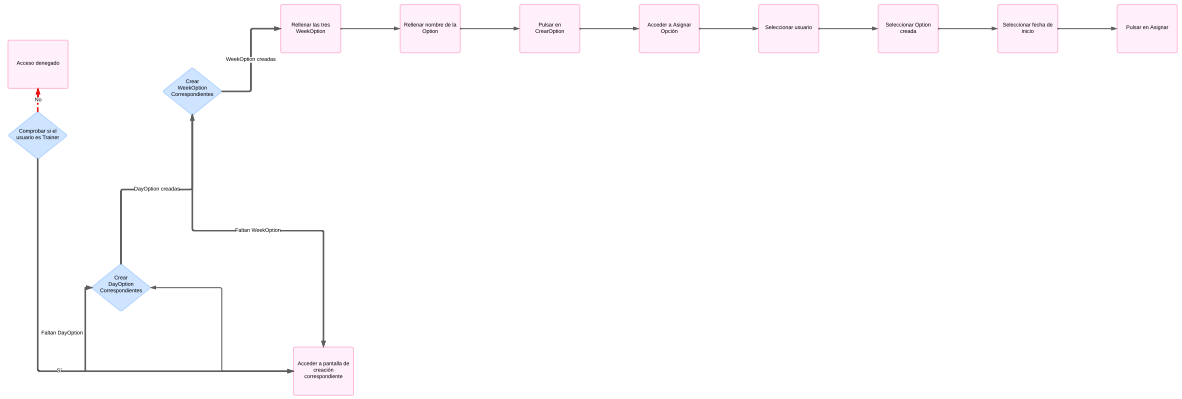

Figura D.3: Diagrama de creación y asignación de una opción en FitFuelBalance

D.0.4. Asignación de entrenador

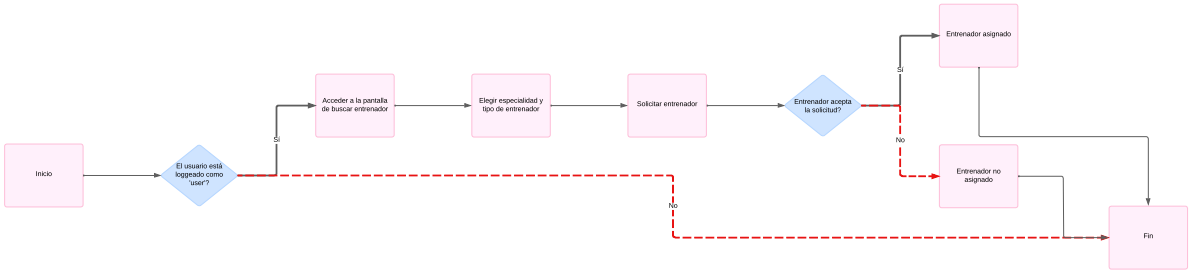

Figura D.4: Diagrama de asignación de entrenador en FitFuelBalance

E

Estructura del Proyecto

En este apéndice se presentan la estructura del backend y del frontend de la aplicación FitFuelBalance. Se han realizado con la herramienta *Lucidchart* y se han exportado a formato *pdf* para su inclusión en este documento.

E.0.1. Backend

El diagrama de la figura E.1 muestra la estructura del backend de la aplicación FitFuelBalance. En él se pueden ver los diferentes módulos que componen el backend de la aplicación, cada uno de los cuales tiene un propósito específico en el funcionamiento del sistema.

- fitfuelbalance: Contiene la configuración global de la aplicación, incluyendo ‘settings.py‘, ‘urls.py‘, y los archivos de arranque ‘wsgi.py‘ y ‘asgi.py‘.
- nutrition: Módulo que maneja toda la lógica relacionada con la nutrición, incluyendo modelos, vistas, serializadores, formularios y filtros.
- sport: Módulo que gestiona las funcionalidades relacionadas con el deporte, como la creación y edición de entrenamientos y ejercicios.
- user: Módulo dedicado a la gestión de usuarios, manejo de autenticación y perfil de usuario.
- media: Carpeta que contiene los archivos multimedia subidos por los usuarios, como imágenes de alimentos y ejercicios.
- templates: Contiene las plantillas HTML utilizadas en las vistas de Django.
- manage.py: Archivo principal de gestión de comandos de Django.
- requirements.txt: Archivo que lista todas las dependencias del proyecto necesarias para la instalación y ejecución del backend.

