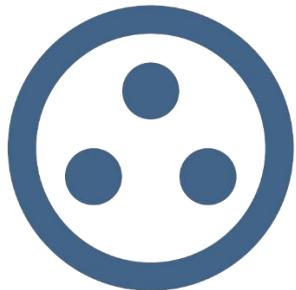


FITNESS TRACKER

MEMORIA



Realizado por Aaron Esono, Guillermo Quintanar y Hugo Pelayo
26 de abril de 2024

Resumen

Fitness Tracker es una aplicación diseñada para promover un estilo de vida saludable a través de la gestión integral de la dieta y el ejercicio físico. Destinada a usuarios de todas las edades y niveles de condición física, la plataforma ofrece herramientas intuitivas para monitorear y mejorar la salud de manera personalizada. En términos de dieta, la aplicación permite a los usuarios registrar su consumo diario de alimentos, ofreciendo recomendaciones nutricionales basadas en metas individuales de calorías y macronutrientes. Esta funcionalidad facilita la adopción de hábitos alimenticios más saludables y mantiene un equilibrio adecuado entre la ingesta y el gasto calórico. En cuanto al ejercicio, el Fitness Tracker permite crear rutinas personalizadas adaptadas a las necesidades y objetivos específicos de cada usuario. Los datos capturados incluyen duración del sueño, calorías quemadas, pasos realizados, frecuencia cardíaca y niveles de oxígeno en sangre, proporcionando una visión completa del rendimiento físico y la salud cardiovascular. La aplicación se distingue por su accesibilidad y facilidad de uso en diferentes dispositivos móviles y sistemas operativos, garantizando una experiencia fluida para todos los usuarios. Con un enfoque en la privacidad y la seguridad de los datos, Fitness Tracker asegura la confianza de los usuarios al proporcionar una plataforma segura para gestionar su bienestar físico. Este enfoque integral convierte al Fitness Tracker en un aliado efectivo para mejorar la salud y la forma física, motivando a los usuarios a adoptar decisiones más saludables en su día a día.

Abstract

Fitness Tracker is an app designed to promote a healthy lifestyle through managing diet and physical exercise comprehensively. Aimed at users of all ages and fitness levels, the platform offers user-friendly tools to monitor and enhance health in a personalized way. Regarding diet, the app allows users to log their daily food intake, providing nutritional recommendations based on individual calorie and macronutrient goals. This functionality helps adopt healthier eating habits and maintain a balanced calorie intake. For exercise, Fitness Tracker enables users to create customized routines tailored to their specific needs and goals. Captured data includes sleep duration, calories burned, steps taken, heart rate, and blood oxygen levels, offering a complete view of physical performance and cardiovascular health. Fitness Tracker also allows users to generate diets according to their needs with the help of a language model. The app stands out for its accessibility and ease of use across various mobile devices and operating systems, ensuring a smooth experience for all users. With a focus on data privacy and security, Fitness Tracker ensures user trust by providing a secure platform to manage their physical well-being.

Índice

| | |
|---|-----|
| Introducción | 6 |
| Análisis del entorno | 7 |
| Análisis del sistema actual | 9 |
| Solución propuesta | 10 |
| Planificación temporal del desarrollo del proyecto | 11 |
| Detalles de la solución adoptada | 12 |
| Funcionalidad del cliente Android | 12 |
| Funcionalidad del cliente web..... | 13 |
| Requisitos técnicos | 14 |
| Seguridad | 15 |
| Comunicación IA | 17 |
| Arquitectura sistema..... | 18 |
| Plataformas y dispositivos compatibles..... | 19 |
| Aspectos de seguridad y privacidad..... | 19 |
| Interfaz de usuario y experiencia de usuario | 21 |
| Estudio de la viabilidad del proyecto | 31 |
| Documentación del diseño e implementación de la solución adoptada. | 32 |
| Arquitectura backend | 32 |
| Código fuente comentado | 47 |
| Estructura de carpetas | 49 |
| Manual de configuración y funcionamiento de la aplicación. | 83 |
| Aplicación Android | 83 |
| Aplicación web..... | 85 |
| Manual de usuario | 87 |
| Plan de formación a los Usuarios de la Aplicación | 102 |
| Bibliografía y fuentes de información | 102 |

Agradecimientos a todos los profesores del instituto de IES Villablanca que han ofrecido soporte para poder resolver las dudas y cuestiones que surgieron durante la realización de este proyecto.

Introducción

Fitness Tracker es una aplicación destinada a la gestión de la actividad física y el cuidado de la salud de nuestros clientes. Como tal, ofrece un entorno web en el cual los usuarios pueden seguir estudiando el progreso del ejercicio físico que realizan a lo largo del día. Paralelamente al entorno web está la plataforma de Android, donde los usuarios tienen mejor acceso a sus datos recolectados de un reloj inteligente con sistema Wear OS, a través de un entorno fácil de usar e intuitivo.

El uso del smartwatch ha experimentado una evolución significativa en los últimos años, pasando de ser un dispositivo principalmente utilizado por atletas de alto rendimiento a convertirse en una herramienta accesible y necesaria para la actividad física en general. Además de servir como una extensión del teléfono celular, los smartwatches ofrecen diversas ventajas para quienes practican ejercicio regularmente.

Una de las principales funciones de los smartwatches es la medición del ritmo cardíaco durante el ejercicio. Mediante sensores ópticos integrados, es posible monitorizar la frecuencia cardíaca de manera precisa, lo que permite ajustar la intensidad del entrenamiento y mejorar la salud cardiovascular.

Además, algunos smartwatches ofrecen funciones de control de peso, solicitando datos como estatura y peso para personalizar planes de entrenamiento que apoyen en la pérdida de peso y mejora del estado físico. Estos dispositivos también pueden actuar como entrenadores personales, proporcionando planes de entrenamiento gratuitos adaptados a las necesidades individuales de cada usuario, como sería el caso de nuestra aplicación, Fitness Tracker.

El monitoreo del sueño es otra característica importante de algunos smartwatches, que registran la calidad del descanso y proporcionan información sobre las diferentes fases del sueño, como sueño profundo, REM (del inglés Rapid Eye Movement o sueño de movimientos oculares rápido en castellano), sueño ligero e insomnio. Comprender y mejorar los hábitos de sueño es fundamental para el bienestar físico y mental.

Además, los smartwatches ofrecen análisis de datos detallados en tiempo real, mostrando información relevante como frecuencia cardíaca, pasos dados, calorías quemadas y

actividad física realizada. Esta capacidad permite a los usuarios realizar un seguimiento exhaustivo de su estilo de vida y tomar decisiones informadas para mejorar su salud y bienestar.

Análisis del entorno

Se prevé un mayor consumo en el futuro cercano de este producto, por ello creemos que es relevante invertir en su desarrollo.

Ventas de relojes inteligentes a nivel mundial de 2016 a 2025 (en millones de unidades)

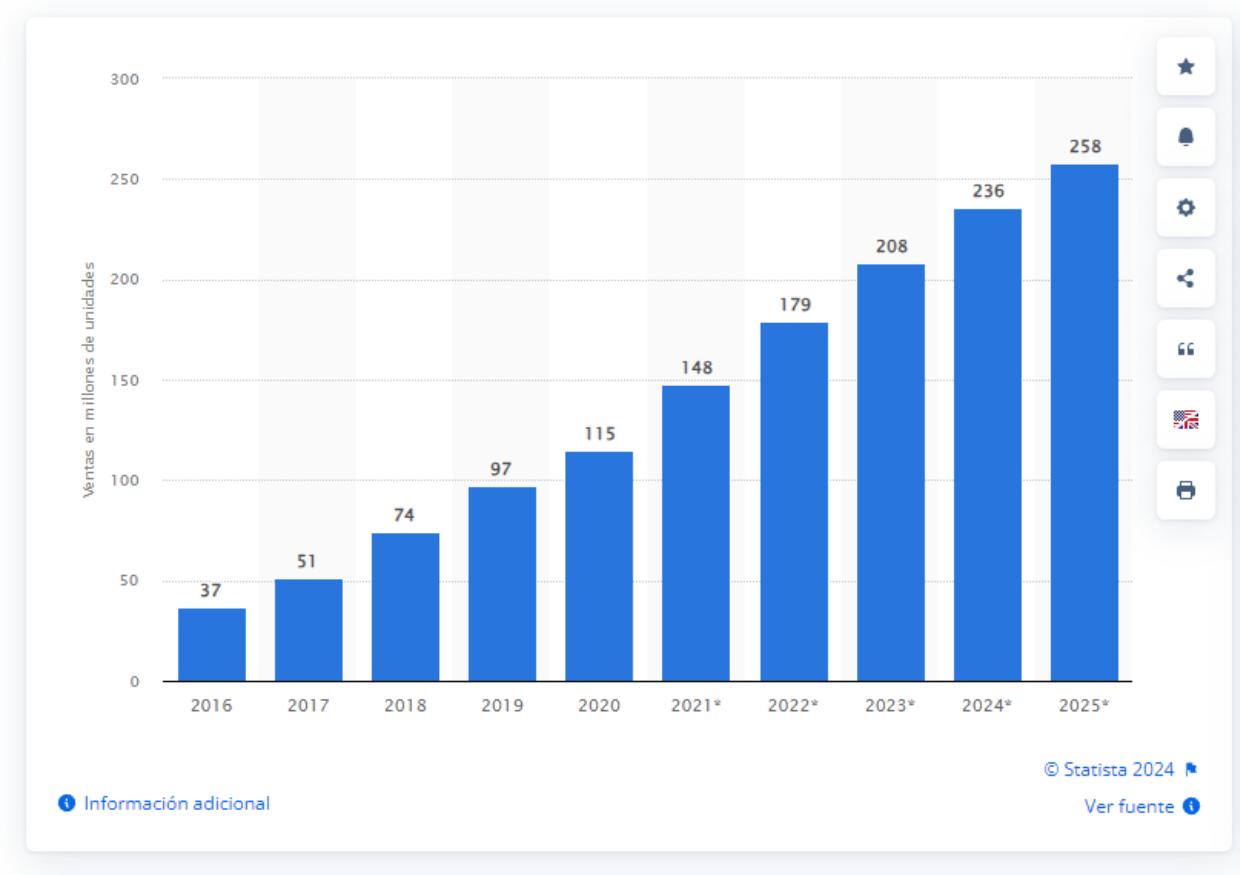


Figura 1. Ventas de relojes inteligentes

A pesar de los avances en la experiencia de usuario, persisten desafíos pendientes, como la gestión de la mensajería, donde la transcripción de voz a texto sigue siendo poco eficaz.

En el ámbito del software, hay una gran diversidad de sistemas operativos para smartwatches, incluyendo WatchOS de Apple, Tizen de Samsung, Harmony OS de Huawei, entre otros. Esta heterogeneidad refleja la naturaleza incipiente y en evolución continua del mercado de los relojes inteligentes. Este es un aspecto importante a tener en cuenta ya que estos dispositivos son la fuente principal de datos de nuestra aplicación y dicha heterogeneidad dificulta más el desarrollo de aplicaciones que estén destinadas a trabajar con estos dispositivos.

Fitness Tracker es una solución que combina una plataforma web con una aplicación móvil para dispositivos Android. Presentamos una aplicación innovadora que redefine la forma en que los usuarios gestionan su actividad física, rendimiento y salud, al integrar una plataforma web con una aplicación móvil para dispositivos Android.

Se ha decidido desarrollar únicamente para Android en lo que respecta a dispositivos móviles, ya que para desarrollar una versión para iOS implica la necesidad de mayor mano de obra (aparte de aprender a desarrollar en un entorno nuevo) paralelamente con varios requisitos de desarrollo en la plataforma de iOS que no podemos asumir que son: la necesidad de un ordenador Mac para desarrollar aplicaciones para esta plataforma; en caso de querer distribuir la aplicación sería necesario pagar una licencia anual de 99 dólares.

Esta solución ofrece un conjunto completo de funciones básicas diseñadas para mejorar el bienestar general del usuario. Desde el seguimiento de la actividad física hasta el monitoreo del estado de salud, nuestro objetivo es proporcionar una experiencia integral que empodere a los usuarios para alcanzar sus objetivos de salud y rendimiento de manera efectiva y conveniente.

Análisis del sistema actual

Hoy en día, si quieres saber cuántas calorías o saber qué rutina tienes que seguir en x tiempo para alcanzar cierta meta, es muy difícil hacerlo por tu cuenta, ya que tienes que tener registrado qué comes en todo momento, cuáles son los nutrientes que contiene lo que consumes. Aparte, también tienes que monitorizar todo el ejercicio diario y lo que consume tu cuerpo simplemente por existir.

Por otro lado, también tienes que conocer tu cuerpo en cuanto a la altura, peso y otros detalles para saber a profundidad qué es lo que necesitas.

Hacer todo eso es muy complejo, ya que tienes que saber sobre nutrición y bienestar, además de perder mucho tiempo realizando todos estos cálculos diariamente.

Por ello, con esta aplicación, lo que conseguiría el usuario sería ahorrarse todos estos datos, los cuales no tendrá que hacer manualmente, y simplemente tendrá que registrar los alimentos consumidos, el ejercicio extra realizado y la cantidad de agua consumida al día. Además, podrá consultar con la inteligencia artificial rutinas y dietas que se adaptarán todo lo posible al usuario.

Con esta aplicación, el usuario podrá modificar sus datos de forma sencilla, le aparecerá la información necesaria, y podrá ver todo su progreso en la App desde que la instaló.

Solución propuesta

Para afrontar el proyecto a desarrollar, necesitaremos distintas tecnologías para desarrollarlo, tanto para la parte de desarrollo web, Android, y la parte del servidor.

Para la parte del Servidor donde guardaremos los datos de los usuarios utilizaremos SpringBoot con Java, ya que nos da las suficientes ventajas para facilitar esos microservicios correctamente. Para la parte de los alimentos y la inteligencia artificial utilizaremos .net core para facilitar esas uniones con el cliente y la IA, aparte que es muy fácil de implementar la seguridad de la misma.

Para la base de datos utilizaremos Mongo, ya que no sabemos cuántos datos puede llegar a meter el usuario, aparte que habrá más escrituras que lecturas

Para la parte Android utilizaremos Jetpack compose y Kotlin, ya que Jetpack compose nos permite crear aplicaciones Android más compactas y que tiene más funcionalidades como por ejemplo el manejo de los estados en la App.

Para la parte Web, utilizaremos React porque es sencillo de manejar y para aplicaciones pequeñas viene mejor.

El proyecto a realizar tendrá un pequeño coste debido a que hay que pagar una pequeña cuota para utilizar la IA y comunicar a través del backend, y para el tema de los alimentos, ya que no se puede hacer llamadas ilimitadas. Para intentar recuperar esa inversión, la opción de poder utilizar la IA será de pago para amortizar esa pérdida entre la IA y los alimentos.

Planificación temporal del desarrollo del proyecto

El desarrollo del proyecto se estructura en varias fases bien definidas, cada una con tiempos estimados para asegurar un avance ordenado y eficiente. La primera fase es el Análisis de Requisitos del Producto, que abarca tanto nuestras aplicaciones web y Android como el backend necesario para soportarlas. En esta etapa inicial, se recoge toda la información relevante de los usuarios, se identifican las funcionalidades esenciales y se establecen los objetivos específicos del proyecto. Este proceso detallado nos permite tener una visión clara de las necesidades y expectativas de los usuarios, y suele estimamos que nos tomaría aproximadamente dos semanas.

Una vez completado el análisis de requisitos, se avanza hacia la fase de Diseño del Modelo de Datos. Aquí, se define la estructura de la base de datos que soportará todas las funcionalidades del sistema. Este modelo debe ser robusto y flexible para manejar eficientemente los datos de los usuarios y las operaciones del sistema. Esta fase incluye la creación de diagramas entidad-relación y la especificación de las tablas y relaciones, y generalmente se completa en un período de dos semanas.

Con el modelo de datos definido, se procede a la fase de Desarrollo del Primer Prototipo. Este prototipo inicial es crucial para visualizar cómo interactuarán los usuarios con el sistema. Se desarrolla tanto la interfaz de la aplicación web como la de Android, junto con las primeras versiones de los servicios backend. El objetivo es tener una versión funcional que permita realizar pruebas iniciales y recibir feedback. Esta fase es intensiva y suele extenderse durante cuatro semanas, permitiendo iterar sobre las primeras versiones de la interfaz y los servicios backend.

Finalmente, se llega a la fase de Despliegue y Puesta en Marcha de los Servicios en el Servidor. En esta etapa, todos los componentes del sistema se integran y se despliegan en los servidores correspondientes. Utilizamos AWS para alojar el backend, aprovechando su fiabilidad y escalabilidad, y Vercel para desplegar la aplicación web, facilitando un flujo de trabajo ágil y efectivo. Esta fase también incluye la configuración de dominios, la seguridad y la optimización del rendimiento. Se estima que esta última fase tome alrededor de dos semanas, asegurando que todo esté correctamente configurado y funcionando antes del lanzamiento oficial.

Detalles de la solución adoptada

Funcionalidad del cliente Android

Registro de usuario

Permite a los usuarios crear una cuenta en la aplicación proporcionando la información necesaria.

Un usuario para poder registrarse necesitará ingresar por un formulario el nombre de usuario, este es un nombre opcional con que el usuario se identifica dentro de la aplicación, se utiliza en casos de generar informes para él, este campo no es obligatorio de llenar en el registro, en cuyo caso se identificaría con el nombre completo en la aplicación (incluyendo apellidos), más tarde puede editarse si se desea; el usuario introducirá su nombre, primer y segundo apellido; la contraseña, el correo electrónico y la fecha de nacimiento con que se puede determinar la edad más tarde si es necesario.

Perfil de usuario

Permite a los usuarios agregar y editar su información personal, donde se incluye la edad, el peso, la altura, la contraseña, el nombre de usuario, el nombre, los apellidos y el sexo, en esencia los mismos datos que en el momento de registro, sin embargo, la dirección de correo no se puede modificar y será siempre la misma.

Notificaciones y recordatorios

Envía notificaciones y recordatorios a los usuarios para mantenerlos informados sobre sus metas, progreso y otras actividades relevantes.

La aplicación de Android será capaz de generar una notificación para el usuario poder revisar las comidas que tocan en un día en concreto, por otra parte.

Establecimiento de objetivos

Permite a los usuarios establecer objetivos personalizados relacionados con la actividad física, el rendimiento y la salud, como la cantidad de ejercicio semanal, la cantidad de pasos diarios, etc.

Monitoreo en tiempo real

Proporciona a los usuarios la capacidad de monitorear su actividad física y rendimiento en tiempo real a través de la aplicación; entre estos datos se incluye la medición del ritmo cardíaco, el número de pasos realizados en un determinado tiempo, el tiempo de sueño, el nivel de oxígeno en sangre y el número estimado de calorías quemadas con el ejercicio físico realizado.

Control de objetivos establecidos

Permite a los usuarios realizar un seguimiento del progreso hacia sus objetivos establecidos, proporcionando estadísticas y métricas relevantes para evaluar su rendimiento.

Aquí se incluye un listado de elementos por cumplir en formato ToDo List, esto es, el usuario podrá establecer objetivos de quema de calorías que automáticamente se marcarán como hechas de acuerdo al progreso que lleve el cliente.

Funcionalidad del cliente web

Este cliente al igual que el cliente Android, ofrece la funcionalidad de registro a inicio de sesión y la sección de perfil de usuario.

Control de objetivos establecidos

Permite a los usuarios realizar un seguimiento de sus objetivos establecidos, tanto en la aplicación móvil como en la plataforma web, para garantizar una experiencia coherente y sin problemas en ambos dispositivos.

Historial de actividades

Proporciona a los usuarios un registro detallado de todas sus actividades pasadas, incluidos entrenamientos, mediciones de salud, logros alcanzados, etc., para ayudar en la evaluación del progreso a lo largo del tiempo.

Requisitos técnicos

Como se ha mencionado anteriormente, se va a elaborar una para la plataforma Android y una aplicación web disponible para escritorio.

Para el desarrollo de la interfaz en Android se utilizará la librería de Jetpack Compose mediante el lenguaje de programación Kotlin, ambos muy comunes para el desarrollo de interfaces en Android. Para mejorar la compatibilidad entre dispositivos Android y maximizar el número de dispositivos móviles en que se puede utilizar nuestra aplicación para Android, se va a desarrollar para la versión 8.0 de este sistema operativo con la API 26.

Para establecer conexión entre esta aplicación y la API REST se utilizará la librería de Retrofit, disponible también para el lenguaje de programación Kotlin.

La aplicación web se va a desarrollar sobre el framework de React usando JavaScript como lenguaje de programación. Se prevé la utilización de la librería de CSS Bootstrap para agilizar el desarrollo del diseño de las interfaces.

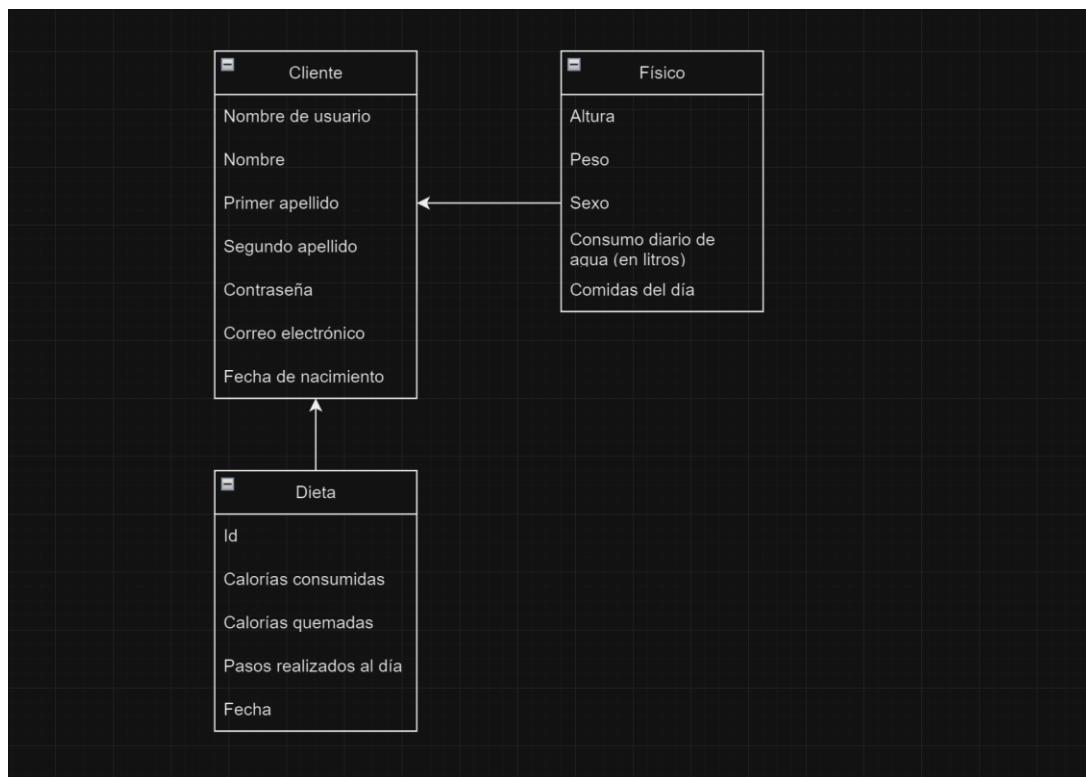
Almacenamiento de datos

Del cliente se van a recoger datos identificativos, a citar: el nombre completo junto con los apellidos; una dirección de correo electrónico la cual será única por cuenta en nuestra base de datos, de modo que no admitimos varias cuentas con la misma dirección de correo electrónico; una contraseña para la cuenta del usuario y finalmente un nickname identificativo de la cuenta del usuario que es opcional a la hora de registrarse.

Un usuario registrado puede almacenar los siguientes datos actualmente: la altura, el peso, el sexo y la fecha de nacimiento, con que se deduce la edad. Estos datos se almacenan para mantener constancia de su metabolismo basal, el Índice de Masa

Muscular o IMC, estimar consumo recomendado de agua al día, requerimiento calórico diario para así poder estimar también el número de proteínas, grasas o carbohidratos que se han de consumir.

Por otro lado, también se pretende almacenar datos de dietas como: el número de comidas al día (desayuno, media mañana, almuerzo, merienda, cena); cantidad de agua consumida al día en un día concreto (valor estimado); ejercicio físico hecho a lo largo del día. A continuación, se muestra un esquema provisional de los datos que se van a guardar en la base de datos.



Seguridad

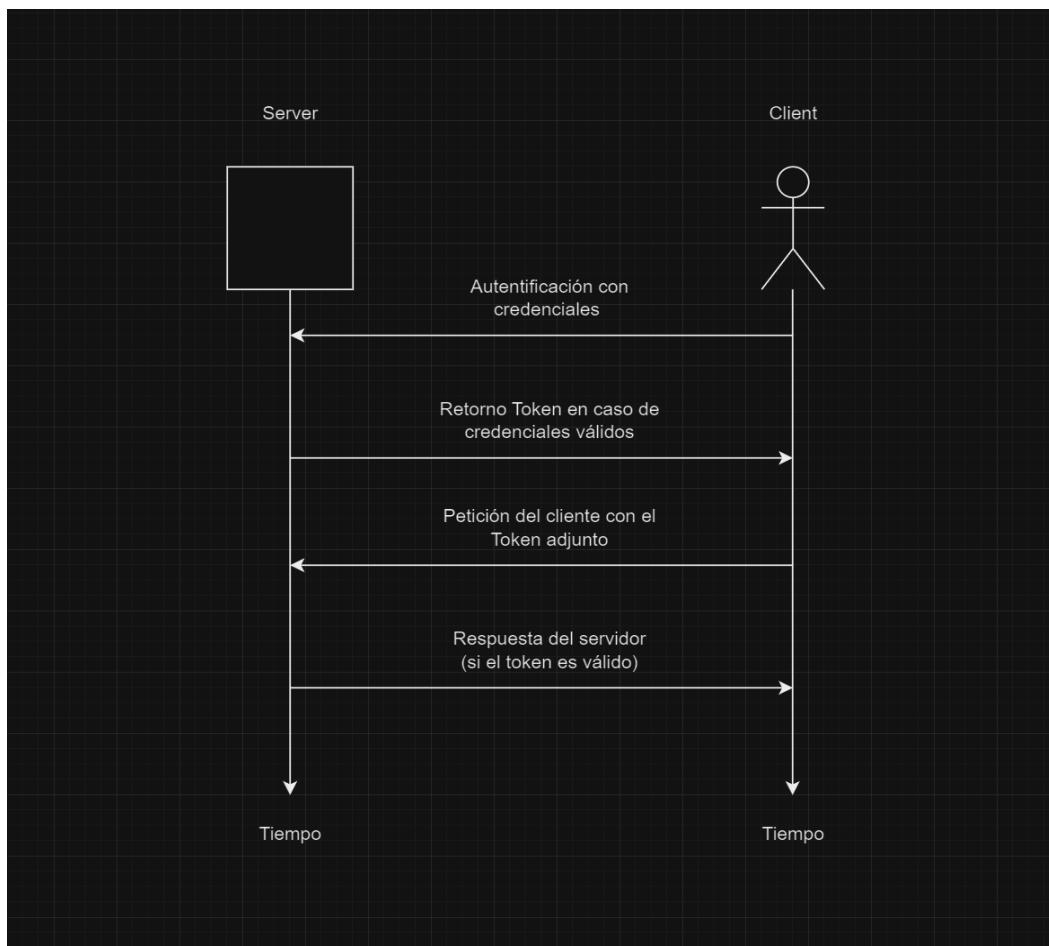
La seguridad es un aspecto importante a tener en cuenta. Este aspecto se va a trabajar tanto en el servidor como en ambos clientes, el de Android y la aplicación web.

Para garantizar la seguridad en la aplicación web, será fundamental restringir el acceso a ciertas rutas únicamente a usuarios autenticados. Esto se va a lograr mediante el uso de

componentes de enrutamiento, mediante **react-router**, que permite definir rutas privadas y públicas. Las rutas privadas estarán protegidas y solo serán accesibles para usuarios que han iniciado sesión correctamente, mientras que las rutas públicas están disponibles para todos los usuarios. Al implementar este enfoque, se puede controlar de manera efectiva qué contenido y funcionalidades son accesibles para cada tipo de usuario.

