Facultad de Ciencias

Asistente nutricional personalizado con Inteligencia Artificial Generativa

Personalized nutritional assistant with Generative Artificial Intelligence

Trabajo de Fin de Grado
para acceder al
Grado en Ingeniería Informática

Autor: Arahí Fernández Monagas
Directora: Cristina Tirnauca
Co-Director: Miguel Viguera del Pozo
Convocatoria: Julio de 2024

Índice general

1. Introducción **1**
1.1. Tecnologías y herramientas utilizadas . . . 2

2. Diseño conceptual **4**
2.1. Requisitos funcionales . . . 4
2.2. Requisitos no funcionales . . . 7
2.3. *Mockup* inicial . . . 9
2.3.1. Registro . . . 9
2.3.2. Inicio de sesión . . . 9
2.3.3. Ficha . . . 10
2.3.4. Objetivo . . . 11
2.3.5. Resumen menú . . . 11
2.3.6. Menú principal . . . 12

3. Diseño e implementación **13**
3.1. Diseño del *backend* . . . 17
3.2. Implementación *backend* . . . 18
3.2.1. Autenticación de usuarios . . . 18
3.2.2. Almacenamiento y gestión de datos . . . 19
3.3. Implementación *fronted* . . . 20
3.3.1. Conexión con la base de datos . . . 22
3.4. Módulo *text_processing* . . . 24
3.5. Creación de la API web . . . 26
3.5.1. Prompting . . . 27
3.6. Pruebas . . . 27
3.6.1. Pruebas unitarias . . . 28
3.6.2. Pruebas de aceptación . . . 29
3.6.3. Pruebas de rendimiento . . . 29
3.6.4. Pruebas de accesibilidad . . . 30

4. Desarrollo de funcionalidades con IA Generativa **32**
4.1. Fundamentos teóricos . . . 33
4.1.1. RAG . . . 33
4.1.2. *Chunking* . . . 35
4.1.3. *Embeddings* . . . 37
4.2. Clasificación de imágenes . . . 39
4.2.1. Fundamentos teóricos . . . 39

5. Conclusiones y trabajos futuros **43**

Índice de cuadros

2.1. Registro de usuario 4
2.2. Análisis de datos 5
2.3. Generación de dietas personalizadas 5
2.4. Planificación de comidas 5
2.5. Cuadro de requisitos funcionales - parte 1 6
2.6. Cuadro de requisitos funcionales - parte 2 7
2.7. Cuadro de requisitos no funcionales 8

3.1. Pautas de accesibilidad WCAG 22
3.2. Resultados de las pruebas de aceptación 29

4.1. Categorías 38

Índice de figuras

3.1. Arquitectura del sistema . . . 14
3.2. Estructura inicial de la solicitud . . . 14
3.3. *Prompt* modelo gpt3.5 . . . 15
3.4. Fichero JSON con el menú semanal . . . 16
3.5. Datos de un usuario . . . 16
3.6. Diagrama de la arquitectura de React Native . . . 21
3.7. Archivo de configuración de Firebase en JSON . . . 23
3.8. Paella de mariscos . . . 25
3.9. Pasta salsa pesto y queso parmesano . . . 25
3.10. Pruebas unitarias componente *RegisterUser* . . . 28
3.11. Prueba de rendimiento . . . 30
3.12. WCAG menú semanal . . . 31
3.13. WCAG selección meta . . . 31
3.14. WCAG inicio sesión . . . 31
3.15. WCAG menú semanal (solucionado) . . . 31
3.16. WCAG selección meta (solucionado) . . . 31
3.17. WCAG inicio sesión (solucionado) . . . 31

4.1. Proceso modelo RAG . . . 34
4.2. Modelo generativo . . . 34
4.3. Chunking con overlap . . . 37
4.4. Visualización de vectores de recetas usando algoritmo de reducción de dimensionalidad . . . 38
4.5. Arquitectura transformer [5] . . . 40
4.6. Matriz de píxeles [3] . . . 41
4.7. Funcionamiento red neuronal convolucional . . . 42

5.1. Inicio de sesión . . . 45
5.2. Registro . . . 46
5.3. Ficha . . . 47
5.4. Selección de metas . . . 48
5.5. Menú semanal . . . 49
5.6. Ficha personal . . . 50
5.7. Pizza parmesana con rúcula . . . 51
5.8. Hamburguesa de ternera con queso cheddar y patatas fritas . . . 51
5.9. Pops de pollo, patatas fritas y bolitas de queso . . . 52
5.10. Nachos con guacamole y crema salvadoreña . . . 52
5.11. Espárragos a la vinagreta y mayonesa . . . 53
5.12. Pollo frito con pan y patatas fritas . . . 53

Agradecimientos

En primer lugar, quiero agradecerle todo el apoyo que he recibido por parte de mis padres, en especial a mi madre, que desde muy pequeña me ha guiado y apoyado en todas mis decisiones.

Agradecerles a mis amigos que llegaron a mi vida hace años y que a día de hoy son una parte importante de mi: Elena, Alejandro, Sofía, Lara, Pedro, Inés y Alexandra os llevo siempre en mis pensamientos.

Quiero agradecerle a LKS Next por acogerme durante mis prácticas y brindarme todas las herramientas y recursos para poder desarrollar el proyecto. Y a mis compis de trabajo que han hecho que tenga una estancia acogedora en la empresa: Raquel, Óscar, Ricardo, Carla, Rubén y Julio.

También quiero expresar mi agradecimiento a mi tutora Cristina, que además de guiarme e involucrarse durante el proyecto ha sido muy buena profesora durante mis dos últimos años de carrera.

Por último, quiero guardar estas líneas a mi abuelo Andrés, que no ha habido ni un solo día en el que no haya dejado de pensar en él. Siempre he llevado conmigo la frase que me decías de pequeña: *"Esta niña será alguien importante algún dia"*. Hoy, a punto de graduarme como ingeniera informática, puedo afirmar con gratitud que su frase ha sido mi mayor impulso en los momentos más difíciles.

Resumen

La alimentación es una necesidad básica para el ser humano y es un pilar fundamental tanto para el bienestar físico como para el mental. Adoptar una dieta adecuada puede ser una tarea compleja especialmente por la falta de tiempo y otras necesidades individuales de cada persona. Esto afecta considerablemente a la alimentación de las personas, llegando a no ser correcta en la mayoría de los casos.

En respuesta a este desafío, el proyecto propone una solución innovadora que utiliza la Inteligencia Artificial Generativa para ofrecer un sistema nutricional personalizado y efectivo, permitiendo al usuario una experiencia eficiente para la gestión de su dieta y alcanzar sus objetivos de salud y bienestar.

La aplicación desarrollada como parte de este proyecto ofrece diversas funcionalidades clave:

- **Menús personalizados:** generación de menús adaptados a las necesidades, preferencias y objetivos del usuario
- **Reconocimiento de imágenes:** identificación de calorías en alimentos mediante escaneo, pudiendo llevar un mayor control y seguimiento.
- **Lista de la compra:** generación automática de una lista de compra con los alimentos necesarios y sus cantidades recomendadas.
- **Sistema de favoritos:** permite al usuario marcar platos que le gustan o no, permitiendo una mayor personalización en la generación de próximos menús.
- **Adaptación dinámica:** el sistema ajusta las recomendaciones según las interacciones del usuario.

Para implementar estas funcionalidades, se utilizan algoritmos de Inteligencia Artificial (IA) avanzados, como *Large Language Models* (LLMs), para comprender las preferencias del usuario, *embeddings* para representar la información nutricional, bases de datos vectoriales para almacenar información sobre alimentos y recetas, y, por último, *Retrieval Augmented Generation* (RAG), para generar los menús personalizados.

Esta solución ofrece una experiencia personalizada y efectiva para la gestión de la dieta, permitiendo a los usuarios planificar comidas, controlar la ingesta calórica y alcanzar sus objetivos. La aplicación, llamada Salvia, combina Inteligencia Artificial (IA) de última generación con un enfoque personalizado para la gestión de la dieta, lo que la diferencia de las soluciones existentes en el mercado.

Palabras clave

IA Generativa, nutrición de precisión, procesamiento de lenguaje natural, modelo RAG, *embeddings*.

Abstract

Eating is a basic necessity for humans and a fundamental pillar for both physical and mental well-being. Adopting an adequate diet can be a complex task, especially due to a lack of time and individual needs. This considerably affects people's eating habits, often resulting in incorrect nutrition.

In response to this challenge, the project proposes an innovative solution that utilizes generative artificial intelligence to offer a personalized and effective nutritional system, allowing users an efficient experience for managing their diet and achieving their health and wellness goals.

The application developed as part of this project offers several key functionalities:

- **Personalized menus:** generation of menus adapted to the user's needs, preferences, and goals.
- **Image recognition:** identification of calories in foods through scanning, allowing for better control and monitoring.
- **Shopping list:** automatic generation of a shopping list with the necessary foods and their recommended quantities.
- **Favorites system:** allows the user to mark dishes they like or dislike, enabling greater personalization in generating future menus.
- **Dynamic adaptation:** the system adjusts recommendations based on user interactions.

To implement these functionalities, advanced Artificial Intelligence (AI) algorithms are used, such as *Large Language Models* (LLMs) to understand user preferences, *embeddings* to represent nutritional information, vector databases to store information about foods and recipes, and *Retrieval Augmented Generation* (RAG) to generate personalized menus.

This solution offers a personalized and effective experience for diet management, allowing users to plan meals, control calorie intake, and achieve their goals. The application, called Salvia, combines state-of-the-art Artificial Intelligence (AI) with a personalized approach to diet management, setting it apart from existing solutions in the market.

Keywords

Generative AI, precision nutrition, natural language processing, RAG model, *embeddings*.

Capítulo 1

Introducción

El presente trabajo fin de grado (TFG) se centra en el desarrollo de un prototipo de **asistente nutricional personalizado con Inteligencia Artificial (IA) generativa**. El objetivo principal es ofrecer una herramienta innovadora que ayude a los usuarios a mejorar sus hábitos alimenticios de una forma personalizada y adaptada a sus necesidades específicas.

A lo largo del documento, se reflejarán en detalle los pasos que se han seguido para la captura de requisitos, el diseño funcional de la aplicación y se expondrá el desarrollo de todas aquellas funcionalidades que emplean Inteligencia Artificial (IA) Generativa.

La alimentación, como necesidad básica del ser humano, va más allá de la simple supervivencia, ya que la evidencia científica demuestra su impacto en la salud mental. Los alimentos que consumimos influyen directamente en la producción de neurotransmisores, moléculas que regulan nuestro estado de ánimo y bienestar emocional.

Los Objetivos de Desarrollo Sostenible (ODS) ponen de manifiesto la malnutrición y la falta de acceso a alimentos saludables como problemas que afectan a millones de personas en el mundo. A esto se suma que las personas con enfermedades mentales pueden tener necesidades nutricionales especiales, haciendo aún más crucial el acceso a una alimentación adecuada.

Consumir alimentos procesados, ricos en grasas saturadas, azúcares y conservantes artificiales, genera inflamación y estrés oxidativo en el cuerpo, afectando negativamente al cerebro y al estado de ánimo. Esto puede derivar en problemas como depresión, ansiedad y fatiga. Diversos estudios avalan la relación entre alimentación y salud mental:

- **Organización mundial de la salud:** la mala nutrición puede contribuir a la aparición de trastornos mentales[1].
- **Harvard Health Publishing[2]:** una dieta saludable con frutas, verduras, granos enteros, proteínas magras y grasas saludables mejora la salud mental.

