UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Gestión personal de dietas saludables (I): menús, recetas y calendario

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Caldentey Dos Passos, Andrés Ignacio

Tutor/a: Canós Cerdá, José Hilario

CURSO ACADÉMICO: 2023/2024

Resum

Aquest Treball de Fi de Grau (TFG) té com a objectiu el desenvolupament d'una aplicació de nutrició que empre dades extretes de pàgines web. Primordialment, es pretén obtindre informació sobre els productes disponibles en supermercats per a gestionar 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. A més, s'incorporarà la capacitat d'afegir receptes automàticament des de la web, mitjançant l'entrenament d'una intel·ligència artificial per a interpretar receptes en llenguatge natural i convertir-les en consultes per a la base de dades, amb la finalitat de poder agregar receptes provinents de diverses fonts. En este treball, que és complementari d'un altre TFG, s'abordarà la problemàtica descrita anteriorment, a més de la gestió de receptes, menús setmanals i alertes de calendari.

Paraules clau: alimentació, dieta saludable, extracció d'informació, intel·ligència artificial

Resumen

Este Trabajo de Fin de Grado (TFG) tiene como objetivo el desarrollo de una aplicación de nutrición que emplee datos extraídos de páginas web. Primordialmente, se pretende obtener información sobre los productos disponibles en supermercados para gestionar sus precios y ofrecer a los usuarios una perspectiva realista del costo 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. Además, se incorpora la capacidad de añadir recetas automáticamente desde la web, mediante el entrenamiento de una inteligencia artificial para interpretar recetas en lenguaje natural y convertirlas en consultas para la base de datos, con el fin de poder agregar recetas provenientes de diversas fuentes. En este trabajo, que es complementario de otro TFG, se abordará la problemática descrita anteriormente, además de la gestión de recetas, menús semanales y alertas de calendario.

Palabras clave: alimentación, dieta saludable, extracción de información, inteligencia artificial

Abstract

This Bachelor's Thesis aims to develop a nutrition application that utilizes data extracted from websites. The primary goal is to obtain information about supermarket products to manage prices and offer users a realistic view of the cost associated with maintaining a balanced diet. Since supermarket websites often lack detailed nutritional information, this data will be collected from other sources. Additionally, the application will include the ability to automatically add recipes from the web by training an artificial intelligence to interpret natural language recipes and convert them into database queries. This project, complementary to another TFG, will address the aforementioned issues, as well as the management of recipes, weekly menus, and calendar alerts.

Key words: alimentation, healthy diet, information extraction, artificial Intelligence

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Este Trabajo de Fin de Grado (TFG) se centra en el desarrollo de una aplicación de nutrición que pretende superar estas limitaciones comunes mediante el uso de herramientas tecnológicas avanzadas. El objetivo principal es mejorar la gestión diaria de la alimentación de los usuarios, proporcionando no solo orientación nutricional, sino también información actualizada sobre los precios de los productos en distintos supermercados. De esta manera, la aplicación permitirá a los usuarios adaptar su dieta no solo a sus necesidades calóricas y preferencias alimentarias, sino también a su presupuesto, algo que resulta especialmente relevante en el contexto económico actual.

A diferencia de las aplicaciones de nutrición existentes en el mercado, este proyecto se distingue por la integración de datos reales y constantemente actualizados sobre precios de alimentos, lo que ofrece una herramienta práctica para la planificación económica de la dieta. Para lograr esto, se emplearán técnicas avanzadas de *web scraping*, que permiten la recolección automatizada de información desde diversas fuentes en Internet, junto con procesamiento de lenguaje natural (PLN) e inteligencia artificial (IA). Estas tecnologías se utilizarán para analizar y organizar grandes volúmenes de datos, extrayendo información relevante sobre productos alimenticios y recetas disponibles en línea.

Además, una de las funcionalidades clave de la aplicación será el desarrollo de un generador automático de menús semanales. Este generador tendrá en cuenta no solo las necesidades calóricas individuales de los usuarios, sino también su presupuesto y posibles restricciones alimentarias, como alergias o preferencias dietéticas específicas. Una vez generado el menú, la aplicación también creará una lista de la compra correspondiente, facilitando así la planificación de las compras semanales y ayudando a los usuarios a mantenerse dentro de su presupuesto alimentario sin comprometer su salud.

El proyecto incluye el diseño de una interfaz de usuario amigable e intuitiva, que hará que la experiencia sea accesible y personalizada. La combinación de una sólida base tecnológica con un enfoque centrado en el usuario pretende ofrecer una herramienta que optimice tanto los aspectos nutricionales como los económicos de la alimentación diaria, contribuyendo así a un estilo de vida más saludable y equilibrado para los usuarios. Este TFG, por lo tanto, no solo tiene un componente técnico significativo, sino también un fuerte enfoque en la mejora del bienestar general, respondiendo a la creciente demanda de soluciones que faciliten una vida más saludable en todos los sentidos.

1.1 Motivación

El principal interés en este proyecto es brindar una aplicación completamente funcional que apoye la nutrición de los usuarios. Aunque actualmente existen diversas apli-

caciones similares en el mercado, nuestro objetivo es ofrecer una experiencia diferente y más personalizada a través de los datos reales que hay en los supermercados y de mecanismos de automatización. También nos motiva explorar el mundo de la inteligencia artificial y su capacidad para procesar lenguaje natural. Queremos experimentar cómo podemos integrar diversas fuentes de sitios web en un esquema común para proporcionar una interfaz transparente y amigable para el usuario. Por último, nos interesa profundizar en el campo de la algoritmia; no es suficiente con extraer datos, queremos procesarlos y presentar resultados de mayor interés y utilidad para nuestros usuarios.

1.2 Objetivos

El objetivo general de este trabajo de fin de grado es construir una aplicación siguiendo todos los pasos establecidos para el desarrollo de software. Para lograrlo, hemos desglosado el proyecto en los siguientes objetivos específicos:

1. Modelo de requisitos: Realizar un estudio de mercado y desarrollar los diferentes diagramas de la arquitectura de la aplicación.
2. Construcción de la arquitectura del sistema: Definir los componentes del sistema, las tecnologías y herramientas a utilizar.
3. Desarrollo general de la aplicación: Implementar todas las funcionalidades necesarias para que la aplicación sea completamente operativa y eficiente, incluyendo la estructura y gestión de la base de datos.
4. Implementación de sistemas de extracción de datos: Utilizar técnicas ETL (*Extract*, *Transform*, *Load*) para *web scraping*.
5. Desarrollo de algoritmos de procesamiento: Crear algoritmos para el procesamiento y análisis de la base de datos.
6. Desarrollo de interfaces de usuario: Diseñar y construir interfaces de usuario amigables y eficientes que proporcionen una experiencia de usuario óptima.
7. Análisis y evaluación: Analizar los resultados obtenidos, identificar posibles mejoras y formular conclusiones.

1.3 Colaboración

Este trabajo de fin de grado se complementa con otro TFG titulado "Gestión personal de dietas saludables (II): Interfaz de usuario y gestión de usuarios y listas de compra"[6]. Ambos proyectos, aunque independientes, se desarrollan de manera conjunta para abordar integralmente la problemática descrita, dividiendo las tareas en dos áreas interrelacionadas, alineadas con los objetivos específicos planteados:

1. Extracción de datos y gestión de recetas y menús: Este aspecto se aborda en el presente TFG y se enfoca en la implementación de sistemas de extracción de datos mediante técnicas ETL, incluyendo *web scraping* y la gestión de recetas. Además, se desarrollan los algoritmos necesarios para la creación automática y gestión del menú semanal, tal como se establece en los objetivos específicos relacionados con el desarrollo de algoritmos de procesamiento y la implementación de sistemas de extracción de datos.

2. Diseño e implementación de interfaces de usuario y gestión de usuarios y listas de compra: Este componente, desarrollado en la contraparte del proyecto, se centra en el diseño de interfaces de usuario amigables y eficientes, en línea con el objetivo específico de desarrollar interfaces de usuario óptimas. También incluye la gestión de usuarios y listas de compra, aspectos que requieren una sólida interconexión con la base de datos y los controladores del sistema, definidos y construidos en este TFG bajo los objetivos de construcción de la arquitectura del sistema y desarrollo general de la aplicación.

La colaboración entre ambos trabajos se refleja en la necesidad de una integración fluida entre la gestión de datos y el *front-end*, asegurando que las funcionalidades implementadas en este TFG, como la creación y gestión del menú semanal, se comuniquen eficazmente con las interfaces de usuario desarrolladas en el otro TFG. En este documento, se detallarán todos los aspectos relacionados con la gestión de datos, desde la extracción de información hasta la generación automática de menús semanales. Esto incluye la implementación de *scrapers*, la creación de los controladores para la interacción con el *front-end*, y el desarrollo de algoritmos que aseguren la precisión y eficiencia del sistema, cumpliendo así con los objetivos generales y específicos del proyecto.

Además, el análisis y evaluación de los resultados obtenidos a lo largo de este TFG permitirá identificar oportunidades de mejora que pueden ser integradas en la interfaz de usuario y en la gestión de usuarios y listas de compra, fortaleciendo la sinergia entre ambos proyectos. Esta colaboración asegura una experiencia de usuario integral y optimizada, alineada con la meta de desarrollar una aplicación operativa y eficiente que cumpla con las expectativas establecidas.

1.4 Estructura del documento

La memoria del Trabajo de Fin de Grado se estructura en varios capítulos, comenzando con una Introducción donde se presenta el contexto, los objetivos y la motivación del proyecto. A continuación, se expone una revisión de los Conceptos Previos y Trabajos Relacionados, centrándose en técnicas de *web scraping* e inteligencia artificial aplicadas a la nutrición. En el Diseño y Arquitectura de la Aplicación, se describe la estructura del sistema y las tecnologías empleadas. La Metodología utilizada en el desarrollo combina enfoques tradicionales y ágiles.

El capítulo de Análisis y Especificación de Requisitos define el público objetivo y los requisitos del sistema, mientras que en el Desarrollo se detallan los *scrapers* y algoritmos implementados. La Implementación aborda la parte técnica, incluyendo la base de datos y patrones de diseño aplicados.

En el apartado de *Testing*, se describen las pruebas realizadas para asegurar la calidad del software. Finalmente, los Resultados evalúan el funcionamiento del sistema, en particular de los *scrapers* y la inteligencia artificial, y el capítulo de Conclusiones y Trabajo Futuro cierra la memoria con un resumen de los logros y sugerencias para futuras mejoras. Los Apéndices proporcionan información adicional y la relación del proyecto con los Objetivos de Desarrollo Sostenible (ODS).

CAPÍTULO 2
Conceptos previos y trabajos relacionados

Este capítulo introduce las técnicas de *web scraping* y su importancia en la recolección automatizada de datos desde sitios web. Se abordan los aspectos legales y éticos involucrados, así como los métodos y técnicas comúnmente empleados en esta práctica. Además, se exploran las herramientas y bibliotecas más utilizadas para el *scraping*. Se dedica un enfoque especial al rol del procesamiento de lenguaje natural (PLN) en el *web scraping*, destacando cómo la inteligencia artificial puede mejorar la extracción y comprensión de datos no estructurados o semi-estructurados, como texto en páginas web. Finalmente, se analiza el estado del arte en aplicaciones móviles centradas en la nutrición, comparando sus principales funcionalidades con las del proyecto desarrollado, y evaluando cómo el PLN puede contribuir a mejorar la precisión y eficiencia en la recopilación y procesamiento de información en este ámbito.

2.1 Técnicas de *Web scraping*

2.1.1. Definición

El *web scraping* es una técnica que consiste en extraer información de sitios web de manera automatizada utilizando software especializado. Este proceso implica el uso de programas que navegan por las páginas web y recolectan datos específicos de su contenido. Según el artículo "*Web scraping*"de Jaime López,[16] el *web scraping* es el proceso de recolección de información de manera automatizada a partir de la estructura HTML de las páginas web. Por esta razón, el *web scraping* se utiliza en diversas aplicaciones como la recopilación de datos para investigación, la monitorización de precios y productos en comercio electrónico, la agregación de contenido y el análisis de la competencia. Este método permite obtener grandes volúmenes de datos de manera eficiente y rápida, facilitando el análisis y la toma de decisiones basadas en información actualizada.

2.1.2. Métodos y Técnicas de *scraping*

Parsing de HTML

El *parsing* de HTML[22] es la técnica fundamental para analizar y extraer datos de documentos HTML. Consiste en interpretar el código HTML de una página web para identificar y extraer la información deseada. Utilizando bibliotecas como BeautifulSoup

en Python, los desarrolladores pueden navegar por el árbol de documentos HTML y seleccionar elementos específicos como etiquetas, clases, o identificadores.

Selectores CSS y XPath

Los selectores de *Cascading Style Sheets*[26] (CSS) y XPath[27] son herramientas poderosas para localizar y extraer información específica de una página web. Los selectores CSS permiten seleccionar elementos basándose en sus clases, identificadores o nombres de etiqueta, lo que facilita la identificación de elementos en el árbol del documento. Por otro lado, XPath, abreviación de XML *Path Language*, utiliza una sintaxis de consulta para navegar por el árbol XML/HTML y seleccionar nodos basados en una variedad de criterios. Estas técnicas son esenciales para lograr precisión en la extracción de datos, permitiendo a los desarrolladores dirigirse exactamente a los elementos deseados dentro de un documento web.

Scraping de Contenido Dinámico

El *scraping* de contenido dinámico implica técnicas para manejar y extraer datos de páginas web que cargan contenido a través de JavaScript y AJAX. Herramientas como Selenium[25] o Puppeteer[7] permiten automatizar navegadores webs completos, lo que permite interactuar con la página, esperar a que se cargue el contenido dinámico, y luego extraer la información. Esto es crucial para sitios web modernos que no cargan todos los datos inicialmente.

2.1.3. Herramientas y Bibliotecas Comunes

Scrapy[24] es un *framework* de *scraping* en Python diseñado para proyectos a gran escala. Ofrece herramientas robustas para la extracción de datos, procesamiento y almacenamiento, permitiendo definir reglas de rastreo y gestionar múltiples solicitudes simultáneamente. Selenium[25], por otro lado, permite automatizar navegadores web, lo cual es esencial para el *scraping* de contenido dinámico cargado con JavaScript. Esta herramienta puede interactuar con elementos de la página, enviar formularios y esperar a que se cargue el contenido antes de extraer los datos. Puppeteer[7], una biblioteca de Node.js, proporciona una API de alto nivel para controlar Chrome o Chromium, siendo ideal para extraer datos de páginas web complejas que requieren interacción con el navegador, como hacer clic en botones y navegar a través de múltiples páginas.

Estas herramientas y bibliotecas son fundamentales para llevar a cabo proyectos de *web scraping* de diferentes escalas y complejidades. La elección de la herramienta adecuada depende de las necesidades específicas del proyecto, la estructura del sitio web y el tipo de contenido a extraer.

2.1.4. Desafíos y Problemas Comunes en *Web Scraping*

Cambios en la Estructura HTML

Los cambios en la estructura HTML de las páginas web pueden hacer que los *scrapers* dejen de funcionar, ya que dependen de una estructura específica. Se identifican etiquetas o nombres de variables y se navega a través del HTML. Cualquier cambio, por mínimo que sea, puede dañar el *scraper*. Para mitigar este problema, es recomendable diseñar *scrapers* simples y fáciles de modificar. Además, es importante realizar monitoreos constantes y actualizar el *scraper* cuando sea necesario.

Restricciones de Velocidad

Realizar demasiadas solicitudes en un corto período puede resultar en bloqueos por parte del sitio web. Una manera de evitar esto es añadir retrasos manualmente para simular el comportamiento humano, evitando así ser detectado como una máquina. También es útil cambiar de IP ocasionalmente, realizando el *scraping* desde diferentes redes para distribuir las solicitudes y reducir el riesgo de ser bloqueado.

2.1.5. Aspectos Legales y Éticos

El *web scraping* está sujeto a diversas normativas legales en Europa. Por ejemplo, el Reglamento General de Protección de Datos (GDPR)[21] protege los datos personales de los ciudadanos y regula cómo se pueden recopilar, procesar y almacenar. El *scraping* de datos personales sin consentimiento puede violar esta normativa. Además, los términos de servicio de los sitios web pueden prohibir el *scraping* sin permiso, y violarlos puede resultar en acciones legales o bloqueos de acceso.

En este proyecto, se usarán datos extraídos de la página de Mercadona [18] con fines educativos, lo cual no infringe las normativas estipuladas en los términos y condiciones de Mercadona[17], siempre que el uso sea personal y no comercial. Según dichos términos de Mercadona, "puede utilizar la información que está disponible en el sitio Mercadona únicamente para uso personal y no lo puede utilizar para su uso comercial". En caso de querer usar esta aplicación de manera comercial, se debe llegar a un acuerdo con Mercadona o con cualquier otro supermercado cuyos datos se puedan utilizar. Esto asegurará que cualquier uso comercial cumpla con las normativas y términos de servicio correspondientes.

Además de las consideraciones legales, es crucial adoptar prácticas éticas en el *web scraping*. Esto implica respetar los términos de servicio de los sitios web, ser transparente sobre el uso de los datos recolectados, y evitar prácticas que puedan dañar la infraestructura del sitio web, como realizar demasiadas solicitudes en un corto período de tiempo. La ética en el *scraping* implica actuar de manera responsable y respetuosa, considerando tanto los derechos de los propietarios de los sitios web como el bienestar de los usuarios cuyos datos puedan ser recolectados. Esto significa no solo cumplir con las normativas legales vigentes, sino también aplicar principios de integridad y respeto en todas las actividades de *scraping*, asegurando que las prácticas utilizadas no invadan la privacidad ni comprometan la seguridad de los datos personales. El equilibrio entre la obtención de información útil y el respeto por los derechos de los usuarios y propietarios de sitios web es esencial para llevar a cabo un *scraping* responsable y sostenible.

2.2 Inteligencia Artificial para *Web Scraping*

El proceso de *web scraping* puede estructurarse efectivamente utilizando el modelo ETL (*Extract-Transform-Load*), que consiste en extraer datos, transformarlos en un formato adecuado y cargarlos en un sistema de almacenamiento o base de datos. Uno de los mayores desafíos de este proceso se presenta en la fase de "Transform", donde los datos extraídos de diversas fuentes deben ser procesados y convertidos a un formato unificado que pueda ser utilizado de manera coherente.

Dado que la información disponible en Internet está estructurada de formas muy variadas, es fundamental aplicar técnicas de mapeo que permitan adaptar los datos al formato deseado. Aquí es donde la inteligencia artificial (IA) juega un papel crucial, especialmente en el campo del procesamiento de lenguaje natural (PLN). La IA mejora

significativamente la capacidad de analizar datos no estructurados obtenidos de sitios web, facilitando su transformación en información estructurada y utilizable.

El PLN con IA implica el uso de algoritmos de aprendizaje automático para interpretar y comprender el lenguaje humano en su forma escrita o hablada. Esto es esencial para convertir datos no estructurados, como textos extraídos de sitios web, en formatos estructurados que puedan ser almacenados y analizados eficientemente. Según Domínguez [12], el PLN utiliza técnicas avanzadas de aprendizaje para analizar y procesar textos, lo que resulta fundamental en el contexto del *web scraping*.

Una de las tecnologías más avanzadas en el procesamiento de lenguaje natural es ChatGPT[20]. Esta herramienta puede ser utilizada para estructurar datos de acuerdo a nuestras necesidades a través de técnicas de *prompt engineering*, que permiten guiar el modelo de lenguaje hacia la obtención de resultados específicos. Aunque el uso de ChatGPT en el procesamiento de datos tiene sus propios desafíos, su capacidad para interpretar y generar lenguaje humano de manera coherente y significativa será aprovechada a lo largo de este proyecto para mejorar la transformación de datos extraídos en procesos de *web scraping*.

2.3 Estado del arte

Se procederá a realizar un estudio del estado del arte. Este análisis se enfoca en evaluar las aplicaciones móviles más relevantes en el ámbito de la nutrición y el fomento de hábitos alimentarios saludables. Se llevó a cabo una investigación detallada para identificar las aplicaciones más destacadas que presentan funcionalidades similares a las que pretendemos desarrollar en nuestro proyecto. Como resultado, se seleccionaron Nutrilio, Fitatu, Fitia y Unimeal como los competidores principales en este campo.

Es importante señalar que todas las aplicaciones son gratis desde Google Play Store. De este modo, se ha logrado examinar exhaustivamente cada una de ellas, con el fin de identificar y distinguir las características que ofrecen. El propósito de este estudio es determinar cuál sería la posición de nuestra aplicación en comparación con las otras, destacando tanto sus fortalezas como debilidades y analizando su cabida en el mercado.

2.3.1. Análisis de las aplicaciones

Nutrilio

Nutrilio[2] es una *app* creada para el control de alimentos, consumo de agua y peso corporal. Facilita llevar un registro de comidas para cada día, adaptar el seguimiento y acceder a detalles sobre la dieta y hábitos de vida del usuario. Además, la aplicación permite registrar factores como el estado emocional, la actividad física y ciertos síntomas de salud. Entre sus funciones se encuentran el monitoreo de peso y hidratación, un modo oscuro, el análisis detallado de comidas.

A pesar de que Nutrilio ofrece una amplia gama de funciones, sin un conocimiento básico en nutrición, puede ser complicado para el usuario aprovechar al máximo sus beneficios. Fuera de registrar comidas, rutinas de ejercicio o el consumo de agua, el usuario puede limitarse a ingresar sus datos semanales, generando estadísticas que deberá interpretar por su cuenta.

Fitatu

Fitatu se trata de una aplicación de monitoreo de la plan nutricional y consumo. Asiste a sus consumidores en aumentar volumen muscular, bajar peso y mantener una dieta saludable. Tiene registradas gran cantidad de alimentos y recetas con la posibilidad de leer el código de barras, también se conecta a otras aplicaciones para hacer deporte y sigue con precisión tu alimentación. Asimismo, posibilita la compartición de dietas, establecer metas personalizadas y supervisar el progreso mediante gráficos e informes.

Esta aplicación permite programar un menú para las siguientes semanas, utilizando los platos que tienen registrados. De este modo el se le crea al usuario una lista con los productos necesarios para sus recetas.

Fitia

Fitia[1] auxilia al usuario planificando las comidas y registrando calorías. Asimismo, se puede personalizar su objetivo nutricional en base a las preferencias, metas y datos personales, tales como la edad, altura, peso y la cantidad de ejercicio.

Se caracteriza por su amplia base de datos de productos y platos aprobados, así como su administración de los alimentos del consumidor, otorgando también recomendaciones ingeniosas

Unimeal

Unimeal[5] es una *app* enfocada en la gestión del peso, proporcionando planes alimenticios a medida, rutinas de ejercicios para para eliminar la grasa corporal y asesoramiento profesional disponible las 24 horas. Implementado para asistir al usuario en el logro de sus metas de salud y nutrición, mediante planes de alimentación ajustados a sus necesidades y preferencias, instrucciones detalladas de recetas y formación sobre hábitos de vida saludables.

Es posible que esta aplicación sea la más completa en términos de funcionalidades, ya que ofrece un plan de comidas personalizado al usuario, además de facilitar la automatización de la lista de compras basada en el plan semanal.

2.3.2. Comparativa entre las aplicaciones

En esta sección se lleva a cabo un estudio comparativo de las características presentes las aplicaciones revisadas. El estudio se realiza considerando varios criterios fundamentales que permiten determinar las fortalezas y debilidades de cada aplicación en relación con los objetivos de nuestra aplicación.

1. **Usabilidad:** Se examina lo fácil e intuitivo que es para los usuarios interactuar con la aplicación. Se toman en cuenta aspectos como la claridad de la interfaz, la facilidad de acceso a las distintas funciones y la comodidad de uso para diferentes tipos de usuarios.

2. **Funcionalidades clave:** Este criterio analiza las herramientas y características tanto básicas como avanzadas que ofrece cada aplicación, desde la configuración de perfiles hasta la programación automática de menús semanales y el seguimiento de hábitos alimenticios.

3. **Eficiencia operativa:** Se evalúa la rapidez y efectividad con la que cada aplicación lleva a cabo sus tareas, incluyendo la carga de datos, el manejo de la información y la respuesta a las acciones del usuario.

4. **Disponibilidad y compatibilidad:** Se revisa en qué plataformas (iOS, Android, web) se encuentra disponible la aplicación, así como su capacidad para sincronizarse con otros dispositivos o servicios, como relojes inteligentes o aplicaciones de seguimiento de actividad física.

5. **Soporte y servicio al usuario:** Se considera la calidad del soporte técnico ofrecido por los desarrolladores, evaluando la accesibilidad a recursos de ayuda, la regularidad de las actualizaciones y la atención al cliente frente a problemas o consultas.

El estudio detallado de cada aplicación basado en estos principios se presenta en las secciones siguientes, donde se destacan las ventajas y limitaciones de cada una. Esta comparación servirá como base para identificar posibles mejoras y diferenciar nuestra propia aplicación en el mercado.

	Menú semanal	Generación automática de menú semanal	Consumo de calorías	Recetas	Lista de compra	Ejercicio diario	Comparador de precios	Estadísticas	Consejos saludables	Cursos interactivos	Asistente Virtual
Nutrilio	✗	✗	✓	✗	✗	✓	✗	👑	✗	✗	✗
Fitatu	✓	✗	✓	✓	✗	✓	✗	✓	✗	✗	✗
Fitia	✓	👑	✓	✓	👑	✓	✗	✓	✗	✗	👑
Unimeal	👑	👑	👑	👑	👑	👑	✗	👑	👑	👑	✗

✓ Funcionalidad incluida
✗ Funcionalidad no incluida
👑 Funcionalidad de pago

Figura 2.1: Comparativa de las características de las aplicaciones estudiadas.

En la Figura 2.1 ilustra en qué aspectos cada aplicación sobresale, lo que ayuda a identificar tanto las fortalezas como las debilidades en comparación con nuestra propia aplicación. Este análisis es clave para definir estrategias que nos permitan mejorar y diferenciarnos en el mercado.

En primer lugar, Nutrilio es la única aplicación que no ofrece la función de planificación de menús semanales. Aunque permite llevar un registro diario de la alimentación, carece de la capacidad para organizar las comidas de la semana. En contraste, Fitia y Unimeal sí incluyen esta característica, destacándose por generar menús de manera automática. Ambas aplicaciones utilizan sus bases de datos de recetas para proponer las más adecuadas según los requerimientos calóricos de los usuarios. Cabe mencionar que, en el caso de Fitia, el menú no se genera automáticamente, sino que se ofrecen sugerencias inteligentes de recetas al planificar el menú semanal.

Por otro lado, Unimeal sobresale como la aplicación más completa en cuanto a funcionalidades, aunque estas solo están disponibles a través de una suscripción paga, a pesar de que la descarga de la aplicación es gratuita. Este modelo es interesante, ya que sugiere que el valor ofrecido por la aplicación es suficiente para justificar un costo para el usuario. Por último, en la Figura 2.2, se muestran las valoraciones otorgadas por los usuarios a cada una de estas 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 2.2: Calificación promedio de las aplicaciones en Google Play Store.

Unimeal, pese a su amplio conjunto de funcionalidades, recibe una calificación inferior en comparación con sus competidores, quienes, aunque ofrecen menos características, logran obtener valoraciones más altas. Esto sugiere que Unimeal, que podría ser un rival importante para nuestro producto, no está logrando cumplir con las expectativas de sus usuarios. Las opiniones indican que existe un problema significativo en la función de creación de menús de la aplicación.

En contraste, Fitatu sobresale por su amplia compatibilidad, siendo accesible en Android, iOS, y también a través de una versión web. Por otro lado, Fitia, Nutrilio y Unimeal se limitan exclusivamente a dispositivos móviles, aunque su desempeño en estas plataformas es efectivo.

2.3.3. Conclusiones

El análisis comparativo de las aplicaciones Nutrilio, Fitatu, Fitia y Unimeal revela aspectos clave para mejorar y diferenciar nuestra propia aplicación en el mercado. Nutrilio destaca por su capacidad de registro diario, pero carece de una función de planificación semanal, una característica presente en Fitia y Unimeal. Sin embargo, aunque Unimeal ofrece la mayor cantidad de funcionalidades, su dependencia de una suscripción paga no ha sido bien recibida, como lo demuestran sus bajas calificaciones en comparación con sus competidores.

Por otro lado, Fitatu sobresale por su compatibilidad con múltiples plataformas, lo que le otorga una ventaja competitiva. Este estudio subraya la importancia de no solo ofrecer funciones avanzadas, sino de asegurar que estas funciones sean valoradas por los usuarios, garantizando así una mejor recepción en el mercado.