En la aplicación Android el enfoque será similar, será necesario iniciar sesión para poder acceder a las funcionalidades de la aplicación.

La comunicación con el servidor se realizará mediante validación por token. El usuario ingresa sus credenciales en alguna de las aplicaciones clientes para poder iniciar una nueva sesión, validados las credenciales, el usuario recibirá un token con que podrá comunicarse con el servidor de manera segura. A continuación, un esquema representativo de esta arquitectura:



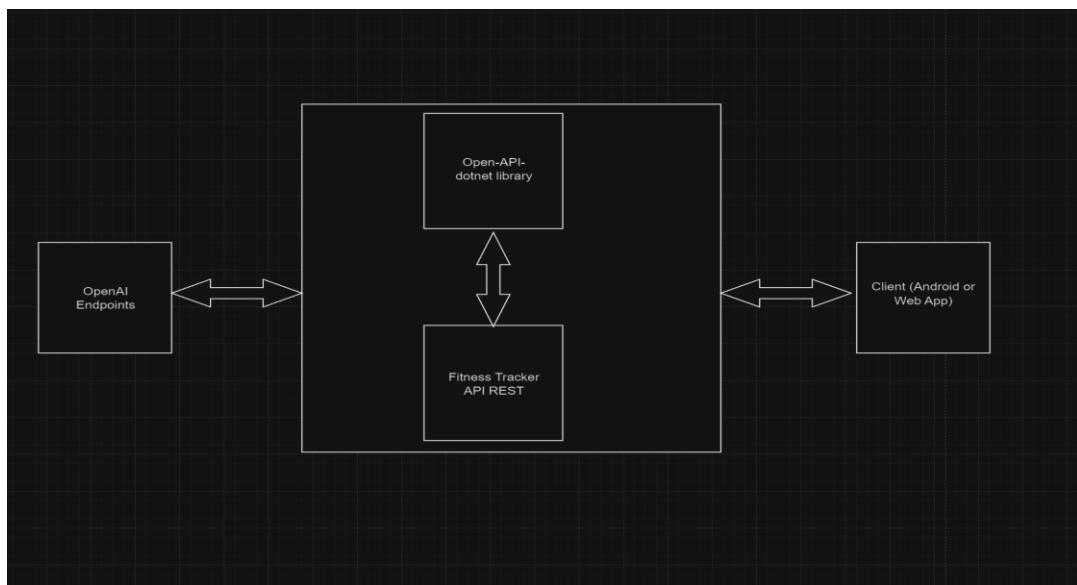
Comunicación IA

Se va a establecer un servicio que se comunica con el modelo de lenguaje GPT 4 Vision mediante peticiones HTTP. Para ello se va a desarrollar una API REST en C# utilizando ASP.NET Core junto con la librería OpenAI-API-dotnet, una librería que facilita la comunicación con los servicios de OpenAI, esta librería nos abstrae de la tarea de realizar las peticiones a los endpoints de OpenAI y nos devuelve los resultados a través de objetos con que podemos trabajar con facilidad en nuestra aplicación.

Importante destacar que este servicio está disponible para usuarios autenticados, como tal, la API REST que se comunica con OpenAI-API-dotnet también deberá validar las credenciales del usuario antes de poder realizar cualquier petición a la inteligencia artificial, para ello usamos una clave que nos ofrece OpenAI.

Para utilizar el servicio de inteligencia artificial, los usuarios tendrán que confirmar sus credenciales, esto es así ya que se pretende cambiar este comportamiento en futuro para restringir su uso a usuarios a través de un plan de pago.

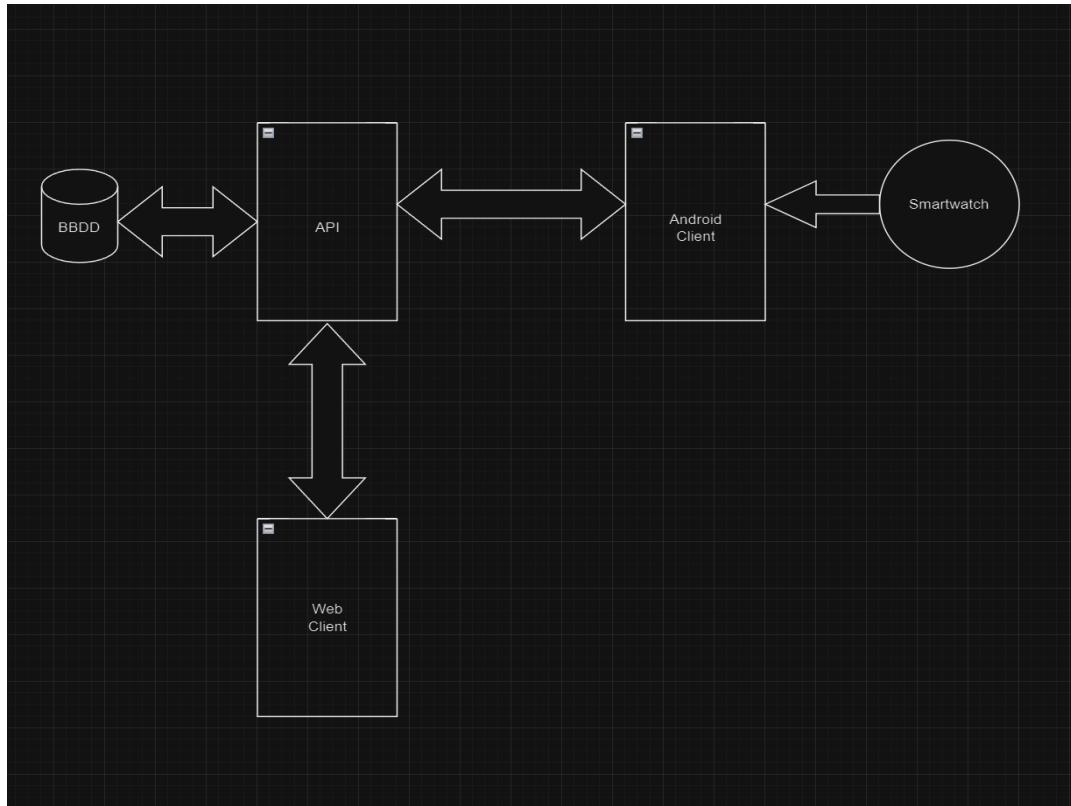
Para comunicarse con las APIs de Fitness Tracker se utilizará objetos JSON o parámetros en la URI, como tal se tiene que validar estos datos. Para validar los JSON de entrada se va a usar la librería de FluentValidation de .NET para asegurar que no se llaman a estos endpoints con datos inadecuados. A continuación, un esquema del sistema (las flechas representan el flujo de datos):



Arquitectura sistema

Nuestro sistema consta de tres elementos principalmente: un backend que consta de una API REST sobre una base de datos mongo y una API REST con ASP.NET que sirve de enlace con los endpoints de la IA con que se va a realizar los prompts, a parte de también realizar la comunicación entre nuestro backend y otros servicios externos que sean necesarios como el de información nutricional Edamam. Nuestros clientes se conectarán a la API de ASP.NET que consta de un API Gateway que redirige las peticiones a los servicios que tenemos en nuestro sistema ejecutándose, de esta forma se simplifica donde implementar la seguridad y tenemos más control sobre qué puertos se deben exponer al exterior y cuáles no; dos clientes que interaccionan con este backend y el último componente sería nuestro reloj inteligente que sirve como fuente de datos para el cliente Android.

Por la naturaleza de los datos que se van a guardar en la base de datos, esta va a ser documental. Para integrar esta base de datos se va utilizar MongoDB. A continuación, se muestra un esquema del sistema:



Plataformas y dispositivos compatibles

La aplicación estará disponible para dispositivos móviles Android, siendo esta desarrollada en Kotlin con Jetpack Compose.

También, esta aplicación estará disponible para la parte web, estado desarrollada en React.

Aparte, esta aplicación estará disponible para smartwatch. En un principio, se desarrollará para los relojes que tengan integrado Wear OS, y llevándolo a todos los demás distintos tipos de relojes en un futuro.

Para verificar que las funciones se complementan correctamente, usaremos Postman para hacer los test y comprobar que las llamadas devuelven los datos correctos, así como Swagger para ver qué parámetros se les pasa a las llamadas y qué devuelven.

Para el cliente, usaremos el emulador y varios dispositivos móviles para comprobar que todo se ve de la forma esperada.

Aspectos de seguridad y privacidad

Garantizar la seguridad de los datos del usuario mediante técnicas de cifrado y autenticación segura.

Ya que nuestra aplicación está pensada para un entorno abierto es necesario cifrar los datos de manera que las claves de cifrado no puedan ser interceptadas por un tercero durante el envío de la misma. Por ello usaremos cifrado de tipo asimétrico.

Utilizaremos dos claves diferentes que están vinculadas entre sí matemáticamente. La app mantendrá en secreto la clave privada, mientras que la pública se compartirá entre todos los clientes de nuestro servicio. Con esto conseguiremos que los datos se muevan de manera cifrada a través de la red y solo puedan ser consultados por el cliente.

En cuanto al autenticador que usaremos tanto para nuestra aplicación web como para la aplicación móvil, usaremos la autentificación por JWT. Para realizar peticiones al backend se debe iniciar sesión en un usuario válido primero, este recibirá un token con que puede hacer peticiones a los endpoints privados.

Cumplimiento de regulaciones de privacidad, como GDPR, para proteger la privacidad de los usuarios y su información personal.

Nuestro principal objetivo es la seguridad y privacidad de nuestros usuarios por lo tanto seguiremos el protocolo de regulación de privacidad GDPR. Este garantiza la protección y privacidad de datos de los usuarios. Los principios básicos de GDPR son:

- **Transparencia:** Debemos tratar los datos personales dentro de la ley, de manera justa y transparente. Esto significa que precisamos siempre de notificar que estamos recolectando información y debemos, aun, especificar cuál información y cómo será utilizada.
- **Propósito delimitado:** Debemos recolectar apenas datos específicos con intenciones específicas. Jamás podemos procesar información más allá de las que contemplan las intenciones específicas e informadas al usuario.
- **Minimización de datos:** Apenas podemos recolectar datos personales adecuados y relevantes para nuestras intenciones. Esto significa que recolectar o cuestionar cualquier información no relacionada al servicio ofrecido está prohibido.
- **Precisión:** Todo dato personal que recolectamos debe ser correcto, claro y, cuando sea necesario, ser actualizado para estar en día con la información personal del usuario.
- **Eliminación de Datos:** Los datos personales de los usuarios solo deben ser mantenidos mientras son necesarios y sean útiles para el propósito original.
- **Seguridad:** Nuestra organización debe usar técnicas apropiadas y medidas de seguridad para proteger datos personales contra el procesamiento no autorizado o vaciamiento. Acceso, pérdida o alteración indebida de los datos implica en penalidades. Dependiendo del caso es recomendado, cuando no obligatorio, el uso de datos segregados, criptografiados, seudonimizados o anonimizados.

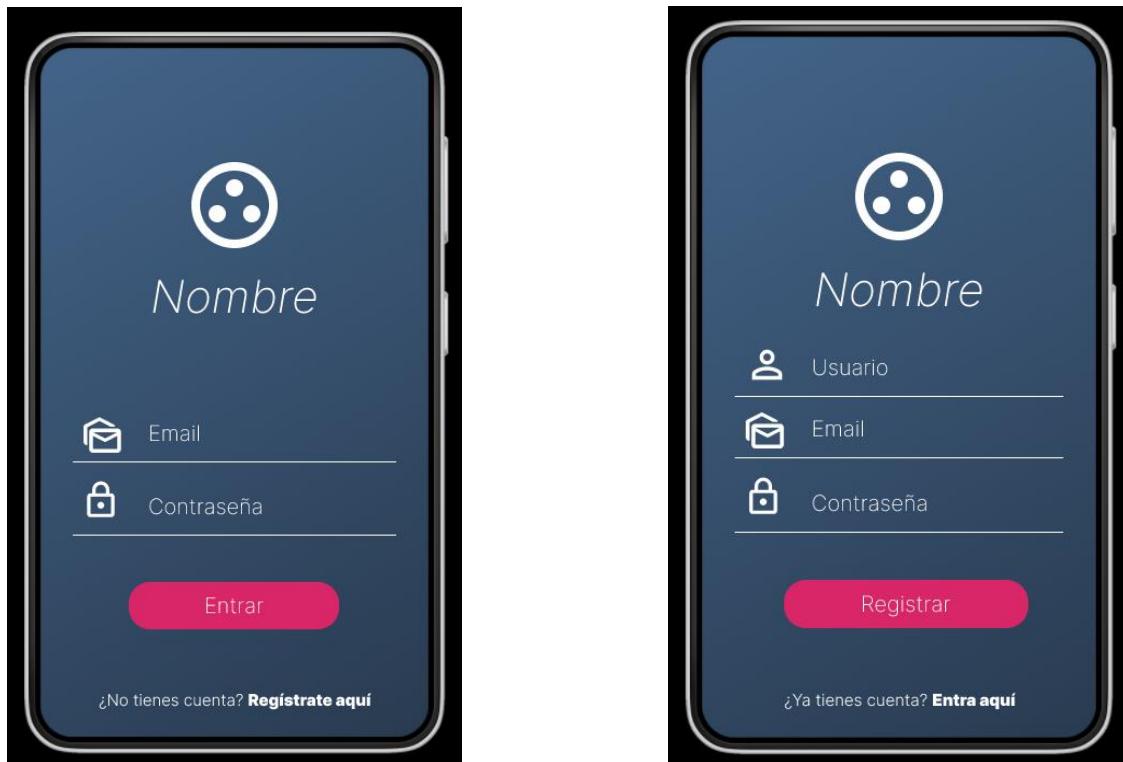
Seguiremos todos y cada uno de estos puntos para preservar la seguridad y privacidad de cada uno de nuestros usuarios.

Interfaz de usuario y experiencia de usuario

Para la interfaz de usuario de la aplicación móvil, se hará de tal forma para que la navegación y la introducción de datos de la misma sea lo más intuitiva y sencilla posible.

Para empezar, en la parte del login, se hará una interfaz sencilla que se centre tanto en los campos de email, contraseña y el botón, y que cuando presione sobre uno de estos quede marcado para que se dé cuenta de qué está situado en uno de los campos, todo en una misma ventana y sin que tenga que scrollear en esta.

Se adjunta una foto de un prototipo de cómo quedaría el login y el register.



En la parte del menú principal, en la parte de arriba, el usuario tendrá un menú donde podrá navegar entre las distintas pantallas de forma sencilla, a parte de un menú desplegable en la parte lateral para que el usuario tenga opciones para poder personalizar algunas partes de la app, como los colores, etc.

En la primera pantalla se verían todos los datos intrínsecos del usuario, permitiéndole cambiar estos mismos cuando él quiera. También, se mostrarán los datos básicos sacados de los hablados anteriormente, como el IME, gasto calórico diario, etc. Estos cambiarán cuando el usuario cambie sus propios datos.

La interfaz quedaría de la siguiente manera:



En la segunda pantalla estará todo lo relacionado con las consumiciones diarias, cuántas calorías, grasas, carbohidratos y proteínas recomendadas tiene que consumir el usuario. Además, debajo de estos vendrán las comidas y las dietas sugeridas para el usuario por la inteligencia artificial. Toda esta información vendrá ordenada en una columna en la que el usuario podrá scrollar para poder tener acceso a la información.

Adjunto una foto del prototipo de como quedaría.



En la última ventana, se mostrarían todos los alimentos consumidos por el usuario en el día, pudiendo ver los registros de otro día si así lo quisiera el usuario. En esta pantalla, se repartiría el desayuno, comida y cena, pudiendo el usuario cambiar los nombres o meter otra comida entre medias, y midiendo las calorías que lleva ese día junto con las que debe suplir. Esto se hace para motivar al usuario que intente hacer un poco más de ejercicio o para que consuma la cantidad de agua necesaria al día, marcándolo adecuadamente si ha conseguido la meta diaria.

Adjunto una foto del prototipo de la pantalla.



El [siguiente enlace](#) redirige a una vista previa desde el navegador de la interfaz:

La interfaz web estará creada de manera que tenga un inicio llamativo para atraer a nuevos clientes, una vez iniciada la sesión en la web el diseño será más sencillo e intuitivo para la facilidad de uso del usuario.

La web de inicio será la siguiente la cual tiene un buen diseño con algunas partes en las que el usuario podrá ver reflejado como es el funcionamiento de nuestra app y para qué es. En esta web de inicio se podrá deslizar hasta llegar al pie de página.

Adjunto imágenes de los diferentes puntos de la página de inicio.



Fitness-Tracker

Home App Nuestros Clientes Sobre Nosotros [Registrarse](#)

Virtualiza tu cuidado personal

Fitness-Tracker provee de un seguimiento personal y de salud para tu beneficio personal en base a tus objetivos y metas.

[Consigue nuestra App](#)



Nuestros Servicios

Le brindamos las mejores opciones para usted. Ajústalo a tus necesidades de salud y metas personales. Puedes consultar con nosotros qué tipo de ejercicios son mejores para alcanzar tus objetivos.



Busqueda Inteligente

Encuentra y conoce todo aquello que necesites gracias a nuestra IA interactiva acerca de la salud y el fitness.



Smartwatch

Escoge cualquier smartwatch y comienza a usar la app para mejorar tu salud y alcanzar tus objetivos.



Consulta Personalizada

Consulta gratuitamente todo aquello que necesites para alcanzar tus objetivos con nuestra IA.



Info detallada

Consulta la informacion acerca de tus entrenamientos y rendimiento diario.



Cuenta

Accede a tus registros desde cualquier lugar y dispositivo descargandote la app e iniciando sesion o desde nuestra web.



Tracking

Trackeo tus entrenamientos, tu dia a dia, incluso tus periodos de sueño!!

[Saber más](#)



Líderes en salud y fitness con nuevas tecnologías

Fitness-Tracker ofrece seguimiento de salud, rutinas de gimnasio y dieta de nutrición para alcanzar tus objetivos, accesible a través de dispositivos móviles y en línea para todos.

[Saber más](#)

Descarga nuestra app para dispositivos móviles

Nuestra aplicación dedicada a la recolección de datos a través del software del smartwatch y soluciones para salud, nutrición y rutinas fitness para el beneficio personal del usuario. Comienza a alcanzar tus objetivos.

[Descargar ↓](#)



Fitness-Tracker

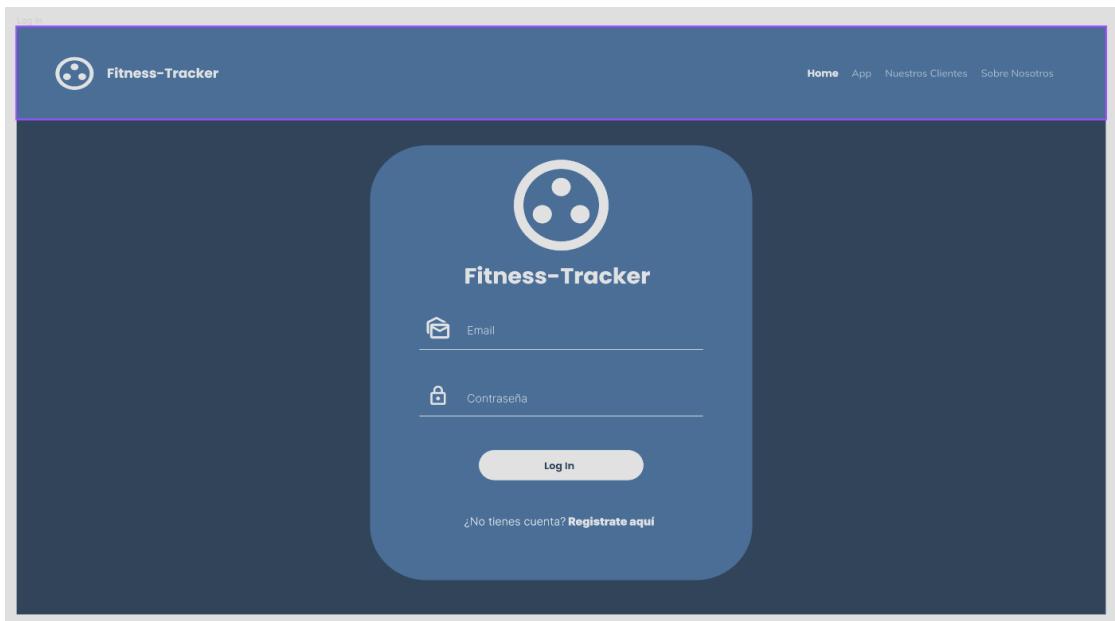
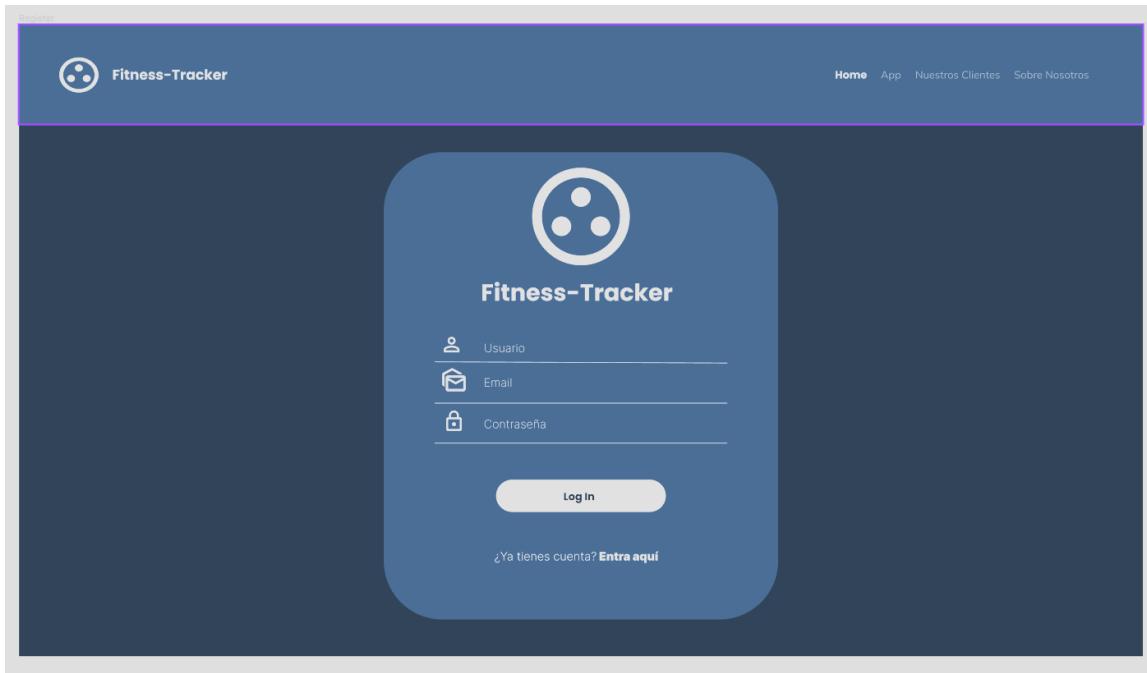
Fitness-Tracker provee de soluciones saludables para tu beneficio personal

©Fitness-Tracker 2024. All rights reserved

| Compañía | Region | Ayuda |
|----------------------|---------------------|-----------------|
| Sobre nosotros | España | Centro de ayuda |
| Encuentra entrenador | Contacto de soporte | Instrucciones |
| App | | Como funciona |

El apartado de login y register para el usuario será una interfaz sencilla que estará compuesta de manera que se centre tanto en los campos de email, contraseña, el botón. Cuando uno de estos campos este seleccionado quedará marcado para que el usuario se dé cuenta de que está situado en uno de los campos todo en una misma ventana y que no tenga que scrollear.

Se adjunta imagen del login y register.



Una vez iniciada la sesión en la web, en la parte de arriba el usuario tendrá una barra de navegación a parte, por donde podrá navegar entre los diferentes apartados de la aplicación web.

En el perfil el usuario podrá ver tanto sus datos personales como su imagen de perfil, objetivos y los resultados actuales. Estos irán cambiando conforme el usuario va avanzando con el uso de la app.

Adjunto apartado del perfil.

The screenshot shows the 'Perfil' (Profile) section of the Fitness-Tracker website. At the top, there's a navigation bar with links for Home, App, Nuestros Clientes, Sobre Nosotros, and a user-specific link 'Guillermo Quintanar'. Below the navigation, the main content area has a dark header with the title 'Fitness-Tracker'. On the left, there's a circular profile picture of a woman wearing a hat and glasses, with the name 'MARISA LOPEZ SÁNCHEZ' and an email icon below it. To the right, there are three sections: 'Datos Personales' (Personal Data) listing Nick (FastMarisa33), Name (Marisa Lopez Sánchez), and Age (25); 'Resultados' (Results) showing three data points (Dato1, Dato2, Dato3) each with a value (Valor_1, Valor_2, Valor_3); and 'Objetivos' (Goals) showing three data points (Dato3, Dato2, Dato1) each with a value (Valor_1, Valor_2, Valor_3).

En el apartado “Hoy” se mostrarían todos los alimentos consumidos por el usuario en el día, pudiendo ver los registros de otro día si así lo quisiera el usuario. Esta pantalla se repartiría entre el desayuno, comida y cena, pudiendo el usuario cambiar los nombres meter otra comida entre medias, y midiendo las calorías que lleva ese día junto con las que debe suplir. Esto se hace para motivar al usuario que intente hacer un poco más de ejercicio o para que consuma la cantidad de agua necesaria al día, marcándolo adecuadamente si ha conseguido la meta diaria.

Adjunto foto del apartado hoy.

The screenshot shows the 'Fitness-Tracker' application interface. At the top, there's a blue header bar with the title 'Fitness-Tracker'. Below the header, a navigation bar includes links for 'Home', 'App', 'Nuestros Clientes', 'Sobre Nosotros', and 'Guillermo Quintanar'. The main content area is divided into two main sections: a large white box on the left and a smaller white box on the right.

Left Box Content:

- Kilocalorías totales**: 0/255 g
- Carbohidratos**: 0/255 g
- Grasas**: 0/255 g
- Proteinas**: 0/255 g

Right Box Content:

- Desayuno**:
 - Dato1.....Valor_1
 - Dato2.....Valor_2
 - Dato3.....Valor_3
- Resultados**:
 - Dato1.....Valor_1
 - Dato2.....Valor_2
 - Dato3.....Valor_3
- Datos Personales**:
 - Dato3.....Valor_1
 - Dato2.....Valor_2
 - Dato1.....Valor_3

En el apartado “Calorías Diarias” el usuario podrá encontrar todo lo relacionado con las consumiciones diarias, cuántas calorías, grasas, carbohidratos y proteínas recomendadas tiene que consumir el usuario. Además, debajo de estos vendrán las comidas y las dietas sugeridas para el usuario por la inteligencia artificial.

Adjunto foto del apartado “Calorías diarias”

The screenshot shows a mobile application interface for 'Fitness-Tracker'. At the top, there's a navigation bar with links for Home, App, Nuestros Clientes, Sobre Nosotros, and a user profile labeled 'Guillermo Quintanar'. Below the navigation is a header 'Requerimiento calórico diario' with a circular icon containing three dots. The main content area is divided into two sections: 'Requerimiento Calórico Diario' on the left and 'Dietas y ejercicios' on the right.

Requerimiento Calórico Diario

0/255 g

- Carbohidratos 255 g
- Grasas 255 g
- Proteínas 255 g

Dietas y ejercicios

- Dato1.....Valor_1
- Dato2.....Valor_2
- Dato3.....Valor_3

Mis comidas

- Dato1.....Valor_1
- Dato2.....Valor_2
- Dato3.....Valor_3

Dietas sugeridas

- Dato3.....Valor_1
- Dato2.....Valor_2
- Dato1.....Valor_3

Link al prototipo o haz clic [aquí](#):

Estudio de la viabilidad del proyecto

El proyecto Fitness Tracker es una aplicación web diseñada para ayudar a los usuarios a monitorear y mejorar su estado físico y salud. La viabilidad del proyecto depende de diversos factores financieros, técnicos y de mercado. Este estudio analiza estos factores para evaluar la factibilidad del desarrollo y mantenimiento de la aplicación.