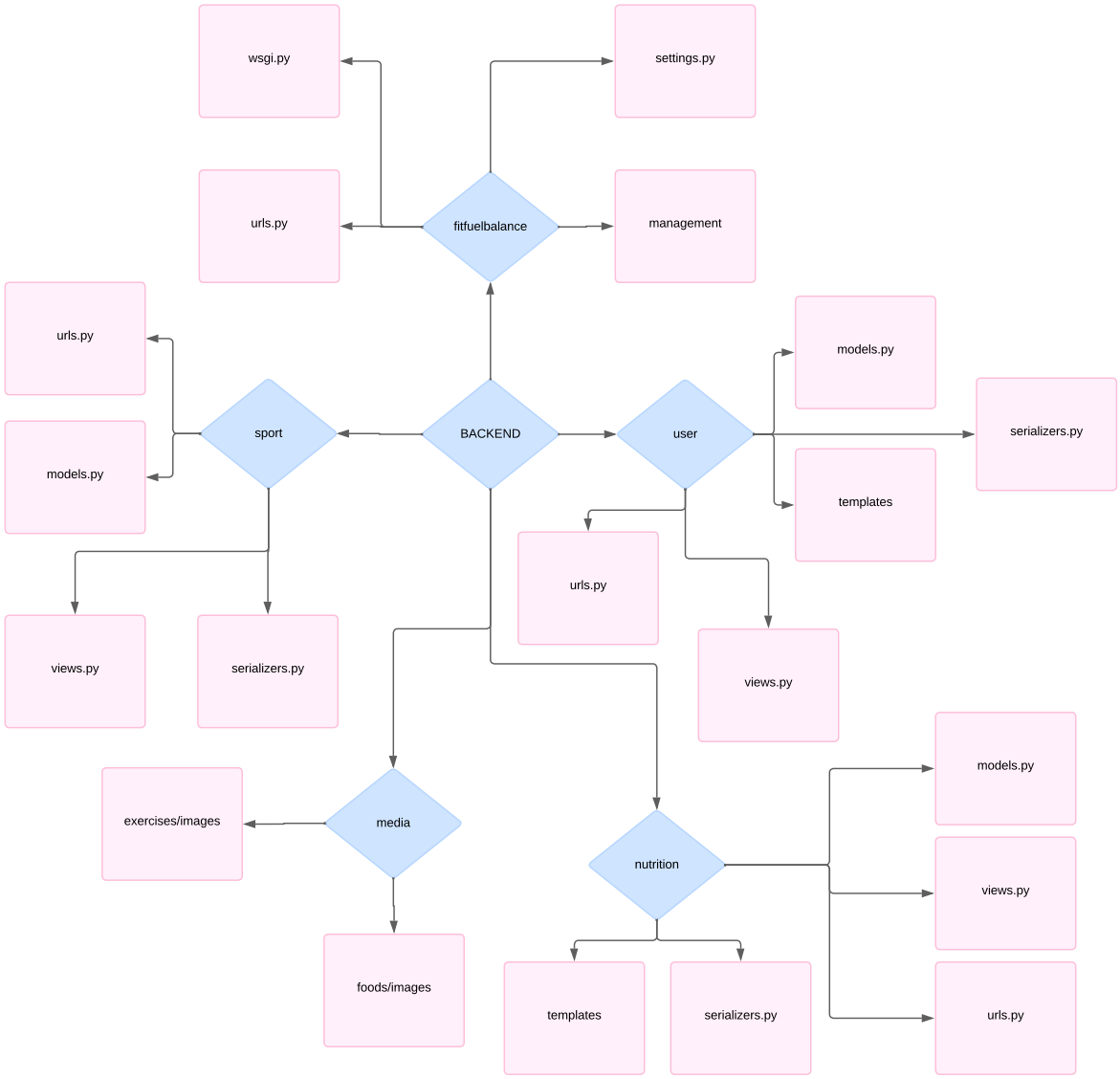

Figura E.1: Diagrama de la estructura del backend de FitFuelBalance

Base de Datos

Los siguientes diagramas representan el diseño de la base de datos, este diseño se ha dividido en dos partes facilitar su visionado. Las figuras E.2 y E.3 muestran las tablas y relaciones de la base de datos de FitFuelBalance.

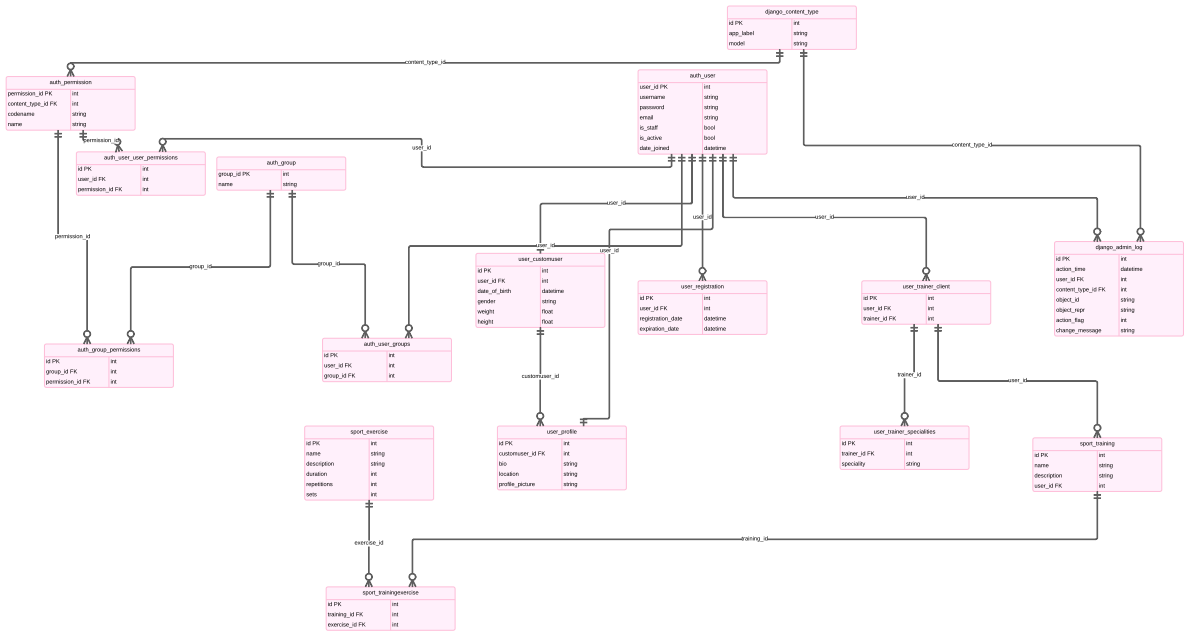

Figura E.2: Diseño de la base de datos de FitFuelBalance (1)