CAPÍTULO 3
Arquitectura de la aplicación

La arquitectura de la aplicación ha sido diseñada siguiendo un enfoque modular y escalable, basado en una estructura de capas que separa de manera clara las responsabilidades de cada componente. Esta separación facilita tanto el mantenimiento como la escalabilidad del sistema, permitiendo que cada capa pueda evolucionar independientemente de las demás.

3.1 Estructura de Capas

La estructura de capas se divide en tres grandes bloques:

- **Capa de Presentación:** Implementada como la aplicación móvil, esta capa es la encargada de la interacción con el usuario final. Utiliza el lenguaje Java y está desarrollada en Android Studio, lo que permite una integración nativa con dispositivos Android. La comunicación con el *back-end* se realiza a través de llamadas a una API REST, facilitando un intercambio de datos eficiente y seguro.
- **Capa de Lógica de Negocio:** Desarrollada en C# utilizando el framework ASP.NET, esta capa maneja la lógica de negocio y se encarga de procesar las solicitudes provenientes del *front-end*. Está compuesta por dos controladores principales: uno que gestiona las solicitudes de la aplicación móvil y otro que se encarga de activar los *scrapers* para la recopilación de datos. Estos *scrapers* obtienen información de diferentes fuentes como Mercadona y FatSecret.
- **Capa de Persistencia:** Esta capa centraliza la gestión de los datos mediante el uso de un almacén de datos ubicado en AWS RDS, una base de datos en la nube que proporciona escalabilidad y alta disponibilidad. Los datos extraídos por los *scrapers* son almacenados y organizados para ser utilizados por la aplicación según sea necesario.

3.2 Interacción entre Capas

La arquitectura de la aplicación sigue un modelo en capas que facilita la separación de responsabilidades, asegurando un desarrollo modular y una fácil mantenibilidad. La Figura 3.1 ilustra cómo interactúan estas capas entre sí, permitiendo una gestión eficiente y ordenada del flujo de datos.

Cuando un usuario interactúa con el *front-end*, esta capa envía solicitudes al *back-end* a través de la API REST. Los controladores en el *back-end* procesan estas solicitudes y, si es

necesario, activan los *scrapers* para obtener la información requerida. Una vez recopilados y procesados los datos, estos se acoplan en el almacén de datos y se envían de vuelta al *front-end* para su visualización o posterior procesamiento.

Este diseño modular asegura que la aplicación no solo sea eficiente en términos de rendimiento, sino también flexible ante futuros cambios o expansiones. La arquitectura permite la fácil integración de nuevas fuentes de datos o funcionalidades adicionales sin afectar la estabilidad del sistema, adaptándose así a las necesidades cambiantes de los usuarios y del entorno tecnológico.

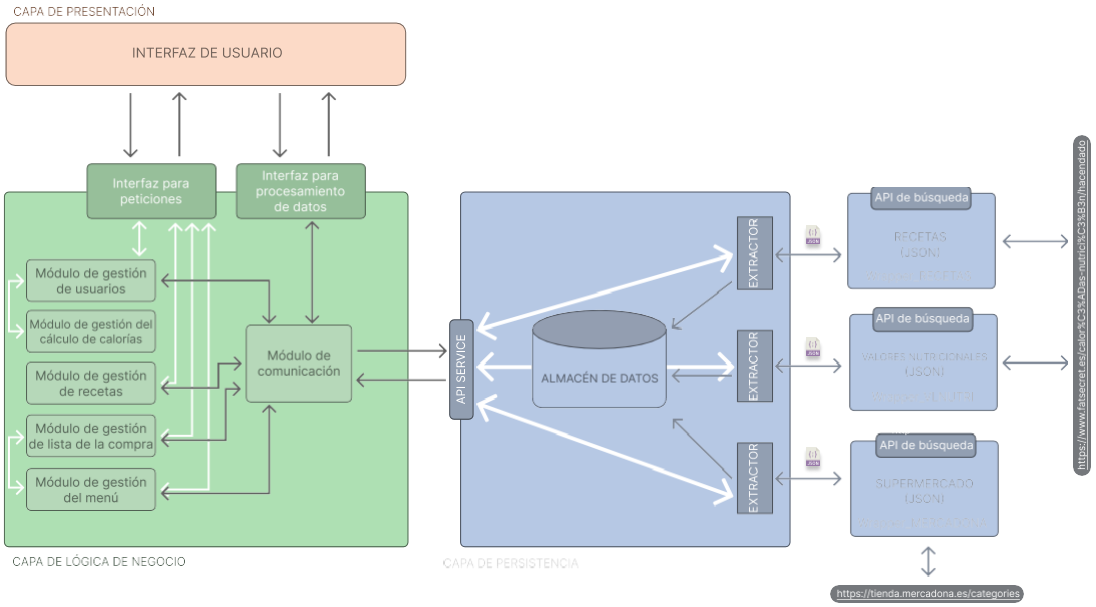

Figura 3.1: Arquitectura de la aplicación

A continuación, se describen los componentes de la arquitectura mostrada en la Figura 3.1:

- **Interfaz de Usuario**: Este componente pertenece a la capa de presentación y es el punto de interacción entre el usuario y la aplicación. A través de esta interfaz, los usuarios pueden acceder a las funcionalidades del sistema, como la búsqueda de productos, gestión de recetas, cálculo de calorías, y otros servicios.

- **Módulo de Gestión de Usuarios**: Ubicado en la capa lógica, este módulo se encarga de manejar todas las operaciones relacionadas con los usuarios, como autenticación, registro, y gestión de perfiles. Su función es asegurar que solo los usuarios autorizados accedan a los recursos del sistema.

- **Módulo de Comunicación**: Este módulo también se encuentra en la capa lógica y es responsable de coordinar la interacción entre los diferentes módulos del sistema y la capa de presentación. Facilita el flujo de datos entre los módulos internos y la interfaz de usuario.

- **Módulo de Gestión del Menú**: Otro componente de la capa lógica, encargado de gestionar y organizar el menú que se presenta a 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 Recetas**: Situado en la capa lógica, este módulo gestiona la base de datos de recetas. Permite a los usuarios buscar, almacenar y acceder a recetas según los ingredientes disponibles o preferencias específicas.
- **Módulo de Gestión de Lista de la Compra**: Este módulo, que forma parte de la capa lógica, permite a los usuarios crear y gestionar sus listas de compras. Se comunica con el módulo de gestión del menú y recetas para actualizar la lista de acuerdo con las selecciones hechas por el usuario.
- **Módulo de Gestión del Cálculo de Calorías**: También dentro de la capa lógica, este módulo calcula el contenido calórico de las recetas seleccionadas o de los alimentos individuales, proporcionando información nutricional valiosa a los usuarios.
- **Interfaz para Peticiones**: Esta interfaz, ubicada en la capa lógica, maneja todas las solicitudes entrantes del usuario, canalizándolas hacia los módulos correspondientes para su procesamiento.
- **Interfaz para Procesamiento de Datos**: Similar a la interfaz de peticiones, este componente se encarga de la manipulación de datos que se derivan de las operaciones realizadas en la capa lógica, asegurando que la información sea procesada correctamente antes de ser almacenada o presentada.
- **Extractor**: Este componente está presente en la capa de datos y actúa como un intermediario que recopila, procesa y distribuye la información entre los diferentes *scrapers*. Se utiliza para manejar la extracción de datos desde fuentes externas.
- **Almacén de Datos**: Localizado en la capa de datos, este componente almacena toda la información relevante del sistema, como recetas, información nutricional, listas de compras y más. Es una base de datos centralizada que sirve como el repositorio de la aplicación.
- **APIs de Búsqueda**: Estas APIs están situadas en la capa de datos y se utilizan para interactuar con fuentes de datos externas. Por ejemplo, realizan solicitudes a las APIs externas de supermercados, valores nutricionales y recetas, recuperando la información necesaria en formato JSON.
- **Wrappers (Wrapper_MERCADONA, Wrapper_RECETAS, Wrapper_VLNUTRI)**: Estos componentes se encargan de adaptar los datos recibidos de las APIs externas a un formato que puede ser procesado internamente por el sistema. Están ubicados en la capa lógica y actúan como intermediarios entre las APIs de búsqueda y el resto de los módulos.
- **Enlaces HTTP**: Representan las conexiones a fuentes de datos externas, como las URLs de supermercados o bases de datos nutricionales. Estos enlaces son esenciales para obtener la información en tiempo real que se procesa a través de las APIs de búsqueda.

Cada uno de estos componentes desempeña un rol específico dentro de la arquitectura global, asegurando que los usuarios puedan interactuar eficientemente con el sistema mientras se gestionan y procesan los datos de manera efectiva.

3.3 Componentes de la Capa de Presentación

La Capa de Presentación, implementada como una aplicación móvil, incluye los siguientes componentes:

- **Interfaz de Usuario (UI):** Diseñada para ser intuitiva y fácil de usar, la interfaz se adapta a diferentes tamaños de pantalla y resoluciones en dispositivos Android.
- **Controladores de Interfaz:** Gestionan la interacción entre la UI y el modelo de datos, enviando y recibiendo información a través de la API REST.
- **Manejadores de Eventos:** Estos componentes responden a las interacciones del usuario, como clics, deslizamientos y entradas de datos, enviando las solicitudes correspondientes a la capa lógica de negocio.

3.4 Componentes de la Capa Lógica de Negocio

Dentro de la Capa Lógica de Negocio, los componentes principales son:

- **Controladores de Aplicación:** Estos componentes manejan las solicitudes entrantes desde la capa de presentación y coordinan la lógica de negocio necesaria para procesarlas.
- **Servicios de Negocio:** Implementan las reglas de negocio y operaciones que transforman los datos según los requisitos del sistema.
- **Scrapers:** Módulos que se encargan de extraer datos de fuentes externas como Mercadona y FatSecret. Estos datos se procesan y almacenan para ser utilizados por la aplicación.

3.5 Consideraciones de Escalabilidad y Mantenimiento

La arquitectura modular de la aplicación permite que cada componente se escale de forma independiente. A medida que la base de usuarios crece o se agregan nuevas funcionalidades, cada capa puede adaptarse sin afectar el funcionamiento general del sistema. Por ejemplo:

- **Escalabilidad Horizontal y Vertical:** La base de datos en AWS RDS permite tanto la escalabilidad vertical (aumentando la capacidad del servidor) como horizontal (añadiendo más instancias de base de datos) según sea necesario.
- **Despliegue y Mantenimiento:** Gracias a la clara separación de responsabilidades y a la implementación de patrones de diseño, la arquitectura soporta un ciclo de vida de desarrollo ágil, donde las nuevas características pueden añadirse o modificarse sin causar regresiones en otras partes del sistema.

CAPÍTULO 4
Metodología

Seleccionar la metodología más adecuada es un paso fundamental en cualquier proyecto de desarrollo de software, ya que define el camino desde la etapa inicial hasta su finalización. Las metodologías de desarrollo proporcionan una estructura organizativa que permite gestionar, planificar y controlar el proceso, garantizando la entrega de un producto de alta calidad dentro del tiempo y presupuesto acordados.

Existen diversas metodologías, cada una adaptada a las particularidades del proyecto, las expectativas del cliente y la dinámica del equipo de desarrollo. Estas metodologías varían en sus consideraciones, en cómo estructuran las fases del proyecto y en los materiales que utilizan, permitiendo flexibilidad y adaptación a diferentes escenarios.

Este capítulo explora diferentes enfoques metodológicos, desde la metodología tradicional, que sigue un proceso lineal y estructurado, hasta las metodologías ágiles, que ofrecen flexibilidad y capacidad de adaptación en entornos cambiantes. También se discute un enfoque adaptado utilizado específicamente para este proyecto, combinando elementos de ambos paradigmas para satisfacer las particularidades del desarrollo en cuestión. Esta combinación de metodologías permitió gestionar eficientemente el proyecto, equilibrando la planificación detallada con la flexibilidad necesaria para ajustarse a las circunstancias del equipo y del proyecto.

4.1 Metodologías tradicionales

Las metodologías tradicionales [10] proporciona un enfoque lineal y estructurado para el desarrollo de software. Estas metodologías surgieron en respuesta a la creciente complejidad de los proyectos de software y la necesidad de manejar de manera efectiva proyectos de mayor envergadura. A medida que el desarrollo de software se fue profesionalizando y los sistemas se volvieron más complejos, se hizo evidente la necesidad de un enfoque sistemático que permitiera planificar y gestionar cada fase del ciclo de vida del proyecto de manera ordenada.

Como se muestra en la Figura 4.1, la metodología tradicional se basa en una serie de fases bien definidas que se ejecutan en un orden cronológico: análisis, diseño, desarrollo, pruebas, implementación y mantenimiento.

Este enfoque es ideal para proyectos de software con requisitos bien definidos y estables, donde se esperan la mínima cantidad de cambios a lo largo del curso del proyecto. No obstante, en contextos más dinámicos o inciertos, resulta imperativo considerar metodologías más adaptables.

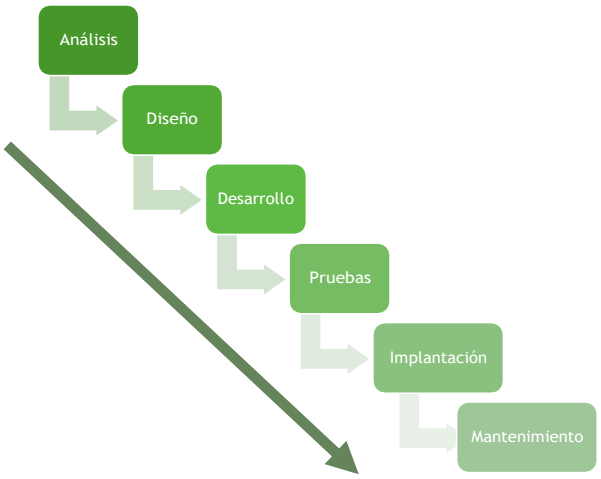

Figura 4.1: Metodología tradicional.

4.2 Metodología ágil

Las metodologías ágiles han emergido como una respuesta a los desafíos que presentan las metodologías tradicionales, especialmente en entornos donde los requisitos y las condiciones del mercado están en constante cambio. A medida que los proyectos de software se volvieron más complejos y la necesidad de adaptabilidad se hizo más evidente, estas metodologías han ganado prominencia desde finales del siglo XX hasta la actualidad. Su principal ventaja radica en la capacidad de adaptarse rápidamente a las necesidades cambiantes del cliente y del mercado, lo que las hace ideales para proyectos donde la flexibilidad es clave.

El enfoque ágil se caracteriza por el uso de ciclos de desarrollo cortos e iterativos, conocidos como *sprints*, que permiten la entrega continua de valor al cliente. Esta estructura iterativa no solo facilita la incorporación de mejoras y cambios, sino que también asegura que el producto final esté más alineado con las expectativas del usuario. Al mismo tiempo, la colaboración estrecha entre el equipo de desarrollo y el cliente se convierte en un pilar fundamental del proceso, asegurando una comunicación constante y un flujo de retroalimentación continuo.

En entornos dinámicos, donde los requisitos pueden cambiar rápidamente, la metodología ágil ha demostrado ser especialmente efectiva, proporcionando a las organizaciones una ventaja competitiva significativa. Su capacidad para responder a los cambios, mantener una alta calidad del producto y fomentar una mayor satisfacción del cliente ha consolidado su popularidad en el desarrollo de software moderno.

4.3 Enfoque Adoptado

El desarrollo de este TFG se llevó a cabo mediante una metodología única que fusiona componentes de enfoques tradicionales con prácticas ágiles, en particular SCRUM [23], adaptadas a las particularidades del proyecto. Dado el reducido tamaño del equipo, compuesto por dos personas, esta metodología híbrida fue seleccionada para equilibrar organización y flexibilidad en el proceso de desarrollo.

El proceso comenzó con una identificación y definición detallada de los requisitos del proyecto, en línea con enfoques tradicionales, lo que permitió establecer una base sólida y clara de los objetivos a alcanzar. Este paso inicial fue esencial para asegurar que todos los aspectos del proyecto fueran considerados desde el principio, minimizando así posibles errores o malentendidos en etapas posteriores.

En lugar de seguir un cronograma rígido, optamos por una planificación adaptable, desglosando el proyecto en tareas más manejables. Estas tareas se organizaron en un calendario flexible, revisado y ajustado semanalmente según el progreso y los retos encontrados. Esta flexibilidad fue clave para responder de manera ágil a cualquier cambio o imprevisto que surgiera durante el desarrollo.

El proyecto se ejecutó a través de tres ciclos de trabajo, o *sprints*, cada uno con una duración de aproximadamente tres semanas. Cada *sprint* tenía objetivos específicos claramente definidos, y la estructura de trabajo dentro de cada ciclo permitió una revisión continua del progreso y ajustes en tiempo real. Además, se incluyó un enfoque en la retroalimentación constante, no solo entre los miembros del equipo, sino también a través de revisiones periódicas con partes interesadas, asegurando que el proyecto se mantuviera alineado con sus objetivos generales.

- ***Sprint* 1:** Nos enfocamos en la investigación detallada de los requisitos. Se elaboraron los primeros esquemas y diagramas necesarios para orientar el desarrollo, y se estableció la estructura general del proyecto.
- ***Sprint* 2:** Se implementaron las funcionalidades principales del proyecto, basándonos en los requisitos previamente definidos. Se trabajó en el desarrollo del núcleo del sistema, asegurando que las funcionalidades esenciales estuvieran operativas y alineadas con las expectativas del proyecto.
- ***Sprint* 3:** Este *sprint* se dedicó principalmente a las pruebas y a la optimización del sistema. Durante esta fase, se realizaron pruebas exhaustivas para identificar y corregir errores.

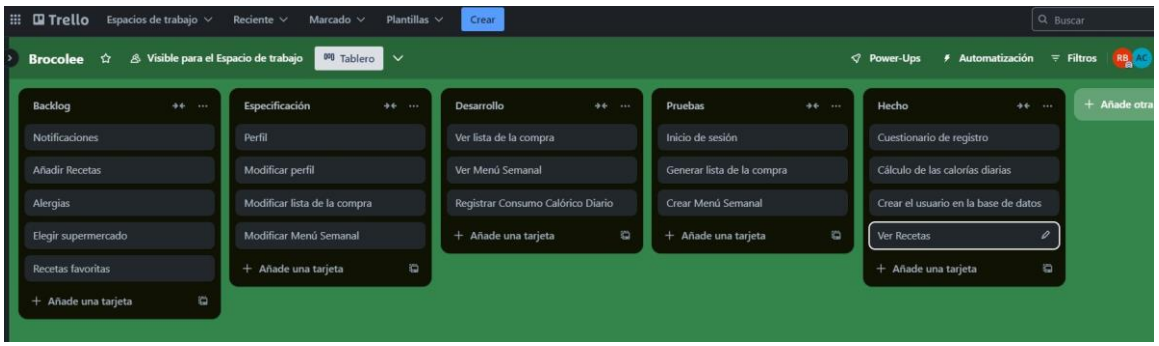

Figura 4.2: Tablero Kanban utilizado en el proyecto.

Para optimizar la organización del proyecto, se implementó el uso de Trello [4] como herramienta para gestionar un tablero Kanban [13], el cual se presenta en la Figura 7.1. En este tablero se definieron cinco fases principales que reflejan el avance de las tareas:

1. **Backlog:** Esta columna contiene todas las tareas y funcionalidades pendientes que aún no han sido priorizadas o asignadas para su ejecución. Funciona como una lista centralizada donde se recopilan todas las actividades por realizar, asegurando que el equipo tenga visibilidad de las tareas futuras y ningún elemento importante se pierda de vista.

2. **Especificación:** Las tareas seleccionadas del backlog se mueven a esta fase, donde se detallan los requisitos, criterios de aceptación y cualquier información adicional necesaria. Aquí se asegura que cada tarea esté bien definida y comprendida antes de iniciar su desarrollo, minimizando así posibles errores o malentendidos.

3. **Desarrollo:** En esta etapa, las tareas especificadas se implementan. Una vez que se trasladan a esta columna, un miembro del equipo las toma y comienza a trabajar en ellas. Es la fase donde se realiza la codificación, configuración o ejecución técnica de las tareas según lo definido previamente.

4. **Pruebas:** Tras completar el desarrollo, las tareas pasan a la fase de pruebas. Aquí se verifican para asegurar que cumplen con los criterios de aceptación y funcionan correctamente sin errores. Esta etapa es crucial para garantizar la calidad del trabajo antes de considerarlo finalizado.

5. **Hecho:** Las tareas que han pasado exitosamente las pruebas se mueven a esta columna. Esto indica que han sido completadas de acuerdo a lo especificado y no requieren más acciones. Representa el cierre de la tarea y el progreso efectivo del proyecto.

Para asegurar un flujo de trabajo constante y coordinado, se realizaron reuniones semanales. Estas sesiones nos permitieron monitorear el progreso, identificar cualquier problema, ajustar las tareas según fuera necesario y coordinar las actividades del equipo, especialmente considerando la interdependencia de las tareas entre los miembros del equipo. Durante estas reuniones, revisamos el tablero Kanban, actualizamos el estado de las tareas y discutimos cualquier obstáculo que pudiera estar ralentizando el avance. Este enfoque promovió una comunicación continua y efectiva, garantizando que ambos miembros del equipo estuvieran alineados y trabajando en las prioridades correctas.

Dado que el proyecto era de pequeña escala y el equipo estaba compuesto por solo dos personas, este enfoque híbrido resultó ser el más adecuado. La planificación inicial y la especificación detallada de los requisitos proporcionaron la claridad y dirección necesarias, mientras que la asignación semanal de tareas y las reuniones regulares nos permitieron mantener la flexibilidad y adaptarnos a cualquier cambio o desafío que surgiera durante el desarrollo.

CAPÍTULO 5
Análisis y especificación de requisitos

5.1 Especificación de requisitos

5.1.1. Público objetivo y sus características

El público objetivo de la aplicación Brocolee está compuesto por personas de ambos sexos, con edades entre 18 y 55 años que tengan acceso a Internet. Este rango de edad se ha seleccionado debido a varias razones clave. Las personas a partir de los 18 años suelen ser más independientes en sus decisiones alimenticias y comienzan a asumir la responsabilidad de gestionar su propia dieta, especialmente aquellos que viven solos o en pareja. En esta etapa, muchos están interesados en mantener un estilo de vida saludable, ya sea para mejorar su bienestar general, prevenir enfermedades o complementar su rutina de ejercicio.

Hasta los 55 años, los individuos se encuentran generalmente en una fase de la vida en la que la salud empieza a ser una prioridad aún mayor. En este periodo, son más conscientes de la necesidad de mantener una buena nutrición para prolongar la vitalidad y prevenir problemas de salud asociados con el envejecimiento. Además, este grupo tiene una mayor capacidad adquisitiva y está más habituado al uso de tecnologías móviles para gestionar diversos aspectos de su vida cotidiana.

Estos usuarios generalmente pertenecen a un nivel socioeconómico medio a alto, lo cual les permite tener acceso a *smartphones* y a aplicaciones de pago.

Son individuos que están interesados en mantener una dieta saludable, equilibrada y que utilizan activamente aplicaciones móviles para gestionar diferentes aspectos de su vida diaria, incluyendo la alimentación. Además, suelen practicar deportes o realizar ejercicios de manera regular y buscan complementar su rutina física con una buena nutrición.

Estos usuarios requieren herramientas que les faciliten la planificación y gestión de su dieta diaria y semanal. Están interesados en obtener información detallada sobre el valor nutricional de los alimentos, ajustar su dieta conforme a su presupuesto, y simplificar tanto la planificación de comidas como la compra de ingredientes necesarios para seguir un régimen alimenticio equilibrado.

5.1.2. Requisitos funcionales

Los requisitos funcionales describen las acciones y capacidades específicas que debe realizar una aplicación o sistema para cumplir con sus objetivos. En el contexto de una aplicación móvil de nutrición, estos requisitos detallan las funcionalidades que permiten a los usuarios interactuar con la aplicación, como registrarse, iniciar sesión, modificar su perfil, generar y modificar menús semanales, y gestionar listas de compra. Estos requisitos son esenciales para definir el comportamiento esperado del sistema desde la perspectiva del usuario, asegurando que todas las funcionalidades necesarias estén cubiertas y operen correctamente.

A continuación, se presenta una serie de tablas que detallan los requisitos funcionales específicos que la aplicación debe cumplir para garantizar su correcto funcionamiento, iniciaremos con los que su implementación esta relacionada directamente con este TFG:

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 5.1: 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 5.2: 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 5.3: 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 5.4: 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 5.5: Requisito funcional: Ver recetas.

Posteriormente, podemos observar los requisitos funcionales que están asociados a la TFG complementario a este llamado Gestión personal de dietas saludables (II): Interfaz de usuario y gestión de usuarios y listas de compra[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 5.6: 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 sitema, 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 5.7: 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 5.8: 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 5.9: 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 5.10: 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 5.11: Requisito funcional: Modificar perfil

5.1.3. Requisitos no funcionales

En este apartado se describen los requisitos no funcionales que deben cumplir el sistema y la aplicación desarrollada. Estos requisitos son esenciales para garantizar la calidad, el rendimiento y la sostenibilidad del software, asegurando que el sistema no solo cumpla con las especificaciones funcionales, sino que también ofrezca una experiencia de usuario robusta y confiable. A continuación, se detallan los requisitos no funcionales identificados, clasificados por su prioridad y relevancia en el desarrollo del proyecto:

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 pensado para ser una aplicación móvil, pero presenta algunas limitaciones. Primero, solo funcionará en dispositivos Android, excluyendo otros sistemas operativos. Segundo, esta siendo desarrollanda utilizando las versiones más recientes de Android Studio, lo cual puede generar incompatibilidades con dispositivos que operan con versiones muy antiguas del sistema operativo Android.

Limitación de Acceso y Despliegue de la Base de Datos

El acceso y despliegue de la base de datos en este proyecto presentan ciertas limitaciones que deben ser consideradas. En primer lugar, la base de datos está diseñada para ser alojada en un servidor local durante el desarrollo y pruebas, lo que implica que su acceso está restringido a entornos controlados. Esto puede generar desafíos cuando se intente escalar la aplicación a un entorno de producción o se busque un acceso remoto más amplio.

Además, la falta de un servidor en la nube para el despliegue de la base de datos limita su disponibilidad y accesibilidad desde diferentes ubicaciones geográficas. La migración a un servidor en la nube sería un paso necesario para garantizar la escalabilidad y la fiabilidad del sistema en un entorno de producción real, permitiendo a los usuarios acceder a la base de datos desde cualquier lugar con conexión a Internet.

Por último, la seguridad de los datos es un aspecto crítico que debe ser abordado. Actualmente, el acceso a la base de datos está limitado a conexiones directas, lo que podría representar un riesgo de seguridad si no se implementan medidas adicionales, como el cifrado de datos en tránsito y en reposo, así como políticas estrictas de control de acceso.

5.2 Modelo de casos de usos

5.2.1. Diagrama de contexto

El diagrama de contexto muestra las interacciones principales entre los diferentes actores y el sistema. Este diagrama es esencial para entender cómo los usuarios y componentes externos se comunican con la aplicación. En la Figura 5.1, se destacan los siguientes actores: el Administrador, el Usuario No Registrado, el Usuario Registrado, el Reloj y la API de ChatGPT. Cada uno de estos actores tiene roles y responsabilidades específicos dentro del sistema, facilitando una visión clara de sus interacciones y la funcionalidad general de la aplicación.

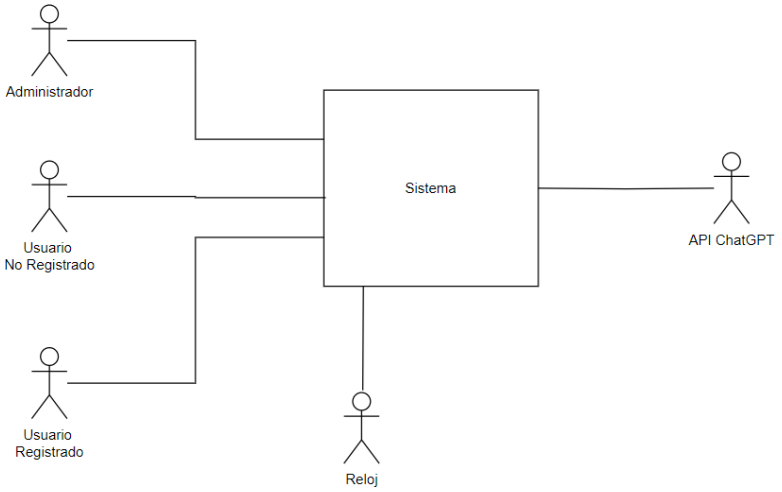

Figura 5.1: Diagrama de contexto

5.2.2. Diagrama de casos de uso

En este apartado se presenta el diagrama de casos de uso que ilustra las interacciones entre los diferentes actores y el sistema. Este diagrama es fundamental para entender cómo se relacionan los usuarios, tanto registrados como no registrados, así como los administradores y otros componentes externos, con las funcionalidades de la aplicación. Cada caso de uso describe una funcionalidad específica, destacando los pasos necesarios para su ejecución, las alternativas posibles y los errores que pueden ocurrir durante el proceso. La finalidad es proporcionar una visión clara y detallada del comportamiento del sistema en distintos escenarios.