El análisis financiero del proyecto comienza con la consideración de los costos de desarrollo y mantenimiento. El uso de la inteligencia artificial a través de la API de OpenAI implica suscribirse a su plan de precios, como se explicó en apartados anteriores. Publicar la aplicación en Google Play Store tiene un costo único de 25 euros. Actualmente, el hosting web se realiza en un servicio gratuito, aunque sin dominio propio. El backend se hospeda en una instancia EC2 de Amazon Web Services (AWS) con una IP elástica, y hasta ahora se ha invertido 6 euros, con un crédito gratuito de \$100 a través de AWS Academy que cubre futuros gastos. Los costos de AWS pueden aumentar dependiendo del tráfico y el uso.

En cuanto a los ingresos potenciales, se pueden implementar modelos de suscripción que ofrecen acceso a funciones premium, dietas personalizadas y rutinas de ejercicio. La publicidad dentro de la aplicación puede generar ingresos adicionales, y la venta de productos complementarios como equipos de fitness o suplementos puede ser otra fuente de ingresos, pero es una característica que para esta entrega del proyecto no se considera implementar debido al corto plazo de tiempo del que se dispone.

Desde una perspectiva técnica, la infraestructura del proyecto es robusta y escalable. AWS EC2 permite escalar fácilmente la infraestructura a medida que crece la base de usuarios. La integración con la API de OpenAI proporciona capacidades avanzadas de inteligencia artificial con el modelo de lenguaje GPT4 Turbo Vision (que es el que se está usando ahora), mejorando la personalización y recomendación de rutinas y dietas. Es necesario planificar actualizaciones regulares para mejorar la aplicación, corregir errores y agregar nuevas funcionalidades. Mantener la seguridad de los datos de los usuarios es crucial, y AWS proporciona herramientas de seguridad robustas que deben ser implementadas y monitoreadas constantemente, entre ellas reglas de entrada para sólo exponer aquellos puertos por los que queremos recibir peticiones, incluso podemos llegar a bloquear

peticiones provenientes de ciertas direcciones IP mejorando así la seguridad de nuestro entorno backend.

El análisis de mercado revela que hay una demanda creciente por aplicaciones de salud y fitness. El interés en la salud y el fitness ha aumentado, especialmente con la creciente conciencia sobre la importancia de un estilo de vida saludable. Sin embargo, existe una competencia intensa en el mercado de aplicaciones de fitness. Fitness Tracker debe diferenciarse ofreciendo funcionalidades únicas y personalizadas a través del uso de inteligencia artificial. Los usuarios objetivos son jóvenes y adultos interesados en el fitness y la salud, que buscan soluciones digitales para seguir y mejorar su régimen de fitness.

Documentación del diseño e implementación de la solución adoptada.

Arquitectura backend

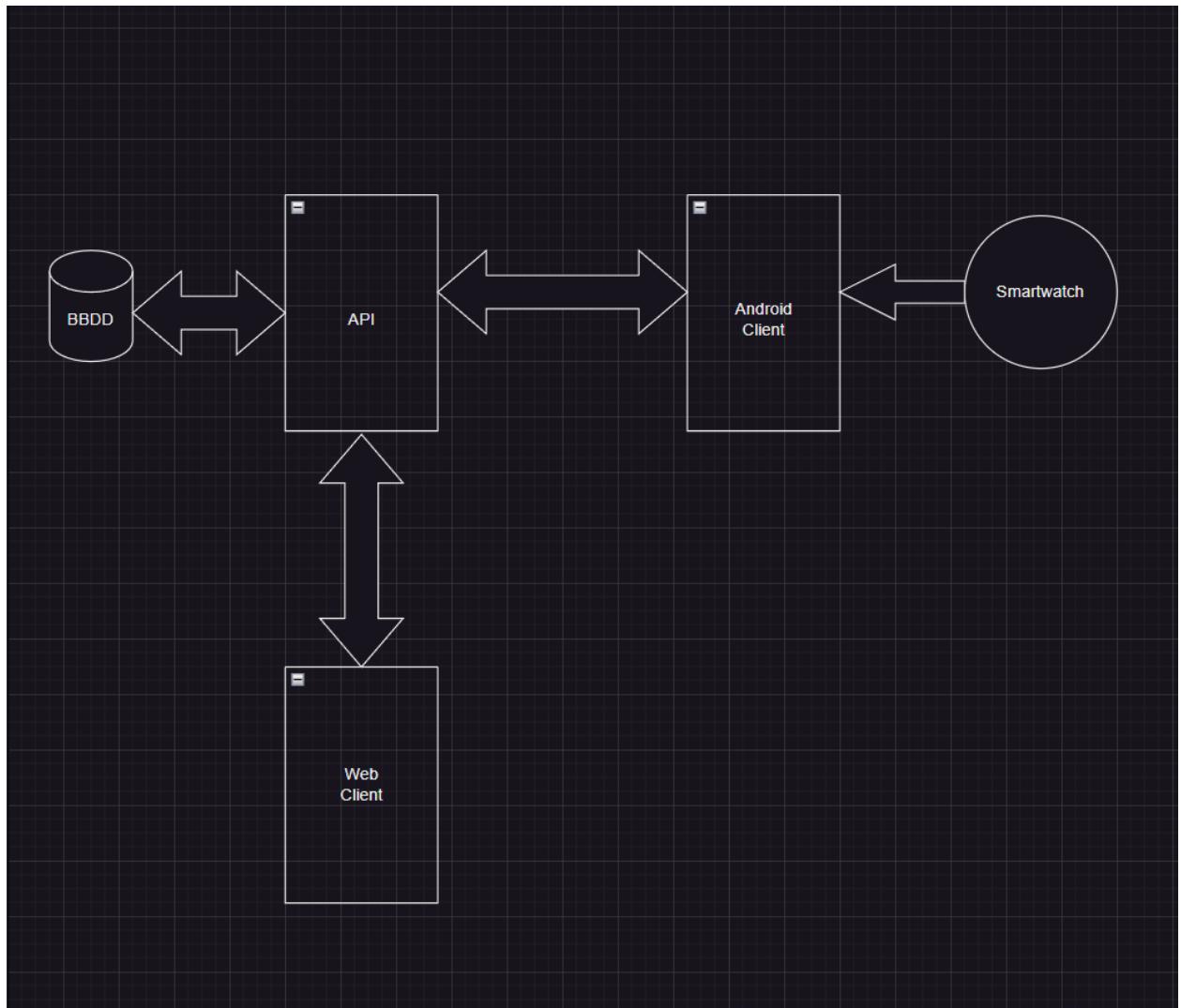
Nuestro sistema consta de tres componentes. Dos servicios REST con sus respectivos endpoints. Uno de los servicios REST es el Base que contiene la lógica de negocio relacionada con el manejo de cuentas de clientes, así como su validación al iniciar sesión; paralelo a este está el servicio REST Sistema External Services, que, entre otras APIs, incluye la de la IA y una sobre información nutricional.

El API de la IA es una que maneja la lógica de negocio respecto la generación de itinerarios relativos a dietas o rutinas y que también puede asistir al cliente en las necesidades que este pueda tener en relación con sus rutinas de ejercicios físicos diaria, entre otros aspectos. Esta API es el centro de nuestro backend, sobre él está integrado el API Gateway, que es la pieza que se expone al exterior, es decir, todos nuestros clientes realizarán peticiones contra el API Gateway, el cual se encarga de enrutar las peticiones a cada uno de los servicios que implementa nuestro backend. Esto especialmente importante ya que nuestro servidor solo expone al exterior el puerto del api Gateway, cortando cualquier petición a otro puerto y dándonos mejor control sobre que servicios pueden realizar peticiones los clientes y cuáles no.

Por otro lado, tenemos dos clientes. Un cliente web y uno Android. Ambos podrán ser capaces de comunicarse con nuestro backend. El cliente Android por su lado consta de otra fuente de datos que es el reloj inteligente, este le proporciona datos como el número de pasos realizados al día, calorías quemadas en un período de tiempo, entre otros.

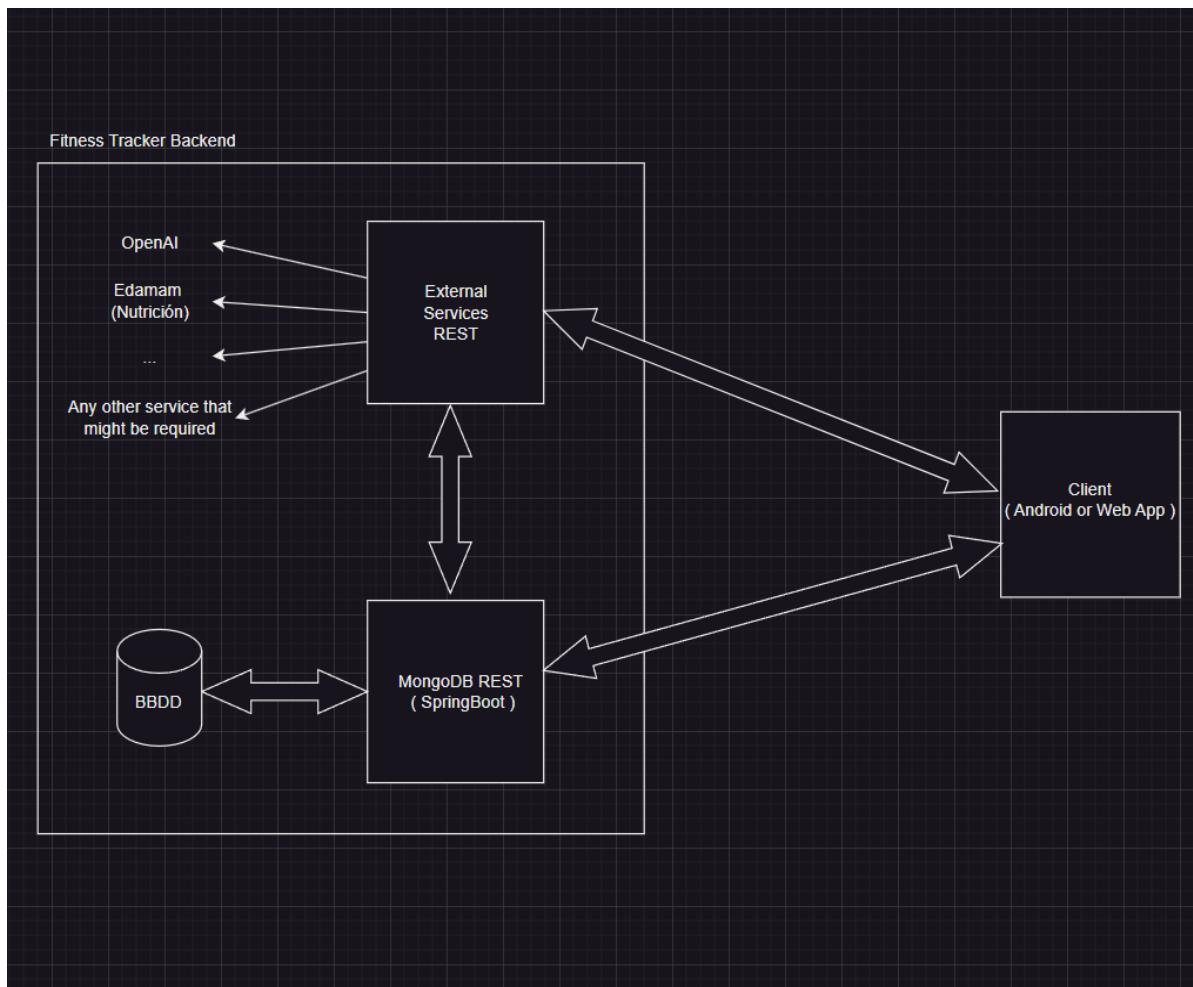
Por la naturaleza de los datos que se van a guardar en la base de datos y cómo están relacionados los modelos, estos se van a serializar a una base de datos documental, para ello se utilizará MongoDB que es un gestor de bases de datos NoSQL con muy eficiente para lecturas frecuentes de datos.

A continuación, se muestra un esquema del sistema:



El backend consta de una, por una parte, de una API REST desarrollada sobre el framework de ASP.NET Core, este es un framework que facilita la creación de un sistema a modo de pasarela entre nuestro backend y servicios externos como el de nutrición y el de OpenAI. ASP.NET Core es un framework con que es sencillo tener una API REST robusta y escalable.

Por otra parte, tenemos la API REST que gestiona la lógica de negocio de los clientes, así como almacenar datos relativos a estos, incluimos aquí las dietas, registros del reloj inteligente, entre otros. Esta API es la que tiene conexión directa con la base de datos y se desarrolla en Java utilizando el framework de Spring Boot. A continuación, un esquema ilustrativo.

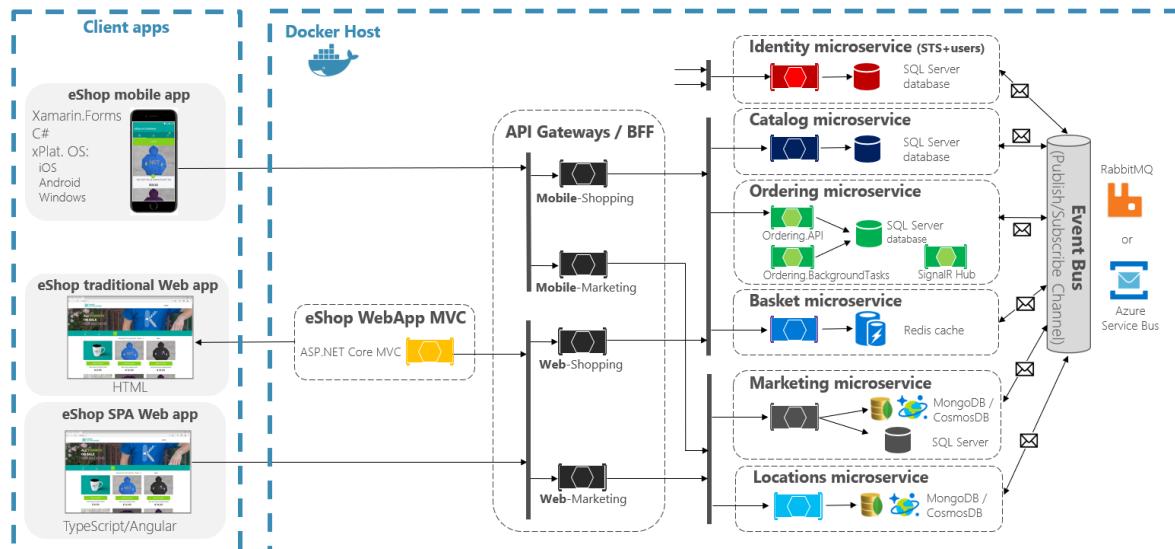


La razón detrás de esta arquitectura yace en que queremos diferenciar los servicios que son propiamente nuestros de los que son dependientes de servicios externos. MongoDB REST gestiona, como se ha mencionado anteriormente, la lógica y negocio relacionados con la manipulación de datos relativos a nuestros clientes. Paralelo a ello, External Services es un módulo que ofrece nuestro backend para ciertas funcionalidades que requieren nuestros clientes pero que no son servicios completamente dependientes de nosotros.

API Gateway REST External Services

Una API Gateway es un componente esencial en arquitecturas de microservicios y sistemas distribuidos. Actúa como un punto de entrada para todas las solicitudes de clientes que acceden a múltiples servicios en la infraestructura. En esencia, proporciona una capa de abstracción entre los clientes y los microservicios subyacentes, simplificando la gestión de las solicitudes, mejorando la seguridad, y permitiendo funcionalidades adicionales como la autenticación, autorización, monitoreo y enrutamiento. En lugar de exponer cada servicio individualmente a través de su propia dirección IP y puerto, se expone una sola dirección IP y puerto para el API Gateway, que enruta las solicitudes a los servicios apropiados.

eShopOnContainers reference application
(Development environment architecture)



Las funciones principales de una API Gateway incluyen el enrutamiento de solicitudes, autenticación y autorización, seguridad, transformación de datos, y monitoreo y análisis. Dirige las solicitudes de los clientes a los servicios correspondientes en función de la URL, el método HTTP u otros criterios. Gestiona la autenticación de los clientes y autoriza el acceso a los servicios según las políticas de seguridad establecidas. Proporciona una capa de seguridad adicional al actuar como un punto de entrada único a la infraestructura, protegiendo los servicios subyacentes de exposiciones no deseadas. Realiza transformaciones en los datos de solicitud y respuesta según sea necesario, para adaptarlos a los formatos esperados por los clientes o servicios. Recopila métricas y registra datos de tráfico para realizar análisis de rendimiento, seguimiento de errores y tomar decisiones informadas sobre la escalabilidad y la optimización de la infraestructura.

Fitness Tracker consta de una API Gateway implementada con Ocelot, una librería de terceros en .NET para integrar este tipo de sistemas en backend.

Ocelot es una herramienta de enrutamiento y API gateway de código abierto para .NET que permite a los desarrolladores crear gateways de API personalizados.

Con Ocelot, podemos implementar características como enrutamiento, autenticación, autorización, control de velocidad, balanceo de carga, almacenamiento en caché, transformación de respuestas, orquestación, monitoreo y más, de manera simple y eficiente, lo que mejora la eficiencia del despliegue y la escalabilidad de la aplicación.

Ocelot se utiliza para proporcionar un único punto de entrada central para las solicitudes a múltiples microservicios. Utiliza un archivo de configuración para definir las rutas de enrutamiento y las políticas de seguridad.

La arquitectura de microservicios ha ganado una gran popularidad en los últimos años, principalmente debido a que mejora la escalabilidad, flexibilidad y rendimiento del flujo de nuestro sistema.

Dado que las aplicaciones basadas en microservicios comprenden varios servicios diferentes, a menudo necesitamos una interfaz común o gateway para llamar a estos servicios para que podamos definir y gestionar todas las peticiones en un solo punto de entrada, en lugar de replicarlas en todos los servicios posteriores.

Aquí es precisamente donde entra en juego un API gateway, proporcionando un único punto de entrada para dirigir el tráfico a varios microservicios y un lugar central para implementar seguridad, monitoreo, entre otros aspectos.

Sin un gateway, tendríamos que implementar estas funcionalidades en cada uno de los servicios, lo que sería una tarea desalentadora.

Sistema inteligencia artificial

El sistema de inteligencia artificial es un módulo tanto para la aplicación Web como el cliente Android que sirve de asistente virtual para el usuario. Permite al usuario realizar consultas a la hora de construir rutinas de ejercicios diarias o semanales.

Empezando por la inteligencia artificial que se utiliza en este módulo. Se va a utilizar el modelo de lenguaje GPT 4 Turbo con Visión, la versión de GPT capaz de aceptar imágenes y trabajar sobre éstas. Es una versión mejorada de GPT 3.5 Turbo, con la mejora de que puede procesar y generar imágenes, juntamente con la capacidad de poder procesar aún más palabras con más fluidez y precisión. Open AI expone una serie de endpoints a través de los cuales se puede realizar peticiones a estos modelos. Para ello es necesario primero una clave API que se puede generar desde su página.

Este servicio no es gratuito y está sujeto a unos límites de uso. Los límites de uso de la API son restricciones cruciales que la plataforma impone para regular el acceso y garantizar una experiencia equitativa para todos los usuarios. Estos límites, también conocidos como "Rate Limits", definen la cantidad máxima de veces que un usuario o cliente puede acceder a los servicios dentro de un período de tiempo específico.

Las razones principales para imponer estas restricciones están recogidas en la guía de desarrollo y son:

Protección contra el Abuso: La API se protege contra posibles abusos o malos usos que podrían sobrecargar los servidores o interrumpir el servicio. Esto se logra limitando la cantidad de solicitudes que un usuario puede realizar en un período de tiempo dado.

Equidad de acceso: los límites de uso garantizan que todos los usuarios tengan acceso justo a nuestros servicios. Al limitar el número de solicitudes que un usuario puede hacer, evitamos que un individuo o entidad monopolice los recursos disponibles, lo que podría ralentizar la experiencia para otros usuarios.

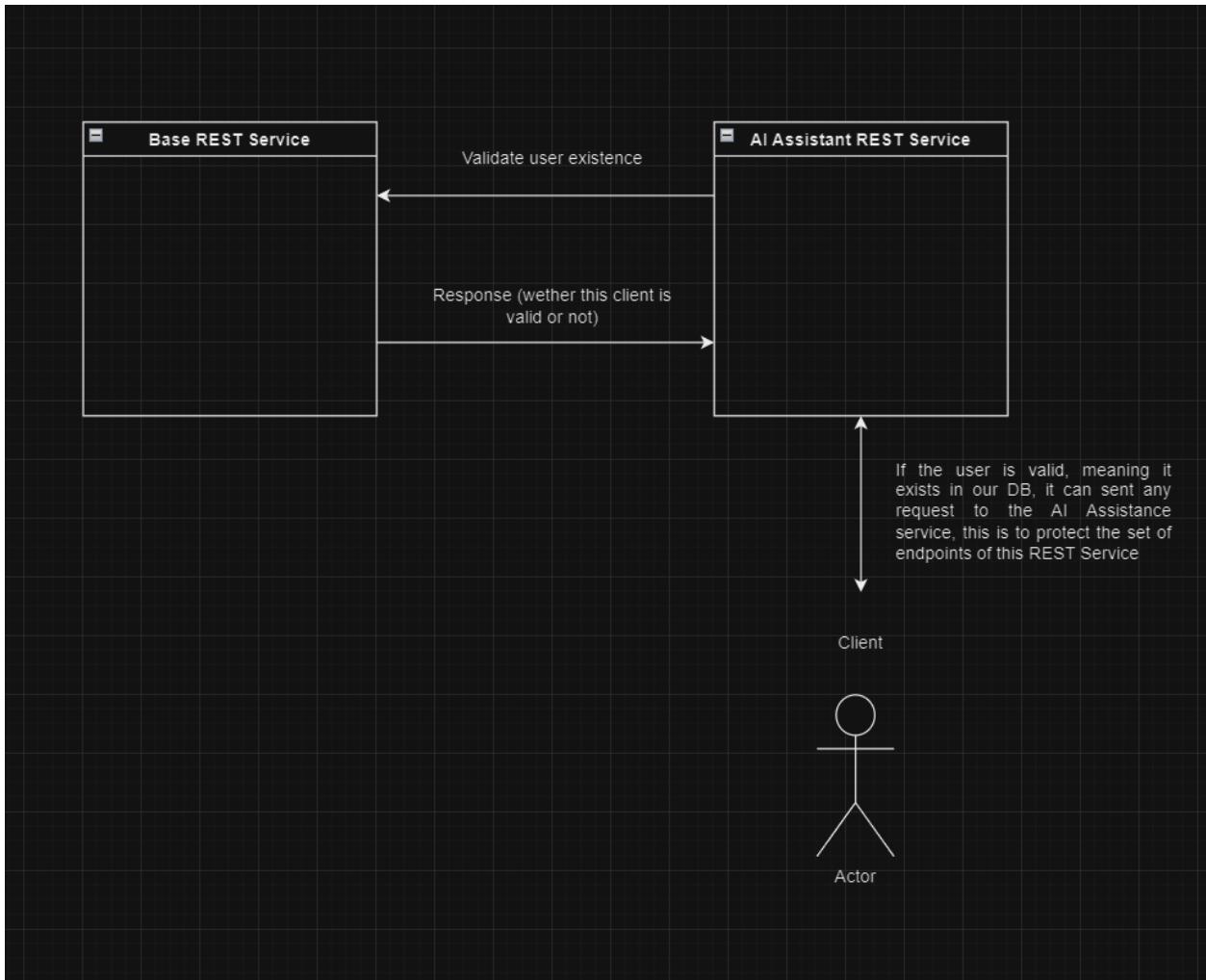
Gestión de la carga de infraestructura: Los límites de uso ayudan a administrar la carga total en nuestra infraestructura. Al controlar la cantidad de solicitudes que recibimos, podemos garantizar un rendimiento estable y consistente de nuestros servidores, incluso durante períodos de alta demanda.

Los límites de uso se expresan en diferentes métricas, que incluyen solicitudes por minuto (RPM o Request Per Minute), solicitudes por día (RPD o Request Per Day), tokens por minuto (TPM o Token Per Minute), tokens por día (TPD o Tokens Per Day) e imágenes por minuto (IPM o Images Per Minute). Estos límites se aplican según lo que ocurra primero en cada caso.

Es importante destacar que los límites de la API se definen tanto a nivel de organización como a nivel de proyecto, no a nivel de usuario individual. Además, los límites pueden variar según el modelo específico que se esté utilizando en la API.

La API REST que se comunicará con los servicios de OpenAI juntamente con los servicios de autenticación de usuarios se va a realizar con el lenguaje de programación C# usando ASP.NET Core, un framework que facilita mucho la creación de servicios REST.

Para realizar peticiones a la API de OpenAI se va a utilizar la librería de .NET OpenAI-API-dotnet. Una librería de código abierto que facilita la tarea de realizar peticiones contra la API de OpenAI, agilizando el trabajo de procesar las respuestas recibidas de los endpoints de Open AI en clases e interfaces más sencillas de usar.



El sistema de inteligencia artificial tendrá dos módulos: uno de asistente y otro como generador de dietas personalizadas.

Para generar las dietas se ingresarán los siguientes parámetros en un formulario a través de la aplicación web:

- Edad: Campo de texto.
- Género: Un menú desplegable que consta de Masculino o Femenino.
- Peso: Campo de texto (en kilogramos).
- Altura: Campo de texto (en centímetros)
- Nivel de Actividad Física: Menú desplegable que consta de sedentario, Moderado, Activo, muy Activo.
- Objetivo Principal: Menú desplegable que consta de Perder Peso, Mantener Peso, Ganar Masa Muscular, Mejorar Salud.

- Restricciones Alimenticias: [Casillas de verificación: Vegetariano / Vegano / Sin Gluten / Sin Lácteos / Sin Frutos Secos / Otro]
- Preferencias de Alimentos: Especificación de alimentos preferidos o no deseados, aquí el usuario podrá seleccionarlos, el backend enviará información nutricional recogida de Edamam de la cual se podrá seleccionar comidas ya con la información nutricional completa.
- Habilidad en la Cocina: Menú desplegable: Principiante, Intermedio, Avanzado, si el usuario quiere restringirse a comidas fáciles de realizar.
- Notas o Comentarios Adicionales: Área de texto para que el usuario pueda agregar cualquier información adicional relevante.

Arquitectura de la base de datos

Dietas

Una dieta se puede definir como el conjunto de alimentos y bebidas que una persona consume regularmente para mantener su salud y nutrición. Sin embargo, en nuestra aplicación consideramos la dieta como un plan estructurado y personalizado de ingesta de alimentos diseñado con objetivos específicos, como perder peso, ganar masa muscular, mejorar la salud cardiovascular, entre otros aspectos.

Para cada usuario se mantiene constancia de las dietas que lleva a lo largo del tiempo. Un usuario puede tener varias dietas a lo largo del tiempo en donde puede incluir todo tipo de comidas, las cuales incluso pueden repetirse entre dietas.

Las dietas se generan con la ayuda del módulo de External Services, a través del soporte de la inteligencia artificial. Luego esta se almacena en MongoDB REST que es nuestra API REST que se encarga de manipular todos los datos e información relativa al cliente.

Para generar una dieta, el usuario deberá cumplir un formulario similar al siguiente:

Ejercicios físicos

El módulo de Ejercicios Físicos de nuestra aplicación complementa el aspecto dietético del plan de salud y bienestar del usuario. Este módulo proporciona un enfoque integral para ayudar a los usuarios a alcanzar sus objetivos de fitness y mejorar su calidad de vida.