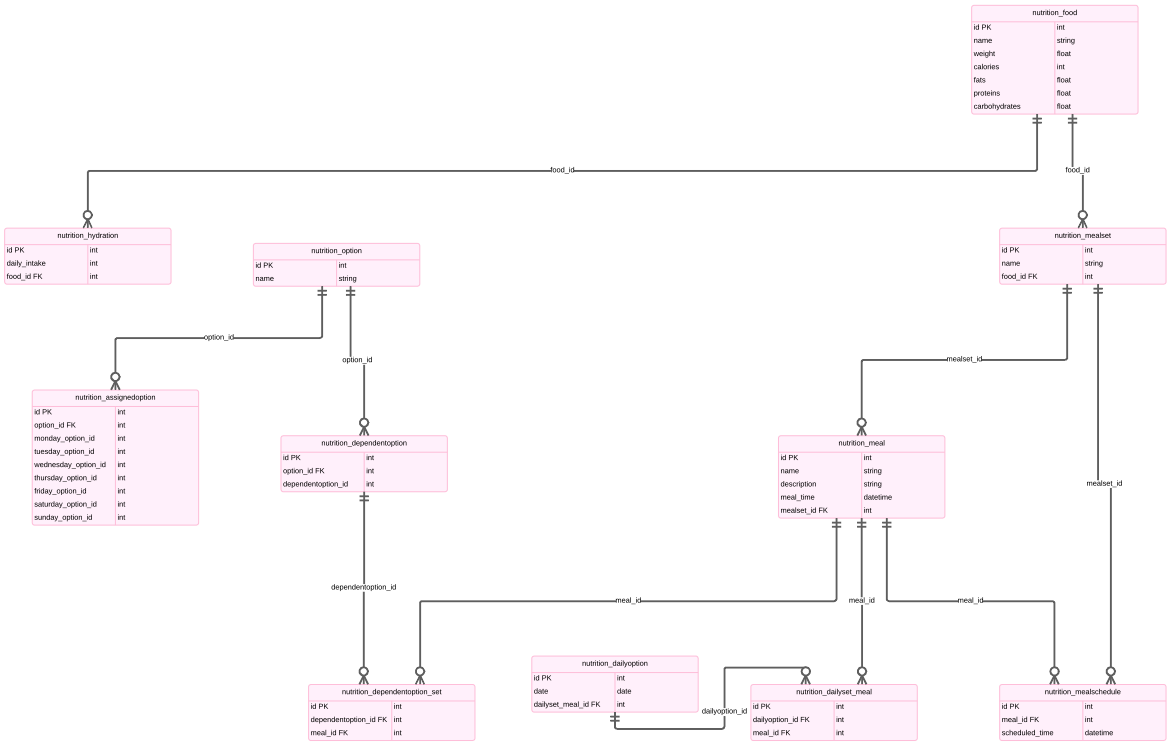

Figura E.3: Diseño de la base de datos de FitFuelBalance (2)

E.0.2. Frontend

El diagrama de la figura E.4 muestra la estructura del frontend de la aplicación FitFuelBalance. En él se pueden ver los diferentes módulos que componen el frontend de la aplicación, cada uno de los cuales contribuye a la interfaz de usuario y la interacción con el backend.

- src: Carpeta principal del código fuente del frontend.
  - App.js: Componente principal de la aplicación que contiene la estructura básica y las rutas principales.
  - App.css: Archivo de estilos principal que aplica estilos globales a la aplicación.
  - components: Contiene todos los componentes específicos de la aplicación.
    - HomePage.js: Componente que representa la página de inicio de la aplicación.
    - layout/Header.js: Componente que contiene la cabecera de la aplicación.
    - nutrition: Carpeta que contiene los componentes relacionados con la funcionalidad de nutrición, como 'AdaptDietOrOption.js', 'CreateDiet.js', 'ListFood.js', etc.
    - sport: Carpeta que contiene los componentes relacionados con la funcionalidad de deporte, como 'CreateExercise.js', 'ListTraining.js', etc.
    - user: Carpeta que contiene los componentes relacionados con la gestión de usuarios, como 'Login.js', 'Profile.js', 'RegularSignUp.js', etc.

    - css/style.css: Archivo de estilos CSS globales que se aplican a toda la aplicación.
- public: Carpeta que contiene los archivos estáticos y el archivo HTML principal.
- nodemodules: Carpeta que contiene todas las dependencias instaladas a través de npm.
- package.json: Archivo que contiene las dependencias y scripts del proyecto.

