UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Gestión personal de dietas saludables (II): Interfaz de usuario y gestión de usuarios y listas de compra

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Ballesteros Rosillo, Raquel

Tutor/a: Canós Cerdá, José Hilario

CURSO ACADÉMICO: 2023/2024

Resum

El present Treball de Fi de Grau se centra en el desenrotllament d'una aplicació mòbil de nutrició dissenyada per a millorar el coneixement i control alimentari dels usuaris, utilitzant dades extretes de diverses pàgines web de supermercats. Principalment, es pretén obtindre informació sobre els productes disponibles en supermercats per a obtindre els seus preus i oferir als usuaris una perspectiva realista del cost associat amb mantindre una dieta equilibrada. No obstant això, atés que les pàgines dels supermercats sovint manquen de detalls sobre el valor nutricional dels aliments, es recopilaran estes dades de tots els productes disponibles en el supermercat en altres fonts. En este treball s'adreçarà la problemàtica descrita anteriorment. Centrant-se en el disseny i implementació de la interfície d'usuari i en diferents funcionalitats com la gestió d'usuaris, el càlcul de calories sobre la base dels perfils de cada usuari, i la generació automàtica de llistes de la compra, que inclouen informació detallada sobre els productes disponibles en supermercats, com a preus i valors nutricionals per a facilitar la compra setmanal d'aliments. Addicionalment, es van realitzar proves unitàries i proves amb usuaris reals. Els resultats d'estes avaluacions van demostrar que l'aplicació no sols és funcional, sinó també fàcil d'usar i acceptada pels usuaris.

Paraules clau: alimentació, dieta saludable, interfície d'usuari, extracció d'informació

Resumen

El presente Trabajo de Fin de Grado se centra en el desarrollo de una aplicación móvil de nutrición diseñada para mejorar el conocimiento y control alimentario de los usuarios, utilizando datos extraídos de diversas páginas web de supermercados. Principalmente, se pretende obtener información sobre los productos disponibles en supermercados para obtener sus precios y ofrecer a los usuarios una perspectiva realista del coste asociado con mantener una dieta equilibrada. No obstante, dado que las páginas de los supermercados a menudo carecen de detalles sobre el valor nutricional de los alimentos, se recopilarán estos datos de todos los productos disponibles en el supermercado en otras fuentes. En este trabajo se abordará la problemática descrita anteriormente. Centrándose en el diseño e implementación de la interfaz de usuario y en distintas funcionalidades como la gestión de usuarios, el cálculo de calorías en base a los perfiles de cada usuario, y la generación automática de listas de la compra, que incluyen información detallada sobre los productos disponibles en supermercados, como precios y valores nutricionales para facilitar la compra semanal de alimentos. Adicionalmente, se realizaron pruebas unitarias y pruebas con usuarios reales. Los resultados de estas evaluaciones demostraron que la aplicación no solo es funcional, sino también fácil de usar y aceptada por los usuarios.

Palabras clave: alimentación, dieta saludable, interfaz de usuario, extracción de información

Abstract

This Final Degree Project focuses on the development of a mobile nutrition application designed to improve users' food knowledge and control, using data extracted from various supermarket websites. The main objective is to obtain information about the products available in supermarkets in order to obtain their prices and offer users a realistic perspective of the cost associated with keeping a balanced diet. However, as supermarket websites often lack details of the nutritional value of foods, this data will be collected for all products available in the supermarket from other sources. This project

will address the issues described above. It focuses on the design and implementation of the user interface and on different functionalities such as user management, calorie calculation based on user profiles, and the automatic generation of shopping lists, which include detailed information on the products available in supermarkets, such as prices and nutritional values to facilitate weekly grocery shopping. In addition, unit tests and user testings were conducted. The results of these evaluations showed that the application is not only functional, but also easy to use and accepted by users.

Key words: alimentation, healthy diet, user interface, information extraction

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En la actualidad, la preocupación por llevar una vida saludable y equilibrada ha aumentado considerablemente. Este interés creciente por el bienestar general está estrechamente ligado a una correcta nutrición, que juega un papel fundamental en la salud de las personas. A medida que se incrementa la conciencia sobre la importancia de una alimentación adecuada, también lo hace la necesidad de herramientas y recursos que faciliten la gestión de la dieta diaria.

La tecnología ha desempeñado un papel importante en este contexto, ya que ha facilitado el acceso a información y herramientas para gestionar y organizar de manera efectiva la alimentación diaria. En particular, las aplicaciones móviles de nutrición han emergido como una solución práctica y accesible para muchos, ofreciendo una manera económica y conveniente de adoptar hábitos saludables y prácticas nutricionales efectivas.

Este Trabajo de Fin de Grado (TFG) se enfoca en el diseño y desarrollo de una aplicación móvil que ayude a mejorar el conocimiento y el control sobre la alimentación de los usuarios. Brocolee está diseñada para ofrecer un menú adaptado a las necesidades individuales de cada usuario, lo que les permitirá no solo seguir un plan alimenticio más saludable, sino también adquirir los conocimientos necesarios para integrar estos hábitos en su vida cotidiana.

Además de su función principal de organizar la dieta, la aplicación también se distingue por su capacidad para ofrecer datos reales sobre productos disponibles en supermercados. Esta funcionalidad permitirá a los usuarios gestionar de manera eficiente sus compras, asegurando que tengan acceso a los ingredientes necesarios para seguir las recetas propuestas. De este modo, la aplicación no solo actúa como una guía nutricional, sino que también optimiza el proceso de compra, contribuyendo a una gestión más efectiva de los recursos y al fomento de hábitos alimenticios saludables.

1.1 Motivación

La nutrición es fundamental para llevar una vida saludable. Sin embargo, muchas personas encuentran dificultades en la gestión de su alimentación de manera adecuada. Esto se puede deber a diversas razones, como el desconocimiento, la falta de tiempo, la falta de acceso a los recursos adecuados o incluso la falta de interés. Además, no siempre saben qué comer y la búsqueda de productos, precios y lugares de compra sigue siendo un desafío. Ante esta situación, Brocolee no solo genera un menú saludable adaptado al usuario, sino que además ofrece una galería de recetas detalladas donde se explican todos los platos de manera clara y sencilla. También cuenta con una lista de la compra

en la que se indica al usuario todos los productos que debe comprar para poder realizar todos los platos que aparecen en el menú semanal.

De esta manera, se ayudaría a los usuarios a tomar decisiones informadas y gestionar su alimentación de manera eficiente, reduciendo el estrés asociado a la planificación de comidas y las compras. La meta a alcanzar es tener un impacto positivo significativo en la vida de las personas, promoviendo hábitos alimentarios saludables y mejorando su bienestar general.

1.2 Objetivos

El objetivo principal de este proyecto consiste en el desarrollo de una aplicación móvil que ayude a los usuarios a mantener una alimentación saludable y les facilite la incorporación de hábitos saludables en su vida.

Para ello, se han definido los siguientes objetivos del proyecto:

1. Realizar un estudio de mercado, para poder reconocer cuáles serían las fortalezas y debilidades que presenta el producto con respecto a las aplicaciones que ya hay en el mercado, identificando qué características nos pueden diferenciar de las ya existentes.
2. Implementar una solución accesible en la que los usuarios puedan acceder a planes nutricionales que se adapten a sus necesidades personales.
3. Implementar una lista de la compra que funcione con datos reales extraídos de supermercados. Esto no solo facilitará la compra, sino que también permitirá a los usuarios hacer elecciones más informadas y económicas.
4. Desarrollar una herramienta que resulte fácil de usar e intuitiva, para mejorar la experiencia de usuario y fomentar un mayor uso de la aplicación.

1.3 Colaboraciones

Este trabajo es complementario de otro TFG Gestión personal de dietas saludables (I): menús, recetas y calendario [6]. De esta forma, en la aplicación que se desarrollará de manera conjunta, se abordará la problemática descrita anteriormente dividiendo el trabajo a realizar en 2 partes:

1. Extracción de los datos de las diferentes fuentes, la gestión de las recetas y la creación y gestión del menú semanal, llevado a cabo en el TFG complementario.
2. Diseño e implementación de las interfaces de usuario, junto con el desarrollo e implementación de la gestión de usuarios y listas de compra, llevados a cabo en el presente TFG.

Por lo tanto, en este trabajo se muestra cómo se ha tratado la gestión de los usuarios, es decir, registro, acceso y modificación de usuarios. El cálculo de las calorías recomendadas que deben consumir en función a sus características físicas y cuales son los criterios y fórmulas que se han seguido para que los resultados tengan rigor científico. Además, se presentará la solución propuesta para la creación de la lista de la compra. Y finalmente, como se han abordado las dificultades encontradas en la implementación de las interfaces.

1.4 Estructura de la memoria

La memoria se organiza en nueve capítulos. Tras la Introducción (Sección 1), se abordan los fundamentos teóricos esenciales y se realiza un análisis del estado del arte en el área en la Sección 2. A continuación la Sección 3 describe cómo es la Arquitectura de la Aplicación desarrollada, detallando la estructura del sistema. La Sección 4 explica la Metodología utilizada durante el desarrollo, discutiendo las estrategias de desarrollo de *software* adoptadas. Después, en la Sección 5, se definen y analizan los Requisitos del sistema. La Sección 6 se enfoca en el Diseño de la Base de datos y las Interfaces. En la Sección 7, se detalla el Desarrollo e Implementación de las principales funcionalidades de la aplicación. Las Pruebas, descritas en la Sección 8, examinan las técnicas empleadas para asegurar la calidad del *software*, incluyendo pruebas unitarias y pruebas con usuarios. Finalmente, la Sección 9 presenta las Conclusiones, resumiendo los objetivos logrados del proyecto y sugiriendo posibles mejoras y trabajos futuros.

CAPÍTULO 2
Conceptos previos y trabajo relacionado

Para entender este proyecto, es de gran importancia explicar ciertos conceptos previos, los cuales ayudarán a comprender el funcionamiento de la aplicación. Además, el estudio de trabajos relacionados facilitará conocer las fortalezas y debilidades de las demás aplicaciones en el mercado, enmarcando así el trabajo en el estado del arte actual.

2.1 Conceptos previos

Los conceptos previos que engloban este trabajo giran en torno al mundo de la nutrición. Para lograr que los usuarios puedan confiar en los planes nutricionales propuestos, estos deben tener unas bases científicas consistentes. Con este fin, se ha hecho un estudio de las distintas opciones disponibles para el cálculo del consumo calórico diario. Cada uno de estos métodos tiene sus particularidades y es necesario comprender sus fundamentos y aplicaciones para poder ofrecer recomendaciones nutricionales precisas y personalizadas.

Es esencial explicar diversos conceptos relacionados con este dominio, como el metabolismo basal o el gasto energético total. De esta manera, se entenderá qué datos que proporcionan los usuarios son necesarios, y cómo estos pueden influir en las necesidades calóricas individuales, tales como la edad, el sexo y el nivel de actividad física.

2.1.1. Tasa metabólica basal

El metabolismo basal, también conocido como tasa metabólica basal (TMB) o Gasto Energético Basal (GEB) [7], es la cantidad mínima de energía que el cuerpo necesita para mantener sus funciones vitales en reposo. Estas funciones incluyen la respiración, la circulación sanguínea, la regulación de la temperatura corporal, la función celular, y la actividad del sistema nervioso. El metabolismo basal representa la mayor parte del gasto calórico diario de una persona promedio.

Conocer el metabolismo basal es el punto de partida para personalizar recomendaciones dietéticas y de actividad física. Determina la cantidad de calorías que una persona necesita consumir para mantener su peso corporal si está en reposo absoluto. Por tanto, es necesario para diseñar planes de alimentación que ayuden a alcanzar objetivos específicos como perder, ganar o mantener el peso.

Además, la TMB se utiliza como base para calcular el gasto energético total (GET), que incluye el gasto energético asociado con la actividad física. Con el GET, se pueden ajustar

las recomendaciones dietéticas para satisfacer las necesidades energéticas globales de un individuo.

Hay varios factores que pueden influir en la TMB, y es importante tenerlos en cuenta al calcularla [8]. Los más importantes son:

- Peso: Este tiene un impacto significativo en la TMB. En general, cuanto mayor sea el peso corporal, mayor será la TMB. Esto se debe a que el tejido muscular quema más calorías en reposo que el tejido graso. Por lo tanto, una persona con más masa muscular y un peso corporal más alto, normalmente tiene una TMB más alta que alguien con menos masa muscular y el mismo peso.
- Edad: La edad también tiene un gran impacto, ya que con los años la TMB tiende a disminuir. A medida que se envejece, se va perdiendo masa muscular y ganando tejido graso.
- Sexo: Otro factor importante es el sexo, ya que como norma general los hombres tienen una TMB más alta. Esto se debe a que los hombres tienen una mayor masa muscular y menos tejido graso en comparación con las mujeres.
- Estado de salud: El estado de salud general también puede afectar la TMB. Por ejemplo, condiciones como el hipotiroidismo (baja actividad de la glándula tiroides) pueden disminuir la TMB, mientras que el hipertiroidismo (alta actividad de la glándula tiroides) puede aumentarla.

Además de estos factores, otros aspectos como el nivel de actividad física, la temperatura ambiente y la dieta también pueden influir en la TMB. Al calcularla, es importante considerar todos estos factores para obtener una estimación precisa de las necesidades energéticas en reposo.

2.1.2. Índice de masa corporal

El índice de masa corporal (IMC) [9] es el estándar internacionalmente más reconocido para determinar un peso saludable y para evaluar el nivel de sobrepeso o delgadez. Este permite llevar a cabo una mejor comparación entre dos personas adultas con diferente estatura y sexo.

$$IMC = peso/altura^2 \tag{2.1}$$

La Ecuación 2.1 con la que se calcula el IMC interpreta con mayor precisión el peso (kg) en relación con la estatura (m).

Finalmente, es necesario saber de que manera interpretar el resultado del cálculo, según la Organización Mundial de la Salud (OMS) un IMC "normal"se encuentra entre 18.5 y 24.9. Un individuo con un IMC por debajo de 18.5 se considera delgado, mientras que un IMC de 25 o más indica sobrepeso. [10]

2.1.3. Conceptos sobre peso

De entre todos los datos y factores a tener en cuenta a la hora de realizar estudios relacionados con el bienestar nutricional, es importante considerar no solo el TMB y el IMC, sino también conceptos relacionados con el peso [7]. El peso es un factor clave en la nutrición y la salud, y al ser objeto de estudios con fines nutricionales, se deben considerar diferentes definiciones del mismo:

- **Peso ideal:** es el peso en kilogramos estimado que una persona debe tener según su altura, edad, sexo y constitución física. Este se puede calcular a través de las Ecuaciones 2.2 y 2.3

$$Peso\_ideal(Hombres) = altura(m)^2 \times 23 \tag{2.2}$$

$$Peso\_ideal(Mujeres) = altura(m)^2 \times 21 \tag{2.3}$$

- **Peso actual:** es el peso que una persona tiene en un momento específico, el cual será totalmente necesario para averiguar cuál es el estado en el que se encuentra y, de esta manera, poder reconocer cuál es el objetivo que quiere alcanzar y a través de qué métodos puede conseguirlo, para así tomar un camino que sea nutricionalmente saludable.

- **Peso ajustado:** es un valor hipotético del peso medido en kilogramos que se emplea como meta inicial en pacientes con sobrepeso y obesidad. Este se calcula a través de la Ecuación 2.4

$$Peso\_ajustado = \frac{(Peso\_actual - Peso\_ideal)}{3} + Peso\_ideal \tag{2.4}$$

Para ajustar las calorías a las necesidades de los usuarios, se realizará un ajuste basado en el IMC. Este ajuste permitirá calcular las calorías necesarias para que el usuario mantenga un IMC saludable, utilizando el peso correspondiente en función de sus necesidades.

2.1.4. Fórmulas para el cálculo del consumo calórico diario

Para calcular el menú adaptado a cada usuario, es necesario llevar a cabo un cálculo de las calorías diarias que estos deberían consumir. De esta forma, el menú propuesto garantizaría el consumo necesario para cubrir las necesidades calóricas de cada individuo.

Con este objetivo, se ha hecho un estudio sobre los diversos métodos existentes para realizar el cálculo, reduciéndose a tres opciones, resumidas en el Sistema experto para cálculo automático de calorías diarias basado en parámetros corporales [5].

Método Harris-Benedict

La fórmula de Harris-Benedict [2] data de 1918 y, tras varias revisiones y adaptaciones, hoy en día sigue siendo una de las fórmulas más utilizadas y más recomendadas por los nutricionistas. Los investigadores que le dan nombre, J. Arthur Harris y Francis G. Benedict, llegaron a la conclusión de que era posible calcular la producción diaria de calorías de un individuo mediante tres parámetros: peso, estatura y edad.

Hombres	h = 66.4730 + 13.7516 w + 5.0033 s - 6.7550 a
Mujeres	h = 655.0955 + 9.5634 w + 1.8496 s - 4.6756 a

Tabla 2.1: Ecuaciones originales de Harris-Benedict. Fuente: [2]

En la Tabla 2.1 se pueden observar las ecuaciones originales de Harris-Benedict, pensadas para poder calcular el metabolismo basal de un sujeto desconocido. Estas fórmulas, donde:

- h = Producción total de calorías en 24 horas (Tasa Metabólica Basal)

- w = Peso en kilogramos
- s = Estatura en centímetros
- a = Edad en años

fueron revisadas y actualizadas en varias ocasiones, siendo la revisión más actual la de Mifflin y otros [3], publicadas en 1990.

Hombres	REE = 9.99 w + 6.25 s – 4.92 a + 5
Mujeres	REE = REE = 9.99 w + 6.25 s - 4.92 a - 161

Tabla 2.2: Actualización de las fórmulas de Harris-Benedict por Mifflin et al. Fuente: [3]

En este caso, como se muestra en la tabla 2.2 se calcula el *Resting Energy Expenditure* (REE), el consumo de calorías cuando se está en reposo, es decir, las que el cuerpo consume por sí mismo sin actividad del individuo, que son el equivalente a la tasa metabólica basal. Además se aplicó un redondeo, ya que según los autores no afectaba a la predictibilidad general y el resultado seguía siendo válido.

Hombres	REE = 10 w + 6.25 s – 5 a + 5
Mujeres	REE = REE = 10 w + 6.25 s - 5 a - 161

Tabla 2.3: Redondeo de las fórmulas de Harris-Benedict por Mifflin et al. Fuente: [3]

Las fórmulas que se muestran en la tabla 2.3 calculan la energía que un individuo necesita cuando se mantiene en reposo. Si lo multiplicamos por un factor de actividad, según muestra la tabla 2.4 conseguiríamos el consumo de calorías totales que se requieren durante el día.

Factor	Categoría	Definición
1.2	Sedentario	Poco o ningún ejercicio, trabajo de escritorio
1.375	Ligero	Actividad Ligera, deporte 1-3 veces por semana.
1.55	Activo	Actividad moderada, deporte 3-5 veces por semana.
1.725	Muy Activo	Ejercicio duro, deporte 6-7 por semana
1.9	Extremadamente Activo	Ejercicio fuerte diario o deportes y trabajo físico muy duro

Tabla 2.4: Factor de actividad de acuerdo con Método Harris-Benedict. Fuente: [4]

Método FAO/OMS/UNU

El método FAO/OMS/UNU [5] es una herramienta utilizada para evaluar las necesidades de proteínas y energía en los seres humanos. Fue desarrollado por la Organización de las Naciones Unidas en el año 2004 para la Agricultura y la Alimentación (FAO), la Organización Mundial de la Salud (OMS) y la Universidad de las Naciones Unidas (UNU).

Las fórmulas que propusieron, se pueden observar en la Tabla 2.5, donde P = peso (kilogramos), derivaban de las publicadas en 1985, pero en este caso solo toma en cuenta el peso, el sexo y la edad del individuo.

Además, al igual que en las fórmulas de Harris-Benedict, se deben multiplicar por un factor de actividad en función de la cantidad de ejercicio físico que realiza cada individuo. Estos factores de actividad se pueden observar en la Tabla 2.6.

Edad (Años)	Hombres	Mujeres
0 a 2	(60.9 * P) - 54	(61.0 * P) - 51
3 a 9	(22.7 * P) + 495	(22.5 * P) + 499
10 a 17	(17.5 * P) + 651	(12.2 * P) + 746
18 a 29	(15.3 * P) + 679	(14.7 * P) + 496
30 a 59	(11.6 * P) + 879	(8.7 * P) + 829
>=60	(13.5 * P) + 487	(10.5 * P) + 596

Tabla 2.5: Tasa Metabólica Basal de acuerdo con FAO/OMS/UNU. Fuente: [5]

Actividad	Hombres	Mujeres	Actividad Física
Sedentaria	1.20	1.20	Sin actividad
Liviana	1.55	1.56	3 horas semanales
Moderada	1.80	1.64	6 horas semanales
Intensa	2.10	1.82	4 a 5 horas diarias

Tabla 2.6: Factor de actividad para calcular calorías diarias. Fuente: [5]

Método Rápido

También existe un método rápido para estimar las calorías necesarias para perder, mantener o aumentar el peso. Este método se basa en multiplicar el peso corporal por un factor específico. La fórmula es la siguiente:

- **Para bajar de peso:** 26 a 29 Kcal por kilogramo de peso.
- **Para mantener el peso:** 33 a 35 Kcal por kilogramo de peso.
- **Para subir de peso:** 40 a 44 Kcal por kilogramo de peso.

Sin embargo, este método solo considera el peso corporal sin tener en cuenta la composición de músculo y grasa, lo que puede afectar la precisión de las estimaciones.

2.2 Estado del arte

A continuación, se realizará un estudio sobre el estado del arte actual en relación con nuestra aplicación. En este caso, nos referimos a las aplicaciones móviles centradas en la nutrición y la integración de buenos hábitos alimenticios en la vida cotidiana de los diferentes usuarios. Para ello, tras realizar una búsqueda de las aplicaciones más destacadas y con funcionalidades más similares a las de nuestro proyecto, se han seleccionado como principales competidores Nutrilio, Fitatu, Fitia y Unimeal.

Cabe destacar que todas las aplicaciones estudiadas se pueden descargar de manera gratuita desde Google Play Store. De esta forma, se ha podido probar cada una de ellas, y así identificar y diferenciar las funcionalidades que ofrecen.

El objetivo de este estudio es conocer las debilidades de los competidores y añadir las funcionalidades necesarias para asegurar la posición de nuestra aplicación en el mercado.

2.2.1. Análisis de las aplicaciones

Nutrilio

Nutrilio [11] es una aplicación diseñada para rastrear alimentos, agua y peso. Permite llevar un diario de comidas, personalizar el seguimiento y obtener información sobre la dieta y estilo de vida del usuario. Este puedes registrar aspectos como el estado de ánimo, el ejercicio y los síntomas de salud. La aplicación incluye características como el seguimiento de la hidratación y el peso, análisis de comidas, modo oscuro y seguridad de datos.

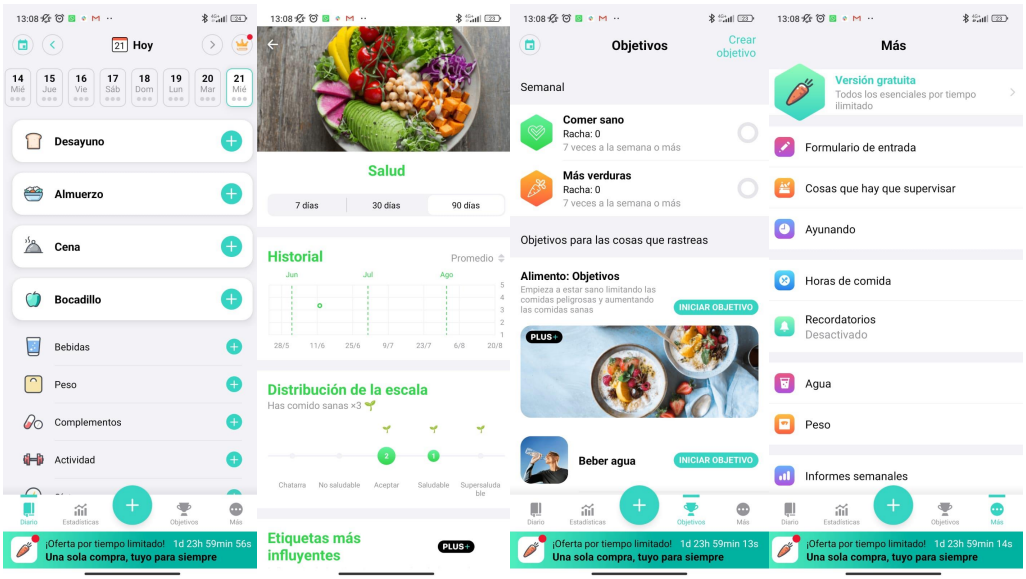


Figura 2.1: Interfaces de la aplicación **Nutrilio**

A pesar de parecer una aplicación con una gran variedad de funcionalidades, es cierto que, sin conocimientos básicos de nutrición, puede resultar más difícil para el usuario encontrar utilidades reales más allá de llevar un registro de comidas, sesiones de ejercicio o vasos de agua. El usuario se va a limitar a apuntar sus datos semanales, de los cuales obtendrá unas estadísticas que él mismo debe interpretar.

Fitatu

Se trata de una aplicación de seguimiento de calorías y dieta [12]. Ayuda a los usuarios a perder peso, ganar masa muscular o mantenerse saludables mediante el registro de alimentos, agua y actividad física. Ofrece una base de datos extensa de productos y platos, escáner de códigos de barras, sincronización con aplicaciones deportivas y seguimiento detallado de nutrientes. También permite compartir dietas, establecer objetivos personalizados y monitorear el progreso a través de gráficos e informes.

Mediante **Fitatu** el usuario puede planificar las comidas de las próximas semanas, contando con la ayuda de las recetas que proporciona la misma aplicación. A través de estas, permite la creación de una lista de la compra. Además, el usuario recibirá sugerencias de alimentos en base a la necesidad calórica del usuario

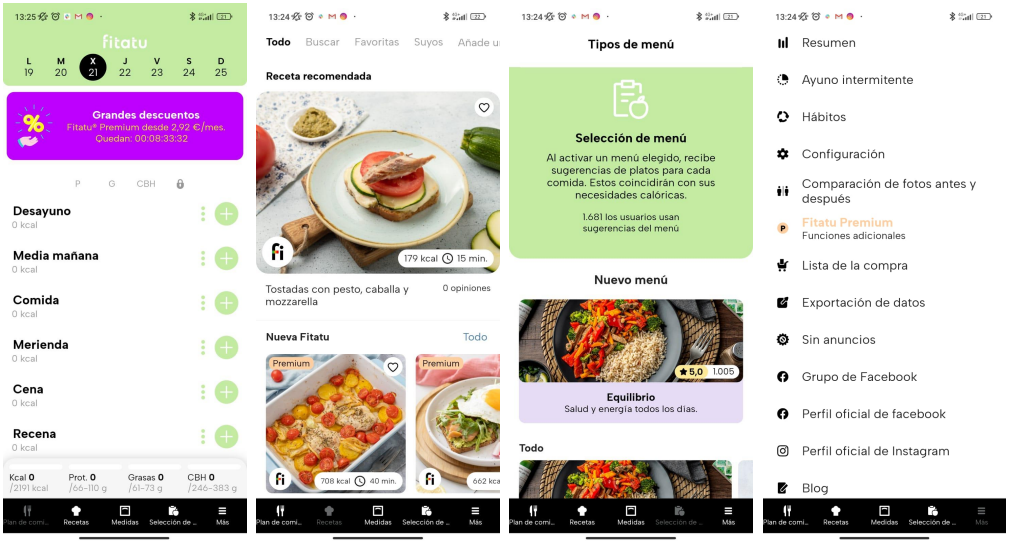


Figura 2.2: Interfaces de la aplicación **Fitatu**

Fitia

Fitia [13] es una aplicación de nutrición inteligente que ayuda a planificar las comidas, contar calorías y alcanzar los objetivos de salud del usuario sin restricciones alimentarias. Además, personaliza su plan alimenticio en función de sus gustos, objetivos (como reducir grasa, mantener peso o ganar músculo) y datos personales como la edad, estatura, peso y nivel de actividad física.

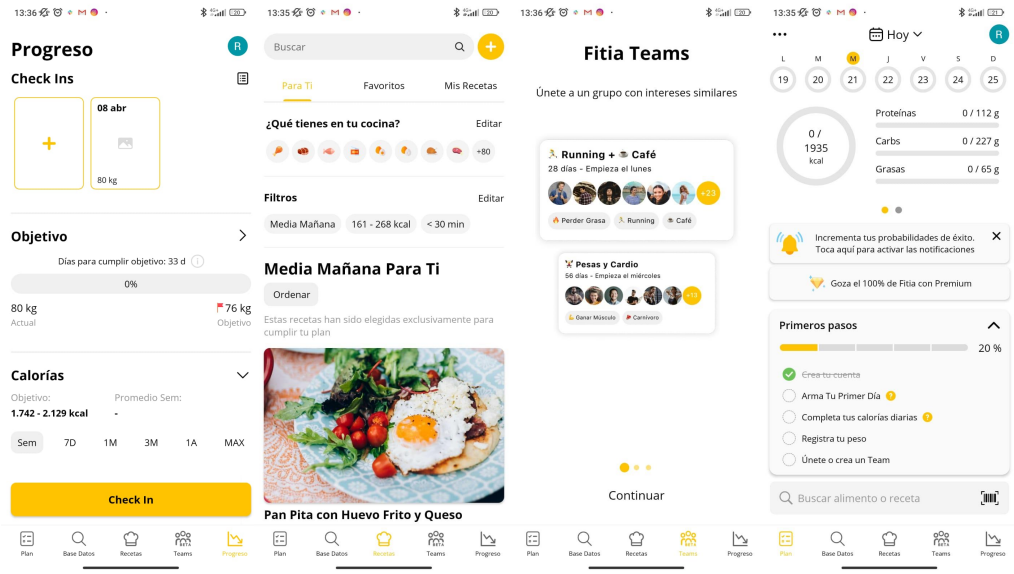


Figura 2.3: Interfaces de la aplicación **Fitia**

Esta aplicación destaca por su extensa base de datos de alimentos y recetas verificadas, además de tener un planificador nutricional que ayuda a los usuarios a organizar sus comidas con sugerencias inteligentes. También, permite realizar un seguimiento detallado del consumo diario de nutrientes y calorías, lo que facilita el control y ajuste del plan de alimentación en tiempo real. **Fitia** ofrece opciones de integración con dispositivos *fitness* para monitorear la actividad física, mejorando aún más la personalización de los planes. La plataforma es fácil de usar, ideal tanto para principiantes como para usuarios

experimentados, y se actualiza constantemente con nuevas funcionalidades y alimentos en su base de datos.