La Figura 5.2 muestra un diagrama de caso de uso que representa el proceso de "Registrar usuario" en la aplicación. En este diagrama, el "Usuario No Registrado" es el actor que interactúa con el sistema para crear una nueva cuenta. La acción principal que realiza este actor es "Registrar usuario", la cual se explica con mayor detalle en la tabla A.1. Este caso de uso ilustra cómo un nuevo usuario ingresa sus datos personales y el sistema los guarda en la base de datos, permitiéndole así acceder a las funcionalidades completas de la aplicación.

Figura 5.2: Caso de uso

En la Figura 5.3 se muestra un diagrama de casos de uso centrado en las funcionalidades disponibles para un usuario registrado en la aplicación, para información mas detalla de estos casos de uso puede acceder a las tablas A.23, A.10, A.6, A.25, A.20, A.24,

A.8, A.12. Este usuario puede realizar diversas acciones que mejoran su experiencia y personalización de la aplicación. Entre las principales funcionalidades se encuentran la configuración de notificaciones, la adición de nuevas recetas, el acceso a la galería de recetas, y la creación manual de un menú semanal. Además, el usuario puede visualizar su menú semanal, registrar su consumo calórico y nutricional diario, y modificar tanto el menú como la lista de la compra según sus necesidades. La interacción con estas funciones se realiza a través de una interfaz amigable que facilita el seguimiento y personalización de sus hábitos alimenticios.

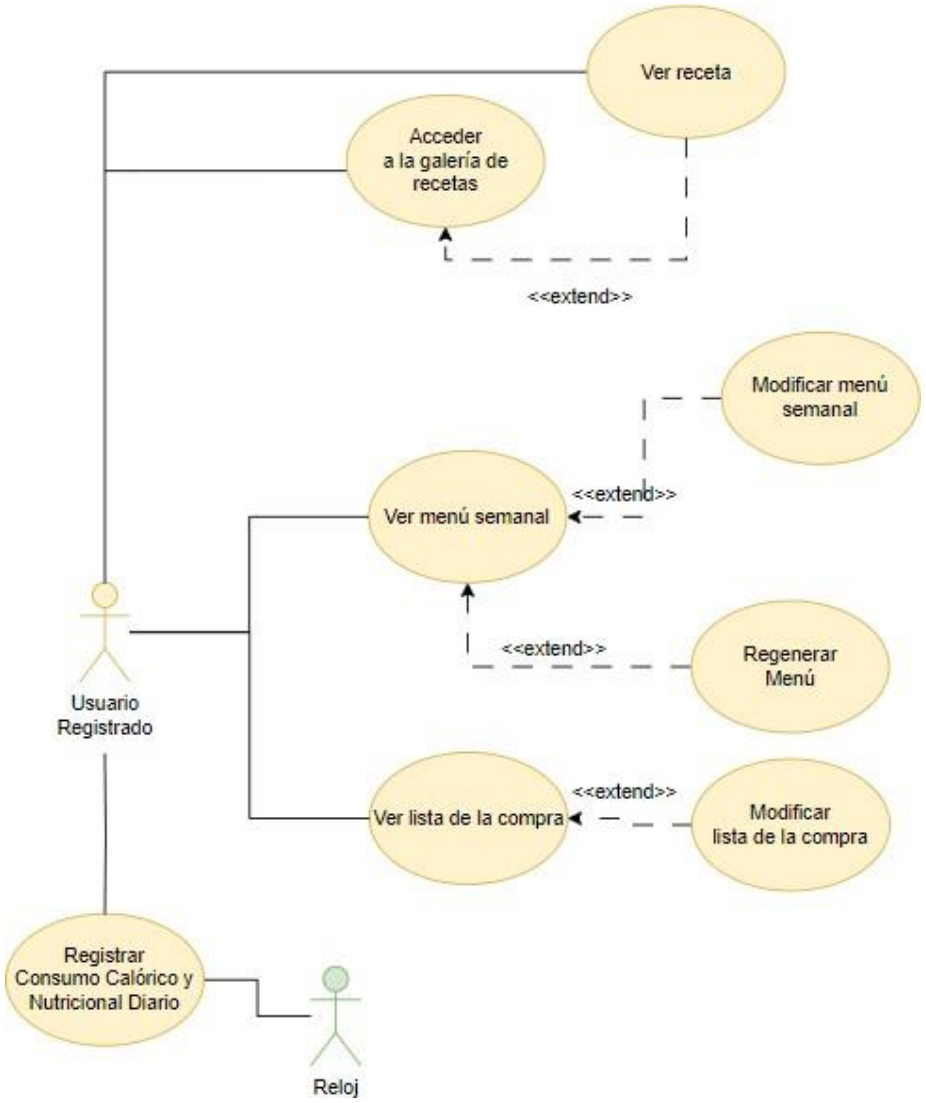

Figura 5.3: Caso de uso

La Figura 5.4 muestra los casos de uso automáticos del sistema, que se inician a través de un reloj programado. Estos casos incluyen la generación automática del menú semanal y el envío de notificaciones, podrás ver mas detalles de estos en las tablas A.5 y A.14. Estos procesos aseguran que el usuario reciba un menú actualizado y recordatorios oportunos, mejorando así la experiencia de uso de la aplicación de manera eficiente y continua.

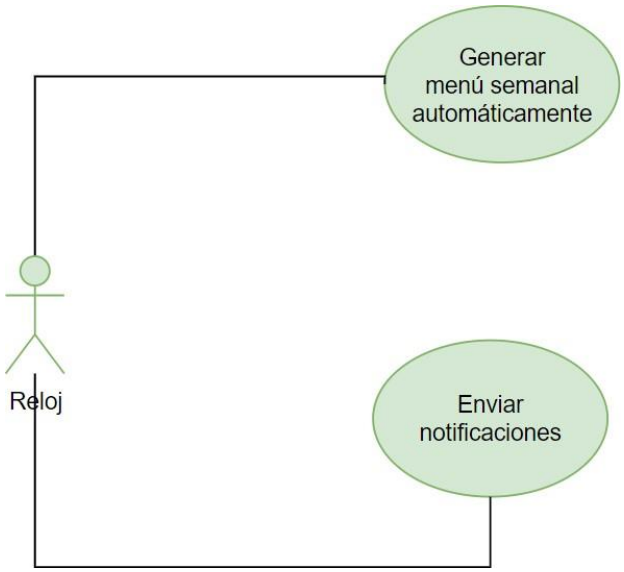

Figura 5.4: Caso de uso

La Figura 5.5 muestra los casos de uso relacionados con la gestión de datos por el administrador. Estos casos incluyen la carga de datos de supermercados A.16, la carga de datos de valores nutricionales A.17, la carga de datos de recetas A.18, la actualización de datos de los supermercados A.19 y la gestión de usuarios A.13. Estas funcionalidades permiten al administrador mantener la base de datos actualizada y precisa, asegurando que los usuarios tengan acceso a información correcta y actual para una mejor experiencia de uso de la aplicación.

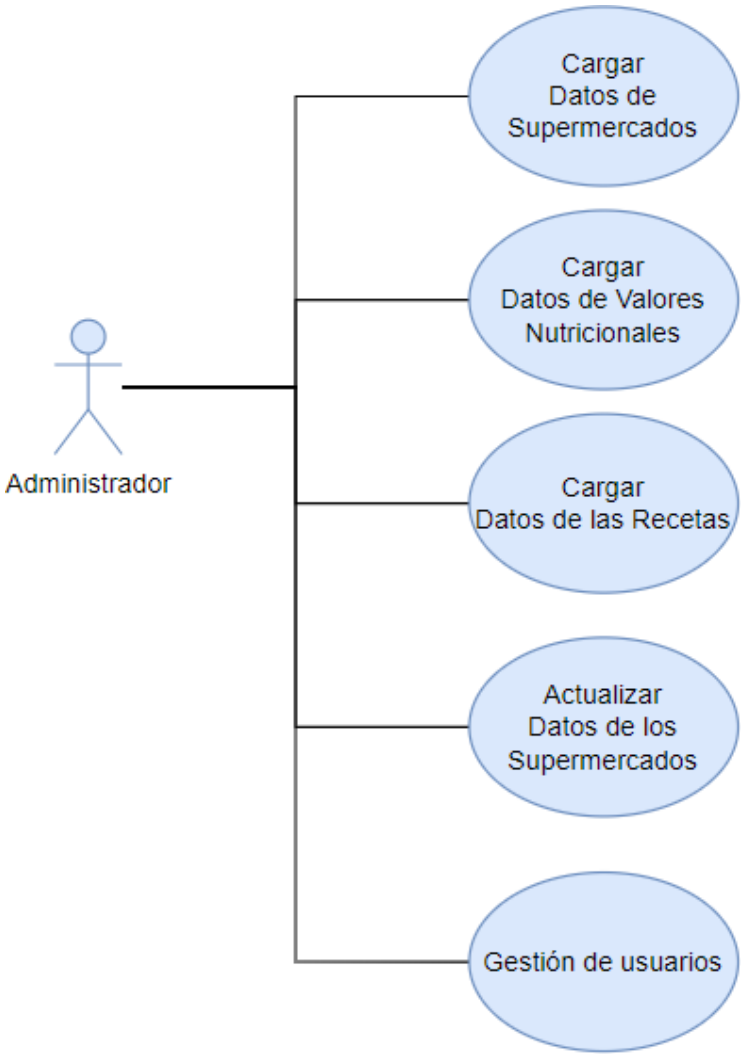

Figura 5.5: Caso de uso

5.2.3. Diagrama de clases

El modelo de dominio es una representación abstracta y conceptual de los datos y las relaciones que existen en el sistema. Este modelo se utiliza para definir cómo los diferentes elementos del sistema interactúan entre sí, proporcionando una visión clara de la estructura y el funcionamiento del sistema. En este proyecto, el modelo de dominio se centra en los componentes clave de la aplicación de nutrición, como usuarios, recetas, menús, productos y listas de compras.

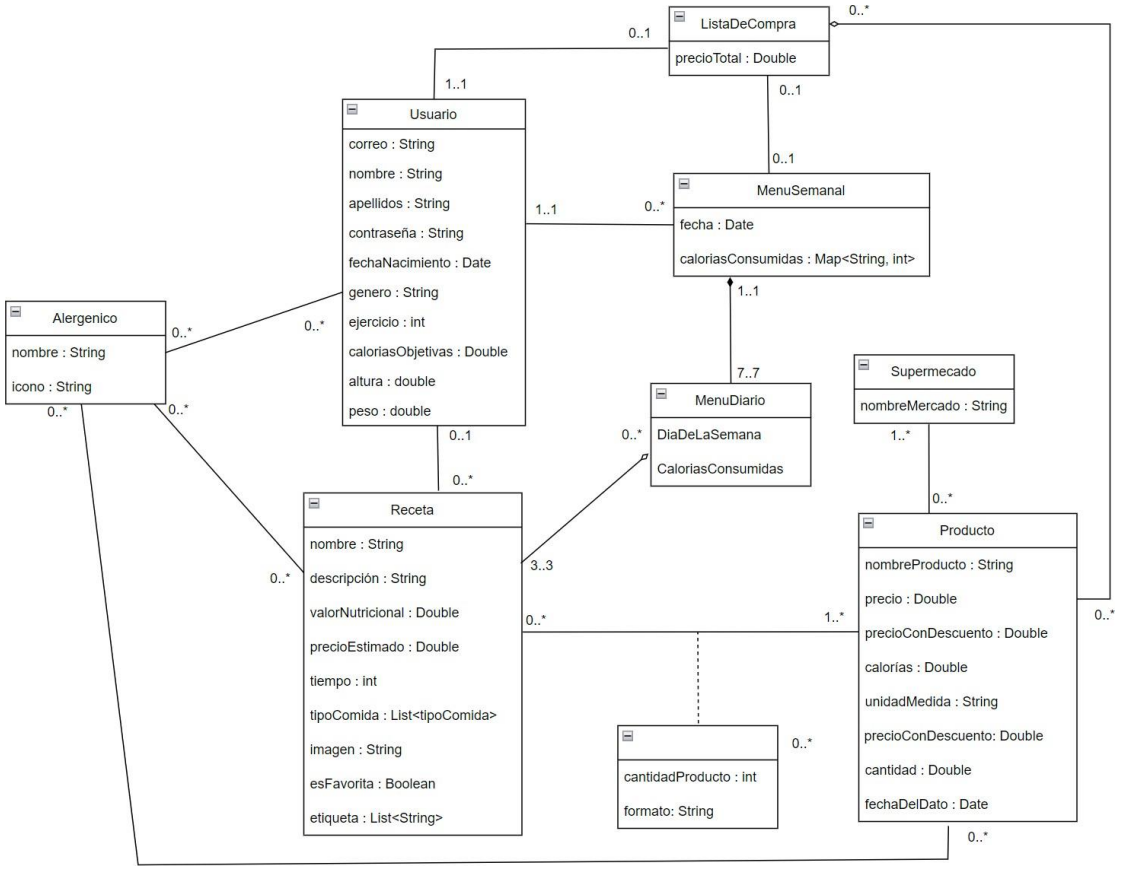

Figura 5.6: Modelo de Dominio del Sistema

La Figura 5.6 muestra el modelo de dominio para nuestra aplicación de nutrición. A continuación se describen las principales entidades y sus relaciones:

- **Usuario:** Representa a los usuarios de la aplicación. Incluye atributos como correo, nombre, apellidos, contraseña, fechaNacimiento, género, ejercicio, caloriasObjetivas, altura y peso. Un usuario puede tener un MenuSemanal y una ListaDeCompra.
- **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 una semana específica, con atributos como fecha y caloriasConsumidas. Cada MenuSemanal se compone de siete MenuDiario.
- **MenuDiario:** Define el menú para un día específico, con atributos como DiaDeLaSemana y CaloriasConsumidas. Cada MenuDiario incluye tres Recetas.

- **ListaDeCompra:** Contiene la lista de productos necesarios para el menú semanal, con un atributo precioTotal. La lista se genera a partir de los Productos de las Recetas del MenuSemanal.
- **Producto:** Representa los productos disponibles en los supermercados, con atributos como nombreProducto, precio, precioConDescuento, calorias, unidadMedida, cantidad y fechaDelDato. Cada producto está asociado a un Supermercado.
- **Supermercado:** Define los supermercados de los cuales se obtienen los productos, con el atributo nombreMercado.
- **Alergénico:** Representa los alérgenos presentes en las recetas, con atributos como nombre e icono. Cada alérgeno puede estar asociado a múltiples recetas.

Este modelo de dominio establece una base sólida para la implementación de la aplicación, asegurando que todas las funcionalidades esenciales estén respaldadas por una estructura de datos coherente y bien definida.

5.3 Análisis de requisitos

5.3.1. Priorización de requisitos

La priorización de requisitos es esencial para el desarrollo eficiente de la aplicación, permitiendo identificar que funcionalidades deben implementarse primero para asegurar un funcionamiento básico y robusto. En este contexto, los requisitos se clasifican generalmente en alta, media y baja prioridad.

- **Alta Prioridad**: Estos requisitos son fundamentales para el funcionamiento básico de la aplicación. Incluyen funcionalidades esenciales como el registro de usuarios, inicio de sesión, cálculo de calorías necesarias, creación de menús semanales y visualización de estos menús. Implementar estos requisitos garantiza que la aplicación sea operativa y útil desde el inicio.
- **Media Prioridad**: Incluyen funcionalidades que mejoran la experiencia del usuario y añaden valor, pero que no son críticas para el funcionamiento inicial de la aplicación. Ejemplos de estos requisitos son 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**: Funcionalidades adicionales que, aunque útiles, no son esenciales para el lanzamiento inicial de la aplicación. Estas pueden incluir características avanzadas o de conveniencia que mejoran aún más la experiencia del usuario una vez que las funcionalidades principales están establecidas y funcionando correctamente.

La clasificación de los requisitos permite una gestión eficaz del proyecto, asegurando que los recursos se asignen de manera eficiente y que la aplicación entregue valor al usuario lo antes posible.

5.3.2. 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 5.6)
  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 5.7)
  Depende del requisito RF1, ya que los usuarios necesitan estar registrados antes de poder iniciar sesión.
- **RF3: Cálculo de Calorías Necesarias Diarias** (Tabla 5.8)
  Requiere los datos ingresados durante el registro de usuario (RF1) para realizar cálculos precisos sobre las necesidades calóricas.
- **RF4: Crear Menú Semanal** (Tabla 5.9)
  Necesita el cálculo de calorías diarias (RF3) para generar menús que se adapten a las necesidades del usuario.
- **RF5: Ver Menú Semanal** (Tabla 5.10)
  Depende de la generación del menú semanal (RF4), permitiendo a los usuarios visualizar los menús creados.
- **RF6: Modificar Menú Semanal** (Tabla 5.11)
  Requiere la funcionalidad de ver (RF5) y crear menús (RF4) para permitir la modificación de los mismos.
- **RF7: Registrar Consumo Calórico Diario** (Tabla 5.1)
  Depende de la existencia de un menú semanal (RF4) para que los usuarios puedan registrar su consumo calórico basado en los menús.
- **RF8: Ver Recetas** (Tabla 5.2)
  Necesita que las recetas estén almacenadas y disponibles para que los usuarios puedan acceder a ellas.
- **RF9: Ver Lista de la Compra** (Tabla 5.3)
  Depende de la creación del menú semanal (RF4), ya que la lista de la compra se genera a partir de los ingredientes del menú.
- **RF10: Ver Perfil** (Tabla 5.4)
  Requiere que los usuarios estén registrados (RF1) e iniciados sesión (RF2) para poder acceder a sus perfiles.
- **RF11: Modificar Perfil** (Tabla 5.5)
  Depende de la capacidad de ver el perfil (RF10) y permite a los usuarios actualizar su información personal y preferencias.

CAPÍTULO 6
Diseño de la Aplicación

Este capítulo presenta el diseño de la arquitectura de la aplicación, enfocándose en la estructura de la base de datos y la implementación de patrones de diseño clave.

Primero, se describe la creación de las tablas de la base de datos, diseñadas meticulosamente para asegurar la integridad y las relaciones adecuadas entre las entidades. Se incluye un diagrama que ilustra cómo estas tablas interactúan, facilitando la comprensión de la estructura general del sistema.

Luego, se analizan los patrones de diseño aplicados, como *Singleton*, *Repositorio*, *MVC*, *Factory* y *Observer*. Estos patrones han sido esenciales para crear un sistema modular, reutilizable y fácil de mantener.

En conjunto, este capítulo proporciona una visión clara de los fundamentos arquitectónicos que soportan el desarrollo de la aplicación, asegurando su robustez y escalabilidad.

6.1 Base de datos

La base de datos es el corazón del sistema, donde se almacena toda la información esencial para el funcionamiento de la aplicación. Un diseño eficiente y bien estructurado de la base de datos es crucial para garantizar que los datos se gestionen de manera segura, se accedan rápidamente y se mantenga la integridad referencial entre las distintas entidades.

El proceso de diseño de la base de datos comenzó con la elaboración de un modelo de datos detallado, que refleja las relaciones entre los diferentes elementos del sistema. Este modelo sirvió como guía para la creación de las tablas, asegurando que cada entidad estuviera correctamente representada y que las relaciones entre ellas fueran claras y consistentes.

En las siguientes secciones, se describen en detalle los pasos tomados para la creación de las tablas y se presenta una representación gráfica del diseño de la base de datos. Este diseño proporciona una base sólida para todas las operaciones de almacenamiento y recuperación de datos, y juega un papel fundamental en la eficiencia y escalabilidad de la aplicación.

6.1.1. Creación de Tablas:

Las tablas fueron diseñadas en base al modelo de datos previamente elaborado, asegurando que cada una contara con las claves primarias y foráneas adecuadas para man-

tener la integridad referencial y las relaciones entre los distintos elementos de la base de datos.

Partiendo del modelo de dominio representado en la Figura 5.6, se implementaron las tablas utilizando la interfaz gráfica que proporciona MySQL Workbench. Este proceso permitió considerar todas las relaciones entre las clases, resultando en la creación de las siguientes tablas:

- **Alergenico:** Tabla que almacena la información de los alérgenos disponibles en la aplicación, tales como nombre e icono.
- **imagen_producto:** Tabla dedicada al almacenamiento de las imágenes asociadas a los productos, permitiendo una mejor identificación visual en la aplicación.
- **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, 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.

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

La Figura 6.1 muestra la representación gráfica del diseño de la base de datos, donde se ilustran las tablas mencionadas anteriormente y sus respectivas relaciones. Este diagrama es fundamental para entender cómo interactúan los diferentes elementos del sistema y cómo se asegura la integridad referencial entre las tablas.

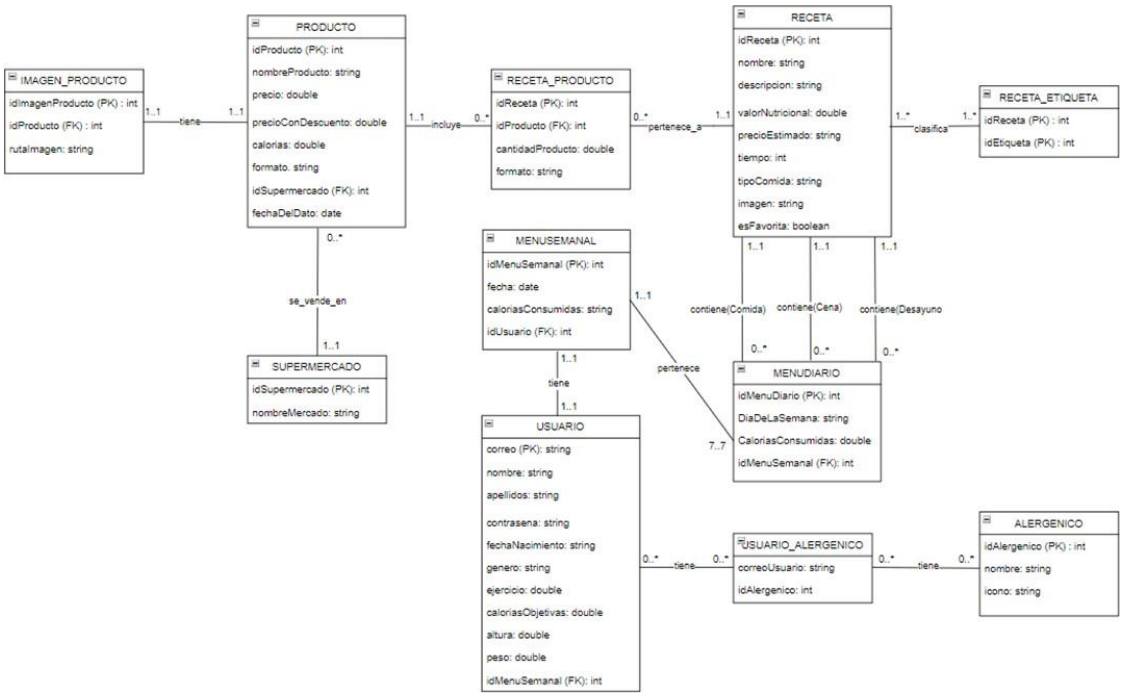

Figura 6.1: Diagrama de la base de datos

6.2 Patrones de Diseño

Los patrones de diseño son soluciones generales y reutilizables para problemas comunes que surgen en el desarrollo de software. Estas soluciones se han refinado a lo largo del tiempo y son ampliamente aceptadas en la comunidad de desarrollo de software [11]. Los patrones de diseño permiten crear un código más estructurado, modular y mantenible, facilitando la colaboración entre desarrolladores y mejorando la escalabilidad del sistema.

En el desarrollo de este proyecto, hemos utilizado varios patrones de diseño clave, entre ellos el patrón *Singleton*, el patrón *Repositorio*, el patrón *MVC*, el patrón *Factory* y el patrón *Observer*. A continuación, se explica cómo se han aplicado estos patrones en el contexto del proyecto, haciendo referencia a los módulos específicos de la arquitectura mostrada en la Figura XX.

6.2.1. Patrón Singleton

El patrón *Singleton* asegura que una clase tenga solo una instancia y proporciona un punto de acceso global a esa instancia. En nuestro proyecto, este patrón se implementa en el Modulo de comunicación, gestionando la conexión a la base de datos (Figura 3.1). Esto es esencial para evitar la creación de múltiples conexiones, lo cual podría generar problemas de concurrencia y aumentar el consumo de recursos. Al utilizar el patrón *Singleton*, garantizamos que todas las operaciones de base de datos se realicen a través de una única conexión compartida, mejorando así la eficiencia y estabilidad del sistema.

6.2.2. Patrón Repository

El patrón *Repository* proporciona una abstracción sobre la lógica de acceso a los datos, permitiendo a las capas superiores interactuar con los datos sin conocer los detalles de la

persistencia. En el Modulo de comunicación de nuestro proyecto (Figura 3.1), este patrón se implementa para gestionar las operaciones de lectura y escritura en la base de datos, representadas en el Almacén de Datos y los Extractores. El uso del patrón *Repository* facilita la realización de pruebas unitarias, ya que permite simular el comportamiento de la base de datos mediante la inyección de dependencias. Además, este patrón ayuda a mantener el código organizado y modular, separando claramente la lógica de negocio de los detalles de acceso a los datos.

6.2.3. Patrón MVC (*Model-View-Controller*)

El patrón *Model-View-Controller* (MVC) divide la aplicación en tres componentes principales: el Modelo, que representa los datos y la lógica de negocio; la Vista, que es responsable de la presentación de la interfaz de usuario; y el Controlador, que maneja la interacción entre el Modelo y la Vista. En nuestro proyecto, este patrón se aplica en la Interfaz de Usuario y los módulos de la Capa de Lógica de Negocio (Figura 3.1). Los controladores, representados por el Módulo de Comunicación, gestionan la lógica de negocio y la comunicación entre la vista (la Interfaz de Usuario) y el modelo (los datos manejados por los módulos de gestión). Este enfoque permite que el sistema sea más fácil de mantener y escalar, ya que cada componente tiene una responsabilidad clara y está desacoplado de los demás.

6.2.4. Patrón Factory

El patrón *Factory* es utilizado para crear objetos sin especificar la clase exacta del objeto que se va a crear. Este patrón es especialmente útil cuando el proceso de creación es complejo o cuando el sistema necesita crear diferentes tipos de objetos que comparten una interfaz común. En nuestro proyecto, el patrón *Factory* se utiliza en el Módulo de Gestión del Menú (Figura 3.1) para la generación de menús semanales personalizados. Aquí, se crean diferentes tipos de menús basados en las preferencias del usuario y otros parámetros. Al centralizar la lógica de creación en una *Factory*, facilitamos la extensión y el mantenimiento del sistema, permitiendo agregar nuevos tipos de menús sin modificar el código existente.

6.2.5. Patrón Observer

El patrón *Observer* define una relación de uno a muchos entre objetos, de manera que cuando un objeto cambia su estado, todos sus dependientes son notificados y actualizados automáticamente. Este patrón se implementa en la funcionalidad del Módulo de Gestión del Menú (Figura 3.1), que permite a los menús semanales actualizarse automáticamente cuando se modifican los datos de los productos o recetas gestionados por otros módulos como el Módulo de Gestión de Recetas. Los menús actúan como *Observers* que están suscritos a cambios en los ingredientes o recetas, asegurando que cualquier cambio en estos datos se refleje instantáneamente en los menús semanales de los usuarios. Este patrón es clave para mantener la coherencia y actualización en tiempo real de la información presentada al usuario.

Los patrones de diseño implementados en este proyecto han sido fundamentales para lograr una arquitectura de software eficiente y escalable. Cada patrón ha proporcionado soluciones específicas a problemas comunes en el desarrollo de la aplicación, desde la gestión de la conexión a la base de datos hasta la creación y actualización dinámica de menús semanales. La aplicación de estos patrones no solo ha permitido mantener el código organizado y fácilmente mantenible, sino que también ha facilitado la colaboración

entre los desarrolladores y ha mejorado la robustez del sistema frente a posibles cambios o expansiones futuras. Estos beneficios demuestran la importancia de los patrones de diseño en la creación de software de alta calidad.