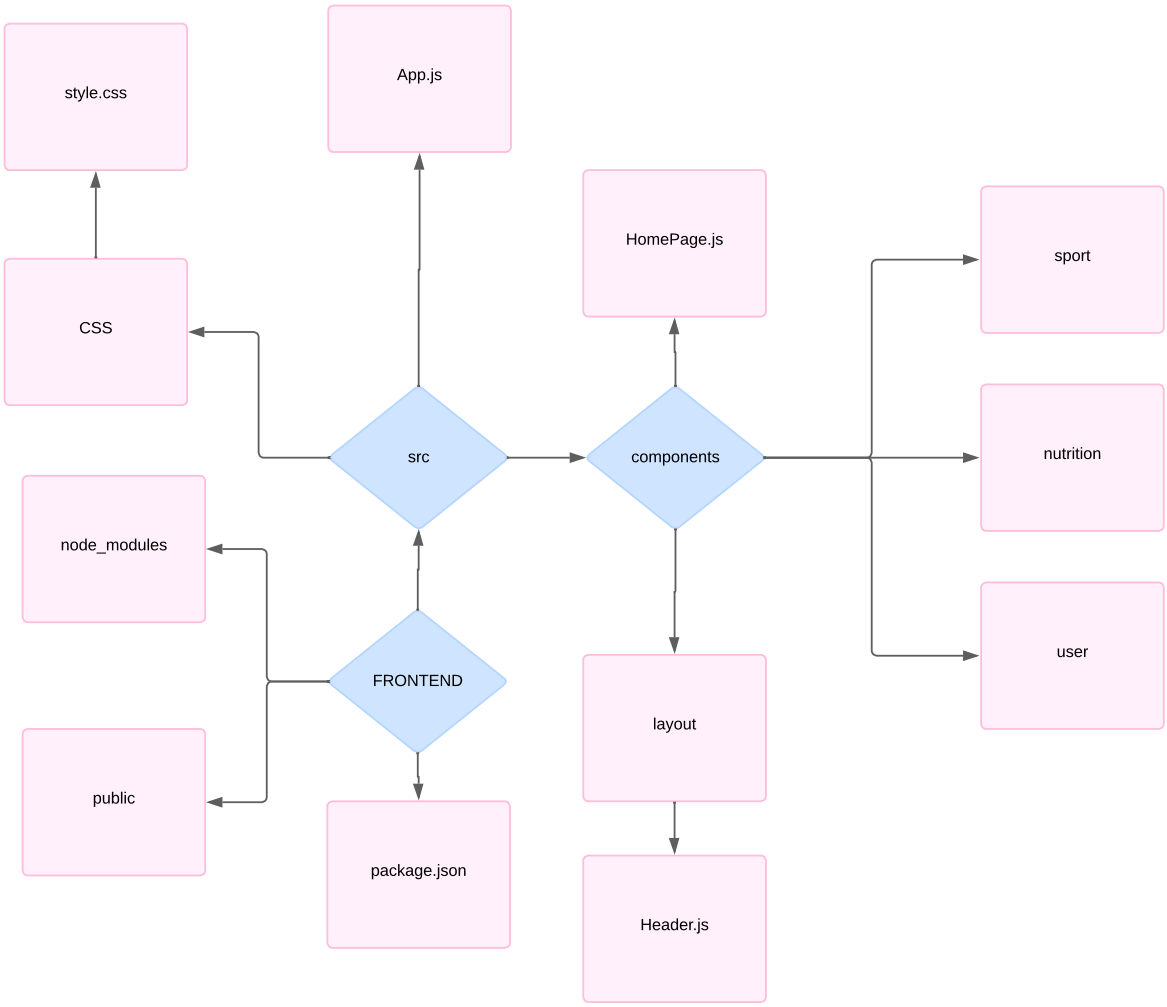

Figura E.4: Diagrama de la estructura del frontend de FitFuelBalance

Aplicación Móvil

A continuación se muestra un esquema de la estructura de la aplicación móvil:

- `src/` - Directorio principal del código fuente.
  - `api/` - Contiene los archivos relacionados con la API.
    - `trainingApi.js` - Archivo para las llamadas a la API relacionadas con los entrenamientos.
  - `AuthContext.js` - Contexto de autenticación.
  - `components/` - Contiene los componentes reutilizables de la UI.
    - `CountdownTimer.js` - Componente para un temporizador de cuenta regresiva.
  - `navigation/` - Configuración de la navegación de la aplicación.
    - `AppNavigator.js` - Archivo de configuración del navegador de la aplicación.
  - `screens/` - Contiene las diferentes pantallas de la aplicación, la más importantes son:
    - `ActiveTrainingScreen.js` - Pantalla de entrenamiento activo.
    - `CalendarScreen.js` - Pantalla del calendario.
    - `FoodDetailsScreen.js` - Pantalla de detalles de alimentos.
    - `HomeScreen.js` - Pantalla de inicio.
    - `LoginScreen.js` - Pantalla de inicio de sesión.
    - `TodayScreen.js` - Muestra los entrenamientos y comidas del día.
    - `TrainingDetailsScreen.js` - Pantalla de detalles del entrenamiento.

Figura E.5: Esquema de la estructura de la aplicación móvil

F

Capturas de Pantalla

G

Código Fuente

En esta sección se presentan los códigos implementados en el proyecto. Se incluirán los fragmentos de código más relevantes.

G.1. Back-End

G.1.1. Configuración de las Rutas y urls

Código G.1: Ejemplo de rutas en Django

```
from django.urls import path, include
    from rest_framework.routers import DefaultRouter

    router = DefaultRouter()
    router.register(r'foods', views.FoodViewSet)
    router.register(r'assigned_options', views.AssignedOptionViewSet)

    urlpatterns = [
        path('', include(router.urls)),
        path('assign_option/', views.assignOption, name='assign_option'),
    ]
```

G.1.2. Vistas y Controladores

Código G.2: Ejemplo de vistas en Django.

```
class TodayDailyDietView(APIView):
        def get(self, request, *args, **kwargs):
            today = timezone.now().date()
            user = request.user

            # Filtra las Dietas por el usuario logueado
            diets = Diet.objects.filter(user=user)

            # Encuentra las DailyDiet dentro del rango de fechas de las Dietas filtradas que coincidan con
                'hoy'
            today_diets = DailyDiet.objects.filter(diet__in=diets, date=today)

            # Serializa y devuelve los resultados
            serializer = DailyDietSerializer(today_diets, many=True)
            return Response(serializer.data)

@api_view(['POST'])
def assignOption(request):
    if not request.user.is_trainer:
        return Response({"error": "Solo los entrenadores pueden crear asignaciones."},
            status=status.HTTP_403_FORBIDDEN)

    user_id = request.data.get('userId')
    option_id = request.data.get('optionId')

    user = get_object_or_404(CustomUser, id=user_id)
    option = get_object_or_404(Option, id=option_id)

    assignment = AssignedOption.objects.create(
        user=user,
        option=option,
        start_date=timezone.now()
    )

    return Response({
        "message": "Option assigned successfully",
        "assignedOptionId": assignment.id, # ID de la asignación
        "optionId": option.id,
        "start_date": assignment.start_date.strftime("%Y-%m-%d")
    }, status=status.HTTP_201_CREATED)
```