La relevancia actual del tema impulsa a presentar una solución innovadora: una aplicación que combina tecnología y nutrición para facilitar la correcta alimentación de las personas, mejorando su estado físico y, como consecuencia, su estado mental. Esta herramienta busca:

[1] World Health Organization: WHO. "Trastornos Mentales". Who.int, 8 de junio de 2022, www.who.int/es/news-room/fact-sheets/detail/mental-disorders. Accedido 19 de octubre de 2023.

[2] Selhub, Eva. "Nutritional Psychiatry: Your Brain on Food". Harvard Health Blog, Harvard Health Publishing, 18 de septiembre de 2022, www.health.harvard.edu/blog/nutritional-psychiatry-your-brain-on-food-201511168626. Accedido 19 de octubre 2023.

- **Facilitar el acceso a información y recursos:** guiar a los usuarios hacia una alimentación adecuada y equilibrada para mejorar su salud mental.
- **Personalización:** adaptarse a las necesidades y objetivos específicos de cada usuario.
- **Acompañamiento continuo:** ofrecer apoyo y seguimiento a lo largo del proceso.

1.1. Tecnologías y herramientas utilizadas

Para el desarrollo del proyecto se han utilizado distintas tecnologías y herramientas que han facilitado el trabajo. El objetivo de esta sección es explicar en detalle la decisión de utilización de estas tecnologías.

- **Python:** se ha elegido como lenguaje de programación principal por ser una herramienta poderosa, versátil y fácil de usar. Su sintaxis es clara y sencilla, y presenta una amplia biblioteca de funciones y herramientas que ha permitido abordar el proyecto.
- **JavaScript:** se ha utilizado como lenguaje de programación para React Native con el que se definirá la lógica de la aplicación.
- **CSS:** se ha elegido como lenguaje de estilos para definir la apariencia de los elementos de la interfaz de usuario de la aplicación.
- **Jupyter Notebook:** se ha utilizado Jupyter Notebook para escribir y ejecutar el código. Jupyter Notebook es una herramienta interactiva que permite combinar código, texto y visualizaciones en un mismo documento.
- **React Native:** se trata de un framework de código abierto para crear aplicaciones web nativas usando JavaScript.
- **Langchain:** es un framework de Python diseñado específicamente para desarrollar aplicaciones de inteligencia artificial conversacional. Proporciona herramientas y funcionalidades para gestionar el flujo de información, agilizar las interacciones entre diferentes componentes y construir agentes inteligentes.
- **FastApi:** es un framework web de alto rendimiento para Python que ofrece un enfoque simplificado para construir una *Application Programming Interface* (API[3]). Cuenta con una sintaxis limpia, inyección automática de dependencias y soporte integrado para operaciones asíncronas.
- **Jest**: es un marco de pruebas JavaScript moderno y popular, creado por Facebook, que facilita la creación, ejecución y organización de pruebas unitarias y de integración para aplicaciones JavaScript.
- **Testing Library:** es una biblioteca para escribir pruebas unitarias y de integración en aplicaciones JavaScript, especialmente en frameworks como React.
- **WAVE (Web Accessibility Evaluation Tool)**: es una herramienta en línea que analiza aplicaciones web para identificar problemas de accesibilidad significativos y detallar soluciones.

[3]Una API es un conjunto de definiciones y protocolos que permite que diferentes aplicaciones se comuniquen entre sí y compartan información y funcionalidades.

En cuanto a los entornos de desarrollo, se ha optado por los siguientes:

- **Google Colab:** se ha utilizado en las fases iniciales para las pruebas y verificación del código. Colab es un entorno de Jupyter Notebook gratuito y basado en la nube que permite ejecutar código Python en la nube sin necesidad de instalar software adicional.
- **Visual Studio Code:** se ha utilizado como editor de código que ofrece una amplia gama de características para facilitar el desarrollo de software.
- **Axure RP 10**: esta herramienta se ha utilizado para el desarrollo del *mockup*.

Para el almacenamiento de los datos, se ha optado por explorar dos opciones principales para la gestión de datos: Pinecone y Firebase. Ambas plataformas ofrecen soluciones en la nube. Pinecone se ha usado para almacenar los *embeddings* de forma eficiente, aprovechando su simplicidad y escalabilidad para gestionar los datos vectoriales. Firebase se ha utilizado para la autenticación de los usuarios, garantizar la seguridad y el acceso controlado de los datos.

Finalmente, se ha utilizado OpenAI, una librería que da acceso a los LLMs: unos modelos de IA capaces de generar texto con calidad humana, traducir texto en otros idiomas, escribir diferentes tipos de contenido creativo y responder a preguntas de manera informativa. Para su uso, se ha necesitado una API-KEY.

Capítulo 2

Diseño conceptual

En este capítulo se detallan los requisitos funcionales y no funcionales del proyecto, así como el diseño inicial del *mockup*. La definición previa de los requisitos funcionales y no funcionales permiten asegurar que el sistema cumpla con los objetivos establecidos y se adapte a las necesidades del usuario. Además, se presentará el *mockup* inicial, el cual facilita la validación temprana del diseño y la identificación de posibles mejoras.

2.1. Requisitos funcionales

Los **requisitos funcionales** son la declaración detallada del comportamiento de un sistema. Estos requisitos se centran en lo que debe hacer el sistema, no en cómo lo debe hacer, y deben ser precisos y no ambiguos para evitar confusiones en el proceso de desarrollo. Deben cubrir todas las funcionalidades del sistema, evitando contradicciones y priorizando aquellas funcionalidades que se consideren más relevantes para el usuario. Para ello, se ha seguido un esquema de preguntas que se presentan en los Cuadros 2.1 y 2.2.

Cuadro 2.1: Registro de usuario

Pregunta	**Descripción**
¿Qué acción debe realizar el sistema?	Permitir al usuario completar un formulario de registro con datos personales y establecer preferencias alimentarias y restricciones.
¿Qué datos necesita el sistema para realizar la acción?	Datos personales del usuario: edad, sexo, altura, peso. Preferencias alimentarias y restricciones. Objetivos de salud del usuario e información sobre la actividad física y estilo de vida
¿Qué información debe mostrar el sistema al usuario como resultado de la acción?	Un mensaje de confirmación de registro y una lista de opciones para establecer preferencias alimentarias y restricciones.
¿En qué casos la acción no se puede realizar y qué mensaje se debe mostrar al usuario?	La acción no se puede realizar si: Los datos personales del usuario son incompletos o inválidos. Las preferencias alimentarias o restricciones son incompatibles. En estos casos, el sistema debe mostrar un mensaje de error al usuario.

Cuadro 2.2: Análisis de datos

Pregunta	Descripción
¿Qué acción debe realizar el sistema?	Permitir al usuario proporcionar información sobre hábitos alimenticios y niveles de actividad y analizar la información proporcionada por el usuario.
¿Qué datos necesita el sistema para realizar la acción?	Detalles sobre hábitos alimenticios y niveles de actividad física del usuario.
¿Qué información debe mostrar el sistema al usuario como resultado de la acción?	Un resumen de la información proporcionada por el usuario y un análisis de sus hábitos alimenticios y niveles de actividad.
¿En qué casos la acción no se puede realizar y qué mensaje se debe mostrar al usuario?	La acción no se puede realizar si la información proporcionada por el usuario es incompleta o inválida. En este caso, el sistema debe mostrar un mensaje de error al usuario.

Cuadro 2.3: Generación de dietas personalizadas

Pregunta	Descripción
¿Qué acción debe realizar el sistema?	Permitir al usuario seleccionar metas específicas e indicar preferencias adicionales para generar una dieta personalizada basada en sus metas y preferencias.
¿Qué datos necesita el sistema para realizar la acción?	Metas específicas del usuario (pérdida de peso, ganancia de masa muscular etc.) y preferencias adicionales (vegetariano, vegano, intolerante a la lactosa etc.).
¿Qué información debe mostrar el sistema al usuario como resultado de la acción?	Una dieta personalizada con detalles sobre las comidas y porciones recomendadas, además de un resumen de las calorías y nutrientes de la dieta.
¿En qué casos la acción no se puede realizar y qué mensaje se debe mostrar al usuario?	La acción no se puede realizar si las metas o preferencias del usuario son incompatibles. En este caso, el sistema debe mostrar un mensaje de error al usuario.

Cuadro 2.4: Planificación de comidas

Pregunta	Descripción
¿Qué acción debe realizar el sistema?	Permitir al usuario seleccionar alimentos de una lista sugerida para cada comida y programar horarios de comidas.
¿Qué datos necesita el sistema para realizar la acción?	Preferencias alimentarias y restricciones del usuario, así como sus horarios y preferencias personales.
¿Qué información debe mostrar el sistema al usuario como resultado de la acción?	Un plan con detalles sobre las comidas y porciones recomendadas para cada día, además de un resumen de las calorías y nutrientes del plan de comidas.
¿En qué casos la acción no se puede realizar y qué mensaje se debe mostrar al usuario?	La acción no se puede realizar si las preferencias o restricciones del usuario no son compatibles con los alimentos disponibles. En este caso, el sistema debe mostrar un mensaje de error al usuario.

A partir de la definición de requisitos, se extraerán individualmente, acompañados de su respectiva descripción, su implementación y definiendo la prioridad en la aplicación; de

esta forma podemos determinar qué funcionalidades deben formar parte de la aplicación, ya que se priorizarán aquellas que se consideren más relevantes para el usuario.

Cuadro 2.5: Cuadro de requisitos funcionales - parte 1

Tabla de requisitos funcionales			
Requisito	**Descripción**	**Implementación**	**Prioridad**
Registro de usuario	El usuario debe poder ingresar su dirección de correo electrónico, número de teléfono móvil y contraseña para registrarse.	Se validarán los datos ingresados y se registrará al usuario en el sistema.	Alta
Inicio de sesión	El usuario debe poder ingresar su dirección de correo electrónico y contraseña para iniciar sesión.	Se validarán los datos ingresados y se iniciará sesión en el sistema.	Alta
Recuperación de contraseña	El usuario debe poder recuperar su contraseña si la ha olvidado.	Se enviará un correo electrónico al usuario con un enlace para restablecer la contraseña.	Media
Ingreso de peso actual	El usuario debe poder ingresar su peso actual en kilogramos.	Se validará que el valor ingresado sea un número positivo.	Media
Ingreso de fecha de nacimiento	El usuario debe poder ingresar su fecha de nacimiento.	Se validará que la fecha ingresada sea válida.	Media
Ingreso de estatura	El usuario debe poder ingresar su estatura en centímetros.	Se validará que el valor ingresado sea un número positivo.	Media
Selección de sexo	El usuario debe poder seleccionar su sexo (hombre, mujer u otro).	Se proporcionará una lista de opciones para que el usuario seleccione su sexo.	Media
Selección de actividad física semanal	El usuario debe poder seleccionar su nivel de actividad física semanal (nada, baja, media, alta).	Se proporcionará una lista de opciones para que el usuario seleccione su nivel de actividad física semanal.	Media

Cuadro 2.6: Cuadro de requisitos funcionales - parte 2

Requisito	Descripción	Implementación	Prioridad
Registro de preferencias alimentarias	El usuario debe poder registrar sus preferencias alimentarias, como alergias, intolerancias o preferencias vegetarianas/veganas.	Se proporcionará una lista de opciones para que el usuario seleccione sus preferencias alimentarias.	Media
Generación de lista de compra	El usuario debe poder generar una lista de compra basada en su plan de dieta y sus preferencias alimentarias.	La lista de compra incluirá los ingredientes necesarios para preparar las recetas del plan de dieta.	Alta
Importación de recetas	El usuario debe poder importar recetas desde otras aplicaciones o sitios web.	Se podrán importar recetas en formato .txt, .json o .xml.	Alta
Exportación de recetas	El usuario debe poder exportar sus recetas a otras aplicaciones o sitios web.	Se podrán exportar recetas en formato .txt, .json o .xml.	Media
Calificación de recetas	El usuario debe poder calificar las recetas que ha probado.	Las calificaciones se podrán usar para recomendar recetas a otros usuarios.	Media

2.2. Requisitos no funcionales

Los requisitos no funcionales son descripciones de las cualidades que un sistema debe tener para ser más efectivo y confiable. Estos requisitos se centran en cómo debe comportarse el sistema, no en lo que debe hacer. Los principales aspectos que abarcan la definición de estos requisitos son:

- **Seguridad**: definen las medidas que el sistema debe tomar para proteger los datos y el acceso al mismo.
- **Usabilidad**: facilidad con la que los usuarios pueden aprender y usar el sistema.
- **Mantenimiento**: facilidad de modificación y actualización del sistema.
- **Multiplataforma**: definen la capacidad del sistema para funcionar en diferentes plataformas o entornos.