CAPÍTULO 7
Desarrollo

7.1 Desarrollo de *Scrapers*

En el contexto de *web scraping*, un esquema fuente se refiere a la estructura o formato de los datos tal como aparecen en las páginas web que se están scrapeando. Esto incluye elementos HTML como etiquetas, atributos, nombres de clases, identificadores (IDs), etc. Por otro lado, un esquema destino es la manera en que estos datos extraídos se organizan y almacenan en nuestro sistema, ya sea en una base de datos, archivos, o estructuras de datos específicas como objetos en un lenguaje de programación.

En nuestro caso, es el esquema fuente proviene de la página de Mercadona [17], de la cual extraeremos los productos para conocer sus precios, y la página de FatSecret [9], de donde obtendremos tanto las calorías de cada producto como recetas publicadas por los usuarios.

A continuación, se describen las fuentes de datos en términos de su esquema fuente y cómo esos datos se mapearán al esquema destino, representado en los objetos `Receta` y `Producto` que podemos ver en la Figura 7.4.

7.1.1. Esquema Fuente en la Página de Mercadona

Para ilustrar el esquema fuente utilizado en el proceso de mapeo semántico, a continuación se presenta un diagrama conceptual que refleja la estructura de datos extraída de la página de Mercadona.

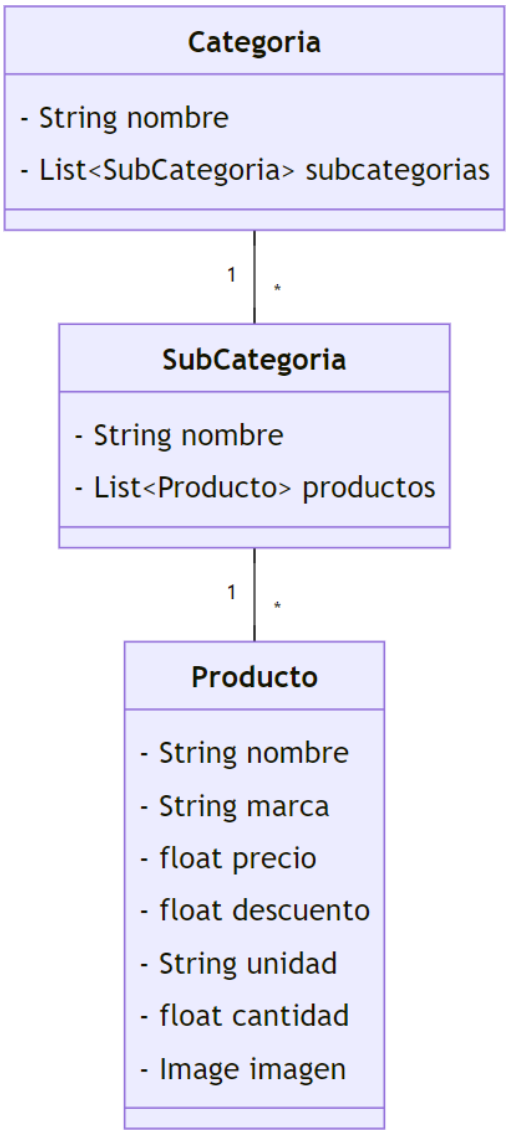

Figura 7.1: Esquema Fuente de Mercadona.

Este esquema conceptual, representado en la figura 7.1, captura los elementos principales y sus relaciones, que posteriormente se mapean al esquema global de nuestro sistema, tal como se describe en la tabla 7.1:

Tabla 7.1: Plantilla de definición de *mapping* semántico

Datos esquema global	Datos esquema origen	Transformaciónes
nombreProducto	`productName`	Copiar
precio	`previousUnitPrice`	Convertir
precioConDescuento	`unitPrice`	Convertir
calorias		
unidadMedida	`fcontainerType`	Copiar
cantidad	`size`	Copiar
fechaDelDato	Meta información	Asignar

En la tabla 7.1, se muestra cómo se realiza el mapeo de los datos desde el esquema conceptual de la página de Mercadona hacia el esquema global de nuestro sistema. Cada

fila describe un campo específico del esquema global, la correspondiente fuente de datos y la transformación necesaria para obtener el dato en el formato deseado.

7.1.2. Esquema Fuente en la Página de FatSecret para Calorías

Para completar el campo de "caloría" que estaba pendiente en la tabla anterior, es necesario extraer la información desde la página de FatSecret. A continuación, se presenta un diagrama conceptual (figura 7.2) que ilustra el esquema de datos utilizado para extraer la información nutricional relevante:

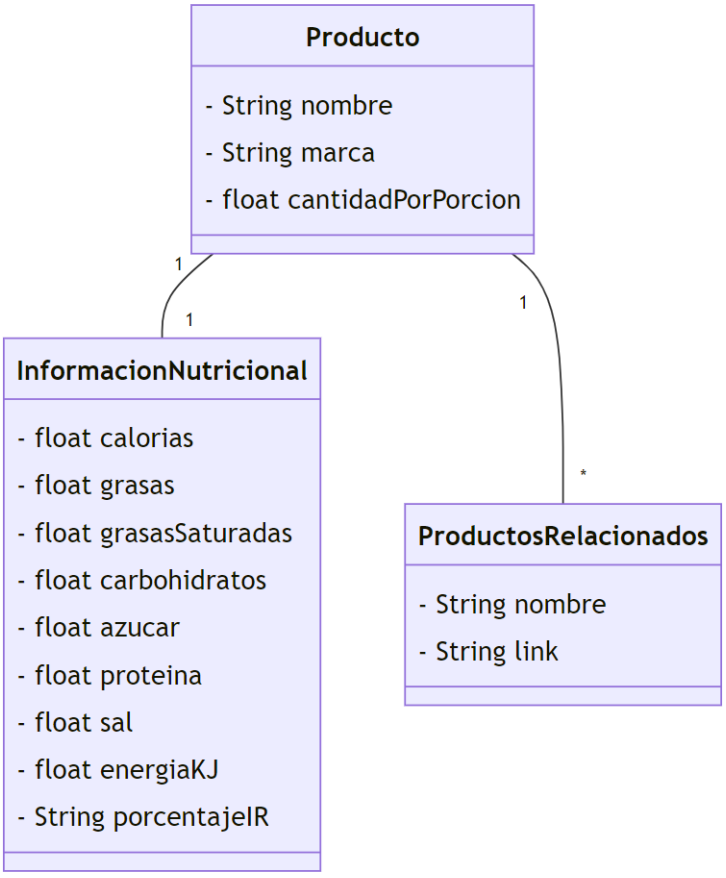

Figura 7.2: Esquema Fuente de FatSecret para Calorías.

En este caso, el valor de las calorías se puede obtener utilizando el campo `totalCalories` dentro del esquema representado en la figura 7.2. Este valor será mapeado directamente al campo "calorías" en nuestro esquema global.

Este proceso garantiza que todos los datos relevantes se extraen y transforman adecuadamente para su integración en nuestro sistema. La tabla 7.2 muestra el *mapping* realizado:

Tabla 7.2: Plantilla de definición de *mapping* semántico

Datos esquema global	Datos esquema origen	Transformación
nombreProducto		
precio		
precioConDescuento		
calorias	`totalCalories`	Copiar
unidadMedida		
cantidad		
fechaDelDato		

7.1.3. Esquema Fuente en la Página de FatSecret para Recetas

Para extraer la información de las recetas desde la página de FatSecret, hemos analizado cómo se organiza la información en dicha plataforma y hemos generado un esquema conceptual que ilustra los elementos clave necesarios para este proceso.

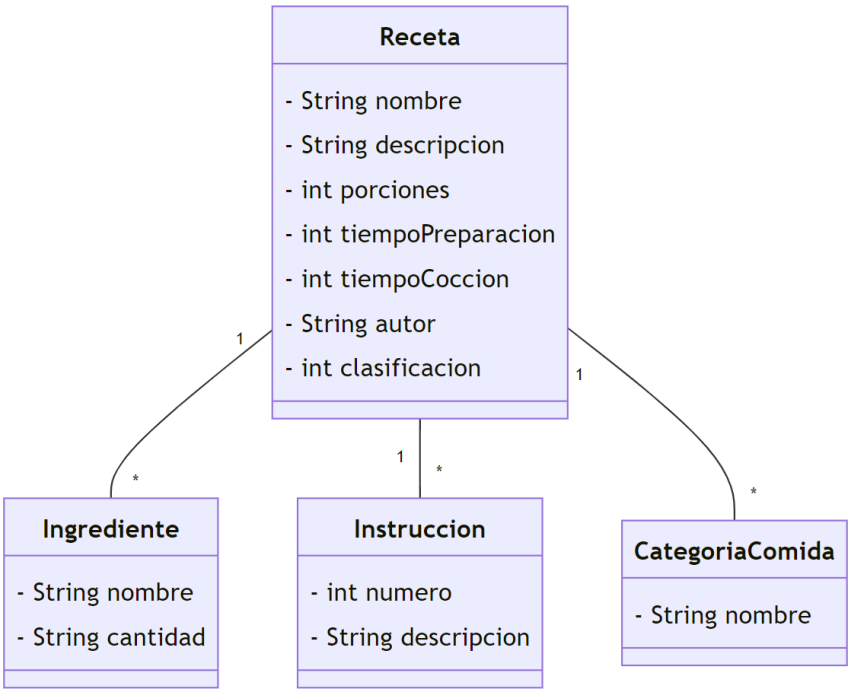

Figura 7.3: Esquema Fuente de FatSecret para Recetas.

La imagen muestra la estructura de datos que hemos identificado para capturar la información relevante de las recetas en FatSecret. Esta estructura incluye campos como el nombre de la receta, descripción, valor nutricional, tiempo de preparación, tipo de comida, entre otros.

La tabla 7.3 detalla cómo estos elementos del esquema fuente son mapeados al esquema destino en nuestro sistema:

Tabla 7.3: Plantilla de definición de *mapping* semántico para recetas

Datos esquema destino	Datos esquema origen	Transformación
nombre	`title`	Copiar
descripción	`summary`	Copiar
valorNutricional	`nutrientValue`	Copiar
precioEstimado		Calcular
tiempo	`prepTime, cookTime`	Copiar y sumar
tipoComida	`mealtypes`	Extraer lista
imagen	`imageUrl`	Extraer URL
esFavorita		
etiqueta		Definir

Este esquema fuente muestra cómo se organiza la información sobre recetas en la página de FatSecret en forma de elementos estructurados. Estos datos serán extraídos y mapeados al esquema destino en nuestro sistema para su almacenamiento y análisis posteriores.

7.1.4. Esquema Destino en el Sistema

El esquema destino define cómo organizaremos estos datos extraídos en nuestro sistema. En la Figura 7.4 se muestran los objetos Receta y Producto, que representan la estructura final de almacenamiento de los datos.

El objeto `Receta` relacionará varios objetos `Producto` y permitirá estructurar recetas completas con sus ingredientes correspondientes, usando los datos extraídos y almacenados en los objetos `Producto`.

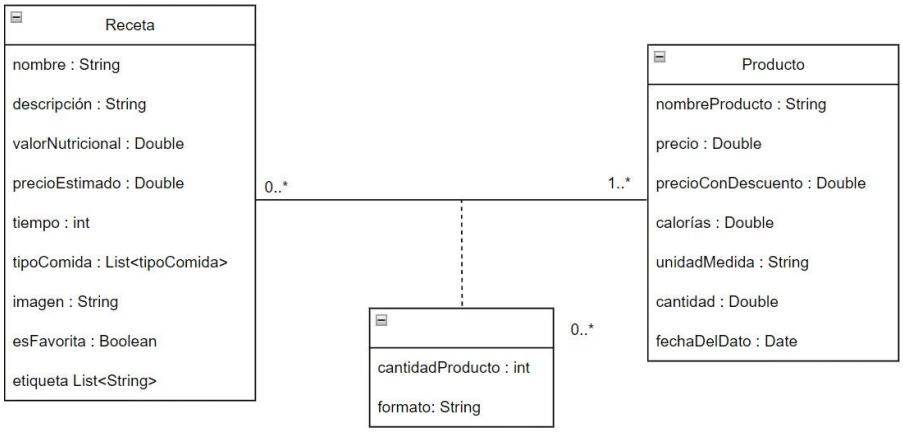

Figura 7.4: Objetos Receta y Producto.

La Figura 7.4 muestra la relación entre los objetos Receta y Producto, representando el esquema destino en nuestro sistema. Cada receta contiene varios productos, especificando la cantidad y el formato del producto utilizado. Los productos tienen atributos como nombre, precio, calorías, y unidad de medida. Este modelo permite organizar y relacionar la información extraída de las fuentes de datos de manera eficiente.

En resumen, el proceso de *scraping* implica mapear los datos desde el esquema fuente de las páginas web hacia el esquema destino en nuestro sistema, representado por los objetos Receta y Producto. Esto asegura que los datos se almacenen de manera estructurada y utilizable para futuras consultas y análisis.

7.1.5. Extracción de productos

A continuación, explicaremos el flujo que realiza cada uno de los *scrapers* para obtener la información desde el esquema fuente hasta el esquema destino.

Primero, nos centraremos en el objeto Producto. Para conseguir la mayoría de los campos que contiene un producto, nos dirigiremos a la página de Mercadona, donde encontraremos una imagen como en la figura 7.5

Figura 7.5: Producto ejemplo Mercadona

Podemos apreciar que contiene prácticamente todos los campos necesarios, excepto uno: las calorías del producto. Dado que este campo es vital para nuestro objetivo, lo obtendremos de otra fuente, que se explicará más adelante. Para explorar la página de Mercadona, debemos entender que esta es dinámica, es decir, carga los datos a medida que se navega. Esto implica que debemos usar técnicas de *scraping* dinámico. Para ello, utilizaremos Selenium, que nos permite añadir tiempos de espera para asegurar que los elementos se carguen antes de intentar extraerlos.

La página de Mercadona tiene la siguiente estructura(figura 7.6):

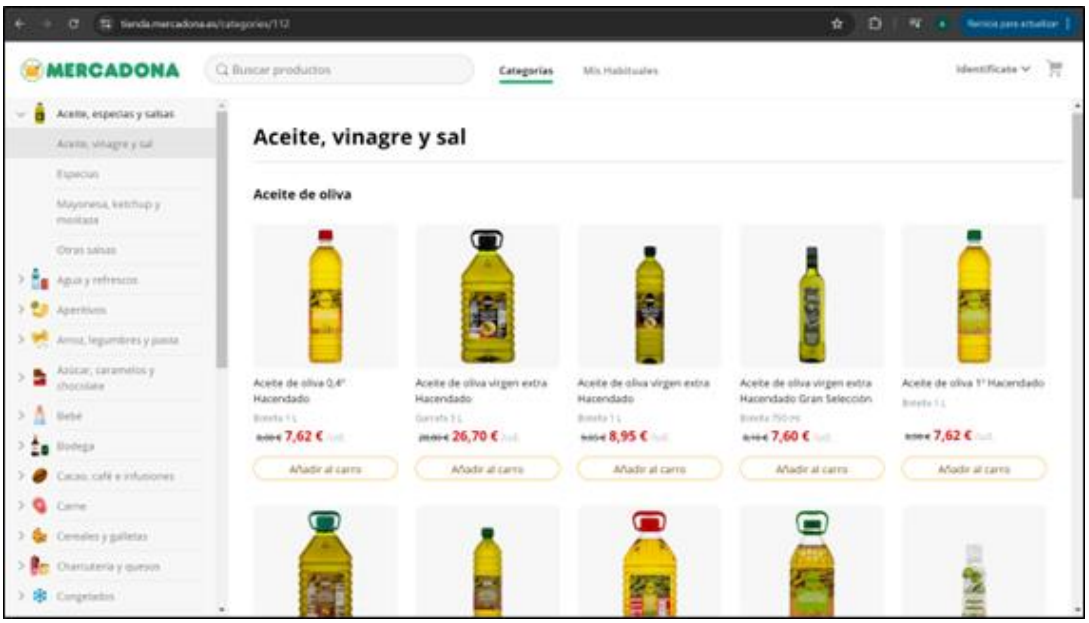

Figura 7.6: Pagina principal Mercadona

A la izquierda, hay categorías que, al hacer clic en cualquiera de ellas, despliegan un *combo box* de subcategorías. Finalmente, al hacer clic en cada subcategoría, tendremos acceso a los productos. Para obtener todos los productos, se desarrolló un *scraper* que sigue el siguiente flujo (figura 7.7):

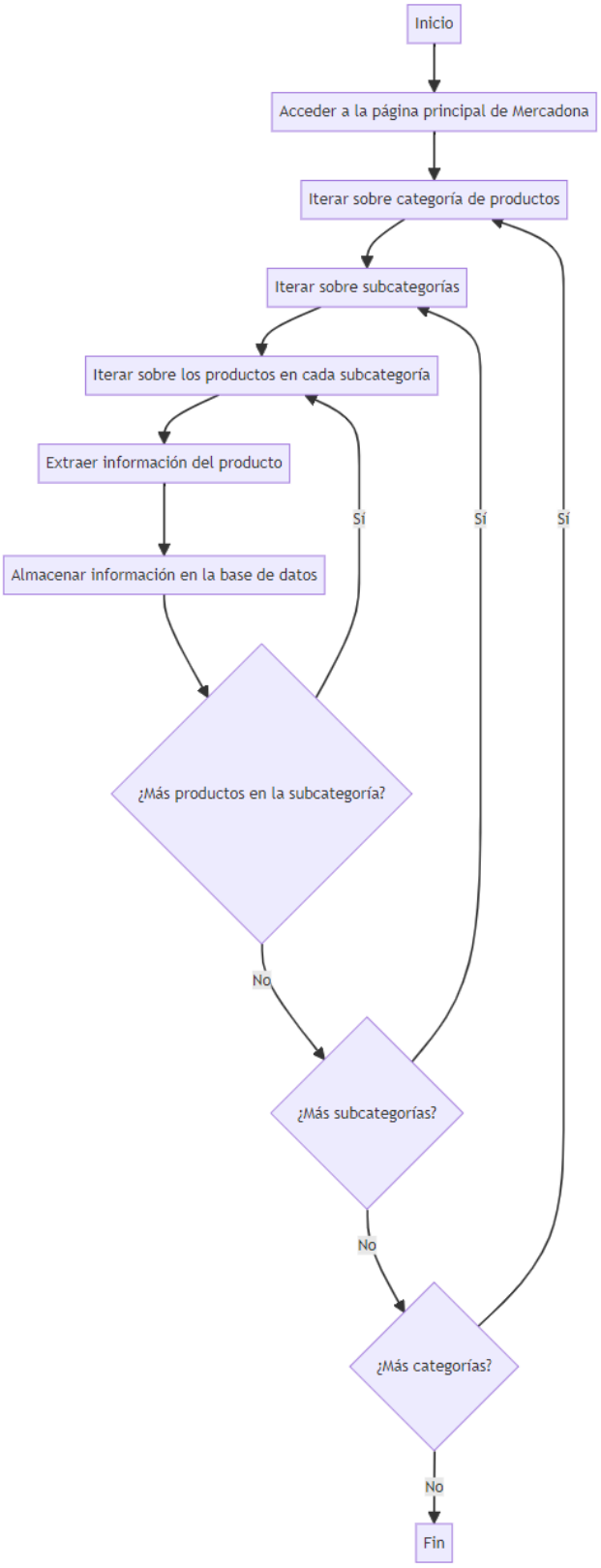

Figura 7.7: Flujo *scraper* Mercadona

Descripción del diagrama de flujo

1. **Inicio:** El proceso comienza accediendo a la página principal de Mercadona.
2. **Iterar sobre Categoría de Productos:** Se selecciona una categoría de productos y se itera dentro de ella.
3. **Iterar sobre Subcategorías:** Se itera sobre cada subcategoría dentro de la categoría seleccionada.
4. **Iterar sobre Productos:** Dentro de cada subcategoría, se itera sobre cada producto.
5. **Extraer Información:** Se extrae la información relevante del producto (nombre, precio, etc.).
6. **Almacenar Información:** La información extraída se almacena en la base de datos.
7. **Decisión de Más Productos:** Se verifica si hay más productos en la subcategoría.
   - Si hay más productos, se continúa la iteración.
   - Si no hay más productos, se verifica si hay más subcategorías.
8. **Decisión de Más Subcategorías:** Se verifica si hay más subcategorías.
   - Si hay más subcategorías, se continúa la iteración.
   - Si no hay más subcategorías, se verifica si hay más categorías.
9. **Decisión de Más Categorías:** Se verifica si hay más categorías.
   - Si hay más categorías, se continúa la iteración.
   - Si no hay más categorías, el proceso finaliza.

Una vez que se extraen los datos, es necesario procesarlos. Para ello, se ha realizado el mapeo establecido en la tabla 7.1

Para completar los campos del objeto Producto que están vacíos, utilizaremos la página de FatSecret. Esta página nos permite buscar cada uno de los productos que existen en Mercadona y obtener la información sobre las calorías. A continuación, la página donde realizaremos la búsqueda (figura 7.8):

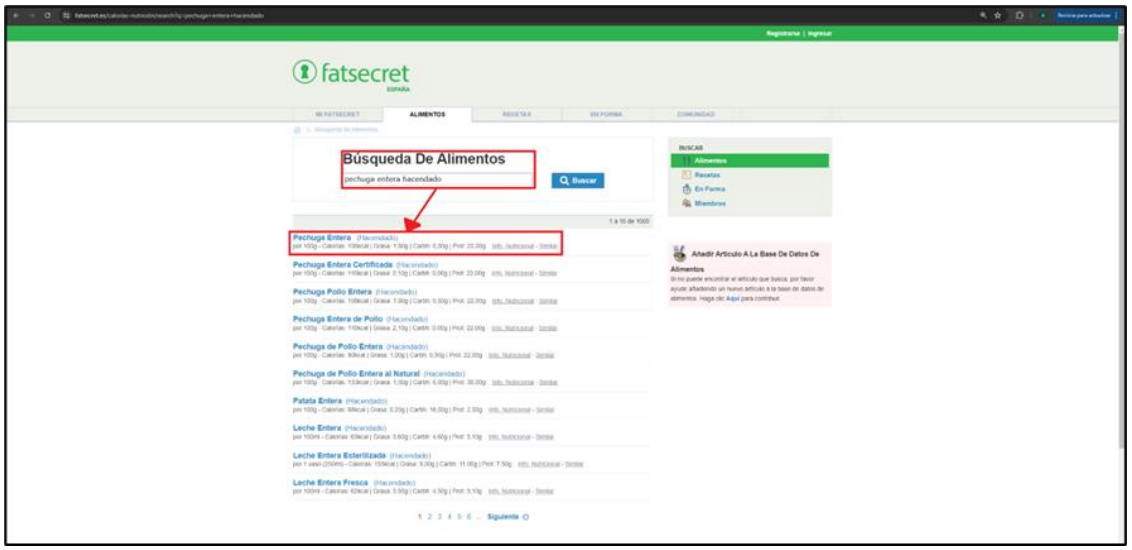

Figura 7.8: Página principal de calorías de FatSecret

Al ingresar el nombre del producto en la caja de búsqueda de FatSecret, tendremos acceso a una lista de resultados. Seleccionamos el producto deseado de la lista mostrada en la figura 7.8, lo que nos llevará a una página específica con la información nutricional del producto, incluyendo las calorías como es mostrado en la figura 7.9.

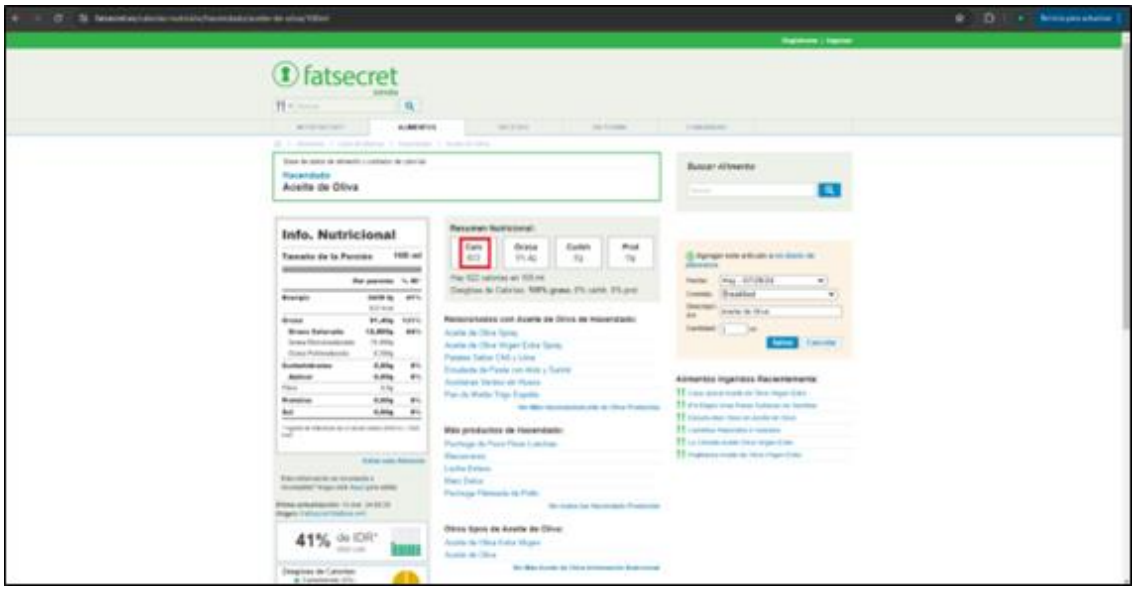

Figura 7.9: Página específica de calorías de FatSecret

A continuación, el flujo (figura 7.10) que realizara el *scraper* automáticamente para extraer todos los datos:

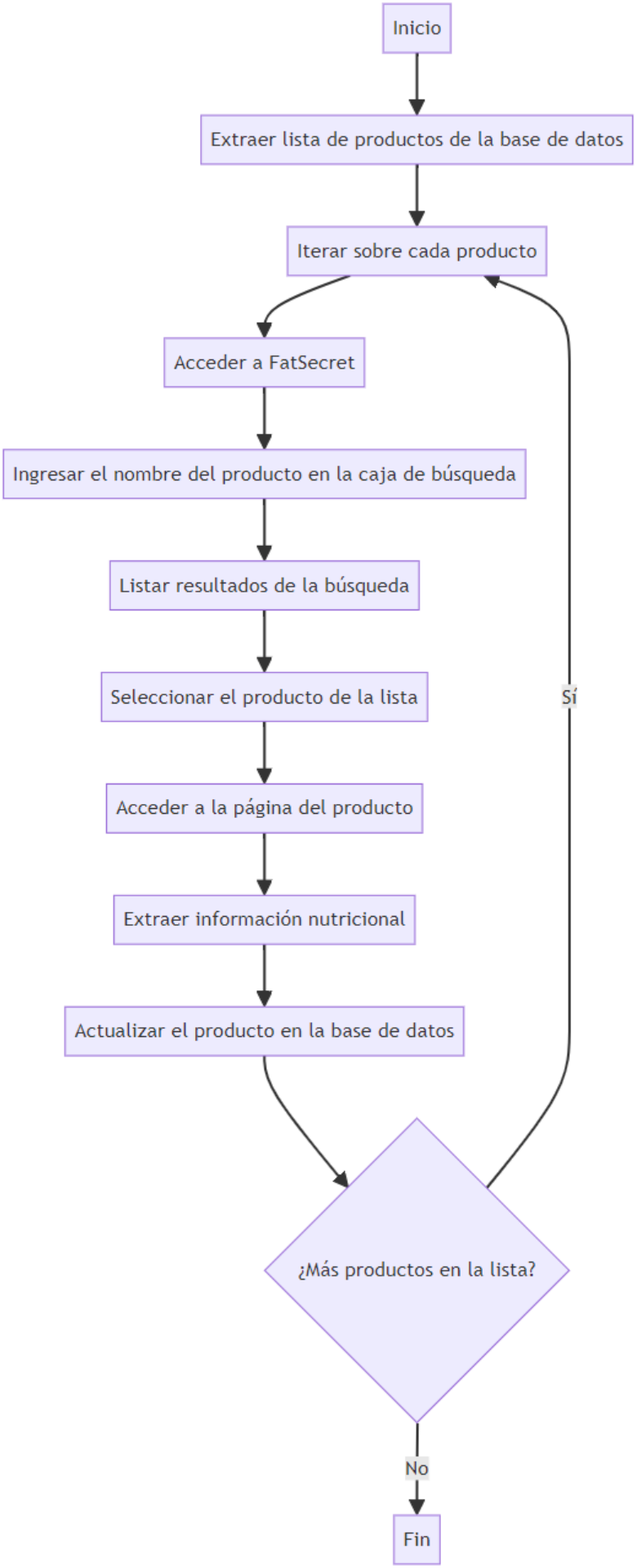

Figura 7.10: Flujo *scraper* FatSecret

Descripción del Diagrama de Flujo

1. **Inicio:** El proceso comienza.