G.1.3. Serializadores

Código G.3: Ejemplo de serializadores en Django.

```
class FoodSerializer(serializers.ModelSerializer):
        class Meta:
            model = Food
            fields = '__all__'

class OptionSerializer(serializers.ModelSerializer):
    week_option_one = serializers.PrimaryKeyRelatedField(queryset=WeekOption.objects.all())
    week_option_two = serializers.PrimaryKeyRelatedField(queryset=WeekOption.objects.all())
    week_option_three = serializers.PrimaryKeyRelatedField(queryset=WeekOption.objects.all())

    class Meta:
        model = Option
        fields = ['id', 'name', 'week_option_one', 'week_option_two', 'week_option_three']
        read_only_fields = ['trainer']

    def create(self, validated_data):
        user = self.context['request'].user
        if user.is_trainer:
            trainer = user.trainer
        else:
            raise serializers.ValidationError("El usuario no está autorizado para crear opciones.")

        # Crea la instancia de Option incluyendo el trainer obtenido del contexto
        option = Option.objects.create(**validated_data, trainer=trainer)
        return option
```

G.1.4. Autenticación con JWT

Código G.4: Vista de autenticación con JWT en Django.

```
class LoginView(APIView):
        authentication_classes = [] # No authentication required
        permission_classes = [] # No permission required

        def post(self, request, *args, **kwargs):
            username = request.data.get('username')
            password = request.data.get('password')
            user = authenticate(username=username, password=password)
            if user is not None:
                token, created = Token.objects.get_or_create(user=user)
                return Response(
                    {'token': token.key, 'userId': user.id},
                    status=status.HTTP_200_OK)
            else:
                return Response(
                    {'detail': 'Invalid_Credentials'},
                    status=status.HTTP_401_UNAUTHORIZED)
```

G.1.5. Ejemplo de Modelo en Django

Código G.5: Modelo de Entrenamiento en Django.

```
class Option(models.Model):
    trainer = models.ForeignKey('user.Trainer',
     on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    week_option_one = models.ForeignKey(
        WeekOption,
        related_name='week_option_one', on_delete=models.CASCADE)
    week_option_two = models.ForeignKey(
        WeekOption, related_name='week_option_two',
         on_delete=models.CASCADE)
    week_option_three = models.ForeignKey(
        WeekOption, related_name='week_option_three',
         on_delete=models.CASCADE)

    def __str__(self):
        return self.name

class Training(models.Model):
    trainer = models.ForeignKey(
        'user.Trainer', on_delete=models.CASCADE,
         related_name='trainer')
    name = models.CharField(max_length=255)
    exercises = models.ManyToManyField(Exercise, through='TrainingExercise')
    date = models.DateField()
    user = models.ForeignKey(
        User, on_delete=models.CASCADE, related_name='user')

    def __str__(self):
        return self.name
```

G.2. Front-End

G.2.1. Uso de Tokens en el Front-End

Código G.6: Uso de tokens en el Front-end.

```
const [username, setUsername] = useState('');
const [password, setPassword] = useState('');
const [error, setError] = useState('');
const navigate = useNavigate();
const apiUrl = process.env.REACT_APP_API_URL;

const handleSubmit = async (event) => {
    event.preventDefault();
    setError('');

    try {
        const response = await fetch(`
            {apiUrl}/user/frontlogin/`, {
            method: 'POST',
            headers: {
        'Authorization': `Token \{localStorage.getItem('token')}`
            }
        });

        if (!response.ok) {
            const errorData = await response.json();
            throw new Error(
                errorData.detail || 'Falló inicio de sesión');
        }

        const { token, userId } = await response.json();
        localStorage.setItem('authToken', token);
        localStorage.setItem('userId', userId);
        onLoginSuccess(token, userId);
        navigate('/dashboard');
    } catch (error) {
        setError(error.message);
    }
};...}
```

G.2.2. Conexión con el Backend desde la Aplicación Web

Código G.7: Ejemplo de servicio en React para obtener planes de entrenamiento.

```
1  const apiUrl = process.env.REACT_APP_API_URL;
2
3          export const getTrainingPlans = async () => {
4              const response = await fetch(`{apiUrl}/sport/trainings/`,{
5                  headers: {
6                    'Authorization': `Token {
7                      localStorage.getItem('token')}`
8                  }
9              });
10             const data = await response.json();
11             return data;
12         };
```

G.2.3. Inicio Creación de una Dieta

Código G.8: Creación de una dieta en React.

```
function CreateDiet() {
    ...
    useEffect(() => {
      fetch(`{apiUrl}/user/regularusers/`, {
        headers: {
          'Authorization': `Token {localStorage.getItem('authToken')}`
        }
      })
        .then(response => response.json())
        .then(data => {
          setUsers(data);
          if (data.length > 0) {
            setSelectedUser(data[0].id.toString());
          }
        });
    }, [apiUrl]);

    const handleSubmit = (event) => {
      event.preventDefault();
      const dietData = {
        name,
        user: selectedUser,
        start_date: startDate,
        end_date: endDate,
      };
      console.log("Sending diet data", dietData);
      fetch(`{apiUrl}/nutrition/diet/`, {
          method: 'POST',
          headers: {
              'Content-Type': 'application/json',
              'Authorization': `Token {
                  localStorage.getItem('authToken')}`
          },
          body: JSON.stringify(dietData),
      })
      ...
```