Unimeal

Unimeal [1] es una aplicación de gestión de pérdida de peso que ofrece planes de comidas personalizados, ejercicios para quemar grasa, y soporte profesional las 24 horas. La aplicación está diseñada para ayudarte a alcanzar tus objetivos de salud y nutrición a través de planes de alimentación adaptados a tus necesidades y preferencias, recetas paso a paso, y educación sobre un estilo de vida saludable.

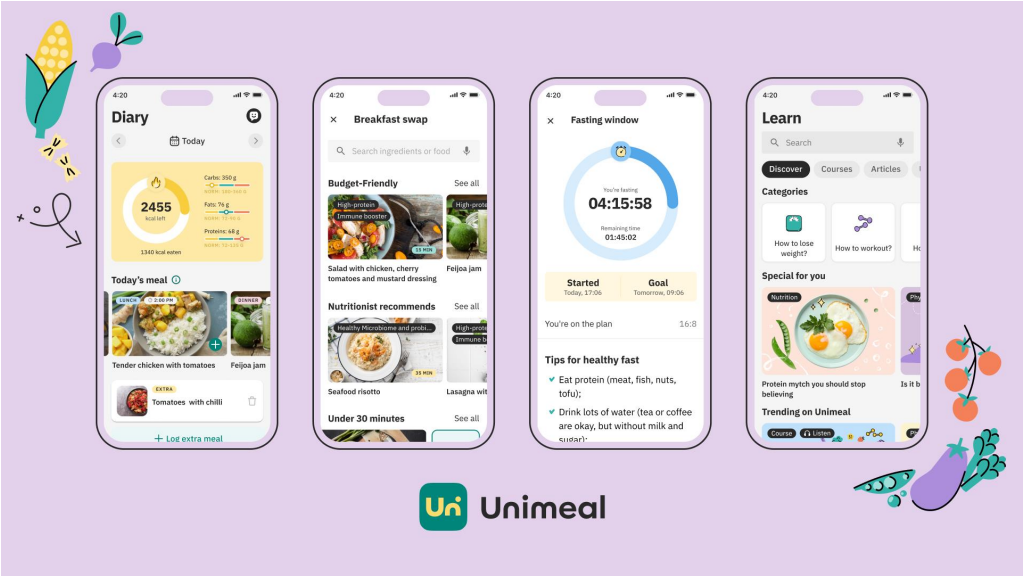


Figura 2.4: Interfaces de la aplicación **Unimeal**. Fuente [1]

Probablemente se trate de la aplicación más completa en cuanto a funcionalidades, ya que, en este caso, el usuario sí que tendrá acceso a un plan de comidas personalizado, además de permitir automatizar la creación de la lista de la compra según el plan semanal.

2.2.2. Comparativa de las aplicaciones

A continuación, se hará un análisis comparativo sobre las funcionalidades incluidas en cada una de las aplicaciones estudiadas. Con el fin de explicar claramente los aspectos evaluados, se detalla a continuación los criterios sobre los cuales se basará este estudio comparativo.

1. **Facilidad de uso:** Evalúa qué tan intuitiva y accesible es la aplicación para los usuarios.
2. **Funcionalidades principales:** Determina las características esenciales que ofrece cada aplicación y su grado de implementación.
3. **Rendimiento:** Analiza la rapidez con la que cada aplicación ejecuta sus tareas.
4. **Compatibilidad:** Examina en qué plataformas está disponible la aplicación.

En la Figura 2.5 se muestra cuáles son las características en las que destaca cada aplicación, lo que permitirá identificar las fortalezas y debilidades de cada una de ellas en

Figura 2.5: Comparativa de las características de las aplicaciones estudiadas.

relación con nuestra aplicación. Este análisis servirá de base para definir estrategias de mejora y diferenciación competitiva.

Para empezar, el único caso que no incluye la opción de menú semanal es **Nutrilio**. Aunque esta aplicación permite llevar un registro diario, carece de la posibilidad de planificar las comidas que se llevaran a cabo a lo largo de la semana. En contraste, **Fitia** y **Unimeal** ofrecen la funcionalidad de menú, destacando especialmente por proporcionar un menú generado automáticamente. Para ello, utilizan sus respectivos repositorios de recetas, seleccionando de esta forma las más adecuadas según las necesidades calóricas de los usuarios. Es importante mencionar que, en el caso de **Fitia**, no se trata de una generación automática del menú, sino que, al crear un plan semanal, ofrece sugerencias inteligentes de recetas.

Las cuatro aplicaciones son bastante intuitivas y fáciles de usar, especialmente una vez que los usuarios se familiarizan con sus funciones y navegación. Aunque cada una tiene su propio enfoque, todas ofrecen una experiencia accesible para diferentes niveles de usuarios. **Fitia** se destaca al incluir un asistente virtual, que guía a los usuarios con mayores dificultades en el manejo de la aplicación. En cuanto al rendimiento, todas las aplicaciones son eficientes en la mayoría de las tareas, aunque las que generan menús semanales pueden experimentar una ligera ralentización durante ese proceso.

Por otro lado, **Unimeal** destaca como la aplicación que más funcionalidades ofrece. Aunque, a pesar de poder descargar la aplicación de manera gratuita, todas las funcionalidades que ofrece son de pago. Este aspecto es llamativo, ya que se esperaría que una aplicación con esas características fuese completa y ofreciese un valor claro por el que los usuarios estarían dispuestos a pagar. Sin embargo, en la Figura 2.6 podemos observar como son las calificaciones que los mismos usuarios ponen a cada una de las aplicaciones en la Play Store.

	Calificación promedio	Cantidad de calificaciones
Nutrilio	4,8	8.22 K
Fitatu	4,6	107 K
Fitia	4,8	103 K
Unimeal	2,9	11.7 K

Figura 2.6: Calificación promedio de las aplicaciones en Google Play Store.

No solo **Unimeal** recibe una calificación relativamente baja, sino que además sus competidores destacan con calificaciones altas, a pesar de ofrecer menos funcionalidades. De

esta forma, la aplicación que podría suponer un mayor competidor para nuestro producto parece no estar cumpliendo las expectativas de los usuarios. A través de las críticas de los usuarios se puede apreciar que existe un gran problema con la creación de los menús en **Unimeal**.

Para terminar, **Fitatu** ofrece la mayor compatibilidad al estar disponible en Android, iOS, y una versión web. **Fitia**, **Nutrilio**, y **Unimeal** están limitadas a dispositivos móviles, pero funcionan bien en sus respectivas plataformas.

2.2.3. Conclusiones sobre el estado del arte

Del análisis realizado, podemos decir que, a pesar de tener funcionalidades y temática en común, cada una de las aplicaciones se diferencia de las demás en algún aspecto. Por ello, es importante destacar cómo nos podemos diferenciar con nuestro producto.

Aunque **Fitia** y **Unimeal** ofrecen menús semanales, **Nutrilio** no lo hace. **Fitia** no automatiza menús, pero brinda sugerencias inteligentes, lo que abre una oportunidad para combinar flexibilidad y automatización. Sin embargo, generar menús semanales puede ralentizar el rendimiento, un área clave a optimizar. En términos de compatibilidad, **Fitatu** lidera al estar disponible en múltiples plataformas, lo que sugiere la necesidad de ampliar la disponibilidad de nuestro producto. Finalmente, la baja calificación de **Unimeal** resalta la importancia de equilibrar funcionalidad, facilidad de uso y rendimiento.

CAPÍTULO 3
Arquitectura de la aplicación

3.1 Diseño de la arquitectura

La arquitectura de la aplicación sigue una estructura de tres capas, es decir, se divide en diferentes partes, cada una con una responsabilidad especifica. Esta separación permite que cada capa evolucione de manera independiente, simplificando las actualizaciones y mejoras del sistema.

3.1.1. Estructura de capas

La arquitectura de la aplicación de este TFG, se divide en las siguientes tres capas:

1. **Capa de lógica de negocio (*Business Logic Layer*):** Aquí es donde se manejan las reglas y la lógica principal de la aplicación, como procesar datos o tomar decisiones basadas en la entrada del usuario. Esta capa asegura que la aplicación funcione correctamente según las reglas establecidas.

2. **Capa de persistencia (*Data Layer*):** Se encarga de gestionar cómo se almacenan, recuperan y manipulan los datos, como bases de datos o servicios externos. Tratándose de una base de datos en la nube, la cual permitirá ofrecer escalabilidad y alta disponibilidad.

3. **Capa de presentación (Frontend):** Es la parte de la aplicación que el usuario ve y con la que interactúa directamente. En este caso, se trata de una aplicación móvil. Esta capa se comunica con el *backend* a través de una API REST. A través de esta API, el *frontend* puede solicitar datos al *backend* y recibir respuestas, como los datos que se mostrarán al usuario o la confirmación de una acción realizada.

3.1.2. Interacción entre las capas

En la Figura 3.1 se puede ver cómo interactúan las capas en la arquitectura, permitiendo una gestión eficaz y estructurada del flujo de datos. La capa de presentación envía las solicitudes del usuario a la capa lógica, que a su vez devuelve los resultados procesados para ser mostrados al usuario. La capa lógica realiza llamadas a la capa de datos para obtener la información necesaria, como datos de recetas, productos de supermercado o valores nutricionales. Finalmente, la capa de datos responde a las solicitudes de la capa lógica, ya sea proporcionando datos almacenados localmente o recuperando información de servicios externos, esta centraliza la información extraída por los *scrapers* de diferentes fuentes, como Mercadona y FatSecret, permitiendo una gestión eficiente y escalable

de los datos. Por ejemplo, si el usuario busca una receta, la solicitud se envía desde la interfaz a través del módulo de gestión de recetas en la capa lógica, los controladores en el backend procesan estas solicitudes y devuelve la información que se mostrará al usuario.

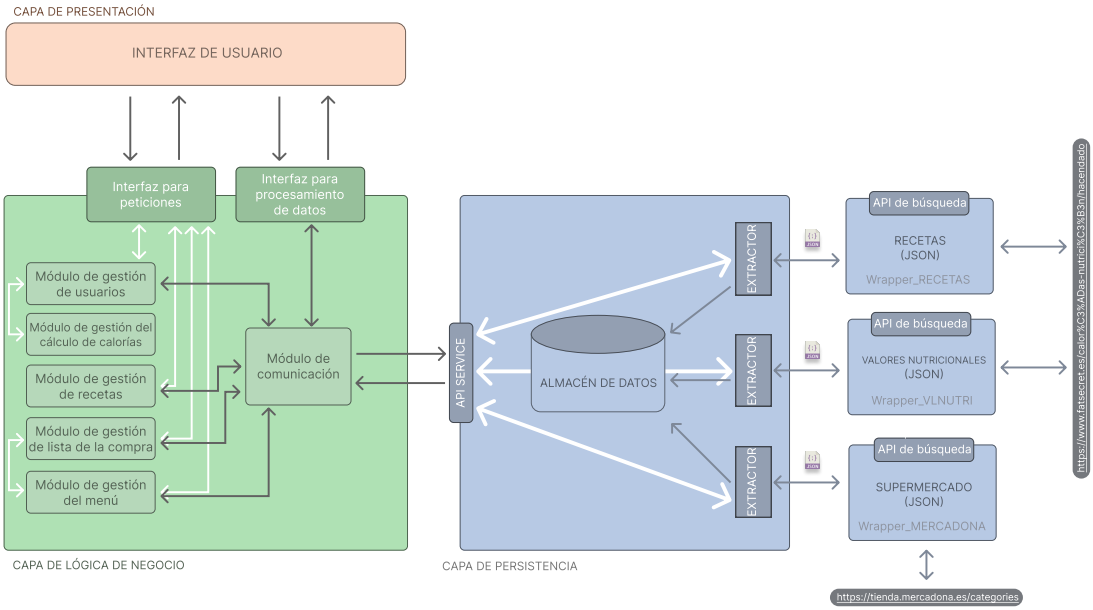


Figura 3.1: Arquitectura del sistema propuesto.

Para explicar correctamente la arquitectura representada en la Figura 3.1 es necesario detallar cada uno de los componentes del sistema, centrándonos en aquellos que forman parte de la capa de presentación y la capa lógica, que son los que se abordan en este Trabajo de Fin de Grado. A continuación, se describe cada componente en función de su ubicación en las capas de la arquitectura:

1. **Capa de lógica:**
    - **Módulo de Gestión de Usuarios:** maneja las peticiones relacionadas con los usuarios, como autenticación, registro, y administración de perfiles. Es responsable de asegurar que las operaciones se realicen en base a los permisos del usuario.
    - **Módulo de Gestión de Recetas:** permite la creación, modificación y eliminación de recetas. También se encarga de gestionar la base de datos de recetas y la relación de estas con otros módulos.
    - **Módulo de Gestión del Menú:** es el encargado de gestionar la creación y modificación del menú semanal en base a las necesidades de los usuarios. Este módulo interactúa con otros módulos como el de recetas y lista de la compra para construir opciones personalizadas basadas en las preferencias y necesidades del usuario.
    - **Módulo de Gestión de la Lista de la Compra:** su función principal es la creación y gestión de la lista de la compra, así como la comunicación con el módulo de gestión del menú semanal.
    - **Módulo de Gestión del Cálculo de Calorías:** en este se realizan los cálculos que permiten averiguar las calorías que necesita cada usuario, este se comunica con el módulo de gestión de usuarios para mandarle la información calorífica antes de crear o modificar algún usuario.
    - **Módulo de Comunicación:** Este módulo se encarga de gestionar las llamadas al API Service que envía las peticiones al backend. Es responsable de recibir

las solicitudes desde la capa de presentación o desde otros módulos y de interactuar con el backend para procesar esas solicitudes y devolver los resultados.

- **Interfaz para peticiones:** es responsable de gestionar todas las solicitudes que provienen del usuario a través de la capa de presentación. Actúa como un punto central de entrada para todas las peticiones, asegurando que cada solicitud sea dirigida al módulo adecuado dentro de la capa lógica para su procesamiento.
- **Interfaz para procesamiento de datos:** su función es asegurar que todos los datos generados, modificados o recuperados durante el procesamiento en la capa lógica sean manejados correctamente antes de ser almacenados en la capa de datos o enviados de vuelta a la capa de presentación.

2. **Capa de persistencia:** esta capa incluye componentes importantes como el **Extractor**, que recopila y distribuye datos de fuentes externas; el **Almacén de Datos**, que es la base de datos central donde se guarda toda la información del sistema; y las **APIs de Búsqueda**, que interactúan con fuentes externas para obtener información. Además, los **Wrappers** adaptan los datos recibidos a un formato procesable internamente, mientras que los **Enlaces HTTP** permiten la conexión con esas fuentes externas para obtener datos en tiempo real.

3. **Capa de presentación:**

   - **Interfaz de Usuario (UI):** Es el diseño y la disposición visual de la aplicación que el usuario ve y con la que interactúa. Está diseñada para ser fácil de usar y comprender, de modo que los usuarios puedan navegar por la aplicación sin problemas.
   - **Controladores de Interfaz:** estos se encargarán de gestionar la interacción entre la interfaz de usuario (UI) y el modelo de datos, facilitando el envío y recepción de información a través de la API REST. Aseguran que los datos que muestra la UI estén actualizados y que cualquier acción del usuario (como solicitar información) sea correctamente manejada por el backend.
   - **Controladores de Eventos:** Estos componentes responden a las acciones del usuario mediante el envío de las solicitudes a la capa lógica de negocio.

En resumen, esta arquitectura bien organizada asegura que la aplicación sea fácil de mantener, mejorar y escalar, además de permitir una comunicación eficiente entre los diferentes componentes de la aplicación. Esto se traduce en un flujo de datos más fluido y una coordinación más efectiva entre las distintas partes del sistema, reduciendo el riesgo de cuellos de botella o conflictos. En conjunto, estos factores permiten una operación más estable y un desarrollo más ágil, contribuyendo al éxito a largo plazo de la aplicación.

CAPÍTULO 4
Metodología

En todo proceso de desarrollo de *software* es fundamental contar con una metodología adecuada que guíe el proyecto desde su concepción hasta su finalización. Las metodologías de desarrollo de *software* proporcionan un marco de trabajo que ayuda a estructurar, planificar y controlar el proceso de desarrollo, con el objetivo de entregar un producto de calidad dentro de los plazos y presupuestos establecidos.

Existen diferentes metodologías que se pueden aplicar dependiendo de las características del proyecto, los requerimientos del cliente, y las circunstancias del equipo de trabajo. Estas metodologías varían en cuanto a su enfoque, sus fases, y las herramientas que emplean.

4.1 Herramientas usadas

- **Git:** Se ha decidido utilizar **Git** [14] como sistema de control de versiones, ya que este permite llevar un registro de todos los cambios realizados en el código fuente del proyecto, haciendo así posible la participación de varios colaboradores.

  Entre las características destacables de **Git**, podemos resaltar que cuenta con un historial detallado de las modificaciones del código, lo que permite averiguar quién ha hecho un cambio o cuándo lo ha hecho. Además, también facilita el desarrollo en paralelo ofreciendo la posibilidad de crear ramas en las que trabajar por separado. De la misma forma, se pueden fusionar esas ramas una vez los cambios están funcionando y no generan conflictos.

- **GitHub:** Se usa como plataforma de control de versiones [15], ya que permitirá una gestión sencilla de las diferentes versiones del *software*. Cabe destacar que **GitHub** se basa en **Git**, es decir, lo utiliza como sistema de control de versiones.

  También es de gran importancia conocer ciertas funcionalidades que ofrece. Una de ellas es la creación de *issues*, la cual permite a los equipos de desarrollo reportar errores, solicitar nuevas características y hacer seguimiento a las tareas pendientes. Los *issues* pueden ser etiquetados, asignados a colaboradores específicos, y enlazados con *commits* y *pull requests*, proporcionando una visión clara del progreso y estado del proyecto. Otras características clave son las *pull requests*, que permiten revisar y discutir el código propuesto antes de fusionarlo con la rama principal del proyecto, y las *merge requests*, las cuales permiten, a través herramientas de comparación y resolución de conflictos, fusionar el código en la rama en cuestión.

- **Trello:** Para mejorar la organización en las tareas se ha usado **Trello** [16]. Se trata de una herramienta de gestión de proyectos que facilita la organización de tareas

y la colaboración en equipo. Utiliza un sistema visual basado en tableros, listas y tarjetas, donde cada tablero representa un proyecto, las listas reflejan etapas o categorías, y las tarjetas contienen tareas específicas. Los usuarios pueden asignar tareas, agregar fechas límite, adjuntar archivos y colaborar en tiempo real, lo que permite una gestión eficiente y flexible del trabajo. Es ampliamente utilizado por equipos para seguir el progreso de los proyectos y mantener la organización en un entorno fácil de usar.

4.2 Metodología tradicional

La metodologías tradicionales [17], por ejemplo la metodología en cascada, proporcionan un enfoque secuencial y estructurado para el desarrollo de *software*. Estas metodologías surgieron a medida que el *software* evolucionaba y aparecían proyectos de mayor envergadura. Se basa en una serie de fases claramente definidas y alineadas en un orden cronológico. Las etapas que la constituyen como se pueden ver en la Figura 4.1 son: **análisis**, **diseño**, **desarrollo**, **pruebas**, **implantación** y **mantenimiento**.

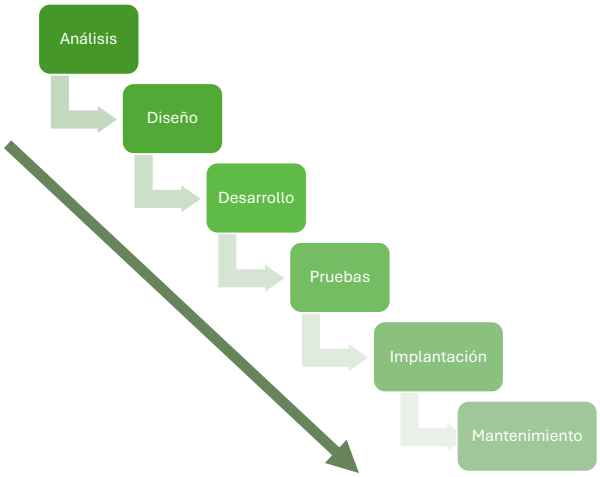


Figura 4.1: Metodología tradicional en cascada.

La metodología tradicional es más adecuada para proyectos de *software* bien definidos y estables, donde los cambios durante el desarrollo son mínimos. Sin embargo, en entornos más dinámicos o inciertos, puede ser necesario considerar otras metodologías más flexibles.

4.3 Metodología ágil

Debido a los altos costes y las limitaciones de las metodologías tradicionales, que no eran adecuadas para proyectos en entornos cambiantes, las metodologías ágiles [17] ganaron gran importancia en el desarrollo de *software* a finales del siglo XX y comienzos del siglo XXI. Estas metodologías ofrecen una mayor flexibilidad y capacidad de adaptación, permitiendo a los equipos responder de manera más eficaz a los cambios en los requisitos del cliente y a las condiciones del mercado. El enfoque ágil se basa en ciclos de desarrollo iterativos y incrementales, lo que facilita la entrega continua de valor y la mejora continua del producto.

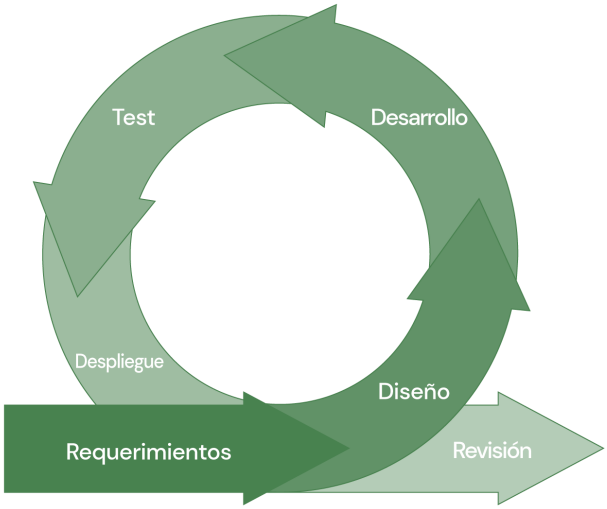


Figura 4.2: Metodología ágil.

Además, las metodologías ágiles fomentan una colaboración más estrecha entre los miembros del equipo y los clientes, promoviendo la comunicación y el *feedback* constante, lo que contribuye a un desarrollo más alineado con las necesidades reales del usuario final. Este enfoque ha demostrado ser especialmente útil en entornos dinámicos donde los requisitos pueden evolucionar rápidamente, proporcionando una ventaja competitiva a las organizaciones que adoptan estas prácticas.

4.4 Enfoque adaptado

Para el desarrollo de este TFG se ha seguido una aproximación personal a una metodología ágil, que combina elementos tanto de enfoques tradicionales como principios de SCRUM [18], adaptándola a nuestras necesidades específicas. Esta metodología ha sido eficaz dada la naturaleza del proyecto y el tamaño del equipo, que consistía en solo dos personas.

Inicialmente, hemos especificado todos los requisitos del proyecto de manera detallada, tal y como se hace en las metodologías tradicionales. Esto nos permitió tener una visión clara y precisa de lo que se necesitaba desarrollar y establecer una base sólida para el trabajo.

Tras definir los requisitos, hemos desglosado el proyecto en tareas específicas y las hemos asignado en base a un cronograma semanal. Esta planificación ágil nos permitió mantener una estructura organizada mientras adaptábamos nuestro enfoque según el progreso y los desafíos encontrados.

El proyecto se dividió en tres *sprints* principales. Cada *sprint* tuvo una duración de aproximadamente tres semanas, con objetivos claros y definidos al inicio de cada ciclo. La estructura de trabajo dentro de cada sprint fue la siguiente:

- ***Sprint* 1:** Durante este *sprint*, nos centramos en la investigación y la definición detallada de los requisitos. Se crearon los primeros esquemas y diagramas necesarios para guiar el desarrollo, y se estableció la arquitectura general del proyecto.
- ***Sprint* 2:** En el segundo *sprint*, se implementaron las principales funcionalidades del proyecto, basándose en los requisitos establecidos previamente. Se trabajó en el de-

sarrollo del núcleo del sistema, asegurándose de que las funcionalidades esenciales estuvieran operativas y en línea con las expectativas del proyecto.

- ***Sprint* 3:** El tercer *sprint* se dedicó principalmente a las pruebas y la optimización del sistema. Durante esta fase, se realizaron pruebas exhaustivas para identificar y corregir errores.

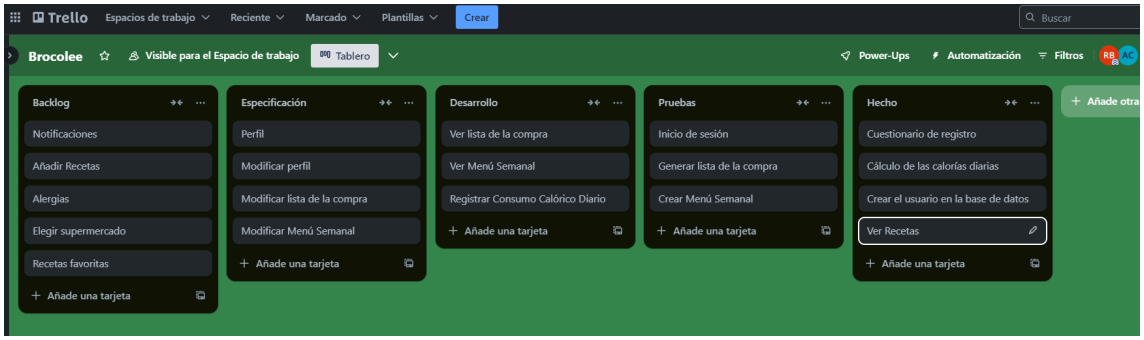


Figura 4.3: Tablero Kanban seguido en el proyecto.

Para mejorar la organización se ha usado Trello [16] para la creación de un tablero Kanban [19] como el mostrado en la Figura 4.3. Como podemos observar se han definido cinco fases en las que se muestra el progreso de la tarea:

1. **Backlog:** En esta columna se recopilan todas las tareas y funcionalidades pendientes de realizar. Es una lista de tareas por hacer que aún no han sido asignadas o priorizadas para el trabajo inmediato. Este es el punto de partida de cualquier tarea.

2. **Especificación:** En esta fase, las tareas que fueron seleccionadas del backlog se detallan aún más, definiendo los criterios de aceptación, los requisitos técnicos y cualquier otra información relevante para asegurar que la tarea esté claramente entendida antes de comenzar el desarrollo.

3. **Desarrollo:** Aquí se lleva a cabo la implementación de las tareas. Las tareas se mueven a esta columna una vez que están listas para ser desarrolladas y se asignan a un miembro del equipo.

4. **Pruebas:** Una vez completado el desarrollo, las tareas pasan a esta fase, donde se realizan pruebas para asegurar que cumplan con los criterios de aceptación y no presenten errores.

5. **Hecho:** Finalmente, las tareas que han superado las pruebas se mueven a esta columna, marcando su finalización.

Para monitorear el avance y coordinar el trabajo, realizamos reuniones todas las semanas. Estas reuniones nos dieron la oportunidad de revisar el progreso, identificar problemas, ajustar tareas y coordinar nuestras actividades, especialmente dado que había dependencias entre las tareas de cada uno de los miembros del equipo. Durante estas reuniones, revisamos el tablero Kanban, actualizamos el estado de las tareas y discutimos cualquier obstáculo que pueda estar afectando el avance. Además, este proceso nos ayudó a mantener una comunicación continua y efectiva, asegurando que todos los miembros del equipo estuvieran alineados y trabajando en las prioridades correctas.

Dado que el proyecto no era muy grande y éramos solo dos personas, este enfoque mixto resultó ser el más eficiente. La planificación inicial y la especificación detallada de

requisitos proporcionaron claridad y dirección, mientras que la asignación de tareas semanales y las reuniones regulares nos permitieron mantener la flexibilidad y adaptarnos a cualquier cambio o problema que surgiera durante el desarrollo.

CAPÍTULO 5
Análisis y especificación de requisitos

5.1 Especificación de requisitos

A continuación, pasaremos a explicar todos los aspectos relativos a la especificación de requisitos de la aplicación **Brocolee**. Para llevar a cabo su desarrollo, ha sido necesario realizar un análisis detallado de los requisitos que la aplicación debía cumplir. Estos requisitos se han dividido en requisitos funcionales y no funcionales. A continuación, se detallan los diferentes aspectos que se han tenido en cuenta y el proceso llevado a cabo:

5.1.1. Público objetivo y sus características

La aplicación está dirigida a personas que tengan entre 18 y 55 años, tanto hombres como mujeres, esto se debe a que este grupo de edad se encuentra en una etapa activa de su vida, es consciente de la importancia de la salud, utiliza activamente la tecnología, tiene capacidad económica para invertir en su bienestar, y está interesado en mantener una rutina de ejercicio complementada con una buena nutrición, lo que hace que sean más propensos a adoptar herramientas digitales que faciliten la gestión de su alimentación y estilo de vida saludable.

Brocolee Tiene como objetivo acercar a los usuarios a unos hábitos alimenticios saludables. Por lo tanto, servirá como punto de partida para los usuarios que tengan interés en el mundo de la nutrición pero, por diversos motivos no puedan acudir a nutricionistas profesionales. Es decir, no se pretende ser un sustituto de los profesionales del sector, sino un puente entre los usuarios que se han adentrado en este mundo y los que están buscando como hacerlo y no saben por donde empezar.

Por lo tanto está destinada a los usuarios que buscan herramientas para planificar y gestionar su dieta, de manera que se simplifique la planificación de las comidas y la compra semanal, mejorando su estilo de vida al acceder a información sobre el valor nutricional de los diferentes alimentos.

5.1.2. Requisitos funcionales

Los requisitos funcionales son especificaciones detalladas que definen las funciones y comportamientos que un sistema, producto o *software* debe cumplir para satisfacer las necesidades del usuario o del negocio. Estos requisitos describen lo que el sistema debe hacer, cómo debe responder a las entradas, y qué resultados o salidas debe generar.

Los requisitos funcionales y los casos de uso que podemos encontrar en el Apéndice A, están estrechamente vinculados en el desarrollo de sistemas: los requisitos funcionales especifican las capacidades y comportamientos que un sistema debe tener, mientras que los casos de uso describen cómo los usuarios interactúan con el sistema para lograr objetivos específicos, detallando la secuencia de acciones necesarias. Los casos de uso se derivan de los requisitos funcionales, proporcionando un contexto práctico y ejemplos que ayudan a validar y clarificar estos requisitos. En esencia, los casos de uso traducen los requisitos funcionales en escenarios concretos y comprensibles, facilitando la comunicación y asegurando que el sistema cumpla con las expectativas de los usuarios.