Cuadro 2.7: Cuadro de requisitos no funcionales

Tabla de requisitos no funcionales	
Requisito	**Descripción**
Seguridad de datos	Los datos personales de los usuarios, incluyendo información de salud y preferencias alimentarias, deben ser encriptados durante la transmisión y almacenamiento para protegerlos de accesos no autorizados.
Disponibilidad continua	Se debe garantizar una disponibilidad del sistema con redundancia de servidores y procedimientos de recuperación ante desastres para minimizar el tiempo de inactividad.
Interfaz de usuario intuitiva	La interfaz de usuario debe ser fácil de usar y estar diseñada de manera intuitiva para guiar a los usuarios a través de las diferentes funcionalidades de la aplicación.
Compatibilidad multiplataforma	La aplicación debe ser compatible con una variedad de dispositivos y sistemas operativos, incluyendo navegadores web, dispositivos móviles (iOS y Android) y posiblemente asistentes virtuales.
Precisión en los cálculos y recomendaciones	Las recomendaciones dietéticas y los cálculos de nutrientes deben ser precisos y basados en evidencia científica actualizada, garantizando la seguridad y efectividad de las dietas generadas.
Accesibilidad para personas con discapacidad	La aplicación debe cumplir con los estándares de accesibilidad web, *Web Content Accessibility Guidelines* (WCAG), para garantizar que sea utilizable por personas con discapacidades visuales, auditivas o motoras.

2.3. *Mockup* inicial

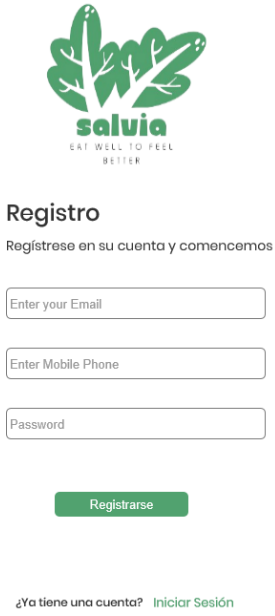
Esta sección se enfoca en explicar el funcionamiento de la aplicación web, una plataforma interactiva accesible a través de un navegador de internet, que a diferencia de una página web estática, permite una mayor interacción y funcionalidad sin necesidad de descargar software adicional, permitiendo su visualización en cualquier tipo de dispositivo (escritorio, móvil o tablet). Para ello, se describe cada paso, comenzando desde el registro, incluyendo para cada uno su *mockup*.

2.3.1. Registro

Las funcionalidades de la pantalla de registro de la aplicación son:

1. **Correo electrónico:** campo para que el usuario ingrese su dirección de correo electrónico.
2. **Teléfono móvil:** campo para que el usuario ingrese su número de teléfono móvil.
3. **Contraseña:** campo para que el usuario ingrese una contraseña para su cuenta.
4. **Botón Registrarse:** botón que envía la información del formulario al servidor para crear una nueva cuenta.

Deben tenerse en cuenta algunas funcionalidades no visibles como son el caso de la **validación de datos:** la aplicación valida la información ingresada por el usuario para asegurar que sea válida y segura.



2.3.2. Inicio de sesión

Las funcionalidades de la pantalla de inicio de sesión son las siguientes:

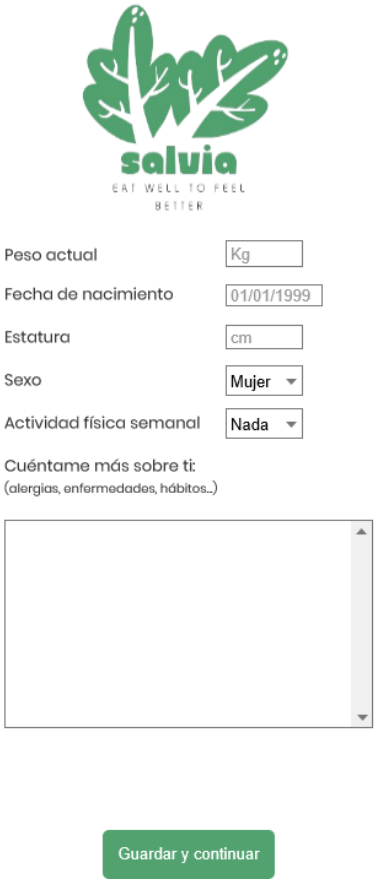
1. **Campos de entrada para el correo electrónico y la contraseña:** los usuarios deben introducir su dirección de correo electrónico y contraseña para iniciar sesión en la aplicación.
2. **Botón de inicio de sesión:** el botón “Iniciar sesión” inicia el proceso de autenticación del usuario.
3. **Enlace de registro:** el enlace “¿Aún no tiene cuenta?” lleva a la pantalla de registro, donde los usuarios pueden crear una nueva cuenta.

2.3.3. Ficha

Permite a los usuarios registrar su información personal. Esta información se utiliza para personalizar posteriormente la experiencia del usuario en la aplicación, permitiendo crear menús personalizados.
Los campos específicos que se muestran en la pantalla incluyen:

1. **Peso actual:** el peso actual del usuario en kilogramos.
2. **Fecha de nacimiento:** la fecha de nacimiento del usuario.
3. **Estatura:** la estatura del usuario en centímetros.
4. **Sexo:** el sexo del usuario, hombre o mujer.
5. **Actividad física semanal:** la cantidad de actividad física que realiza el usuario cada semana.
6. **Información adicional:** un campo opcional donde el usuario puede ingresar información adicional sobre sí mismo, como alergias, enfermedades o hábitos.

Una vez que el usuario ingresa esta información, puede tocar el botón "Guardar y continuar" para continuar con el siguiente paso en el proceso de registro.



2.3.4. Objetivo

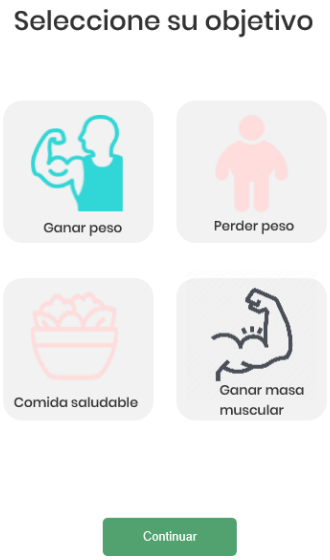
El usuario podrá elegir uno entre cuatro objetivos principales:

1. **Ganar peso:** esta opción es ideal para personas que buscan aumentar su masa corporal, ya sea por motivos estéticos o de salud. La aplicación proporcionará al usuario planes de alimentación personalizados para ayudarle a alcanzar su objetivo.

2. **Perder peso:** esta es una de las opciones más populares en las aplicaciones de *fitness*. La aplicación ayudará al usuario a crear un déficit calórico sostenible para que pueda perder peso de forma saludable y segura.

3. **Comida saludable:** esta opción es ideal para personas que quieren mejorar su alimentación, pero no necesariamente buscan perder o ganar peso. La aplicación proporcionará al usuario consejos y recetas para ayudarle a comer de forma más saludable.

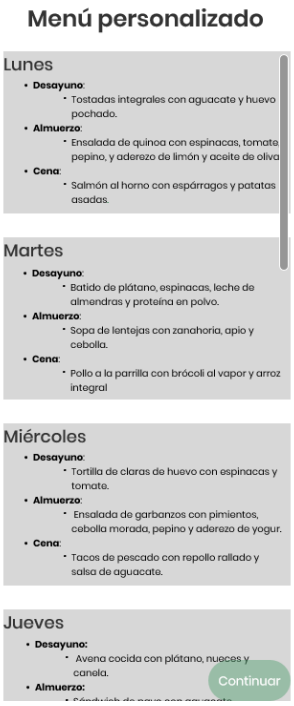
4. **Ganar masa muscular:** esta opción es ideal para personas que quieren aumentar su masa muscular. La aplicación proporcionará al usuario planes de recetas personalizados para ayudarle a alcanzar su objetivo.



Además de los cuatro objetivos principales, la pantalla también incluye un botón de "Continuar". Al tocar este botón, el usuario será dirigido a la siguiente página de la aplicación, donde podrá comenzar a personalizar su plan de *fitness*.

2.3.5. Resumen menú

La pantalla muestra el menú personalizado para toda la semana, dividido en cada día, desde el lunes hasta el domingo. Cada día, a su vez, se encuentra estructurado en tres secciones diferentes, abarcando opciones para el desayuno, comida y cena.
La finalidad de esta pantalla es proporcionar al usuario una visión panorámica de las comidas que tiene planeado consumir durante la semana, de forma que le permita planificar sus compras de ingredientes con la antelación suficiente, así como la posibilidad de ajustar cualquier plato que no le satisfaga o que no encaje con sus preferencias alimenticias.



2.3.6. Menú principal

La pantalla principal muestra el menú del día indicado que incluye tres comidas al día: desayuno, almuerzo y cena. Para cada comida, se incluye una receta y una lista de ingredientes. Los usuarios también pueden cambiar el plato en caso de que no les guste, pulsando el botón de alternativa.
Además del menú, la pantalla principal también incluye las siguientes funcionalidades:

1. **Evolución:** esta sección muestra el progreso del usuario a lo largo del tiempo. Los usuarios pueden ver cómo ha cambiado su peso, su índice de masa corporal (IMC) y sus niveles de energía.

2. **Mi ficha:** esta sección contiene información personal del usuario, como su nombre, edad, sexo y altura. Los usuarios también pueden establecer objetivos de salud en esta sección.

3. **Descargar lista de la compra:** esta función permite a los usuarios descargar una lista de la compra con todos los ingredientes que necesitan para preparar las comidas del menú.

4. **Añadir progreso:** esta función permite a los usuarios registrar su progreso en sus objetivos de salud.

5. **Preparación**: esta función permite a los usuarios saber cuáles serán los ingredientes necesarios para elaborar el plato, así como las instrucciones para hacerlo.

6. **Alternativa**: esta función permite al usuario modificar el plato en caso de que no le guste la opción propuesta; puede cambiar de plato tantas veces como desee y podrá guardarlo en su menú o no.