G.2.4. Return de una Dieta

Código G.9: Return de una dieta en React.

```
return (
    <div className="container mt-5">
        <h2 className="mb-4">Crear Nueva Dieta</h2>
        <form onSubmit={handleSubmit}>
        <div className="mb-3">
            <label htmlFor="name"
            className="form-label">Nombre de la Dieta:</label>
            <input type="text"
            className="form-control" id="name" value={name}
            onChange={e => setName(e.target.value)} required />
        </div>
        <div className="mb-3">
            <label htmlFor="userSelect"
            className="form-label">Usuario:</label>
            <select className="form-select" id="userSelect"
            value={selectedUser} onChange={
                e => setSelectedUser(e.target.value)}
             required>
            <option value="">Seleccione un usuario</option>
            {users.map(user => (
                <option key={user.id}
                value={user.id}>{user.username}</option>
            ))}
            </select>
        </div>
        <div className="mb-3">
            <label htmlFor="startDate"
            className="form-label">Fecha de Inicio:</label>
            <input type="date" className="form-control" id="startDate"
            value={startDate} onChange={e => setStartDate(e.target.value)}
             required />
        </div>
        ...
        <button type="submit" className="btn btn-primary">
        Crear Dieta</button>
        </form>
    </div>);}
export default CreateDiet;
```

G.3. Aplicación Móvil

G.3.1. Conexión con el Backend desde Aplicación Móvil

Código G.10: Ejemplo de servicio en React Native para obtener planes de entrenamiento.

```
import { API_URL } from '../config';
        export const fetchTrainingPlans = async () => {
            try {
                const response = await fetch(
                    `{API_URL}/training-plans/`);
                if (!response.ok) throw new Error(
                    'Network response was not ok');
                const data = await response.json();
                return data;
            } catch (error) {
                console.error('Error fetching training plans:', error);
                return [];
            }
        };
```

G.3.2. Configuración de la Navegación

Código G.11: Configuración de navegación en React Navigation.

```
const AppNavigator = () => {
    return (
        <Stack.Navigator initialRouteName="LoginScreen">
            <Stack.Screen name="HomeScreen" component={HomeScreen} />
            <Stack.Screen name="LoginScreen" component={LoginScreen} />
            <Stack.Screen name="TrainingDetailsScreen"
            component={TrainingDetailsScreen} />
    </Stack.Navigator>
    );
}
export default AppNavigator;
```

G.3.3. DetailsScreen

Código G.12: Pantalla de detalles en React Native.

```
const DetailsScreen = ({ route }) => {
    const { date, dietEvents, trainingEvents } = route.params;
    return (
        ...
        <Text style={styles.subtitle}>Entrenamientos:</Text>
        {trainingEvents && trainingEvents.length > 0 ? (
            trainingEvents.map((training, index) => (
            <View key={index} style={styles.section}>
                <Text style={styles.sectionTitle}>{training.name}</Text>
                {training.exercises_details.map((detail, idx) => (
                <Text key={idx} style={styles.itemText}>
                    {detail.exercise.name} -
                    Series: {detail.sets}, ...
                    {detail.weight && `, Peso: {detail.weight}kg`}
                    {detail.time && `, Tiempo: {detail.time}min`}
                </Text>))}</View>))
        ) : (
            <Text style={styles.noDataText}>
            No hay entrenamientos para este día.</Text>
        )}
        <Text style={styles.subtitle}>Detalles de la Dieta Diaria:</Text>
        {dietEvents && dietEvents.length > 0 ? (
            ...
                <Text style={styles.sectionTitle}>Nutrientes:</Text>
                ...
                <Text style={styles.sectionTitle}>Comidas:</Text>
                {diet.mealsDetails && diet.mealsDetails.length > 0 ? (
                diet.mealsDetails.map((meal, mealIndex) => (
                    <Text key={mealIndex} style={styles.itemText}>
                    {meal.name}</Text>
                ))
                ) : (
                <Text style={styles.noDataText}>No hay ...</Text>
                )}
            </View>
        ...
```

G.4. Pruebas del Back-End

G.4.1. Configuracion de testsettings.py

Código G.13: Configuración de pruebas en Django.

```
from .settings import *
# Configuración de la base de datos para pruebas
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': ':memory:',
        'ATOMIC_REQUESTS': True,
    }
}
# Otros ajustes específicos para pruebas
SECRET_KEY = 'test-secret-key'
DEBUG = True
# Usar el backend de contraseñas de hashing rápido
PASSWORD_HASHERS = [
    'django.contrib.auth.hashers.MD5PasswordHasher',
]
# Desactivar middleware y aplicaciones en pruebas
MIDDLEWARE = [mw for mw in MIDDLEWARE if mw not in
['debug_toolbar.middleware.DebugToolbarMiddleware']]
INSTALLED_APPS = [app for
app in INSTALLED_APPS if app != 'debug_toolbar']

# Configuración de REST framework para pruebas
REST_FRAMEWORK['DEFAULT_AUTHENTICATION_CLASSES'] = (
    'rest_framework.authentication.SessionAuthentication',
    'rest_framework.authentication.BasicAuthentication',
)
# Configuración de CORS para pruebas (si es necesario)
CORS_ALLOW_ALL_ORIGINS = True
# Configuración de las rutas de archivos estáticos
STATIC_URL = '/static/'
MEDIA_URL = '/media/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static_test')
MEDIA_ROOT = os.path.join(BASE_DIR, 'media_test')
# Configuración de la caché para pruebas
CACHES = {
    'default': {
    'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',}}
```