A continuación, se presentan los principales requisitos funcionales de la aplicación, organizados en función de las funcionalidades que ofrecen a los usuarios:

- **Registro y Autenticación**: La aplicación debe permitir a los usuarios registrarse y autenticarse de manera segura. Esto incluye funciones como el registro de usuarios (ver Tabla 5.1) y el inicio de sesión (ver Tabla 5.2).
- **Gestión de Nutrición:** Un aspecto central de la aplicación es la gestión de la nutrición del usuario. Esto incluye el cálculo de las calorías necesarias diarias (ver Tabla 5.3), la creación de un menú semanal adaptado (ver Tabla 5.7), y la visualización y modificación de este menú (ver Tablas 5.8 y 5.9).
- **Seguimiento y Control**: La aplicación debe permitir a los usuarios registrar su consumo calórico diario (ver Tabla 5.10), acceder a recetas (ver Tabla 5.11), y gestionar una lista de la compra basada en el menú semanal (ver Tabla 5.4).
- **Gestión de Perfil**: Los usuarios deben poder ver y modificar su perfil, incluyendo la actualización de sus datos y el recalculo de sus necesidades calóricas (ver Tablas 5.5 y 5.6).

Concretamente este TFG se centra en el desarrollo e implementación de los requisitos funcionales: **Registrar usuario** (**RF1** Tabla 5.1), **Iniciar sesión** (**RF2** Tabla 5.2), **Cálculo de las calorías necesarias diarias** (**RF3** Tabla 5.3), **Ver lista de la compra** (**RF4** Tabla 5.4), **Ver perfil** (**RF5** Tabla 5.5) y **Modificar perfil** (**RF6** Tabla 5.6).

Requisito	**RF1**
Nombre	Registrar Usuario
Características	Se permite a usuarios no registrados, registrarse en la aplicación creando una nueva cuenta.
Descripción	El nuevo usuario podrá registrarse proporcionando, a través de un breve cuestionario, información como: nombre, correo, contraseña, sexo, ejercicio semanal, altura, peso y edad.
Requisitos no funcionales	RNF1, RNF2, RNF3 y RNF4
Prioridad	Alta

Tabla 5.1: Requisito funcional: Registrar usuario.

Requisito	**RF2**
Nombre	Iniciar sesión
Características	Se permite a un usuario que ya ha sido registrado con anterioridad entrar en el sistema, iniciando sesión.
Descripción	El usuario puede entrar al sistema proporcionando sus credenciales, correo y contraseña.
Requisitos no funcionales	RNF1, RNF2, RNF3, y RNF4
Prioridad	Alta

Tabla 5.2: Requisito funcional: Iniciar sesión.

Requisito	**RF3**
Nombre	Cálculo de las calorías necesarias diarias
Características	Realiza, de forma automática, al terminar el cuestionario del registro, el cálculo de las calorías que un usuario necesita consumir diariamente.
Descripción	El sistema realiza el cálculo establecido a través de los datos proporcionados por el usuario: sexo, nivel de ejercicio físico, altura, peso y edad. De esta manera, el resultado del cálculo se utilizará en el momento en el que se crea el menú semanal, para adaptarlo a las calorías en cuestión.
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4 y RNF5
Prioridad	Alta

Tabla 5.3: Requisito funcional: Cálculo de las calorías necesarias diarias.

Requisito	**RF4**
Nombre	Ver lista de la compra
Características	Permite al usuario ver la lista de la compra
Descripción	El usuario podrá acceder a una lista donde se encontrarán todos los ingredientes que tiene que comprar para poder realizar las recetas del menú semanal.
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4 y RNF5
Prioridad	Alta

Tabla 5.4: Requisito funcional: Ver lista de la compra.

Requisito	**RF5**
Nombre	Ver perfil
Características	Permite al usuario visualizar su perfil.
Descripción	El usuario podrá acceder a su perfil para comprobar los datos.
Requisitos no funcionales	RNF1, RNF2, RNF3 y RNF4
Prioridad	Media

Tabla 5.5: Requisito funcional: Ver perfil.

Requisito	**RF6**
Nombre	Modificar perfil
Características	Permite al usuario modificar su perfil.
Descripción	El usuario podrá acceder a los datos de su perfil y cambiar la contraseña o volver a hacer el cuestionario inicial para recalcular las calorías a consumir.
Requisitos no funcionales	RNF1, RNF2, RNF3 y RNF4
Prioridad	Media

Tabla 5.6: Requisito funcional: Modificar perfil

Requisito	**RF7**
Nombre	Crear menú semanal
Características	Realiza la creación de un menú semanal, adaptado a las necesidades del usuario.
Descripción	El sistema genera al inicio de la semana un menú de manera automática, basándose en las calorías que debe consumir el usuario al día, estas se han calculado previamente en base a los datos del usuario.
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4 y RNF5
Prioridad	Alta

Tabla 5.7: Requisito funcional: Crear menú semanal.

Requisito	**RF8**
Nombre	Ver menú semanal
Características	Permite al usuario acceder al menú semanal y ver todos los platos que lo conforman.
Descripción	El usuario puede ver el menú semanal generado por el sistema, accediendo a los distintos platos que contiene para cada uno de los días de la semana.
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4 y RNF5
Prioridad	Alta

Tabla 5.8: Requisito funcional: Ver menú semanal.

Requisito	**RF9**
Nombre	Modificar menú semanal
Características	Permite modificar los platos que constituyen el menú.
Descripción	El usuario puede cambiar los platos que contiene el menú y sustituirlos por otros diferentes.
Requisitos no funcionales	RNF1, RNF2, RNF3 y RNF4
Prioridad	Media

Tabla 5.9: Requisito funcional: Modificar menú semanal.

Requisito	**RF10**
Nombre	Registrar Consumo Calórico Diario
Características	Permite el registro de las calorías que el usuario ha consumido.
Descripción	El usuario podrá registrar cuando termine una de las comidas y marcarla como "hecha". De esta forma, se sumarán las calorías consumidas y se indicarán las restantes.
Requisitos no funcionales	RNF1, RNF2, RNF3 y RNF4
Prioridad	Media

Tabla 5.10: Requisito funcional: Registrar consumo calórico diario.

Requisito	**RF11**
Nombre	Ver recetas
Características	Permite ver el contenido de las recetas.
Descripción	El usuario podrá acceder a la información de las distintas recetas, las cuales mostrarán: foto, descripción, ingredientes y procedimiento.
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4 y RNF5
Prioridad	Alta

Tabla 5.11: Requisito funcional: Ver recetas.

5.1.3. Requisitos no funcionales

Se trata de las cualidades y restricciones del sistema que no están directamente relacionadas con las funcionalidades específicas, pero que son fundamentales para garantizar su rendimiento, usabilidad y fiabilidad. Estos requisitos establecen los estándares de calidad que el sistema debe cumplir, como la eficiencia, la seguridad y la escalabilidad, asegurando que el sistema no solo funcione correctamente, sino que también lo haga de manera óptima en diferentes condiciones y escenarios.

A continuación, se enumeran los requisitos no funcionales principales para el sistema:

- Compatibilidad con las últimas versiones de Android (ver Tabla 5.12).
- Un nivel de disponibilidad del 99.9 % del tiempo (ver Tabla 5.14).
- Facilidad de mantenimiento para actualizaciones y correcciones (ver Tabla 5.15).
- Garantía de tiempos de respuesta adecuados para operaciones críticas (ver Tabla 5.16).

Número de requisito	**RNF1**
Descripción	Se requiere compatibilidad de la aplicación con las últimas dos versiones principales de Android.
Prioridad	Alta

Tabla 5.12: Requisito no funcional RNF1

Número de requisito	**RNF2**
Descripción	El sistema debe cumplir con las regulaciones de protección de datos de la Unión Europea.
Prioridad	Alta

Tabla 5.13: Requisito no funcional RNF2

Número de requisito	**RNF3**
Descripción	La aplicación debe estar disponible 99.9 % del tiempo, con un tiempo de inactividad máximo de 8.76 horas al año.
Prioridad	Alta

Tabla 5.14: Requisito no funcional RNF3

Número de requisito	**RNF4**
Descripción	La aplicación debe ser fácilmente mantenible, permitiendo la implementación de actualizaciones y correcciones de errores sin interrumpir significativamente el servicio.
Prioridad	Alta

Tabla 5.15: Requisito no funcional RNF4

Número de requisito	**RNF5**
Descripción	La aplicación debe garantizar un tiempo de respuesta adecuado, asegurando que las operaciones críticas se realicen en un tiempo aceptable para no afectar negativamente la experiencia del usuario.
Prioridad	Media

Tabla 5.16: Requisito no funcional RNF5

5.1.4. Restricciones y limitaciones tecnológicas

Compatibilidad de sistemas operativos

El proyecto está diseñado para ser una aplicación móvil, pero enfrenta algunas limitaciones importantes. Esto se debe a que solo es compatible con dispositivos que utilicen el sistema operativo Android, lo que deja fuera a aquellos que usan otros sistemas operativos, como iOS. Además, Brocolee se está desarrollando con la versión más reciente de **Android Studio**, lo que podría causar problemas de compatibilidad con algunas versiones de Android, en especial con las que son más antiguas.

Limitación de Acceso y Despliegue de la Base de Datos

El acceso e implementación de la base de datos en este proyecto plantea ciertas limitaciones que deben tenerse en cuenta. Esto se debe a que la base de datos está desplegada en un servidor local, para el proceso de desarrollo y pruebas, por lo tanto su acceso se reduce a entornos controlados.

Además, como la base de datos no está alojada en un servidor en la nube, su disponibilidad y accesibilidad se ven limitadas geográficamente. Por el momento, solo es posible acceder desde la misma red. Para garantizar que la aplicación pueda escalar adecuadamente y ser más confiable en un entorno de producción, sería necesario migrar la base de datos a un servidor en la nube, lo que permitiría a los usuarios acceder a ella siempre que tengan acceso a Internet.

Finalmente, como el acceso a la base de datos se realiza a través de conexiones directas, se debería mejorar la seguridad de los datos, ya que, si no se implementan medidas adicionales, supondría un riesgo de seguridad. Para poner solución a este problema se debería incorporar cifrado de datos, tanto en tránsito como en reposo, y establecer políticas estrictas de control de acceso para proteger la información almacenada en la base de datos.

5.1.5. Modelo de casos de uso

En el desarrollo de sistemas de *software*, es especialmente importante tener una representación clara y comprensible de cómo los diferentes componentes del sistema interactúan entre sí y con los actores externos. Para lograr esto, se utilizan herramientas visuales como el **Diagrama de Contexto** y el **Diagrama de Casos de Uso**. Estas herramientas son esenciales para capturar y comunicar los requisitos funcionales y operacionales del sistema, facilitando una visión general y detallada de sus funcionalidades.

Diagrama de contexto

Un diagrama de contexto es una herramienta visual que representa de manera simplificada un sistema y sus interacciones con entidades externas, como usuarios, otros sistemas u organizaciones. Este diagrama es fundamental para comprender cómo los actores externos y componentes interactúan con la aplicación, mostrando de manera clara los límites del sistema y los flujos de información entre estos. En la Figura 5.1, se destacan los siguientes actores clave:

1. **Administrador**: Responsable de la gestión de usuarios y de los datos extraídos.
2. **Usuario no registrado**: Interactúa con el sistema con acceso limitado.
3. **Usuario registrado:** Tendrá acceso completo a las funcionalidades del sistema.
4. **Reloj:** Utilizado para la sincronización y control de tiempo dentro del sistema.
5. **API de ChatGPT:** Procesará datos en lenguaje natural para mejorar la interacción del usuario con el sistema.

Este diagrama facilita una visión integral de las interacciones y responsabilidades de cada actor, proporcionando un panorama claro de la funcionalidad general del sistema y su comunicación con el entorno.

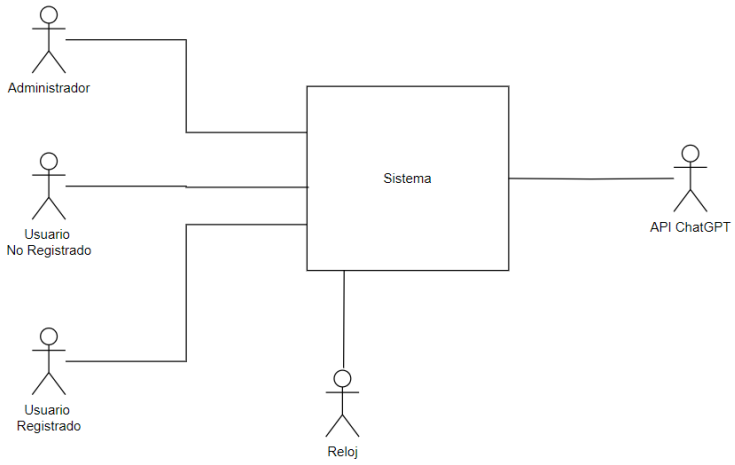


Figura 5.1: Diagrama de contexto.

Diagrama de casos de uso

El diagrama de casos de uso es fundamental en la fase de análisis y diseño de *software*, ya que ayuda a entender los requisitos funcionales del sistema, identificando quién hace qué y cómo interactúan los diferentes componentes del sistema. Es importante para comprender cómo se relacionan los usuarios con las diferentes funcionalidades que ofrece **Brocolee**. Cada caso de uso explica los pasos necesarios para cubrir una funcionalidad especifica. Además, se detallan las posibles alternativas que los usuarios pueden seguir y los errores que podrían surgir durante el proceso. Esto permite no solo visualizar

cómo deberían funcionar las interacciones en situaciones ideales, sino también cómo el sistema maneja excepciones o problemas. El objetivo es obtener una visión clara de las expectativas de los usuarios y garantizar que el sistema cumpla con sus necesidades.

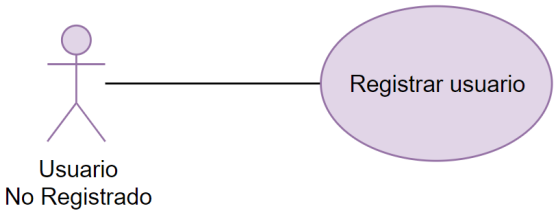


Figura 5.2: Caso de uso de Usuario No Registrado.

En la Figura 5.2 se puede observar el diagrama de casos de uso, en el que el actor **Usuario No Registrado** es el que interactúa con la aplicación para registrar una nueva cuenta. Se trata del proceso de **Registrar usuario** en el sistema, la definición de este caso de uso se encuentra en la Tabla A.1.

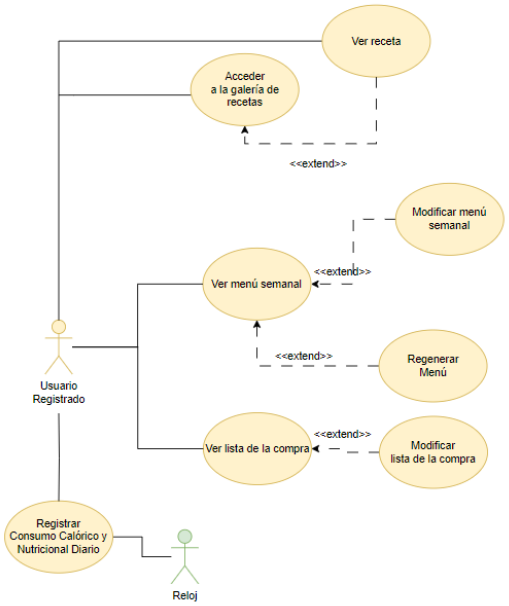


Figura 5.3: Caso de uso de Usuario Registrado.

En el caso de la Figura 5.3 se trata del diagrama de casos de uso del **Usuario registrado**, este actor tiene acceso a funciones que permiten gestionar su alimentación y mejorar su experiencia a través de la personalización del menú y de la aplicación. Un **Usuario Registrado** puede: visualizar su menú semanal (Tabla A.23), registrar su consumo calórico y nutricional diario (Tabla A.12), modificar el menú (Tabla A.6) y la lista de la compra (Tabla A.8), crear manualmente un menú semanal (Tabla A.20), acceder a la galería de recetas (Tabla A.10), ver las recetas y su información (Tabla A.25), añadir recetas (Tabla A.11) y configurar las notificaciones (Tabla A.14). Todas las interfaces las puede realizar a través de interfaces intuitivas y fáciles de usar.

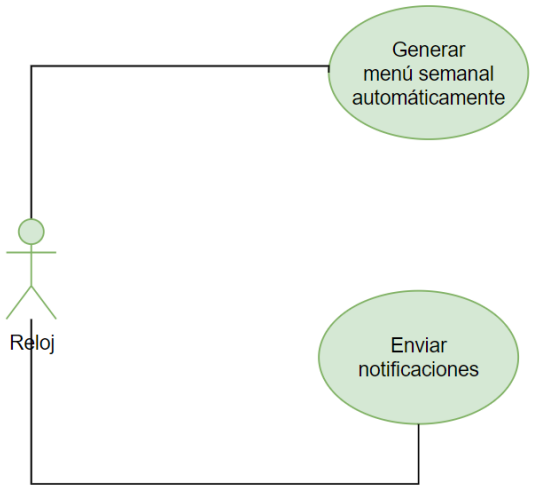


Figura 5.4: Casos de uso de Reloj.

La Figura 5.4 muestra los casos de uso que se realizan de manera automática por el sistema con un reloj programado. Los casos que se hacen de manera automática son: la generación automática del menú semanal (Tabla A.5) y el envío de notificaciones (Tabla A.13). De esta manera el usuario puede obtener un menú cada semana, siendo notificado en el mismo momento.

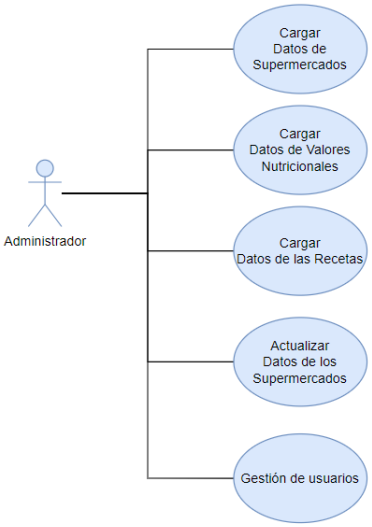


Figura 5.5: Caso de uso Administrador.

En cuanto a los casos de uso relativos a el **Administrador**, se muestran en la Figura 5.5. Estos casos describen la gestión de los datos como la carga de datos de valores nutricionales (Tabla A.16), la carga de datos de supermercados (Tabla A.15), la carga de datos de recetas (Tabla A.17), la actualización de datos de los supermercados (Tabla A.18) y la gestión de usuarios (Tabla A.24). De esta forma el administrador podrá actualizar los datos para que los usuarios reciban una mejor experiencia al usar **Brocolee**.

5.1.6. Diagrama de clases

El diagrama de clases es una manera de representar de forma abstracta y conceptual cómo se organizan y relacionan los diferentes elementos dentro de un sistema. Ofrece una visión general de cómo estos componentes clave están relacionados y cómo interactúan dentro del sistema de la aplicación de nutrición. Es útil para planificar y entender cómo se debe construir y organizar la aplicación sin entrar en detalles técnicos específicos. En este proyecto, el modelo de dominio se enfocará en los elementos principales que forman parte de la aplicación, como usuarios, recetas, menús, productos y listas de compras.

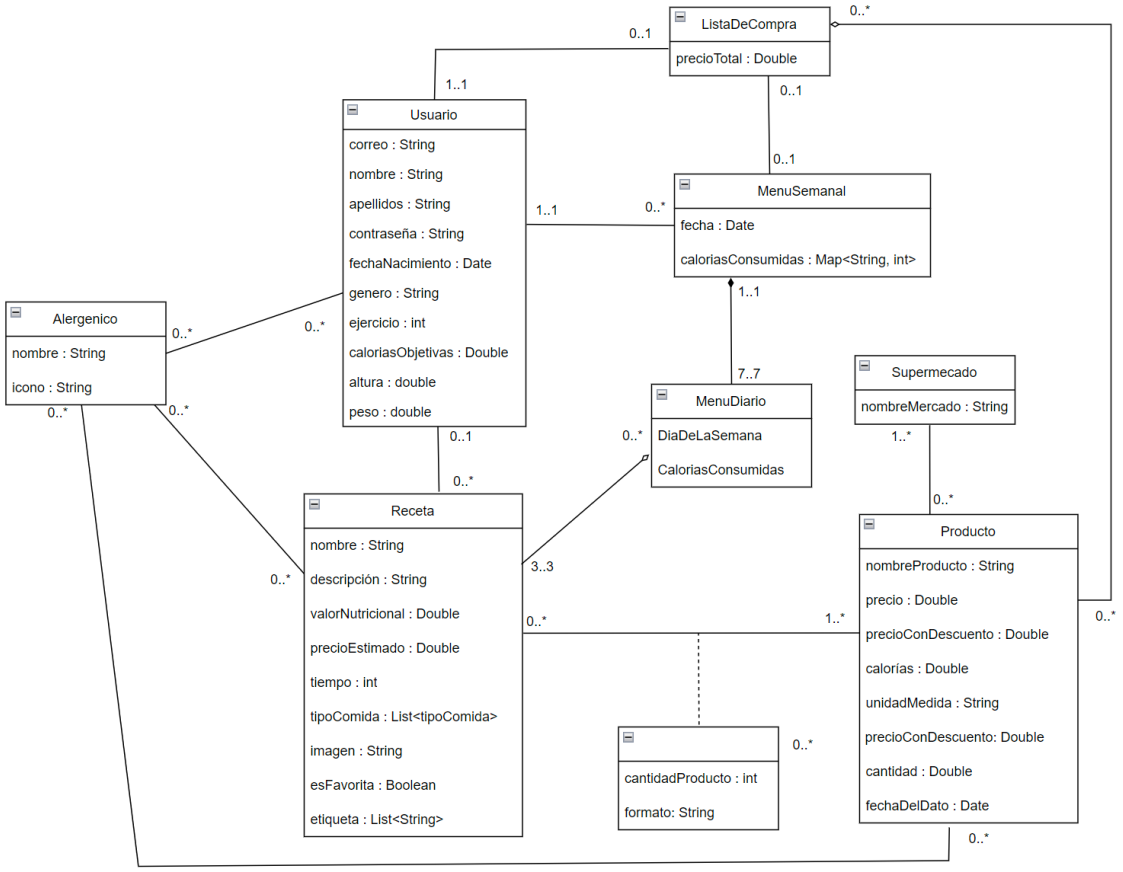


Figura 5.6: Diagrama de clases.

En la Figura 5.6 se puede observar el modelo de dominio de **Brocolee**. A continuación se describen las principales entidades y sus relaciones:

- **Usuario:** Representa a las personas que utilizan la aplicación. Sus atributos incluyen correo, nombre, apellidos, contraseña, fecha de nacimiento, género, nivel de ejercicio, calorías objetivo, altura y peso. Además, cada usuario puede tener asociado un Menú Semanal y una Lista de Compra.

- **Receta:** Define las recetas disponibles en la aplicación, con atributos como nombre, descripción, valorNutricional, precioEstimado, tiempo, tipoComida, imagen, esFavorita y etiqueta. Una receta puede tener múltiples Alergenicos y Productos.

- **MenuSemanal:** Contiene el menú planificado para la semana actual, con atributos como fecha y caloriasConsumidas. Cada MenuSemanal se compone de siete MenuDiario.

- **MenuDiario:** Define el menú para cada día de la semana actual, con atributos como DiaDeLaSemana y CaloriasConsumidas. Cada MenuDiario incluye tres Recetas (desayuno, comida y cena).
- **ListaDeCompra:** Contiene la lista de productos necesarios para realizar las recetas que aparecen en cada uno de los menús diarios de la semana actual, con un atributo precioTotal.
- **Producto:** Son los productos que se encuentran disponibles en los supermercados que incluye la base de datos, con atributos como nombreProducto, precio, precioConDescuento, calorias, unidadMedida, cantidad y fechaDelDato.
- **Supermercado:** Se trata de los supermercados, cada uno contendrá Productos especificos de ese supermercado.
- **Alergenico:** Son los alérgenos que contienen las recetas, cada uno puede asociarse a varias recetas, tienen atributos como nombre e icono.

En resumen, el modelo de dominio actúa como un mapa detallado que asegura que la aplicación se desarrolle de manera ordenada y eficiente, alineando la estructura de datos con las funcionalidades requeridas.

5.2 Análisis de requisitos

5.2.1. Priorización de requisitos

La priorización de requisitos es una práctica importante en el desarrollo de *software* que ayuda a organizar y gestionar de manera eficiente los distintos elementos necesarios para construir una aplicación. Esta práctica asegura que las funcionalidades más importantes se implementen primero, permitiendo un desarrollo ordenado y un lanzamiento más efectivo. A continuación, se explica cómo se clasifica esta priorización en alta, media y baja prioridad:

- **Alta Prioridad**: Estos requisitos son fundamentales para el funcionamiento básico de la aplicación. Son las funcionalidades que deben estar presentes desde el principio para que la aplicación pueda operar de manera efectiva y ofrecer un valor inmediato a los usuarios. Por ejemplo: el registro de usuarios, inicio de sesión, cálculo de calorías necesarias, creación de menús semanales y visualización de estos menús.
- **Media Prioridad**: Estos requisitos mejoran la experiencia del usuario y añaden valor adicional, pero no son fundamentales para el funcionamiento inicial de la aplicación. Incluyen características que enriquecen la aplicación una vez que las funcionalidades básicas ya están en funcionamiento. Por ejemplo: la modificación de menús, el registro de consumo calórico diario, la visualización y modificación de perfiles, y la gestión de listas de compra.
- **Baja Prioridad**:Son funcionalidades adicionales que no son esenciales para el lanzamiento inicial, pero que pueden mejorar la aplicación y proporcionar características adicionales. Por ejemplo: atajos para usuarios avanzados o de conveniencia u opciones de personalización. Estas funcionalidades se implementan después de haber completado los requisitos de alta y media prioridad, y sirven para añadir valor adicional y diferenciar la aplicación en el mercado.

En resumen, priorizar los requisitos ayuda a organizar el trabajo de manera efectiva, asegurando que la aplicación sea funcional y útil desde el principio, mientras se añaden gradualmente características adicionales que mejoran la experiencia del usuario.

5.2.2. Dependencias entre requisitos

La identificación de las dependencias entre requisitos es fundamental para asegurar un desarrollo coherente y funcional de la aplicación. Algunos requisitos necesitan ser implementados antes que otros debido a su interrelación y necesidad de datos o funcionalidades previas.

- **RF1: Registrar Usuario** (Tabla 5.1)
  Este es el requisito inicial que permite la creación de cuentas de usuario, esencial para cualquier otra interacción con el sistema.
- **RF2: Iniciar Sesión** (Tabla 5.2)
  Depende del requisito RF1, ya que los usuarios necesitan estar registrados antes de poder iniciar sesión.
- **RF3: Cálculo de Calorías Necesarias Diarias** (Tabla 5.3)
  Requiere los datos ingresados durante el registro de usuario (RF1) para realizar cálculos precisos sobre las necesidades calóricas.
- **RF4: Ver Lista de la Compra** (Tabla 5.4)
  Depende de la creación del menú semanal (RF7), ya que la lista de la compra se genera a partir de los ingredientes del menú.
- **RF5: Ver Perfil** (Tabla 5.5)
  Requiere que los usuarios estén registrados (RF1) e iniciados sesión (RF2) para poder acceder a sus perfiles.
- **RF6: Modificar Perfil** (Tabla 5.6)
  Depende de la capacidad de ver el perfil (RF5) y permite a los usuarios actualizar su información personal y preferencias.
- **RF7: Crear Menú Semanal** (Tabla 5.7)
  Necesita el cálculo de calorías diarias (RF3) para generar menús que se adapten a las necesidades del usuario.
- **RF8: Ver Menú Semanal** (Tabla 5.8)
  Depende de la generación del menú semanal (RF7), permitiendo a los usuarios visualizar los menús creados.
- **RF9: Modificar Menú Semanal** (Tabla 5.9)
  Requiere la funcionalidad de ver (RF8) y crear menús (RF7) para permitir la modificación de los mismos.
- **RF10: Registrar Consumo Calórico Diario** (Tabla 5.10)
  Depende de la existencia de un menú semanal (RF7) para que los usuarios puedan registrar su consumo calórico basado en los menús.
- **RF11: Ver Recetas** (Tabla 5.11)
  Necesita que las recetas estén almacenadas y disponibles para que los usuarios puedan acceder a ellas.

CAPÍTULO 6
Diseño

En el desarrollo de *software*, el diseño es una etapa fundamental que influye directamente en la usabilidad, funcionalidad y éxito general de la aplicación. Este capítulo aborda el proceso de diseño que ha guiado la creación y desarrollo de la aplicación.

6.1 Herramientas de diseño

- **Figma:** Herramienta de diseño de interfaz de usuario (UI) [20], en la cual se han creado los prototipos interactivos de las interfaces a implementar en la aplicación. Esto resulta especialmente útil para realizar pruebas de usabilidad y poder validar los diseños.

  Figma cuenta con características esenciales para facilitar el desarrollo *software*. Una de ellas es la integración de diversos *plugins*, que agilizarán la realización de tareas y la búsqueda de recursos. Un ejemplo de uno de los *plugins* usados sería ***Android Vector Drawable*** [21], que permite la transformación de los diseños y objetos a lenguaje XML (*Extensible Markup Language*). Esta conversión es necesaria para los recursos que se utilizan en las aplicaciones de **Android Studio**, ya que se almacenan como *drawables*, en lenguaje XML. Otro ejemplo sería el *plugin* ***SVG Repo - Free Icons and Vectors***[1], que se trata de un repositorio de iconos de licencia abierta en formato SVG (*Scalable Vector Graphics*).

- **Draw.io:**

  Aplicación en linea para la creación de todo tipo de diagramas [22], desde diagramas UML (*Unified Modeling Language*), de flujo, organigramas, etc.

  Esta se caracteriza por su facilidad para la representación visual de ideas y estructuras de datos, gracias que ofrece una gran variedad de formas e iconos predefinidos. Además, no solo permite guardar y sincronizar diagramas con **Google Drive**, **OneDrive**, **Dropbox**, y **GitHub**, sino que también permite la colaboración en linea de diferentes usuarios editando simultáneamente.