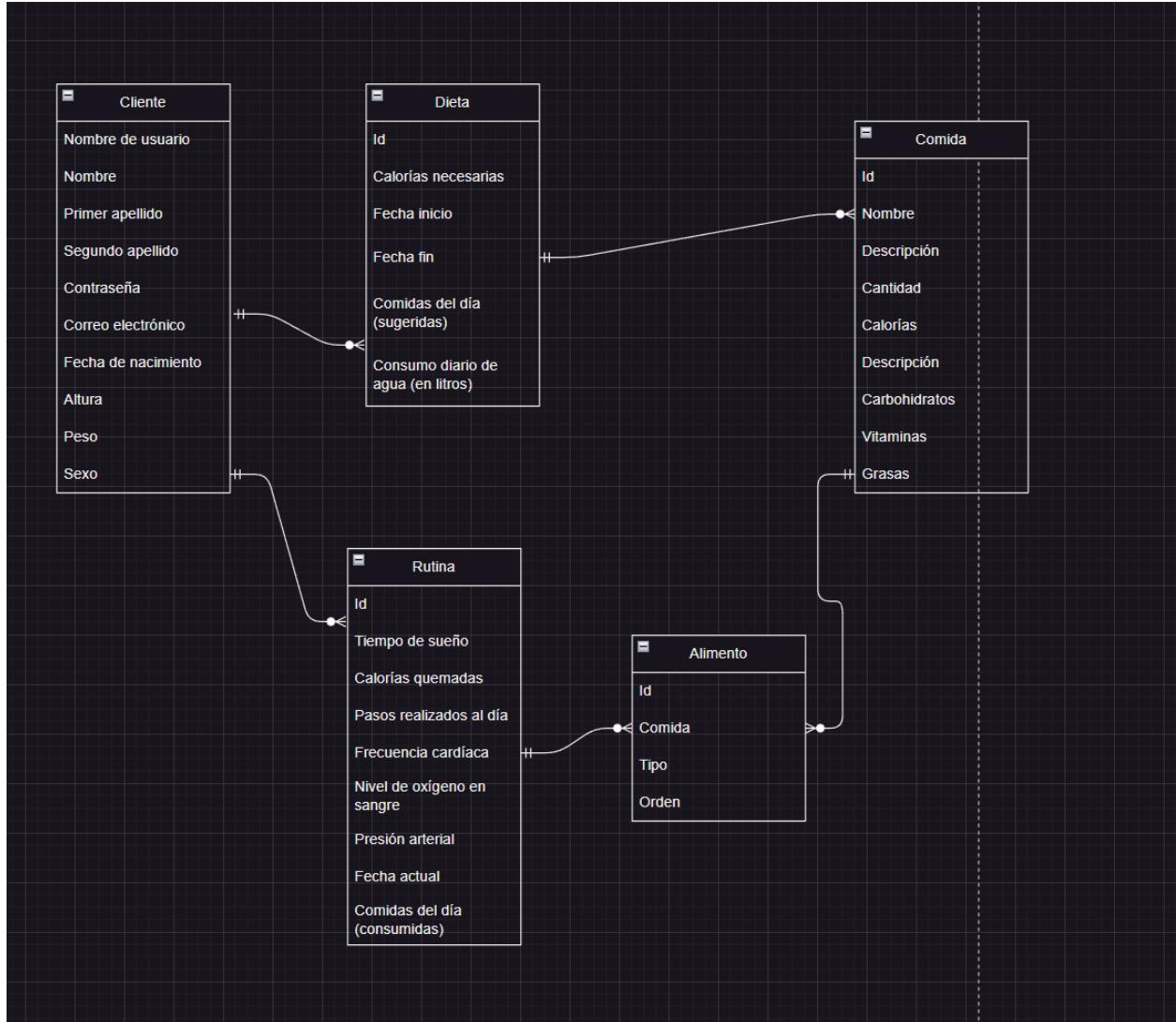
Este módulo permite a los usuarios registrar sus sesiones de ejercicio, incluyendo el tipo de actividad, la duración y la intensidad. También muestra a los usuarios un resumen de su actividad física a lo largo del tiempo, lo que les permite visualizar su progreso y establecer metas alcanzables.

Consejos de alimentación y ejercicio

El módulo de Consejos de Alimentación y ejercicio proporciona a los usuarios orientación y apoyo adicional para ayudarles a adoptar hábitos de vida más saludables.

Este módulo ofrece consejos específicos de alimentación adaptados a las necesidades y objetivos de cada usuario, teniendo en cuenta factores como edad, género, nivel de actividad y preferencias alimenticias. Además, proporciona ideas y recetas de comidas nutritivas y equilibradas, ayudando a los usuarios a diversificar su dieta y disfrutar de opciones más saludables. Facilita la planificación de comidas diarias de acuerdo con los objetivos de salud y las preferencias personales de los usuarios, y proporciona recomendaciones de ejercicios específicos y estrategias de entrenamiento adaptadas a sus objetivos de fitness.

A continuación, se muestra un esquema del modelo de datos de nuestra base de datos.



En la arquitectura de la base de datos de nuestra aplicación, hemos identificado cuatro modelos principales: Cliente, Dieta, Ejercicio físico (Rutina) y Comida. Cada uno de estos modelos tiene su propio propósito y están interrelacionados de acuerdo con las funcionalidades de nuestra aplicación.

Modelo de Cliente

El modelo de Cliente representa a los usuarios de nuestra aplicación, cada uno de los cuales tiene una cuenta registrada en nuestra base de datos, la cuenta mantiene constancia de los datos personales del cliente. A continuación, se listan los datos personales del cliente que se consideran relevantes para la aplicación:

- **Nombre de usuario:** Campo opcional con que el usuario puede indicar cómo quiere ser dirigido en la aplicación, si no se especifica, se dirige a él por su nombre.
- **Nombre:** Nombres del cliente.
- **Primer apellido:** Primer apellido del cliente.
- **Segundo apellido:** Segundo apellido del cliente, no obligatorio en caso de que el cliente no lo tenga.
- **Contraseña:** Contraseña que se guarda encriptada en nuestra base de datos.
- **Correo electrónico:** Correo electrónico del cliente.
- **Fecha de nacimiento:** Fecha de nacimiento del cliente, con que se puede deducir la edad donde es necesario.
- **Altura:** Altura en centímetros del cliente.
- **Peso:** Peso en kilogramos del cliente.
- **Sexo:** Sexo del cliente. Donde puede especificar mujer u hombre. Este campo hace referencia al sexo biológico ya que se considera un factor importante para luego poder determinar las dietas.

Modelo de Dieta

El modelo de Dietas representa los planes de dieta disponibles en nuestra aplicación. Un cliente podrá tener varias dietas a lo largo del tiempo, tantas como quiera y, obviamente podamos almacenar en nuestro servidor. Las dietas son planes de comida semanales que se ajustan a las necesidades nutricionales y alimenticias que quiere llevar el cliente en un intervalo de tiempo determinado, por ende, son recomendaciones, y no necesariamente lo que el usuario ha consumido en concreto. La dieta una vez creada se puede modificar. A continuación, se listan las propiedades de una dieta:

- **Id:** Identificador único para cada comida, generado automáticamente por el sistema.
- **Calorías objetivo:** Cantidad de calorías a quemar acorde a la dieta, medidas en Kcal o Julios.
- **Fecha de inicio:** Fecha de inicio de la dieta.
- **Fecha de fin:** Fecha de fin de la dieta.
- **Comidas del día (sugeridas):** Comidas sugeridas para consumir en esta dieta.
- **Consumo de agua diario:** Consumo de agua diario mínimo recomendado.

Modelo de Comida

El modelo de Comidas representa los alimentos específicos o recetas específicas que forman parte de cada dieta o de una rutina en concreto, es decir, se diferencia entre las comidas que se sugieren para seguir una dieta en concreto y las que el usuario ha consumido. A continuación, se listan sus propiedades:

- **Id:** Identificador único para cada comida, generado automáticamente por el sistema.
- **Nombre:** Nombre de la comida (puede ser una receta completa).
- **Descripción:** Descripción de la comida.
- **Calorías:** Cantidad de calorías proporcionadas por esta comida.
- **Carbohidratos:** Carbohidratos proporcionados por esta comida.
- **Vitaminas:** Tipo de vitaminas proporcionadas por esta comida.
- **Grasas:** Valor en grasas proporcionado por esta dieta.

Cada alimento puede tener atributos como nombre, descripción, valor nutricional, etcétera. Los alimentos están relacionados con las dietas, ya que cada alimento específico

solo se relaciona con una dieta en particular. Es el cliente quien puede seleccionar los alimentos dentro de una dieta, en función de sus necesidades y preferencias dietéticas.

Modelo de Ejercicio Físico (Rutina)

El modelo de rutina representa la información recopilada de dispositivos del seguimiento de la actividad física del cliente, incluyendo datos recopilados de relojes inteligentes y las comidas consumidas al día.

- **Id:** Identificador único para rutinas, generado y asignado automáticamente por el sistema gestor de la base de datos.
- **Tiempo de sueño:** Indica el tiempo de sueño invertido para inhibir la somnolencia durante el día siguiente, representado en minutos.
- **Calorías quemadas:** Cantidad de calorías quemadas por ejercicio físico realizado durante el día, medido en kcal o Julios (J)
- **Pasos realizados al día:** Número de pasos realizados al día.
- **Frecuencia cardíaca:** Frecuencia de las pulsaciones del corazón por minuto. Medido en BPM o *Beats Per Minute*.
- **Nivel de oxígeno en sangre:** Porcentaje de oxígeno en sangre.
- **Presión arterial:** Presión con que circula la sangre por nuestro organismo. Medido en milímetros de mercurio, ya que resulta más informativo de cara al cliente.
- **Comidas del día:** Alimentos consumidos al día, ver el modelo de Alimento.

Modelo de Alimento

Este modelo nace como consecuencia de que el usuario no necesariamente va a ceñirse a las restricciones de la dieta y acabará consumiendo algún alimento del día. Este modelo mantiene constancia de ello y siempre es parte de una rutina, no existe por sí mismo, es decir, tiene una relación de agregación con el modelo de rutina.

Los alimentos se obtienen del servicio externo de FT - Alimentos. Estos luego se serializan a nuestra base de datos cuando se genera la dieta ya que la dieta se serializa a la base de datos y esta debe mantener constancia de los alimentos que la componen. En esencia esto implica que tenemos datos duplicados y parte de ellos en una base de datos totalmente

ajena a nosotros. Esto es así porque esto nos libera, como desarrolladores, de la carga de mantener actualizada una base de datos con información nutricional lo suficientemente extensa como para asistir las necesidades de un cliente potencial.

A continuación, se listan sus atributos:

- **Id:** Identificador único generado automáticamente por la base de datos para identificar los diferentes alimentos.
- **Comida:** Comida consumida en una fecha específica.
- **Tipo de comida:** Especifica el tipo de comida, que puede ser el desayuno, almuerzo, merienda o la cena.
- **Orden:** Especifica el orden de plato, que puede ser primer plato, segundo plato, tercer plato.

Interrelaciones

Cliente y Dietas: Un cliente puede tener una relación de uno (obligatorio) a muchos (opcional) con las dietas, lo que significa que un cliente puede tener asociadas varias dietas a lo largo del tiempo, los clientes pueden obtener dietas similares, pero estas son únicas por cada uno de ellos.

Dietas y Comidas: Existe una relación de muchos (opcional) a muchos (opcional) entre las dietas y los alimentos, lo que significa que cada alimento específico está asociado con varias dietas, al igual que varias dietas pueden tener asociados alimentos varios alimentos.

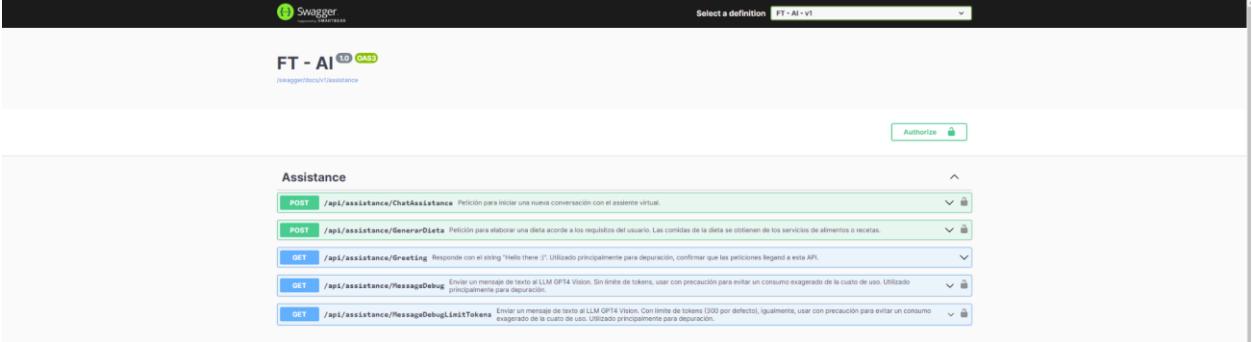
Cliente y Rutina: Un cliente tiene una relación de uno a muchos con la rutina, lo que significa que un cliente puede tener múltiples registros de datos diarios relativos a su actividad física, consumo de alimentos, entre otros aspectos.

Alimento y Comida: Como se ha explicado anteriormente, los alimentos son las comidas que el usuario ha consumido, no necesariamente los mismos que sugiere la dieta para un día en concreto. La relación es de muchos (opcional) a muchos (opcional).

Código fuente comentado

Como se ha mencionado anteriormente nuestro sistema consta de un backend desarrollado sobre .NET y uno de los servicios desarrollado sobre Spring Boot con Java. Los métodos que nuestra API REST expone son los siguientes:

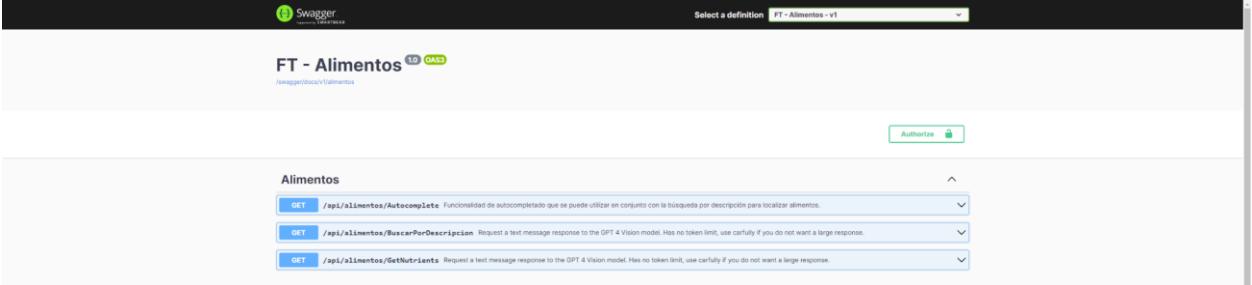
- Endpoints de inteligencia artificial.



The screenshot shows the Swagger UI interface for the 'FT - AI' API. At the top, there's a navigation bar with the title 'FT - AI' and a 'Authorize' button. Below the title, it says 'Select a definition: FT - AI - v1'. The main content area is titled 'Assistance' and lists five endpoints:

- POST /api/assistance/ChatAssistance**: Petición para iniciar una nueva conversación con el asistente virtual.
- POST /api/assistance/GenerarDia**: Petición para elaborar una dieta acorde a los requisitos del usuario. Las comidas de la dieta se obtienen de los servicios de alimentos o recetas.
- GET /api/assistance/Greeting**: Responde con el string "Hello there!" Utilizado principalmente para depuración, confirmar que las peticiones llegan a esta API.
- GET /api/assistance/MessageDebug**: Envíar un mensaje de texto al LLM GPT4 Vision. Sin límite de tokens, usar con precaución para evitar un consumo exagerado de la cuota de uso. Utilizado principalmente para depuración.
- GET /api/assistance/MessageDebugLimitTokens**: Envíar un mensaje de texto al LLM GPT4 Vision. Con límite de tokens (300 por defecto), igualmente, usar con precaución para evitar un consumo exagerado de la cuota de uso. Utilizado principalmente para depuración.

- Endpoints de información nutricional



The screenshot shows the Swagger UI interface for the 'FT - Alimentos' API. At the top, there's a navigation bar with the title 'FT - Alimentos' and a 'Authorize' button. Below the title, it says 'Select a definition: FT - Alimentos - v1'. The main content area is titled 'Alimentos' and lists three endpoints:

- GET /api/alimentos/Autocomplete**: Funcionalidad de autocompletado que se puede utilizar en conjunto con la búsqueda por descripción para localizar alimentos.
- GET /api/alimentos/BuscarPorDescripcion**: Request a text message response to the GPT 4 Vision model. Has no token limit, use carefully if you do not want a large response.
- GET /api/alimentos/GetNutrients**: Request a text message response to the GPT 4 Vision model. Has no token limit, use carefully if you do not want a large response.

- Endpoints relativos al usuario

The screenshot shows the Swagger UI for the 'FT - Base' API. The 'Base' endpoint group is selected. It contains several API operations:

- PUT /api/client/CambiarPassword**: Cambia la contraseña de un usuario.
- GET /api/client/GetDatosUsuario**: Obtiene los datos del usuario.
- GET /api/client/GetDietaDeUsuario**: Obtiene la dieta por ID de un usuario específico.
- GET /api/client/GetListDietasDeUsuario**: Obtiene la lista de dietas del usuario.
- GET /api/client/GetListRutinasUsuario**: Obtiene las rutinas de actividad física y alimentación del usuario a lo largo de un período de tiempo.
- GET /api/client/GetRutinaPorId**: Obtiene los datos de la rutina de actividad física y alimentación del usuario identificada por su ID.
- POST /api/client/Login**: Endpoint para solicitar iniciar sesión.
- PUT /api/client/ModificarDatosUsuario**: Modifica los datos del usuario.
- PUT /api/client/ModificarRutina**: Modifica una nueva rutina.
- GET /api/client/ModificarRutina**: Actualiza los datos de la rutina de actividad física y alimentación del usuario identificado por su ID.
- POST /api/client/RegistrarDieta**: Registra una nueva dieta.
- POST /api/client/RegistrarRutina**: Registra una rutina de actividad física y alimentación del usuario.
- POST /api/client/RegistrarUsuario**: Endpoint para registrar un nuevo usuario.

La API REST de ASP.NET sigue la misma estructura en todos los endpoints se recibe un objeto JSON de entrada que se valida, si es válido se redirecciona la petición al servicio y si no se devuelve un Bad Request a la petición. En los servicios se realiza la lógica de negocio, así como acceder a bases de datos o servicios externos si fuera necesario.

Ejemplo de un endpoint de backend:

```
/// Las comidas de la dieta se obtienen de los servicios de alimentos o recetas.
/// </summary>
/// <param name="model">Requisitos de la dieta</param>
/// <returns>Respuesta del modelo de vista. Ver: <see cref="ResponseGenerarDietaVM"/></returns>
[HttpPost("GenerarDieta")]
[Authorize] // authorize
[ProducesResponseType(StatusCodes.Status200OK, Type = typeof(ResponseGenerarDietaVM))]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
0 references
public async Task<ActionResult<ResponseGenerarDietaVM>> GenerarDieta([FromBody] RequestGenerarDieta model)
{
    var result = _validatorRequestDieta.Validate(model);
    if (result == null || !result.IsValid)
    {
        return BadRequest(result?.Errors);
    }

    var res = await _assistanceService.GenerarDieta(model);

    return Ok(res);
}
```

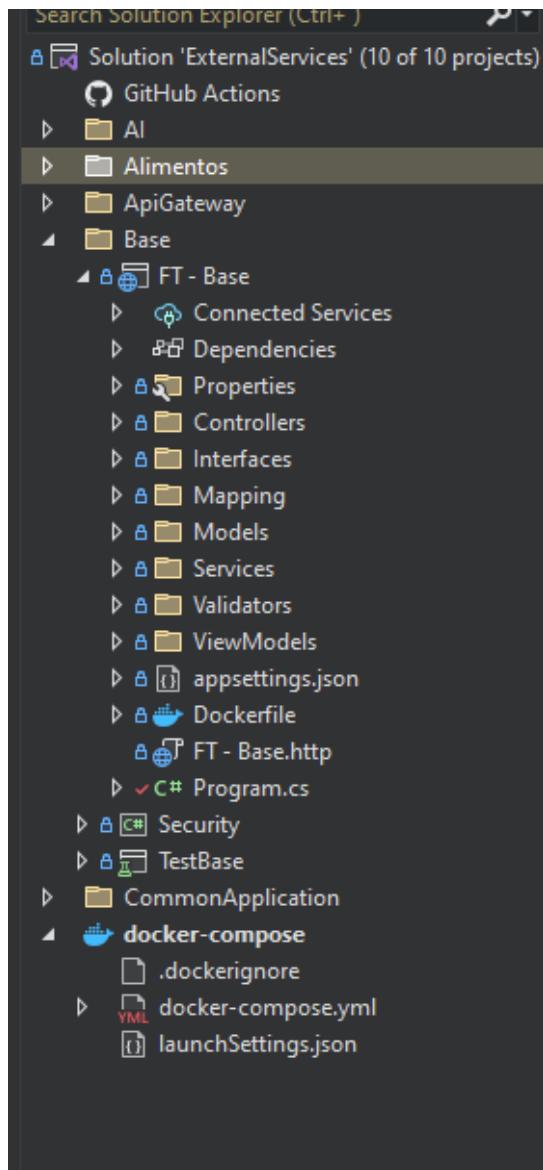
Esta estructura se repite a lo largo del resto de endpoints de nuestro backend.

Estructura de carpetas

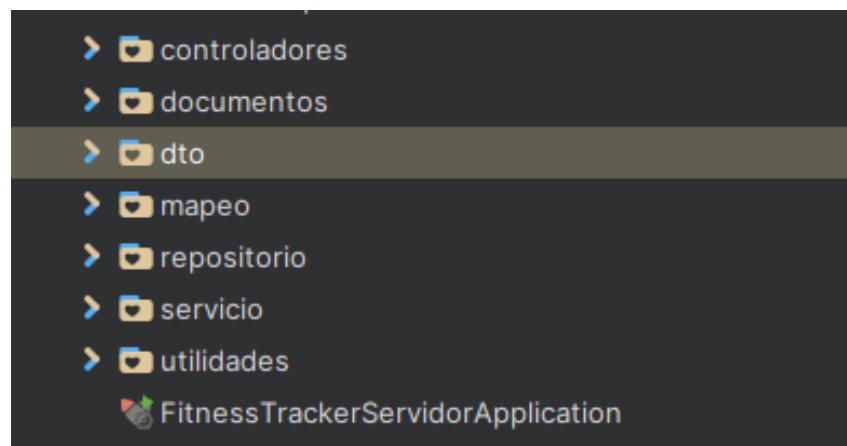
Nuestros servicios siguen la misma estructura. Tenemos varias carpetas, cada una de ellas con una funcionalidad determinada. Ciertas carpetas como la de Properties las crea el editor al crear un proyecto nuevo de ASP.NET y es allí donde se guarda información relativa al proyecto como los puertos que utiliza, protocolos de comunicación, entre otros aspectos. En la carpeta Controllers tenemos los controladores. En la carpeta Interfaces tenemos las interfaces, definiciones de funciones de uso común entre diferentes clases y que también facilitan la posibilidad de inyección de dependencias. En la carpeta Mapping tenemos los perfiles de mapeo, que son clases que nos permiten construir POJOs a partir de otros, esto es especialmente útil a la hora de deserializar objetos JSON de servicios externos para poder trabajar con ellos en nuestro backend. En la carpeta Services tenemos los servicios, son bloques de código que llevan a cabo la lógica de negocio. En la carpeta Validators tenemos los validadores, que son clases que se encargan de validar los datos que recibe nuestro backend; y, finalmente, en la carpeta ViewModels tenemos los modelos de vista que son clases que encapsulan el conjunto de datos que nosotros queremos hacer visibles al cliente exterior.

Cada servicio consta de un dockerfile, ya que, por cuestión de despliegue, construimos las imágenes para poder generar contenedores de nuestros servicios y así poder desplegar nuestra aplicación en un servidor remoto.

Por otra parte, ciertos servicios constan de los llamados Class Library, que son librerías que contienen utilidades que no son comunes a todos los microservicios pero que tienen sentido ser extraídos a un módulo aparte, en la imagen se puede ver por ejemplo el módulo de Security, que es un módulo para gestionar el sistema de autenticación con JWT.



Por otra parte, tenemos una aplicación API REST desarrollada sobre Spring Boot que tiene la siguiente estructura de carpetas:



La estructura de carpetas de una aplicación Spring Boot con MongoDB está diseñada para organizar de manera clara y eficiente las distintas responsabilidades del código, facilitando su mantenibilidad y escalabilidad. En la carpeta "controladores" se encuentran las clases responsables de manejar las solicitudes HTTP entrantes y definir los endpoints de la API REST.

La carpeta "documentos" contiene las clases que representan las entidades o modelos que se almacenarán en la base de datos MongoDB. Estas clases están anotadas con las anotaciones de Spring Data MongoDB para mapear las colecciones de la base de datos a objetos Java, definiendo los campos que serán persistidos.

En la carpeta "dto" se agrupan las clases utilizadas para transferir datos entre las distintas capas de la aplicación y hacia o desde el cliente. Los DTO encapsulan los datos enviados en las solicitudes y respuestas de la API, proporcionando una forma segura y controlada de transferir información, además de incluir validaciones y transformaciones necesarias para la comunicación con el cliente.

La carpeta "mapeo" contiene clases encargadas de convertir entre entidades de dominio y DTOs. Estos mappers centralizan la lógica de transformación de datos, asegurando que la conversión entre diferentes representaciones sea manejada de manera consistente y separada del resto del código.

En la carpeta "repositorio" se encuentran las interfaces que extienden de Spring Data MongoDB Repository. Estas interfaces proporcionan métodos CRUD y consultas personalizadas para interactuar con la base de datos MongoDB, abstrayendo la lógica de acceso a datos y facilitando la realización de operaciones con la base de datos.

La carpeta "servicio" alberga las clases que implementan la lógica de negocio de la aplicación. Los servicios son responsables de procesar la lógica compleja, realizar validaciones y coordinar las operaciones entre diferentes partes de la aplicación. Los controladores llaman a los métodos de los servicios para cumplir con las solicitudes del cliente, manteniendo una clara separación entre la lógica de negocio y la de presentación.

Por último, la carpeta "utilidades" incluye clases y métodos que proporcionan funcionalidades auxiliares y utilitarias reutilizables en diferentes partes de la aplicación. Estas utilidades pueden incluir funciones de ayuda para manipulación de cadenas, fechas y manejo de excepciones, entre otros.

La estructura de los endpoints por lo general es la misma.

```
 531  @PutMapping("modificarrutina")
 532  ResponseEntity<ResponseModificarRutina> modificarRutina(@RequestBody RequestModificarRutina model) {
 533      ResponseModificarRutina responseData = ResponseModificarRutina.builder().build();
 534      ResponseEntity<ResponseModificarRutina> response;
 535
 536      try {
 537          Boolean result = usuarioServicio.modificarRutina(model);
 538          responseData.setSuccess(true);
 539
 540          if (result) {
 541              responseData.setModifiedAt(LocalDateTime.now());
 542              responseData.setResponseDescription("Rutina modificada con éxito");
 543          } else {
 544              responseData.setSuccess(false);
 545              responseData.setResponseDescription("No se pudo modificar la rutina. Esta es inválida o el usuario no existe.");
 546          }
 547
 548          response = new ResponseEntity<>(responseData, HttpStatus.OK);
 549      } catch (Exception e) {
 550          responseData.setSuccess(false);
 551          responseData.setResponseDescription(e.getMessage());
 552          response = new ResponseEntity<>(responseData, HttpStatus.INTERNAL_SERVER_ERROR);
 553      }
 554
 555      return response;
 556  }
```

Los endpoints siempre reciben un objeto y devuelven un objeto. Esta aplicación sólo consta de un controlador porque está sumamente enfocada sobre el usuario.

Al recibir los datos estos se redirigen a los servicios, los cuales entre otras cosas validan que existan los objetos con los ids que se suministran. Se realiza control de errores mediante bloques try y catch en caso de haber una excepción, en cuyo caso se responde con un Internal Server Error.

Si la excepción es de tipo RuntimeExcepción se devuelve todavía un OK para facilitar el desarrollo con Retrofit en el entorno de Android.