G.4.2. Configuración de Pruebas de Usuario

Código G.14: Configuración de pruebas en Django.

```
def setUp(self):
    self.client = APIClient(enforce_csrf_checks=False)
    self.user_data = {
        'username': 'testuser',
        'password': 'TestPassword123',
        'email': 'testuser@example.com',
        'first_name': 'Test',
        'last_name': 'User',
    }
    self.trainer_data = {
        'username': 'testtrainer',
        'password': 'TestPassword123',
        'email': 'testtrainer@example.com',
        'first_name': 'Test',
        'last_name': 'Trainer',
        'trainer_type': 'trainer'
    }
```

G.4.3. Pruebas de Usuario

En Código G.15 se presentan algunas de las pruebas más relevantes de usuario en Django.

Código G.15: Pruebas de usuario en Django.

```
def test_create_regular_user(self):
    url = reverse('regularuser_signup')
    response = self.client.post(url, self.user_data, format='json')
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    self.assertEqual(CustomUser.objects.count(), 1)
    self.assertEqual(CustomUser.objects.get().username, 'testuser')

def test_create_trainer(self):
    url = reverse('trainer_signup')
    response = self.client.post(url, self.trainer_data, format='json')
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    self.assertEqual(CustomUser.objects.count(), 1)
    self.assertEqual(CustomUser.objects.get().username, 'testtrainer')

def test_login_regular_user(self):
    self.test_create_regular_user()
    url = reverse('frontlogin')
    response = self.client.post(url, self.user_data, format='json')
    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertIn('token', response.data)

def test_search_trainer(self):
    self.test_create_regular_user()
    self.test_create_trainer()
    url = reverse('frontlogin')
    response = self.client.post(url, self.user_data, format='json')
    token = response.data['token']
    self.client.credentials(HTTP_AUTHORIZATION='Token ' + token)
    search_url = reverse('search_trainer')
    response = self.client.get(search_url, {'trainer_type': 'trainer'})
    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertGreaterEqual(len(response.data), 1)
```

G.4.4. Configuración de Pruebas de Deporte

Código G.16: Configuración de pruebas en Django.

```
def setUp(self):
    self.trainer_data = {
        'username': 'testtrainer',
        'password': 'TestPassword123',
        'email': 'testtrainer@example.com',
        'first_name': 'Test',
        'last_name': 'Trainer',
    }
    self.trainer = Trainer.objects.create_user(**self.trainer_data)
    self.client.force_authenticate(user=self.trainer)

    self.exercise_data = {
        'name': 'Push_Up',
        'description': 'Push_up_exercise',
        'type': 'FUERZA'
    }

    self.training_data = {
        'trainer': self.trainer,
        'name': 'Morning_Workout',
        'date': date.today(),
    }

    self.user_data = {
        'username': 'testuser',
        'password': 'TestPassword123',
        'email': 'testuser@example.com',
        'first_name': 'Test',
        'last_name': 'User',
    }
    self.user = CustomUser.objects.create_user(**self.user_data)
```

G.4.5. Pruebas de Deporte

En Código G.17 se presentan algunas de las pruebas más relevantes de deporte en Django.

Código G.17: Pruebas de deporte en Django.

```
def test_create_exercise(self):
    url = reverse('exercise-list')
    response = self.client.post(url, self.exercise_data, format='json')
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    self.assertEqual(Exercise.objects.count(), 1)
    self.assertEqual(Exercise.objects.get().name, 'Push Up')

def test_create_training_exercise(self):
    exercise = Exercise.objects.create(**self.exercise_data)
    training = Training.objects.create(trainer=self.trainer,
     name='Morning Workout', date=date.today(), user=self.user)
    training_exercise_data = {
        'training': training.id,
        'exercise': exercise.id,
        'repetitions': 10,
        'sets': 3,
        'weight': 50,
    }
    url = reverse('trainingexercise-list')
    response = self.client.post(url, training_exercise_data,
     format='json')
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    self.assertEqual(TrainingExercise.objects.count(), 1)
    self.assertEqual(TrainingExercise.objects.get().repetitions, 10)

def test_get_today_trainings(self):
    training = Training.objects.create(trainer=self.trainer,
     name='Morning Workout', date=date.today(), user=self.user)
    url = reverse('today-trainings')
    response = self.client.get(url, {'user': self.user.id})
    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertEqual(len(response.data), 1)
    self.assertEqual(response.data[0]['name'], 'Morning Workout')
```

G.4.6. Configuración de Pruebas de Nutrición

Código G.18: Configuración de pruebas en Django.

```
def setUp(self):
    self.trainer_data = {
        'username': 'testtrainer',
        'password': 'TestPassword123',
        'email': 'testtrainer@example.com',
        'first_name': 'Test',
        'last_name': 'Trainer',
    }
    self.trainer = Trainer.objects.create_user(**self.trainer_data)
    self.client.force_authenticate(user=self.trainer)

    self.food_data = {
        'name': 'Apple',
        'unit_weight': 100,
        'calories': 52,
        'protein': 0.26,
        'carbohydrates': 14,
        'sugar': 10,
        'fiber': 2.4,
        'fat': 0.17,
        'saturated_fat': 0.03,
    }

    self.ingredient_data = {
        'name': 'Apple_Slice',
        'food': Food.objects.create(**self.food_data).id,
        'quantity': 150
    }

    self.user_data = {
        'username': 'testuser',
        'password': 'TestPassword123',
        'email': 'testuser@example.com',
        'first_name': 'Test',
        'last_name': 'User',
    }
    self.user = CustomUser.objects.create_user(**self.user_data)
```

G.4.7. Pruebas de Nutrición

En el Código G.19 se presentan algunas de las pruebas más relevantes de nutrición en Django.