6.2 Diseño de Base de datos

Una base de datos es un sistema organizado para almacenar, gestionar y acceder a datos de manera eficiente. Utiliza una estructura de tablas, donde la información se organiza en filas y columnas, lo que permite un manejo ordenado y rápido. Un diseño

[1] Todos los SVG obtenidos desde SVG Repo.

eficiente y bien estructurado de la base de datos es esencial para asegurar que los datos se gestionen de manera segura, se acceda a ellos rápidamente y se mantenga la integridad referencial entre las diferentes entidades.

A continuación se describen los pasos necesarios para crear una tabla y proporcionar una representación gráfica de la estructura de la base de datos. Este diseño proporciona una base sólida para todas las operaciones de almacenamiento y recuperación de datos y es clave para el rendimiento y la escalabilidad de la aplicación.

6.2.1. Creación de Tablas:

Tras elaborarse un modelo de datos se diseñaron las tablas que conforman la base de datos, teniendo en cuenta que contases con las claves primarias y ajenas necesarias en cada caso para representar las relaciones entre las diferentes tablas y elementos.

A través de MySQL Workbench y teniendo en cuenta el modelo de dominio representado en la Figura 5.6, se crearon las siguientes tablas:

- **Alergenico:** Tabla que almacena la información de los alérgenos disponibles en la aplicación, incluyendo el nombre e icono.
- **imagen_producto:** Tabla dedicada al almacenamiento de las imágenes asociadas a los productos.
- **MenuDiario:** Tabla que contiene la información sobre los menús diarios, incluyendo el día de la semana y las calorías consumidas.
- **MenuSemanal:** Almacena los menús semanales creados por los usuarios, vinculados a los menús diarios.
- **Producto:** Tabla que contiene los detalles de los productos disponibles en los supermercados, como nombre, precio, calorías y formato.
- **Receta:** Almacena las recetas disponibles, incluyendo información como nombre, descripción, valor nutricional, tipo de comida, entre otros.
- **Receta_Etiqueta:** Tabla intermedia que gestiona la relación entre recetas y etiquetas, permitiendo clasificar y organizar las recetas según diferentes categorías.
- **Receta_Producto:** Tabla intermedia que vincula recetas con productos, especificando las cantidades y formatos utilizados en cada receta.
- **Supermercado:** Almacena información sobre los supermercados asociados, como su nombre y otros detalles relevantes.
- **Usuario:** Tabla que almacena la información de los usuarios registrados en la aplicación, incluyendo datos personales, preferencias alimentarias, y metas nutricionales.
- **Usuario_alergenico:** Tabla intermedia que vincula usuarios con alérgenos, permitiendo personalizar las recomendaciones de recetas y productos según las alergias de cada usuario.

6.2.2. Representación Gráfica del Diseño de la Base de Datos:

La Figura 6.1 muestra el diagrama de diseño de la base de datos, en el cual se representan visualmente las tablas y sus interrelaciones. Este diagrama es una herramienta

esencial para comprender la estructura del sistema, ya que muestra cómo las distintas tablas están conectadas entre sí y cómo se relacionan los datos entre ellas, proporcionando una visión clara de cómo se organiza y gestiona la información dentro del sistema.

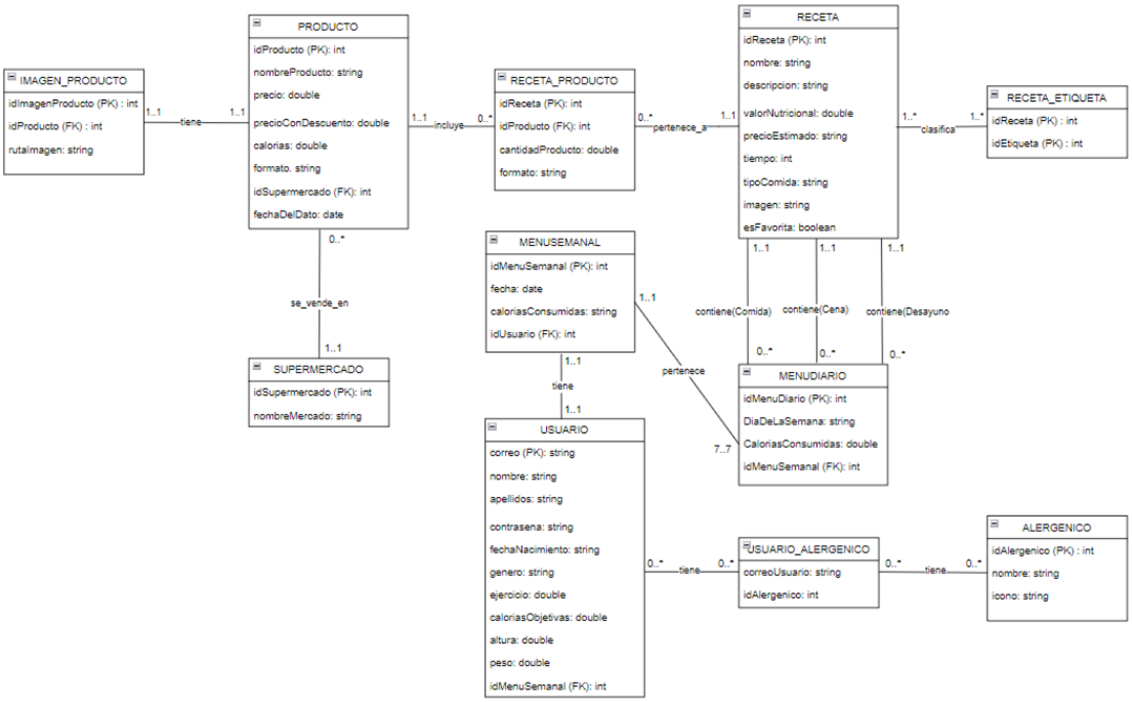


Figura 6.1: Diagrama de la base de datos

6.3 Diseño de interfaces

En esta sección encontramos las interfaces de la aplicación y el proceso de diseño por el que ha pasado cada una de ellas. En cada uno de los casos podemos ver tres componentes, que representan las distintas fases necesarias para conseguir cada una de las interfaces finales.

El primer componente es el **boceto** realizado a mano alzada, este sirve para crear una idea de cómo funcionaría la aplicación y comprobar que todos los requisitos funcionales y casos de uso se cumplen.

En el segundo componente nos encontramos los ***mockups***, diseñados en Figma a partir de los bocetos. En este caso, se trata de una versión más realista de la interfaz, refinando los detalles y dotando de una identidad a la aplicación. Además, tiene especial importancia, ya que a través del *plugin* ***Android Vector Drawable*** [21] obtendremos los *drawables* que se utilizarán en Android Studio en la implementación final de las interfaces.

Finalmente, en la última tenemos una captura de la aplicación en la que se muestra la **implementación** de las interfaces.

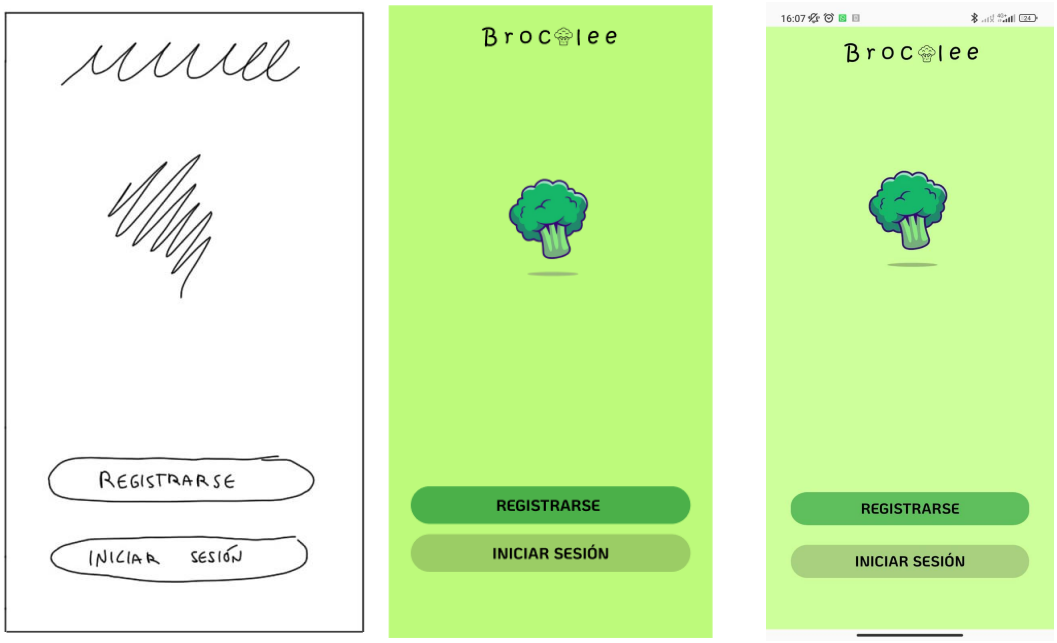


Figura 6.2: Proceso de diseño interfaz inicio.

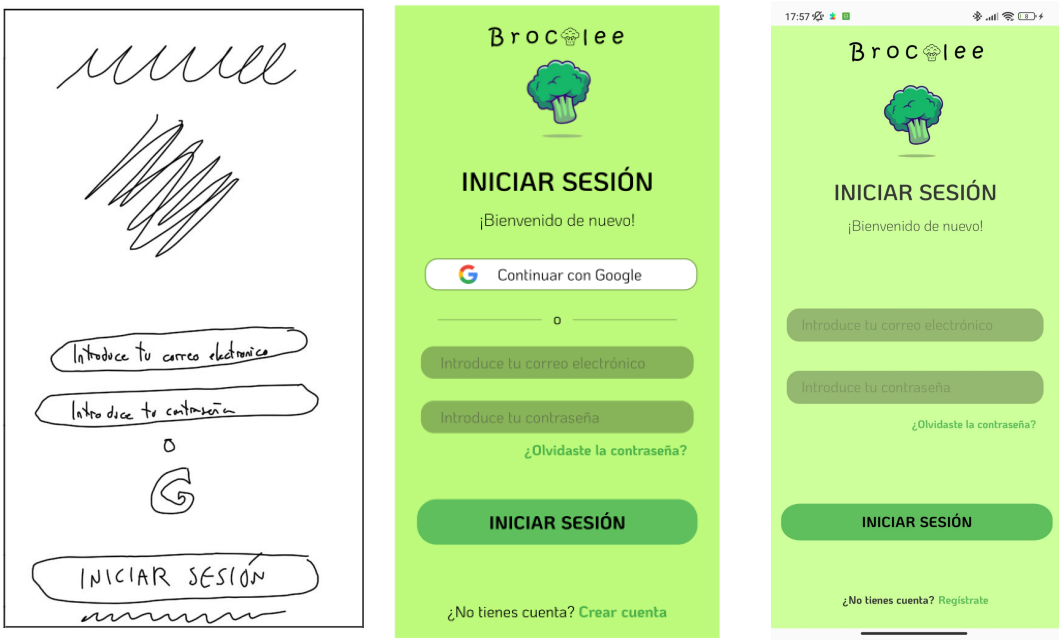


Figura 6.3: Proceso de diseño interfaz inicio de sesión.

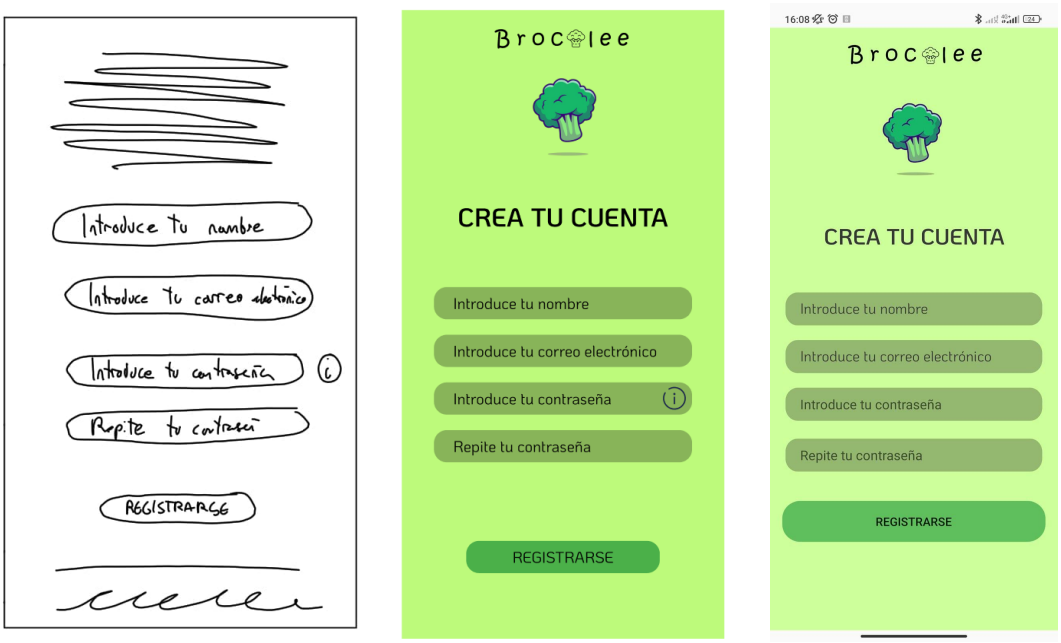


Figura 6.4: Proceso de diseño interfaz del cuestionario de registro: introducción de datos personales.

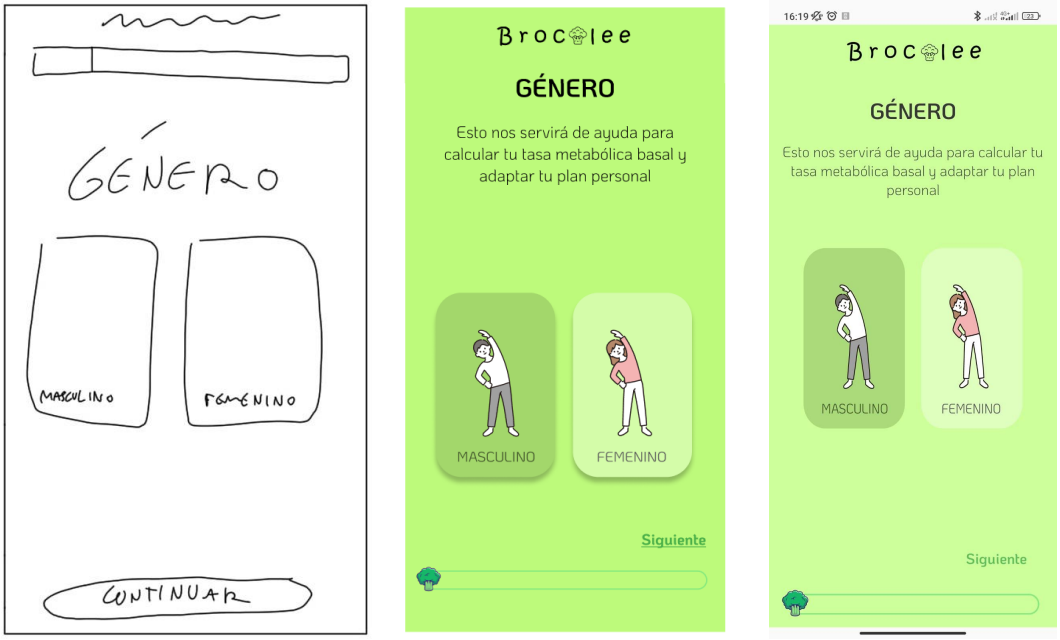


Figura 6.5: Proceso de diseño interfaz del cuestionario de registro: introducción del género.

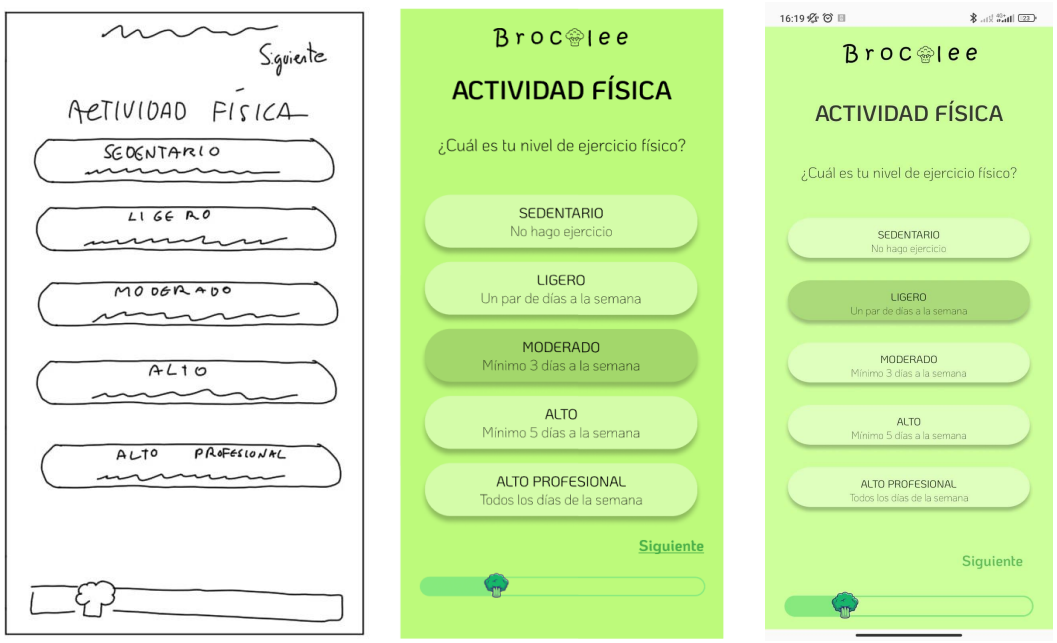


Figura 6.6: Proceso de diseño interfaz del cuestionario de registro: introducción de la actividad física.

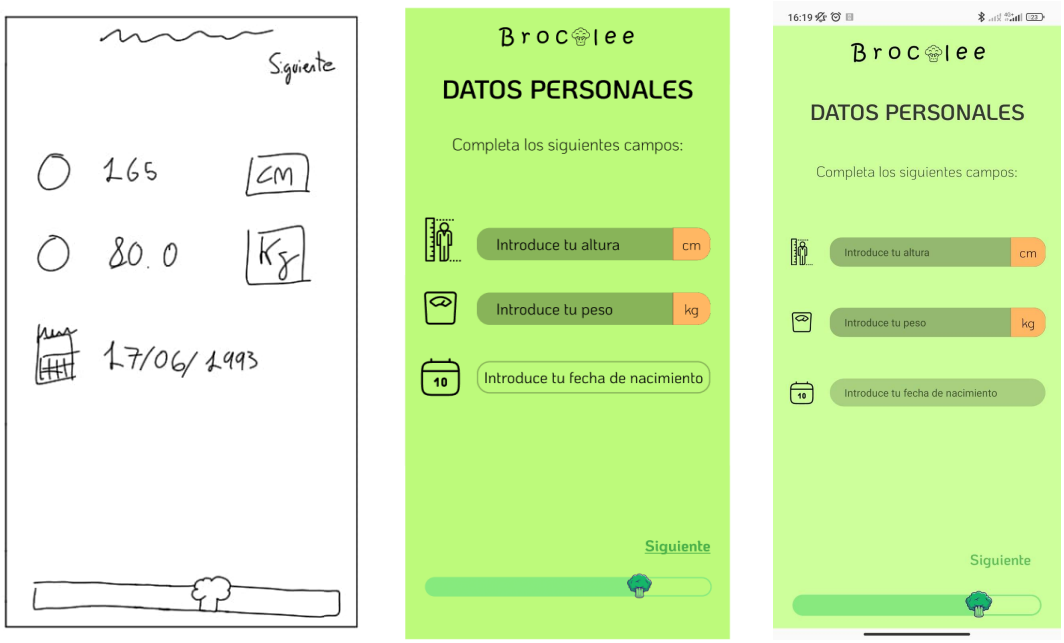


Figura 6.7: Proceso de diseño interfaz del cuestionario de registro: introducción de los datos físicos.

Figura 6.8: Proceso de diseño interfaz del cuestionario de registro: Bienvenida e información para el usuario.

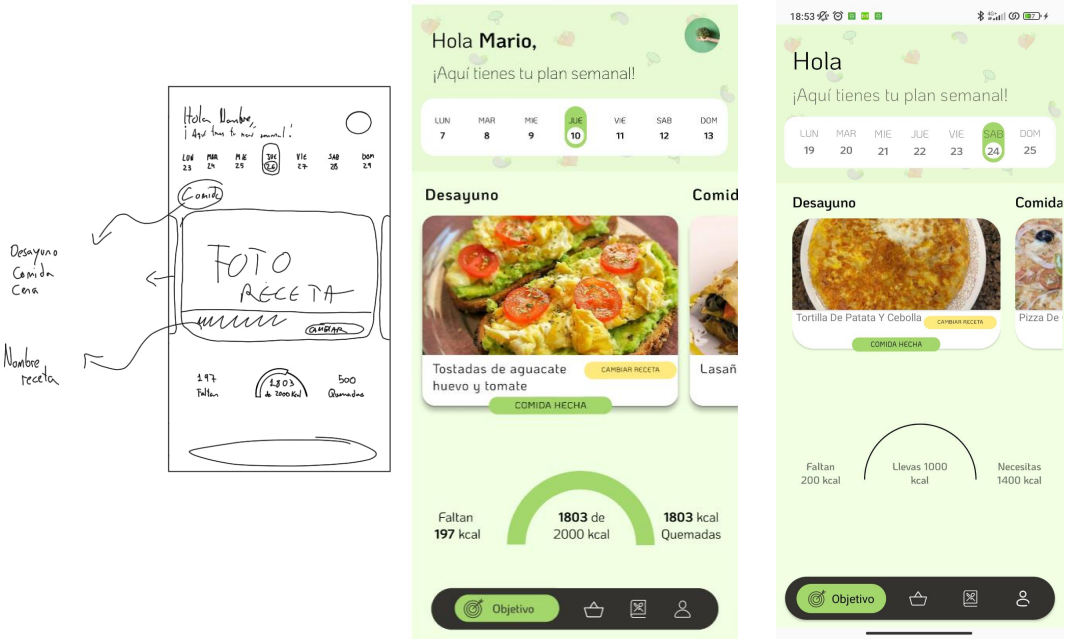


Figura 6.9: Proceso de diseño interfaz del objetivo.

Figura 6.10: Proceso de diseño interfaz de la lista de la compra.

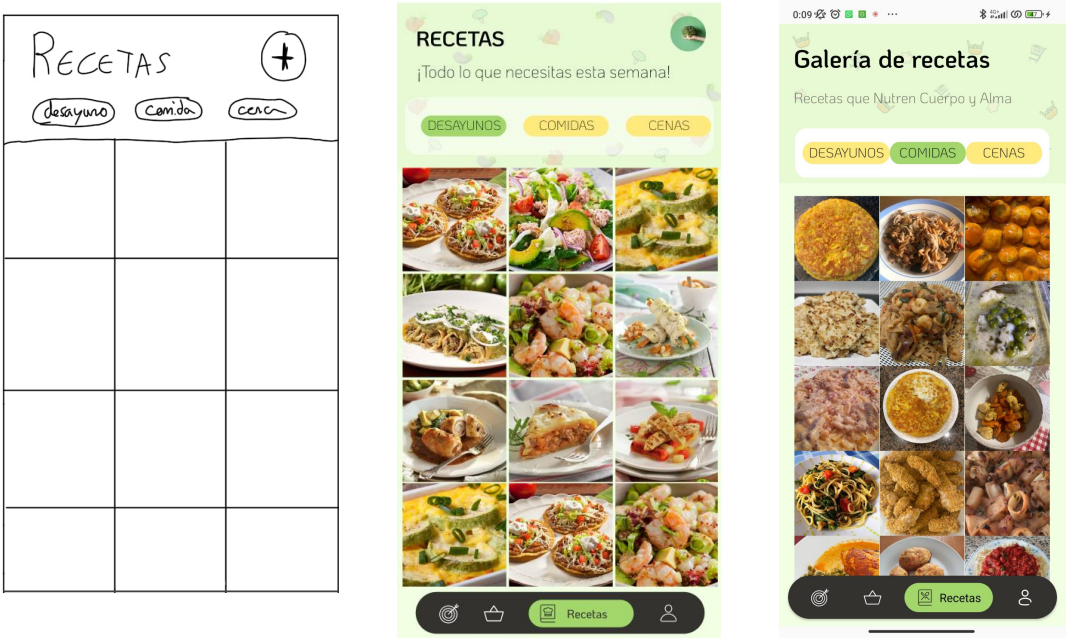


Figura 6.11: Proceso de diseño interfaz de la galería de recetas.

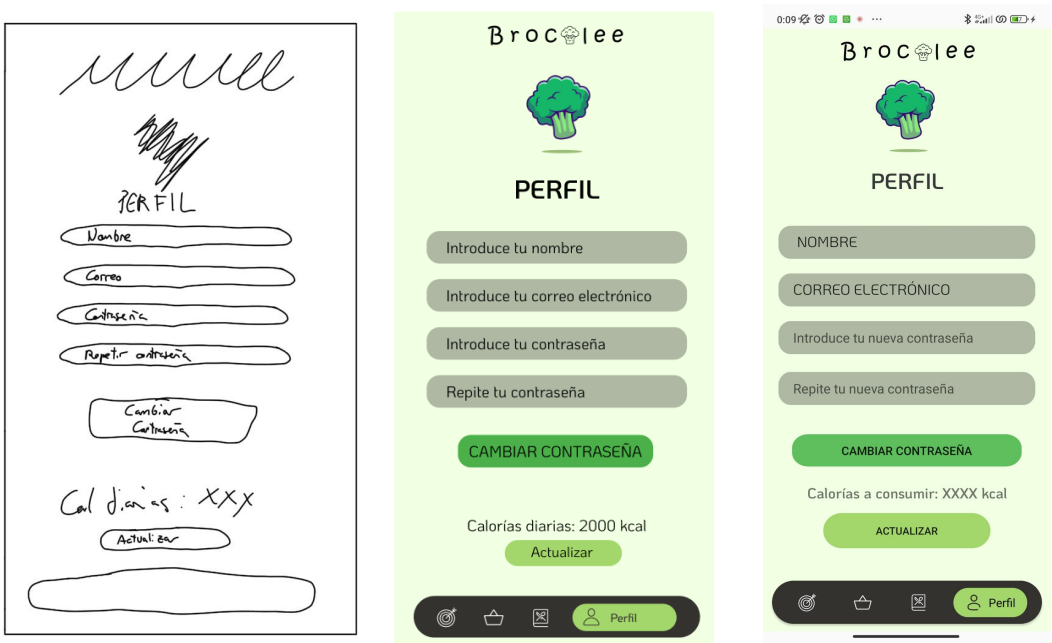


Figura 6.12: Proceso de diseño interfaz del perfil.

CAPÍTULO 7
Desarrollo e implementación

Este TFG se centra en el desarrollo e implementación de dos complejas e importantes características que forman parte de la aplicación. También incluye el diseño e implementación de las interfaces.

Con el fin de comprender las soluciones propuestas para la implementación de las diferentes características, así como el funcionamiento de Android Studio, es necesario explicar los siguientes conceptos:

- **Activity:** es uno de los componentes principales en una aplicación Android. Representa una sola pantalla con una interfaz de usuario (UI). Las aplicaciones suelen tener varios Activities, con los que el usuario podrá navegar entre las diferentes pantallas.
- **Fragment:** es una porción reutilizable de la interfaz de usuario que puede formar parte de una Activity, que permite una mayor flexibilidad y organización del código.
- **Intent:** Un Intent en Android Studio es un objeto utilizado para enviar mensajes entre diferentes componentes de una aplicación, como actividades, servicios y receptores de *broadcast*. Los Intents permiten la comunicación dentro de una misma aplicación o incluso entre aplicaciones diferentes.
- **Adaptadores:** un adaptador es una clase que actúa como intermediario entre un conjunto de datos y una vista en la interfaz de usuario. Su propósito principal es tomar datos de una fuente, como una base de datos o una lista, y convertirlos en una forma que se pueda mostrar en una vista, como un *ListView*, *GridView*, o *RecyclerView*.

7.1 Herramientas de desarrollo

Para desarrollar esta aplicación se ha recurrido a diferentes herramientas y tecnologías que han facilitado su creación y han asegurado una experiencia de usuario de alta calidad. Entre estas herramientas y tecnologías se encuentran:

- **Android Studio:** Como entorno de desarrollo integrado (IDE) para la creación de la aplicación se ha elegido **Android Studio** [23]. Esta decisión se debe a que además de ser una herramienta especifica para el desarrollo de aplicaciones en dispositivos Android, también cuenta con herramientas de edición de código, depuración, pruebas y empaquetado.

Destacando con características como un emulador Android en el que se pueden hacer pruebas con una amplia variedad de dispositivos, sin necesidad de hardware físico. Al mismo tiempo, también permite el monitoreo y el análisis del rendimiento y el uso de recursos, a través de **Android Profiler**. De esta manera, facilita la detección de problemas al poder averiguar que pantallas o llamadas son las que pueden estar saturando la CPU, la memoria o la red.

- **MySQL Workbench:** Se ha utilizado **MySQL Workbench** [24] como gestor de la base de datos. Esta aplicación es una herramienta visual empleada para el modelado, gestión y generación de la base de datos.

  Destaca por características como la facilidad para crear y visualizar esquemas de bases de datos de manera gráfica, además de contar con herramientas para gestionar usuarios, realizar copias de seguridad, restaurar bases de datos y monitorizar el rendimiento del servidor **MySQL**. También es relevante la existencia de un editor de consultas que permite escribir y ejecutar código **SQL**, optimizando las consultas.

- **AWS RDS (*Amazon Web Services Relational Database Service*):** Servicio de base de datos en la nube que proporciona escalabilidad y fiabilidad [25].

  Entre las características destacadas de **AWS RDS** se encuentran su capacidad de escalabilidad y su fiabilidad, ya que permite ajustar los recursos de la base de datos de forma flexible, adaptándose a las necesidades de las aplicaciones, ya sea aumentando la capacidad de almacenamiento o añadiendo réplicas de lectura para mejorar el rendimiento.

Lenguajes de programación

- **Java:** Este es el principal lenguaje de programación utilizado en el desarrollo de la aplicación, seleccionado por su extensa disponibilidad de bibliotecas y *frameworks*, lo que lo convierte en una opción ideal para crear aplicaciones, incluidas las móviles, de manera más eficiente.

- **Kotlin:** En el caso de Kotlin, se ha utilizado en ciertos módulos para los cuales existían más facilidades de implementación que en **Java**. **Kotlin**, es totalmente interoperable con **Java**, además ofrece una sintaxis más sencilla y concisa, lo que permite escribir un código más limpio y menos propenso a errores. Esto ha llevado a que se utilice en módulos donde se requiere una mayor productividad y menor cantidad de código repetitivo, optimizando el desarrollo en comparación con **Java**.