2. **Extraer Lista de Productos:** Se extrae una lista con todos los productos de la base de datos.

3. **Iterar sobre Cada Producto:** Se itera sobre cada producto de la lista.

4. **Acceso a FatSecret:** Se accede a la página de FatSecret.

5. **Búsqueda del Producto:** Se ingresa el nombre del producto en la caja de búsqueda de FatSecret.

6. **Listar Resultados:** FatSecret muestra una lista de resultados coincidentes.

7. **Seleccionar Producto:** Se selecciona el producto deseado de la lista de resultados.

8. **Acceso a la Página del Producto:** Al seleccionar el producto, se accede a una página específica con la información nutricional.

9. **Extracción de Información:** Se extrae la información nutricional relevante, incluyendo las calorías.

10. **Actualizar Producto:** La información extraída se actualiza en la base de datos.

11. **Decisión de Más Productos:** Se verifica si hay más productos en la lista.

    - Si hay más productos, se continúa la iteración.
    - Si no hay más productos, el proceso finaliza.

12. **Fin:** El proceso se completa una vez que todos los productos han sido procesados.

7.1.6. Extracción de Recetas

La extracción de recetas inicialmente la realizaremos a partir de la página de FatSecret, ya que esta contiene una gran variedad y una estructura que facilita la extracción. Para recorrer las recetas, encontramos la página dividida convenientemente en “Platos Principales”, que usaremos para comidas y cenas, y “Desayunos”, que utilizaremos para los desayunos. A continuación, en la figura 7.11 se muestra cómo se ve la página desde donde extraeremos las recetas:

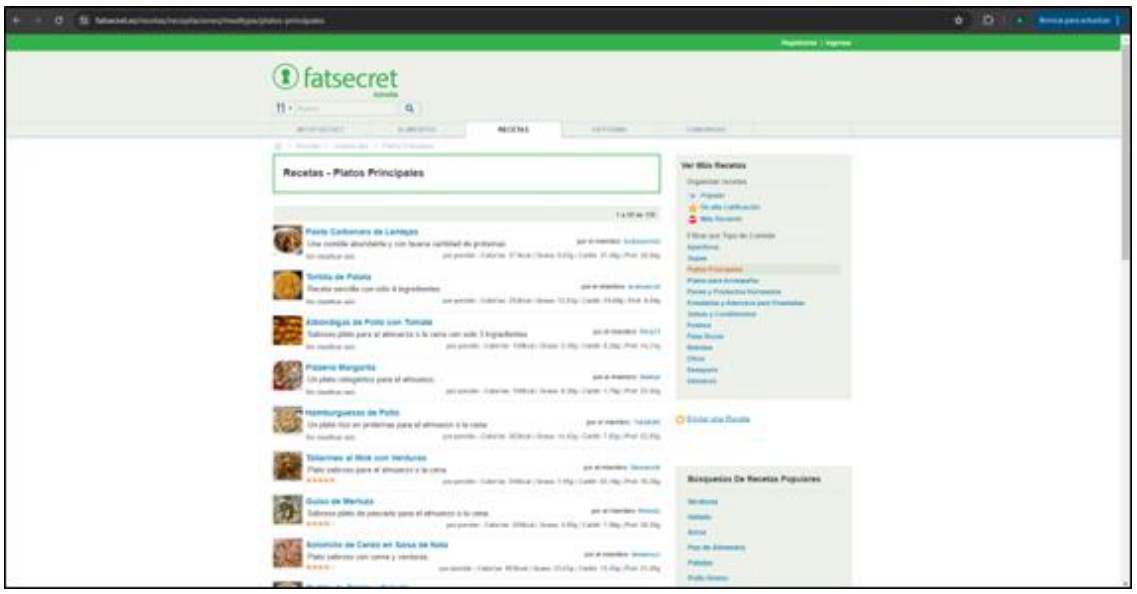

Figura 7.11: Página de recetas de FatSecret

Al acceder a cada una de las recetas, podemos ver toda la información de la siguiente manera(figura 7.12):

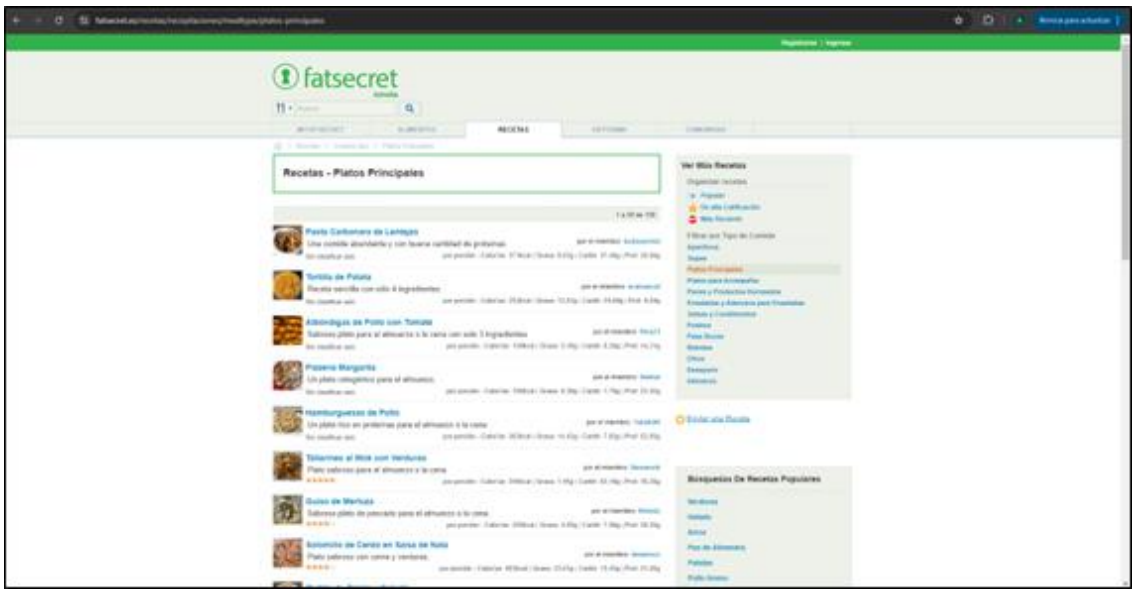

Figura 7.12: Página de receta especifica de FatSecret

En este caso, extraer la información es la parte más sencilla, ya que las recetas son de diferentes usuarios, quienes pueden escribirlas de múltiples maneras distintas, haciendo la tarea del mapeo muy compleja. Aquí es donde entra en juego el uso de la Inteligencia Artificial, que veremos en el siguiente punto. El flujo (figura 7.13) que realizaremos para la extracción de las recetas es el siguiente:

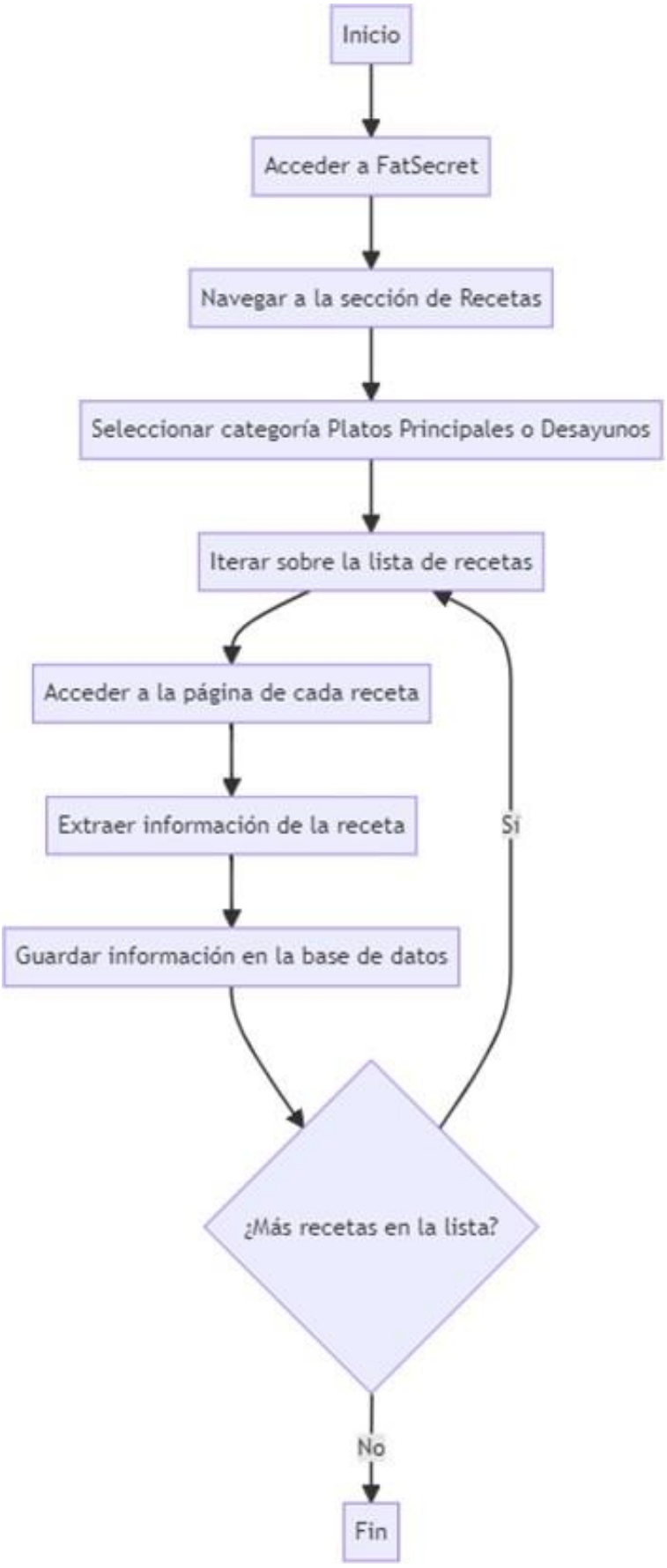

Figura 7.13: Flujo del *scraper* de recetas

Descripción del Diagrama de Flujo

1. **Inicio:** El proceso comienza.
2. **Acceder a FatSecret:** Se accede a la página de FatSecret.
3. **Navegar a la sección de Recetas:** Se dirige a la sección de recetas en FatSecret.
4. **Seleccionar Categoría:** Se selecciona la categoría “Platos Principales” o “Desayunos”.
5. **Iterar sobre la Lista de Recetas:** Se itera sobre cada receta en la lista.
6. **Acceder a la Página de la Receta:** Se accede a la página específica de la receta seleccionada.
7. **Extraer Información de la Receta:** Se extrae la información relevante de la receta.
8. **Guardar Información en la Base de Datos:** Los datos extraídos se guardan en la base de datos.
9. **Decisión de Más Recetas:** Se verifica si hay más recetas en la lista.
    - Si hay más recetas, se continúa la iteración.
    - Si no hay más recetas, el proceso finaliza.
10. **Fin:** El proceso se completa una vez que todas las recetas han sido procesadas.

Transformación de datos automática con IA

En este apartado entra en juego el procesamiento de lenguaje natural (PLN). Como mencionamos anteriormente, el objetivo es crear los objetos que se pueden ver en la figura 7.4. Si analizamos la imagen, notamos que una Receta está compuesta por varios Productos. La pregunta es cómo crear un mecanismo que, al extraer las recetas, asigne a cada ingrediente a un Producto existente en nuestra base de datos. La solución la encontramos en la API de ChatGPT[19] , un servicio que permite a los desarrolladores interactuar con el modelo de lenguaje de OpenAI a través de llamadas REST. Esta API ofrece funcionalidades avanzadas de procesamiento de lenguaje natural, permitiendo que las aplicaciones interpreten y generen texto en lenguaje natural con gran precisión.

Primero, entendamos cómo funciona la API. Al conectar nuestra aplicación con los servicios de ChatGPT, debemos establecer los siguientes parámetros:

- **Modelo:** Seleccionar la versión de ChatGPT que queremos usar, como “gpt-4” o “gpt-3”.
- **Rol de la máquina:** Configurar el rol de la máquina para que procese nuestras peticiones de manera adecuada.
- **Rol del usuario:** Indicar a la aplicación que somos usuarios.

Para esta parte del proyecto, utilizamos el último modelo disponible en ese momento, que era ChatGPT-4. Configuramos el rol de la máquina para que recibiera una serie de ingredientes y buscara en nuestra base de datos el producto que más se asemejara al ingrediente de la receta. Es importante destacar que la API no tiene acceso directo a nuestra base de datos. Por ello, ideamos exportar la base de datos a un archivo JSON, ya que ChatGPT puede procesar documentos.

Finalmente, generamos una consulta que devolviera exactamente lo que necesitábamos. Cabe mencionar que esto es un proceso automático, a diferencia del uso de la interfaz de OpenAI, donde se puede tener una conversación para ajustar la respuesta. Por lo tanto, fue necesario crear el *prompt* perfecto para asegurar que cada respuesta fuera exitosa.

A pesar de los significativos avances en inteligencia artificial, esta tecnología aún presenta limitaciones y es propensa a cometer errores, especialmente en tareas complejas que requieren un alto grado de precisión. Durante el desarrollo de nuestro proyecto, nos enfrentamos al desafío de encontrar el *prompt* ideal para la API de ChatGPT que nos permitiera obtener respuestas precisas y útiles. Probamos una variedad de enfoques, desde consultas complejas que intentaban abarcar todo el contexto de la receta hasta las más simples que se centraban en tareas específicas. Sin embargo, muchos de estos enfoques fueron descartados debido a la imprecisión en las respuestas generadas.

En nuestro primer intento, proporcionamos la receta completa en el *prompt* y pedimos a la API que devolviera una respuesta en formato JSON. La respuesta debía estructurarse de tal manera que cada ingrediente de la receta se correspondiera con un producto existente en nuestra base de datos. Nuestro objetivo era lograr una coincidencia exacta entre los ingredientes y los productos, incluyendo detalles como el precio, la cantidad, el formato, y otros atributos relevantes. A continuación, se muestra un ejemplo del JSON que esperábamos obtener:

```
{
    "id": 1,
    "nombre": "Receta de Ejemplo",
    "descripcion": "Esta es una receta de ejemplo",
    "tiempo": 30,
    "ingredientes": {
        "Leche Entera": {
            "producto":{
                "id": 1,
                "nombreProducto": "Leche Entera hacendado",
                "precio":1.20,
                "precioConDescuento": 1.00,
                "cantidad":1.0,
                "formato": "litro",
                "fechaDelDato": "2024-06-01T00:00:00",
                "supermercado": "NombreSupermercado"
            },
            "cantidad": 1.0,
            "formato": "litro"
        },
        "Huevos": {
            "producto": {
                "id": 2,
                "nombreProducto": "Huevos",
                "productoGenerico": "Proteína",
                "precio":2.50,
                "precioConDescuento": 2.00,
                "calorias":70.0,
                "cantidad": 6.0,
                "formato": "unidades",
                "fechaDelDato": "2024-06-01T00:00:00",
                "supermercado": "NombreSupermercado"
            },
            "cantidad": 6.0,
            "formato": "unidades"
        }
    }
}
```

Listing 7.1: Ejemplo de JSON para una receta

Aunque este enfoque inicial generó algunas respuestas prometedoras, pronto descubrimos que la API de ChatGPT a menudo devolvía productos que no estaban presentes en nuestra base de datos, evidenciando una limitación en su capacidad para manejar múltiples tareas simultáneamente con precisión. Esto nos llevó a la conclusión de que era necesario simplificar nuestra solicitud y dividir las tareas para evitar errores.

Reconocimos que el aspecto más crítico del uso de ChatGPT en este contexto era establecer una relación precisa entre los ingredientes de la receta y los productos de nuestra base de datos. Por ello, decidimos abordar las partes no relacionadas con los ingredientes mediante métodos tradicionales, utilizando mapeos manuales. Para optimizar el proceso, creamos un nuevo *prompt* que proporcionaba únicamente la lista de ingredientes junto con un archivo JSON que contenía los productos de nuestra base de datos. Además, especificamos el formato exacto que queríamos en la respuesta, reduciendo así el margen de error y asegurando una mayor precisión en los resultados.

A pesar de estas mejoras, los resultados seguían siendo insuficientes. Si bien la respuesta estaba correctamente formateada, solo el primer ingrediente coincidía con la base de datos; el resto de los ingredientes no coincidían en absoluto. Con base en estos resultados, concluimos que la única forma de obtener resultados precisos era realizar consultas a la API para cada ingrediente individualmente. Por ejemplo, para una receta con tres ingredientes, se realizarían tres consultas a la API. Esta metodología proporcionó los resultados esperados, devolviendo un JSON preciso para cada ingrediente con el formato correcto.

El formato requerido para estas consultas individuales es el siguiente:

```
{
    "Ingredientes": {
        "Producto":{
            "NombreProducto": "Jamoncitos de pollo",
            "cantidad": 100,
            "formato": "g"
        },
        "Producto": {
            "NombreProducto": "Harina de arroz Hacendado",
            "cantidad": 100,
            "formato": "g"
        }
    }
}
```

Listing 7.2: Formato JSON para los ingredientes

Es importante recordar que este proceso es fundamental para establecer la relación entre el objeto Receta y el objeto Producto(figura 7.4). La información necesaria se encuentra en la forma en que los ingredientes están escritos en la receta. Esto es más claro con un ejemplo: si un ingrediente está escrito como “100 g arroz de arroz blanco” debemos extraer la cantidad (100), la unidad de medida (g), y encontrar el nombre del producto en la base de datos que coincida con “arroz blanco”, por ejemplo, “Arroz largo Hacendado”. A continuación, un diagrama del flujo(figura 7.14) que representa el proceso a la llamada del api y el resultado:

Los resultados de este método fueron excelentes, con respuestas precisas en todas las pruebas realizadas. Sin embargo, presentaba un gran problema. Como mencionamos anteriormente, el uso de la API de ChatGPT es similar a utilizar la interfaz de usuario proporcionada por OpenAI. Para las pruebas, utilizamos la interfaz dada, ya que usar la

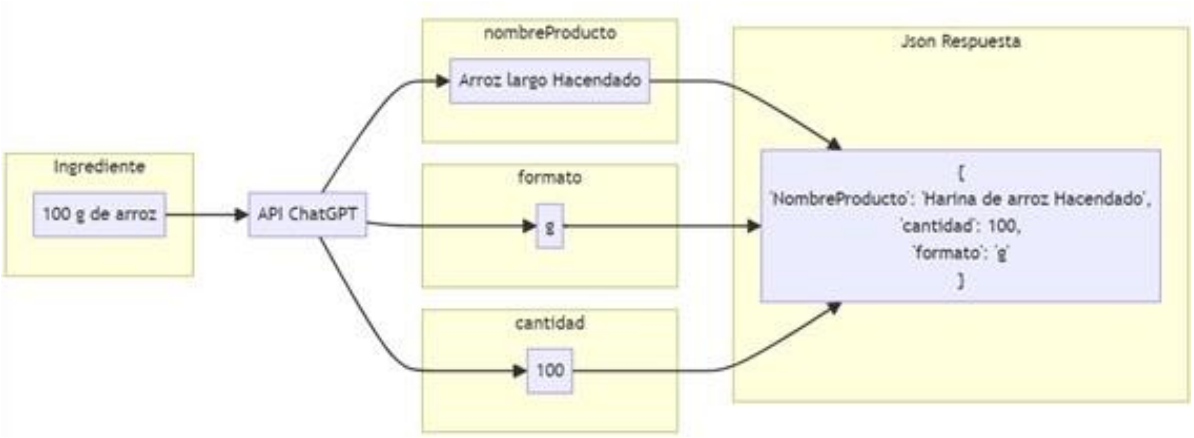

Figura 7.14: Flujo del mapeo del formato de una receta

API implicaba un costo por token. El costo es de aproximadamente $0.40 por cada mil tokens, lo cual es bastante asequible, ya que una consulta normal suele rondar entre 100 y 200 tokens.

El problema surge cuando nuestras consultas a través de la API necesitaban incluir el JSON con los productos de la base de datos, lo cual ocupaba alrededor de mil tokens. Cada consulta costaba aproximadamente $0.20, y si había un producto con cinco ingredientes, procesar la receta entera costaba $1.00. Este método se volvió inviable debido a los costos acumulados. Para superar esta limitación financiera, desarrollamos una alternativa semiautomática, que explicaremos en el siguiente punto.

Otro uso de la automatización con IA fue la asignación de etiquetas. Como se observa en la figura 7.4, el último atributo de la clase Receta es una lista de etiquetas. Este campo se completa al finalizar la recopilación de los demás datos del objeto. Las etiquetas son esenciales para la generación automática de menús semanales, lo cual se discutirá más adelante. Sirven para clasificar las recetas en categorías como Cereales y granos, Verduras, Proteínas, Lácteos, Grasas y Aceites, y Dulces. Para asignar etiquetas a cada receta, se utilizó la API de ChatGPT. Se le proporcionó un *prompt* con la lista de ingredientes, el nombre de la receta y el formato deseado para la respuesta. Los resultados fueron satisfactorios y, tras procesar la respuesta de la API, se asignaron las etiquetas correspondientes a cada receta.

Transformación de datos semi-automática con IA

A pesar de no poder usar la inteligencia artificial para asignar productos a una receta, aún era necesario crear los objetos. Se dividió el trabajo en dos procesos: uno para asignar ingredientes a los productos y otro para darles formato. Para asignar los productos, se utilizó un algoritmo de similitud de *strings*. Este algoritmo toma la lista de todos los productos en la base de datos y determina cuál es más similar al ingrediente de la receta. Para este proceso, se empleó el algoritmo "distancia de Levenshtein"[14], que calcula el número mínimo de operaciones necesarias para transformar una cadena de caracteres en otra.

Aunque este algoritmo daba resultados bastante precisos, no era lo suficientemente exacto para automatizar el proceso por completo. Un problema recurrente se presentaba con el ingrediente "sal". En la base de datos, el producto “Sal fina Hacendado” solía aparecer al final de la lista de coincidencias, ya que productos productos como “salsa” y “ensalada” mostraban mayor porcentaje de similitud. Por esta razón, fue necesario seleccionar manualmente el producto correcto de la lista de coincidencias.

Por otro lado, los ingredientes se recibían en un texto como “100 gramos de pollo“ y era necesario extraer automáticamente el ingrediente, la unidad de medida y la cantidad para las múltiples posibles combinaciones en que se puede escribir estos textos. Para resolver esto, se utilizó el mecanismo desarrollado previamente para intentar automatizar el proceso (ver figura 7.14), con la única diferencia de que lo que se devolvía en el campo del nombre del producto no era un nombre existente en la base de datos, sino simplemente el nombre del ingrediente.

En conclusión, con los fondos apropiados sería muy factible automatizar por completo el proceso de extracción, transformación y carga de datos. Sin embargo, debido a la limitación de recursos, se utilizará la automatización solo en la selección de etiquetas y en el formateo de los ingredientes.

7.2 Generación automática de Menús Semanales

En este proyecto, uno de los principales atractivos es la generación automática de menús semanales. Este enfoque tiene como objetivo facilitar al usuario la planificación de sus comidas, eliminando la necesidad de pensar en cada comida de la semana. Con el algoritmo desarrollado, logramos satisfacer las necesidades del usuario de manera rápida y eficaz, brindando soluciones personalizadas que se ajustan a sus objetivos y preferencias nutricionales.

7.2.1. Consideraciones Previas

Antes de la implementación del algoritmo de generación de menús, se definieron varios objetivos y restricciones que el sistema debe cumplir:

Objetivo Calórico Diario

Cada día de la semana debe aproximarse a un objetivo calórico específico proporcionado la fórmula de Harris-Benedict [15]. Este objetivo se divide entre las distintas comidas del día (desayuno, comida y cena).

Distribución de Calorías por Comida

El objetivo calórico diario se distribuye entre las diferentes comidas del día, por ejemplo, asignando un tercio de las calorías para el desayuno y los otros dos tercios para las comidas y cenas.

Etiquetas y Categorías de Comida

Las recetas están categorizadas por tipo de comida (desayuno, comida, cena) y etiquetadas según su contenido nutricional, tipo de plato, etc. Esto permite filtrar y seleccionar recetas adecuadas para cada momento del día.

Margen de Error

Para permitir cierta flexibilidad en la selección de recetas y evitar que el algoritmo sea demasiado estricto, se introduce un margen de error en el objetivo calórico diario. Este

margen de error define un rango dentro del cual la suma total de calorías de las recetas seleccionadas es aceptable.

7.2.2. Algoritmo de Generación de Menús

El algoritmo tiene como objetivo seleccionar un conjunto de recetas que cumpla con dos requisitos principales:

1. **Aproximación a un objetivo calórico diario especificado.**

2. **Cumplimiento de ciertas restricciones basadas en etiquetas de ingredientes o categorías alimenticias.**

Estructura General del Algoritmo

El algoritmo se basa en un proceso iterativo de selección aleatoria, evaluando en cada iteración si el conjunto de recetas seleccionadas cumple con los requisitos establecidos. Si después de un número determinado de intentos no se encuentra un conjunto óptimo, el algoritmo devuelve la mejor opción encontrada hasta ese momento.

Paso a Paso del Algoritmo

1. **Inicialización de Variables:**

   - **Calorías Mínimas y Máximas:** Se calculan `caloriasMin` y `caloriasMax` usando el objetivo calórico diario (`caloriasObjetivoDiarias`) y el margen de error (`margenError`). Estas variables determinan el rango aceptable de calorías para el conjunto de recetas seleccionadas.
   - **Mejor Selección:** Se inicializa `mejorSeleccion` como una lista vacía que almacenará el mejor conjunto de recetas encontrado.
   - **Diferencia Calórica Menor:** `menorDiferencia` se inicializa a un valor muy alto (`double.MaxValue`) para asegurar que cualquier diferencia calórica encontrada en el primer intento sea considerada la mejor inicialmente.
   - **Contador de Intentos:** `intentos` se inicializa en 0 y se establece un máximo de intentos (`maxIntentos`) para evitar bucles infinitos.

```
caloriasMin <- caloriasObjetivoDiarias * (1 - margenError)
caloriasMax <- caloriasObjetivoDiarias * (1 + margenError)
mejorSeleccion <- ListaVacia
menorDiferencia <- Infinito
intentos <- 0
maxIntentos <- 200
```

Como se muestra en la Figura 7.15, estas inicializaciones preparan las variables necesarias para comenzar el proceso de selección.

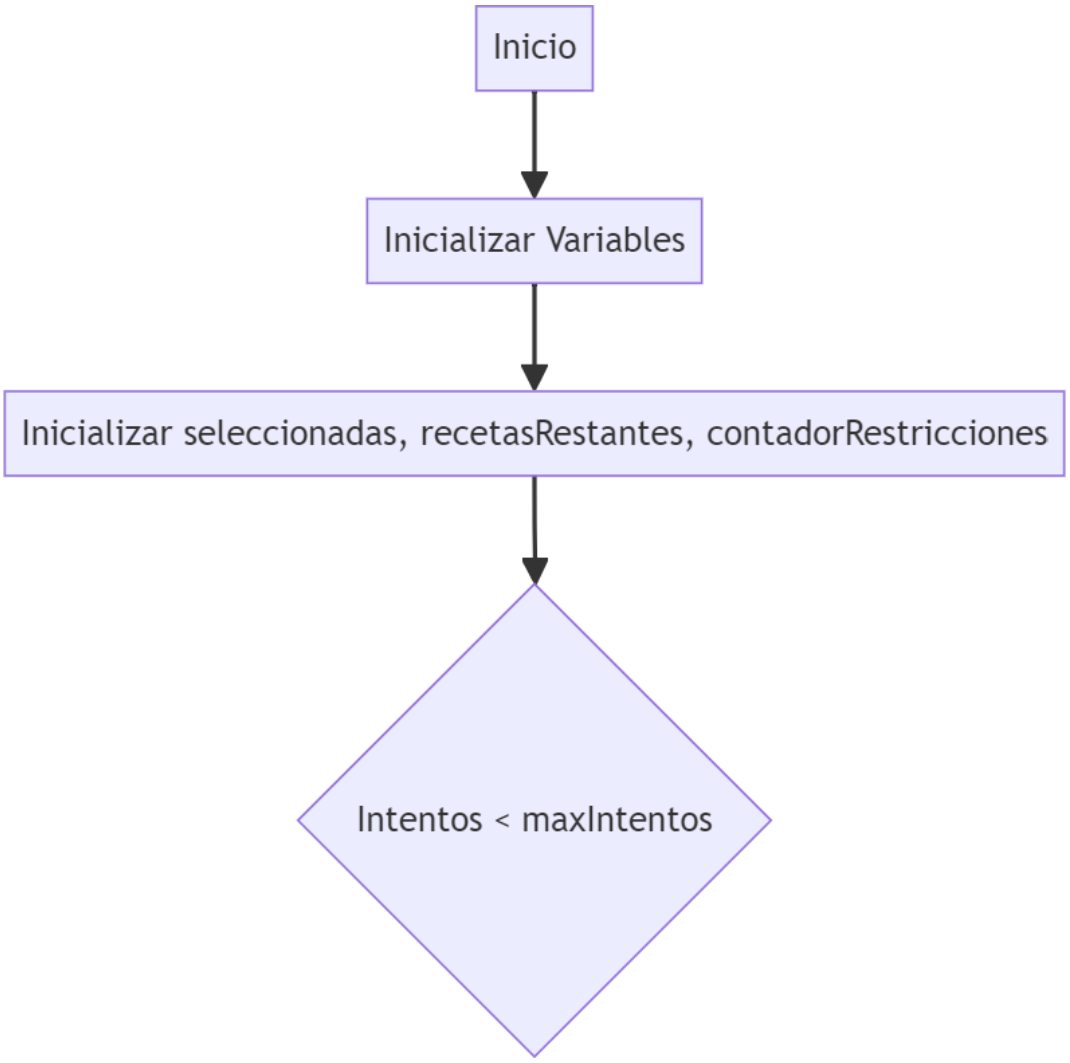

Figura 7.15: Inicialización de variables y preparación para el bucle principal.