7. **Favorito**: esta función permite al usuario marcar platos como favoritos; esto servirá en un futuro para ofrecerle menús más personalizados de acuerdo a sus preferencias.

8. **Escaneo alimentos**: esta función permite al usuario hacer una foto del plato y poder determinar cuántas calorías tiene.

Capítulo 3

Diseño e implementación

Los cuatro componentes principales de Salvia (presentados de manera esquemática en la Figura 3.1) son el *frontend* (Sección 3.3), la API Web (Sección 3.5), el *backend* (Sección 3.2) y el módulo de procesamiento de texto, o *Text Processing Module* en inglés (Sección 3.4):

- ***Frontend* (React):** la interfaz de usuario de Salvia, desarrollada en React, donde los usuarios interactúan directamente. El *frontend* envía solicitudes HTTP a la API web para realizar diversas acciones.
- **API Web (FastAPI):** actúa como una capa intermedia entre el *frontend* y Firebase. Define *endpoints* para manejar las solicitudes entrantes del *frontend*, procesarlas según sea necesario y comunicarse con Firebase para realizar operaciones en la base de datos.
- ***Backend* (Firebase y Pinecone):** se encarga del almacenamiento de datos en la nube. Aquí es donde se almacenan los datos de Salvia como recetas, respuestas a preguntas y cualquier otra información relevante. La API web interactúa con Firestore para realizar operaciones de lectura y escritura en la base de datos, y Pinecone para el almacenamiento y acceso a los *embeddings* generados (véase la Sección 4.1.3).
- ***Text Processing Module*:** un módulo en el *backend* que se encarga del procesamiento de texto, utilizado para tareas como dividir el texto de las recetas en trozos más pequeños o analizar preguntas para generar respuestas relevantes. Este módulo puede interactuar tanto con la API web como con Firebase, según sea necesario.

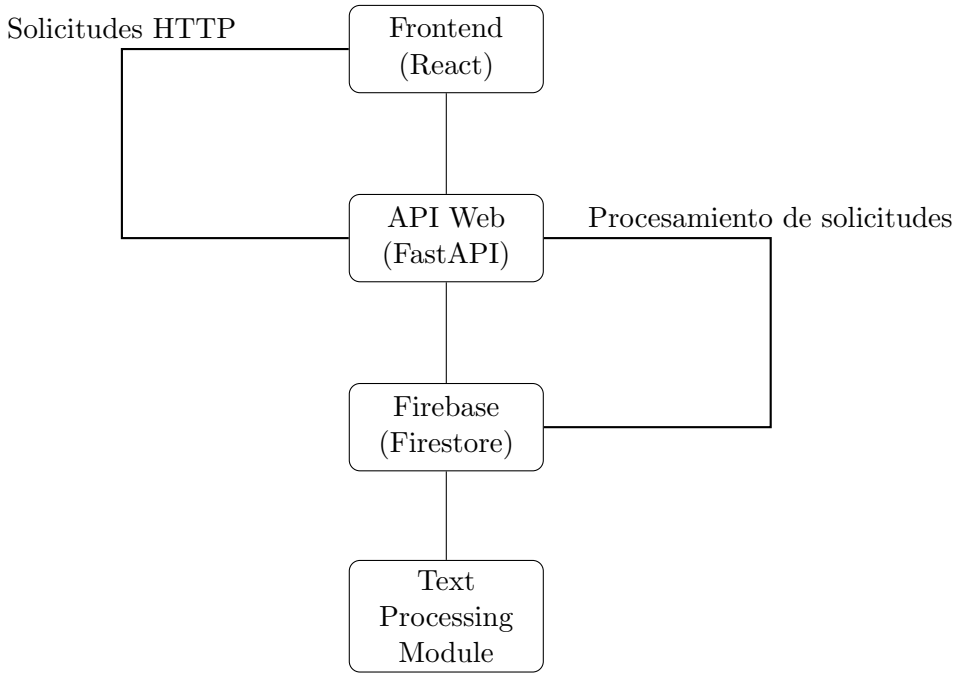


Figura 3.1: Arquitectura del sistema

En cuánto al funcionamiento general de la aplicación web, comienza con el registro del usuario, rellenando aquellos campos necesarios (email, teléfono, contraseña). Estos datos se almacenan en Firebase Authentication, donde cada usuario registrado tiene su propio documento dentro de esta colección, identificado por un ID único generado automáticamente por Firebase. Además de los datos básicos de autenticación, cada usuario completa una ficha con información adicional relevante para personalizar su dieta (edad, género, peso actual, altura y alergias) y la selección de su objetivo (bajar de peso, aumentar de peso, ganar masa muscular o dieta saludable). Para almacenar estos nuevos datos en la nube, se accede al ID del documento correspondiente al usuario creado y se actualiza con esta nueva información. Firestore también proporciona sincronización en tiempo real, lo que significa que cualquier cambio realizado en la ficha del usuario se reflejará instantáneamente en todos los dispositivos conectados.

Tras obtener la información fundamental del usuario, es necesario enviarla como *prompt* a los modelos para generar los menús. Por ello, se accede al usuario del que se desea obtener este menú y se obtiene la meta y las alergias, con lo que se construirá el *prompt* que se le pasará al modelo GPT 3.5. Este *prompt* se construye a partir de la estructura inicial de la solicitud:

```
const baseUrl = 'http://localhost:8000/get_answer?question=Genera%un%men%C3%BA%semanal%con%las%recetas%que%tengas,';
```

Figura 3.2: Estructura inicial de la solicitud

Según el objetivo dietético del usuario, se define un parámetro específico que se añadirá a la URL:

- "bajar de peso" se convierte en "que %quiero %bajar %de %peso".
- "subir de peso" se convierte en "que %quiero %subir %de %peso".
- "ganar masa muscular" se convierte en "que %quiero %ganar %masa %muscular".

- "comer sano" se convierte en "que %quiero %comer %sano".

Este *prompt* (véase en la Figura 3.3) se pasa al modelo como el *input* del usuario, a través de una solicitud HTTP desde el *frontend* al *endpoint* del *Text Processing Module* (*get_answer*)

```
1     messages=[
2         {"role": "system", "content": "Eres un asistente
              especialista en nutricion y tu objetivo es generar
              menus semanales estructurados por dias y comidas en
              formato JSON y eliminando todos los platos y
              sustituyendolo por otra alternativa para aquellos que
              indique el usuario como alergia o que no quiera.
              Respondes en base a la informacion que te indicare
              como 'contexto'"},
3         {"role": "user", "content": prompt}
4     ]
```

Figura 3.3: *Prompt* modelo gpt3.5

Para que el modelo pueda procesar el *prompt* adecuadamente, es necesario que haya sido previamente entrenado. El entrenamiento se realiza utilizando el contenido del libro de recetas (*"Recetas Saludables para Cada Día"*). Este libro es una guía culinaria que promueve el consumo de frutas y vegetales como parte de una dieta saludable y balanceada. Su primera versión data del 19 de agosto de 2009, con modificaciones hasta el 17 de mayo de 2023, lo que indica un esfuerzo continuo por mantener la relevancia y actualidad de su contenido. Aunque no se menciona un autor individual específico, el libro es resultado de la colaboración entre diversas organizaciones, incluyendo la *"California Pear Advisory Board"*, la *"California Avocado Commission"* y la *"California Table Grape Commission"*. Estas entidades aportan su experiencia y conocimiento sobre sus respectivos productos, garantizando la calidad y autenticidad de las recetas presentadas. Se puede encontrar en la web Campeones por el cambio.

A partir de este libro, se generan fragmentos de texto (*chunks*), los cuales son almacenados localmente en una variable específica. Posteriormente, es fundamental generar los *embeddings* de estos *chunks* utilizando el modelo *text-embedding-ada-002*. Los *embeddings* generados se almacenan en Pinecone, para lo cual se utilizan las funciones de la API correspondiente para su inserción. Una vez almacenados los *embeddings* de los *chunks*, se procede a generar los *embeddings* de la consulta (*query*) realizada por el usuario. Con estos *embeddings*, se realiza una consulta en Pinecone con el objetivo de obtener las respuestas más relevantes. Para ello, se fija el parámetro *top_k* en 3, de manera que se obtengan las tres mejores respuestas potenciales. Es importante definir el contexto en el que operará el asistente, en este caso, un asistente nutricional que ofrece menús. Tras recibir el *prompt* y teniendo en cuenta el contexto indicado, el modelo generará un archivo JSON como el ilustrado en la Figura 3.4.

```
    {
      "lunes": {
        "desayuno": "Tortilla de espinacas",
        "almuerzo": "Estofado de ternera con verduras",
        "cena": "Ensalada de quinoa y aguacate"
      }
    }
```

Figura 3.4: Fichero JSON con el menú semanal

Esta respuesta se almacena en Firebase en una nueva colección llamada *menuSemanal* que contiene para cada día de la semana con sus correspondientes comidas (desayuno, almuerzo y cena), estos datos se almacenarán en sus campos correspondientes. Para mostrar al usuario esta información basta con acceder al usuario y extraer los campos con las etiquetas correspondientes. En la Figura 3.5 se puede ver un ejemplo de la estructura de un documento (usuario) y sus respectivas colecciones y campos:

```
{
  "menuSemanal": {
    "domingo": {
      "almuerzo": "Estofado de ternera con verduras",
      "cena": "Ensalada de quinoa y aguacate",
      "desayuno": "Tortitas de platano sin harina"
    },
    "lunes": {
      "almuerzo": "Ensalada de pollo a la parrilla con aderezo de
          limon",
      "cena": "Pescado al horno con verduras asadas",
      "desayuno": "Omelette de espinacas y champi ones"
    },
    "martes": {
      "almuerzo": "Salmon a la plancha con esparragos",
      "cena": "Brochetas de carne con vegetales",
      "desayuno": "Avena con frutas frescas"
    },
    ...
    ...
  },
  "objetivo": "comer sano",
  "userId": "2scr4AtdmGYaPdHo7VQi28jpM8i1"
}
```

Figura 3.5: Datos de un usuario

Por último, el usuario podrá tomar una foto para analizar las calorías. Esta foto no se almacena ya que su uso es temporal. Se almacena de forma temporal en un repositorio para su posterior procesamiento (convertir a base64) y pasarla al modelo GPT 4-vision-preview como una URL, el cual devuelve un fichero JSON del que se extrae el contenido de la etiqueta *"content"* y se muestra al usuario.