7.2 Cálculo de las calorías diarias

La aplicación pretende ofrecer un menú semanal al usuario, de manera que esté adaptado a sus necesidades. Por ello, es necesario conocer cuál es la ingesta calórica que se debe sugerir al usuario. Tras el estudio de las distintas alternativas para llevar a cabo este cálculo, explicadas en la Sección 2.1.4, se ha tomado la decisión de hacer uso del **Método de Harris-Benedict** con la adaptación de **Miffin y otros** que nos deja con las ecuaciones de la Tabla 2.3, con los factores de multiplicación basados en el ejercicio físico que aparecen en la Tabla 2.4.

Esta decisión se debe a que, en comparación con los demás métodos, este es el más completo, ya que es el que más factores tiene en cuenta (sexo, edad, altura, peso y ejercicio físico), en contraste al Método FAO/OMS/UNU (peso, sexo, edad y ejercicio físico) y al Método Rápido (peso).

Además, con el objetivo de realizar una adaptación de las calorías a consumir, el peso que se tiene en cuenta en la fórmula dependerá de las necesidades calóricas que el usuario tenga. Para ello, se seguirán los siguientes pasos:

1. Partiendo del *peso actual* del usuario, se calculará su IMC.
2. Se comparará con el IMC ideal (entre 18.5 y 24.9) que debería tener.
    a) Si este está por debajo de 18.5, el peso que se utilizará para el cálculo será el *peso ideal*, que se calcula con las Ecuaciones 2.2 y 2.3.
    b) Si está dentro del rango del IMC ideal, el peso que se usará será el *peso actual*.
    c) Si está por encima del IMC ideal, significando que el usuario podría presentar sobrepeso u obesidad, se utilizará el *peso ajustado* calculado con la Ecuación 2.4.

Así, las calorías siempre se calcularán en base a las necesidades del usuario.

7.3 Gestión de usuarios

La Gestión del Usuario incluye las funcionalidades más básicas y fundamentales de la aplicación, permitiendo tanto el registro de nuevos usuarios como la administración de perfiles para usuarios ya registrados.

7.3.1. Creación de usuarios

La creación de usuarios tiene como objetivo permitir a un usuario no registrado crear una cuenta para entrar al sistema. Para llevar a cabo este proceso el usuario deberá rellenar un cuestionario en el que se le pedirán no solo los datos necesarios para crear su cuenta, sino también los datos para realizar el cálculo de las calorías diarias que debe consumir, ya que posteriormente el menú se creará en base a este dato.

Para crear el usuario, se pide el nombre, correo y contraseña. Esta información se envía a la base de datos; si existe el correo, se informa al usuario, y al cambiarlo por un correo válido, se pasa a la toma de datos de información relevante para hacer el cálculo de calorías mediante la formula presentada en la Sección 2.1.4 (sexo, edad, altura, actividad física). Para contraseñas, 1 mayúscula, mínimo 8 caracteres, debe contener al menos 1 número.

Después de aportar los datos básicos para la creación de la cuenta, se pasa al cuestionario para el cálculo de las calorías.

Este proceso se ha hecho a través de un *Activity*, **CuestionarioDatosIU**, en el cual se presentarán los diferentes *Fragments*, cada uno de ellos dedicado a recoger unos datos en especifico. En el *Activity* están las variables de los datos que queremos obtener inicializadas a *null* y cada uno de los *Fragments* pasará los datos correspondientes para rellenarlos en el *Activity*.

Para pasar los datos entre las diferentes clases se hace uso de la interfaz **OnDataPassListener** y la clase **DataObject**. Esto es una implementación típica en Android para comunicar datos entre *Fragment* o entre un *Fragment* y su actividad padre. Esta interfaz define un método llamado **onDataPass** que recibe un objeto de tipo **DataObject**. La idea principal es que cualquier clase que implemente esta interfaz se compromete a definir cómo manejará los datos que se le pasen a través de este método.

DataObject es una clase que encapsula dos campos privados: **source** y **data**, ambos de tipo *String*. El campo **source** puede utilizarse para identificar la procedencia del dato (por ejemplo, el *Fragment* que lo envía), mientras que el campo **data** contiene la información que se desea transmitir.

Por lo tanto, cuando un *Fragment* necesita enviar datos a su actividad o a otro *Fragment*, se utiliza **OnDataPassListener**. El *Fragment* invoca el método **onDataPass** a través de un *listener*, pasando un objeto **DataObject** que contiene la información. La actividad o *Fragment* receptor, que ha implementado la interfaz, recibe y maneja esos datos en su método **onDataPass**, permitiendo una comunicación eficiente y desacoplada entre componentes.

Al enviar el último de los datos al *Activity*, este invocará a la función de cálculo de calorías y se creará finalmente el usuario. Después, enviará el resultado al último de los *Fragments*, **Bienvenida**, que explica al usuario lo que encontrará en la aplicación y muestra el resultado del cálculo.

7.3.2. Perfil de usuario

En la interfaz del perfil, el usuario tiene la posibilidad de llevar a cabo dos acciones principales que son fundamentales para una experiencia personalizada y segura.

En primer lugar, el usuario puede actualizar su contraseña. Esta función le permite cambiar su información de acceso, un paso especialmente importante para mantener la seguridad de su cuenta. Al actualizar la contraseña, el usuario protege su perfil personal contra accesos no autorizados, garantizando así que solo él pueda acceder a su información y datos privados.

En segundo lugar, el usuario tiene la opción de volver a completar el cuestionario diseñado para recalcular las calorías diarias recomendadas. Este cuestionario ayuda a ajustar las necesidades nutricionales del usuario según sus objetivos actuales, como el aumento de peso, la pérdida de peso o el mantenimiento de su estado físico. Al proporcionar información actualizada, el sistema puede generar un nuevo menú semanal adaptado a las nuevas necesidades y metas del usuario. Esta función es importante para asegurar que las recomendaciones nutricionales sean siempre relevantes y efectivas, permitiendo al usuario seguir un plan alimenticio que se ajuste a sus circunstancias cambiantes.

Ambas funciones son clave no solo para mantener la seguridad de la cuenta del usuario, sino también para garantizar que el apoyo nutricional proporcionado sea preciso y adecuado para sus objetivos en evolución.

7.4 Gestión de la lista de la compra

Uno de los aspectos más desafiantes de implementar en nuestra aplicación ha sido la gestión de la lista de la compra, principalmente debido a la complejidad de la base de datos y la lógica necesaria para procesar la información.

La lista de la compra se genera a partir de un conjunto de elementos de tipo **IngredienteFormato**. Cada **IngredienteFormato** representa un ingrediente específico de una receta dentro del menú y está compuesto por: el ID de la receta, el ID del producto en un supermercado concreto, la cantidad de producto necesaria y la unidad de medida. La lista inicial de elementos **IngredienteFormato** se saca del menú semanal.

El proceso de gestión de la lista de la compra incluye varios pasos complejos:

1. **Agrupación de Ingredientes:** Una vez obtenida la lista de **IngredienteFormato**, debemos agrupar estos ingredientes en función de su ID de Producto junto con la unidad de medida. Este paso es crítico para consolidar todas las instancias de un mismo producto, sumando las cantidades necesarias y ajustando las unidades de medida. Esta agrupación se realiza con el método del Fragmento 7.1.

2. **Consulta a la API:** Inicialmente, es necesario hacer una llamada a la API para obtener la información detallada de todos los productos disponibles en el supermercado que están en nuestra lista de **IngredienteFormato**, devolviendo una lista de elementos de tipo **Producto**.

3. **Comparación y Consolidación:** La lista reducida de **IngredienteFormato** se compara con la lista de productos obtenida de la API. Este paso permite determinar la cantidad final de cada producto que debe ser comprada. Aquí es necesario realizar ajustes de unidades de medida y sumar cantidades para obtener una lista única y precisa de productos necesarios para el menú.

```
1   public static List<IngredienteFormato> reducirIngredientesFormato() {
2       List<IngredienteFormato> listaOriginal = MenuSemanal.
            getListaIngredienteFormato();
3       // Mapa para almacenar los ingredientes sin duplicados
4       Map<String, IngredienteFormato> mapaIngredientes = new HashMap<>();
5
6       // Iterar sobre la lista original
7       for (IngredienteFormato ingrediente : listaOriginal) {
8           // Crear una clave única combinando id y formato
9           String clave = ingrediente.getId() + "-" + ingrediente.getFormato()
                ;
10
11          // Verificar si la clave ya existe en el mapa
12          if (mapaIngredientes.containsKey(clave)) {
13              // Si existe, sumamos la cantidad al ingrediente existente
14              IngredienteFormato existente = mapaIngredientes.get(clave);
15              existente.setCantidad(existente.getCantidad() + ingrediente.
                    getCantidad());
16          } else {
17              // Si no existe, lo agregamos al mapa
18              mapaIngredientes.put(clave, ingrediente);
19          }
20      }
21
22      // Retornar la lista de ingredientes del mapa
23      return new ArrayList<>(mapaIngredientes.values());
24  }
```

Listing 7.1: Método para la reducción de la lista de **IngredientesFormato**.

La complejidad radica en que cada **IngredienteFormato** puede estar asociado con más de una unidad de medida específica diferente dependiendo de la receta, por lo que la base de datos contiene a un número muy alto de unidades de medida (e.g., cucharada, cucharadita, unidades, gramos), y nuestra tarea es consolidar estas cantidades en una única lista de productos que sea coherente y utilizable.

La solución propuesta se centra en tratar de manera especial los casos de unidades de medida más comunes y simplificar el proceso para las menos comunes. A continuación, se detalla el enfoque adoptado:

1. **Identificación de Unidades de Medida Especiales:** Hemos identificado ciertas unidades de medida que requieren un tratamiento especial debido a su naturaleza o frecuencia en la base de datos. En nuestro caso, estas unidades incluyen:

- **Gramos** (g)
- **Mililitros** (ml)
- **Grandes, Medianos y Pequeños** (aplicables específicamente a bandejas de huevos)

2. **Proceso para Unidades Especiales:** Para las unidades de medida especiales, el algoritmo busca su equivalente en la lista de productos disponible en la base de datos, compara la cantidad de **IngredienteFormato** con la cantidad disponible en el **Producto** del supermercado. Basado en esta comparación, se determina cuántas unidades del producto son necesarias para cumplir con la cantidad requerida del ingrediente. Se ajusta la cantidad de productos en la lista de compra en consecuencia.

3. **Proceso para Unidades de Medida Comunes:** estas unidades comunes se añaden directamente a la lista de compra. Se incluye un único elemento por tipo de producto, simplificando el proceso y evitando complicaciones innecesarias.

Finalmente, en el mismo método que donde se realiza la reducción de la lista de la compra, se calcula también el precio total de la lista. Este código se puede observar en el Fragmento 7.2.

```
public static List<Producto> reducirListaCompra() {
        // Reiniciar el precio total
        precioTotal = 0.0;

        // Obtener la lista reducida de ingredientes
        List<IngredienteFormato> listaReducida = reducirIngredientesFormato();

        // Obtener la lista de productos actuales de la compra
        List<Producto> listaDeLaCompraActual = getListaDeLaCompraActual();

        // Iterar sobre cada ingrediente de la lista reducida
        for (IngredienteFormato ingrediente : listaReducida) {
            // Verificar si el formato es "g" o "ml"
            if (ingrediente.getFormato().equals("g") || ingrediente.getFormato
                ().equals("ml") || ingrediente.getFormato().equals("grandes")
                || ingrediente.getFormato().equals("medianos")
                    || ingrediente.getFormato().equals("pequeños")) {
                // Buscar el producto correspondiente en la lista de la compra
                for (Producto producto : listaDeLaCompraActual) {
                    if (producto.getIdProducto() == ingrediente.getId()) {
                        // Comparar la cantidad del ingrediente con la del
                            producto
                        if (ingrediente.getCantidad() > producto.getCantidad())
                            {
                            // Calcular cuántas veces se necesita el producto
                            int cantidadNecesaria = (int) Math.ceil(ingrediente
                                .getCantidad() / producto.getCantidad());

                            // Cambiar el nombre del producto para reflejar la
                                cantidad necesaria
                            producto.setNombreProducto(producto.
                                getNombreProducto() + " x" + cantidadNecesaria)
                                ;

                            // Sumar el precio del producto la cantidad
                                necesaria de veces
                            precioTotal += producto.getPrecio() *
                                cantidadNecesaria;
```

```
                        } else {
                            // Sumar el precio del producto solo una vez si no
                                se requiere más
                            precioTotal += producto.getPrecio();
                        }
                    }
                }
            }
        }
        precioTotal = Math.floor(precioTotal * 100) / 100.0;

        return listaDeLaCompraActual;
    }
```

Listing 7.2: Método para la reducción de la lista de objetos **Producto**.

7.5 Implementación de interfaces

En esta sección, desarrollaremos cómo se han afrontado los retos y problemas que han surgido en la implementación de las interfaces diseñadas, la cuales podemos encontrar en la Sección 6.3. Para poder entender la implementación llevada a cabo, es necesario comprender el significado de los dos siguientes términos explicados en la Sección 7

7.5.1. Inicio e identificación

En cuanto a la implementación de las interfaces relacionadas con la identificación del usuario, el mayor reto ha sido la creación del cuestionario que recopila la información necesaria para calcular las calorías diarias que el usuario debe consumir. Cabe destacar que las pantallas de registro e inicio de sesión están conformadas por un *Activity* cada una. Por lo tanto, tanto la implementación como la comunicación entre ellas se gestionan de manera sencilla a través de *Intents*. En el Fragmento 7.3 se puede ver un ejemplo de la implementación de un *Intent*. Este método debe asignarse en el correspondiente XML con el objeto que lo invocará.

```
    public void clickRegistro(View view){
        Intent intent = new Intent(Splash.this, CuestionarioDatosIU.class);
        startActivity(intent);
    }
```

Listing 7.3: Ejemplo de uso Intent.

Cuestionario de registro de nuevo usuario

El cuestionario de registro está compuesto por una serie de interfaces, como se muestra en la Figura 7.1, estas interfaces se utilizan para cumplir los requisitos funcionales: **Registrar usuario** (**RF1** Tabla 5.1) y **Cálculo de las calorías necesarias diarias** (**RF3** Tabla 5.3). A través de estas interfaces, el usuario proporciona todos los datos necesarios, lo que permite tanto la creación de su perfil en la aplicación como el cálculo de las calorías diarias que se utilizarán para personalizar su menú.

Todas las pantallas del cuestionario son *Fragments* que van cambiando a medida que se completan los datos solicitados en cada uno. La pantalla principal que se encuentra en el *Activity* es **datos_personales.xml** con su correspondiente clase Java **CuestionarioDatosIU**.

Figura 7.1: Interfaces que forman parte del cuestionario de registro.

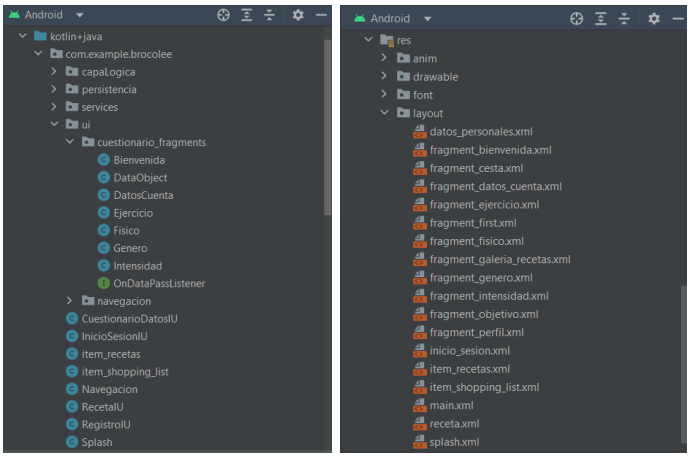


Figura 7.2: Clases relacionadas con la cuestionario de registro.

En la Figura 7.2, se puede observar que se han organizado las clases controladoras de los *Fragments* en una carpeta llamada **cuestionario_fragments**. Dado que para cada *Fragment* existe una clase controladora Java, la comunicación entre ellos se complica. Cabe destacar que debe existir comunicación entre cada uno de los *Fragments* con el *Activity*, esto se debe a que cada *Fragment* recoge uno o más datos y el usuario solo se puede crear una vez que se han recopilado todos los datos necesarios.

La solución implementada está basada en la clase **DataObject** y la interfaz **OnDataPassListener**, cuyo funcionamiento se explica en la Sección 7.3.1. En el Fragmento 7.4 se puede observar el código correspondiente a la recepción de los datos en el *Activity* y en el Fragmento 7.5 el código del envío de los datos desde el *Fragment* al *Activity*.

```
1     public void onDataPass(DataObject data) {
2         if (data.getSource().equals("correo")) {
3             correo = data.getData();
4
5         } else if (data.getSource().equals("nombre")) {
6             nombre = data.getData();
7
8         } else if (data.getSource().equals("contraseña")) {
9             contraseña = data.getData();
10
11        } else if (data.getSource().equals("genero")) {
12            genero = data.getData();
```

```
13
14          } else if (data.getSource().equals("ejercicio")) {
15              ejercicio = Double.parseDouble(data.getData());
16
17          } else if (data.getSource().equals("intensidad")) {
18              intensidad = Double.parseDouble(data.getData());
19
20          } else if (data.getSource().equals("altura")) {
21              altura = Double.parseDouble(data.getData());
22
23          } else if (data.getSource().equals("peso")) {
24              peso = Double.parseDouble(data.getData());
25          } else if (data.getSource().equals("fechaNacimiento")) {
26              fechaNacimiento = data.getData();
27          }
28      }
```

Listing 7.4: Fragmento de código de la clase CuestionarioIU.java. Recepción de los datos de los Fragments

```
1       @Override
2       public void onAttach(@NonNull Context context) {
3           super.onAttach(context);
4           try {
5               dataPassListener = (OnDataPassListener) context;
6           } catch (ClassCastException e) {
7               throw new ClassCastException(context.toString() + " debe
                    implementar OnDataPassListener");
8           }
9       }
10      private void sendGenero(String genero) {
11          DataObject dataObject = new DataObject("genero", genero);
12          dataPassListener.onDataPass(dataObject);
13      }
```

Listing 7.5: Fragmento de código de la clase Genero.java. Envío de los datos al Activity.

7.5.2. Perfil

En este caso, la interfaz de perfil se centra en ofrecer las funcionalidades básicas necesarias para la seguridad y la gestión de la cuenta, tal como se explica en la Sección 7.3.2. Además, esta interfaz se utilizará para cumplir el requisito funcional **Ver perfil** (**RF5** Tabla 5.5).

La interfaz de perfil está diseñada para ser simple y directa, tal como se ve en la Figura 7.3. Su propósito principal es ofrecer opciones básicas y esenciales para el usuario, que ayudan a mantener la experiencia de usuario clara y libre de distracciones. Además, la interfaz reutiliza elementos de diseño del cuestionario de registro para garantizar coherencia en la experiencia del usuario. Esto no solo simplifica el diseño, sino que también reduce la necesidad de crear múltiples interfaces para tareas similares.

Figura 7.3: Interfaz del perfil de usuario.

7.5.3. Barra de navegación

Para implementar la navegación entre las pantallas principales, se ha utilizado la biblioteca ***Chip Navigation Bar*** [26]. Esta biblioteca facilita la creación de una barra de navegación con un diseño moderno y minimalista, que se podrá personalizar en función a la identidad de la aplicación, tal como podemos ver en la Figura 7.4. La barra está compuesta por chips, cada uno representando una de las opciones del menú.

Figura 7.4: Interfaz de barra de navegación.

En primer lugar, es necesario agregar la dependencia en el archivo *build.gradle* de la aplicación para disponer de esta biblioteca. El siguiente paso para usar este recurso dentro de un *Activity*, consiste en agregar el componente en el *layout*, a través del archivo de diseño XML, tal como se muestra en el Fragmento 7.6.

```
1    <com.ismaeldivita.chipnavigation.ChipNavigationBar
2    android:id="@+id/barraNav"
3    android:layout_width="match_parent"
4    android:layout_height="60dp"
5    android:background="@color/Barra"
6    app:cnb_iconSize="24dp"
7    app:cnb_menuResource="@menu/barra_navegacion"
```

```
8       app:cnb_textAppearance="@style/TextAppearance.AppCompat.Light.SearchResult.
            Subtitle"
9       app:cnb_unselectedColor="@color/white" />
```

Listing 7.6: Código XML de la barra de navegación

En el XML del *layout* es posible personalizar ligeramente el elemento, cambiando atributos como el tamaño de la barra o de los iconos, el color o el tipo de letra. Pero, si se desea personalizar el componente cambiando los datos de la navegación de la aplicación, se debe recurrir a el archivo **barra_navegacion.xml**, mostrado en el Fragmento 7.7 para mayor precisión. En esta clase se pueden configurar los chips que aparecen en la barra de navegación.

```
1       <?xml version="1.0" encoding="utf-8"?>
2   <menu xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto">
4       <item
5           android:id="@+id/objetivo"
6           android:title="Objetivo"
7           android:icon="@drawable/objetivo_claro"
8           app:cnb_iconColor="@color/Barra"
9           app:cnb_textColor="@color/black"
10          app:cnb_backgroundColor="@color/Verde_Objetos"
11          />
12      <item
13          android:id="@+id/lista"
14          android:title="Lista de la compra"
15          android:icon="@drawable/cesta_compra_claro"
16          app:cnb_iconColor="@color/Barra"
17          app:cnb_textColor="@color/black"
18          app:cnb_backgroundColor="@color/Verde_Objetos"
19          />
20      <item
21          android:id="@+id/recetas"
22          android:title="Recetas"
23          android:icon="@drawable/cooking_book_claro"
24          app:cnb_iconColor="@color/Barra"
25          app:cnb_textColor="@color/black"
26          app:cnb_backgroundColor="@color/Verde_Objetos"
27          />
28      <item
29          android:id="@+id/perfil"
30          android:title="Perfil"
31          android:icon="@drawable/user"
32          app:cnb_iconColor="@color/Barra"
33          app:cnb_textColor="@color/black"
34          app:cnb_backgroundColor="@color/Verde_Objetos"
35          />
36  </menu>
```

Listing 7.7: Código de el archivo barra_navegacion.xml

Finalmente, desde la clase Java de la interfaz en la que se encuentra el objeto, se tiene que crear un *listener* para la barra de navegación. De esta forma, cuando se interactúe con ella, el comportamiento dependerá del chip que se accione. En este caso, esto se implementa en la clase Navegación.java, configurando el funcionamiento de cada chip dentro del *listener*, como se puede ver en el Fragmento 7.8.

```
1
2
3   barraNav.setOnItemSelectedListener(new ChipNavigationBar.OnItemSelectedListener
        () {
4       @Override
```

```
5       public void onItemSelected(int i) {
6           if(i == R.id.objetivo) {
7               previousIndex = i;
8               getSupportFragmentManager().beginTransaction().replace(R.id.
                    mainFragmentContainer, new Objetivo()).commit();
9           }
10          if(i == R.id.lista) {
11              previousIndex = R.id.lista;
12              getSupportFragmentManager().beginTransaction().replace(R.id.
                    mainFragmentContainer, new Cesta()).commit();
13          }
14          if(i == R.id.recetas) {
15              previousIndex = R.id.recetas;
16              getSupportFragmentManager().beginTransaction().replace(R.id.
                    mainFragmentContainer, new GaleriaRecetas()).commit();
17          }
18          if(i == R.id.perfil) {
19              previousIndex = R.id.perfil;
20              getSupportFragmentManager().beginTransaction().replace(R.id.
                    mainFragmentContainer, new Perfil()).commit();
21          }
22      }
23  });
```

Listing 7.8: Fragmento de código de la clase Navegacion

7.5.4. Objetivo

La pantalla del objetivo permite al usuario consultar su menú y acceder a todas las recetas que lo componen, cumpliendo de esta forma los requisitos funcionales: ver menú semanal (**RF8** Tabla 5.8) y modificar menú semanal (**RF9** Tabla 5.9). Esta interfaz está diseñada para mostrar los platos del desayuno, la comida y la cena según el día de la semana. Como se puede ver en la Figura 7.5 el usuario podrá seleccionar el día de la semana cuyo menú quiere ver, cambiar las recetas que no sean de su agrado y registrar las comidas hechas, según es necesario para los requisitos funcionales: **Ver menú semanal** (**RF8** Tabla 5.8), **Modificar menú semanal** (**RF9** Tabla 5.9) y **Registrar consumo calórico diario** (**RF10** Tabla 5.10).

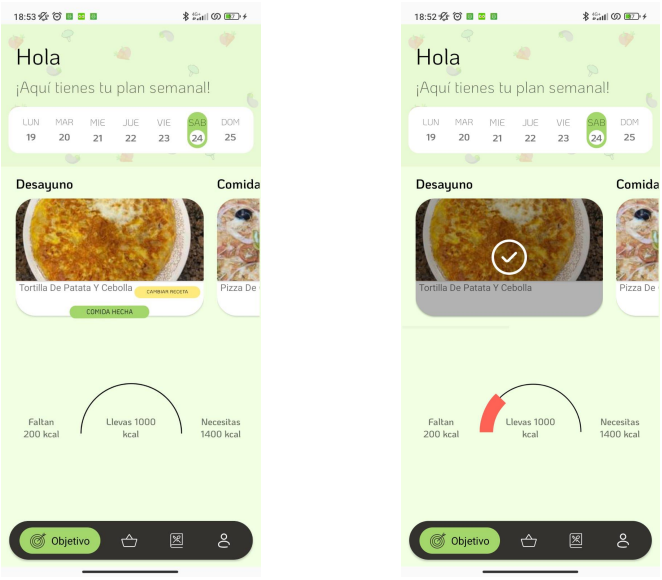


Figura 7.5: Interfaz del objetivo semanal.

El mayor reto en la implementación de esta interfaz ha sido la previsualización de las recetas diarias, especialmente considerando que cada imagen tiene un tamaño diferente y todas deben ajustarse al mismo formato.

Como solución a este problema se ha utilizado la biblioteca ***CardView*** [27]. Este componente se utiliza comúnmente para crear tarjetas rectangulares con bordes redondeados y una sombra, lo que genera un efecto de elevación, proporcionando una apariencia más moderna y consistente. Además, en este caso, nos permitirá ajustar la imagen automáticamente al *Cardview*.

```
<androidx.cardview.widget.CardView
    android:layout_width="match_parent"
    android:layout_height="125dp"
    app:cardBackgroundColor="#FFFF8D" >

    <ImageView
        android:id="@+id/plato2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerCrop"
        android:src="@drawable/ic_launcher_background" />
</androidx.cardview.widget.CardView>
```

Listing 7.9: Ejemplo de uso de un componente *CardView*

En el diseño de la interfaz, se utilizan dos *CardView* anidados en cada receta, cada uno con un propósito específico:

1. ***CardView* interno:** Este es el *CardView* que contiene la imagen, podemos ver su código en el Fragmento 7.9. Su principal función es ajustar la imagen para que se acomode dentro de sus límites, manteniendo las proporciones correctas y asegurándose de que la imagen no sobresalga. Este *CardView* actúa como un contenedor para la imagen, sin añadir ningún efecto visual adicional (como bordes redondeados).

2. ***CardView* externo:** Este *CardView* encapsula al primero (el que contiene la imagen). Su función es darle estilo a la tarjeta completa, añadiendo bordes redondeados y cualquier otro efecto visual que desees. Básicamente, este *CardView* es el que define la apariencia de la tarjeta completa, mientras que el primero se encarga únicamente de la imagen dentro de la tarjeta.

7.5.5. Lista de la compra

En la Figura 7.6 podemos ver la interfaz correspondiente a la lista de la compra necesaria para cumplir el requisito funcional **Ver lista de la compra** (**RF4** Tabla 5.4) En el diseño inicial de la interfaz, se contempló la opción de que el usuario pudiera seleccionar el supermercado para el cual quería generar su lista de la compra, con el objetivo de ofrecer una experiencia personalizada y adaptada a las preferencias del usuario. No obstante, en el contexto del desarrollo de esta prueba de concepto, se ha decidido simplificar esta funcionalidad. En futuras versiones, se incluirá la opción de selección de supermercado, siguiendo el planteamiento original del proyecto.

Para implementar la interfaz de la lista de la compra, enfrentamos el reto de crear los ítems de la lista en tiempo de ejecución.

Una de las soluciones más eficientes y recomendadas es utilizar un *RecyclerView* junto con un adaptador. Un *RecyclerView* es una vista de Android diseñada para manejar grandes conjuntos de datos de manera eficiente y fluida. Esta solución implica definir en un archivo XML el diseño de cada ítem de la lista. Este diseño se repetirá para cada elemento

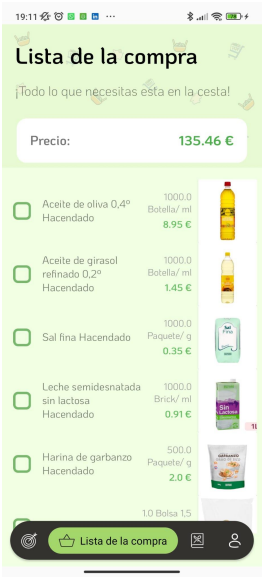


Figura 7.6: Interfaz lista de la compra.

en la lista de la compra. Luego, mediante un adaptador y una clase modelo para los ítems, podemos reciclar la misma vista de ítem para todos los elementos en el *RecyclerView*.

Para comenzar, se realizó el diseño del archivo XML. Dado que se trata de productos que el usuario debe comprar, cada ítem en la lista incluye la cantidad y el precio del producto, además de una imagen para facilitar su identificación y el nombre del producto.

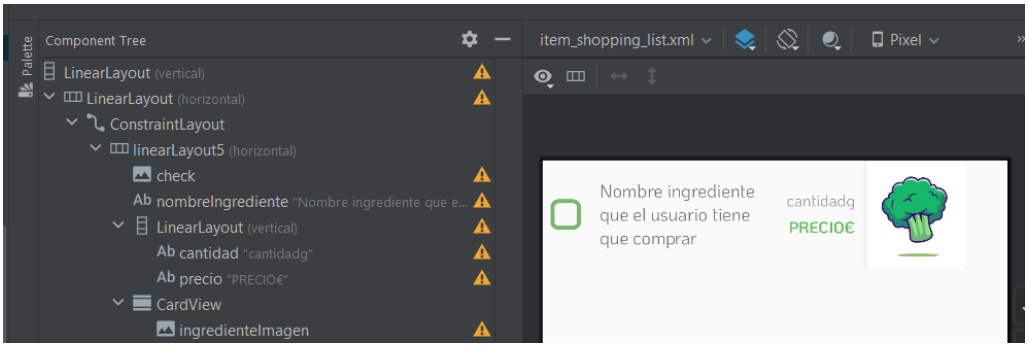


Figura 7.7: Diseño del ítem de la lista de la compra.