2. **Inicio del Bucle Principal (`do-while`):**

   - Este bucle se repite hasta que se alcancen `maxIntentos`. En cada iteración, el algoritmo intenta generar un conjunto válido de recetas, como se ilustra en la Figura 7.16.

```
hacer {
    seleccionadas <- ListaVacia
    recetasRestantes <- CopiarLista(recetas)
    caloriasTotales <- 0
    contadorRestricciones <- CopiarDiccionario(restricciones)
```

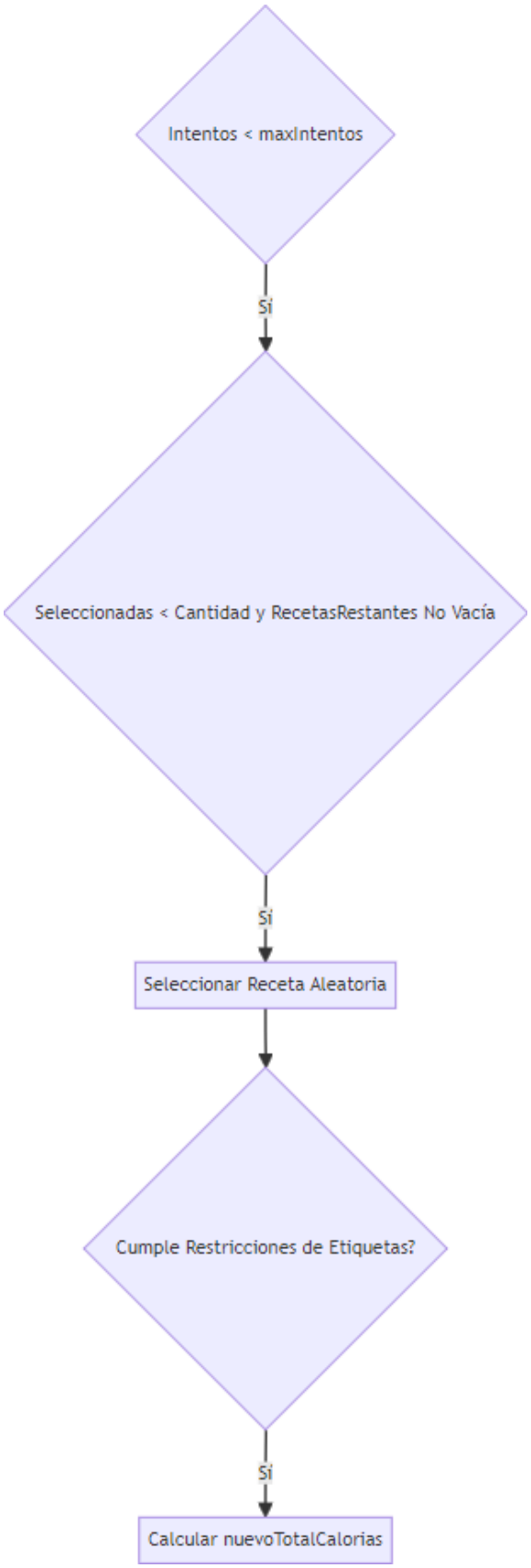

Figura 7.16: Inicio del bucle principal para la selección de recetas.

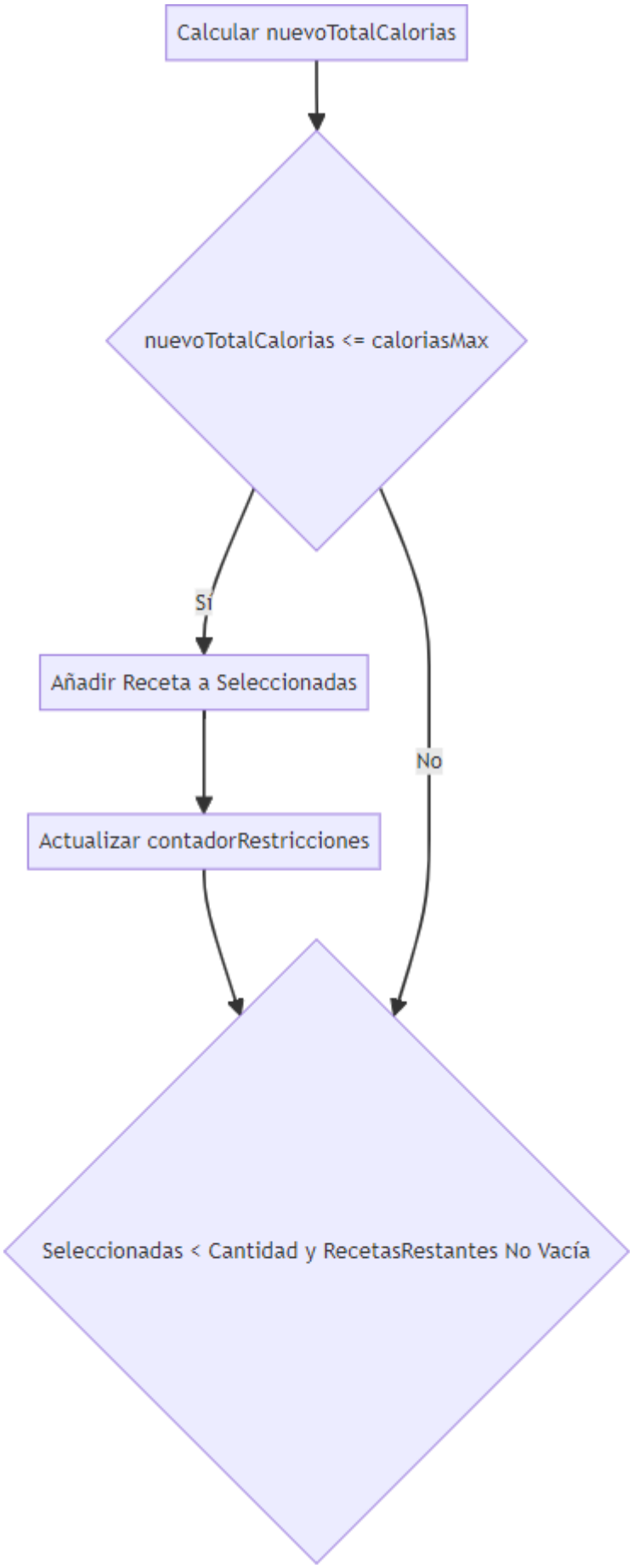

Figura 7.17: Proceso de selección y verificación de restricciones.

3. **Selección Aleatoria de Recetas:**
   - Se inicializan listas para `seleccionadas` (recetas seleccionadas en este intento) y `recetasRestantes` (recetas que aún no han sido seleccionadas).
   - **Contador de Restricciones:** Se crea un contador (`contadorRestricciones`) para llevar la cuenta de cuántas veces se ha seleccionado una receta que cumple con cada etiqueta, de acuerdo con las restricciones impuestas.
4. **Proceso de Selección Dentro del Bucle** `while`**:**
   - **Selección de una Receta Aleatoria:** Se selecciona una receta al azar de `recetasRestantes`.
   - **Verificación de Restricciones:** El algoritmo verifica si la receta seleccionada cumple con las restricciones de etiquetas. Si una etiqueta ha alcanzado su límite según el contador, la receta es descartada y el algoritmo continúa con la siguiente.
   - **Simulación de Calorías:** Si la receta cumple con las restricciones de etiquetas, se simula su inclusión en el conjunto sumando sus calorías a `caloriasTotales`.
   - **Agregado de la Receta:** Si la simulación muestra que las calorías totales aún están dentro del rango permitido (`caloriasMax`), la receta se agrega a `seleccionadas` y se actualiza el contador de etiquetas correspondiente.
   - **Relajación de Restricciones:** Si todas las recetas disponibles han sido evaluadas y no se ha alcanzado el número requerido de recetas (`cantidad`), el algoritmo relaja las restricciones (reinicia `recetasRestantes` y `contadorRestricciones`) y vuelve a intentarlo. Esto previene un bucle infinito en situaciones donde es imposible cumplir con todas las restricciones.

```
mientras (seleccionadas.Tamaño < cantidad && recetasRestantes NO Vacía) {
  index <- NumeroAleatorio(0, recetasRestantes.Tamano)
  recetaSeleccionada <- recetasRestantes[index]

  // Verificar restricciones
  cumpleRestricciones <- True
  por cada etiqueta en recetaSeleccionada.Etiquetas {
      si (contadorRestricciones[etiqueta] >= restricciones[etiqueta]) {
          cumpleRestricciones <- False
          romper
      }
  }

  // Simulación de calorías
  si (cumpleRestricciones) {
    nuevoTotalCalorias <- caloriasTotales + recetaSeleccionada.ValorNutricional
    seleccionadas.Agregar(recetaSeleccionada)
    caloriasTotales <- nuevoTotalCalorias
    recetasRestantes.Remover(index)

    // Actualizar contador de etiquetas
    por cada etiqueta en recetaSeleccionada.Etiquetas {
        contadorRestricciones[etiqueta]++
    }
  }
```

```
        // Relajación de restricciones
        si (recetasRestantes.Vacía && seleccionadas.Tamano < cantidad) {
            recetasRestantes <- CopiarLista(recetas)
            contadorRestricciones <- CopiarDiccionario(restricciones)
        }
    }
```

La Figura 7.18 muestra cómo el algoritmo gestiona la selección aleatoria y la verificación de las restricciones de etiquetas y calorías.

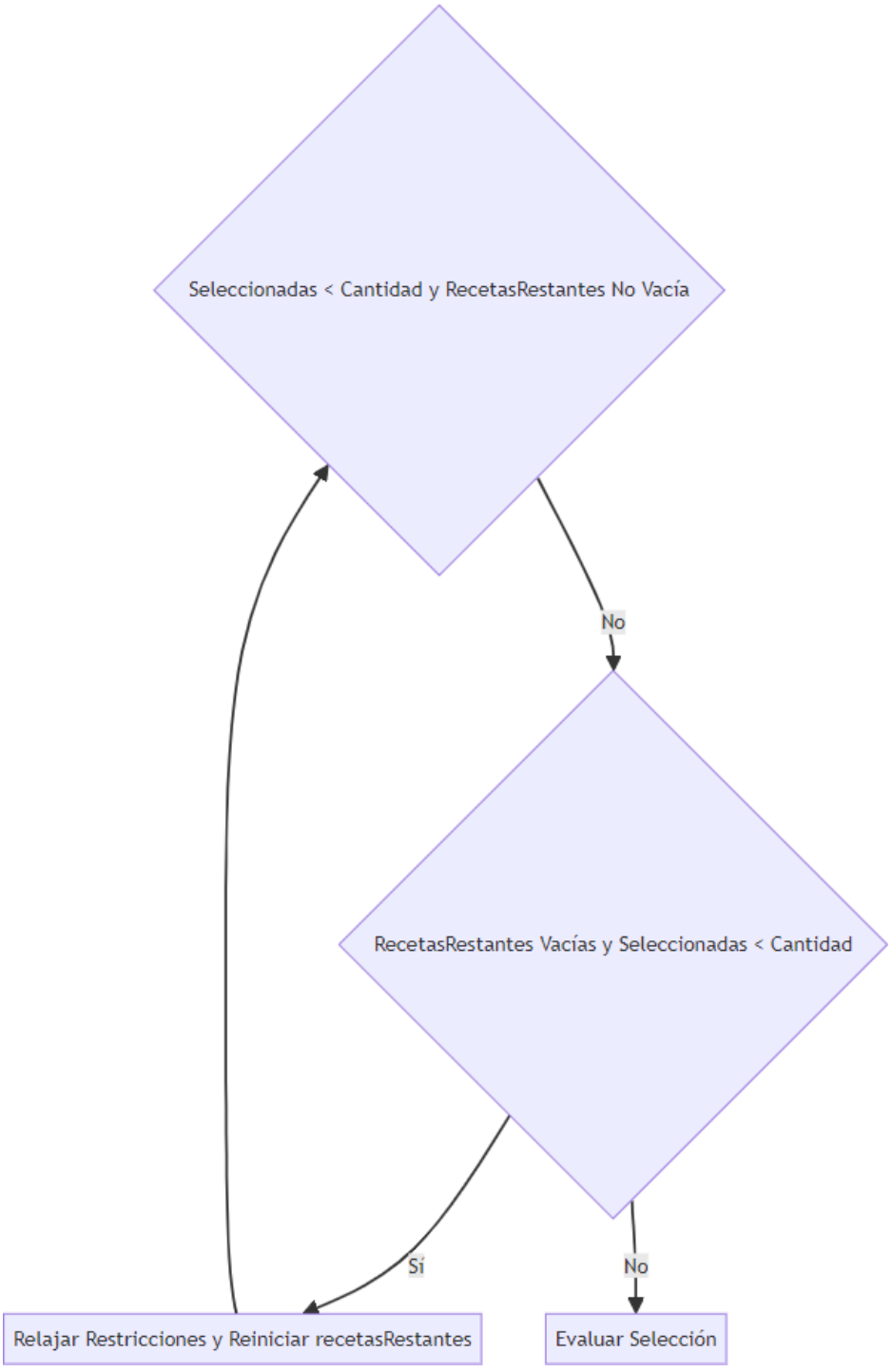

Figura 7.18: Proceso de selección aleatoria y verificación de restricciones.

5. Evaluación de la Selección:

- **Comparación con el Mejor Resultado:** Si las calorías totales de la selección actual están dentro del rango aceptable (`caloriasMin` ≤ `caloriasTotales` ≤ `caloriasMax`) y la diferencia calórica (`diferenciaCalorica`) es menor que la mejor diferencia registrada hasta ahora, se actualiza `mejorSeleccion` con el conjunto actual.

- ***Fallback*:** Si no se encuentra una selección que cumpla con todas las restricciones pero se ha llenado la lista con el número requerido de recetas, se guarda esta selección como la mejor opción en caso de no encontrar nada mejor.

```
    // Evaluación de la selección
    si (caloriasTotales >= caloriasMin && caloriasTotales <= caloriasMax) {
        diferenciaCalorica <- Abs(caloriasObjetivoDiarias - caloriasTotales)
        si (diferenciaCalorica < menorDiferencia) {
            mejorSeleccion <- CopiarLista(seleccionadas)
            menorDiferencia <- diferenciaCalorica
        }
    }
    si (seleccionadas.Tamano == cantidad && mejorSeleccion.Tamano == 0) {
        mejorSeleccion <- CopiarLista(seleccionadas)
    }

    intentos++
} mientras (intentos < maxIntentos)
```

La Figura 7.19 ilustra cómo el algoritmo evalúa cada selección de recetas y decide si debe actualizar la mejor selección encontrada hasta el momento.

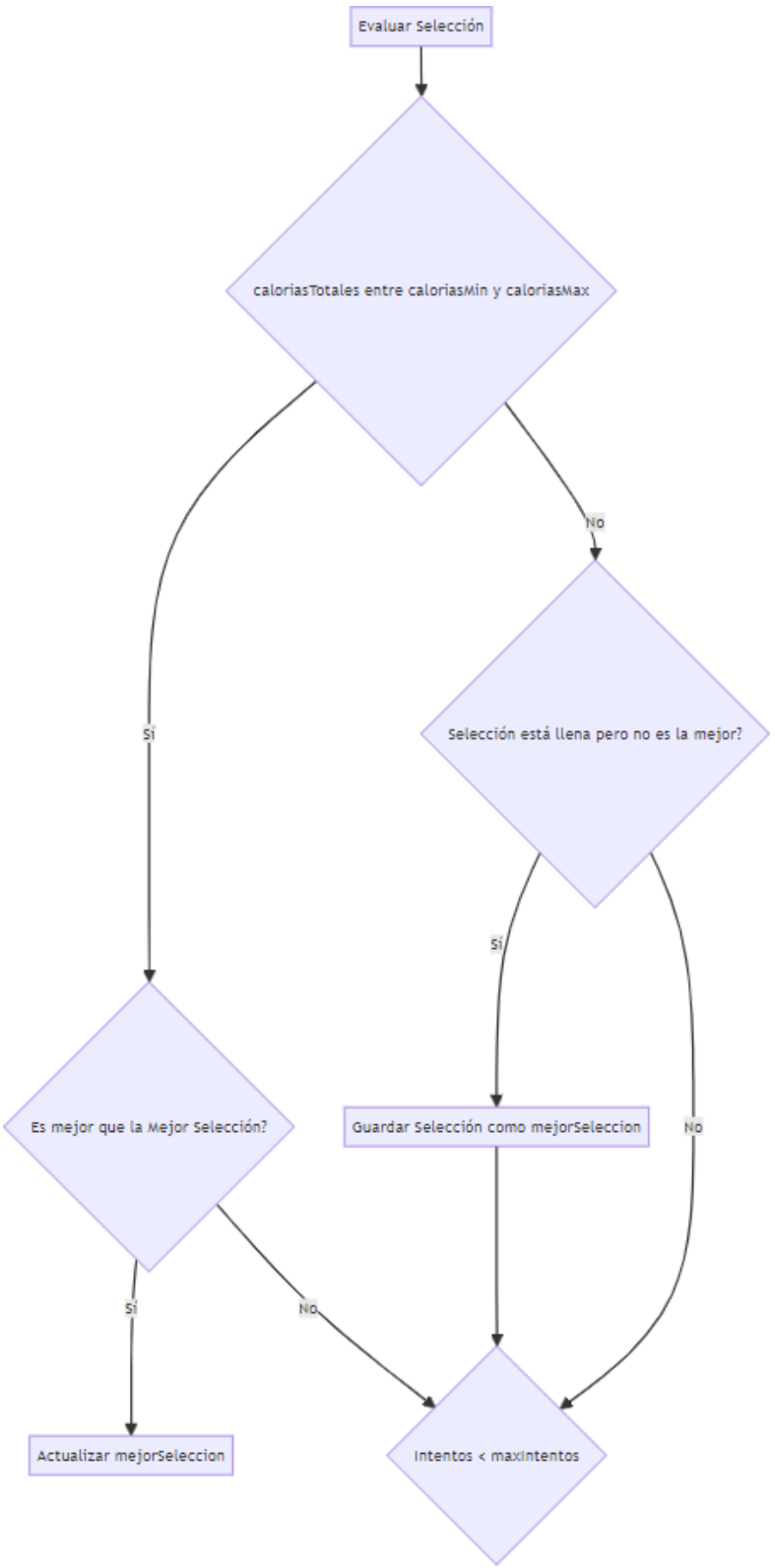

Figura 7.19: Evaluación de la selección de recetas y actualización de la mejor opción.

6. **Finalización del Bucle y Retorno del Resultado:**

   - Si después de `maxIntentos` no se encuentra un conjunto de recetas que cumpla con todas las restricciones, el algoritmo retorna la mejor selección encontrada.

Consideraciones Importantes

- **Evitar Bucle Infinito:** El algoritmo está diseñado para evitar quedarse atrapado en un bucle infinito mediante la relajación de las restricciones cuando es necesario.
- **Flexibilidad:** El algoritmo permite cierta flexibilidad en cuanto a las calorías y las restricciones de etiquetas, asegurando que siempre se pueda generar un conjunto de recetas, aunque no cumpla perfectamente con todos los criterios.

Ventajas y Limitaciones

Ventajas:

- **Optimización:** Busca activamente optimizar las selecciones de recetas en función de calorías y restricciones, adaptándose a las condiciones impuestas.
- **Robustez:** Evita fallos críticos al relajar restricciones cuando es necesario, lo que garantiza que siempre se obtenga un resultado.

Limitaciones:

- **Desviación del Objetivo:** Si las restricciones son demasiado estrictas, el algoritmo podría devolver un conjunto de recetas que no cumpla perfectamente con los objetivos calóricos o las etiquetas, aunque intentará minimizar esta desviación.
- **Complejidad Computacional:** Dado que realiza múltiples intentos y evaluaciones, el algoritmo puede ser computacionalmente intensivo, especialmente con un número elevado de intentos y restricciones complejas.

CAPÍTULO 8
Implementación

En este capítulo, se detallará la implementación técnica de la aplicación, abordando los aspectos fundamentales que permitieron convertir el diseño conceptual en un sistema funcional. Se explorarán los componentes clave que forman la base del sistema, incluyendo la creación y gestión de la base de datos, la organización y funcionamiento de los repositorios, y la estructura de los controladores que facilitan la interacción entre el *front-end* y el *back-end*. Además, se explicará cómo se ha estructurado el código *back-end* para asegurar un desarrollo escalable y fácil de mantener.

Las secciones que seguirán a esta introducción proporcionarán una visión detallada de cada uno de estos aspectos, explicando cómo se conectan y contribuyen al funcionamiento general de la aplicación.

8.1 Tecnologías Utilizadas

Para el desarrollo de la aplicación se han empleado diversas tecnologías, cuidadosamente seleccionadas por su capacidad para satisfacer las necesidades específicas del proyecto. A continuación, se detallan las principales herramientas y plataformas utilizadas:

8.1.1. Lenguajes de Programación

- ***Back-end*:** El desarrollo del *back-end* se realizó en **C**, utilizando Visual Studio como entorno de desarrollo. Este lenguaje ofrece robustez y rendimiento, especialmente en combinación con ASP.NET para la construcción de la API REST.
- ***Front-end*:** Para la capa de presentación se optó por **Java**, utilizando Android Studio para desarrollar la aplicación móvil, asegurando así una integración nativa con el sistema operativo Android.

8.1.2. *Frameworks* y Librerías

- **ASP.NET:** *Framework* utilizado para construir la API REST del *back-end*, proporcionando una base sólida para la gestión de las solicitudes HTTP y la integración con la capa de datos.
- **Android SDK:** Conjunto de herramientas proporcionado por el SDK de Android, que permitió el desarrollo de una aplicación móvil rica en funcionalidades, con una interfaz de usuario intuitiva y adaptable.

8.1.3. Bases de Datos

- **AWS RDS:** Servicio de base de datos en la nube que ofrece alta disponibilidad, escalabilidad y seguridad, características cruciales para el almacenamiento de datos a gran escala que maneja la aplicación.
- **MySQL:** Utilizado en combinación con AWS RDS, MySQL se encargó de gestionar el almacenamiento de datos estructurados, permitiendo realizar consultas eficientes y optimizar el rendimiento general del sistema.

8.1.4. Herramientas de Desarrollo

- **Visual Studio:** *Integrated development environment* (IDE) utilizado para el desarrollo del *back-end* en C, ofreciendo un amplio conjunto de herramientas para el desarrollo, la depuración y el despliegue del código.
- **Android Studio:** IDE utilizado para desarrollar la aplicación móvil en Java, proporcionando un entorno optimizado para la creación de aplicaciones nativas de Android.
- **MySQL Workbench:** Herramienta esencial para la gestión de bases de datos MySQL, facilitando la administración de esquemas, la creación de consultas y el monitoreo del rendimiento de la base de datos.
- **Git y GitHub:** Git se utilizó para el control de versiones del código, mientras que GitHub sirvió como plataforma para alojar el repositorio del proyecto, permitiendo la colaboración eficiente entre los miembros del equipo.

Estas tecnologías no solo han sido seleccionadas por su capacidad para cumplir con los requisitos del proyecto, sino también por su escalabilidad y flexibilidad, lo que asegura que la aplicación pueda evolucionar y adaptarse a nuevas necesidades y desafíos en el futuro.

8.2 Base de Datos

8.2.1. Creación de la Base de Datos en AWS RDS

Para garantizar la escalabilidad, alta disponibilidad y seguridad de la base de datos de la aplicación, se decidió utilizar Amazon Web Services (AWS) Relational Database Service (RDS). AWS RDS es un servicio administrado que facilita la configuración, operación y escalado de bases de datos relacionales en la nube. Ofrece soporte para varios motores de bases de datos, como MySQL, PostgreSQL, MariaDB, Oracle y SQL Server, proporcionando actualizaciones automáticas, copias de seguridad, recuperación ante desastres y replicación, todo ello gestionado por AWS.

La creación de la base de datos en AWS RDS implicó los siguientes pasos:

- **Selección del Motor de Base de Datos:** Se optó por MySQL debido a su robustez, familiaridad y la compatibilidad con las necesidades del proyecto.
- **Configuración del Entorno:** Se configuró una instancia RDS con los recursos necesarios (CPU, memoria, almacenamiento) para manejar la carga esperada de la aplicación.

- **Establecimiento de Parámetros de Seguridad:** Se configuraron las políticas de seguridad, incluyendo el acceso a la base de datos desde direcciones IP específicas y el uso de contraseñas seguras para la autenticación.
- **Creación de la Instancia:** Una vez configurados los parámetros, se procedió a la creación de la instancia, permitiendo que AWS RDS gestionara la infraestructura subyacente.

8.2.2. Gestión de la Base de Datos con MySQL Workbench

MySQL Workbench es una herramienta gráfica unificada que facilita el desarrollo, modelado y administración de bases de datos MySQL. Para gestionar la base de datos creada en AWS RDS, se utilizó MySQL Workbench, que permitió conectar la base de datos remota y administrar sus tablas y esquemas.

Conexión a la Base de Datos RDS:

- Se configuró una conexión remota en MySQL Workbench utilizando la dirección *endpoint* proporcionada por AWS RDS, junto con las credenciales de acceso configuradas previamente.
- Se aseguraron los permisos y accesos necesarios para realizar operaciones de administración y desarrollo en la base de datos desde Workbench.

8.3 Implementación de Repositorios (*Data Access Layer*)

En la arquitectura de la aplicación, la capa de acceso a datos se implementa utilizando el patrón de repositorio, que actúa como un intermediario entre la aplicación y la base de datos. Esta capa se encarga de gestionar las operaciones CRUD (*CREATE, READ, UPDATE, DELETE*) sobre las entidades que interactúan con la base de datos.

Librerías utilizadas

Para la implementación de los repositorios, se han utilizado varias librerías clave, entre las cuales destacan:

- **MySQL.Data**: Es la librería principal utilizada para la conexión y manejo de consultas SQL en la base de datos MySQL. Esta librería permite interactuar con la base de datos mediante comandos SQL desde el código C#.
- **Dapper**: Aunque no siempre es necesario, Dapper es una micro ORM (*Object-Relational Mapper*) que se puede utilizar para mapear los resultados de las consultas SQL directamente a objetos C#. Proporciona una forma rápida y eficiente de realizar consultas SQL y mapear los resultados a los modelos de datos sin la sobrecarga de un ORM completo.

Interfaz `IRepositorio` y su Implementación

Para estandarizar el acceso a los datos y asegurar que todas las operaciones CRUD estén disponibles en cada repositorio, se ha creado una interfaz genérica denominada `IRepositorio`. Esta interfaz define los métodos CRUD básicos que cada repositorio debe

implementar, asegurando que exista un contrato común para todas las entidades de la aplicación.

```
public interface IRepositorio<T> where T : class
{
    IEnumerable<T> GetAll();
    T GetById(int id);
    void Add(T entity);
    void Update(T entity);
    void Delete(int id);
}
```

Listing 8.1: Definición de la interfaz IRepositorio

Llamadas a la Base de Datos

La implementación de la interfaz `IRepositorio` se lleva a cabo en clases concretas como `UsuarioRepositorio`, `ProductoRepositorio`, entre otras. Cada una de estas clases hereda de `IRepositorio` y proporciona la implementación específica de los métodos CRUD para su respectiva entidad.