Aplicación Android

Para empezar, tenemos la actividad principal, donde a través de dependencias con Hilt y Dagger, podremos crear los viewmodels en cada composable. Además, creamos un navegador para la aplicación y un viewModel general, que tendrá un patrón Singleton que controlará los estados, como el estado de cargando o error, por ejemplo.

```
* MainActivity es el punto de entrada principal de la aplicación Android.
* Está anotada con @AndroidEntryPoint para habilitar la inyección de dependencias usando Hilt.
*/
@AndroidEntryPoint
class MainActivity : ComponentActivity() {

    /**
     * ViewModel para gestionar los datos relacionados con la UI en el ciclo de vida de la actividad.
     */
    private val eventosViewModel: EventosViewModel by viewModels()

    /**
     * Llamado cuando la actividad se está iniciando. Aquí es donde debe ir la mayor parte de la inicialización.
     *
     * @param savedInstanceState Si la actividad está siendo re-inicializada después de haber sido
     * previamente cerrada, entonces este Bundle contiene los datos más
     * recientes proporcionados en {@link #onSaveInstanceState(Bundle)}.
     *
     * Nota: En caso contrario, es null.
     */
    @RequiresApi(Build.VERSION_CODES.O)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val navController = rememberNavController()
            val estado = eventosViewModel.uiState.collectAsState().value

            FitnessTrackerAppTheme {
                // Un contenedor de superficie usando el color de 'background' del tema
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    navegacion(navController)

                    when (estado) {
                        EventosUIState.Cargando -> { cargando() }
                        EventosUIState.Done -> { /* No se necesita acción */ }
                        is EventosUIState.Error -> {
                            MensajeError(texto = estado.texto) {
                                eventosViewModel.setState(EventosUIState.Done)
                            }
                        }
                    }
                }
            }
        }
    }
}
```

En la parte del viewModel de los eventos, tendrá una variable que contendrá un enumerado, el cual luego en la aplicación principal se encargará de poner el estado correspondiente

```


/**
 * ViewModel para gestionar el estado de la interfaz de usuario relacionado con los eventos.
 * Esta clase está anotada con @HiltViewModel para habilitar la inyección de dependencias usando Hilt.
 */
@HiltViewModel
class EventosViewModel @Inject constructor() : ViewModel() {

    /**
     * Estado mutable de la interfaz de usuario, inicializado con el estado Done.
     */
    private var _uiState = MutableStateFlow<EventosUIState>(EventosUIState.Done)

    /**
     * Estado inmutable de la interfaz de usuario expuesto para la observación.
     */
    val uiState: StateFlow<EventosUIState> get() = _uiState.asStateFlow()

    /**
     * Función para actualizar el estado de la interfaz de usuario.
     * Utiliza viewModelScope para lanzar una coroutine que actualiza el valor del estado.
     *
     * @param eventosUIState El nuevo estado a establecer.
     */
    fun setState(eventosUIState: EventosUIState) = viewModelScope.launch { this: CoroutineScope
        _uiState.value = eventosUIState
    }
}


```

```


/**
 * Clase sellada que representa los posibles estados de la interfaz de usuario relacionados con los eventos.
 */
sealed class EventosUIState {

    /**
     * Estado de error, contiene un mensaje de texto que describe el error.
     *
     * @property texto El mensaje de error.
     */
    data class Error(val texto: String) : EventosUIState()

    /**
     * Estado de carga, indica que se está cargando información.
     */
    data object Cargando : EventosUIState()

    /**
     * Estado finalizado, indica que no hay ninguna operación en curso.
     */
    data object Done : EventosUIState()
}


```

En el composable de la navegación, tendremos un scaffold que contendrá una topbar para toda la aplicación, aunque esta solo se mostrará si no estamos en el login. Luego dentro de este scaffold, se contendrán las distintas pantallas de la aplicación. Para controlar esto,

utilizaremos un listener que nos dirá en todo momento donde estamos y así mostrar lo correcto. Aparte, tendremos una clase sellada donde especificaremos todas las rutas de la aplicación.

```
/*
 * Clase sellada que representa las diferentes pantallas de la aplicación.
 *
 * @property route La ruta asociada a la pantalla.
 */
sealed class Pantallas(var route: String) {
    /** Pantalla de inicio de sesión */
    data object Login : Pantallas(route: "login")
    /** Pantalla de registro */
    data object Register : Pantallas(route: "register")
    /** Pantalla de menú */
    data object Menu : Pantallas(route: "menu")
    /** Pantalla de información */
    data object Info : Pantallas(route: "info")
    /** Pantalla del diario */
    data object Diario : Pantallas(route: "diario")
    /** Pantalla de búsqueda */
    data object Buscar : Pantallas(route: "buscar")
}

/**
 * Función de navegación que configura las pantallas y la visibilidad de la barra de herramientas.
 *
 * @param navController Controlador de navegación de Jetpack Compose.
 */
@RequiresApi(Build.VERSION_CODES.O)
@Composable
fun navegacion(navController: NavHostController) {
    val showToolbar = remember { mutableStateOf(value: false) }
    hideOrShowToolbar(navController = navController, showToolbar = showToolbar)

    if (showToolbar.value) {
        scaffoldPantallas(navHostController = navController)
    } else {
        navegacionPantallas(navController = navController)
    }
}
```

```
/*
 * Función para mostrar u ocultar la barra de herramientas según la pantalla actual.
 *
 * @param navController Controlador de navegación de Jetpack Compose.
 * @param showToolbar Estado mutable que indica si la barra de herramientas debe mostrarse.
 */
@RequiresApi(Build.VERSION_CODES.O)
@Composable
fun hideOrShowToolbar(
    navController: NavHostController,
    showToolbar: MutableState<Boolean>
) {
    // Función que se llama cada vez que cambia el destino de la navegación
    navController.addOnDestinationChangedListener(NavController.OnDestinationChangedListener { controller, destination, arguments ->
        // Si estamos en la pantalla de login o registro, no mostrar la barra de herramientas; en todas las demás pantallas, sí.
        when (controller.currentDestination?.route) {
            Pantallas.Login.route -> {
                showToolbar.value = false
            }
            Pantallas.Register.route -> {
                showToolbar.value = false
            }
            else -> {
                showToolbar.value = true
            }
        }
    })
}
```

Esta es la función de navegación, que se encarga de mandar el composable que se le ha indicado

```
/*
 * Función que configura la navegación entre las diferentes pantallas de la aplicación.
 *
 * @param navController Controlador de navegación de Jetpack Compose.
 */
@RequiresApi(Build.VERSION_CODES.O)
@Composable
fun navegacionPantallas(navController: NavHostController) {
    NavHost(navController = navController, startDestination = Pantallas.Login.route) { this: NavGraphBuilder
        // Configura las diferentes pantallas y sus rutas
        composable(Pantallas.Login.route) { this: AnimatedContentScope | it: NavBackStackEntry
            PantallaLogin(navController)
        }
        composable(Pantallas.Register.route) { this: AnimatedContentScope | it: NavBackStackEntry
            VentanaRegister(navController)
        }
        composable(Pantallas.Menu.route) { this: AnimatedContentScope | it: NavBackStackEntry
            pantallaPrincipal()
        }
        composable(Pantallas.Info.route) { this: AnimatedContentScope | it: NavBackStackEntry
            pantallaInformacion()
        }
        composable(Pantallas.Diario.route) { this: AnimatedContentScope | it: NavBackStackEntry
            PantallaInfoDiaria(navHostController = navController)
        }
        composable(
            route = "${Pantallas.Buscar.route}/{number}",
            arguments = listOf(navArgument(name = "number") { type = NavType.IntType })
        ) { this: AnimatedContentScope | it: NavBackStackEntry
            // Obtiene el argumento "number" de la ruta y muestra la pantalla de búsqueda
            val number = it.arguments?.getInt(key = "number")
            PantallaBuscar(numero = number ?: 0, navController)
        }
    }
}
```

Y la función del Scaffold:

```
/*
 * Función para configurar el scaffold de las pantallas.
 *
 * @param navHostController Controlador de navegación de Jetpack Compose.
 * @param informacionViewModel ViewModel para gestionar la información del usuario.
 */
@RequiresApi(Build.VERSION_CODES.O)
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun scaffoldPantallas(
    navHostController: NavHostController,
    informacionViewModel: InformacionViewModel = hiltViewModel()
) {
    val scrollBehavior = TopAppBarDefaults.pinnedScrollBehavior(rememberTopAppBarState())
    val nombre = informacionViewModel.username.collectAsState().value

    Scaffold(
        modifier = Modifier.nestedScroll(scrollBehavior.nestedScrollConnection),
        topBar = {
            CenterAlignedTopAppBar(
                colors = TopAppBarDefaults.topAppBarColors(
                    containerColor = azul1,
                    titleContentColor = Color.White,
                ),
                title = {
                    Text(
                        nombre,
                        maxLines = 1,
                        overflow = TextOverflow.Ellipsis
                    )
                },
                actions = { this: RowScope ->
                    IconButton(onClick = {}) {
                        Icon(
                            imageVector = Icons.Filled.Settings,
                            tint = Color.White,
                        )
                    }
                }
            )
        }
    )
}
```

```

        horizontalArrangement = Arrangement.SpaceAround,
        verticalAlignment = Alignment.CenterVertically
    ) { this: RowScope
        Column(
            modifier = Modifier
                .fillMaxHeight()
                .fillMaxWidth( fraction: 0.33f)
                .clickable { navHostController.navigate( route: "menu") },
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) { this: ColumnScope
            Icon(Icons.Rounded.Person, contentDescription = "", tint = Color.White)
        }

        Column(
            modifier = Modifier
                .fillMaxHeight()
                .fillMaxWidth( fraction: 0.5f)
                .clickable { navHostController.navigate( route: "info") },
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) { this: ColumnScope
            Icon(Icons.Rounded.Addchart, contentDescription = "", tint = Color.White)
        }

        Column(
            modifier = Modifier
                .fillMaxHeight()
                .fillMaxWidth()
                .clickable { navHostController.navigate( route: "diario") },
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) { this: ColumnScope
            Icon(Icons.Rounded.DateRange, contentDescription = "", tint = Color.White)
        }
    }
    // Configura la navegación entre las pantallas
    navegacionPantallas(navController = navHostController)
}
}

```

Para la base de datos local, hemos utilizado room, donde guardamos los datos del usuario, comidas, dietas y otros valores para tenerlos a mano o por si no hay internet y tenemos que registrar los datos. Para ello, se ha creado una clase donde se guardan las distintas clases que va a tener la BBDD.

```
* Clase de base de datos que define la estructura y la versión de la base de datos.
*
* @param entities Arreglo de clases de entidad que representan las tablas de la base de datos.
* @param version Número de versión de la base de datos. Debe incrementarse si se modifican las entidades.
* @param exportSchema Indica si el esquema de la base de datos debe ser exportado cuando se exporta el archivo APK.
*/
@Database(
    entities = [UsuarioInfo::class, Comida::class, Rutina::class, FechaDia::class],
    version = 10,
    exportSchema = false
)
abstract class BaseDatos : RoomDatabase() {
    /**
     * Método abstracto para obtener el DAO de usuarios.
     *
     * @return Objeto de acceso a datos (DAO) para la entidad de Usuario.
     */
    abstract fun userDao(): UsuarioDao

    /**
     * Método abstracto para obtener el DAO de fechas.
     *
     * @return Objeto de acceso a datos (DAO) para la entidad de Fecha.
     */
    abstract fun fechaDao(): FechaDao

    /**
     * Método abstracto para obtener el DAO de rutinas.
     *
     * @return Objeto de acceso a datos (DAO) para la entidad de Rutina.
     */
    abstract fun rutinaDao(): RutinaDao

    /**
     * Método abstracto para obtener el DAO de comidas.
     *
     * @return Objeto de acceso a datos (DAO) para la entidad de Comida.
     */
    abstract fun comidaDao(): ComidaDao
}
```

```

    /**
     * Clase que representa la entidad de información de usuario en la base de datos.
     *
     * @property email Correo electrónico del usuario (clave primaria).
     * @property nombreUsuario Nombre de usuario del usuario.
     * @property contrasena Contraseña del usuario.
     * @property nombre Nombre del usuario.
     * @property apellido Apellido del usuario.
     * @property segundoApellido Segundo apellido del usuario.
     * @property fechaNacimiento Fecha de nacimiento del usuario (en formato de cadena).
     * @property fechaRegistro Fecha de registro del usuario (en formato de cadena).
     * @property token Token de autenticación del usuario.
     * @property fechaUltimaModificación Fecha de última modificación del usuario (en formato de cadena).
     * @property altura Altura del usuario en metros.
     * @property peso Peso del usuario en kilogramos.
     * @property sexo Sexo del usuario (Hombre o Mujer).
     */
    @Entity(tableName = "USUARIO")
    data class UsuarioInfo(
        @PrimaryKey
        var email: String = "",
        var nombreUsuario: String = "",
        var contrasena: String = "",
        var nombre: String = "",
        var apellido: String = "",
        var segundoApellido: String = "",
        var fechaNacimiento: String = "",
        var fechaRegistro: String = "",
        var token: String = "",
        var fechaUltimaModificacion: String = "",
        var altura: Float = 0f,
        var peso: Float = 0f,
        var sexo: Sexo = Sexo.Hombre
    )

```

Para comunicar la aplicación con la base de datos, lo haremos a través de los viewModels, pero es necesario crear antes unos daos que nos proporcionen las funciones necesarias y repositorios para llamar a esas funciones. Esto hay que hacerlo para cada entidad o grupo de entidades relacionadas

```


    /**
     * Data Access Object (DAO) para la entidad Comida en la base de datos.
     */
    @Dao
    interface ComidaDao {

        /**
         * Inserta una nueva comida en la base de datos.
         *
         * @param comida La comida a insertar.
         */
        @Insert(onConflict = OnConflictStrategy.IGNORE)
        fun insertar(comida: Comida)

        /**
         * Obtiene todas las comidas para una fecha específica.
         *
         * @param fecha La fecha en formato de cadena (yyyy-MM-dd) para la cual se desean obtener las comidas.
         * @return Una lista de comidas para la fecha especificada.
         */
        @Query("SELECT * FROM Comidas WHERE Date(horaConsumo) = Date(:fecha)")
        fun getComidasDia(fecha: String): List<Comida>

        /**
         * Verifica si una comida específica existe para una fecha dada.
         *
         * @param id El ID de la comida a verificar.
         * @param fecha La fecha en formato de cadena (yyyy-MM-dd) para la cual se desea verificar la existencia de la comida.
         * @return La comida correspondiente al ID y fecha especificados, si existe; de lo contrario, null.
         */
        @Query("SELECT * FROM Comidas WHERE comidaId = :id AND Date(horaConsumo) = Date(:fecha)")
        fun verificarComidas(id: String, fecha: String): Comida
    }


```

```


    /**
     * Clase repositorio para gestionar operaciones relacionadas con la entidad de Comida.
     *
     * @property comidaDao Objeto de acceso a datos (DAO) para la entidad de Comida.
     */
    class ComidaRepository @Inject constructor(private val comidaDao: ComidaDao) {

        /**
         * Inserta una nueva comida en la base de datos.
         *
         * @param comida La comida a insertar.
         */
        fun insertar(comida: Comida) = comidaDao.insertar(comida)

        /**
         * Obtiene todas las comidas para una fecha específica.
         *
         * @param fecha La fecha en formato de cadena (yyyy-MM-dd) para la cual se desean obtener las comidas.
         * @return Una lista de comidas para la fecha especificada.
         */
        fun getComidasDia(fecha: String) = comidaDao.getComidasDia(fecha)

        /**
         * Verifica si una comida específica existe para una fecha dada.
         *
         * @param id El ID de la comida a verificar.
         * @param fecha La fecha en formato de cadena (yyyy-MM-dd) para la cual se desea verificar la existencia de la comida.
         * @return La comida correspondiente al ID y fecha especificados, si existe; de lo contrario, null.
         */
        fun verificarComida(id: String, fecha: String) = comidaDao.verificarComidas(id, fecha)
    }


```

Para conectar la base de datos de forma que cuando se creen los daos se refieran a la misma base utilizando el patrón singleton al igual que el viewmodel general, se ha utilizado hilt y dagger, que es un sistema de inyección de dependencias para hacer esto más cómodo. Las clases son las siguientes.

```
 /**
 * Módulo de Dagger Hilt para proporcionar instancias de contexto de aplicación.
 */
@Module
@InstallIn(SingletonComponent::class)
class ApplicationModule {

    /**
     * Provee el contexto de la aplicación.
     *
     * @param application La instancia de la aplicación.
     * @return El contexto de la aplicación.
     */
    @Provides
    @Singleton
    fun provideContext(application: Application): Context {
        return application.applicationContext
    }
}
```

```
 /**
 * Clase de aplicación base que inicializa Hilt para la inyección de dependencias.
 */
@HiltAndroidApp
class AplicacionBase : Application() {}
```

```
 /**
 * Módulo de Dagger Hilt para proporcionar instancias de base de datos y DAOs.
 */
@Module
@InstallIn(SingletonComponent::class)
object DatabaseModule {

    /**
     * Provee la instancia de la base de datos.
     *
     * @param context El contexto de la aplicación.
     * @return La instancia de la base de datos.
     */
    @Provides
    @Singleton
    fun provideYourDatabase(
        context: Context
    ): BaseDatos {
        return synchronized( lock: this) {
            val instance = Room.databaseBuilder(
                context.applicationContext,
                BaseDatos::class.java,
                name: "baseDatos"
            ).fallbackToDestructiveMigration().build()
            instance ^synchronized
        }
    }
}
```

```

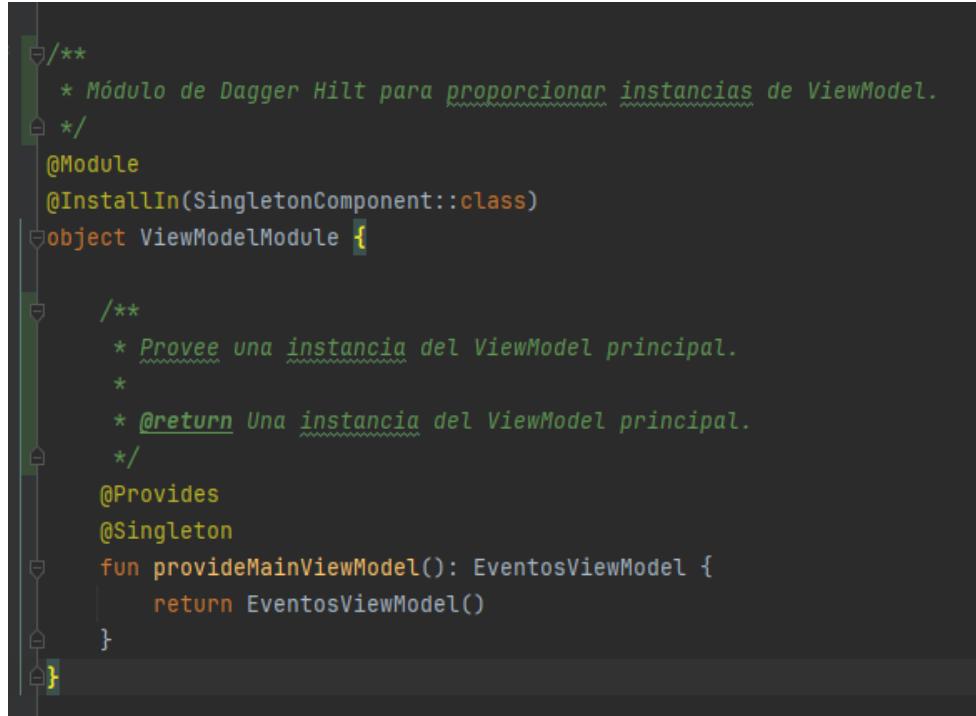
/**
 * Provee el DAO de usuario.
 *
 * @param db La instancia de la base de datos.
 * @return El DAO de usuario.
 */
@Provides
@Singleton
fun provideYourDao(db: BaseDatos): UsuarioDao = db.usuarioDao()

/**
 * Provee el DAO de fecha.
 *
 * @param db La instancia de la base de datos.
 * @return El DAO de fecha.
 */
@Provides
@Singleton
fun provideFechaDao(db: BaseDatos): FechaDao = db.fechaDao()

/**
 * Provee el DAO de comida.
 *
 * @param db La instancia de la base de datos.
 * @return El DAO de comida.
 */
@Provides
@Singleton
fun providesComidaDao(db: BaseDatos): ComidaDao = db.comidaDao()

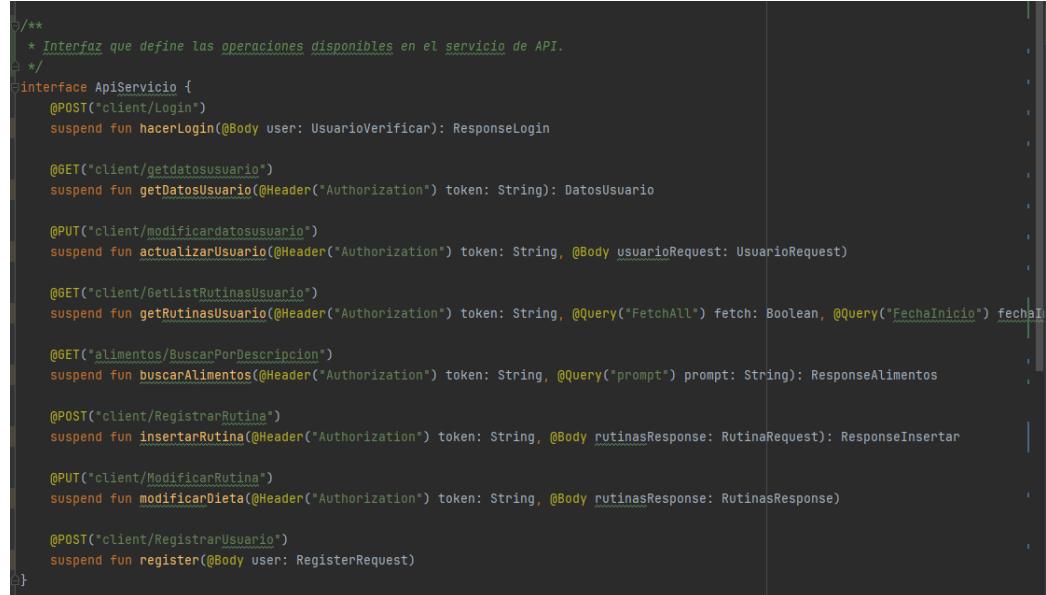
/**
 * Provee el DAO de rutina.
 *
 * @param db La instancia de la base de datos.
 * @return El DAO de rutina.
 */
@Provides
@Singleton
fun providesRutinaDao(db: BaseDatos): RutinaDao = db.rutinaDao()

```



```
/**  
 * Módulo de Dagger Hilt para proporcionar instancias de ViewModel.  
 */  
@Module  
@InstallIn(SingletonComponent::class)  
object ViewModelModule {  
  
    /**  
     * Provee una instancia del ViewModel principal.  
     *  
     * @return Una instancia del ViewModel principal.  
     */  
    @Provides  
    @Singleton  
    fun provideMainViewModel(): EventosViewModel {  
        return EventosViewModel()  
    }  
}
```

Para conectar nuestra aplicación con la Api, utilizaremos Retrofit para hacer las llamadas. Al igual que la base de datos, necesita de una clase donde se cree una instancia y luego una interfaz donde crearemos todas las llamadas. Luego, utilizaremos un repositorio para hacer las llamadas



```
/**  
 * Interfaz que define las operaciones disponibles en el servicio de API.  
 */  
interface ApiService {  
    @POST("client/Login")  
    suspend fun hacerLogin(@Body user: UsuarioVerificar): ResponseLogin  
  
    @GET("client/getdatosusuario")  
    suspend fun getDatosUsuario(@Header("Authorization") token: String): DatosUsuario  
  
    @PUT("client/modificardatosusuario")  
    suspend fun actualizarUsuario(@Header("Authorization") token: String, @Body usuarioRequest: UsuarioRequest)  
  
    @GET("client/GetListRutinasUsuario")  
    suspend fun getRutinasUsuario(@Header("Authorization") token: String, @Query("FetchAll") fetch: Boolean, @Query("FechaInicio") fechaInicio: String): ResponseRutinas  
  
    @GET("alimentos/BuscarPorDescripcion")  
    suspend fun buscarAlimentos(@Header("Authorization") token: String, @Query("prompt") prompt: String): ResponseAlimentos  
  
    @POST("client/RegistrarRutina")  
    suspend fun insertarRutina(@Header("Authorization") token: String, @Body rutinasResponse: RutinaRequest): ResponseInsertar  
  
    @PUT("client/ModificarRutina")  
    suspend fun modificarRuta(@Header("Authorization") token: String, @Body rutinasResponse: RutinasResponse)  
  
    @POST("client/RegistrarUsuario")  
    suspend fun register(@Body user: RegisterRequest)  
}
```

```

/**
 * Función para obtener una instancia de Retrofit para realizar llamadas a la API.
 *
 * @return Una instancia de Retrofit configurada para la API.
 */
@RequiresApi(Build.VERSION_CODES.O)
fun getRetrofitClient(): Retrofit {

    // Configuración para serializar/deserializar fechas y horas en formato LocalDateTime
    val gsonDateTime = GsonBuilder().registerTypeAdapter(
        LocalDateTime::class.java,
        JsonDeserializer { json, _, _ ->
            LocalDateTime.parse(
                json.asJsonPrimitive.asString,
                DateTimeFormatter.ofPattern(pattern: "yyyy-MM-dd'T'HH:mm:ss.SSSSSS")
            )
        }
    ).create()

    // Crear un interceptor para registrar las solicitudes y respuestas HTTP
    val loggingInterceptor = HttpLoggingInterceptor().apply { this.setLevel(HttpLoggingInterceptor.Level.BODY) }

    // Configurar OkHttpClient con el interceptor de registro
    val client = OkHttpClient.Builder()
        .addInterceptor(loggingInterceptor)
        .build()

    // Construir y devolver una instancia de Retrofit con la URL base y el convertidor Gson personalizado
    return Retrofit.Builder().baseUrl("http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/")
        .addConverterFactory(GsonConverterFactory.create(gsonDateTime))
        .client(client)
        .build()
}
//https://192.168.1.132:50598/api/fitnesstracker/

```

```

/**
 * Clase que actúa como repositorio para realizar llamadas a la API utilizando Retrofit.
 *
 * @param apiServicio Instancia de Retrofit que proporciona las operaciones de la API.
 */
@RequiresApi(Build.VERSION_CODES.O)
class RepositorioRetrofit(
    private val apiServicio: Retrofit = getRetrofitClient()
) {
    /**
     * Realiza la verificación de usuario mediante la API.
     *
     * @param user El objeto de usuario para la verificación.
     * @return La respuesta de la API para la verificación de usuario.
     */
    suspend fun verificar(user: UsuarioVerificar): ResponseLogin {
        return apiServicio.create(ApiServicio::class.java).hacerLogin(user)
    }

    /**
     * Obtiene los datos del usuario utilizando el token de autorización.
     *
     * @param token El token de autorización para la solicitud.
     * @return Los datos del usuario obtenidos de la API.
     */
    suspend fun getDatosUsuario(token: String): DatosUsuario {
        return apiServicio.create(ApiServicio::class.java).getDatosUsuario( token: "Bearer $token")
    }

    /**
     * Actualiza los datos del usuario utilizando el token de autorización.
     *
     * @param token El token de autorización para la solicitud.
     * @param usuarioRequest El objeto de solicitud para actualizar los datos del usuario.
     */
    suspend fun actualizarUsuario(token: String, usuarioRequest: UsuarioRequest) {
        return apiServicio.create(ApiServicio::class.java).actualizarUsuario( token: "Bearer $token", usuarioRequest)
    }
}

```

Lo único que queda ya son las pantallas componibles y su respectivo viewmodel. Para poner un ejemplo mostrare el composable del login con la lógica del viewModel por detrás, y como se utiliza los eventos, la base de datos room y retrofit.

```


) /**
 * Composable que representa la pantalla de inicio de sesión.
 *
 * @param navController Controlador de navegación para gestionar la navegación entre pantallas.
 * @param loginViewModel ViewModel para gestionar el estado y la lógica de la pantalla de inicio de sesión.
 */
@RequiresApi(Build.VERSION_CODES.O)
@OptIn(ExperimentalComposeUiApi::class)
@Composable
fun PantallaLogin(navController: NavHostController, loginViewModel: LoginViewModel = hiltViewModel()) {
    val brush = Brush.linearGradient(listOf(azul2, azul1))
    val image: Painter = painterResource(id = R.drawable.grupo)
    val fuente = FontFamily(Font(R.font.extraLight))

    var textoEmail = remember { mutableStateOf(value: "") }
    var textoPass = remember { mutableStateOf(value: "") }

    var firstTimeButton by remember { mutableStateOf(value: false) }
    val (focusRequester) = FocusRequester.createRefs()

    val usuarioInfo = loginViewModel.usuarioInfo.collectAsState().value

    LaunchedEffect(key1: true){ this: CoroutineScope
        loginViewModel.setUser()
    }
}