Esta estructura que podemos ver en la Figura 7.7, se utilizará para todos los ítems de la lista, cambiando los datos que contiene y posicionándolos dentro del *RecyclerView*.

El siguiente paso fue la creación de la clase modelo para los ítems junto con el adaptador. La clase modelo **AdaptadorItemHolder**, nos permitirá signar los datos a cada ítem y dotar de funcionalidad a los elementos que lo componen. Esta clase es la que define el comportamiento de los elementos del ítem. En el caso de la Figura 7.7, definiría el funcionamiento del *checkbox*.

```
1  public void imprimir(int position) {
2          //Este método se encarga de imprimir el ingrediente
3          nombre.setText(ingredientes.get(position).getNombreProducto());
4          imagen.setImageBitmap(pasarStringAImagen(ingredientes.get(position)
               .getImagen()));
5          cantidad.setText(String.valueOf(ingredientes.get(position).
               getCantidad()) + " " + ingredientes.get(position).getFormato())
               ;
```

```
            precio.setText(String.valueOf(ingredientes.get(position).getPrecio
                ()) + " €" );


            check.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    Drawable on = ContextCompat.getDrawable(view.getContext(),
                        R.drawable.check_on);
                    Drawable off = ContextCompat.getDrawable(view.getContext(),
                         R.drawable.check_off);
                    if (isChecked) {
                        check.setImageDrawable(off);
                        isChecked = false;
                        selected.setVisibility(View.GONE);
                    } else {
                        check.setImageDrawable(on);
                        isChecked = true;
                        selected.setVisibility(View.VISIBLE);
                    }

                }
            });


        }
```

Listing 7.10: Metódo imprimir de la clase AdaptadorItemHolder.

En el Fragmento 7.10 se puede observar cómo el método **imprimir** configura el funcionamiento de los ítems. En este caso, crea un *listener* en el ítem para permitir su selección y deselección.

Por otra parte, la clase adaptador **item_shopping_list** es lo que permite gestionar y visualizar una lista de ingredientes en un *RecyclerView*. Esta clase conecta los datos con las vistas, con el método **onBindViewHolder**, que vemos en el Fragmento 7.11, es fundamental para entender cómo se actualizan las vistas con la información de la lista.

```
    @Override
public void onBindViewHolder(@NonNull item_shopping_list.AdaptadorItemHolder
    holder, int position) {
    holder.imprimir(position);
}
```

Listing 7.11: Metódo onBindViewHolder de la clase itemListaAdapter.

Finalmente, tal como se puede observar en el Fragmento 7.12 en la clase **ListaDeCompra.java**, se configura el *RecyclerView*, se asigna el adaptador y se define el diseño de la lista, en este caso lineal.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
    savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_cesta,
    container, false);

    // Inicializacimos RecyclerView
    recyclerView = view.findViewById(R.id.recyclerViewRecetas);
    //Inicializamos la lista de la compra
    ingredientes = new ArrayList<>();
    ApiClient apiClient = ApiClient.getInstance();
    manager.procesarMenusDiarios(MenuDiario.allMenuDiarios, apiClient,
```

```
12      productos -> {
13          for (Producto producto : productos) {
14              Log.d("Producto", "ID: " + producto.getIdProducto() + ", Nombre: "
                    +
15              producto.getNombreProducto());
16              ingredientes.add(producto.getNombreProducto());
17          }
18      });
19
20      // Configurar el adapter
21      adapter = new itemListaAdapter(ingredientes, getActivity());
22      recyclerView.setAdapter(adapter);
23
24      // Configurar el LayoutManager
25      recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));
26      return view;
27  }
```

Listing 7.12: Metódo onCreateView de la clase ListaDeCompra.

7.5.6. Recetas

Para gestionar las recetas de manera eficiente, hemos diseñado dos interfaces clave en nuestra aplicación:

1. **Galería de Recetas:** esta interfaz funciona como un repositorio visual donde los usuarios pueden ver una colección de recetas a través de miniaturas o fotos. Es una vista general que facilita la exploración rápida de las opciones disponibles.

2. **Interfaz de Receta:** al seleccionar una receta desde la Galería de Recetas, los usuarios son dirigidos a esta interfaz detallada. Aquí, se proporciona toda la información necesaria sobre la receta.

En la Figura 7.8, se pueden observar la interfaces implementadas, estas son necesarias para cumplir el requisito funcional **Ver recetas** (**RF11** Tabla 5.11).

En el caso de la **interfaz de receta**, está diseñada como una actividad que se superpone a la Galería de Recetas. En este modo, la Galería de Recetas queda oculta y no es interactuable hasta que el usuario cierre la Interfaz de Receta. El usuario puede consultar la receta seleccionada, que incluye el nombre del plato, una foto ampliada, los ingredientes y el procedimiento detallado para su preparación. Una vez que haya terminado de leer la receta, puede cerrar la Interfaz de Receta y volver a la Galería de Recetas para seguir explorando otras opciones.

La **galeria de recetas**, supone un reto mayor, aunque la implementación es similar a la de la lista de la compra. Las recetas aparecen en la interfaz como ítems que se generan en tiempo de ejecución.

El procedimiento es el mismo, solo que en este caso el ítem receta es mucho más simple, ya que consta solo de un *ImageView*, tal como podemos ver en el Fragmento 7.13.

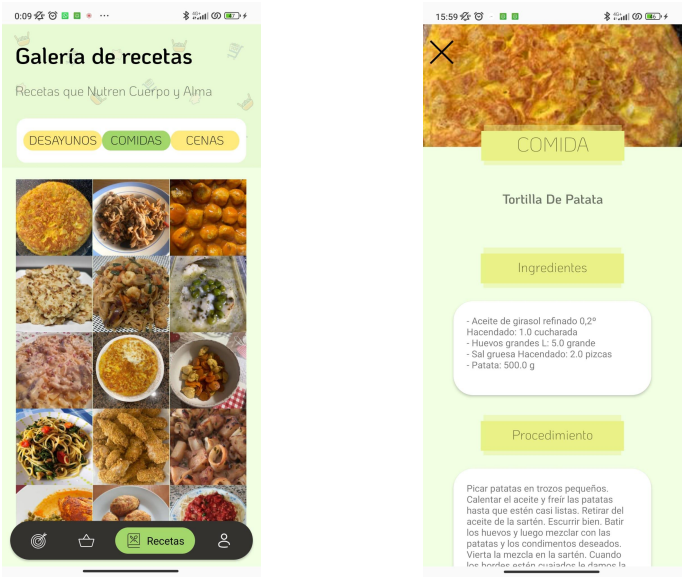


Figura 7.8: Interfaz galería de recetas e interfaz información de recetas.

```
    <?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <ImageView
        android:id="@+id/fotoReceta"
        android:layout_width="116dp"
        android:layout_height="116dp"
        android:layout_weight="1"
        app:srcCompat="@drawable/altura" />
</LinearLayout>
```

Listing 7.13: Archivo item_receta.xml

De igual manera, la implementación de la clase modelo se simplifica, como podemos ver en Fragmento 7.14, esto se debe a que el método imprimir se encargará únicamente de mostrar la foto de la receta y asignarle un *listener*. Este *listener* permitirá que, al seleccionarla, se abra el *Activity* que contiene la información de la receta.

```
public void imprimir(int position) {
    if(recetasList.size() != 0) {
        recetasList.get(position).getImagen();
        String imagen64 = recetasList.get(position).getImagen();
        imagen.setImageBitmap(pasarStringAImagen(imagen64));
        imagen.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if (view.getContext() instanceof Navegacion) {
                    Navegacion actividad = (Navegacion) view.getContext();
                    actividad.abrirReceta(view, recetasList.get(position));
                }
            }
        });
    }

}
```

Listing 7.14: Método imprimir de la clase AdaptadorItemReceta.

Finalmente, como podemos observar en la Figura 7.8, esta vez el diseño de la lista no es lineal, sino que consiste en un diseño en forma de cuadrícula. Por lo que es necesario especificarlo en el método **OnCreateView** de la clase **GaleriaRecetas**, tal como se ve en el Fragmento 7.15.

```
1  @Override
2  public View onCreateView(LayoutInflater inflater, ViewGroup container,
3                           Bundle savedInstanceState) {
4      recetas = Receta.allRecetas;
5
6      View view = inflater.inflate(R.layout.fragment_galeria_recetas,
           container, false);
7
8      recyclerViewRecetas = view.findViewById(R.id.recyclerViewRecetas);
9
10     adapter = new itemRecetasAdapter(recetas, getActivity());
11     recyclerViewRecetas.setAdapter(adapter);
12
13     // Configurar el LayoutManager
14     recyclerViewRecetas.setLayoutManager(new GridLayoutManager(getContext()
           , 3));
15     adapter.notifyDataSetChanged();
16     return view;
17 }
```

Listing 7.15: Método onCreateView de la clase GaleriaRecetas.

CAPÍTULO 8
Pruebas

Para garantizar la calidad del *software* desarrollado, se han implementado una serie de pruebas utilizando distintas herramientas y metodologías. Estas pruebas incluyen tanto pruebas unitarias como pruebas de usuario (*User Testing*), lo que permite la una evaluación no solo de la funcionalidad sino también de la usabilidad del sistema.

8.1 Herramientas de *testing*

JUnit se ha utilizado para realizar las pruebas unitarias a los diferentes componentes de **Brocolee**. Se trata de un *framework* de pruebas unitarias [28] para el lenguaje de programación Java. Permite a los desarrolladores escribir y ejecutar pruebas automatizadas para su código, ayudando a asegurar que el *software* funcione como se espera.

Es importante destacar que **JUnit** utiliza anotaciones con diferentes funciones para identificar los métodos de prueba. Las más comunes son:

- **@Test** que sirve para marcar métodos como casos de prueba.
- **@Before** para los métodos que deben ejecutarse antes de cada prueba.
- **@After** para los métodos que deben ejecutarse después de cada prueba.
- **@AfterClass y @BeforeClass** se utilizan para marcar los métodos que tienen que ejecutarse después o antes que todos los métodos de la clase.

8.2 Pruebas unitarias

Las pruebas unitarias son un tipo de pruebas de *software* que se centran en verificar el funcionamiento de las unidades más pequeñas de un programa, de manera aislada.

Para la ejecución de los métodos de prueba de las clases **Usuario** y **ListaDeCompra** desde **Android Studio**, se ha utilizado el *framework* **JUnit** [28]. Para la realización de estas pruebas, se han creado las clases **UsuarioTest**, **ListaDeCompraTest**, **ProductoTest** e **IngredienteFormatoTest** como podemos ver en la Figura 8.1. En estas clases, encontraremos todos los métodos de prueba de cada una de las clases que estamos probando.

8.2.1. Gestión de usuarios

Para testear la gestión de usuarios se han hecho pruebas unitarias de los métodos de la clase **Usuario** que se encuentra en la capa lógica. En esta clase se encuentran los métodos

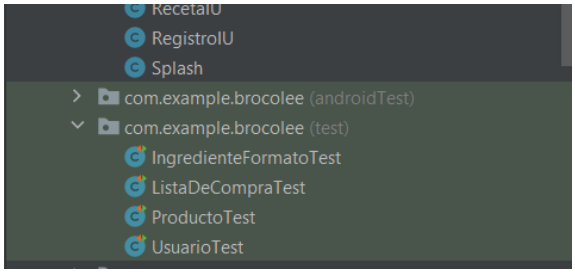


Figura 8.1: Orgnización de las clases de *testing*.

básicos para la gestión de un usuario, desde su propia creación, hasta los métodos para obtener o cambiar cada uno de sus datos.

El primer paso es escribir un método **setup()** etiquetado como **@Before**, es decir, se ejecutará antes de cada método. En este, se lleva a cabo la creación de un usuario de prueba, que se usará en la ejecución de todos los test, excepto en el del constructor. Esto se debe a que el test del método constructor debe comprobar que un usuario se crea correctamente desde cero, por lo tanto no serviría un usuario ya creado. Después de esto podemos escribir todos los casos de prueba como se ve en el Fragmento 8.1.

```
1
2  package com.example.brocolee;
3
4  import org.junit.Before;
5  import org.junit.Test;
6  import java.util.ArrayList;
7  import java.util.List;
8
9  import static org.junit.Assert.*;
10 import com.example.brocolee.capaLogica.Usuario;
11
12 public class UsuarioTest {
13
14     private Usuario usuario;
15
16     @Before
17     public void setUp() {
18
19         usuario = new Usuario("correo@ejemplo.com",
20                 "Nombre Ejemplo", "password123",
21                 "01/01/1990", "Masculino", 3.5,
22                 5, 2000.0);
23     }
24
25     @Test
26     public void testGetCorreo() {
27         assertEquals("correo@ejemplo.com", usuario.getCorreo());
28     }
29
30     @Test
31     public void testSetCorreo() {
32         usuario.setCorreo("nuevo@correo.com");
33         assertEquals("nuevo@correo.com", usuario.getCorreo());
34     }
35
36     @Test
37     public void testGetNombre() {
38         assertEquals("Nombre Ejemplo", usuario.getNombre());
39     }
40
```

```
41      @Test
42      public void testSetNombre() {
43          usuario.setNombre("Nuevo Nombre");
44          assertEquals("Nuevo Nombre", usuario.getNombre());
45      }
46
47      @Test
48      public void testGetContraseña() {
49          assertEquals("password123", usuario.getContraseña());
50      }
51
52      @Test
53      public void testSetContraseña() {
54          usuario.setContraseña("nuevaContraseña");
55          assertEquals("nuevaContraseña", usuario.getContraseña());
56      }
57
58      @Test
59      public void testGetFechaNacimiento() {
60          assertEquals("01/01/1990", usuario.getFechaNacimiento());
61      }
62
63      @Test
64      public void testSetFechaNacimiento() {
65          usuario.setFechaNacimiento("02/02/1992");
66          assertEquals("02/02/1992", usuario.getFechaNacimiento());
67      }
68
69      @Test
70      public void testGetGenero() {
71          assertEquals("Masculino", usuario.getGenero());
72      }
73
74      @Test
75      public void testSetGenero() {
76          usuario.setGenero("Femenino");
77          assertEquals("Femenino", usuario.getGenero());
78      }
79
80      @Test
81      public void testGetEjercicio() {
82          assertEquals(3.5, usuario.getEjercicio(), 0.01);
83      }
84
85      @Test
86      public void testSetEjercicio() {
87          usuario.setEjercicio(4.0);
88          assertEquals(4.0, usuario.getEjercicio(), 0.01);
89      }
90
91      @Test
92      public void testGetCaloriasObjetivas() {
93          assertEquals(2000.0, usuario.getCaloriasObjetivas(), 0.01);
94      }
95
96      @Test
97      public void testSetCaloriasObjetivas() {
98          usuario.setCaloriasObjetivas(2200.0);
99          assertEquals(2200.0, usuario.getCaloriasObjetivas(), 0.01);
100     }
101
102     @Test
103     public void testGetMenusSemanales() {
104         assertEquals(5, usuario.getMenusSemanales());
```

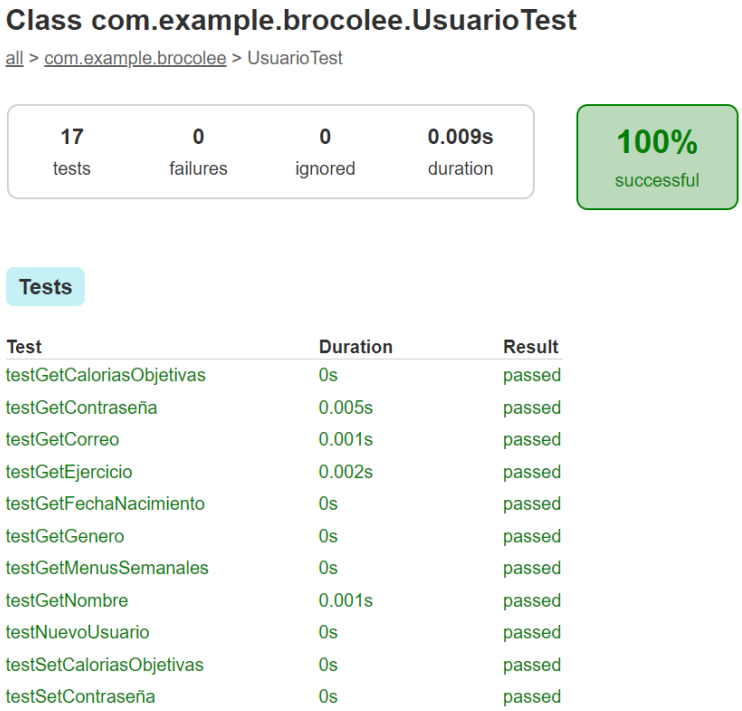
```
105     }
106
107     @Test
108     public void testSetMenusSemanales() {
109         usuario.setMenusSemanales(6);
110         assertEquals(6, usuario.getMenusSemanales());
111     }
112
113     @Test
114     public void testGetAlergias() {
115         assertEquals(alergias, usuario.getAlergias());
116     }
117
118     @Test
119     public void testNuevoUsuario() {
120         // Crea un nuevo usuario utilizando el constructor
121         Usuario usuario = new Usuario(
122                 "correo@ejemplo.com",
123                 "Nombre Ejemplo",
124                 "password123",
125                 "01/01/1990",
126                 "Masculino",
127                 3.5,
128                 5,
129                 2000.0
130         );
131
132         // Verifica que todos los atributos se han inicializado correctamente
133         assertEquals("correo@ejemplo.com", usuario.getCorreo());
134         assertEquals("Nombre Ejemplo", usuario.getNombre());
135         assertEquals("password123", usuario.getContraseña());
136         assertEquals("01/01/1990", usuario.getFechaNacimiento());
137         assertEquals("Masculino", usuario.getGenero());
138         assertEquals(3.5, usuario.getEjercicio(), 0.01);
139         assertEquals(2000.0, usuario.getCaloriasObjetivas(), 0.01);
140         assertEquals(5, usuario.getMenusSemanales());
141
142     }
143
144 }
```

Listing 8.1: Clase **UsuarioTest**.

Cuando se ejecutan pruebas en **Android Studio**, se genera automáticamente un informe (o *report*) que resume los resultados de esas pruebas. Este informe es muy útil para revisar el rendimiento de las pruebas, identificar pruebas fallidas, y analizar cualquier problema. Como podemos ver en la Figura 8.2, correspondiente al archivo, generado por **Android Studio**, de los resultados de la clase de pruebas **UsuarioTest**, todos los *tests* han pasado exitosamente. Esto asegura que nuestros métodos funcionan como se esperaba.

8.2.2. Gestión de la lista de la compra

En las pruebas unitarias de la lista de la compra, es necesario poner a prueba tres clases. Como sabemos, la lista de la compra se genera a partir de una lista de **IngredientesProducto**, que debe ser reducida y comparada con la lista de **Productos** del mismo menú semanal. Por ello, al realizar pruebas en la clase **ListaDeCompra**, es recomendable probar también las clases **IngredienteFormato** y **Producto**. De está forma, se podrán cubrir todos los métodos que se ven implicados en esta funcionalidad.



Class com.example.brocolee.UsuarioTest

all > com.example.brocolee > UsuarioTest

17	0	0	0.009s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
testGetCaloriasObjetivas	0s	passed
testGetContraseña	0.005s	passed
testGetCorreo	0.001s	passed
testGetEjercicio	0.002s	passed
testGetFechaNacimiento	0s	passed
testGetGenero	0s	passed
testGetMenusSemanales	0s	passed
testGetNombre	0.001s	passed
testNuevoUsuario	0s	passed
testSetCaloriasObjetivas	0s	passed
testSetContraseña	0s	passed

Figura 8.2: Resultados de la clase **UsuarioTest**.

Clase ListaDeCompra

La clase ListaDeCompra está compuesta por dos métodos de gran importancia. El primero es **ReducirIngredientesFormato**, que reducirá la lista de **IngredientesFormato** que proporciona el menú, para que contenga **IngredientesFormato** sin repetir el conjunto de nombre y formato. El segundo método, compara la lista reducida de **IngredientesFormato**, con la lista de **Productos** del menú, de manera que devolverá una lista de **Productos**, en la cual los nombres de los productos incluye las unidades que se deben comprar de cada elemento. Este último método también debe calcular el precio total de la lista y almacenarlo en la variable **precioTotal**.

Como se puede observar en el Fragmento 8.2, existe un método **setup()** que inicializa la lista de **IngredientesFormato** y la lista de **Productos**. Por lo tanto se espera que estos métodos puedan calcular cuantas unidades de cada producto se tiene que comprar.

```
1
2  public class ListaDeCompraTest {
3
4      private Producto producto1;
5      private Producto producto2;
6      private List<Producto> listaProductos;
7      private List<IngredienteFormato> listaIngredientesFormato;
8
9      @Before
10     public void setUp() {
11         // Inicializar productos de prueba
12         producto1 = new Producto("Arroz", 1.50, 1.00, 200, 500, "Supermercado A
               ", "imagen_url", "g", new Date());
13         producto1.setIdProducto(1);
14
15         producto2 = new Producto("Leche", 1.20, 1.00, 100, 1000, "Supermercado
               B", "imagen_url", "ml", new Date());
```

```
16          producto2.setIdProducto(2);
17
18          listaProductos = new ArrayList<>();
19          listaProductos.add(producto1);
20          listaProductos.add(producto2);
21
22          ListaDeCompra.setListaDeLaCompraActual(listaProductos);
23
24          // Inicializar lista de IngredienteFormato
25          listaIngredientesFormato = new ArrayList<>();
26          listaIngredientesFormato.add(new IngredienteFormato(1, 300, "g"));
27          listaIngredientesFormato.add(new IngredienteFormato(1, 400, "g"));
28          listaIngredientesFormato.add(new IngredienteFormato(2, 1000, "ml"));
29
30          // Simular la lista de ingredientes en MenuSemanal
31          MenuSemanal.setListaIngredienteFormato(listaIngredientesFormato);
32      }
33
34      @Test
35      public void testReducirIngredientesFormato() {
36          // Ejecutar el método que queremos probar
37          List<IngredienteFormato> listaReducida = ListaDeCompra.
                reducirIngredientesFormato();
38
39          // Verificar que la lista se redujo correctamente (2 elementos: Arroz y
                Leche)
40          assertEquals(2, listaReducida.size());
41
42          // Verificar que las cantidades se sumaron correctamente
43          IngredienteFormato arroz = listaReducida.get(0);
44          IngredienteFormato leche = listaReducida.get(1);
45
46          assertEquals(700, arroz.getCantidad(), 0.001);
47          assertEquals(1000, leche.getCantidad(), 0.001);
48      }
49
50      @Test
51      public void testReducirListaCompra() {
52          // Reducir la lista de compra
53          List<Producto> listaReducida = ListaDeCompra.reducirListaCompra();
54
55          // Verificar que los nombres de los productos reflejan la cantidad
                necesaria
56          assertEquals("Arroz x2", listaReducida.get(0).getNombreProducto());
57          assertEquals("Leche", listaReducida.get(1).getNombreProducto());
58      }
59
60      @Test
61      public void testGetPrecioTotal() {
62          // Reducir la lista de compra y calcular el precio total
63          ListaDeCompra.reducirListaCompra();
64
65          // Verificar que el precio total calculado es correcto
66          assertEquals(4.2, ListaDeCompra.getPrecioTotal(), 0.001);
67      }
68  }
```

Listing 8.2: Clase **ListaDeCompraTest**.

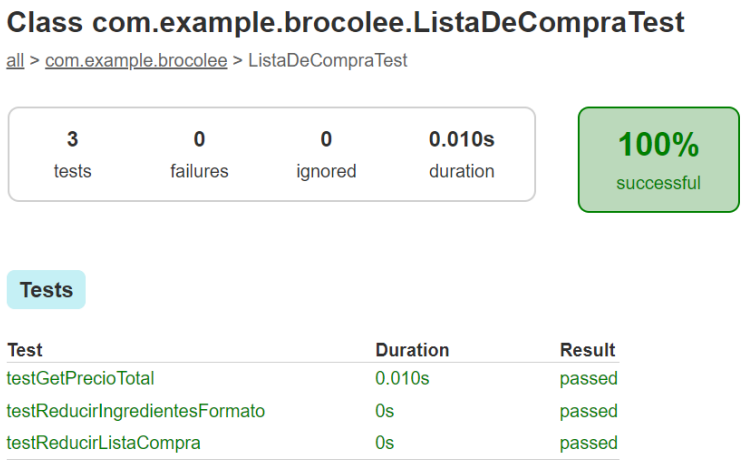


Figura 8.3: Resultados de la clase **ListaDeCompraTest**.

Si prestamos atención al código, la lista de **IngredientesProducto** contenía un total de 700 gramos de arroz y 1000 mililitros de leche; al comparar con las cantidades de la lista de Productos se deduce que el usuario debe comprar dos unidades de arroz y una de leche. En la Figura 8.3, se puede ver el archivo de los resultados, el cual indica que todos los métodos funcionan correctamente.

Clase IngredienteFormato

```
1  public class IngredienteFormatoTest {
2
3      // Test for constructor
4      @Test
5      public void testConstructor() {
6          int id = 1;
7          float cantidad = 2.5f;
8          String formato = "gramos";
9
10         IngredienteFormato ingredienteFormato = new IngredienteFormato(id,
               cantidad, formato);
11
12         assertNotNull(ingredienteFormato);
13     }
14
15     // Tests for getters
16     @Test
17     public void testGetId() {
18         IngredienteFormato ingredienteFormato = new IngredienteFormato(1, 2.5f,
               "gramos");
19         assertEquals(1, ingredienteFormato.getId());
20     }
21
22     @Test
23     public void testGetCantidad() {
24         IngredienteFormato ingredienteFormato = new IngredienteFormato(1, 2.5f,
               "gramos");
25         assertEquals(2.5f, ingredienteFormato.getCantidad(), 0.001);
26     }
27
28     @Test
29     public void testGetFormato() {
30         IngredienteFormato ingredienteFormato = new IngredienteFormato(1, 2.5f,
               "gramos");
```

```
31          assertEquals("gramos", ingredienteFormato.getFormato());
32      }
33
34      // Tests for setters
35      @Test
36      public void testSetId() {
37          IngredienteFormato ingredienteFormato = new IngredienteFormato(0, 0, ""
                );
38          ingredienteFormato.setId(2);
39          assertEquals(2, ingredienteFormato.getId());
40      }
41
42      @Test
43      public void testSetCantidad() {
44          IngredienteFormato ingredienteFormato = new IngredienteFormato(0, 0, ""
                );
45          ingredienteFormato.setCantidad(3.5f);
46          assertEquals(3.5f, ingredienteFormato.getCantidad(), 0.001);
47      }
48
49      @Test
50      public void testSetFormato() {
51          IngredienteFormato ingredienteFormato = new IngredienteFormato(0, 0, ""
                );
52          ingredienteFormato.setFormato("litros");
53          assertEquals("litros", ingredienteFormato.getFormato());
54      }
55
56
57      // Test Parcelable array creation
58      @Test
59      public void testParcelableArray() {
60          IngredienteFormato[] array = IngredienteFormato.CREATOR.newArray(2);
61          assertEquals(2, array.length);
62          assertNull(array[0]);
63          assertNull(array[1]);
64      }
65  }
```

Listing 8.3: Clase **IngredienteFormatoTest**.

Class com.example.brocolee.IngredienteFormatoTest

all > com.example.brocolee > IngredienteFormatoTest

8	0	0	0.015s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
testConstructor	0s	passed
testGetCantidad	0s	passed
testGetFormato	0s	passed
testGetId	0.014s	passed
testParcelableArray	0.001s	passed
testSetCantidad	0s	passed
testSetFormato	0s	passed
testSetId	0s	passed

Figura 8.4: Resultados de la clase **IngredienteFormatoTest**.

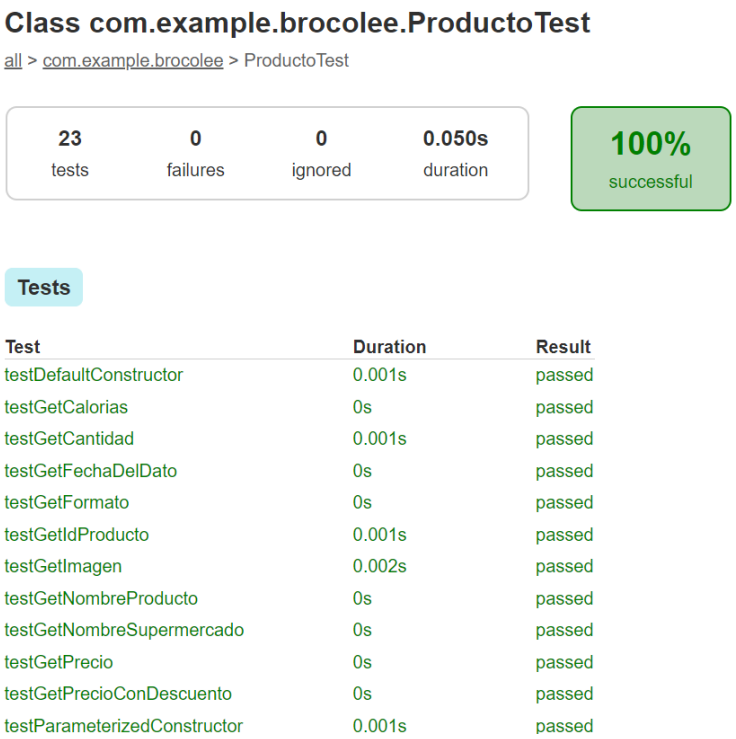
Clase Producto