3.1. Diseño del *backend*

El desarrollo del Asistente Nutricional se basa en una robusta *Representational State Transfer* (API REST[1]) que presenta varias funcionalidades. La arquitectura de la API sigue las convenciones y mejores prácticas establecidas, garantizando coherencia, facilidad de uso y escalabilidad. A través de esta API, los usuarios pueden acceder a varios servicios, desde la gestión de su perfil y generación del menú personalizado hasta el seguimiento nutricional y gestión de la lista de la compra. Cabe destacar que la estructura que se expondrá a continuación corresponde a la aplicación completa, sin embargo, el alcance ha sido centrado en desarrollar las funcionalidades que requieren IA:

- **GET /usuario/{id_usr}/registro**: obtiene los detalles del registro de un usuario específico a partir de su identificador.
- **POST /usuario/login**: inicia sesión en la aplicación con las credenciales proporcionadas.
- **POST /usuario/ficha**: crea una nueva ficha de usuario con los datos proporcionados.
- **PUT /usuario/ficha/{id_usr}**: actualiza la ficha de usuario existente con el identificador proporcionado.
- **GET /usuario/ficha/{id_usr}**: obtiene la ficha de usuario con el identificador especificado.
- **POST /usuario/objetivo**: guarda los objetivos nutricionales para un usuario específico.
- **PUT /usuario/objetivo/{id_usr}**: actualiza los objetivos nutricionales para el usuario con el identificador proporcionado.
- **GET /usuario/objetivo/{id_usr}**: obtiene los objetivos nutricionales del usuario con el identificador especificado.
- **GET /menu/generar{id_usr}**: genera un menú personalizado basado en las preferencias del usuario con el identificador especificado.
- **POST /menu/guardar{id_usr}**: guarda un menú generado para el usuario con el identificador especificado..
- **GET /menu/{id_usr}**: obtiene los menús guardados para un usuario dado su identificador (ID).
- **GET /menu/preparacion**: obtiene instrucciones de preparación para los platos incluidos en el menú.
- **GET /menu/alternativa**: obtiene alternativas para los platos del menú.
- **PUT /menu/modificar**: modifica un menú existente guardando alternativas de platos.

[1]Una API REST es una interfaz de comunicación entre sistemas de información que sigue los principios del estilo arquitectónico REST (Transferencia de Estado Representacional). Permite a las aplicaciones acceder y modificar datos de forma sencilla y uniforme a través de HTTP.

- **GET /menu/home{id_usr}**: obtiene la página principal del menú diario para el usuario con el identificador especificado.
- **POST /usuario/personalizacion/favoritos{id_usr}**: guarda platos favoritos para el usuario con el identificador especificado.
- **POST /usuario/personalizacion/platosdescartados{id_usr}**: guarda platos descartados para el usuario actual.
- **GET /menu/personalizado{id_usr}**: obtiene un menú completamente personalizado para el usuario con el identificador especificado.
- **POST /usuario/progreso{id_usr}**: guarda el progreso nutricional del usuario con el identificador especificado.
- **GET /usuario/progreso/{id_usr}**: obtiene el progreso nutricional del usuario con el identificador especificado.
- **GET /usuario/evolucion/{id_usr}**: obtiene la evolución del progreso nutricional del usuario con el identificador especificado.
- **GET /logistica/listacompra**: obtiene la lista de la compra diaria para el usuario con el identificador especificado.
- **GET /usuario/fotoPlato**: permite al usuario hacer una foto para identificar las calorías presentes.

3.2. Implementación *backend*

En esta sección se presenta el uso de Firebase como *backend* en el desarrollo de Salvia. Se explora cómo Firebase se integra en el proyecto para manejar la autenticación de usuarios, el almacenamiento y gestión de datos, el almacenamiento de archivos y la ejecución de lógica del *backend* mediante funciones *serverless*[2].

3.2.1. Autenticación de usuarios

Para integrar este servicio en la aplicación, se han utilizado las funciones *getAuth*, *signInWithEmailAndPassword* y *onAuthStateChanged*.

La función *getAuth* se utiliza para inicializar y obtener una instancia del servicio de autenticación, permitiendo así que la aplicación gestione la autenticación de usuarios de manera efectiva. Cuando un usuario pretenda iniciar sesión en la aplicación utilizando su dirección de correo electrónico y contraseña, recurre a la función *signInWithEmailAndPassword*. Esta función valida las credenciales proporcionadas por el usuario y, si son correctas, permite el acceso a la cuenta del usuario en la aplicación. Por otro lado, *onAuthStateChanged* actúa como un observador de cambios en el estado de autenticación del usuario. Esta función se activa cada vez que el estado de autenticación del usuario cambia, por ejemplo, cuando un usuario inicia sesión o cierra sesión en la aplicación. Este observador es importante para gestionar la sesión del usuario, ya que permite realizar acciones específicas en respuesta a cambios en el estado de autenticación, como redirigir al usuario a páginas específicas o actualizar la interfaz de usuario.

[2]Las funciones *serverless* son un modelo de computación en la nube donde los desarrolladores pueden ejecutar fragmentos de código sin la necesidad de gestionar o aprovisionar servidores.

La integración de Firebase Authentication ha permitido verificar de manera efectiva el estado de autenticación del usuario al iniciar sesión en la aplicación. Mediante la función *useEffect*, se ha suscrito a los cambios en el estado de autenticación del usuario y se ha realizado una consulta a Firestore para determinar si el usuario tiene datos guardados en la base de datos. En función del resultado de esta consulta, se redirige al usuario a la página correspondiente, ya sea la página de datos del usuario o la página de registro de nuevos datos.

3.2.2. Almacenamiento y gestión de datos

Además de la autenticación de usuarios, se ha utilizado Cloud Firestore para el almacenamiento y la gestión de datos en la aplicación. Para interactuar con Firestore, se han importado las funciones *getFirestore*, *doc* y *getDoc*.

Estas funciones se han empleado para consultar la base de datos y verificar si un usuario tiene datos guardados en ella. La integración de Firebase Authentication y Firestore ha permitido gestionar de manera efectiva la interacción del usuario con la base de datos en tiempo real en la aplicación. Un aspecto relevante es la definición de las reglas de seguridad de Cloud Firestore utilizando un lenguaje específico que permite especificar condiciones bajo las cuales se permite la lectura y escritura de documentos en la base de datos. A continuación se presenta la configuración de las reglas de seguridad implementadas:

```
1    rules_version = '2';
2    service cloud.firestore {
3        match /databases/{database}/documents {
4            match /DatosUsuarios/{document=**} {
5                allow read, write: if request.auth != null;
6            }
7        }
8    }
```

- **Versión de las reglas:** la declaración *rules_version = '2'* indica que se está utilizando la versión 2 del lenguaje de reglas de Firestore. Esto es importante para asegurar la compatibilidad con las características y sintaxis más recientes y seguras del servicio.
- **Servicio Cloud Firestore:** *service cloud.firestore* especifica que las reglas definidas a continuación aplican a Firestore.
- **Emparejamiento de documentos:** *match /databases/database/documents* define que las reglas siguientes aplicarán a todos los documentos dentro de cualquier base de datos en el proyecto Firestore.
- **Acceso a la colección *DatosUsuarios*:** *match /DatosUsuarios/{document=**}* específica la colección *DatosUsuarios*. Aquí, *{document=**}* denota que la regla se aplica a todos los documentos dentro de esta colección, sin importar su estructura o profundidad.
- **La instrucción *allow read, write*:** *if request.auth != null;* establece que se permite tanto la lectura como la escritura en cualquier documento de la colección *DatosUsuarios* únicamente si la solicitud proviene de un usuario autenticado *(request.auth != null)*. Esto significa que el usuario debe haber iniciado sesión para acceder a los datos.

La configuración de seguridad implementada cumple con los requisitos de Salvia, asegurando que solo los usuarios autenticados puedan interactuar con los datos sensibles almacenados en la colección *DatosUsuarios*. Esto protege la privacidad de los usuarios y evita accesos no autorizados, contribuyendo a la integridad y seguridad general de la aplicación.

Además, al centralizar la lógica de seguridad en las reglas de Firestore, se simplifica el manejo de permisos y se reduce el riesgo de errores de seguridad en el código de la aplicación.

3.3. Implementación *fronted*

En este apartado se aborda el desarrollo del *frontend*. Se analizan las características y funcionalidades implementadas, así como el proceso técnico y creativo detrás de la creación de una experiencia de usuario integral. Desde el inicio del proyecto, se ha utilizado un *mockup* inicial como punto de referencia para la visualización de la apariencia y disposición de la interfaz de usuario. Este *mockup* ha servido como guía conceptual, si bien es necesario señalar que su función ha sido interpretativa y adaptable durante el proceso de desarrollo del *frontend*.

Durante la etapa de desarrollo, se realizaron modificaciones sustanciales en el diseño original con el objetivo de mejorar la funcionalidad y la accesibilidad de la aplicación. Estas adaptaciones han sido el resultado de una comprensión profunda de las necesidades y expectativas del usuario, así como de los principios de diseño centrados en el usuario y en la accesibilidad web. En el Apéndice se puede ver el resultado final.

A continuación se detalla la arquitectura seguida para la implementación de la aplicación, la cual se divide en tres capas principales:

- **JavaScript core:** es el núcleo de la lógica de la aplicación y se refiere al código JavaScript principal que define tanto la funcionalidad como la interfaz de usuario de la aplicación. En este núcleo, se escribe el código responsable de manejar el estado de la aplicación, gestionar eventos, interactuar con APIs y definir la estructura y comportamiento de los componentes de la interfaz de usuario.

- ***Bridge* (Puente):** el puente es un canal de comunicación entre JavaScript y el sistema operativo nativo (Web). Este puente permite que el código JavaScript interactúe con los componentes nativos y las APIs de la plataforma subyacente. La comunicación a través de este puente es asíncrona y se basa en el intercambio de mensajes, lo que significa que las instrucciones escritas en JavaScript se traducen en llamadas a APIs nativas utilizando mensajes en formato JSON. Este enfoque asíncrono permite que JavaScript y el código nativo operen de manera independiente, mejorando la eficiencia y la capacidad de respuesta de la aplicación.

- **Componentes nativos:** estos son los elementos de la interfaz de usuario y las funcionalidades específicas de la plataforma que se renderizan utilizando componentes nativos. React Native proporciona una serie de componentes prediseñados como *View*, *Text*, e *Image*, que se traducen a sus equivalentes nativos en Web. Esto significa que cuando se usa un componente *View* en React Native, se renderiza como un *View* nativo en Web. Estos componentes nativos aseguran que la interfaz de usuario se vea y se comporte de manera consistente con otras aplicaciones nativas del dispositivo. Además de los componentes prediseñados, se pueden diseñar componentes nativos personalizados utilizando Java (para Web).

En la Figura 3.6 se representa el diagrama de la arquitectura de React Native:

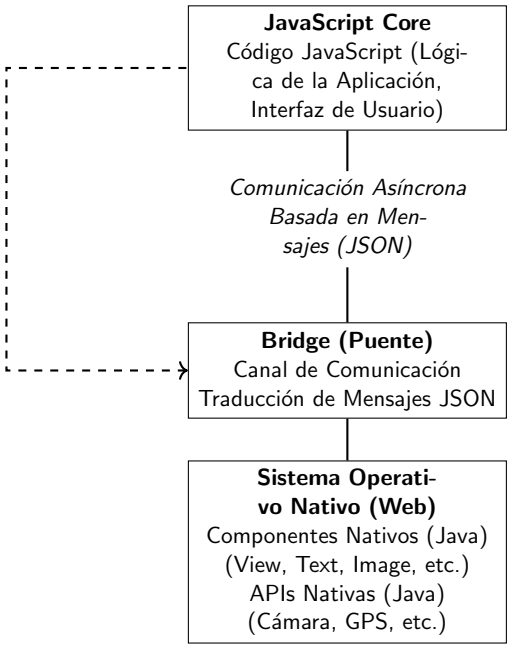


Figura 3.6: Diagrama de la arquitectura de React Native

El diseño *responsive* ha sido una prioridad desde el inicio del proyecto. Se han aplicado técnicas avanzadas de diseño *responsive* para garantizar que la aplicación se vea bien y funcione correctamente en diferentes tamaños de pantalla, desde dispositivos móviles hasta monitores de escritorio. Esta adaptación ha mejorado significativamente la experiencia del usuario, permitiendo un acceso cómodo y eficiente desde cualquier dispositivo.