Las llamadas a la base de datos se realizan utilizando la librería `MySQL.Data`. Dentro de los repositorios, cada método construye y ejecuta una consulta SQL específica, utilizando los parámetros necesarios para interactuar con la base de datos de manera segura y eficiente. A continuación, se muestra un ejemplo de una consulta SQL básica para insertar datos en la tabla `Usuario`:

```
MySqlCommand(@"INSERT INTO Usuario
                (correo, nombre, contrasea, fechaNacimiento, genero,
                    ejercicio,caloriasObjetivas,idMenuSemanal)
                VALUES (@correo, @nombre, @contrasea, @fechaNacimiento,
                    @genero,@ejercicio,@caloriasObjetivas, @idMenuSemanal);",
                 conn);
```

Listing 8.2: Ejemplo de consulta SQL en un repositorio

Este ejemplo ilustra una consulta SQL sencilla que se utiliza para insertar un nuevo registro en la tabla `Usuario`. La consulta emplea parámetros para protegerse contra inyecciones SQL y para asegurar un manejo adecuado de los datos.

Además de estas consultas simples, los repositorios también pueden realizar consultas mucho más complejas, como la que se muestra a continuación, que implica la unión de múltiples tablas para recuperar datos relacionados:

```
diarioCommand = new MySqlCommand(@"
                SELECT md.idMenuDiario, md.diaDeLaSemana,
                    rDesayuno.idReceta AS DesayunoId, rDesayuno.nombre AS
                        DesayunoNombre,
                    rDesayuno.descripcion AS DesayunoDescripcion, rDesayuno.
                        valorNutricional AS DesayunoValorNutricional,
                    rDesayuno.precioEstimado AS DesayunoPrecioEstimado,
                        rDesayuno.tiempo AS DesayunoTiempo,
                    rDesayuno.imagen AS DesayunoImagen, rDesayuno.esFavorito
                        AS DesayunoEsFavorito,
                    rDesayuno.tipoComida AS DesayunoTipoComida,
                    rComida.idReceta AS ComidaId, rComida.nombre AS
                        ComidaNombre,
                    rComida.descripcion AS ComidaDescripcion, rComida.
                        valorNutricional AS ComidaValorNutricional,
                    rComida.precioEstimado AS ComidaPrecioEstimado, rComida.
                        tiempo AS ComidaTiempo,
```

```
11                  rComida.imagen AS ComidaImagen, rComida.esFavorito AS
                        ComidaEsFavorito,
12                  rComida.tipoComida AS ComidaTipoComida,
13                  rCena.idReceta AS CenaId, rCena.nombre AS CenaNombre,
14                  rCena.descripcion AS CenaDescripcion, rCena.
                        valorNutricional AS CenaValorNutricional,
15                  rCena.precioEstimado AS CenaPrecioEstimado, rCena.tiempo
                         AS CenaTiempo,
16                  rCena.imagen AS CenaImagen, rCena.esFavorito AS
                        CenaEsFavorito,
17                  rCena.tipoComida AS CenaTipoComida
18              FROM MenuDiario md
19              LEFT JOIN Receta rDesayuno ON md.idRecetaDesayuno = rDesayuno.
                    idReceta
20              LEFT JOIN Receta rComida ON md.idRecetaComida = rComida.
                    idReceta
21              LEFT JOIN Receta rCena ON md.idRecetaCena = rCena.idReceta
22              WHERE md.idMenuDiario = @idMenuDiario", conn);
```

Listing 8.3: Ejemplo de consulta SQL compleja en un repositorio

Esta consulta SQL más compleja permite obtener información detallada de varios elementos relacionados entre sí, como los menús diarios y sus respectivas recetas, mediante la unión de tablas `MenuDiario`, `Receta`, y otras relacionadas. Este tipo de consultas es fundamental para manejar casos de uso donde la información está distribuida entre múltiples tablas en la base de datos.

En conclusión, se crearon varios repositorios específicos, como MenuDiarioRepository, MenuSemanalRepository, ProductoRepository, RecetaRepository, y UsuarioRepository, para gestionar las operaciones de datos relacionadas con cada entidad del sistema. Estos repositorios encapsulan la lógica de acceso a datos y utilizan las consultas SQL necesarias para interactuar eficientemente con la base de datos.

8.4 Implementación de Controladores

En el contexto de nuestra aplicación, un controlador es un componente de software fundamental para gestionar las interacciones entre diferentes partes del sistema. Estos controladores actúan como intermediarios que reciben peticiones desde el *front-end* (en este caso, la aplicación Android), activan los procesos correspondientes en el *back-end* y devuelven los resultados al usuario [8]. Para manejar las diversas funcionalidades de la aplicación, los controladores reciben y procesan solicitudes HTTP de distintos tipos, como *GET*, *POST*, *PUT*, y *DELETE*.

Es importante mencionar que los *scrapers* utilizados en estos controladores fueron previamente desarrollados y detallados en el apartado numero 7, titulado Desarrollo, de este documento. Estos *scrapers* son fundamentales para extraer y procesar la información de diferentes fuentes, como productos de supermercados y recetas en línea.

A continuación, se describen brevemente los principales controladores implementados en el proyecto:

8.4.1. Controlador de *Scraper* para Productos del Supermercado

Este controlador se encarga de activar el *scraper* que extrae los productos de un supermercado específico. El proceso incluye la extracción de datos como el nombre del producto, precio, cantidad disponible, y otros detalles relevantes. Estos datos se almacenan en nuestra base de datos para ser utilizados posteriormente. Este controlador es crucial

para mantener la base de datos de productos actualizada, asegurando que la información disponible para los usuarios sea precisa y actual.

8.4.2. Controlador de *Scraper* para Asignación de Calorías

El segundo controlador tiene la responsabilidad de activar el *scraper* que asigna las calorías a los productos extraídos del supermercado. Este proceso es necesario porque la información nutricional no está siempre disponible en las páginas de los supermercados. El *scraper*, desarrollado y explicado en la Sección 7.1.5, busca en una fuente específica las calorías asociadas a cada producto y actualiza nuestra base de datos en consecuencia. Esta funcionalidad es esencial para proporcionar a los usuarios información nutricional completa y precisa.

8.4.3. Controlador para la Extracción de Recetas

Este controlador se encarga de activar el *scraper* responsable de extraer recetas desde una fuente en línea. El funcionamiento de este *scraper* se detalla en la Sección 7.1.6, donde se describe el proceso de captura de información como ingredientes, cantidades, pasos de preparación, y otras características relevantes de cada receta. Estos datos se almacenan en la base de datos, permitiendo que los usuarios accedan a una amplia variedad de recetas que se actualizan continuamente.

8.4.4. Controlador de Interacción con la Aplicación Android

El último controlador maneja todas las llamadas y peticiones que provienen de la aplicación Android. Se encarga de procesar las solicitudes de los usuarios, interactuar con la base de datos, y devolver la información necesaria al *front-end*. Este controlador es fundamental para la interacción en tiempo real entre el usuario y la aplicación.

En la siguiente sección, se explican en detalle las llamadas más importantes que se realizan desde la aplicación hacia estos controladores.

8.5 Controlador para el Manejo de Peticiones desde Android

El proceso de comunicación entre la aplicación Android y los controladores implica la serialización de datos. Los datos enviados en una solicitud deben ser convertidos a un formato estándar, como JSON, que pueda ser transmitido a través de la red. Una vez que estos datos llegan al servidor, se deserializan para reconstruir los objetos originales y permitir su procesamiento en el *back-end*. Este proceso se repite a la inversa cuando el servidor envía una respuesta al cliente. Es crucial que las clases que definen estos objetos sean idénticas en el *front-end* y el *back-end* para evitar errores en la comunicación y asegurar la correcta interpretación de los datos. Esto es particularmente delicado en el caso de objetos complejos, donde la serialización y deserialización deben manejarse con sumo cuidado para evitar inconsistencias o pérdida de información.

A continuación, se describen las peticiones más importantes gestionadas por el controlador que interactúa con la aplicación Android:

GET `/api/ControladorAndroid/GetAllRecetas`

Esta solicitud es esencial para la funcionalidad de la galería de recetas dentro de la aplicación. Al realizar una petición *GET* a esta ruta, se obtiene un listado completo de todas las recetas almacenadas en la base de datos. Este listado incluye detalles como los ingredientes, las instrucciones de preparación, y la información nutricional, que son deserializados y presentados al usuario en la interfaz de la aplicación.

GET `/api/ControladorAndroid/GetMenuSemanalPorId/{id}`

Este *endpoint* es clave para la generación del menú semanal personalizado para cada usuario. Al realizar una petición *GET* con el `id` del usuario, el controlador genera un menú semanal basado en las preferencias y necesidades del usuario, proceso que se explica en detalle en secciones anteriores de este documento. El menú generado es luego deserializado y presentado en la aplicación, permitiendo al usuario visualizar su plan de comidas para la semana.

PUT `/api/ControladorAndroid/ActualizarUsuario`

Esta ruta es utilizada cuando se necesita crear o actualizar la información de un usuario en la base de datos. Al recibir una solicitud *PUT*, los datos del usuario, previamente serializados en el *front-end*, son deserializados por el controlador y luego almacenados o actualizados en la base de datos. Este proceso es esencial para la gestión de perfiles de usuario, asegurando que la información esté siempre actualizada.

GET `/api/ControladorAndroid/GetUsuarioPorCorreo/{correo}`

Este endpoint trabaja en conjunto con la actualización del usuario. Cuando un usuario intenta iniciar sesión, la aplicación realiza una solicitud *GET* utilizando esta ruta, con el correo electrónico del usuario como parámetro. El controlador busca en la base de datos el perfil correspondiente y devuelve los datos del usuario, que son deserializados y utilizados para autenticar al usuario en la aplicación.

Estas operaciones permiten que la aplicación Android interactúe de manera eficiente y efectiva con el servidor, asegurando que los usuarios puedan acceder a las recetas, menús personalizados, y gestionar sus perfiles sin problemas. La correcta serialización y deserialización de los objetos es crucial para mantener la integridad y precisión de los datos transferidos entre el *front-end* y el *back-end*.

8.5.1. Estructura del Proyecto y Organización de Carpetas

La estructura del proyecto *BrocoleeServer* se organiza en una serie de carpetas que permiten una clara separación de las responsabilidades y funcionalidades del código. A continuación, se detalla el contenido y propósito de las principales carpetas que conforman el proyecto:

- **Capa Logica**: Esta carpeta contiene la lógica de negocio de la aplicación. Aquí se implementan las reglas y procesos que rigen cómo los datos se manejan y transforman antes de ser enviados al *front-end* o almacenados en la base de datos.
- ***Controllers***: En esta carpeta se encuentran los controladores que manejan las peticiones HTTP que llegan al servidor. Estos controladores se encargan de recibir

las solicitudes del cliente, interactuar con la capa lógica y devolver las respuestas apropiadas. Cada controlador está asociado a un conjunto de rutas específicas que definen los *endpoints* de la API.

- **Persistencia**: Esta carpeta agrupa todos los componentes relacionados con el acceso y la gestión de la base de datos. Incluye varias subcarpetas:
  - **Interfaces**: Contiene interfaces que definen los contratos para la implementación de los repositorios. Estas interfaces aseguran que los métodos para interactuar con la base de datos sean consistentes y estandarizados.
  - **Respository**: Aquí se implementan los repositorios, que son responsables de la interacción directa con la base de datos. Siguen los contratos definidos en las interfaces para ejecutar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre las entidades de la base de datos.
  - ***Scrappers***: Esta subcarpeta contiene los *scrapers*, que son scripts automatizados utilizados para extraer datos de fuentes externas, como páginas web de supermercados o sitios de recetas. Los *scrapers* juegan un papel esencial en la actualización de la base de datos con la información más reciente.
  - **SingletonConnection.cs**: Este archivo contiene la implementación del patrón de diseño Singleton, que se utiliza para garantizar que la conexión a la base de datos sea única y compartida a lo largo de toda la aplicación, mejorando la eficiencia y el manejo de recursos.
- ***Testing***: Esta carpeta está destinada a las pruebas de la aplicación. Aquí se incluyen los *tests* automatizados que verifican el correcto funcionamiento de los diferentes componentes del sistema, asegurando que cumplan con los requisitos establecidos y que se comporten de manera esperada en diferentes escenarios.
- ***Connected Services***: Esta carpeta se utiliza para agregar servicios conectados a la aplicación, como servicios web o referencias a otras APIs. Ayuda a integrar servicios externos dentro del proyecto de manera ordenada y modular.

Esta organización modular permite que el proyecto sea fácil de navegar y mantener, facilitando la colaboración entre desarrolladores y asegurando que cada parte del sistema esté claramente definida y estructurada.

CAPÍTULO 9
Testing

El proceso de *testing* es una fase crucial en el desarrollo de software, ya que garantiza que la aplicación funcione correctamente y cumpla con los requisitos establecidos. A través de pruebas sistemáticas y bien estructuradas, es posible identificar y corregir errores, mejorar la estabilidad del sistema y asegurar la calidad del producto final.

En este capítulo, se describen las herramientas y técnicas utilizadas para realizar las pruebas en el proyecto, abarcando desde pruebas unitarias que verifican la funcionalidad de componentes individuales, hasta pruebas de integración que aseguran que los diferentes módulos del sistema interactúen correctamente. También se detallan las pruebas de acceso a la base de datos, esenciales para validar la correcta manipulación de los datos almacenados.

Cada tipo de prueba ha sido diseñado e implementado para detectar posibles fallos en las primeras etapas del desarrollo, lo que ha permitido mejorar la eficiencia del proceso y reducir los riesgos de errores en el entorno de producción. A lo largo de este capítulo, se ilustran ejemplos específicos de pruebas y se discuten los resultados obtenidos, proporcionando una visión completa del enfoque de *testing* adoptado en este proyecto.

9.1 Herramientas utilizadas

En el proceso de *testing* de este proyecto, se han utilizado diversas herramientas para garantizar la calidad y la fiabilidad del código. Algunas de las herramientas más destacadas incluyen:

- **XUnit**: Es un *framework* para pruebas unitarias en .NET. Nos permite estructurar y ejecutar *tests* automatizados, facilitando la detección de errores en las primeras etapas del desarrollo.
- **MySQL**: La base de datos utilizada en el proyecto fue MySQL. Para las pruebas de acceso a la base de datos, se configuró una instancia de MySQL específica para *testing*, asegurando que las pruebas no interfieran con los datos de producción.

9.2 Pruebas unitarias

Las pruebas unitarias son un componente esencial en el desarrollo de software, ya que permiten verificar que cada clase y método en el sistema funcione según lo esperado de manera aislada. En este proyecto, se realizaron pruebas unitarias exhaustivas

para cada una de las clases principales, abarcando desde los modelos de datos hasta los controladores que gestionan la lógica de negocio.

El enfoque de las pruebas unitarias fue asegurar que:

- Los constructores inicialicen correctamente las propiedades de los objetos.
- Los métodos principales devuelvan los resultados esperados bajo diferentes condiciones.
- Los métodos *toString()* u otros métodos similares formateen las salidas de manera coherente y correcta.
- Los métodos de manipulación de datos (como agregar, actualizar o eliminar elementos) funcionen correctamente y mantengan la integridad de los datos.

A continuación, se presenta un ejemplo de prueba unitaria aplicada a la clase `Receta`, que es responsable de representar las recetas en la aplicación.

9.2.1. Ejemplo: Pruebas unitarias para la clase Receta

La clase `Receta` contiene varias propiedades clave, como el nombre de la receta, la descripción, el valor nutricional, el precio estimado, el tiempo de preparación, el tipo de comida, la imagen, la lista de ingredientes y las etiquetas asociadas. Es fundamental que esta clase funcione correctamente, ya que es central en la gestión y visualización de recetas dentro de la aplicación.

A continuación se describen algunas de las pruebas unitarias implementadas para esta clase:

```
1  public class RecetaTests
2  {
3      [Fact]
4      public void Constructor_ShouldInitializePropertiesCorrectly()
5      {
6          // Arrange
7          int expectedId = 1;
8          string expectedNombre = "Ensalada C sar";
9          string expectedDescripcion = "Ensalada con pollo, lechuga y aderezo
               C sar";
10         double expectedValorNutricional = 250.0;
11         double expectedPrecioEstimado = 5.99;
12         int expectedTiempo = 15;
13         string expectedTipoComida = "Comida";
14         string expectedImagen = "ensalada_cesar.png";
15         var expectedIngredientes = new List<IngredienteFormato>
16         {
17             new IngredienteFormato(1, 200, "gramos")
18         };
19         var expectedEtiquetas = new List<string> { "Saludable", "R pido" };
20
21         // Act
22         var receta = new Receta(expectedId, expectedNombre, expectedDescripcion
               ,
23             expectedValorNutricional, expectedPrecioEstimado, expectedTiempo,
24             expectedTipoComida, expectedImagen, expectedIngredientes,
                   expectedEtiquetas);
25
26         // Assert
27         Assert.Equal(expectedId, receta.Id);
```

```
28          Assert.Equal(expectedNombre, receta.Nombre);
29          Assert.Equal(expectedDescripcion, receta.Descripcion);
30          Assert.Equal(expectedValorNutricional, receta.ValorNutricional);
31          Assert.Equal(expectedPrecioEstimado, receta.PrecioEstimado);
32          Assert.Equal(expectedTiempo, receta.Tiempo);
33          Assert.Equal(expectedTipoComida, receta.TipoComida);
34          Assert.Equal(expectedImagen, receta.Imagen);
35          Assert.Equal(expectedIngredientes, receta.IngredientesEspecificos);
36          Assert.Equal(expectedEtiquetas, receta.Etiquetas);
37      }
38
39      [Fact]
40      public void ToString_ShouldReturnFormattedString()
41      {
42          // Arrange
43          var receta = new Receta(1, "Ensalada Cesar", "Ensalada con pollo,
                lechuga y aderezo Cesar",
44              250.0, 5.99, 15, "Comida", "ensalada_cesar.png",
45              new List<IngredienteFormato>
46              {
47                  new IngredienteFormato(1, 200, "gramos")
48              },
49              new List<string> { "Saludable", "Rpido" });
50
51          string expectedString = "Receta: Ensalada Cesar\n" +
52                                  "ID: 1\n" +
53                                  "Descripcin: Ensalada con pollo, lechuga y
                                        aderezo Csar\n" +
54                                  "Valor Nutricional: 250\n" +
55                                  "Tiempo: 15 minutos\n" +
56                                  "Tipo de Comida: Comida\n" +
57                                  "Ingredientes Especficos: 1: 200 gramos\n" +
58                                  "Etiquetas: Saludable, Rpido\n";
59
60          // Act
61          string result = receta.ToString();
62
63          // Assert
64          Assert.Equal(expectedString, result);
65      }
66
67
68  }
69 }
```

Listing 9.1: Pruebas unitarias para la clase Receta

Prueba del constructor

En la primera prueba, `Constructor_ShouldInitializePropertiesCorrectly`, se verifica que el constructor de la clase `Receta` inicialice todas las propiedades correctamente con los valores proporcionados. Se crean instancias de la clase `Receta` con valores específicos, y luego se compara cada propiedad de la instancia con los valores esperados utilizando aserciones.

Prueba del método `ToString`

La segunda prueba, `ToString_ShouldReturnFormattedString`, verifica que el método `ToString` devuelva una cadena formateada correctamente. Esta prueba es crucial para

asegurar que la información de la receta se presenta de manera coherente y legible cuando se convierte a texto.

9.3 Pruebas de integración

Las pruebas de integración son un tipo crucial de pruebas de software que verifican si los diferentes módulos del sistema interactúan correctamente cuando se combinan. A diferencia de las pruebas unitarias, que se centran en la funcionalidad aislada de clases o métodos individuales, las pruebas de integración buscan asegurar que los módulos del sistema funcionen juntos sin problemas y que el sistema completo se comporte como se espera cuando estos módulos están integrados.

En nuestro proyecto, se realizaron varias pruebas de integración para verificar la correcta interacción entre los componentes más críticos. A continuación, se describen algunas de las pruebas de integración clave que fueron implementadas:

9.3.1. Prueba de Integración para el Cálculo de Calorías

Esta prueba verifica la funcionalidad de cálculo de calorías, asegurando que los datos como productos del supermercado y sus respectivas calorías se integren y calculen correctamente. Se evalúa que los cálculos realizados sobre los datos extraídos sean precisos y reflejen la información esperada.

9.3.2. Prueba de Integración de Flujo Completo

Esta prueba simula un flujo completo de uso de la aplicación, desde la creación de un usuario hasta la generación de un menú semanal y la lista de compras correspondiente. Esta prueba garantiza que todas las partes del sistema trabajen en conjunto de manera coherente y sin errores, validando que los datos se transmitan correctamente entre los módulos, incluyendo la base de datos, los generadores de menús, y la interfaz de usuario.

9.3.3. Prueba de Integración para Generar Lista de Compras

La prueba de generación de lista de compras asegura que la funcionalidad que permite a los usuarios crear una lista de compras a partir de su menú semanal funcione correctamente. Se verifica que todos los ingredientes y productos necesarios sean extraídos de las recetas seleccionadas y presentados adecuadamente en una lista.

9.3.4. Prueba de Integración del Supermercado

Esta prueba asegura la correcta integración de la funcionalidad de extracción de productos de un supermercado en línea. Verifica que los productos se extraigan, procesen y almacenen adecuadamente en la base de datos, y que se integren sin problemas en las funciones relacionadas, como el cálculo de calorías o la generación de menús.

9.3.5. Prueba de Integración de Usuario y Menú

Esta prueba verifica la integración entre la funcionalidad de gestión de usuarios y la generación de menús personalizados. Se asegura de que los datos del usuario, como sus preferencias y objetivos nutricionales, se utilicen adecuadamente para crear un menú

semanal ajustado a sus necesidades, y que la información se gestione correctamente en la base de datos.

9.3.6. Ejemplo de Prueba de Integración de Flujo Completo

A continuación, se muestra un ejemplo de una prueba de integración que cubre un flujo completo dentro de la aplicación. Esta prueba valida desde la creación de un usuario hasta la generación de un menú semanal y la correspondiente lista de compras, verificando que todos los componentes trabajen juntos de manera coherente:

```
public class FlujoCompletoIntegrationTests
{

    [Fact]
    public void Usuario_CreaMenuSemanal_GeneraListaDeCompra_
    VerificaProductosEnSupermercado()
    {
        // Arrange
        var usuario = new Usuario("test@example.com", "Test User", "password123
            ","Fecha","Masculino",3.5,1);

        var ingredientesDesayuno = new List<IngredienteFormato>
        {
            new IngredienteFormato(1, 2, "unidades"), // Representa "Huevo"
            new IngredienteFormato(2, 1, "taza")       //Representa otro
                ingrediente
        };
        var ingredientesAlmuerzo = new List<IngredienteFormato>
        {
            new IngredienteFormato(3, 250, "gramos"), // Representa "Pollo"
            new IngredienteFormato(4, 1, "taza")       // Representa "Arroz"
        };
        var ingredientesCena = new List<IngredienteFormato>
        {
            new IngredienteFormato(5, 3, "unidades"), // Representa "Tomate"
            new IngredienteFormato(6, 2, "tazas")      // Representa "Lechuga"
        };

        var menuDiarioLunes = new MenuDiario(1, DiaDeLaSemana.LUNES,
            new Receta(1, "Desayuno", "Huevos revueltos", 300, 2.5, 15, "
                Desayuno","huevos.jpg",ingredientesDesayuno,new List<string>
                 {"Proteina"}),
            new Receta(2, "Almuerzo", "Pollo con arroz", 600, 5.0, 30, "
                Almuerzo", "pollo_arroz.jpg", ingredientesAlmuerzo, new List<
                string>{"Proteina","Carbohidratos"}),
            new Receta(3, "Cena", "Ensalada", 200, 3.0, 10, "Cena", "ensalada.
                jpg",ingredientesCena,new List<string>{"Vegetales"})
        );

        var menuSemanal = new MenuSemanal
        {
            Fecha = DateTime.Today,
            PlatosDiarios = new List<MenuDiario> { menuDiarioLunes }
        };

        var supermercado = new Brocolee Server.capa Logica.Supermercado("
            Mercadona",new List<Producto>
        {
            new Producto("Huevo", 0.10, 0.10, 155, 1, "Mercadona", "huevo.jpg",
                 "unidades",DateTime.Today),
            new Producto("Leche", 0.99, 0.89, 42.0, 1000.0, "Mercadona", "leche
                .png", "litros", DateTime.Today),
```

```
43              new Producto("Pollo", 5.00, 4.50, 250.0, 1000.0, "Mercadona", "
                    pollo.jpg","gramos", DateTime.Today),
44              new Producto("Arroz", 1.00, 0.90, 130.0, 1000.0, "Mercadona", "
                    arroz.jpg","gramos", DateTime.Today),
45              new Producto("Tomate",2.00,1.80, 18.0, 500.0, "Mercadona", "
                    tomate.jpg", "gramos", DateTime.Today),
46              new Producto("Lechuga",1.50,1.20, 15.0, 500.0, "Mercadona", "
                    lechuga.jpg","gramos", DateTime.Today)
47          });
48
49          // Act
50          var listaDeCompra = new ListaDeCompra(usuario, menuSemanal);
51          var productos = new List<Producto>();
52          foreach (var menuDiario in menuSemanal.PlatosDiarios)
53          {
54              foreach (var receta in new[]{menuDiario.RecetaDesayuno,
                    menuDiario.RecetaComida, menuDiario.RecetaCena})
55              {
56                  foreach (var ingrediente in receta.IngredientesEspecificos)
57                  {
58                      // Buscar el producto en el supermercado por nombre
59                      //(usamos un mapeo directo para este ejemplo)
60                      var nombreIngrediente = ingrediente.Id switch
61                      {
62                          1 => "Huevo",
63                          2 => "Leche",
64                          3 => "Pollo",
65                          4 => "Arroz",
66                          5 => "Tomate",
67                          6 => "Lechuga",
68                          _ => null
69                      };
70
71                      if (nombreIngrediente != null)
72                      {
73                          var producto = supermercado.Productos.Find(p => p.
                                NombreProducto == nombreIngrediente);
74                          if(producto != null)
75                          {
76                              productos.Add(producto);
77                          }
78                      }
79                  }
80              }
81          }
82
83          listaDeCompra.Productos = productos;
84          listaDeCompra.PrecioTotal = listaDeCompra.Productos.Sum(p => p.Precio);
85
86          // Assert
87          // Verifica que la lista de compra tiene todos los productos necesarios
88          Assert.Equal(6, listaDeCompra.Productos.Count);
89          // Verifica que cada producto necesario est en la lista de compra
90          Assert.Contains(supermercado.Productos[0], listaDeCompra.Productos);
91          // Verifica que el precio total se ha calculado
92          Assert.True(listaDeCompra.PrecioTotal > 0);
93      }
94  }
```

Listing 9.2: Ejemplo de Prueba de Integración de Flujo Completo

En este ejemplo, la prueba verifica el flujo completo de un usuario que crea un menú semanal, genera la lista de compra correspondiente y verifica que los productos necesa-

rios están disponibles en el supermercado seleccionado. La prueba garantiza que todos los pasos del flujo funcionan correctamente cuando se integran en el sistema.

9.4 Pruebas de acceso a la base de datos

Las pruebas de acceso a la base de datos son esenciales para verificar que las operaciones CRUD se realicen correctamente en las diferentes tablas de la base de datos. Estas pruebas aseguran que la persistencia de los datos funciona como se espera y que los datos almacenados y recuperados son consistentes y correctos.

En nuestro proyecto, se han implementado pruebas de acceso a la base de datos para todas las clases con repositorios. Estas clases incluyen `UsuarioRepository`, `ProductoRepository`, `RecetaRepository`, `MenuDiarioRepository` y `MenuSemanalRepository`. Cada una de estas clases interactúa con la base de datos para realizar operaciones específicas sobre los datos relacionados con los usuarios, productos, recetas, menús diarios y menús semanales.