```

```


Column (modifier = Modifier
    .background(brush)
    .fillMaxSize(), horizontalAlignment = Alignment.CenterHorizontally){ this: ColumnScope

    Column (modifier = Modifier
        .fillMaxHeight(fraction: 0.4f)
        .fillMaxWidth()
        , horizontalAlignment = Alignment.CenterHorizontally, verticalArrangement = Arrangement.Center){ this: ColumnScope
        Image(painter = image, contentDescription = "Logo", modifier = Modifier.size(120.dp))
        Text(text = "Nombre", color = Color.White, fontSize = 45.sp, fontFamily = fuente)
    }

    Column (modifier = Modifier
        .fillMaxWidth(fraction: 0.8f)
        .fillMaxHeight(fraction: 0.6f)
        ,horizontalAlignment = Alignment.CenterHorizontally, verticalArrangement = Arrangement.Center){ this: ColumnScope
        BotonLogin(
            campo = textoEmail,
            firstTimeButton = firstTimeButton,
            labelTexto = "Email",
            textoError = "Escriba el email",
            accountCircle = Icons.Rounded.Email,
            focusRequester = focusRequester,
        )
    }
}


```

```

        Spacer(modifier = Modifier.padding(0.dp, 15.dp))

        BotonLogin(
            campo = textoPass,
            firstTimeButton = firstTimeButton,
            labelTexto = "Contraseña",
            textoError = "Escriba la contraseña",
            PasswordVisualTransformation(),
            KeyboardType.Password,
            accountCircle = Icons.Rounded.Lock,
            focusRequester = focusRequester
        )
    }

Column (modifier = Modifier.fillMaxSize()){ this: ColumnScope
    Column (modifier = Modifier.fillMaxWidth(),
        horizontalAlignment = Alignment.CenterHorizontally){ this: ColumnScope |
        ElevatedButton(
            onClick = {
                firstTimeButton = true
                hacerLlamada(textoEmail.value, textoPass.value, loginViewModel)
            }, modifier = Modifier
                .height(60.dp)
                .fillMaxWidth(fraction: 0.8f)
                .padding(15.dp, 0.dp),
            shape = RoundedCornerShape(40.dp, 0.dp, 40.dp, 0.dp),
            colors = ButtonDefaults.buttonColors(

```

```

        }
    }
    Column (modifier = Modifier.fillMaxSize()
        , horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Bottom){ this: ColumnScope
        Row (modifier = Modifier.fillMaxWidth(),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.Center){ this: RowScope
            Text(text = "¿No tienes cuenta?", modifier = Modifier.padding(0.dp, 10.dp)
                , fontSize = 17.sp, color = Color.White)
            Text(text = " Regístrate aquí", modifier = Modifier
                .padding(0.dp, 10.dp)
                .clickable { navController.navigate(route: "register") }
                , fontSize = 17.sp, color = Color.White, fontWeight = FontWeight.Black)
        }
    }
}

if(usuarioInfo.success){
    navController.navigate(route: "menu")
}

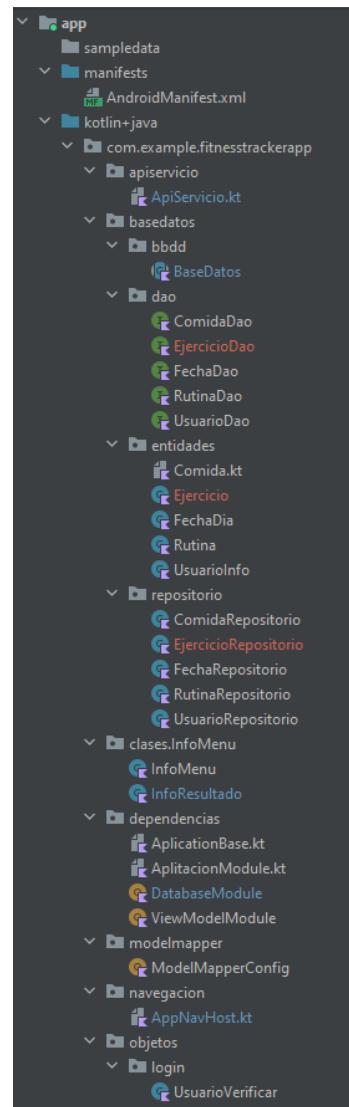
```

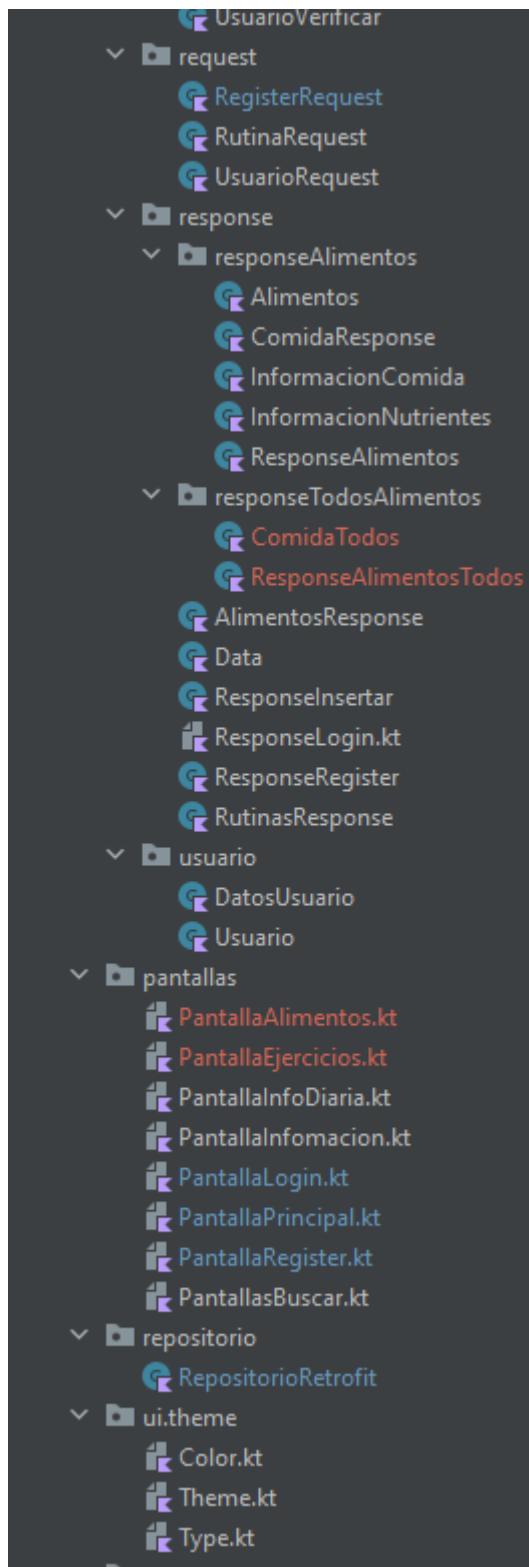
```

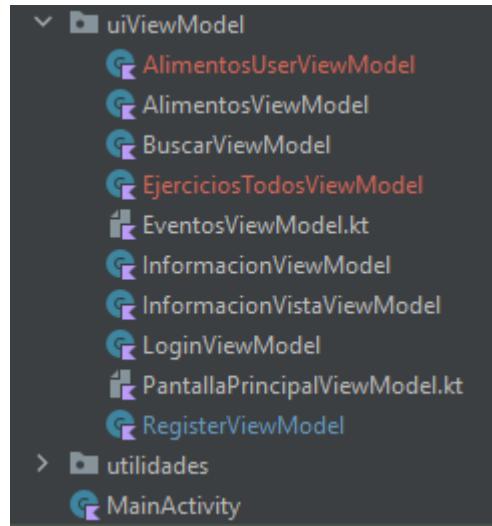
    /**
     * Realiza una llamada para iniciar sesión utilizando los datos proporcionados.
     *
     * @param textoEmail El correo electrónico del usuario.
     * @param textoPass La contraseña del usuario.
     * @param viewModelFitness El ViewModel para gestionar el estado y la lógica del inicio de sesión.
     */
    @RequiresApi(Build.VERSION_CODES.O)
    fun hacerLlamada(textoEmail: String, textoPass: String, viewModelFitness: LoginViewModel) {
        if (textoEmail.isNotEmpty() && textoPass.isNotEmpty())
            viewModelFitness.login(textoEmail, textoPass)
    }
}

```

Este sería el esquema de carpetas de la aplicación:







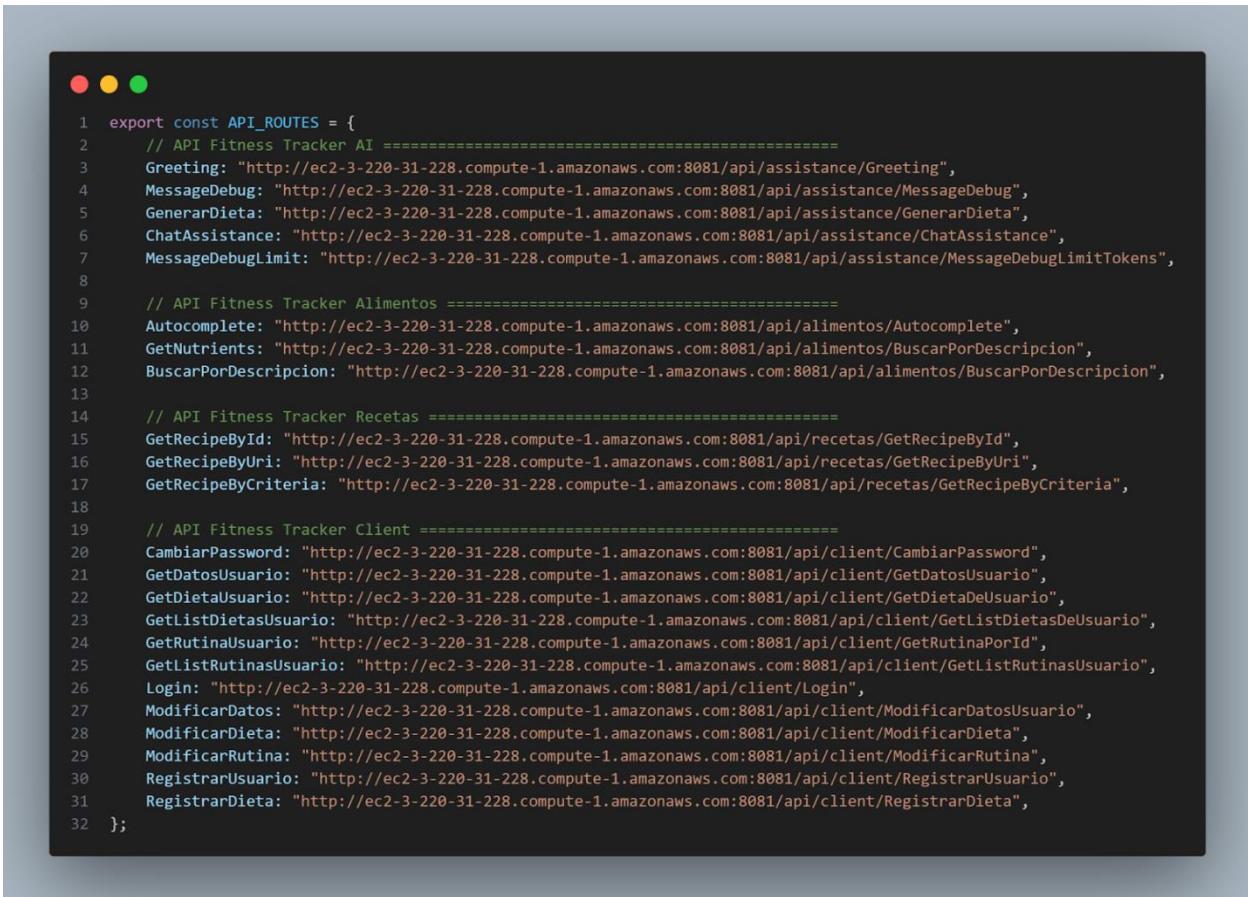
En el apartado web comenzamos definiendo las diferentes rutas que vamos a usar. En React se usa el fichero App.jsx para definir las diferentes rutas.

```

 1  export default function App() { // Definimos el componente App
 2    return (
 3      <AuthProvider> {/* Envolvemos la aplicación con AuthProvider para manejar el contexto de autenticación */}
 4        <BrowserRouter> {/* Utilizamos BrowserRouter para manejar las rutas */}
 5          <Routes>
 6            {/* Rutas Públicas */}
 7            <Route path="/" element={<InicioPage />} /> {/* Ruta para la página de inicio */}
 8            <Route path="/Inicio" element={<Navigate replace to="/" />}></Route> {/* Redirige /Inicio a / */}
 9            <Route path="/Login" element={<LoginPage />}></Route> {/* Ruta para la página de login */}
10            <Route path="/Register" element={<RegisterPage />}></Route> {/* Ruta para la página de registro */}
11
12            <Route element={<ProtectedRoutes />}> {/* Envolvemos las rutas protegidas dentro de ProtectedRoutes */}
13              {/* Usuario */}
14              <Route path="/Perfil" element={<PerfilPage />}></Route> {/* Ruta para la página de perfil */}
15              <Route path="/Editar" element={<EditarPerfilPage />}></Route> {/* Ruta para editar el perfil */}
16
17              {/* Dietas */}
18              <Route path="/GenerarDieta" element={<DietGeneratorPage />}></Route> {/* Ruta para generar dietas */}
19              <Route path="/ListadoDietas" element={<ListadoDietas />}></Route> {/* Ruta para listar dietas */}
20
21              {/* Rutinas */}
22              <Route path="/Today" element={<DailyPage />}></Route> {/* Ruta para la página diaria */}
23              <Route path="/ListadoRutinas" element={<ListadoRutinasPage />}></Route> {/* Ruta para listar rutinas */}
24
25              {/* Pendiente de borrar o work in progress */}
26              <Route path="/DailyCalorie" element={<DailyCalorie />}></Route> {/* Ruta para el cálculo diario de calorías */}
27              <Route path="/ChatAssistance" element={<ChatAssistancePage />}></Route> {/* Ruta para la asistencia por chat */}
28            </Route>
29          </Routes>
30        <BrowserRouter>
31        </AuthProvider>
32      )
33  }

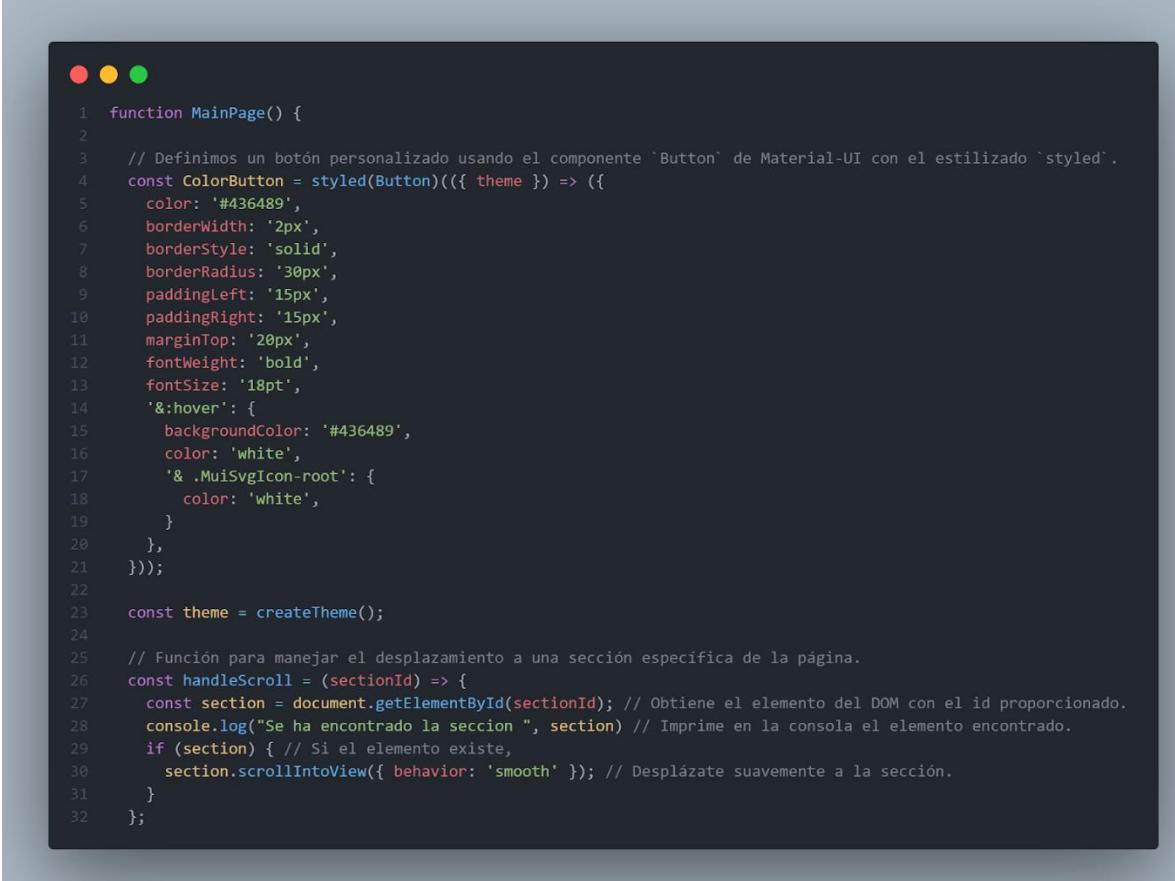
```

En el fichero ApiRoutes.jsx se definen las diferentes rutas de conexión con el servidor para las diferentes peticiones.



```
1 export const API_ROUTES = {
2     // API Fitness Tracker AI =====
3     Greeting: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/assistance/Greeting",
4     MessageDebug: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/assistance/MessageDebug",
5     GenerarDieta: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/assistance/GenerarDieta",
6     ChatAssistance: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/assistance/ChatAssistance",
7     MessageDebugLimit: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/assistance/MessageDebugLimitTokens",
8
9     // API Fitness Tracker Alimentos =====
10    Autocomplete: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/alimentos/Autocomplete",
11    GetNutrients: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/alimentos/BuscarPorDescripcion",
12    BuscarPorDescripcion: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/alimentos/BuscarPorDescripcion",
13
14    // API Fitness Tracker Recetas =====
15    GetRecipeById: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/recetas/GetRecipeById",
16    GetRecipeByUri: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/recetas/GetRecipeByUri",
17    GetRecipeByCriteria: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/recetas/GetRecipeByCriteria",
18
19    // API Fitness Tracker Client =====
20    CambiarPassword: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/client/CambiarPassword",
21    GetDatosUsuario: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/client/GetDatosUsuario",
22    GetDietaUsuario: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/client/GetDietaDeUsuario",
23    GetListDietasUsuario: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/client/GetListDietasDeUsuario",
24    GetRutinaUsuario: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/client/GetRutinaPorId",
25    GetListRutinasUsuario: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/client/GetListRutinasUsuario",
26    Login: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/client/Login",
27    ModificarDatos: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/client/ModificarDatosUsuario",
28    ModificarDieta: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/client/ModificarDieta",
29    ModificarRutina: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/client/ModificarRutina",
30    RegistrarUsuario: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/client/RegistrarUsuario",
31    RegistrarDieta: "http://ec2-3-220-31-228.compute-1.amazonaws.com:8081/api/client/RegistrarDieta",
32};
```

En la página de inicio establecemos una función para manejar el desplazamiento por las diferentes secciones. Además, se crea un componente personalizado de un Botón.



```
1  function MainPage() {
2
3    // Definimos un botón personalizado usando el componente `Button` de Material-UI con el estilizado `styled`.
4    const ColorButton = styled(Button)(({ theme }) => ({
5      color: '#436489',
6      borderWidth: '2px',
7      borderStyle: 'solid',
8      borderRadius: '30px',
9      paddingLeft: '15px',
10     paddingRight: '15px',
11     marginTop: '20px',
12     fontWeight: 'bold',
13     fontSize: '18pt',
14     '&:hover': {
15       backgroundColor: '#436489',
16       color: 'white',
17       '& .MuiSvgIcon-root': {
18         color: 'white',
19       }
20     },
21   }));
22
23   const theme = createTheme();
24
25   // Función para manejar el desplazamiento a una sección específica de la página.
26   const handleScroll = (sectionId) => {
27     const section = document.getElementById(sectionId); // Obtiene el elemento del DOM con el id proporcionado.
28     console.log("Se ha encontrado la sección ", section) // Imprime en la consola el elemento encontrado.
29     if (section) { // Si el elemento existe,
30       section.scrollIntoView({ behavior: 'smooth' }); // Desplázate suavemente a la sección.
31     }
32   };

```

En el login se hace uso de la función loginUser del autenticador para loguearse con el usuario.



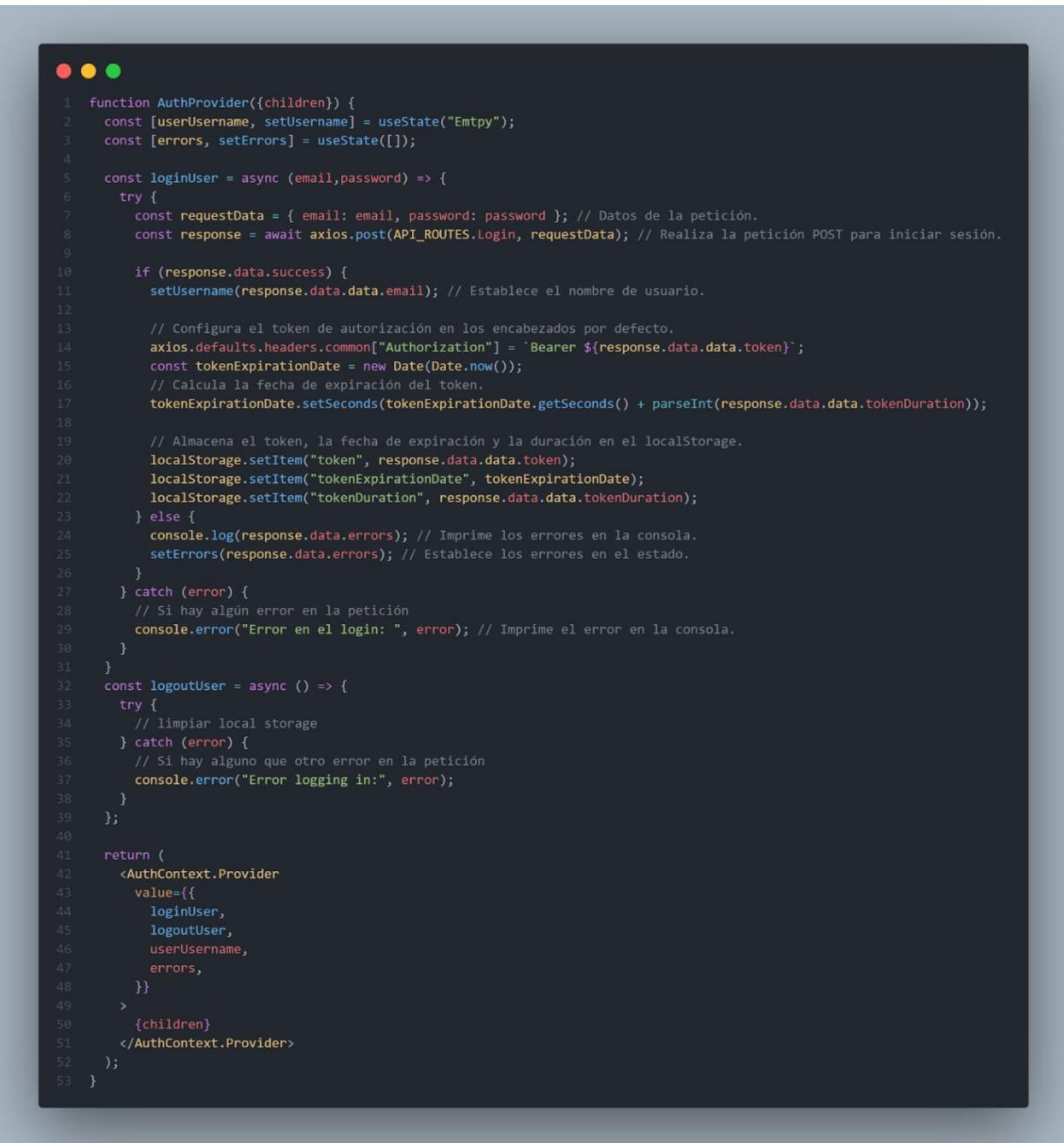
```
1 function LoginPage() {
2     // Pre setup
3     const navigate = useNavigate(); // Hook para la navegación.
4     const { loginUser } = useAuthContext(); // Obtiene la función `loginUser` del contexto de autenticación.
5
6     // State setup
7     const [email, setEmail] = useState(''); // Estado para el email del usuario.
8     const [password, setPassword] = useState(''); // Estado para la contraseña del usuario.
9     const token = localStorage.getItem("token"); // Obtiene el token del localStorage.
10
11    // Función para manejar el inicio de sesión.
12    const handleLogin = async () => {
13        await loginUser(email, password); // Llama a la función `loginUser` con el email y la contraseña.
14    };
15
16    // Efecto para verificar el token y redirigir si el usuario ya está autenticado.
17    useEffect(() => {
18        console.log("tokenls " + token); // Imprime el token en la consola.
19
20        // Verifica si el token no es nulo ni indefinido.
21        if (token != null || token != undefined) {
22            console.log("User is already logged");
23            navigate(PAGE_ROUTES.Today); // Redirige al usuario a la página de hoy.
24        }
25    }, [token]); // El efecto se ejecuta cada vez que el valor del token cambia.
```

En el Register se registra el usuario y si se ha podido registrar se navega a la página de login.