```
1
2  public class ProductoTest {
3
4      // Test for default constructor
5      @Test
6      public void testDefaultConstructor() {
7          Producto producto = new Producto();
8          assertNotNull(producto);
9      }
10
11     // Test for parameterized constructor
12     @Test
13     public void testParameterizedConstructor() {
14         Date fecha = new Date();
15         Producto producto = new Producto("Pan", 1.99, 1.49, 200, 1.0, "
               Supermercado A", "imagen_url", "kg", fecha);
16
17         assertNotNull(producto);
18         assertEquals("Pan", producto.getNombreProducto());
19         assertEquals(1.99, producto.getPrecio(), 0.001);
20         assertEquals(1.49, producto.getPrecioConDescuento(), 0.001);
21         assertEquals(200, producto.getCalorias(), 0.001);
22         assertEquals(1.0, producto.getCantidad(), 0.001);
23         assertEquals("Supermercado A", producto.getNombreSupermercado());
24         assertEquals("imagen_url", producto.getImagen());
25         assertEquals("kg", producto.getFormato());
26         assertEquals(fecha, producto.getFechaDelDato());
27     }
28
29     // Test for getters
30     @Test
31     public void testGetIdProducto() {
32         Producto producto = new Producto();
33         producto.setIdProducto(5);
34         assertEquals(5, producto.getIdProducto());
35     }
36
37     @Test
38     public void testGetNombreProducto() {
39         Producto producto = new Producto();
40         producto.setNombreProducto("Leche");
41         assertEquals("Leche", producto.getNombreProducto());
42     }
43
44     @Test
45     public void testGetPrecio() {
46         Producto producto = new Producto();
47         producto.setPrecio(2.99);
48         assertEquals(2.99, producto.getPrecio(), 0.001);
49     }
50
51     @Test
52     public void testGetPrecioConDescuento() {
53         Producto producto = new Producto();
54         producto.setPrecioConDescuento(2.49);
55         assertEquals(2.49, producto.getPrecioConDescuento(), 0.001);
56     }
57
58     @Test
59     public void testGetCalorias() {
60         Producto producto = new Producto();
61         producto.setCalorias(150);
```

```
62          assertEquals(150, producto.getCalorias(), 0.001);
63      }
64
65      @Test
66      public void testGetCantidad() {
67          Producto producto = new Producto();
68          producto.setCantidad(0.5);
69          assertEquals(0.5, producto.getCantidad(), 0.001);
70      }
71
72      @Test
73      public void testGetFormato() {
74          Producto producto = new Producto();
75          producto.setFormato("litro");
76          assertEquals("litro", producto.getFormato());
77      }
78
79      @Test
80      public void testGetFechaDelDato() {
81          Date fecha = new Date();
82          Producto producto = new Producto();
83          producto.setFechaDelDato(fecha);
84          assertEquals(fecha, producto.getFechaDelDato());
85      }
86
87      @Test
88      public void testGetNombreSupermercado() {
89          Producto producto = new Producto();
90          producto.setNombreSupermercado("Supermercado B");
91          assertEquals("Supermercado B", producto.getNombreSupermercado());
92      }
93
94      @Test
95      public void testGetImagen() {
96          Producto producto = new Producto();
97          producto.setImagen("url_imagen");
98          assertEquals("url_imagen", producto.getImagen());
99      }
100
101     // Test for setters
102     @Test
103     public void testSetIdProducto() {
104         Producto producto = new Producto();
105         producto.setIdProducto(10);
106         assertEquals(10, producto.getIdProducto());
107     }
108
109     @Test
110     public void testSetNombreProducto() {
111         Producto producto = new Producto();
112         producto.setNombreProducto("Pan integral");
113         assertEquals("Pan integral", producto.getNombreProducto());
114     }
115
116     @Test
117     public void testSetPrecio() {
118         Producto producto = new Producto();
119         producto.setPrecio(1.99);
120         assertEquals(1.99, producto.getPrecio(), 0.001);
121     }
122
123     @Test
124     public void testSetPrecioConDescuento() {
125         Producto producto = new Producto();
```