Código G.19: Pruebas de nutrición en Django.

```
def test_create_food(self):
    url = reverse('food-list')
    response = self.client.post(url, self.food_data, format='json')
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    self.assertEqual(Food.objects.count(), 1)
    self.assertEqual(Food.objects.get().name, 'Apple')

def test_create_dish(self):
    ingredient = Ingredient.objects.create(**self.ingredient_data)
    dish_data = {
        'user': self.user.id,
        'name': 'Apple Salad',
        'ingredients': [ingredient.id]
    }
    url = reverse('dish-list')
    response = self.client.post(url, dish_data, format='json')
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    self.assertEqual(Dish.objects.count(), 1)
    self.assertEqual(Dish.objects.get().name, 'Apple Salad')

def test_get_today_dailydiets(self):
    diet = Diet.objects.create(user=self.user, name='Weight Loss',
    start_date=timezone.now().date(),
    end_date=timezone.now().date() + timedelta(days=7))
    daily_diet = DailyDiet.objects.create(diet=diet,
     date=timezone.now().date())
    daily_diet.meals.add(Meal.objects.create(user=self.user,
     name='Breakfast'))
    url = reverse('today-dailydiets')
    self.client.force_authenticate(user=self.user)
    response = self.client.get(url)
    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertEqual(len(response.data), 1)
    self.assertEqual(response.data[0]['diet'], diet.id)
```

G.5. Pruebas del Front-End

En esta sección se presentan algunas de las pruebas realizadas en el front-end de la aplicación web.

G.5.1. Pruebas de la Aplicación Web

Prueba de HomePage

Código G.20: Prueba de HomePage.js en React.

```
1 import React from 'react';
2 import { render } from '@testing-library/react';
3 import HomePage from './HomePage';
4
5 test('renders HomePage component', () => {
6     const { getByText } = render(<HomePage />);
7     const linkElement = getByText(/Welcome to HomePage/i);
8     expect(linkElement).toBeInTheDocument();
9 });
```

G.5.2. Pruebas de Login

Código G.21: Prueba de Login en React.

```
1  import React from 'react';
2  import { render, fireEvent } from '@testing-library/react';
3  import Login from './Login';
4
5  test('renders Login component', () => {
6      const { getByLabelText, getByText } = render(<Login />);
7      const usernameInput = getByLabelText(/username/i);
8      const passwordInput = getByLabelText(/password/i);
9      const loginButton = getByText(/login/i);
10
11     expect(usernameInput).toBeInTheDocument();
12     expect(passwordInput).toBeInTheDocument();
13     expect(loginButton).toBeInTheDocument();
14 });
15
16 test('allows the user to log in', () => {
17     const { getByLabelText, getByText } = render(<Login />);
18     const usernameInput = getByLabelText(/username/i);
19     const passwordInput = getByLabelText(/password/i);
20     const loginButton = getByText(/login/i);
21
22     fireEvent.change(usernameInput, { target: { value: 'testuser' } });
23     fireEvent.change(passwordInput, { target: { value: 'password' } });
24     fireEvent.click(loginButton);
25         });
```

G.5.3. Pruebas de Profile

Código G.22: Prueba de Profile.js en React.

```
1  import React from 'react';
2  import { render } from '@testing-library/react';
3  import Profile from './Profile';
4
5  test('renders Profile component', () => {
6      const { getByText } = render(<Profile />);
7      const headingElement = getByText(/Profile/i);
8      expect(headingElement).toBeInTheDocument();
9  });
```

G.5.4. Pruebas de la Aplicación Móvil

En esta sección se presentan algunas de las pruebas realizadas en la aplicación móvil.

Prueba de LoginScreen

Código G.23: Prueba de LoginScreen en React Native.

```
import React from 'react';
import { render, screen, fireEvent } from '@testing-library/react';
import LoginScreen from './LoginScreen';
import AuthContext from '../AuthContext';

const mockLogin = jest.fn();

describe('LoginScreen', () => {
    test('renders LoginScreen and performs login', () => {
    render(
        <AuthContext.Provider value={{ login: mockLogin }}>
        <LoginScreen />
        </AuthContext.Provider>
    );

    fireEvent.change(screen.getByPlaceholderText('Username'), {
        target: { value: 'testuser' },
    });
    fireEvent.change(screen.getByPlaceholderText('Password'), {
        target: { value: 'password123' },
    });
    fireEvent.click(screen.getByText('Login'));

    expect(mockLogin).toHaveBeenCalledWith('testuser', 'password123');
    });
});
```

Prueba de HomeScreen

Código G.24: Prueba de HomeScreen en React Native.

```
1  import React from 'react';
2  import { render, screen } from '@testing-library/react';
3  import HomeScreen from './HomeScreen';
4
5  describe('HomeScreen', () => {
6      test('renders HomeScreen with welcome message', () => {
7      render(<HomeScreen />);
8      expect(screen.getByText('Welcome to the Home Screen')).toBeInTheDocument();
9      });
10 });
```


UÁM
Universidad Autónoma
de Madrid