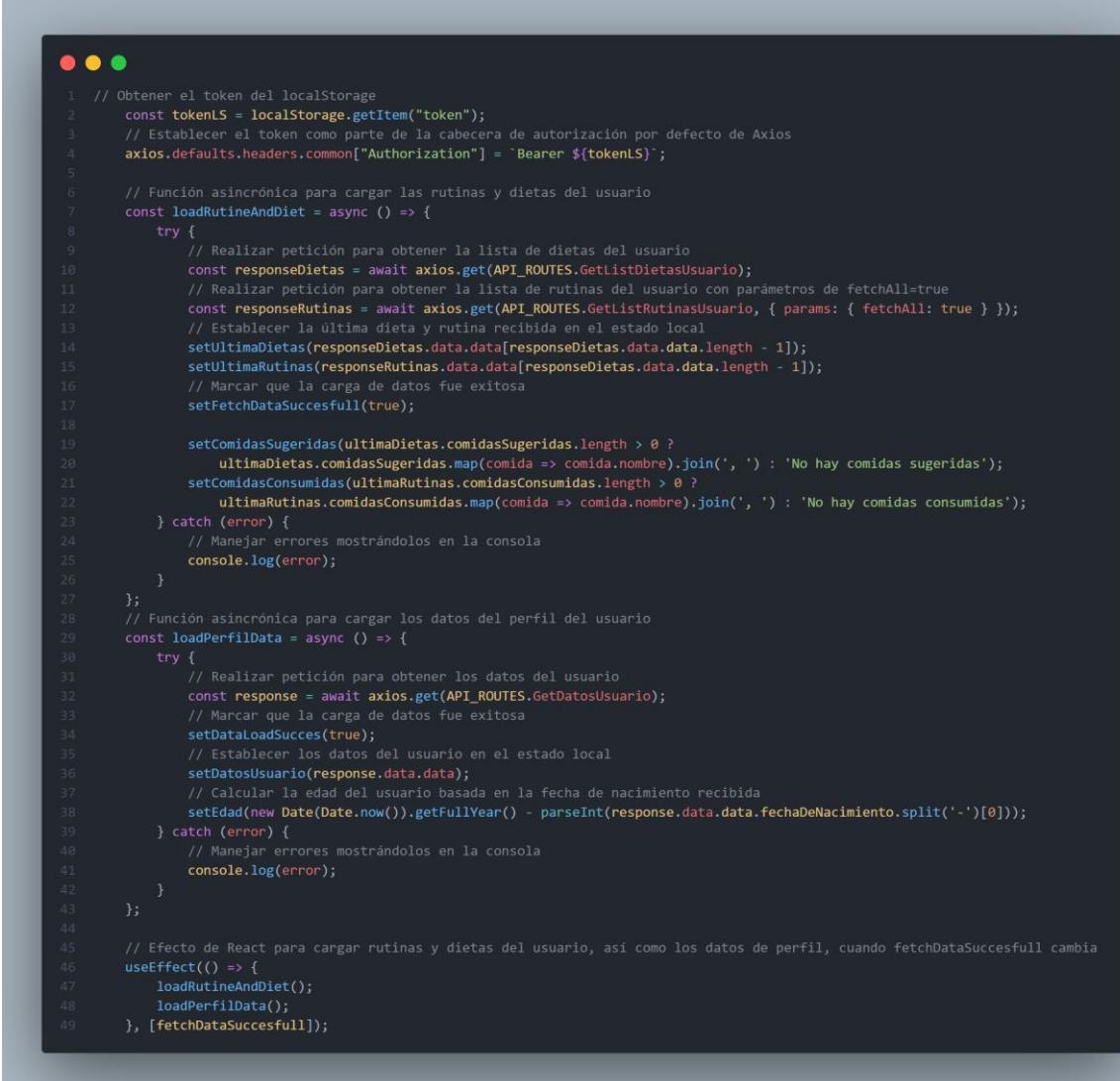
```
1 /**
2  * Función para manejar el registro del usuario.
3  * Envía una petición para registrar un nuevo usuario con los datos proporcionados.
4  */
5 const handleRegister = async () => {
6     try {
7         // Datos de la petición de registro.
8         const requestData = { email, username, password };
9
10        // Realiza la petición POST para registrar el usuario.
11        const response = await axios.post(API_ROUTES.RegistrarUsuario, requestData);
12
13        if (response.data.success) {
14            // Navega a la página de inicio de sesión si el registro es exitoso.
15            navigate(PAGE_ROUTES.Login);
16        } else {
17            // Imprime los errores en la consola si hubo algún problema.
18            console.log(response.data.errors);
19        }
20    } catch (error) {
21        // Imprime cualquier error en la consola si la petición falla.
22        console.error('Error al registrar el usuario: ', error);
23    }
24};
```

Este es el proveedor de autenticación.



```
1 function AuthProvider({children}) {
2   const [userUsername, setUsername] = useState("Empty");
3   const [errors, setErrors] = useState([]);
4
5   const loginUser = async (email,password) => {
6     try {
7       const requestData = { email: email, password: password }; // Datos de la petición.
8       const response = await axios.post(API_ROUTES.Login, requestData); // Realiza la petición POST para iniciar sesión.
9
10      if (response.data.success) {
11        setUsername(response.data.data.email); // Establece el nombre de usuario.
12
13        // Configura el token de autorización en los encabezados por defecto.
14        axios.defaults.headers.common["Authorization"] = `Bearer ${response.data.data.token}`;
15        const tokenExpirationDate = new Date(Date.now());
16        // Calcula la fecha de expiración del token.
17        tokenExpirationDate.setSeconds(tokenExpirationDate.getSeconds() + parseInt(response.data.data.tokenDuration));
18
19        // Almacena el token, la fecha de expiración y la duración en el localStorage.
20        localStorage.setItem("token", response.data.data.token);
21        localStorage.setItem("tokenExpirationDate", tokenExpirationDate);
22        localStorage.setItem("tokenDuration", response.data.data.tokenDuration);
23      } else {
24        console.log(response.data.errors); // Imprime los errores en la consola.
25        setErrors(response.data.errors); // Establece los errores en el estado.
26      }
27    } catch (error) {
28      // Si hay algún error en la petición
29      console.error("Error en el login: ", error); // Imprime el error en la consola.
30    }
31  }
32  const logoutUser = async () => {
33    try {
34      // limpiar local storage
35    } catch (error) {
36      // Si hay alguno que otro error en la petición
37      console.error("Error logging in:", error);
38    }
39  };
40
41  return (
42    <AuthContext.Provider
43      value={{
44        loginUser,
45        logoutUser,
46        userUsername,
47        errors,
48      }}
49    >
50      {children}
51    </AuthContext.Provider>
52  );
53}
```

En la página de inicio cargamos las dietas y el perfil para mostrar los diferentes datos del usuario y los diferentes datos de las rutinas y las dietas



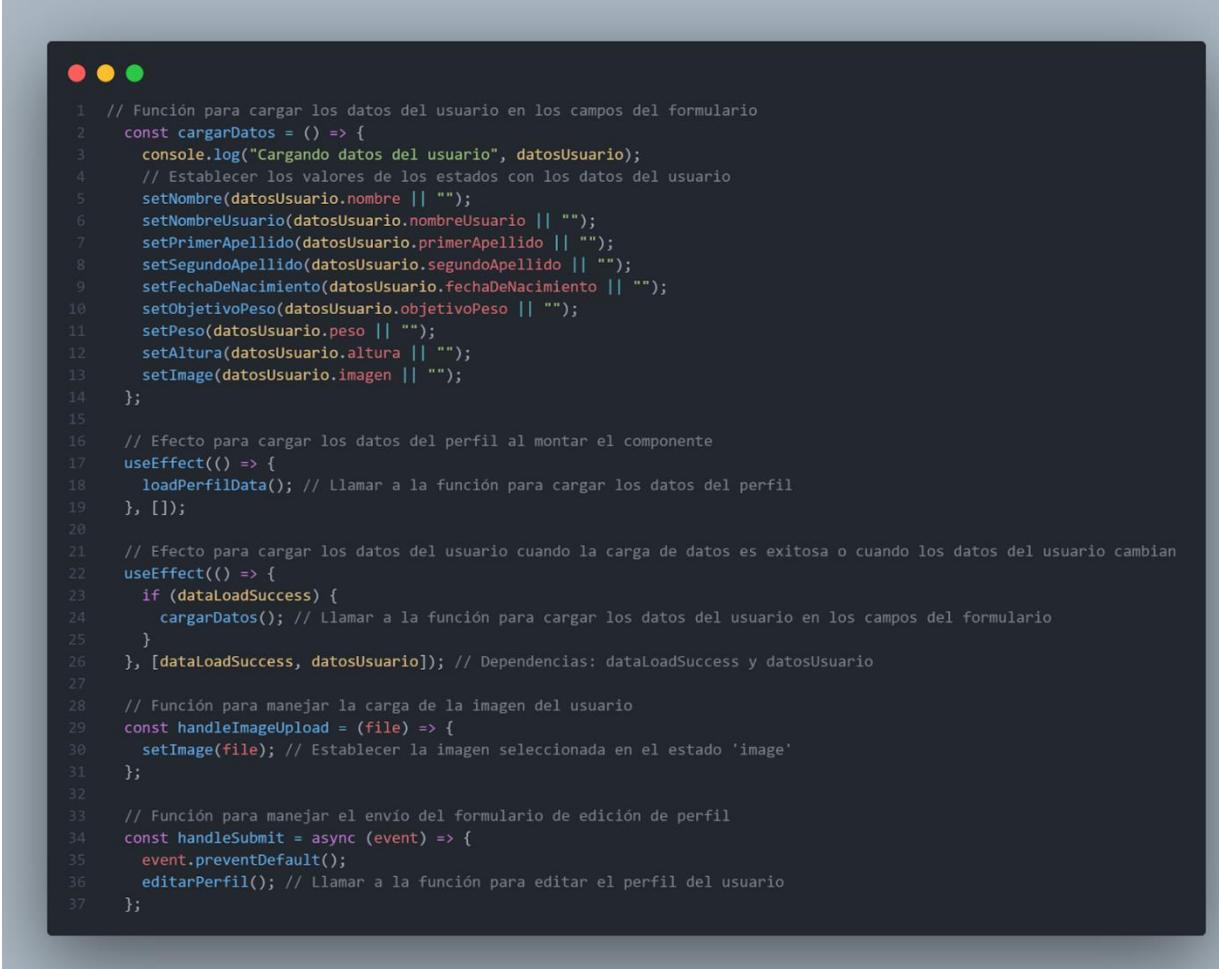
```
1 // Obtener el token del localStorage
2 const tokenLS = localStorage.getItem("token");
3 // Establecer el token como parte de la cabecera de autorización por defecto de Axios
4 axios.defaults.headers.common["Authorization"] = `Bearer ${tokenLS}`;
5
6 // Función asincrónica para cargar las rutinas y dietas del usuario
7 const loadRoutineAndDiet = async () => {
8     try {
9         // Realizar petición para obtener la lista de dietas del usuario
10        const responseDietas = await axios.get(API_ROUTES.GetListDietaUsuario);
11        // Realizar petición para obtener la lista de rutinas del usuario con parámetros de fetchAll=true
12        const responseRutinas = await axios.get(API_ROUTES.GetListRutinasUsuario, { params: { fetchAll: true } });
13        // Establecer la última dieta y rutina recibida en el estado local
14        setUltimaDieta(responseDietas.data.data[responseDietas.data.data.length - 1]);
15        setUltimaRutina(responseRutinas.data.data[responseRutinas.data.data.length - 1]);
16        // Marcar que la carga de datos fue exitosa
17        setFetchDataSuccessfull(true);
18
19        setComidasSugeridas(ultimaDieta.comidasSugeridas.length > 0 ?
20            ultimaDieta.comidasSugeridas.map(comida => comida.nombre).join(', ') : 'No hay comidas sugeridas');
21        setComidasConsumidas(ultimaRutina.comidasConsumidas.length > 0 ?
22            ultimaRutina.comidasConsumidas.map(comida => comida.nombre).join(', ') : 'No hay comidas consumidas');
23    } catch (error) {
24        // Manejar errores mostrándolos en la consola
25        console.log(error);
26    }
27};
28 // Función asincrónica para cargar los datos del perfil del usuario
29 const loadPerfilData = async () => {
30     try {
31         // Realizar petición para obtener los datos del usuario
32         const response = await axios.get(API_ROUTES.GetDatosUsuario);
33         // Marcar que la carga de datos fue exitosa
34         setDataLoadSuccess(true);
35         // Establecer los datos del usuario en el estado local
36         setDatosUsuario(response.data.data);
37         // Calcular la edad del usuario basada en la fecha de nacimiento recibida
38         setEdad(new Date(Date.now()).getFullYear() - parseInt(response.data.data.fechaDeNacimiento.split('-')[0]));
39     } catch (error) {
40         // Manejar errores mostrándolos en la consola
41         console.log(error);
42     }
43 };
44
45 // Efecto de React para cargar rutinas y dietas del usuario, así como los datos de perfil, cuando fetchDataSuccessfull cambia
46 useEffect(() => {
47     loadRoutineAndDiet();
48     loadPerfilData();
49 }, [fetchDataSuccessfull]);
```

En el generador de dietas tenemos una función que se encarga de hacer la petición post, también encontramos una función la cual maneja los cambios de los checkbox.



```
1 //Estado de los diferentes checkbox
2 const [state, setState] = React.useState({
3   vegetariano: false,
4   vegano: false,
5   gluten: false,
6   lacteos: false,
7   frutoSeco: false,
8   otros: false
9 });
10
11 // Función para manejar cambios en las opciones de los checkbox
12 const handleChange = (event) => {
13   setState({
14     ...state,
15     [event.target.name]: event.target.checked,
16   });
17 };
18
19 // Desestructurar opciones de dieta
20 const { vegetariano, vegano, gluten, lacteos, frutoSeco, otros } = state;
21
22 // Axios setup: Obtener y establecer el token de autorización
23 const token = localStorage.getItem("token");
24 axios.defaults.headers.common["Authorization"] = `Bearer ${token}`;
25
26 // Función asincrónica para generar la dieta
27 const generarDieta = async () => {
28   try {
29     // Obtener datos del usuario
30     const responseUserData = await axios.get(API_ROUTES.GetDatosUsuario);
31     const datosUsuario = responseUserData?.data;
32
33     // Construir objeto de datos para enviar en la solicitud POST
34     const requestData = {
35       fechaNacimiento: "2024-05-28T13:46:50.406Z",
36       sexo: genero, // Asumiendo que 'genero' está definido en otro lugar del código
37       altura: datosUsuario?.altura,
38       nivelActividadFisica: actividad,
39       objetivoPrincipal: objetivo,
40       habilidadCulinaria: habilidad,
41       comentariosAdicionales: notas,
42       fechaInicio: "2024-05-28T13:46:50.406Z",
43       fechaFin: "2024-05-28T13:46:50.407Z",
44     };
45     // Realizar solicitud POST para generar la dieta
46     const response = await axios.post(API_ROUTES.GenerarDieta, requestData);
47     // Marcar que la solicitud de datos fue exitosa
48     setDataFetchSuccess(true);
49   } catch (error) {
50     // Manejar errores mostrándolos en la consola
51     console.log(error);
52   }
53 };
54 // Función para manejar la presentación del formulario
55 const handleSubmit = async (event) => {
56   event.preventDefault();
57   generarDieta(); // Llamar a la función asincrónica para generar la dieta
58 };
```

En el apartado de edición de perfil hemos establecido que cargue los datos de usuario para llenar el formulario con los datos del usuario para que este pueda ver los que tiene y modificar aquellos que sean necesario.



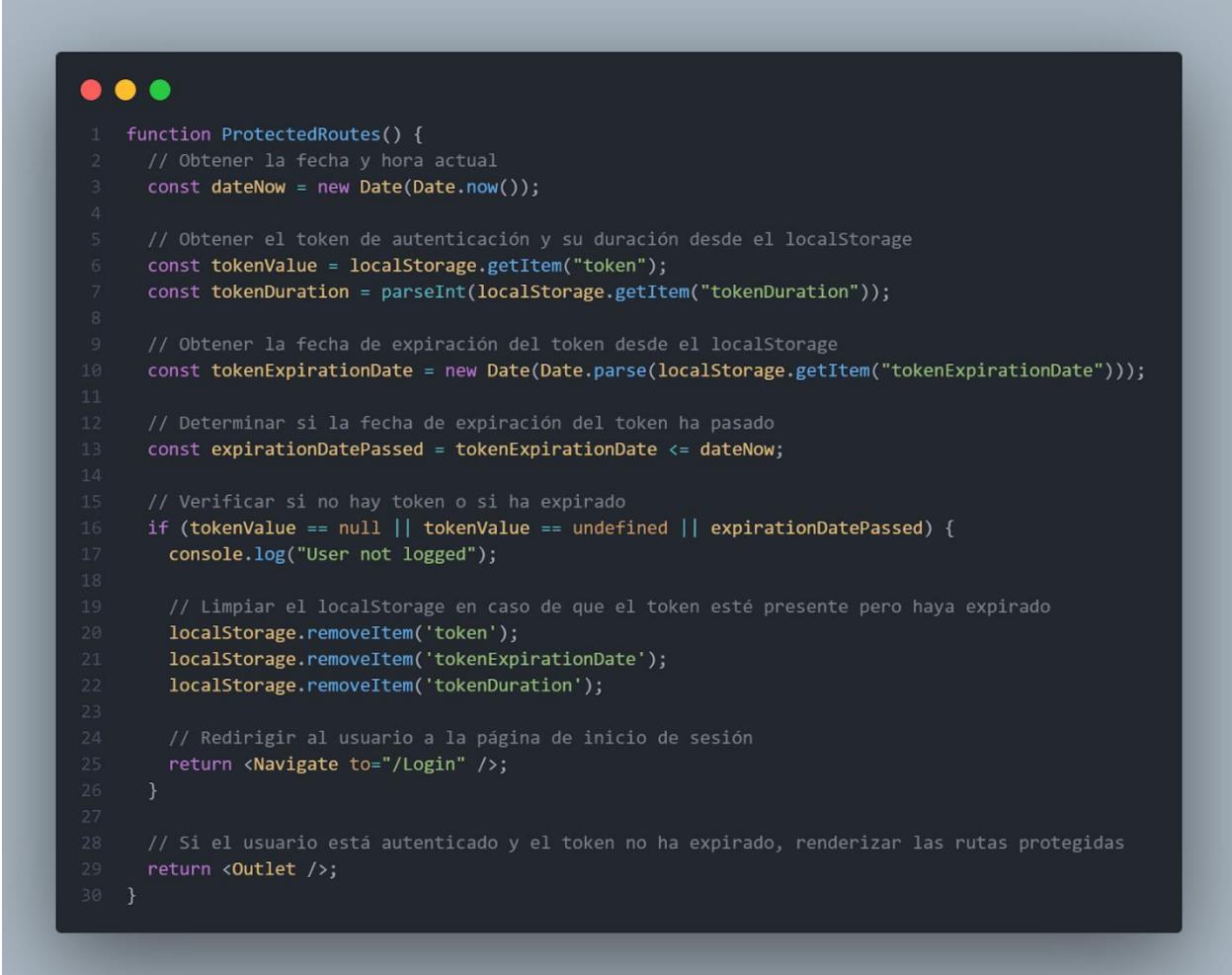
The screenshot shows a code editor window with a dark theme. At the top left are three colored window control buttons (red, yellow, green). The code itself is a React component with the following structure:

```
1 // Función para cargar los datos del usuario en los campos del formulario
2 const cargarDatos = () => {
3   console.log("Cargando datos del usuario", datosUsuario);
4   // Establecer los valores de los estados con los datos del usuario
5   setNombre(datosUsuario.nombre || "");
6   setNombreUsuario(datosUsuario.nombreUsuario || "");
7   setPrimerApellido(datosUsuario.primerApellido || "");
8   setSegundoApellido(datosUsuario.segundoApellido || "");
9   setFechaDeNacimiento(datosUsuario.fechaDeNacimiento || "");
10 setObjetivoPeso(datosUsuario.objetivoPeso || "");
11 setPeso(datosUsuario.peso || "");
12 setAltura(datosUsuario.altura || "");
13 setImage(datosUsuario.imagen || "");
14 };
15
16 // Efecto para cargar los datos del perfil al montar el componente
17 useEffect(() => {
18   loadPerfilData(); // Llamar a la función para cargar los datos del perfil
19 }, []);
20
21 // Efecto para cargar los datos del usuario cuando la carga de datos es exitosa o cuando los datos del usuario cambian
22 useEffect(() => {
23   if (dataLoadSuccess) {
24     cargarDatos(); // Llamar a la función para cargar los datos del usuario en los campos del formulario
25   }
26 }, [dataLoadSuccess, datosUsuario]); // Dependencias: dataLoadSuccess y datosUsuario
27
28 // Función para manejar la carga de la imagen del usuario
29 const handleImageUpload = (file) => {
30   setImage(file); // Establecer la imagen seleccionada en el estado 'image'
31 };
32
33 // Función para manejar el envío del formulario de edición de perfil
34 const handleSubmit = async (event) => {
35   event.preventDefault();
36   editarPerfil(); // Llamar a la función para editar el perfil del usuario
37 };
```

En el apartado de edición de los datos de usuario introducimos que se pueda modificar la imagen de perfil. Esta se añade a través del siguiente componente.

```
1 // Componente funcional ImageUploader
2 const ImageUploader = ({ onImageUpload }) => {
3     // Estados para manejar la imagen y la URL de vista previa de la imagen
4     const [image, setImage] = useState(null);
5     const [imagePreviewUrl, setImagePreviewUrl] = useState('');
6
7     // Función para manejar el cambio de la imagen seleccionada
8     const handleImageChange = (event) => {
9         const file = event.target.files[0]; // Obtener el archivo seleccionado desde el evento
10
11        // Verificar si se seleccionó un archivo de tipo imagen y que su tamaño sea menor a 10 MB
12        if (file && file.type.startsWith('image/') && file.size <= 10 * 1024 * 1024) {
13            setImage(file); // Establecer el archivo en el estado 'image'
14
15            const reader = new FileReader(); // Crear un lector de archivos
16            reader.onloadend = () => {
17                const base64String = reader.result; // Obtener el resultado como cadena base64
18                setImagePreviewUrl(base64String); // Establecer la URL de vista previa de la imagen
19                onImageUpload(base64String); // Llamar a la función 'onImageUpload' con la cadena base64
20            };
21            reader.readAsDataURL(file); // Leer el archivo como una URL de datos
22            // Llamar a la función 'onImageUpload' con el archivo completo (opcional)
23            // onImageUpload(file);
24
25        } else {
26            // Si el archivo no cumple con las condiciones de tipo imagen o tamaño menor a 10 MB
27            if (!file) {
28                alert('Por favor, seleccione un archivo de imagen.'); // Alerta si no se selecciona ningún archivo
29            } else if (file.size > 10 * 1024 * 1024) {
30                alert('El tamaño del archivo debe ser menor a 10 MB.'); // Alerta si el archivo excede los 10 MB
31            } else {
32                alert('Por favor, seleccione un archivo de imagen válido.'); // Alerta si no es un archivo de imagen válido
33            }
34        }
35    };
36    // Renderización del componente
37    return (
38        <div>
39            {/* Input oculto para seleccionar un archivo */}
40            <input
41                accept="image/*" // Aceptar solo archivos de tipo imagen
42                style={{ display: 'none' }} // Ocultar el input
43                id="raised-button-file" // ID del input
44                type="file" // Tipo de input
45                onChange={handleImageChange} // Manejar el cambio de la imagen seleccionada
46            />
47            {/* Etiqueta para el input de archivo */}
48            <label htmlFor="raised-button-file">
49                /* Botón para cargar la imagen */
50                <Button variant="contained" component="span">
51                    Cargar Imagen
52                </Button>
53            </label>
54            {/* Mostrar la vista previa de la imagen si existe */}
55            {imagePreviewUrl &&
56                <div>
57                    {/* Imagen con la vista previa */}
58                    <img src={imagePreviewUrl} alt="Preview" style={{ maxWidth: '100%', maxHeight: '300px' }} />
59                </div>
60            }
61        </div>
62    );
63};
```

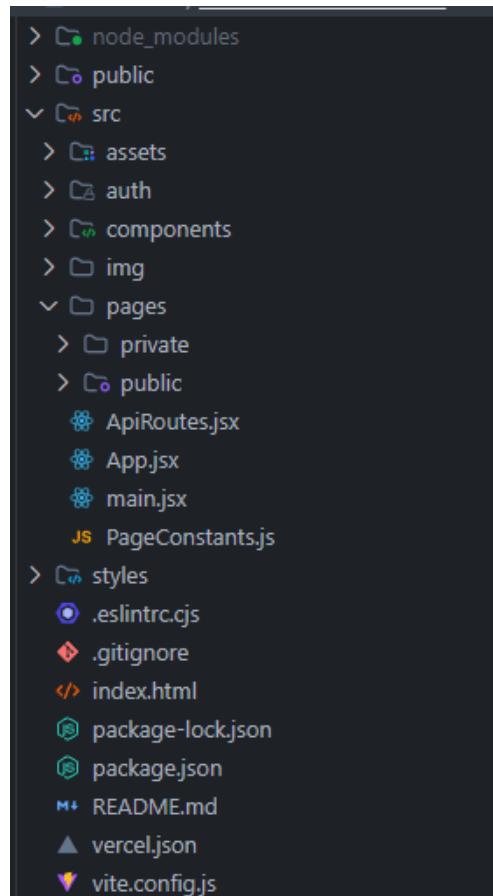
Este componente es usado para autenticar el usuario y que solo tenga acceso a las rutas de su interior en el caso de que esté logueado y tenga el token correspondiente. Además, contiene un sistema de expiración por tiempo. Este elimina los diferentes tokens. Si no se ha encontrado ningún token te redirige a el Login para que te logges.



```
1 function ProtectedRoutes() {
2     // Obtener la fecha y hora actual
3     const dateNow = new Date(Date.now());
4
5     // Obtener el token de autenticación y su duración desde el localStorage
6     const tokenValue = localStorage.getItem("token");
7     const tokenDuration = parseInt(localStorage.getItem("tokenDuration"));
8
9     // Obtener la fecha de expiración del token desde el localStorage
10    const tokenExpirationDate = new Date(Date.parse(localStorage.getItem("tokenExpirationDate")));
11
12    // Determinar si la fecha de expiración del token ha pasado
13    const expirationDatePassed = tokenExpirationDate <= dateNow;
14
15    // Verificar si no hay token o si ha expirado
16    if (tokenValue == null || tokenValue == undefined || expirationDatePassed) {
17        console.log("User not logged");
18
19        // Limpiar el localStorage en caso de que el token esté presente pero haya expirado
20        localStorage.removeItem('token');
21        localStorage.removeItem('tokenExpirationDate');
22        localStorage.removeItem('tokenDuration');
23
24        // Redirigir al usuario a la página de inicio de sesión
25        return <Navigate to="/Login" />;
26    }
27
28    // Si el usuario está autenticado y el token no ha expirado, renderizar las rutas protegidas
29    return <Outlet />;
30 }
```

La estructura de carpetas en el apartado Web se sigue la siguiente estructura de carpetas donde se pueden encontrar:

- Assets: Carpeta proveniente de la creación del proyecto con React.
- Auth: Carpeta que contiene los ficheros de comprobaciones de autenticación y seguridad.
- Componentes: En esta carpeta se guardan los diferentes componentes creados que son necesarios y se usan en el proyecto.
- Img: En esta carpeta se guardan todas las imágenes que son necesarias para la web.
- Pages: Es la estructura de ficheros principal y se divide en dos
 - Public: Son todas aquellas páginas a las que se puede llegar sin estar autenticado.
 - Private: Son todas aquellas páginas en las que se necesita estar autenticado para poder tener acceso.
- Styles: Carpeta proveniente de la creación del proyecto con React.

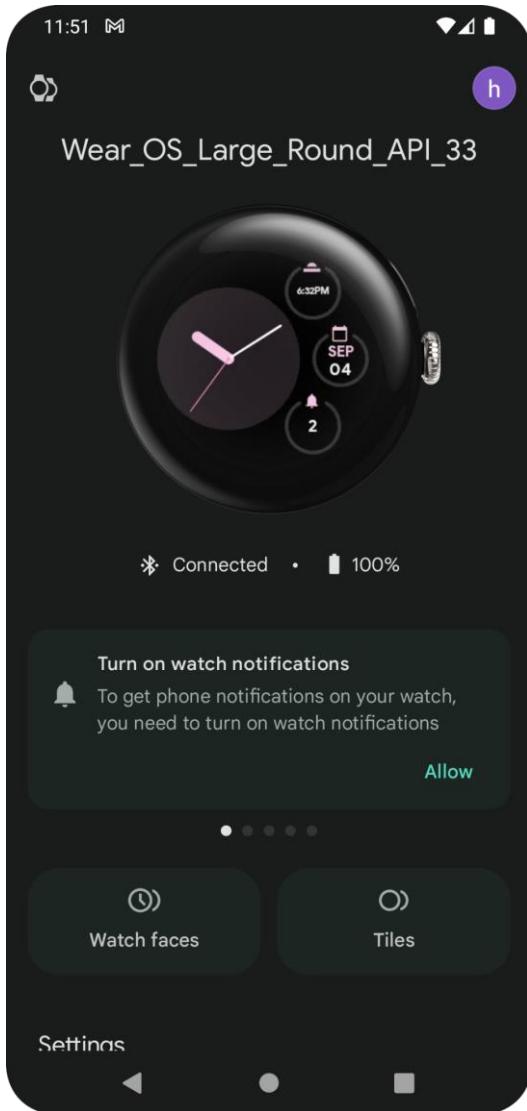


Manual de configuración y funcionamiento de la aplicación.

Aplicación Android

La aplicación móvil Fitness Tracker es una herramienta diseñada para ayudar a los usuarios a monitorear y mejorar su estado físico. Esta guía proporciona instrucciones detalladas sobre cómo instalar, configurar y utilizar la aplicación para aprovechar al máximo sus funciones.

Antes de instalar la aplicación, asegúrese de que su dispositivo cumpla con los siguientes requisitos: sistema operativo Android 8.0 o superior, al menos 87 MB de espacio libre en el almacenamiento y una conexión a Internet activa para sincronizar datos y acceder a funcionalidades en línea. Es importante también tener el móvil sincronizado con el Wear OS, esto se realiza mediante conexión Bluetooth y debemos instalarnos para esto primero la aplicación de Watch, para ello vamos a la Play Store y la instalamos. Emparejados los dispositivos nos saldría una pantalla como la siguiente:



Para instalar la aplicación en un dispositivo Android, siga estos pasos. De forma preliminar se permitirá la descarga de los instaladores mediante un link a GitHub. Bajados los instaladores, podrá hacer clic sobre ellos y darle al botón de “Instalar”. Finalizada la instalación será cuestión de iniciar la aplicación tanto en el dispositivo Android como en el dispositivo del reloj inteligente.