Además, se ha puesto un énfasis especial en la accesibilidad web. Se han seguido las pautas de accesibilidad WCAG (véase el Cuadro 3.1) para asegurarse de que la aplicación sea utilizable por personas con discapacidades.

Cuadro 3.1: Pautas de accesibilidad WCAG

Funcionalidad	Principio	Descripción y Accesibilidad
Generación de planes de comida personalizados	Perceptible	**Descripción**: la aplicación genera planes de comida personalizados basados en las preferencias, necesidades dietéticas y objetivos de salud del usuario.
		Accesibilidad: ▪ Asegurar que el texto tenga suficiente contraste con el fondo para facilitar la lectura a personas con discapacidades visuales.
Compatibilidad con tecnologías de asistencia	Robusto	**Descripción**: La aplicación está diseñada para ser compatible con una amplia variedad de dispositivos y tecnologías de asistencia.
		Accesibilidad: ▪ Asegurar que el código de la aplicación sea compatible con los estándares web actuales y futuros, facilitando la interpretación correcta por parte de tecnologías de asistencia. ▪ Probar la aplicación con diferentes dispositivos de asistencia para garantizar su funcionalidad y usabilidad en todos los casos.

3.3.1. Conexión con la base de datos

Para conectar el *frontend* de Salvia a la base de datos en Firebase (siendo esta el propio *backend*), se utilizan varios componentes clave:

- **SDK (Software Development Kit) de Firebase:** el SDK de Firebase es una colección de bibliotecas y herramientas que proporciona Firebase para permitir la integración de sus servicios en aplicaciones. Contiene APIs específicas para cada plataforma (JavaScript para aplicaciones web). Estas APIs permiten interactuar con los servicios de Firebase, incluida la base de datos, de manera programática.
- **Configuración del proyecto:** al configurar un proyecto en la consola de Firebase, se genera un archivo de configuración que contiene información relevante sobre el proyecto, como el ID del proyecto, las claves de API y otras configuraciones específicas de Firebase. Este archivo se utiliza para inicializar Firebase en la aplicación y autenticarla con los servidores de Firebase. En la Figura 3.7 se puede observar el formato del archivo de configuración.

```
{
  "apiKey": "API_KEY",
  "authDomain": "AUTH_DOMAIN",
  "projectId": "PROJECT_ID",
  "storageBucket": "STORAGE_BUCKET",
  "messagingSenderId": "MESSAGING_SENDER_ID",
  "appId": "APP_ID"
}
```

Figura 3.7: Archivo de configuración de Firebase en JSON

- **Inicialización de Firebase:** la inicialización de Firebase en la aplicación implica cargar el archivo de configuración generado durante la configuración del proyecto y utilizarlo para inicializar Firebase en el entorno de ejecución de la aplicación. Esto se hace típicamente al principio del ciclo de vida de la aplicación, antes de cualquier interacción con los servicios de Firebase.
- **Instancia de la base de datos:** después de inicializar Firebase en la aplicación, se puede obtener una instancia de la base de datos utilizando la API correspondiente proporcionada por el SDK de Firebase. Esta instancia representa una conexión activa a la base de datos en la nube de Firebase y proporciona métodos para interactuar con ella. En Firestore, se utiliza el método *getFirestore()*.
- **Interacción con la base de datos:** una vez que se tiene una instancia de la base de datos, se pueden realizar diversas operaciones de lectura y escritura en la base de datos utilizando los métodos proporcionados por la instancia. Estas operaciones incluyen leer datos, escribir nuevos datos, actualizar datos existentes y eliminar datos. Cada plataforma y cada tipo de base de datos en Firebase tienen sus propias API específicas para realizar estas operaciones de manera eficiente y segura.

La comunicación entre el *frontend* y el *backend* se realiza principalmente a través de solicitudes HTTP. Se han utilizado métodos como GET, POST, PUT y DELETE para interactuar con el *backend*, permitiendo al *frontend* realizar operaciones como autenticación, gestión de datos y otras acciones específicas.

En Salvia, la integración del *frontend* y el *backend* es esencial para un funcionamiento fluido. En el *backend* se definen *endpoints* de API para que el *frontend* realice operaciones como autenticación y gestión de datos. En el *frontend* se llaman a estos *endpoints* utilizando solicitudes HTTP para realizar acciones específicas, como guardar datos de usuario o recuperar recetas.

Es importante manejar las respuestas del *backend* adecuadamente en el *frontend*. Para ello se intepretan los datos devueltos y se actualiza la interfaz de usuario según sea necesario. Además, se implementan medidas de seguridad como *tokens* de autenticación y HTTPS para proteger la comunicación entre el *frontend* y el *backend*. Finalmente, se optimiza el rendimiento en ambos lados para mejorar la velocidad de carga y la capacidad de respuesta de la aplicación. Esto implica estrategias como el almacenamiento en caché de datos y la minimización de solicitudes al servidor.

En el siguiente repositorio de github se encuentra el código desarrollado.

3.4. Módulo *text_processing*

Para el procesamiento y análisis de textos y datos visuales es necesario un módulo. Este módulo es fundamental para la extracción de información relevante y la generación de respuestas automatizadas. Las funcionalidades de este módulo son:

- **Lectura y fragmentación de PDF (*create_chunks*):** esta función se encarga de leer el libro de recetas y extraer su contenido textual. Luego, utiliza *RecursiveCharacterTextSplitter*, de la librería Langchain, para dividir el texto en fragmentos de tamaño manejable. Esta función recibe unos parámetros en los que se especifica el tamaño del fragmento (fijado a 800 caracteres) y el *overlap* de 100 caracteres. Estos términos (fragmentación y *overlap*) se explican en profundidad en la Sección 4.1.2.

- **Generación de *embeddings* (*generate_embeddings*):** para generar los *embeddings* (véanse en la Sección 4.1.3 los fundamentos teóricos), se hace uso del modelo *"text-embedding-ada-002"* de la API de OpenAI, del cual se obtienen los *embeddings* y se almacenan en un diccionario que tiene un identificador, los *embeddings* y la *metadata* con el texto original. El motivo por el que se ha almacenado en diccionarios ha sido para mejorar la depuración y para las pruebas iniciales de concepto, ya que de esa forma se podía identificar a qué fragmentos de texto pertenecían.

- **Codificación de imágenes en base64 (*encode_image*):** esta función convierte imágenes en formato base64, siendo este el formato que admite el modelo usado para la identificación de calorías (*"gpt-4-vision-preview"*)

- **Obtención de respuestas a preguntas (*get_answer*):** se compone de varios elementos clave que interactúan para generar una respuesta precisa a las necesidades nutricionales de un usuario. El parámetro de entrada principal es *question*, que es una cadena de texto conteniendo la pregunta formulada por el usuario. Esta pregunta es utilizada por la función para generar un *embedding*, que luego se compara con los datos almacenados en una base de datos vectorial. Estos *embeddings* capturan el significado semántico de la pregunta, permitiendo que sea comparada de manera efectiva con los *embeddings* generados de las recetas. Para formar el contexto necesario para el modelo de lenguaje, las respuestas potenciales encontradas en la base de datos se formatean en una cadena de texto. Este texto formateado se crea concatenando las respuestas potenciales y se utiliza como contexto en el *prompt* (véanse en la Sección 3.5.1 los fundamentos teóricos) para el modelo de lenguaje. Este *prompt* guía al modelo de lenguaje en la generación de una respuesta que sea relevante y coherente con la información proporcionada. El modelo de lenguaje utilizado en esta función es GPT-3.5 Turbo, desarrollado por OpenAI. El modelo está configurado para actuar como un asistente especialista en nutrición, siguiendo instrucciones específicas para generar menús semanales y ajustar los platos según las preferencias y restricciones del usuario. La configuración y las instrucciones específicas ayudan a garantizar que la respuesta generada sea útil y esté en el formato requerido.

- **Análisis de imágenes (*analisis_imagen*):** a través de la API de OpenAI, esta función analiza imágenes codificadas en base64 para proporcionar la estimación de calorías en un plato de comida. El modelo especificado es *"gpt-4-vision-preview"* (véase en la Sección 4.2). Los mensajes o *payload* que se envían al modelo están estructurados para incluir tanto texto como la imagen en sí. El contenido del mensaje especifica que el modelo debe proporcionar una estimación de las calorías del plato

en la imagen, limitándose a indicar el nombre del plato y el número de calorías, y eliminando cualquier información adicional. La solicitud HTTP se envía utilizando la biblioteca *requests* de Python. La función realiza una solicitud POST a la API de OpenAI, incluyendo los encabezados y la carga útil que se ha construido (*payload* o mensaje). Esta solicitud envía tanto las instrucciones como la imagen al modelo de lenguaje, que procesa la información y genera una respuesta basada en el análisis de la imagen y las instrucciones proporcionadas. Finalmente, la función recibe la respuesta de la API de OpenAI. Esta respuesta contiene la información generada por el modelo de lenguaje, que incluye el nombre del plato y la estimación de las calorías. La respuesta se obtiene en formato JSON. Para ilustrar este proceso de manera más práctica, se presentan dos ejemplos concretos. El primero corresponde a una paella de mariscos (Figura 3.8), y el segundo a una pasta con salsa pesto y queso parmesano (Figura 3.9). En el Apéndice se pueden encontrar más ejemplos con diferentes platos.

Figura 3.8: Paella de mariscos

Figura 3.9: Pasta salsa pesto y queso parmesano

```
{
  ...
    "content": 'Paella de
       mariscos:
       aproximadamente
       400-500 calorias'
  ...
}
```

```
{
  ...
    "content": 'Rigatoni con
        pesto y queso
       parmesano:
       aproximadamente
       400-600 calorias.'
  ...
}
```

3.5. Creación de la API web

La API web actúa como una interfaz entre el *frontend* y el *backend* de Salvia, permitiendo la comunicación y el intercambio de datos de manera eficiente y segura. A través de *endpoints*, la API web expone funcionalidades de la aplicación, como el procesamiento de recetas, la generación de respuestas y el análisis de imágenes. Como se ha mencionado anteriormente, la API web se encarga de desarrollar los componentes y *endpoints* necesarios siendo estos:

- **Función buildUrl:** esta función se encarga de construir URLs dinámicas que apuntan a un *endpoint* específico en la API web. Toma dos parámetros como entrada: la meta del usuario y una lista de alimentos a excluir. Basándose en estos parámetros, construye una URL personalizada que será utilizada para hacer una solicitud HTTP al *backend*.