```
126         producto.setPrecioConDescuento(1.59);
127         assertEquals(1.59, producto.getPrecioConDescuento(), 0.001);
128     }
129
130     @Test
131     public void testSetCalorias() {
132         Producto producto = new Producto();
133         producto.setCalorias(300);
134         assertEquals(300, producto.getCalorias(), 0.001);
135     }
136
137     @Test
138     public void testSetCantidad() {
139         Producto producto = new Producto();
140         producto.setCantidad(2.0);
141         assertEquals(2.0, producto.getCantidad(), 0.001);
142     }
143
144     @Test
145     public void testSetFormato() {
146         Producto producto = new Producto();
147         producto.setFormato("unidad");
148         assertEquals("unidad", producto.getFormato());
149     }
150
151     @Test
152     public void testSetFechaDelDato() {
153         Date fecha = new Date();
154         Producto producto = new Producto();
155         producto.setFechaDelDato(fecha);
156         assertEquals(fecha, producto.getFechaDelDato());
157     }
158
159     @Test
160     public void testSetNombreSupermercado() {
161         Producto producto = new Producto();
162         producto.setNombreSupermercado("Supermercado C");
163         assertEquals("Supermercado C", producto.getNombreSupermercado());
164     }
165
166     @Test
167     public void testSetImagen() {
168         Producto producto = new Producto();
169         producto.setImagen("imagen_url");
170         assertEquals("imagen_url", producto.getImagen());
171     }
172
173     // Test for toString method
174     @Test
175     public void testToString() {
176         Date fecha = new Date();
177         Producto producto = new Producto("Pan", 1.99, 1.49, 200, 1.0, "
                Supermercado A", "imagen_url", "kg", fecha);
178
179         String expected = "Producto{idProducto=0, nombreProducto='Pan', precio
                =1.99, precioConDescuento=1.49, calorias=200.0, cantidad=1.0,
                formato='kg', fechaDelDato=" + fecha + ", nombreSupermercado='
                Supermercado A'}";
180         assertEquals(expected, producto.toString());
181     }
182 }
```

Listing 8.4: Clase **ProductoTest**.



Class com.example.brocolee.ProductoTest

all > com.example.brocolee > ProductoTest

23 tests	0 failures	0 ignored	0.050s duration	100% successful

Tests

Test	Duration	Result
testDefaultConstructor	0.001s	passed
testGetCalorias	0s	passed
testGetCantidad	0.001s	passed
testGetFechaDelDato	0s	passed
testGetFormato	0s	passed
testGetIdProducto	0.001s	passed
testGetImagen	0.002s	passed
testGetNombreProducto	0s	passed
testGetNombreSupermercado	0s	passed
testGetPrecio	0s	passed
testGetPrecioConDescuento	0s	passed
testParameterizedConstructor	0.001s	passed

Figura 8.5: Resultados de la clase **ProductoTest**

8.3 User Testing

El *User Testing* se realizó para evaluar la usabilidad y la experiencia de usuario del sistema. Esta metodología permite identificar problemas de usabilidad y áreas de mejora a través de la observación directa de usuarios reales mientras interactúan con la aplicación.

8.3.1. Planificación

Para realizar una evaluación objetiva, se siguieron las guías propuestas por Rubin y Chisnell [29], estructurando el *User Testing* en tres fases: antes del test, durante el test y después del test.

Fase anterior al test:

En esta fase inicial, se establecen las bases del *User Testing* mediante la definición de los objetivos, la selección de los participantes y la creación de escenarios y tareas relevantes que los usuarios deberán completar durante la prueba. Con esto, se comprueba que el test esté alineado con las metas del proyecto y se enfoque en los aspectos más importantes a evaluar de la aplicación.

- **Objetivo:** El objetivo principal del test de usuario es evaluar la experiencia de usuario y la usabilidad de la aplicación, asegurando que esta cumpla con los estándares esperados en términos de facilidad de uso, eficiencia, control, atractivo, innovación y estimulación.
- **Selección de participantes:** Se seleccionaron cinco participantes, siguiendo la recomendación de Nielsen[30], de distintos perfiles para poder tener distintos puntos de vista con distintos géneros y edades.

- **Creación de escenarios y tareas:** Se desarrollaron escenarios y tareas específicos que los usuarios debían completar, diseñados para evaluar las funciones principales de la aplicación. Concretamente, las tareas que debían seguir los participantes son las siguientes:

  1. Abrir la aplicación.
  2. Darse de alta en la aplicación, generar un usuario con su información.
  3. Iniciar sesión con la cuenta que han creado anteriormente.
  4. Interactuar con el menú semanal para ver las recetas de los días de la semana.
  5. Cambiar una receta.
  6. Registrar una comida.
  7. Acceder a la lista de la compra y marcar como comprado un producto.
  8. Acceder a la galería de recetas.
  9. Filtrar por desayunos y abrir una receta.
  10. Acceder al perfil.
  11. Cambiar la contraseña.
  12. Actualizar los datos para el cálculo de calorías.

Fase durante el test:

En esta fase, los participantes interactúan directamente con la aplicación bajo observación, mientras completan las tareas previamente definidas. Esta fase sirve para recoger datos sobre cómo los usuarios reales perciben y utilizan la aplicación.

1. **Entrega de documentos iniciales:** Se proporcionaron a los participantes formularios demográficos y de consentimiento de toma de datos, con el fin de seguir con la Ley Orgánica de Protección de Datos [31], y se les explicó el propósito del test.

2. **Ejecución de las tareas:** Los usuarios realizaron las tareas definidas mientras se observaba su interacción con la aplicación.

3. **Evaluación con Cuestionario:** Al finalizar, se les pidió que completaran el *User Experience Questionnaire* [32], un cuestionario estandarizado, que consta de 26 items, diseñado para medir la experiencia de usuario en 6 dimensiones:

   - **ATRACCIÓN:** Mide la impresión general del usuario sobre la aplicación, incluyendo aspectos como la estética y la primera impresión.
   - **TRANSPARENCIA:** Evalúa la facilidad con la que los usuarios pueden comprender la interfaz y cómo se les presentan las opciones y la información.
   - **EFICIENCIA:** Mide la capacidad de la aplicación para permitir a los usuarios realizar sus tareas de manera rápida y efectiva.
   - **INNOVACIÓN:** Evalúa el grado de innovación y creatividad de la aplicación, así como su capacidad para captar el interés de los usuarios.
   - **ESTIMULACIÓN:** Mide cuán interesante y motivante es la aplicación para el usuario, incentivándolo a continuar utilizándola.
   - **CONTROLABILIDAD:** Evalúa el grado en que la aplicación permite al usuario sentirse en control de sus acciones, sin generar frustración o dependencia.

La ATRACCIÓN es una medición de cuán agradable o desagradable es la experiencia para el usuario. TRANSPARENCIA, EFICACIA y CONTROLABILIDAD son aspectos de calidad pragmáticos (orientados a objetivos), mientras que ESTIMULACIÓN e INNOVACIÓN son aspectos de calidad hedónicos (no orientados a objetivos). Estas dimensiones proporcionan una visión completa de la experiencia de usuario, permitiendo identificar tanto los puntos fuertes como las áreas que requieren mejoras.

4. **Cierre:** Se dio por finalizada la sesión y se agradeció a los participantes por su tiempo y colaboración.

La duración estimada de cada sesión fue de aproximadamente 25 minutos.

Fase posterior al test:

Después de completar el test, se procede al análisis de los datos recolectados y a la elaboración de un informe con los hallazgos más relevantes. En esta fase, se transforman los datos en información útil que guíe las mejoras futuras de la aplicación.

- **Análisis de resultados:** Los datos obtenidos de las observaciones y del UEQ fueron analizados, a través de la herramienta disponible *online* para analizar los resultados del UEQ[1], para identificar patrones de comportamiento, problemas de usabilidad recurrentes y áreas de satisfacción o insatisfacción entre los usuarios. Este análisis incluyó métricas cuantitativas a través de los valores obtenidos con del UEQ. Además, se complementó el análisis con datos cualitativos obtenidos a través de comentarios espontáneos de los usuarios durante su interacción con la aplicación. Este enfoque cualitativo permitió explorar en mayor profundidad las percepciones de los usuarios sobre aspectos específicos, como la adecuación de las calorías propuestas o la facilidad de uso de ciertas funcionalidades.
- **Informe de hallazgos:** Se elaboró un informe con las observaciones más relevantes, sugerencias de mejora y la priorización de problemas.

8.3.2. Ejecución

La ejecución de las pruebas de usuario se realizó en un entorno controlado, asegurando que los participantes pudieran interactuar con la aplicación sin interrupciones.

Cinco participantes realizaron las pruebas, incluyendo 2 hombres y 3 mujeres, con edades comprendidas entre 21 y 58 años. Cada uno de ellos completó las tareas específicas previamente definidas que abarcaban las funciones principales de la aplicación. Durante esta fase, se observó atentamente la interacción de los usuarios con la aplicación, tomando notas detalladas sobre su comportamiento, comentarios y dificultades encontradas.

8.3.3. Resultados

Tras la ejecución de las pruebas de usuario, se procedió a un análisis de los datos recopilados, tanto cuantitativos como cualitativos. Este análisis permitió identificar los puntos fuertes de la aplicación, así como las áreas que requieren mejoras para optimizar la experiencia del usuario.

[1] https://www.ueq-online.org/

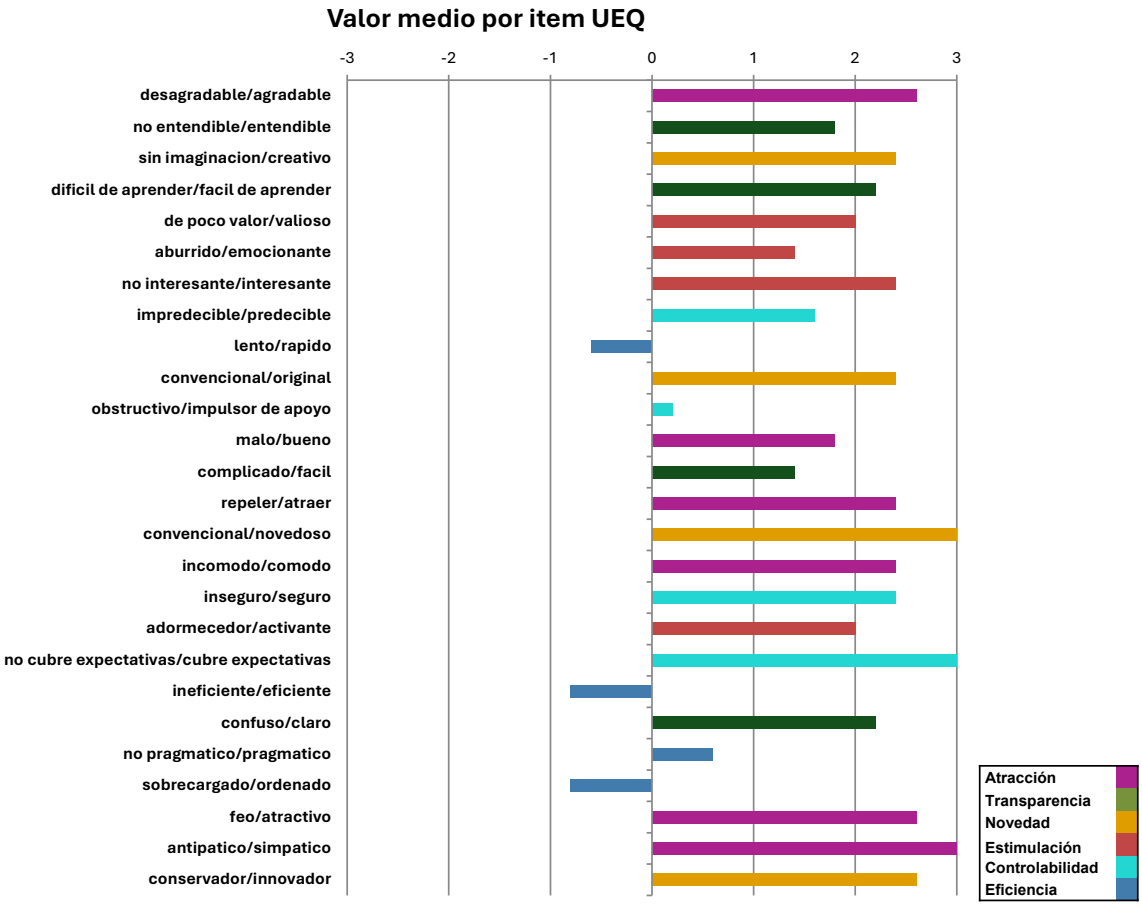


Figura 8.6: Distribución de las respuestas por pregunta para los 5 participantes del *User Testing*.

Análisis cuantitativo

Los valores obtenidos en cada dimensión del UEQ fueron procesados utilizando la herramienta en línea de análisis del UEQ. Todas las escalas tienen valores de -3 a +3. En la Figura 8.6, se presenta la distribución de las respuestas por pregunta para los 5 participantes del *User Testing*.

Se puede observar una tendencia general positiva. La mayoría de las respuestas se encuentran en el rango positivo, lo que sugiere que los usuarios, en general, tienen una percepción favorable de la aplicación. Cabe destacar, que las preguntas relacionadas con la atracción (e.g., "agradable/desagradable"), innovación (e.g., "conservador/innovador"), y controlabilidad (e.g., "inseguro/seguro") muestran una concentración de respuestas cercanas a +2 y +3. Sin embargo, algunas preguntas muestran una dispersión más amplia o incluso respuestas negativas, especialmente en las preguntas relacionadas con la eficiencia (e.g., "lento/rápido" o "ineficiente/eficiente"). Esto sugiere que ciertos aspectos de la aplicación, como el rendimiento o la rapidez de respuesta, no cumplieron completamente con las expectativas de los usuarios. Finalmente, es interesante notar que para varias preguntas, como aquellas relacionadas con la estética y la usabilidad básica (e.g., "entendible/no entendible"), las respuestas son bastante uniformes y están muy inclinadas hacia el lado positivo de la escala. Esto indica una percepción común entre los usuarios sobre estos aspectos de la aplicación.

A continuación, tal como se recoge en la Tabla 8.1, se presentan los resultados promedio para cada dimensión:

Métrica	Media	Desv.Estándar
Atracción	2.467	0.19
Transparencia	1.900	0.08
Eficiencia	-0.400	0.18
Innovación	2.600	0.08
Estimulación	1.950	0.01
Controlabilidad	1.800	0.14

Tabla 8.1

- ATRACCIÓN: Los usuarios calificaron positivamente la estética y la primera impresión de la aplicación, con un valor promedio de $2,467$. Esto indica que la aplicación resulta visualmente agradable y atractiva para los usuarios.

- TRANSPARENCIA: La facilidad de uso y la comprensión de la interfaz fueron bien valoradas, con un valor promedio de 1,90 lo que sugiere que los usuarios encontraron la aplicación intuitiva y fácil de aprender.

- EFICIENCIA: La capacidad de los usuarios para completar tareas de manera rápida y efectiva obtuvo un valor de $-0,400$ lo que refleja un rendimiento mejorable en términos de eficiencia. Es decir, a pesar de se un valor negativo, no es un valor muy bajo en la escala, aunque recalca la necesidad de mejorar la eficiencia de la aplicación, reduciendo los tiempos de latencia de peticiones.

- INNOVACIÓN: La aplicación fue percibida como innovadora y creativa, con un valor de $2,600$, lo que indica que los usuarios apreciaron los elementos innovadores de la aplicación.

- ESTIMULACIÓN: Los usuarios encontraron la aplicación motivante e interesante, obteniendo un valor de $1,950$ en esta dimensión, lo que sugiere que la aplicación es capaz de mantener el interés de los usuarios.

- CONTROLABILIDAD: La percepción de control y la seguridad al interactuar con la aplicación fueron bien evaluadas, con un valor promedio de $1,800$, indicando que los usuarios se sintieron cómodos y en control mientras usaban la aplicación.

Los resultados cuantitativos indican que la aplicación cumple con los estándares esperados en términos de usabilidad y experiencia de usuario, destacándose especialmente en las dimensiones de ATRACCIÓN e INNOVACIÓN. Sin embargo, se identificaron oportunidades de mejora en la dimensión de EFICIENCIA, donde es necesario optimizar el rendimiento para asegurar una experiencia de usuario más fluida y rápida.

Análisis cualitativo

Además de los datos cuantitativos, se recopiló información cualitativa a través de observaciones y comentarios de los usuarios durante la ejecución de las pruebas. Este análisis cualitativo reveló patrones de comportamiento y actitudes hacia la aplicación que no siempre se reflejan en los datos numéricos.

Los usuarios destacaron de forma positiva el diseño atractivo y la organización clara de la aplicación, mencionando específicamente que la estética y la disposición de los elementos facilitaban la navegación. La estructura intuitiva del menú semanal fue particularmente apreciada, ya que los usuarios la encontraron lógica y fácil de seguir.

Sin embargo, también se identificaron algunos problemas de usabilidad que podrían afectar negativamente la experiencia de usuario. Por ejemplo, algunos usuarios señalaron que los botones eran demasiado pequeños, lo que dificultaba su uso.

Algunos comentarios recurrentes incluyeron:

- "La aplicación es fácil de usar, pero me costó entender como se podían ver las recetas del menú."
- "Me gustó mucho la forma en que el menú semanal está diseñado."
- "Sería útil si se pudiera deshacer la acción de comida terminada. Me he equivocando dándole al botón de 'Comida hecha' y luego no pude deshacer esta acción."
- "La aplicación es muy agradable, pero para mi gusto la pantalla de carga es muy larga".

Estos comentarios proporcionan un valioso *feedback* que ayudará a orientar futuras mejoras en la aplicación. En particular, se sugiere que se considere una revisión del diseño de los botones para mejorar su usabilidad, así como la implementación de funciones de deshacer en acciones críticas y una optimización del tiempo de carga. Esto no solo mejoraría la eficiencia de la aplicación, sino que también aumentaría la satisfacción general de los usuarios.

8.3.4. Conclusiones (*User Testing*)

En términos generales, los resultados indican que la aplicación es bien recibida por los usuarios, destacándose particularmente en las dimensiones de ATRACCIÓN e INNOVACIÓN. Los usuarios aprecian el diseño estético y la interfaz intuitiva, lo que facilita una navegación agradable y lógica. Además, la percepción de la aplicación como innovadora sugiere que los usuarios valoran las funcionalidades ofrecidas que otras aplicaciones no cumplen.

Sin embargo, el análisis también reveló áreas que necesitan atención para mejorar la experiencia del usuario. La dimensión de EFICIENCIA mostró resultados por debajo de lo esperado, además reforzado con comentarios específicos sobre la lentitud de la aplicación y la necesidad de optimizar el rendimiento. Estos hallazgos subrayan la importancia de reducir los tiempos de respuesta y mejorar la velocidad general de la aplicación.

En conclusión, el *User Testing* ha demostrado que la aplicación tiene un buen punto de partida, con una base sólida en diseño y funcionalidades innovadoras. No obstante, para maximizar la experiencia de usuario, en siguientes fases del desarrollo se debería centrar los esfuerzos en mejorar la eficiencia, refinar los elementos de usabilidad que respondan mejor a las necesidades de los usuarios. Implementar estas mejoras permitirá ofrecer una experiencia más fluida, rápida y satisfactoria para los usuarios finales.

CAPÍTULO 9
Conclusiones

En esta sección final del Trabajo de Fin de Grado, se abordan los resultados alcanzados en relación con los objetivos inicialmente establecidos para el desarrollo de la aplicación móvil de nutrición. A lo largo del proceso de creación e implementación, hemos explorado cómo la aplicación ha cumplido con sus metas fundamentales, así como sus repercusiones en diversos ámbitos. Primero, evaluaremos en qué medida se han logrado los objetivos específicos planteados al inicio del proyecto, centrándonos en la funcionalidad, usabilidad y aceptación por parte de los usuarios.

A continuación, examinaremos las consecuencias sociales, económicas y culturales derivadas del uso de la aplicación, considerando tanto los beneficios tangibles como las posibles áreas de mejora. Reflexionaremos sobre el impacto de la aplicación en el comportamiento alimentario de los usuarios, su potencial para fomentar hábitos saludables y su contribución al bienestar general de la comunidad.

Finalmente, discutiremos la utilidad y las implicaciones de la aplicación en el contexto actual, evaluando cómo ha influido en la vida de los usuarios y qué lecciones se pueden aprender para futuros desarrollos en el campo de la nutrición móvil. La finalidad de esta sección es ofrecer una visión integral sobre el impacto y las consecuencias de la aplicación, proporcionando una base sólida para futuras investigaciones y mejoras en este ámbito.

9.1 Conclusiones

Finalmente, se presentan las conclusiones derivadas del desarrollo y evaluación de la aplicación móvil de nutrición. A lo largo del proyecto, hemos abordado diversos aspectos relacionados con los objetivos establecidos inicialmente, y ahora reflexionamos sobre cómo la aplicación ha cumplido con esas metas y sus repercusiones en distintos ámbitos.

En cuando a los objetivos establecidos, se ha conseguido desarrollar una aplicación final que es completamente funcional. La validación obtenida a través de las pruebas de usuario (*User Testing*) ha demostrado que la aplicación no solo es intuitiva y estéticamente agradable, sino que también aborda un tema relevante para los usuarios: la nutrición. Este aspecto ha sido clave para su aceptación y éxito.

La integración de artículos científicos y estudios en el desarrollo de la aplicación ha permitido que los planes nutricionales sean adaptativos y adecuados a las necesidades individuales de cada usuario. Este enfoque basado en la ciencia garantiza que las recomendaciones no solo sean prácticas y accesibles, sino también respaldadas por investigaciones actuales en el campo de la nutrición. La funcionalidad de la aplicación permite generar planes nutricionales personalizados, basados en las características físicas del

usuario. La adaptación a las necesidades individuales es clave para maximizar la efectividad de las recomendaciones y apoyar a los usuarios en la consecución de sus metas de salud.

El estudio de mercado realizado ha sido fundamental para entender la cabida de nuestra aplicación en el mercado actual. Se ha identificado que, aunque existen numerosas aplicaciones con funciones similares, **Brocolee** se diferencia significativamente frente a las aplicaciones analizadas (**Fitia**, **Unimeal**, **Nutrilio** y **Fitatu**) por su funcionalidad única: la lista de la compra integrada con datos de supermercados reales. Mientras **Fitia** y **Unimeal** proporcionan menús semanales, Brocolee añade valor al automatizar la compra de ingredientes específicos. Por último, la comparación con **Unimeal** subraya la importancia de mantener un equilibrio entre funcionalidad, usabilidad y rendimiento, aspectos en los que **Brocolee** continúa mejorando.

El estudio de mercado también ha revelado algunas debilidades en la aplicación, particularmente en términos de eficiencia, reconociendo que se debe mejorar en cuanto a rendimiento y velocidad. Además de que sería importante aumentar la compatibilidad de la aplicación a diferentes dispositivos, siendo **Fitatu** la aplicación que lidera en compatibilidad al estar disponible en múltiples plataformas. Destacando la importancia de desarrollar estrategias para escalar la aplicación de manera efectiva, asegurando que pueda manejar un aumento en el número de usuarios y la cantidad de datos sin comprometer el rendimiento.

En cuanto a las consecuencias sociales, económicas y culturales, la capacidad de acceder a precios y disponibilidad de productos en supermercados puede contribuir a una mejor gestión económica del presupuesto alimentario de los usuarios. Además, la aplicación puede establecer asociaciones con supermercados y proveedores, potencialmente generando oportunidades de ingresos adicionales o beneficios para los usuarios.

Cabe destacar que el desarrollo de esta aplicación ha sido posible gracias a los conocimientos adquiridos a lo largo de la carrera de Ingeniería Informática. Durante estos años, no solo he profundizado en el ámbito de la programación, sino que también he aprendido aspectos importantes como el *testing* de *software*, lo que ha permitido asegurar la calidad y funcionalidad del producto final. Además, la implementación de prácticas ágiles ha sido fundamental para gestionar el proyecto de manera eficiente, permitiendo adaptaciones y mejoras continuas a lo largo del desarrollo. Asimismo, he aplicado conocimientos en el diseño y desarrollo de aplicaciones funcionales, asegurando que la interfaz y la experiencia de usuario sean intuitivas y efectivas. La gestión de bases de datos también ha sido un componente completamente necesario, facilitando el manejo y almacenamiento de la información de los usuarios de forma segura y eficiente. De esta forma, este proyecto ha representado una oportunidad para integrar y aplicar una amplia gama de competencias técnicas, demostrando la relevancia y aplicabilidad de los estudios en la creación de soluciones tecnológicas innovadoras y funcionales, desde la especificación de requisitos hasta las pruebas con usuarios finales, pasando por todas las fases del desarrollo *software*.

En resumen, este trabajo no solo aborda el aspecto práctico de la elaboración de planes nutricionales, sino que también se sumerge en el análisis teórico y científico que sustenta dichas recomendaciones, asegurando así su validez y eficacia.

9.2 Trabajo futuro

En esta sección se explorarán las posibles direcciones para el futuro desarrollo y mejora de la aplicación móvil de nutrición, basándose en las conclusiones obtenidas durante la evaluación del proyecto. Estas recomendaciones buscan optimizar la funcionalidad,

mejorar la experiencia del usuario y ampliar el impacto positivo de la aplicación en la vida de los usuarios.

1. **Implementación de funciones de configuración:** Uno de los aspectos que se debe abordar es la inclusión de una sección de configuración que permita a los usuarios personalizar la experiencia de uso. Esto incluiría la opción de elegir entre diferentes temas visuales, activar o desactivar notificaciones, y establecer recordatorios personalizados para ayudar a los usuarios a cumplir con sus planes nutricionales y objetivos de salud.

2. **Integración de recetas y favoritos:** Se planea agregar la posibilidad de que los usuarios puedan contribuir a la galería de recetas añadiendo sus propias creaciones. Además, se incluirá una funcionalidad que permita marcar recetas como favoritas para facilitar su acceso en el futuro. Esta característica no solo enriquecerá la experiencia del usuario, sino que también fomentará la creación de una comunidad más activa y participativa dentro de la aplicación.

3. **Ampliación de la base de datos de supermercados:** Actualmente, la aplicación está limitada a un solo supermercado. Se planea ampliar esta base de datos para incluir más establecimientos, lo que permitirá a los usuarios tener un abanico más amplio de opciones al generar sus listas de compra. Esto mejorará la usabilidad de la aplicación y su capacidad para adaptarse a las necesidades de un mayor número de usuarios.

4. **Personalización de la lista de la compra:** Para mejorar la experiencia de compra, se añadirá la opción de elegir sobre qué supermercado se desea generar la lista de la compra. De este modo, los usuarios podrán seleccionar su establecimiento de preferencia, haciendo el proceso de compra más eficiente y personalizado.

5. **Comparadores nutricionales de productos:** Una nueva funcionalidad que se considera implementar es la inclusión de comparadores nutricionales de productos. Esto permitirá a los usuarios evaluar y comparar los valores nutricionales de diferentes alimentos, ayudándoles a tomar decisiones más informadas sobre su alimentación.

6. **Calculadora de calorías de recetas:** Otra funcionalidad planificada es una calculadora de calorías para recetas, que permitirá a los usuarios estimar el aporte calórico de sus comidas antes de prepararlas. Esto será particularmente útil para aquellos que siguen dietas específicas y necesitan un control preciso de su ingesta calórica.

7. **Seguimiento de peso:** Para ofrecer una experiencia más completa en el cuidado de la salud, se incluirá una funcionalidad de seguimiento de peso. Los usuarios podrán registrar su peso de manera regular y seguir su progreso en función de sus objetivos de salud y nutrición.

8. **Mejora de la compatibilidad con dispositivos móviles:** Finalmente, se trabajará en mejorar la compatibilidad de la aplicación con un mayor número de dispositivos móviles y sistemas operativos. Esto asegurará que más usuarios puedan acceder a la aplicación sin importar el dispositivo que utilicen, ampliando así el alcance de la misma.

9. **Generación de menús semanales teniendo en cuenta alérgenos:** Para hacer la aplicación más inclusiva y segura, se añadirá la funcionalidad de generar menús semanales que tengan en cuenta los alérgenos específicos de cada usuario. Esto permitirá a las personas con alergias alimentarias recibir recomendaciones seguras y adaptadas a sus necesidades, mejorando la personalización y la seguridad de los planes nutricionales.

Con estas futuras mejoras, la aplicación seguirá evolucionando para ofrecer una experiencia más personalizada, inclusiva y eficiente, adaptándose a las necesidades y expectativas de los usuarios y manteniéndose a la vanguardia en el ámbito de la nutrición móvil.

Bibliografía

[1] Weight loss management app — Unimeal — unimeal.com. https://unimeal.com/. [Accessed 21-06-2024].

[2] J Arthur Harris and Francis G Benedict. A biometric study of human basal metabolism. *Proceedings of the National Academy of Sciences*, 4(12):370–373, 1918.

[3] MD Mifflin, ST St Jeor, LA Hill, BJ Scott, SA Daugherty, and YO Koh. A new predictive equation for resting energy expenditure in healthy individuals. *The American Journal of Clinical Nutrition*, 51(2):241–247, 1990.

[4] Beatriz Puente. Harris Benedict y su ecuación para medir el metabolismo — lineaysalud.com. https://www.lineaysalud.com/nutricion/calorias-harris-benedict, 2014. [Accessed 06-07-2024].

[5] Holzen Atocha Martinez-Garcia. Sistema experto para cálculo automático de calorías diarias basado en parámetros corporales. *Revista del centro de graduados e investigación*, 09 2018.

[6] Andrés Caldentey-Dos-Passos. Gestión personal de dietas saludables (i): menús, recetas y calendario. Trabajo de Fin de Grado, Universidad de Politécnica de Valencia, Valencia, 2024.

[7] Departamento de Fisiología, Facultad de Medicina, UNAM . Gasto energético y requerimientos nutricionales diarios. https://fisiologia.facmed.unam.mx/index.php/gasto-energetico-y-requerimientos-nutricionales-diarios/. [Accessed 06-07-2024].

[8] Organización de las Naciones Unidas para la Alimentación y la Agricultura. Parte II: Nutrición básica — fao.org. https://www.fao.org/4/w0073s/w0073s0c.htm#:~:text=Los%20factores%20generales%20m%C3%A1s%20importantes,y%20grasa%20en%20el%20cuerpo. [Accessed 15-07-2024].

[9] Erik Ramírez López, Nohemi Liliana Negrete Lopez, and Alexandra Tijerina Sáenz. El peso corporal saludable: Definición y cálculo en diferentes grupos de edad. *Revista salud pública y nutrición*, 13(4), 2012.

[10] Organización Mundial de la Salud. Obesidad y sobrepeso — who.int. https://www.who.int/es/news-room/fact-sheets/detail/obesity-and-overweight, 2024. [Accessed 06-07-2024].

[11] Nutrilio - Food Journal, Water & Weight Tracking — nutrilio.net. https://nutrilio.net/. [Accessed 21-06-2024].

[12] Fitatu: the simplest calorie calculator. https://www.fitatu.com/. [Accessed 21-06-2024].

[13] Fitia - Pierde o Gana Peso con Un Plan Inteligente — fitia.app. https://fitia.app/es/. [Accessed 21-06-2024].

[14] Git — git-scm.com. https://git-scm.com/. [Accessed 01-07-2024].

[15] Github: Let's build from here — github.com. https://github.com/. [Accessed 26-06-2024].

[16] Trello — trello.com. https://trello.com/. [Accessed 01-09-2024].

[17] Roberth G Figueroa, Camilo J Solís, and Armando A Cabrera. Metodologías tradicionales vs. metodologías ágiles. *Universidad Técnica Particular de Loja, Escuela de Ciencias de la Computación*, 9(1):1–10, 2008.

[18] Ken Schwaber and Jeff Sutherland. La guía de scrum. *Scrumguides. Org*, 1:21, 2013.

[19] Surendra M. Gupta, Yousef A.Y. Al-Turki, and Ronald F. Perry. Flexible kanban system. *International Journal of Operations & Production Management*, 19(10):1065–1093, Jan 1999.

[20] Figma: La herramienta de diseño de interfaz colaborativa — figma.com. https://www.figma.com/es-es/?context=confirmLocalePref. [Accessed 26-07-2024].

[21] Ashung. android-vector-drawable-figma. https://github.com/Ashung/android-vector-drawable-figma, 2020.

[22] Flowchart Maker & Online Diagram Software — app.diagrams.net. https://app.diagrams.net/. [Accessed 16-07-2024].

[23] Herramientas para desarrolladores de aplicaciones para dispositivos móviles Android – Android Developers — developer.android.com. https://developer.android.com/?hl=es-419. [Accessed 20-07-2024].

[24] MySQL :: MySQL Workbench — mysql.com. https://www.mysql.com/products/workbench/. [Accessed 20-07-2024].

[25] AWS | Servicio de bases de datos relacionales (RDS) — aws.amazon.com. https://aws.amazon.com/es/rds/. [Accessed 26-07-2024].

[26] ismaeldivita. chip-navigation-bar. https://github.com/ismaeldivita/chip-navigation-bar, 2022.

[27] CardView | Android Developers — developer.android.com. https://developer.android.com/reference/androidx/cardview/widget/CardView. [Accessed 21-07-2024].

[28] JUnit 5 — junit.org. https://junit.org/junit5/. [Accessed 01-08-2024].

[29] Jeffrey Rubin and Dana Chisnell. *Handbook of usability testing: How to plan, design, and conduct effective tests*. John Wiley & Sons, 2011.

[30] Jakob Nielsen. Why you only need to test with 5 users. https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/, 2000.

[31] Gobierno de España. Ley orgánica de protección de datos. Technical report, BOE-A-1999-23750). Retrieved from: https://www. boe. es/eli/es/lo/1999/12/13 . . . , 1999.

[32] Bettina Laugwitz, Theo Held, and Martin Schrepp. Construction and evaluation of a user experience questionnaire. In *HCI and Usability for Education and Work: 4th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society, USAB 2008, Graz, Austria, November 20-21, 2008. Proceedings 4*, pages 63–76. Springer, 2008.

[33] Salud - Desarrollo Sostenible — un.org. `https://www.un.org/sustainabledevelopment/es/health/`. [Accessed 20-07-2024].

APÉNDICE A
Casos de uso

Los casos de uso conectan los requisitos funcionales con el diseño y desarrollo del sistema. Los casos de uso son descripciones detalladas de cómo los usuarios interactuarán con el sistema para lograr un objetivo específico. Estos escenarios prácticos permiten identificar las acciones que deben llevarse a cabo, los actores involucrados, y las interacciones necesarias entre el usuario y el sistema.

La importancia de los casos de uso radica en su capacidad para traducir los requisitos funcionales, que suelen ser descripciones más abstractas y generales, en situaciones concretas y aplicables en el contexto del sistema. Al definir claramente qué tareas debe realizar el sistema desde la perspectiva del usuario, los casos de uso aseguran que todos los requisitos funcionales sean implementados de manera coherente y alineada con las expectativas de los usuarios.

Nombre	Registrar usuario
Descripción	El usuario no registrado se registra en la aplicación creando una nueva cuenta de usuario.
Precondición	El usuario no está registrado.
Secuencia principal	1. El usuario accede a la pantalla de registro. 2. El usuario introduce los datos (correo, nombre, apellido, contraseña). 3. El sistema muestra que los datos son correctos 4. El sistema guarda los datos del registro.
Alternativas/Errores	2.1 Si los datos no cumplen el formato adecuado, el sistema muestra un error y vuelve al paso 2. 3.1 Si sucede un error almacenando los datos del usuario, el sistema muestra un error y el caso de uso termina.
Postcondición	El nuevo usuario se almacena en el sistema.

Tabla A.1: Caso de uso: Registrar usuario

Nombre	Iniciar sesión
Descripción	El usuario se identifica en el sistema iniciando sesión para acceder a la aplicación.
Precondición	El usuario ya se ha registrado con una cuenta, pero no ha iniciado la sesión.
Secuencia principal	1. El usuario se encuentra en la pantalla de inicio de sesión. 2. El usuario introduce sus credenciales. 3. El sistema valida las credenciales comprobando que los datos sean correctos.
Alternativas/Errores	3.1 Si los datos no son correctos, el sistema muestra un error y vuelve al paso 2 hasta un máximo de 3 intentos.
Postcondición	El sistema inicia la sesión del usuario.

Tabla A.2: Caso de uso: Iniciar sesión

Nombre	Modificar perfil de usuario
Descripción	El usuario puede revisar y modificar sus datos personales y objetivo.
Precondición	El usuario ha iniciado sesión.
Secuencia principal	1. El usuario accede a la ventana de perfil desde la opción de la barra de navegación. 2. El sistema muestra por pantalla los datos del usuario. 3. El usuario modifica los datos. 4. El sistema almacena los cambios.
Alternativas/Errores	2.1 Si sucede un error al recuperar los datos, el sistema muestra un error y este caso de uso termina. 4.1 Si los datos no cumplen el formato correcto, el sistema muestra un error y se vuelve al paso 3.
Postcondición	Se guardan los cambios en el sistema.

Tabla A.3: Caso de uso: Modificar perfil de usuario

Nombre	Ingresar datos personales
Descripción	Cuando el usuario ha iniciado sesión por primera vez se le solicitarán sus datos fisonómicos.
Precondición	El usuario se ha registrado e inicia sesión.
Secuencia principal	1. El sistema pedirá los datos al usuario (peso, altura, ejercicio físico diario, restricciones alimenticias). 2. El usuario introduce los datos. 3. El sistema almacena los datos.
Alternativas/Errores	3.1 Si los datos no cumplen el formato correcto, el sistema muestra un error y se vuelve al paso 2.
Postcondición	Los datos se almacenan en el sistema.

Tabla A.4: Caso de uso: Ingresar datos personales

Nombre	Generar menú semanal automáticamente
Descripción	El sistema genera un menú semanal de manera automática.
Precondición	El usuario ha proporcionado sus datos fisonómicos.
Secuencia principal	1. Cada lunes el sistema genera un nuevo menú para la semana, a partir de las recetas de las fuentes y de las recetas propias del usuario. 2. El sistema notifica al usuario de que el nuevo menú está disponible.
Alternativas/Errores	1.1 Si sucede un error almacenando los datos del menú, el sistema muestra un error y el caso de uso termina.
Postcondición	Se genera el menú semanalmente.

Tabla A.5: Caso de uso: Generar menú semanal automáticamente

Nombre	Modificar menú semanal
Descripción	El usuario puede modificar los platos que aparecen en el menú generado aleatoriamente, cambiando así los que no sean de su agrado.
Precondición	El usuario debe haber iniciado sesión y el sistema tiene que haber generado un menú.
Secuencia principal	1. El sistema muestra los diferentes platos de todos los días del menú. 2. El usuario selecciona el plato que desea cambiar. 3. Elige si seleccionar uno de la galería de recetas o poner otro plato de manera aleatoria. 4. El sistema guarda los cambios en el menú.
Alternativas/Errores	1.1 Si sucede un error al recuperar los datos, el sistema muestra un error y este caso de uso termina. 4.1 Si sucede un error almacenando los datos del menú, el sistema muestra un error y el caso de uso termina.
Postcondición	El sistema guarda los cambios en el menú.

Tabla A.6: Caso de uso: Modificar menú semanal

Nombre	Generar lista de la compra
Descripción	En base a los platos que se encuentran en el menú semanal se generará una lista de los productos que el usuario debe comprar.
Precondición	Tener un menú semanal.
Secuencia principal	1. El sistema genera la lista de la compra para toda la semana. 2. En el caso de que un ingrediente se utilice en más de una receta, se tendrá en cuenta las cantidades para que no se compre más de lo necesario.
Alternativas/Errores	1.1 Si sucede un error al generar la lista, el sistema muestra un error y el caso de uso termina.
Postcondición	La lista de la compra se genera teniendo en cuenta todos los platos del menú semanal.

Tabla A.7: Caso de uso: Generar lista de la compra

Nombre	Modificar lista de la compra
Descripción	El usuario puede modificar la lista de la compra en base a los productos que ya tiene en casa o si quiere cambiarlo por algún equivalente.
Precondición	El usuario tiene un menú semanal y ya se ha generado la lista de la compra.
Secuencia principal	1. El sistema muestra los datos de la lista de la compra. 2. El usuario selecciona el producto a cambiar. 3. El usuario selecciona un producto para sustituirlo o simplemente lo elimina de la lista indicando que ya lo tiene en casa. 4. El sistema almacena los cambios e indica el precio total de la compra.
Alternativas/Errores	1.1 Si sucede un error al recuperar los datos, el sistema muestra un error y este caso de uso termina. 4.1 Si sucede un error almacenando los datos de la lista de la compra, el sistema muestra un error y el caso de uso termina.
Postcondición	El sistema almacena los cambios e indica el precio total de la compra.

Tabla A.8: Caso de uso: Modificar lista de la compra

Nombre	Calcular valor nutricional de las recetas
Descripción	El usuario carga sus propias recetas en el sistema y el sistema calcula el valor nutricional de estas.
Precondición	El usuario ha iniciado sesión.
Secuencia principal	1. El usuario selecciona el apartado de recetas en la barra de navegación. 2. Entra en el apartado de mis recetas. 3. Escoge la opción de añadir receta. 4. El sistema calcula el valor nutricional de la receta. 5. El sistema almacena los datos de la receta.
Alternativas/Errores	5.1 Si sucede un error almacenando los datos de la receta, el sistema muestra un error y el caso de uso termina.
Postcondición	Se almacena la nueva receta en el sistema.

Tabla A.9: Caso de uso: Calcular valor nutricional de las recetas

Nombre	Acceder a la galería de recetas
Descripción	El usuario puede acceder a la galería de las recetas almacenadas en el sistema.
Precondición	El usuario debe haber iniciado sesión.
Secuencia principal	1. El usuario se encuentra en el apartado de recetas en la barra de navegación. 2. El usuario elige la opción de galería. 3. El sistema carga las recetas.
Alternativas/Errores	3.1 Si sucede un error al recuperar los datos, el sistema muestra un error y este caso de uso termina.
Postcondición	El sistema muestra por pantalla las diversas recetas.

Tabla A.10: Caso de uso: Acceder a la galería de recetas

Nombre	Añadir receta
Descripción	El usuario desea agregar nuevas recetas a la aplicación para ampliar la variedad de opciones disponibles en su menú semanal.
Precondición	El usuario ha iniciado sesión en la aplicación y se encuentra en la sección de gestión de recetas.
Secuencia Principal	1. El usuario selecciona la opción de .Añadir Receta"desde la interfaz de usuario. 2. Ingresa los detalles de la receta, como nombre, ingredientes, instrucciones y valores nutricionales. 3. Opcionalmente, el usuario puede adjuntar imágenes o enlaces relacionados con la receta. 4. Confirma la adición de la receta.
Alternativas	1.1 Si el usuario decide cancelar la operación, se regresa a la interfaz principal de gestión de recetas sin realizar cambios.
Postcondición	La nueva receta se añade con éxito a la base de datos de la aplicación y está disponible para su inclusión en el menú semanal.

Tabla A.11: Caso de uso: Añadir receta

Nombre	Registrar Consumo Calórico y Nutricional Diario
Descripción	El usuario desea hacer un seguimiento de su consumo diario de calorías y verificar su progreso hacia los objetivos nutricionales establecidos.
Precondición	El usuario ha iniciado sesión en la aplicación y se encuentra en la sección de monitoreo de objetivos.
Secuencia Principal	1. El usuario accede a la sección de monitoreo de objetivos diarios. 2. Una vez creado el menú el usuario podrá ver una barra de progreso de las calorías diarias. 3. La pantalla mostrará los platos que se deben consumir en ese día y al marcarlos como consumidos la barra de progreso incrementará. 4. Al día siguiente se reinicia el proceso con los platos correspondientes.
Alternativas	(No hay alternativas o errores especificados)
Postcondición	El sistema muestra el progreso de las calorías consumidas a lo largo del día.

Tabla A.12: Caso de uso: Registrar Consumo Calórico y Nutricional Diario

Nombre	Enviar notificaciones
Descripción	La aplicación enviará notificaciones al usuario para ayudarle a seguir su plan nutricional y sus actividades relacionadas.
Precondición	El usuario ha dado permisos para recibir notificaciones y recordatorios.
Secuencia Principal	1. La aplicación envía notificaciones programadas para recordar al usuario sobre las comidas, cambios en el menú semanal o eventos importantes. 2. El usuario recibe notificaciones en tiempo real sobre nuevos mensajes, actualizaciones o eventos relevantes.
Alternativas	1.1 Si el usuario decide desactivar las notificaciones, dejará de recibir recordatorios programados.
Postcondición	El usuario recibe notificaciones y recordatorios según sus preferencias, lo que facilita el seguimiento de su plan nutricional y actividades relacionadas.

Tabla A.13: Caso de uso: Enviar notificaciones

Nombre	Configurar las notificaciones
Descripción	El usuario desea personalizar las configuraciones de notificaciones según sus preferencias y necesidades.
Precondición	El usuario ha iniciado sesión en la aplicación y se encuentra en la sección de configuración de notificaciones.
Secuencia Principal	1. El usuario accede a la sección de configuración de notificaciones desde la interfaz de usuario. 2. Configura las preferencias de notificación, como la frecuencia, tipo de recordatorios y horarios específicos. 3. Guarda las configuraciones personalizadas.
Alternativas	1.1 Si el usuario decide no realizar cambios, simplemente cierra la sección de configuración sin guardar.
Postcondición	Las configuraciones de notificaciones se actualizan según las preferencias del usuario, asegurando que reciba notificaciones de acuerdo con sus necesidades específicas.

Tabla A.14: Caso de uso: Configurar las notificaciones

Nombre	Cargar Datos en la Base de Datos (Supermercado)
Descripción	El administrador desea cargar nuevos datos en la base de datos de la aplicación a través de una acción específica al presionar un botón desde la vista de administrador.
Precondición	El administrador ha iniciado sesión en la aplicación y se encuentra en la interfaz de administración con acceso a la funcionalidad de carga de datos.
Secuencia Principal	1. El administrador accede a la interfaz de administración y selecciona la opción de Çargar Datos.º similar.
Alternativas	1.1 Si el administrador decide cancelar la operación, se regresa a la interfaz principal de administración sin realizar cambios en la base de datos.
Postcondición	Los nuevos datos se han cargado con éxito en la base de datos de la aplicación, y cualquier acción adicional o actualización en la interfaz de administración reflejará estos cambios.

Tabla A.15: Caso de uso: Cargar Datos en la Base de Datos (Supermercado)

Nombre	Cargar Datos en la Base de Datos (Calorías)
Descripción	El administrador desea cargar nuevos datos en la base de datos de la aplicación a través de una acción específica al presionar un botón desde la vista de administrador.
Precondición	El administrador ha iniciado sesión en la aplicación y se encuentra en la interfaz de administración con acceso a la funcionalidad de carga de datos.
Secuencia Principal	1. El administrador accede a la interfaz de administración y selecciona la opción de Çargar Datos.º similar.
Alternativas	1.1 Si el administrador decide cancelar la operación, se regresa a la interfaz principal de administración sin realizar cambios en la base de datos.
Postcondición	Los nuevos datos se han cargado con éxito en la base de datos de la aplicación, y cualquier acción adicional o actualización en la interfaz de administración reflejará estos cambios.

Tabla A.16: Caso de uso: Cargar Datos en la Base de Datos (Calorías)

Nombre	Cargar Datos en la Base de Datos (Recetas)
Descripción	El administrador desea cargar nuevos datos en la base de datos de la aplicación a través de una acción específica al presionar un botón desde la vista de administrador.
Precondición	El administrador ha iniciado sesión en la aplicación y se encuentra en la interfaz de administración con acceso a la funcionalidad de carga de datos.
Secuencia Principal	1. El administrador accede a la interfaz de administración y selecciona la opción de Çargar Datos.º similar.
Alternativas	1.1 Si el administrador decide cancelar la operación, se regresa a la interfaz principal de administración sin realizar cambios en la base de datos.
Postcondición	Los nuevos datos se han cargado con éxito en la base de datos de la aplicación, y cualquier acción adicional o actualización en la interfaz de administración reflejará estos cambios.

Tabla A.17: Caso de uso: Cargar Datos en la Base de Datos (Recetas)

Nombre	Actualizar Datos en la Base de Datos (Supermercado)
Descripción	El administrador desea actualizar datos en la base de datos de la aplicación a través de una acción específica al presionar un botón desde la vista de administrador.
Precondición	El administrador ha iniciado sesión en la aplicación y se encuentra en la interfaz de administración con acceso a la funcionalidad de carga de datos.
Secuencia Principal	1. El administrador accede a la interfaz de administración y selecciona la opción de .Actualizar Datos.º similar.
Alternativas	1.1 Si el administrador decide cancelar la operación, se regresa a la interfaz principal de administración sin realizar cambios en la base de datos.
Postcondición	Los nuevos datos se han actualizado con éxito en la base de datos de la aplicación, y cualquier acción adicional o actualización en la interfaz de administración reflejará estos cambios.

Tabla A.18: Caso de uso: Actualizar Datos en la Base de Datos (Supermercado)

Nombre	Regenerar Menú
Descripción	El usuario desea generar automáticamente otro menú semanal diferente para variar las opciones alimenticias.
Precondición	El usuario ha iniciado sesión en la aplicación y se encuentra en la sección de gestión de menús.
Secuencia Principal	1. El usuario selecciona la opción "Generar Otro Menú"desde la interfaz de gestión de menús. 2. La aplicación utiliza algoritmos o reglas predefinidas para generar un nuevo menú semanal basado en las preferencias y restricciones del usuario. 3. El usuario visualiza el nuevo menú propuesto antes de confirmar. 4. Opcionalmente, el usuario puede realizar ajustes manuales en el menú propuesto antes de aceptarlo.
Alternativas	3.1 Si el usuario no está satisfecho con el menú generado, puede optar por generar otro menú o realizar ajustes manuales.
Postcondición	El usuario tiene la opción de aceptar el nuevo menú automáticamente generado o realizar modificaciones antes de aceptarlo. El menú aceptado se actualiza en la aplicación.

Tabla A.19: Caso de uso: Regenerar Menú

Nombre	Crear Menú Manualmente
Descripción	El usuario desea tener un control total sobre la composición de su menú semanal y opta por crearlo manualmente.
Precondición	El usuario ha iniciado sesión en la aplicación y se encuentra en la sección de gestión de menús.
Secuencia Principal	1. El usuario selecciona la opción Çrear Menú Manualmente"desde la interfaz de gestión de menús. 2. Se presenta una interfaz que permite al usuario seleccionar y añadir manualmente los platos para cada día de la semana. 3. El usuario busca, elige y añade los platos deseados para cada día, considerando sus preferencias y objetivos nutricionales. 4. Confirma y guarda el menú creado manualmente.
Alternativas	1.1 Si el usuario decide cancelar la operación, puede optar por generar automáticamente otro menú o aceptar un menú predefinido.
Postcondición	El menú creado manualmente se guarda en la base de datos de la aplicación y se implementa en el plan nutricional del usuario.

Tabla A.20: Caso de uso: Crear Menú Manualmente

Nombre	Ver perfil usuario
Descripción	El usuario puede ver tanto los datos personales como fisionómicos introducidos en su perfil en la aplicación.
Precondición	El usuario debe estar registrado y haber iniciado la sesión.
Secuencia Principal	1. El usuario accede a la pestaña de perfil. 2. El sistema carga y muestra los datos.
Alternativas	2.1 Si sucede un error mostrando los datos del usuario, el sistema muestra un error y el caso de uso termina.
Postcondición	El sistema muestra los datos del usuario correctamente.

Tabla A.21: Caso de uso: Ver perfil usuario

Nombre	Ver lista de la compra
Descripción	El usuario puede ver la lista de la compra generada por el sistema.
Precondición	El usuario debe estar registrado y haber iniciado la sesión.
Secuencia Principal	1. El usuario accede a la pestaña de la lista de la compra. 2. El sistema muestra por pantalla los datos de la lista de la compra.
Alternativas	2.1 Si sucede un error mostrando los datos de la lista de la compra, el sistema muestra un error y el caso de uso termina.
Postcondición	El sistema muestra los datos de la lista de la compra correctamente.

Tabla A.22: Caso de uso: Ver lista de la compra

Nombre	Ver menú semanal
Descripción	El usuario puede ver el menú semanal generado por el sistema.
Precondición	El usuario debe estar registrado y haber iniciado la sesión.
Secuencia Principal	1. El usuario accede a la pestaña del menú semanal. 2. El sistema muestra por pantalla los datos del menú semanal.
Alternativas	2.1 Si sucede un error mostrando los datos del menú semanal, el sistema muestra un error y el caso de uso termina.
Postcondición	El sistema muestra los datos del menú semanal correctamente.

Tabla A.23: Caso de uso: Ver menú semanal

Nombre	Gestionar usuario
Descripción	El usuario puede modificar su información personal, como nombre, correo electrónico, contraseña, y preferencias de la aplicación.
Precondición	El usuario debe estar registrado, haber iniciado la sesión y tener permisos para modificar sus datos.
Secuencia Principal	1. El usuario accede a la sección de configuración o perfil. 2. El usuario selecciona la opción para editar su información personal. 3. El sistema permite al usuario modificar los campos necesarios. 4. El usuario guarda los cambios realizados. 5. El sistema actualiza la información del usuario en la base de datos.
Alternativas	4.1 Si ocurre un error al guardar los cambios, el sistema muestra un mensaje de error y el caso de uso termina sin realizar modificaciones. 3.1 Si el usuario intenta modificar campos restringidos o inválidos, el sistema muestra un mensaje de advertencia y solicita correcciones.
Postcondición	El sistema guarda correctamente los cambios realizados en la información del usuario.

Tabla A.24: Caso de uso: Gestionar usuario

Nombre	Ver receta
Descripción	El usuario puede visualizar los detalles de una receta específica, incluyendo su descripción, valor nutricional, ingredientes, y pasos de preparación.
Precondición	El usuario debe estar registrado y haber iniciado la sesión.
Secuencia Principal	1. El usuario selecciona una receta de la galería o del menú semanal. 2. El sistema muestra por pantalla todos los detalles de la receta seleccionada.
Alternativas	2.1 Si la receta no se encuentra disponible o ocurre un error, el sistema muestra un mensaje de error y el caso de uso termina.
Postcondición	El sistema muestra los detalles de la receta seleccionada correctamente.

Tabla A.25: Caso de uso: Ver receta

APÉNDICE B
Objetivos de Desarrollo Sostenible

B.1 Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**			X	
ODS 3. **Salud y bienestar.**	X			
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**		X		
ODS 9. **Industria, innovación e infraestructuras.**				X
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**	X			
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Tabla B.1: Grado de relación del trabajo con los ODS.

B.2 Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Los Objetivos de Desarrollo Sostenible (ODS) son un conjunto de 17 metas globales adoptadas por las Naciones Unidas en 2015 como parte de la Agenda 2030 para el Desarrollo Sostenible. Son objetivos que pretenden los mayores desafíos, económicos, sociales y ambientales a los que se enfrenta el mundo [33].

La aplicación desarrollada a lo largo de este TFG se enfoca en facilitar a los usuarios un acercamiento hacia una vida más saludable a través de la nutrición. Como se puede observar en la Tabla B.1, el *software* desarrollado contribuye tanto de manera directa, como indirecta al desarrollo de varios Objetivos de desarrollo sostenible. Los ODS a los que proporciona una mayor aportación son:

- **ODS 3. Salud y bienestar:** este objetivo busca garantizar una vida sana y promover el bienestar para todos en todas las edades. Es el ODS más directamente relacionado con el proyecto. Esto se debe a que a través de la aplicación, se proporciona al usuario menús personalizados que fomentan un peso saludable y una alimentación equilibrada. De esta manera, la aplicación contribuye a la prevención de enfermedades como la obesidad, la diabetes y las enfermedades cardiovasculares, que están relacionadas con una mala nutrición.
- **ODS 12. Producción y consumo responsables:** este objetivo consiste en garantizar modalidades de consumo y producción sostenibles. En este caso, es un ODS altamente relacionado con la aplicación, ya que una de sus funcionalidades principales es la generación automática de una lista de la compra basada en el menú semanal. Es decir, se busca facilitar al usuario la organización a la hora de hacer la compra, proporcionando una lista optimizada que fomenta un consumo consciente y reduce el desperdicio de alimentos.

Por otra parte, también encontramos ODS a los que se contribuye, aunque en menor medida.

- **ODS 2. Hambre cero:** Este TFG puede ayudar a reducir la pobreza al facilitar el acceso a una alimentación saludable y asequible. Al generar menús semanales basados en un cálculo preciso de las necesidades nutricionales y al crear listas de compras optimizadas en tiendas accesibles, la aplicación puede ayudar a las personas a gestionar mejor su presupuesto alimentario. Esto es especialmente relevante para aquellos con recursos limitados, ya que pueden planificar sus comidas de manera eficiente, evitando gastos innecesarios y reduciendo el desperdicio de alimentos. Además, al mejorar la salud a través de una buena nutrición, se pueden reducir los costos asociados con problemas de salud, lo que contribuye indirectamente a aliviar la pobreza.
- **ODS 8. Trabajo decente y crecimiento económico:** El desarrollo y mantenimiento de la aplicación pueden generar empleo en el sector tecnológico, desde programadores y diseñadores hasta especialistas en soporte técnico. A medida que la aplicación evoluciona, también puede requerir la colaboración de nutricionistas, especialistas en salud, y otros profesionales, creando oportunidades laborales en estos campos. Además, si la aplicación crece, puede abrir nuevas áreas de negocio, como servicios de asesoría personalizada o asociaciones con proveedores de alimentos, lo que contribuye al crecimiento económico y la creación de empleo en distintas industrias.