Una vez instalada, abra la aplicación desde el menú de aplicaciones. Si es un nuevo usuario, seleccione "Registrarse" y complete el formulario con su nombre, correo electrónico y una contraseña segura. Si ya tiene una cuenta, seleccione "Iniciar sesión" e ingrese sus credenciales.

La pantalla principal muestra un resumen de su actividad diaria, incluyendo pasos, calorías quemadas, frecuencia cardíaca, requerimiento de agua diario en litros, metabolismo basal y el IMC. Desde aquí puede acceder a diferentes secciones de la aplicación como Estadísticas, Dietas, Rutinas de Ejercicio y Cuenta.

En la sección de cuenta, puede actualizar su información personal como peso, sexo y altura.

Aplicación web

La aplicación web Fitness Tracker es una herramienta integral diseñada para ayudar a los usuarios a monitorear y mejorar su estado físico y salud desde cualquier dispositivo con acceso a Internet. Este manual sirve de guía a través del proceso de instalación, configuración y uso de la aplicación web, asegurando que pueda aprovechar al máximo sus funcionalidades.

Asegúrese de que su navegador web esté actualizado. La aplicación es compatible con las versiones más recientes de Google Chrome, Mozilla Firefox y Microsoft Edge.

Abra su navegador preferido y visite la página oficial de la aplicación web Fitness Tracker en <https://fitness-tracker-opal.vercel.app/>.

Si es un nuevo usuario, haga clic en "Registrarse" y complete el formulario con su información personal. Si ya tiene una cuenta, haga clic en "Iniciar sesión" e ingrese sus credenciales.

Una vez iniciada la sesión, complete su perfil proporcionando información como peso, altura, edad en el apartado de perfil, este se puede modificar.

La aplicación web Fitness Tracker está diseñada para ser intuitiva y fácil de usar. Aquí se detalla el funcionamiento de sus principales características:

Al iniciar sesión, será dirigido al panel principal que muestra un resumen de su actividad diaria, información sobre su físico y la dieta que esté activa actualmente.

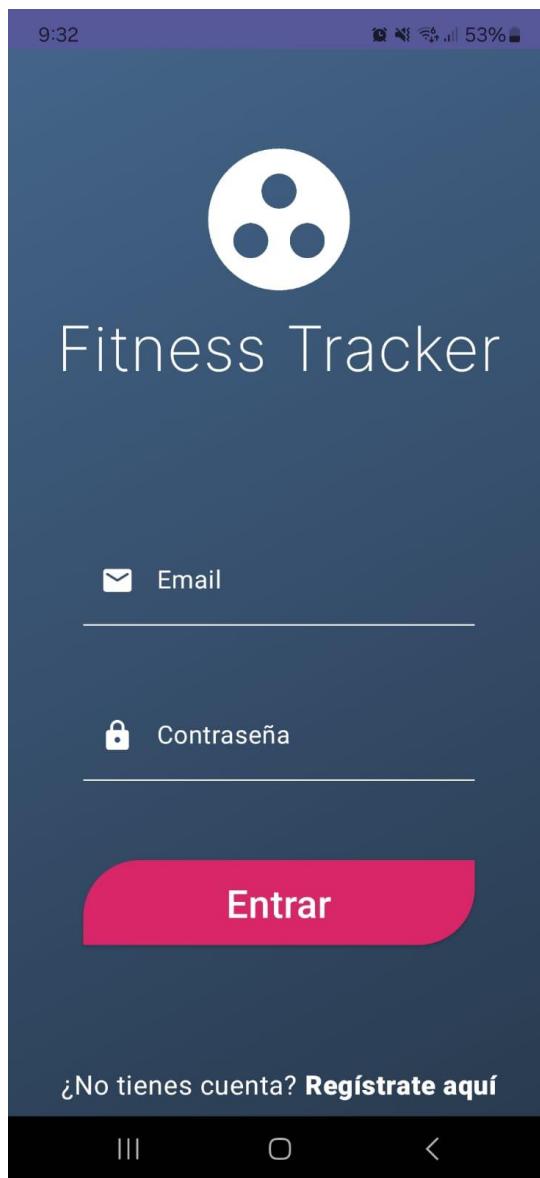
Registro de Actividades: En el menú lateral, seleccione "Actividades" para registrar nuevas actividades físicas. Complete los detalles como tipo de actividad, duración, distancia y calorías quemadas. Puede ver y editar sus registros en cualquier momento.

Manual de usuario

- Aplicación móvil

La aplicación móvil Fitness Tracker es una herramienta integral diseñada para ayudar a los usuarios a seguir y mejorar su estado físico y salud. Este manual de usuario sirve de guía a través de las diversas funcionalidades de la aplicación, desde la navegación básica hasta el registro y monitoreo de actividades físicas.

Nada más entrar en la aplicación, el usuario verá la pantalla de Login. En esta, podrá meter sus credenciales y, si son correctas, podrá entrar en la aplicación. En el caso que no tenga, debajo de la pantalla podrá acceder al Register para meter las nuevas credenciales.



En la pantalla principal el usuario podrá ver y modificar los datos relacionados a su altura, peso y sexo, y además podrá ver los datos relacionados a los mismos, como el gasto de Kcal o el requerimiento de agua mínimo.

Aparte, el usuario puede ver los alimentos que ha ingerido a lo largo de su uso con la aplicación además de sus ejercicios.

Al logearse la aplicación, será recibido con la pantalla principal que muestra un resumen de su información. En la parte superior de la pantalla, encontrará una barra de navegación con accesos directos a las principales secciones de la aplicación, información del perfil, seguimiento físico diario y el seguimiento distribuido por fechas.



Aparte, en cada fila de información de usuario, este mismo puede pulsar sobre él y cambiar el valor.



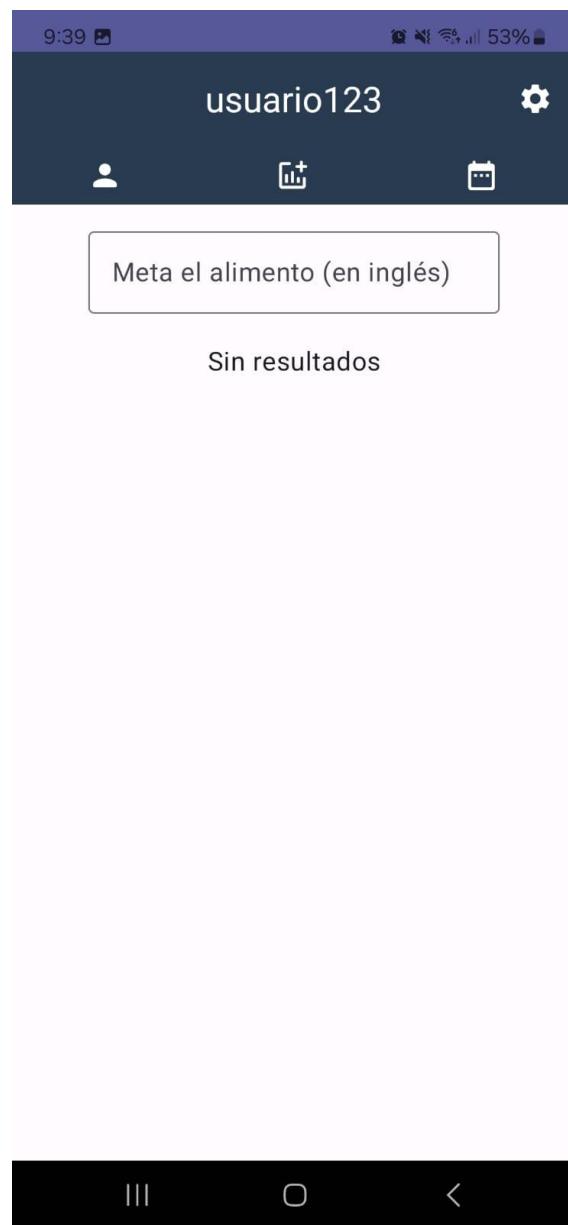
En la segunda pantalla verá toda la información mas detallada, en este no solo verá el gasto en Kcal que tiene que hacer, sino que también verá los gramos de proteínas, grasas y carbohidratos que tiene que consumir a lo largo del día.



En la última pantalla se verán los todos los alimentos que ha consumido el usuario en el día, y puede ver tanto sus ejercicios como el agua consumida. Debajo de cada comida, el usuario tiene un botón para añadir un alimento nuevo. En la parte de arriba, tiene dos flechas para navegar entre los distintos días y ver como le han ido los demás días. En la parte del agua también tiene dos flechas para aumentar o disminuir el agua en 250ml.



Cuando el usuario le de al más para añadir otro elemento, se le abrirá otra pantalla con un textfield donde podrá meter el alimento en cuestión. Cuando el campo esté relleno, podrá darle a enviar.



Una vez enviado, le mostrará todas las opciones que hay en la base de datos, y se le mostrará el nombre, datos y descripción del alimento. El usuario deberá pulsar en uno de ellos para marcarlo como alimento consumido.



- Aplicación web

La aplicación web Fitness Tracker es una plataforma completa que le permite llevar un registro detallado de su actividad física, dieta y progreso de fitness. Este manual de usuario le guiará a través de las diversas funcionalidades de la aplicación, asegurando una experiencia de usuario eficiente y efectiva. Nada más entrar en la web encontraremos el Inicio donde se podrán observar que posibilidades tiene la aplicación.

The screenshot shows the homepage of the Fitness-Tracker website. At the top, there is a navigation bar with links for Home, Nuestros Servicios, Sobre Nosotros, App, and a prominent 'REGISTER' button. To the left, there is a sidebar with a grid icon and a 'Virtualiza tu cuidado personal' section containing a subtext about personal health tracking and a 'CONSIGUE NUESTRA APP' button. The main content area features a central smartwatch icon surrounded by various health-related icons (lungs, pills, virus, heart, blood drop, eye, etc.). Below this, there is a 'Nuestros servicios' section.

En la barra de navegación podemos encontrar tres apartados diferentes los cuales son:

- Home: Que te dirige a la parte superior de la web.

This screenshot shows the 'Nuestros servicios' (Our Services) page of the website. It has a similar layout to the homepage, with a sidebar on the left and a central content area. The central area features a smartwatch icon surrounded by various health icons. Below this, there is a section titled 'Nuestros servicios'.

Le brindamos las mejores opciones para usted. Ajústalo a tus necesidades de salud y metas personales. Puedes consultar con nosotros qué tipo de ejercicios son mejores para alcanzar tus objetivos.

- Nuestros Servicios: Que te lleva a los diferentes servicios que te ofrece la aplicación

Nuestros servicios

Le brindamos las mejores opciones para usted. Ajústalo a tus necesidades de salud y metas personales Puedes consultar con nosotros qué tipo de ejercicios son mejores para alcanzar tus objetivos.


Busqueda Inteligente
 Encuentra y conoce todo aquello que necesites gracias a nuestra IA interactiva acerca de la salud y el fitness


Smartwatch
 Escoge cualquier smartwatch y comienza a usar la app para mejorar tu salud y alcanzar tus objetivos


Consulta Personalizada
 Consulta gratuitamente todo aquello que necesites para alcanzar tus objetivos con nuestra IA


Info Detallada
 Consulta la información acerca de tus entrenamientos y rendimiento diario


Cuenta
 Accede a tus registros desde cualquier lugar y dispositivo descargandote la app e iniciando sesión o desde nuestra web


Tracking
 Trackea tus entrenamientos, tu dia a dia ¡¡INCLUSO TUS PERIODOS DE SUEÑO!!

- Sobre nosotros: Que explica que somos y qué es lo que hacemos.



Lideres en salud y fitness con nuevas tecnologías

Le brindamos las mejores opciones para usted. Ajústalo a tus necesidades de salud y metas personales Puedes consultar con nosotros qué tipo de ejercicios son mejores para alcanzar tus objetivos.

- App: La cual tiene un botón desde donde se podrá descargar la aplicación móvil.

Descarga nuestra app para dispositivos móviles

Nuestra aplicación dedicada a la recolección de datos a través del software del smartwatch y soluciones para salud, nutrición y rutinas fitness para el beneficio personal del usuario. Comienza a alcanzar tus objetivos.

DESCARGAR

Fitness-Tracker Compañía Region

Home App Sobre Nosotros Nuestros Servicios

- Register: El cual te redirige a el apartado de creación de una cuenta donde nos encontraremos con 3 campos los cuales son el usuario, la contraseña, y el correo electrónico, los cuales debemos llenar para así crear una cuenta satisfactoriamente.

Fitness-Tracker

Usuario

Password

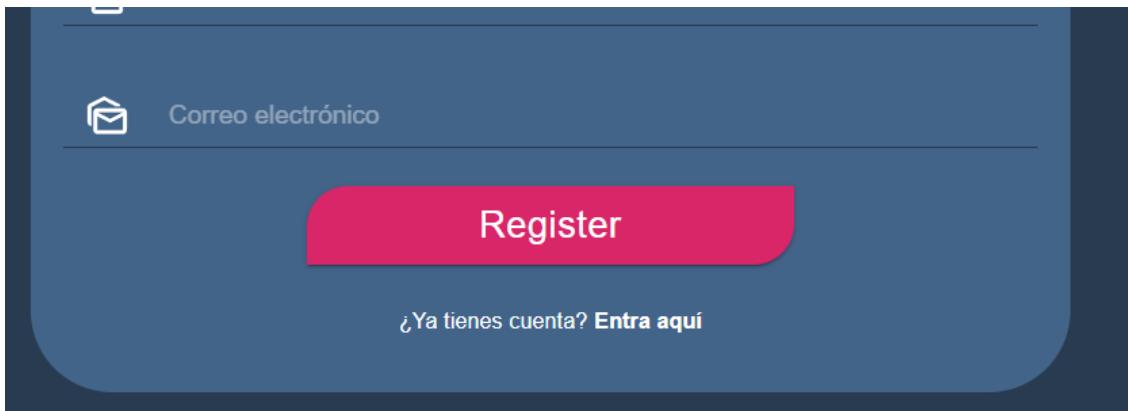
Correo electrónico

Register

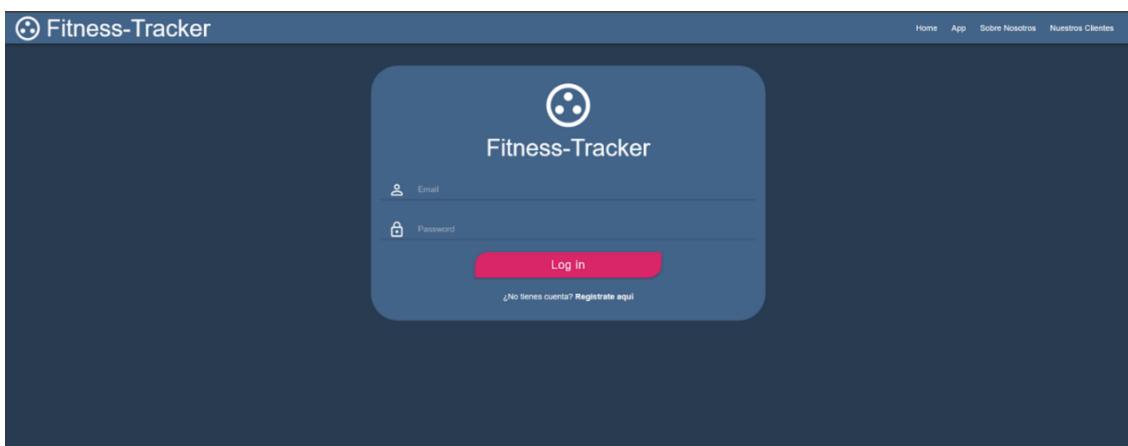
¿Ya tienes cuenta? Entrá aquí

Una vez nos hayamos registrado nos redirigirá a el login, donde podremos iniciar sesión con nuestra cuenta.

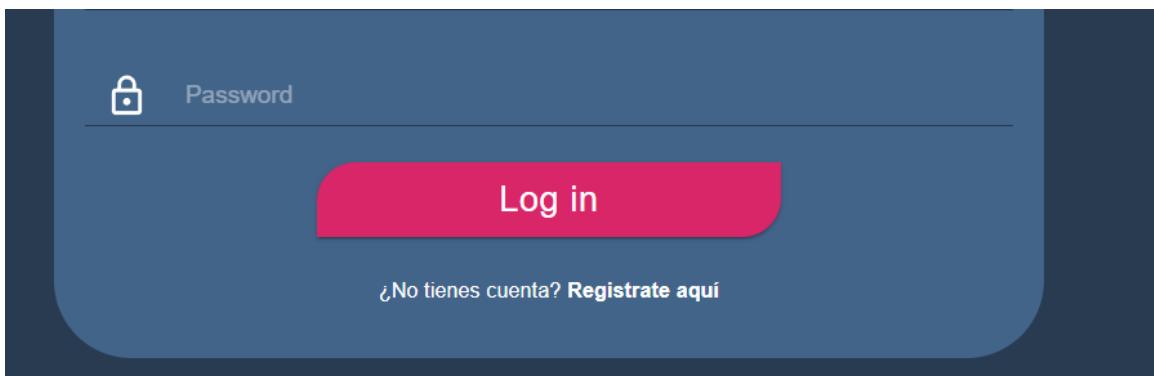
En el caso de que tengamos creada una cuenta, en el apartado inferior podremos encontrar un apartado para dirigirnos a login.



En el apartado del login encontramos 2 campos los cuales debemos llenar con su correo y contraseña para que así entren en su cuenta de Fitness-Tracker.



En el caso de que no se tenga cuenta de Fitness-Tracker en la parte de abajo se puede encontrar un botón para redirigir de nuevo al Register.



Al ingresar a la aplicación, se encontrará con el panel principal que proporciona un resumen rápido de su actividad diaria y detalles de su físico. La navegación principal se encuentra en el menú lateral izquierdo, desde donde puede acceder a las diferentes secciones de la aplicación:

The screenshot displays the Fitness-Tracker application interface. On the left, there is a sidebar menu with the following items:

- Overview**
- Generar dieta**
- Listado de dietas**
- Listado de rutinas**
- Información de cuenta**
- Cerrar sesión**

The main content area shows a user profile picture of a cartoon character wearing headphones and sunglasses, with the name "ElGuilleDEV" below it. To the right of the profile is a summary section titled "Rutina" (Routine) which includes:

| | |
|-------------------|-------|
| Tiempo de Sueño | 8 |
| Pasos Realizados | 10000 |
| Calorías Quemadas | 500 |

Below this is a "Dieta" (Diet) section:

| | |
|---------------------|-----------------------------|
| Calorías Requeridas | 2000 |
| Agua Requerida | 2.5 |
| Comidas Sugeridas | Ensalada César, Pollo Asado |

Finally, there is a "Físico" (Physical) section:

| | |
|---------------------|--|
| Frecuencia Cardíaca | 60 |
| Oxígeno Sangre | 98 |
| Comidas Consumidas | Ensalada César, Pollo Asado, Batido de Proteínas |

- **Overview:** Muestra una vista general de su actividad diaria y progreso.
- **Generar dieta:** Permite interactuar con el módulo de inteligencia artificial para registrar nuevas dietas rellenando un formulario.
- **Listado de dietas:** Redirige a la pantalla del listado de dietas del usuario junto con la dieta activa actualmente.
- **Listado de rutinas:** Redirige a la pantalla con el listado de rutinas del usuario.
- **Información de cuenta:** Ofrece información del perfil del usuario.
- **Cerrar sesión:** Permite cerrar la sesión del usuario.

Puede acceder a este sidebar a través de el botón de tres barras en la parte superior de la página:



De igual manera el botón de Inicio le redirige al Overview, el de Rutinas al Listado de las rutinas y el de Dietas al Listado de dietas.

En el apartado de “Generar Dietas” podemos encontrar lo siguiente. Esto es un formulario para generar una dieta a partir de tu edad, género, peso, altura, el nivel de actividad física que sueles hacer, cuál es tu objetivo principal, las diferentes restricciones alimenticias que puedes tener, las habilidades en la cocina que posees, el tiempo del que dispones para cocinar los alimentos que pueden aparecer en la dieta, y diferentes notas que puedes dejar como con cosas que prefieras, por ejemplo. Una vez se genere la dieta, esta te redirigirá al apartado de dietas.

A screenshot of a web page titled "Fitness-Tracker". At the top, there's a navigation bar with icons for Home, App, Sobre Nosotros, and Nuestros Servicios. Below the navigation is a sub-navigation bar with "Inicio", "Rutinas", and "Dietas". The main content area has a title "Modelar dieta". It contains several input fields: "Edad" (Age), "Peso (kg)" (Weight), "Nivel de Actividad Física" (Physical Activity Level), "Habilidad en la Cocina" (Cooking Skill), "Género" (Gender), "Altura (en centímetros)" (Height), "Objetivo Principal" (Main Goal), "Tiempo disponible para Cocinar" (Cooking Time Available), and "Notas y comentarios adicionales" (Additional Notes and Comments). There are also several checkboxes for dietary restrictions: "Vegetariano", "Vegano", "Gluten", "Lacteos", "Frutos Secos", and "Otros". A "GENERAR" (Generate) button is at the bottom of the form.

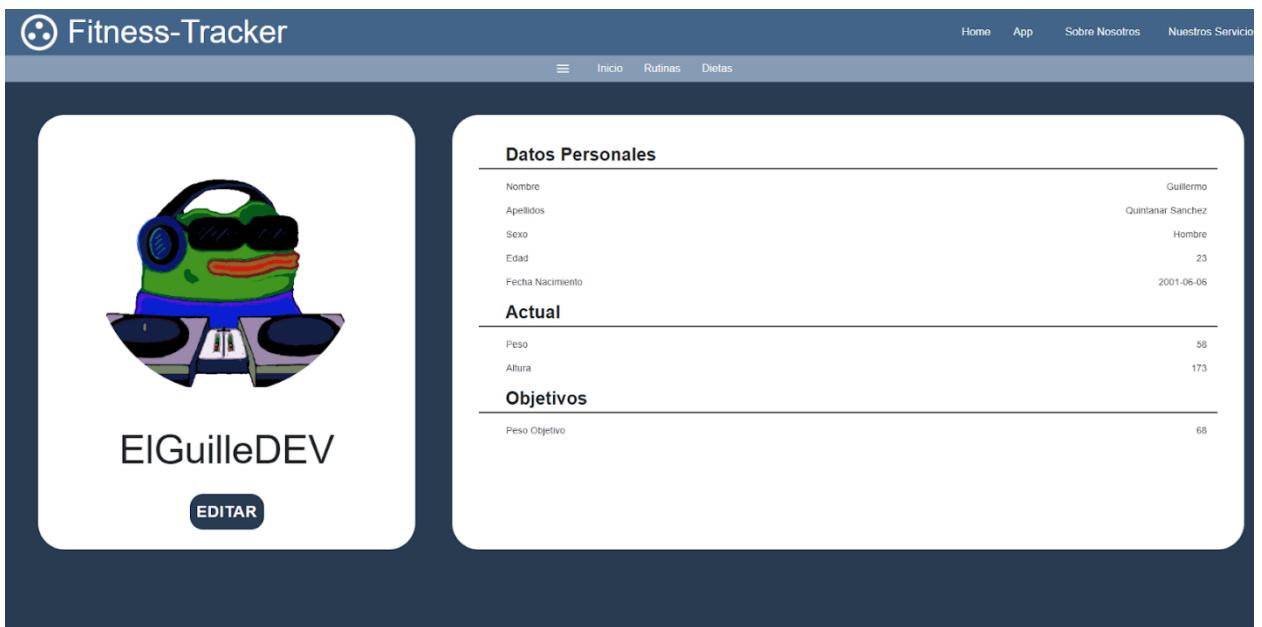
En el apartado de dietas podemos encontrar las diferentes dietas que hemos generado a lo largo del tiempo.

The screenshot shows the 'Dietas' (Diets) section of the Fitness-Tracker website. At the top, there's a header with the logo 'Fitness-Tracker' and navigation links for Home, App, Sobre Nosotros, and Nuestros Servicios. Below the header, a title 'Objetivo en calorías' (Calorie Goal) is displayed with a value of '2000'. Underneath this, there's a note indicating the start date as 'May 1, 2024' and the end date as 'May 31, 2024'. A large button labeled 'VISUALIZAR' (VIEW) is present. Below this, a section titled 'Listado de dietas' (List of diets) shows a preview of a diet plan with a blue background and white text. It includes a placeholder image of a meal, the title 'Dieta', and the same start and end dates as the main goal. Another 'VISUALIZAR' button is located at the bottom of this preview section.

En el apartado de rutinas podemos encontrar las diferentes rutinas que han sido creadas desde la aplicación móvil. Estas contienen los datos físicos que vamos recopilando.

The screenshot shows the 'Rutinas' (Routines) section of the Fitness-Tracker website. At the top, there's a header with the logo 'Fitness-Tracker' and navigation links for Home, App, Sobre Nosotros, and Nuestros Servicios. Below the header, a title 'Listado de Rutinas' (List of routines) is displayed. Two routine cards are shown side-by-side. The first card is for 'Rutina del May 21, 2024' and the second for 'Rutina del June 13, 2024'. Each card contains a summary of physical activity data: sleep time, calories burned, steps taken, heart rate, and blood oxygen level. The first routine shows 8 minutes of sleep, 500 kcal burned, 10000 steps, 60 BPM heart rate, and 98% blood oxygen. The second routine shows 0 minutes of sleep, 0 kcal burned, 0 steps taken, 0 BPM heart rate, and 0% blood oxygen.

En el apartado de información de cuenta podemos encontrar la siguiente pantalla con toda la información acerca del usuario. En esta pantalla podemos encontrar un botón de Editar el cual nos redirige a un formulario para poder editar los datos del usuario.



En el formulario para editar los datos del usuario encontramos lo siguiente. El nombre, el nombre de usuario, el género, primer apellido, segundo apellido, la fecha de nacimiento, el peso objetivo del usuario, peso actual del usuario y la altura actual del usuario. Abajo podemos encontrar un botón que, al pulsar, nos dejará cargar una imagen interna. Al darle a editar los datos se actualizan y te redirige a el apartado de información de cuenta.

The screenshot shows the "Editar Perfil" (Edit Profile) form. It contains the following fields:

| | | |
|---|-----------------------------|------------------|
| Nombre Guillermo | Username ElGuilleDEV | Género Hombre |
| Primer Apellido Quintanar | Segundo Apellido Sanchez | |
| Fecha de Nacimiento(yyyy-mm-dd) 2001-06-06 | Objetivo de Peso 68 | |
| Peso 58 | Altura 173 | |

Below the form is a placeholder for a user profile picture with the text "ElGuilleDEV" and an "EDITAR" button. At the bottom center is a "CARGAR IMAGEN" (Upload Image) button.

Plan de formación a los Usuarios de la Aplicación

El plan de formación a los usuarios de la aplicación Fitness Tracker está diseñado para asegurar que todos los usuarios puedan aprovechar al máximo las funcionalidades y beneficios que ofrece la aplicación.

Para garantizar un aprendizaje continuo, ofrecemos el correo fitness.tracker.opal@gmail.com donde los usuarios pueden exponer sus preguntas.

Bibliografía y fuentes de información

(s.f.). Obtenido de Dveloper Android:

<https://developer.android.com/training/wearables/data/data-layer>

API Gateway. (s.f.). Obtenido de <https://www.redhat.com/en/topics/api/what-does-an-api-gateway-do>

AWS Academy. (s.f.). Obtenido de <https://aws.amazon.com/training/awsacademy/>

Cómo dockerizar una aplicación de Spring Boot. (s.f.). Obtenido de Epam Anywhere:
<https://anywhere.epam.com/es/blog/como-dockerizar-una-aplicacion-de-spring-boot>

Dieta y nutrición. (s.f.). Obtenido de <https://mhanational.org/dieta-y-nutricion>

Edamam. (s.f.). Obtenido de <https://www.edamam.com/>

Implement API Gateway. (s.f.). Obtenido de <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/implement-api-gateways-with-ocelot>

Material UI. (1 de Septiembre de 2021). Obtenido de Mui: <https://mui.com/material-ui/all-components/>

Statista. (2024). Obtenido de Ventas de relojes inteligentes a nivel mundial de 2016 a 2025: <https://es.statista.com/estadisticas/664393/prevision-de-las-ventas-mundiales-de-smartwatches/>

Tutorial de Docker. (2 de Octubre de 2021). Obtenido de Tutorial de Docker:
<https://atareao.es/tutorial/docker/>