- ***Endpoint* recetas:** este *endpoint* se activa cuando se recibe una solicitud GET a la ruta recetas. Está diseñado para procesar un PDF que contiene recetas. Para ello, la solicitud primero carga el PDF desde una ruta especificada. Luego, el texto de las recetas se divide en trozos más pequeños utilizando la función *create_chunks* del módulo *text_processing*. Posteriormente, se generan *embeddings* de estos trozos de receta utilizando la función *generate_embeddings* del mismo módulo. Finalmente, se establece una conexión con Pinecone utilizando una clave API proporcionada, y los *embeddings* generados se almacenan en un índice de búsqueda de similitud vectorial para su posterior recuperación.

```
1     GET "/recetas":
2         chunks = dividir_pdf("recetas.pdf")
3         embeddings = generar_embeddings(chunks)
4         indexar_en_pinecone(embeddings)
5         return {"message": "Embeddings generados"}
```

- ***Endpoint /get_answer*:** este *endpoint* se activa al recibir una solicitud GET a la ruta */get_answer*. Su función principal es procesar una pregunta como parámetro de entrada y generar una respuesta relevante. En el *backend*, esta solicitud invoca la función *get_answer* para procesar la pregunta y generar una respuesta.

```
1         GET "/get_answer" (question):
2         respuesta = obtener_respuesta(question)
3         return {"response": respuesta}
```

- ***Endpoint /get_base64_image/image_path:*** este *endpoint* se activa al recibir una solicitud GET a la ruta /get_base64_image/image_path. Su objetivo es procesar una imagen del plato y obtener información. La solicitud recibe la ruta de la imagen a analizar como parámetro de entrada. Luego, la imagen se codifica en formato base64 utilizando la función *encode_image*. Posteriormente, se invoca la función *analisis_imagen* para procesar la imagen codificada y obtener información sobre las calorías.

```
1         POST "/get_base64_image/" (imagen):
2         base64_data = extraer_base64(imagen)
3         resultado = analizar_imagen(base64_data)
4         return {"resultado": resultado}
```

3.5.1. Prompting

En la fase de creación de la API web hay que tener en cuenta que la aplicación usa la búsqueda semántica para extraer los documentos relevantes basados en consultas en lenguaje natural. Para hacer esto posible son necesarias las técnicas de *prompting*. Para explicarlo de una manera más sencilla e intuitiva vamos a suponer que la ingeniería de *prompts* es como ser un director de orquesta para la inteligencia artificial. Se trata de crear instrucciones claras y precisas (los *prompts*) que le indiquen al LLM qué se espera de él. Un buen *prompt* guía a la Inteligencia Artificial hacia el resultado deseado. Esta técnica necesita tres pilares fundamentales: dominio del lenguaje, conocimiento del dominio y comprensión del LLM. Durante el desarrollo del proyecto se estudió como identificar un mal *prompting* y como elaborar uno correcto. Según [4], las instrucciones de los *prompts* deben ser claras y precisas. Por ejemplo, si queremos pedirle que nos diga las calorías del plato que le indiquemos:

- **Mal *prompt***: *"Ofrece una idea general del total de las posibles calorías presentes en el plato."* Este *prompt*, si bien describe el objetivo general de la consulta, carece de la precisión necesaria para guiar al LLM hacia la respuesta deseada. Al ser demasiado abierto, el modelo podría generar información irrelevante o incompleta, como descripciones del plato, recetas alternativas o comentarios sobre su sabor
- **Buen *prompt***: *"Ofrece una idea general del total de las posibles calorías presentes en el plato. Limítate exclusivamente a poner nombre del plato y el número de calorías"*. Este prompt proporciona instrucciones claras y directas al LLM, enfocándolo en la información específica que se requiere. De esta manera, se evita la generación de texto innecesario y se aumenta la probabilidad de obtener una respuesta precisa y concisa

La diferencia es que con el segundo *prompt* vamos a personalizar la respuesta, evitando que genere texto que no necesitamos, ya que lo único que queremos es el nombre del plato y la cantidad de calorías, mientras que con el primer *prompt* se generará una respuesta con información irrelevante que no necesitamos. Al utilizar el modelo GPT-3.5, es preferible proporcionar instrucciones detalladas, especialmente en contextos médicos, para asegurar respuestas precisas y adecuadas. Una instrucción más extensa y específica, como la anterior, proporciona el contexto necesario, reduce la ambigüedad y minimiza errores. En contraste, una instrucción más breve como *"proporciona el nombre del plato y el número exacto de calorías"* puede llevar al modelo a interpretar erróneamente que se necesita una cantidad precisa. Esto podría resultar en respuestas que indican al usuario la imposibilidad de cumplir con la solicitud debido a posibles imprecisiones en un contexto médico, ya que la información proporcionada debe ser considerada como orientativa y no debe ser tomada literalmente.

3.6. Pruebas

En esta sección se detallan las pruebas realizadas para validar el correcto funcionamiento del asistente nutricional. El objetivo principal de estas pruebas es garantizar que la aplicación cumple con los requisitos funcionales y de usabilidad establecidos. Cada caso de prueba se enfoca en verificar diferentes funcionalidades, desde el registro de usuarios hasta la generación de los menús. A continuación, se presentan los detalles de cada prueba junto con los resultados obtenidos durante el proceso de validación.

3.6.1. Pruebas unitarias

Las pruebas unitarias son importantes en el desarrollo de software, especialmente en aplicaciones basadas en React como la que se ha desarrollado. El componente *RegisterUser* en Salvia es fundamental para permitir que los usuarios se registren correctamente en el sistema. A continuación, se detallan los componentes específicos de las pruebas y su aplicación directa en este escenario:

- ***Mock* de Firebase Authentication:** se utiliza Jest para simular el comportamiento de Firebase Authentication mediante *mocks*. Esto permite controlar las respuestas de las funciones como *createUserWithEmailAndPassword* y asegurar que el componente reaccione adecuadamente a diferentes escenarios, como registros exitosos o errores.
- **Renderización y emulación de eventos:** Testing Library permite renderizar el componente *RegisterUser* dentro de un contexto controlado, como el *MemoryRouter*[3]. Luego, se simulan eventos como cambios en los campos de email, teléfono y contraseña, así como *clicks* en el botón de registro para verificar el flujo completo de registro.

Se han implementado varias pruebas unitarias (véase la Figura 3.10) que abordan diferentes aspectos del componente *RegisterUser*:

- **Validación de email y teléfono:** se verifica que el componente muestre mensajes de error apropiados cuando los usuarios ingresan formatos inválidos de email (correo@invalido) o teléfono (123456789). Esto asegura que los datos introducidos cumplan con los criterios requeridos antes de proceder con el registro.
- **Activación del botón de registro:** se prueba que el botón de registro se active solo cuando todos los campos obligatorios (email, teléfono y contraseña) estén completos y en un formato válido (validemail@example.com, 612345678, validpassword). Esto garantiza una experiencia fluida para los usuarios al asegurar que no intenten registrarse con datos incompletos o incorrectos.
- **Navegación después del registro exitoso:** se simula un registro exitoso al completar todos los campos con datos válidos y hacer clic en el botón de registro. Luego, se verifica que la aplicación navegue correctamente a la página de inicio (/) después del registro exitoso, validando así el flujo completo de registro desde la perspectiva del usuario.

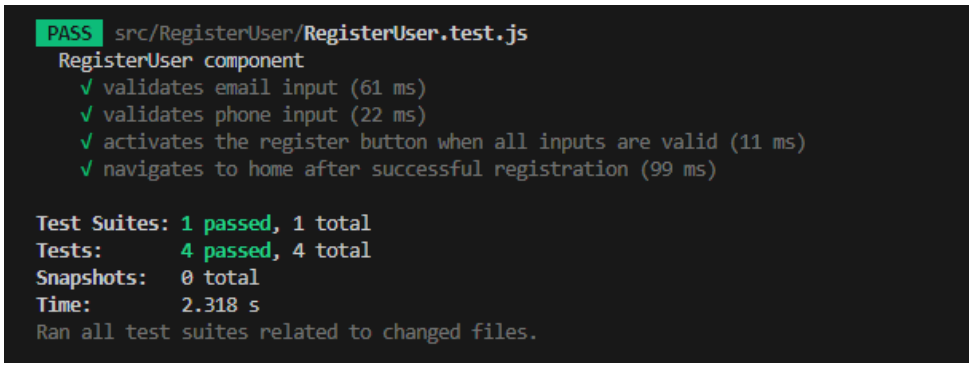


Figura 3.10: Pruebas unitarias componente *RegisterUser*

[3]Componente que simula el enrutamiento del navegador en un entorno de prueba, permitiendo un control completo sobre el historial de navegación sin depender del navegador real.

3.6.2. Pruebas de aceptación

Las pruebas de aceptación del asistente nutricional fueron realizadas en un entorno empresarial por el codirector del TFG y miembros del equipo de Inteligencia Artificial Generativa. En el Cuadro 3.2 se detallan los casos de prueba y los resultados obtenidos.

Cuadro 3.2: Resultados de las pruebas de aceptación

Resultados de las pruebas de aceptación del asistente nutricional		
Funcionalidad	**Caso de Prueba**	**Resultado**
Registro de usuario	CP-01: Registro exitoso con datos válidos	Conforme
	CP-02: Error al dejar campos obligatorios en blanco	Conforme
	CP-03: Error al ingresar un correo electrónico incorrecto	Conforme
Inicio de sesión	CP-04: Inicio de sesión con credenciales correctas	Conforme
	CP-05: Error al ingresar contraseña incorrecta	Conforme
	CP-06: Error al intentar iniciar sesión con correo no registrado	Conforme
Creación de plan nutricional	CP-07: Creación de plan con datos completos	Conforme
	CP-08: Error al omitir información importante	Conforme
Análisis de las calorías	CP-09: Análisis correcto de alimentos	Conforme
	CP-10: Error al fotografiar cualquier otro objeto	Conforme

3.6.3. Pruebas de rendimiento

En este apartado se presenta el análisis de rendimiento de la aplicación web, que consistió en capturar una foto y enviarla al modelo de GPT-4 para estimar las calorías de los alimentos presentes en la imagen. Como se puede ver en la Figura 3.11, se realizó una prueba de rendimiento que abarca un intervalo de 7.207 segundos, desglosando el tiempo empleado en diversas actividades. La ejecución de *scripts* consumió 34 ms, mientras que la representación de elementos visuales tomó 25 ms y el renderizado de píxeles en la pantalla, 18 ms. Las operaciones del sistema, que incluyen tareas de bajo nivel, ocuparon 123 ms. Sin embargo, la mayor parte del tiempo, 7006 ms, correspondió a tiempo inactivo, indicando que la aplicación estuvo esperando, esto es debido a las operaciones de red necesarias para enviar la imagen y recibir la respuesta del modelo GPT-4. El alto tiempo de inactividad, representando un 97.21 % del tiempo total, tiene un impacto significativo en la aplicación. Este prolongado tiempo de espera puede afectar negativamente la experiencia del usuario, haciendo que la aplicación parezca lenta o no reactiva. Para solventar el impacto del tiempo inactivo y mejorar el rendimiento general de la aplicación, será necesario optimizar el código, sin embargo, al ser un prototipo inicial será una tarea a futuro.