Las pruebas de acceso a la base de datos permiten validar que cada repositorio funcione correctamente, garantizando que las operaciones de inserción, actualización, eliminación y obtención de registros se realicen sin errores. A continuación, se muestra un ejemplo de las pruebas realizadas para la clase `UsuarioRepository`:

```
public class UsuarioRepositoryTests
{
    private readonly UsuarioRepository _repository;

    public UsuarioRepositoryTests()
    {
        // Inicializar el repositorio
        _repository = new UsuarioRepository();
    }

    [Fact]
    public void Add_Usuario_ShouldAddUsuarioToDatabase()
    {
        // Arrange
        var usuario = new Usuario("test@example.com", "Test User", "password123
            ","1990-01-01","Masculino",1.2,1);

        // Act
        var result = _repository.Add(usuario);

        // Assert
        Assert.NotNull(result);
        Assert.Equal("test@example.com", result.Correo);
    }

    [Fact]
    public void GetByCorreo_ShouldReturnCorrectUsuario()
    {
        // Arrange
        var correo = "test@example.com";

        // Act
        var usuario = _repository.GetByCorreo(correo);

        // Assert
        Assert.NotNull(usuario);
        Assert.Equal(correo, usuario.Correo);
    }
```

```
38
39      [Fact]
40      public   void   Update_Usuario_ShouldUpdateUsuarioInDatabase ()
41      {
42          // Arrange
43          var usuario = new Usuario ("test@ example .com","Updated Name","
                newpassword123","1990-01-01","Masculino",1.5,1)
44          {
45              CaloriasObjetivas = 2500
46          };
47
48          // Act
49          var  updatedUsuario  = _repository.Update(usuario);
50
51          // Assert
52          Assert.NotNull(updatedUsuario);
53          Assert.Equal("Updated Name",updatedUsuario.Nombre);
54          Assert.Equal(2500,updatedUsuario.CaloriasObjetivas);
55      }
56
57      [Fact]
58      public   void   Remove_Usuario_ShouldRemoveUsuarioFromDatabase ()
59      {
60          // Arrange
61          var  usuario  =  new  Usuario("test@ example .com", "Test  User",  "password123
                ","1990-01-01","Masculino",1.2,1);
62
63          // Act
64          var  removed  =  _repository.Remove(usuario);
65
66          // Assert
67          Assert.True(removed);
68
69          var usuarioInDb = _repository.GetByCorreo("test@ example .com");
70          Assert.Null(usuarioInDb);
71      }
72
73      [Fact]
74      public   void   GetAll_Should Return All Usuarios ()
75      {
76          // Act
77          var  usuarios  =  _repository.GetAll();
78
79          // Assert
80          Assert.NotNull(usuarios);
81          Assert.NotEmpty(usuarios);
82      }
83  }
```

Listing 9.3: Ejemplo de Pruebas de Acceso a la Base de Datos para UsuarioRepository

En este ejemplo, las pruebas verifican que la clase `UsuarioRepository` pueda realizar correctamente las siguientes operaciones:

- **Add**: Añadir un nuevo usuario a la base de datos.
- **GetByCorreo**: Obtener un usuario de la base de datos utilizando su correo electrónico como clave.
- **Update**: Actualizar los detalles de un usuario en la base de datos.
- **Remove**: Eliminar un usuario de la base de datos.

- **GetAll**: Obtener una lista de todos los usuarios almacenados en la base de datos.

Estas pruebas garantizan que el repositorio maneje adecuadamente la persistencia de datos y que las operaciones se realicen de manera confiable en la base de datos, evitando errores y asegurando la integridad de los datos almacenados.

9.5 Resultados del *Testing*

El proceso de *testing* realizado durante el desarrollo del proyecto ha sido exhaustivo y multifacético, abarcando una variedad de pruebas que han permitido asegurar la calidad, estabilidad y fiabilidad de la aplicación. En esta sección se detallan los resultados obtenidos a partir de las pruebas unitarias, de integración y de acceso a la base de datos, así como el impacto de estos resultados en la evolución del proyecto.

9.5.1. Pruebas Unitarias

Las pruebas unitarias se centraron en verificar la funcionalidad de componentes individuales del sistema, como clases y métodos específicos, asegurando que cada uno operara correctamente en aislamiento. Los resultados obtenidos de estas pruebas son los siguientes:

- **Cobertura de Código Superior al 90 %**: Se alcanzó una cobertura de código superior al 90 %, lo que significa que la mayoría de las rutas de ejecución en las clases y métodos fueron probadas. Este alto nivel de cobertura es crucial para minimizar la posibilidad de errores no detectados, ya que asegura que se hayan probado tanto los casos de uso estándar como los escenarios borde y excepcionales.
- **Detección de Errores Tempranos**: Las pruebas unitarias permitieron identificar y corregir múltiples errores en etapas tempranas del desarrollo. Esto incluyó fallos en la inicialización de objetos, manejo incorrecto de excepciones y errores de lógica en métodos clave. La detección temprana de estos problemas evitó su propagación a fases más avanzadas, donde habrían sido más costosos de corregir, tanto en términos de tiempo como de recursos.
- **Estabilidad de la Funcionalidad Básica**: Las pruebas unitarias confirmaron la estabilidad y consistencia de las funcionalidades básicas de la aplicación. Por ejemplo, se garantizó que las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre las entidades clave como `Usuario`, `Producto`, y `Receta`, se realizaran de manera correcta y sin pérdida de datos. Este resultado es esencial para asegurar que la aplicación sea fiable y mantenga la integridad de los datos a lo largo del tiempo.

9.5.2. Pruebas de Integración

Las pruebas de integración verificaron que los diferentes módulos y componentes del sistema interactuaran correctamente cuando se combinaban, asegurando la cohesión y el funcionamiento del sistema en su conjunto. Los resultados obtenidos incluyen:

- **Verificación de Flujos Complejos**: Las pruebas de integración simularon flujos completos de uso de la aplicación, desde la creación de usuarios hasta la generación de menús y listas de compras. Se comprobó que todos los componentes del

sistema funcionaran de manera integrada y que los datos se transfirieran correctamente entre ellos, sin pérdida de información ni errores en la lógica de negocio. Esto asegura que los usuarios finales experimenten un flujo de trabajo coherente y libre de interrupciones.

- **Corrección de Inconsistencias en la Comunicación entre Módulos**: Durante las pruebas de integración, se detectaron y corrigieron inconsistencias en la comunicación entre la base de datos y los módulos de la aplicación. Por ejemplo, en algunos casos, las relaciones entre recetas y productos no se gestionaban correctamente, lo que llevó a ajustes en las consultas SQL y en la lógica de la aplicación para asegurar que los datos se integraran adecuadamente. Estas correcciones fueron esenciales para evitar errores que podrían haber afectado la experiencia del usuario y la integridad de los datos.

- **Validación de la Integridad de los Datos**: Las pruebas confirmaron que los datos permanecían consistentes y completos a lo largo de los flujos de integración, incluso en casos de múltiples operaciones concurrentes. Esto incluyó asegurar que los menús generados reflejaran correctamente las preferencias del usuario y que cualquier cambio en los ingredientes o productos se actualizara de manera coherente en todas las partes del sistema. Este resultado es fundamental para garantizar que la aplicación ofrezca resultados precisos y confiables.

- **Pruebas de Interoperabilidad**: Se realizaron pruebas para asegurar que la aplicación pudiera interactuar correctamente con servicios externos, como la extracción de datos de supermercados en línea. Estas pruebas verificaron que los productos fueran extraídos, procesados y almacenados correctamente en la base de datos, y que pudieran ser utilizados en funciones relacionadas, como el cálculo de calorías o la generación de menús. La correcta interoperabilidad con servicios externos amplía la funcionalidad de la aplicación y mejora su utilidad para los usuarios.

9.5.3. Pruebas de Acceso a la Base de Datos

Las pruebas de acceso a la base de datos se enfocaron en validar las operaciones de persistencia de datos, garantizando que las transacciones sobre las diferentes tablas se realizaran correctamente y que los datos almacenados fueran consistentes y accesibles según lo esperado. Los resultados obtenidos en esta fase fueron:

- **Consistencia de las Operaciones CRUD**: Se verificó que las operaciones CRUD en las tablas clave de la base de datos, como `Usuario`, `Producto`, `Receta`, y `MenuDiario`, se ejecutaran sin errores y mantuvieran la integridad de los datos. Se probaron diferentes escenarios, incluidos casos de manejo de errores, como intentos de insertar datos duplicados o eliminar registros con restricciones de claves foráneas. Esto asegura que la base de datos funcione de manera confiable y que los datos estén protegidos contra inconsistencias.

- **Optimización de Consultas SQL**: A lo largo de las pruebas, se identificaron algunas consultas SQL que inicialmente presentaban problemas de rendimiento o devolvían resultados incorrectos en ciertos casos. Estas consultas fueron optimizadas para mejorar la eficiencia del acceso a la base de datos y asegurar que todas las operaciones de búsqueda y filtrado se ejecutaran correctamente y en un tiempo razonable. La optimización de estas consultas es crucial para mantener un rendimiento adecuado en entornos de producción con grandes volúmenes de datos.

- **Pruebas de Integridad Referencial**: Se realizaron pruebas específicas para validar que las relaciones entre tablas, definidas mediante claves foráneas, se mantuvieran consistentes. Esto incluyó la validación de que al eliminar o actualizar un registro en una tabla, todas las tablas relacionadas reflejaran estos cambios de manera adecuada, evitando la creación de datos huérfanos o inconsistentes. Mantener la integridad referencial es esencial para la coherencia general de la base de datos.

- **Resiliencia ante Fallos**: Se probaron escenarios de fallo, como interrupciones en la conexión a la base de datos o intentos de acceso concurrentes, para evaluar la resiliencia del sistema. Los resultados mostraron que la aplicación era capaz de manejar estos fallos de manera adecuada, con mecanismos de recuperación que evitaban la corrupción de datos y mantenían la estabilidad del sistema. La resiliencia es fundamental para garantizar que el sistema pueda continuar operando incluso en condiciones adversas.

9.5.4. Impacto en la Calidad del Proyecto

Los resultados obtenidos a partir del proceso de *testing* han tenido un impacto significativo en la calidad final del proyecto. Algunos de los beneficios más destacados incluyen:

- **Reducción de Errores en Producción**: Gracias a la detección temprana y corrección de errores a través de pruebas unitarias e integración, se logró reducir significativamente la cantidad de errores presentes en la versión final de la aplicación. Esto ha contribuido a una experiencia de usuario más fluida y confiable, disminuyendo la necesidad de correcciones urgentes post-lanzamiento.

- **Mejora en el Rendimiento del Sistema**: La optimización de consultas SQL y la verificación de la eficiencia de las operaciones de base de datos resultaron en un sistema más rápido y con menor carga en los recursos del servidor. Esto es especialmente importante en un entorno de producción, donde el rendimiento es crítico para la satisfacción del usuario y la capacidad del sistema para manejar altos volúmenes de tráfico.

- **Aumento de la Confiabilidad y Estabilidad**: El exhaustivo proceso de pruebas aseguró que la aplicación es estable y confiable, incluso bajo condiciones de uso intensivo o en escenarios de fallo. Esto ha incrementado la confianza en el sistema y reducido la necesidad de intervenciones de mantenimiento de emergencia, lo que a su vez disminuye los costos operativos y mejora la disponibilidad del sistema.

- **Facilidad para la Escalabilidad**: La implementación de pruebas de integración y acceso a la base de datos ha permitido diseñar un sistema que es más fácil de escalar. Con una arquitectura bien probada y validada, futuras expansiones o modificaciones pueden ser implementadas con un riesgo mínimo de introducir nuevos errores o degradar el rendimiento. Esto es crucial para el crecimiento sostenido de la aplicación a medida que se agregan nuevas funcionalidades y se expande la base de usuarios.

- **Documentación y Aprendizaje**: El proceso de testing también contribuyó a la creación de una documentación más rica y precisa. Cada prueba implementada y cada error encontrado sirvieron como base para mejorar la documentación técnica del proyecto, lo que facilitará el trabajo de futuros desarrolladores que necesiten mantener o extender el sistema. Una buena documentación también acelera la incorpo-

ración de nuevos miembros al equipo de desarrollo, mejorando la productividad general del proyecto.

9.5.5. Conclusiones y Recomendaciones

El proceso de *testing* realizado ha demostrado ser un componente vital para el éxito del proyecto. Las pruebas han asegurado que el sistema sea robusto, eficiente y preparado para su uso en un entorno de producción. Sin embargo, es importante continuar con una estrategia de testing a medida que el sistema evoluciona, especialmente al agregar nuevas funcionalidades o realizar actualizaciones importantes.

Se recomienda:

- **Mantener y Ampliar la Cobertura de las Pruebas**: A medida que se agregan nuevas funcionalidades, es fundamental mantener y ampliar la cobertura de las pruebas para asegurar que todas las nuevas rutas de ejecución y casos de uso estén adecuadamente cubiertos.
- **Automatizar las Pruebas**: En la medida de lo posible, se recomienda automatizar las pruebas para asegurar una rápida detección de errores en futuros ciclos de desarrollo. La automatización permite ejecutar pruebas de regresión de manera eficiente, asegurando que las nuevas modificaciones no introduzcan errores en funcionalidades existentes.
- **Realizar Pruebas de Carga y Estrés**: Es importante llevar a cabo pruebas de carga y estrés adicionales para evaluar el comportamiento del sistema bajo condiciones extremas y asegurar que pueda manejar un aumento en el número de usuarios o en la complejidad de las operaciones. Esto es esencial para preparar el sistema para su uso en un entorno de producción con alta demanda.

En resumen, el enfoque de testing adoptado no solo ha garantizado un producto de alta calidad, sino que también ha establecido una base sólida para el crecimiento y mantenimiento futuro de la aplicación.

CAPÍTULO 10
Conclusiones y trabajo futuro

10.1 Conclusión

Este proyecto ha demostrado el inmenso valor que se puede obtener al combinar técnicas tradicionales de recolección de datos, como el *scraping*, con herramientas más avanzadas y emergentes, como la inteligencia artificial. A lo largo del desarrollo de la aplicación, los *scraper* se establecieron como componentes absolutamente esenciales, permitiendo la extracción automatizada y masiva de datos desde diversas fuentes web. Estas fuentes incluían, principalmente, supermercados como Mercadona y bases de datos de recetas culinarias.

El proceso de *scraping* demostró ser una tecnología sumamente eficaz para la recolección de datos. En este proyecto, fue posible obtener más de 4500 productos del supermercado Mercadona, lo que constituye un logro significativo en términos de volumen de datos recolectados. Sin embargo, es crucial destacar que este proceso no estuvo exento de desafíos. Uno de los principales retos encontrados fue la lentitud del proceso. La extracción completa de todos los productos requirió aproximadamente 8 horas, un tiempo considerable que se explica, en gran parte, por la necesidad de que los *scraper* simulen un comportamiento humano para evitar bloqueos por parte de los sitios web. Este tiempo de procesamiento se duplicó cuando se añadió la tarea de asignar valores nutricionales a los productos mediante otro *scraper*. Esta adición, aunque necesaria para completar los datos de cada producto, incrementó la carga de trabajo y extendió el tiempo total requerido para completar el proceso de recolección de datos. A pesar de la duración prolongada, se logró cumplir con el objetivo final de procesar y almacenar todos los productos en la base de datos, lo que estableció una base sólida y confiable para la aplicación.

En contraste, el *scraper* diseñado para la extracción de recetas mostró un rendimiento considerablemente más eficiente. Esto se debió, en gran medida, a que la fuente de datos utilizada poseía una estructura HTML bien organizada, lo que facilitó enormemente la extracción de todos los componentes necesarios para completar la clase Receta. Este proceso de extracción fue significativamente más rápido, y de no haber sido por la necesidad de utilizar inteligencia artificial para vincular y procesar los ingredientes de las recetas con los productos del supermercado, se podrían haber extraído cientos de recetas en apenas unos minutos. Al final, se lograron extraer alrededor de 70 recetas en un par de horas, un volumen que resultó ser más que suficiente para construir los menús semanales dentro de la aplicación.

La integración de la inteligencia artificial en este proyecto representó un avance significativo y una apuesta ambiciosa para mejorar los procesos automatizados. La esperanza original era que la IA pudiera unificar todos los datos relevantes de las recetas y los productos del mercado en un único objeto que pudiera ser utilizado de manera directa por

la aplicación. Sin embargo, debido a las limitaciones de tiempo y recursos, este objetivo no pudo ser alcanzado en su totalidad. A pesar de ello, la IA logró cumplir un rol crucial al ofrecer un formato de datos mucho más fácil de procesar para la aplicación. La inteligencia artificial fue especialmente útil para normalizar y estandarizar los ingredientes escritos por los usuarios en la página web de origen, lo que facilitó de manera significativa su integración con los productos del supermercado. Este logro es particularmente notable, ya que evitar la creación manual de un mapeo para todas las posibles variaciones en la escritura de un ingrediente habría sido una tarea extremadamente ardua y propensa a errores. Aunque la automatización completa de la unificación de datos quedó fuera del alcance de este proyecto, el uso de la IA fue de gran importancia y resultó ser un factor fundamental para el desarrollo exitoso del sistema.

Además de los aspectos técnicos, este proyecto subraya la importancia crucial de una buena organización y planificación en el desarrollo de software. Desde la concepción inicial de la idea hasta la implementación final de la aplicación, cada etapa del proceso fue fundamental para asegurar el éxito del proyecto. La complejidad de algunos de los objetos creados, como el Menú Semanal, no solo residía en su construcción lógica y funcional, sino también en las diversas etapas que implicaban su almacenamiento en la base de datos, su serialización en formato JSON, la transmisión segura de estos datos a través de la web y su deserialización correcta en la interfaz de usuario. Estos pasos fueron críticos para garantizar que el objeto mantuviera su integridad y funcionalidad a lo largo de todo el proceso. La correcta estructuración en capas, donde cada componente del proyecto estaba claramente separado y definido, fue un elemento clave que facilitó la identificación y resolución de errores durante todo el desarrollo. Esta organización meticulosa permitió gestionar la complejidad de manera eficiente, lo que resultó en un desarrollo más fluido y en la creación de una aplicación final que es sólida, confiable y capaz de cumplir con los objetivos planteados desde el inicio.

A partir del análisis comparativo con las aplicaciones Nutrilio, Fitatu, Fitia y Unimeal, nuestra aplicación presenta varias ventajas competitivas y también áreas de oportunidad. Mientras que aplicaciones como Unimeal ofrecen un conjunto robusto de funcionalidades, nuestra propuesta sobresale en la planificación automática de menús semanales y en la integración directa con productos de supermercados a través de técnicas de *scraping*, lo que no es ofrecido por la mayoría de las aplicaciones analizadas. A diferencia de Unimeal, que requiere una suscripción paga para acceder a la mayoría de sus funcionalidades avanzadas, nuestra aplicación busca ofrecer acceso a una mayor cantidad de herramientas sin costo adicional, lo cual puede mejorar su aceptación en el mercado. Además, si bien Fitatu destaca por su compatibilidad con múltiples plataformas, nuestra aplicación está diseñada para aprovechar al máximo la personalización de menús y la integración con bases de datos nutricionales en tiempo real, lo que nos diferencia de competidores como Fitia, que se centra más en sugerencias inteligentes basadas en preferencias. Esta combinación única de recolección automatizada de datos, personalización y accesibilidad posiciona a nuestra aplicación como una alternativa innovadora y altamente adaptable en el sector de la nutrición.

Asimismo los aspectos técnicos y organizativos, en este proyecto ha sido una aplicación práctica directa de muchos conceptos y técnicas aprendidos durante mi formación universitaria. Los principios de diseño modular, la importancia de la separación de responsabilidades en la arquitectura de software, y las técnicas de manipulación de bases de datos y gestión de grandes volúmenes de datos, fueron fundamentales para el éxito de este proyecto. Asignaturas como Estructuras de Datos y Algoritmos, Proyecto de Ingeniería de Software, Bases de Datos y Integración e interoperabilidad proporcionaron una base teórica sólida que me permitió enfrentar los desafíos técnicos que surgieron durante el desarrollo. En particular, la implementación de técnicas de *web scraping*, así como la

utilización de tecnologías como REST APIs y ORM (Object-Relational Mapping), reflejan directamente los conocimientos adquiridos en mis clases. Este proyecto no solo me permitió consolidar y aplicar esos conocimientos en un entorno real, sino también ver cómo las teorías y técnicas estudiadas en un aula pueden traducirse en soluciones efectivas y eficientes en el desarrollo de aplicaciones complejas.

En resumen, este proyecto no solo alcanzó sus metas técnicas, sino que también ofreció lecciones valiosas sobre la integración de tecnologías avanzadas y la importancia de una planificación cuidadosa en el desarrollo de software complejo. Gracias a la combinación de *scraping*, inteligencia artificial y una estructura de proyecto bien organizada, se logró desarrollar una aplicación que no solo cumple con los requisitos, sino que también tiene el potencial de escalar y adaptarse a futuras necesidades y desafíos tecnológicos.

10.2 Trabajo Futuro

Aunque el proyecto ha alcanzado muchos de los objetivos planteados inicialmente, existen varias áreas en las que se podría expandir y mejorar en el futuro. A continuación, se presentan algunas ideas para trabajos futuros que podrían enriquecer y ampliar las funcionalidades de la aplicación:

- **Incorporación de productos de diferentes supermercados:** Actualmente, la aplicación se centra en productos de un único supermercado, Mercadona. Un paso importante sería expandir la base de datos para incluir productos de otros supermercados. Esto requeriría el desarrollo e implementación de nuevos *scraper* específicos para cada supermercado, permitiendo así una comparación de precios y opciones más amplia para los usuarios.
- **Implementación completa del objetivo inicial con inteligencia artificial:** Uno de los objetivos originales del proyecto era utilizar inteligencia artificial para procesar recetas de cualquier fuente y unificar estos datos con los productos disponibles en la base de datos de supermercados. A futuro, se podría mejorar y entrenar la IA para alcanzar este objetivo, logrando una mayor flexibilidad y precisión en el manejo de recetas provenientes de múltiples fuentes y en formatos variados.
- **Funcionalidad para que los usuarios añadan sus propias recetas:** Una característica valiosa que podría añadirse es la capacidad para que los usuarios puedan introducir y almacenar sus propias recetas en la aplicación. Esto no solo aumentaría la personalización de la experiencia del usuario, sino que también permitiría a la comunidad compartir recetas, enriquecer la base de datos y crear una interacción más dinámica dentro de la aplicación.
- **Servidor y API activos de manera continua:** Actualmente, la aplicación requiere intervención manual para la activación de los *scraper*. Un paso futuro sería la implementación de un servidor con la API activa en todo momento, lo que permitiría automatizar completamente la extracción de productos de los supermercados. Esto podría programarse para que ocurriera semanalmente durante horarios nocturnos, minimizando el impacto en la disponibilidad del servicio y asegurando que la base de datos de productos esté siempre actualizada.
- **Inclusión de información sobre alérgenos:** Una mejora significativa sería la incorporación de información sobre alérgenos en los productos y recetas. Esto permitiría que el algoritmo de creación de menús semanales considere las alergias y restricciones alimenticias de los usuarios, personalizando aún más las recomendaciones y aumentando la seguridad y satisfacción del usuario.

- **Expansión de opciones dietéticas:** Actualmente, el sistema está diseñado para generar un menú basado en una dieta balanceada. Sin embargo, se podrían incluir más opciones dietéticas, como dietas altas en proteínas, vegetarianas, veganas o bajas en carbohidratos. Esto ampliaría el alcance de la aplicación a un público más diverso y permitiría a los usuarios ajustar sus menús a sus necesidades y preferencias alimenticias específicas.

Implementar estas mejoras y expansiones proporcionaría una experiencia de usuario mucho más completa y personalizada, además de mejorar la robustez y funcionalidad de la aplicación. Estas propuestas de trabajo futuro representan una oportunidad significativa para continuar desarrollando y refinando la aplicación, asegurando que siga siendo relevante y útil en un mercado en constante evolución.

Bibliografía

[1] Fitia - Pierde o Gana Peso con Un Plan Inteligente — fitia.app. https://fitia.app/es/. [Accessed 21-08-2024].

[2] Nutrilio - Food Journal, Water & Weight Tracking — nutrilio.net. https://nutrilio.net/. [Accessed 21-08-2024].

[3] Salud - Desarrollo Sostenible — un.org. https://www.un.org/sustainabledevelopment/es/health/. [Accessed 20-08-2024].

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

[5] Weight loss management app — Unimeal — unimeal.com. https://unimeal.com/. [Accessed 21-08-2024].

[6] Ballesteros-Rosillo, R. Gestión personal de dietas saludables (ii): Interfaz de usuario y gestión de usuarios y listas de compra. Trabajo de Fin de Grado, Universidad de Politécnica de Valencia, Valencia, 2024.

[7] Developers, G. Puppeteer - headless chrome node.js api, 2024. Accessed: 2024-08-21.

[8] Doe, J., and Smith, J. *Design Patterns for Software Controllers*, 2nd ed. TechPress, New York, USA, 2020.

[9] FatSecret. Calorías y nutrición - fatsecret, 2024. Accedido: 2024-07-23.

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

[11] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[12] Gelbukh, A. Procesamiento de lenguaje natural y sus aplicaciones, 2013. Accedido: 2024-07-30.

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

[14] Hanasi, M. Distancia de levenshtein, 2023. Accedido: 2024-07-23.

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

[16] López, J. Web scraping, 2018. Accedido: 2024-07-30.

[17] MERCADONA. Términos y condiciones. https://info.mercadona.es/es/terminos-y-condiciones, 2022. Mercadona, (s. f.).

[18] MERCADONA. Tienda online de mercadona, 2024. Accedido: 2024-07-22.

[19] OPENAI. Api de chatgpt. https://platform.openai.com/docs/api-reference/chat. Accedido: 2024-08-26.

[20] OPENAI. Chatgpt, 2024. Accedido: 2024-08-21.

[21] REGLAMENTO (UE) 2016/679 DEL PARLAMENTO EUROPEO Y DEL CONSEJO. Relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la directiva 95/46/ce (reglamento general de protección de datos), 2016. Accedido: 2024-07-30.

[22] RICHARDSON, L. Beautiful soup documentation, 2024. Accedido: 2024-07-20.

[23] SCHWABER, K., AND SUTHERLAND, J. La guía de scrum. *Scrumguides. Org 1* (2013), 21.

[24] SCRAPY. Scrapy - a fast and powerful scraping and web crawling framework, 2024. Accessed: 2024-08-21.

[25] SELENIUMHQ. Selenium - web browser automation, 2024. Accessed: 2024-08-21.

[26] (W3C), W. W. W. C. Css: Cascading style sheets, 2024. Accessed: 2024-07-21.

[27] (W3C), W. W. W. C. Xml path language (xpath), 2024. Accessed: 2024-07-21.

APÉNDICE A
Casos de uso

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	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.13: Caso de uso: Gestionar usuario

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.14: 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.15: 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.16: 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.17: 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.18: 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.19: 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.20: 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.21: 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.22: Caso de uso: Ver perfil 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.23: Caso de uso: Ver receta

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.24: 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.25: Caso de uso: Ver menú semanal

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 con los ODS más relevantes

Los Objetivos de Desarrollo Sostenible (ODS) son una serie de 17 metas interconectadas establecidas por las Naciones Unidas en 2015, diseñadas para abordar los desafíos globales más apremiantes, incluyendo aquellos relacionados con la pobreza, la desigualdad, el cambio climático, la degradación ambiental, la paz y la justicia [3]. Estas metas forman un marco para el desarrollo sostenible, orientado a mejorar el bienestar humano y asegurar la prosperidad del planeta para las generaciones presentes y futuras.

La aplicación desarrollada en este TFG está diseñada para facilitar un estilo de vida más saludable mediante el enfoque en la nutrición. Como se describe en la Tabla B.1, este software contribuye tanto directa como indirectamente al cumplimiento de varios ODS. Los objetivos que se ven más significativamente beneficiados por este proyecto incluyen:

- **ODS 3. Salud y bienestar:** Este objetivo se centra en garantizar una vida saludable y promover el bienestar para todos, sin importar la edad. El proyecto se alinea estrechamente con este ODS, ya que la aplicación ofrece menús personalizados que promueven una dieta equilibrada y un peso saludable, lo cual ayuda a prevenir enfermedades crónicas relacionadas con la alimentación, como la obesidad, la diabetes y las enfermedades cardiovasculares.
- **ODS 12. Producción y consumo responsables:** Este objetivo busca fomentar patrones sostenibles de consumo y producción. La aplicación apoya este ODS mediante la generación automática de listas de compras basadas en el menú semanal, lo que ayuda a los usuarios a organizar mejor sus compras, promoviendo un consumo consciente y reduciendo el desperdicio de alimentos.
- **ODS 2. Hambre cero:** Al facilitar el acceso a una alimentación saludable y asequible, esta aplicación puede contribuir a reducir la pobreza. Ofreciendo menús semanales que optimizan el uso de recursos y minimizan el desperdicio, la aplicación ayuda a los usuarios a manejar mejor sus presupuestos alimentarios, algo crucial para quienes tienen recursos limitados. Esto no solo permite una planificación eficiente de las comidas, sino que también reduce los costos asociados con problemas de salud derivados de una mala alimentación.
- **ODS 8. Trabajo decente y crecimiento económico:** El desarrollo de esta aplicación crea oportunidades de empleo en el sector tecnológico, involucrando a desarrolladores, diseñadores y especialistas en soporte técnico. A medida que la aplicación crece, podría necesitar la colaboración de expertos en nutrición y salud, además de abrir nuevas oportunidades de negocio, como servicios de asesoría personalizada o alianzas con proveedores de alimentos, contribuyendo así al crecimiento económico y a la creación de empleo.