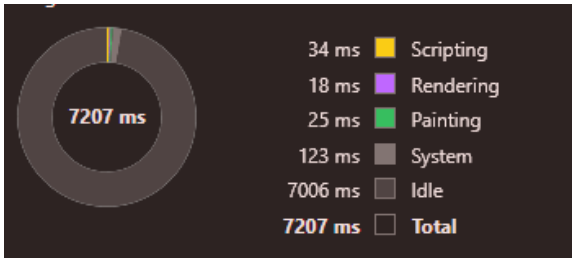


Figura 3.11: Prueba de rendimiento

3.6.4. Pruebas de accesibilidad

El análisis de accesibilidad se realizó utilizando la herramienta WAVE (*Web Accessibility Evaluation Tool*) para evaluar todas las interfaces de la aplicación (véanse en las Figuras 3.12, 3.13, 3.14), sin embargo, se va a detallar a continuación el análisis de la interfaz del menú semanal. El análisis de WAVE reveló un total de diez errores de contraste, seis alertas, seis características positivas de accesibilidad, tres elementos estructurales y veintitrés implementaciones de atributos *Accessible Rich Internet Applications* (ARIA)[4]:

- **Errores de contraste:** se identificaron diez errores de contraste en la página. Estos errores indican que las combinaciones de colores de texto y fondo no cumplen con las pautas de contraste mínimo recomendadas por las WCAG. Un contraste insuficiente entre el texto y el fondo puede dificultar la lectura del contenido para usuarios con baja visión o daltonismo. Para mejorar la accesibilidad, es necesario ajustar los colores para que el contraste cumpla con los estándares de WCAG, que recomiendan un ratio mínimo de 4.5:1 para texto normal y 3:1 para texto grande.
- **Alertas:** se detectaron seis alertas durante el análisis. Estos problemas pueden incluir el uso inapropiado de elementos HTML o la falta de ciertos atributos que podrían mejorar la accesibilidad.
- **Características de accesibilidad:** WAVE identificó seis características positivas de accesibilidad en la página. Estas características incluyen implementaciones que mejoran la accesibilidad del sitio, como el uso correcto de etiquetas ARIA y elementos HTML semánticos.
- **Elementos estructurales:** se encontraron tres elementos estructurales en la página. Estos elementos, como encabezados, listas y secciones, son importantes para la organización y estructura del contenido. Una correcta estructuración del contenido no solo mejora la accesibilidad, sino que también facilita la navegación y comprensión de la información por parte de todos los usuarios.
- **Implementaciones ARIA:** el análisis identificó veintitrés implementaciones de atributos ARIA. Los atributos ARIA proporcionan roles, propiedades y estados que describen la estructura y el comportamiento interactivo de la página a los usuarios de tecnologías asistidas.

[4]Es una colección de atributos que definen como realizar contenido y aplicaciones web más accesibles para las personas con discapacidades.

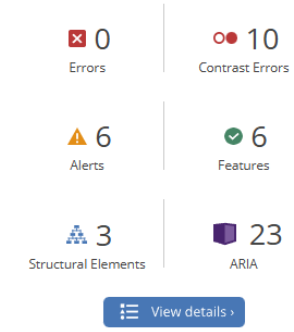


Figura 3.12: WCAG menú semanal

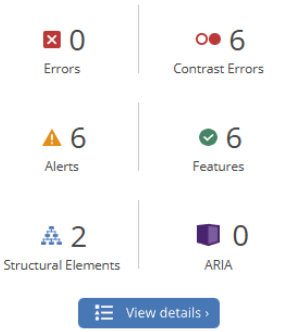


Figura 3.13: WCAG selección meta

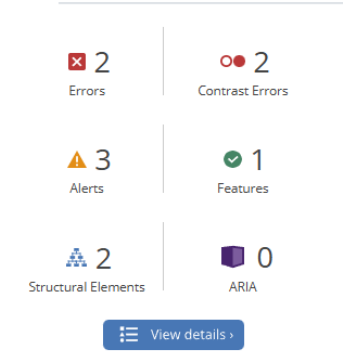


Figura 3.14: WCAG inicio sesión

El análisis reveló áreas específicas donde se necesitaban ajustes para mejorar la accesibilidad de todas las interfaces, permitiendo corregir estos errores de contraste e implementar de forma correcta prácticas de accesibilidad y atributos ARIA. La corrección de estos errores implicó modificar el estilo y el tamaño de las fuentes, con el objetivo de mejorar la accesibilidad para personas con discapacidad. Tras las respectivas modificaciones, se volvieron a realizar las pruebas obteniendo el resultado mostrado en las Figuras 3.15, 3.16 y 3.17.

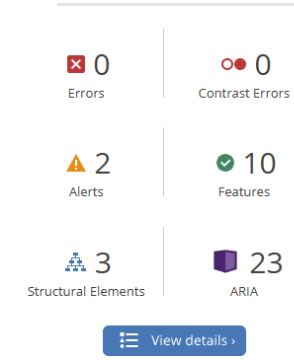


Figura 3.15: WCAG menú semanal (solucionado)

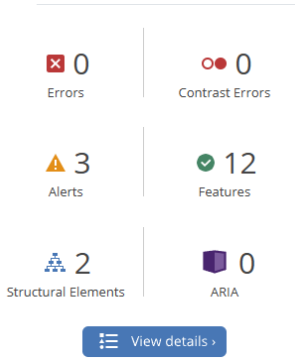


Figura 3.16: WCAG selección meta (solucionado)

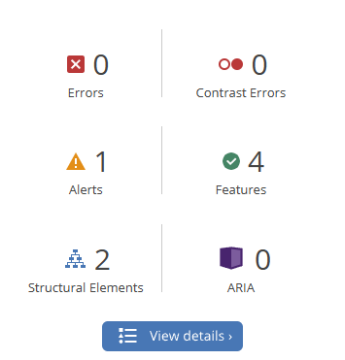


Figura 3.17: WCAG inicio sesión (solucionado)

Capítulo 4

Desarrollo de funcionalidades con IA Generativa

Este proyecto se enfoca en el desarrollo de funcionalidades que requieren Inteligencia Artificial (IA) Generativa. A lo largo de esta sección se especifican todas aquellas funcionalidades que se llevaron a cabo, en base a la definición teórica del *backend*. Los beneficios clave de emplear IA son:

- **Personalización:** los menús y preparaciones se adaptan a las necesidades y preferencias de cada usuario.
- **Eficiencia:** se automatiza la generación de menús, listas de compra y preparaciones.
- **Información nutricional:** se facilita el control de las calorías y la alimentación saludable.

La primera fase se dedica a tareas previas para asegurar que los modelos generativos produzcan resultados coherentes y de alta calidad. Estas tareas son el preprocesamiento del texto del libro de recetas y la construcción del modelo RAG. Estas dos etapas son fundamentales para garantizar la precisión, eficiencia y calidad de los menús personalizados.

- **Procesamiento y *chunking*:** el texto del libro de recetas se divide en secciones semánticas más pequeñas llamadas fragmentos o *chunks*.
- ***Embeddings*:** se aplica esta técnica para convertir el texto de cada fragmento o *chunk* en vectores numéricos, capturando la semántica del texto.
- **Construcción del modelo RAG:** se crea el modelo RAG a partir de los embeddings, combinando la generación de texto con la recuperación de información relevante del libro.

Como se mencionó anteriormente, en este proyecto se desarrollaron aquellas funcionalidades que emplean IA Generativa, siendo estas:

- **GET /menu/personalizado:** la IA Generativa se aplica a la generación de menús personalizados, utilizando diferentes técnicas de *prompting* (véase la Sección 3.5.1) con los datos proporcionados por el usuario.
- **GET /menu/preparacion:** se genera la preparación de cada plato con las cantidades necesarias de ingredientes y las instrucciones para su elaboración, también utilizando técnicas de *prompting*.

- **GET /logistica/listacompra:** genera la lista de compra para los ingredientes del menú.
- **GET /menu/alternativa:** ofrece alternativas a los platos del menú personalizado.
- **GET /usuario/fotoPlato:** se utiliza el modelo GPT Visión Preview para identificar las calorías en los platos.

4.1. Fundamentos teóricos

La aplicación no solo se limita a proporcionar recomendaciones genéricas, sino que se adapta a las necesidades de los usuarios (generación menús personalizados, reconocimiento de imágenes para la identificación de calorías y generación de listas de compra). Este apartado se centra en tres elementos fundamentales para la generación: el RAG, el *chunking* y los *embeddings*. Estas técnicas de procesamiento del lenguaje natural (NLP) permiten una mejor interpretación semántica y sintáctica de la información de interés para el Asistente Nutricional. Además, se analizará cómo se integran dentro del marco de la aplicación y cómo contribuyen a su funcionamiento. Desde la comprensión de las preferencias del usuario, se procede a la generación de recomendaciones de menú personalizados.

4.1.1. RAG

El RAG es una técnica que se utiliza en el procesamiento del lenguaje natural, combinando la potencia de los modelos basados en la recuperación (*retrieval*) y los modelos generativos, mejorando la relevancia del texto generado.

Los modelos de recuperación están diseñados para extraer información relevante de una colección de documentos o base de conocimientos. Utilizan técnicas como la recuperación de información y búsqueda semántica para identificar la información más relevante en respuesta a una consulta. Su objetivo principal es encontrar fragmentos de texto que sean adecuados a una pregunta o contexto dado.

Los modelos generativos están diseñados para crear nuevo contenido basado en un mensaje. Estos modelos, como los LLM, aprenden patrones y estructuras del lenguaje natural a partir de grandes conjuntos de datos de entrenamiento. Generan texto nuevo que sigue las estructuras y estilos del lenguaje aprendido durante el entrenamiento.

El modelo utiliza la información recuperada para mejorar la generación de texto, incorporando datos relevantes en el proceso para producir resultados más precisos. Esta combinación permite que el modelo generativo se beneficie de la precisión y especificidad del modelo de recuperación. Los modelos pueden entrenarse desde cero o ajustarse a conjuntos de datos específicos y dominios de aplicación mediante técnicas de ajuste fino. Esto permite adaptarlos a contextos particulares.

En la Figura 4.1 se representa el proceso de RAG. A continuación, se detallan los seis pasos a seguir:

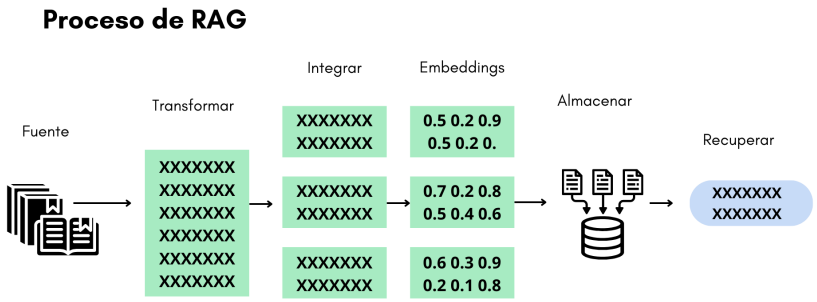


Figura 4.1: Proceso modelo RAG

El proceso RAG comienza con la recuperación de datos de una fuente. Estos datos sirven como base para generar nuevo texto. La imagen muestra un ejemplo de datos que se cargan desde una fuente, representada por el cuadro «Fuente». Una vez recuperados los datos, se transforman para prepararlos para su posterior procesamiento. Esto puede implicar limpiar y normalizar los datos con contexto adicional. El cuadro «Transformar» en la imagen representa esta etapa de transformación. Los datos transformados se integran en una representación vectorial, lo que permite que los modelos de aprendizaje automático los procesen de manera eficiente. Los cuadros «Integrar» y «Embeddings» en la imagen representan este paso. Los datos integrados se almacenan en una base de datos vectorial. El cuadro «Almacenar» en la imagen representa esta etapa de almacenamiento. Al generar nuevo texto, el proceso recupera datos integrados relevantes del almacén de vectores. Estos datos recuperados proporcionan el contexto y la información necesarios para guiar el proceso de generación. El cuadro «Recuperar» en la imagen representa este paso de recuperación. La etapa final del proceso implica generar nuevo texto basado en los datos integrados recuperados. Esto implica utilizar modelos de lenguaje y técnicas de aprendizaje automático para producir texto coherente y significativo.

Como se mencionó anteriormente, RAG aprovecha la información recuperada por los modelos de recuperación para mejorar la generación de texto. En el diagrama de la Figura 4.2 se describen las tres principales etapas: **recuperación**, **generación** y **respuesta**.

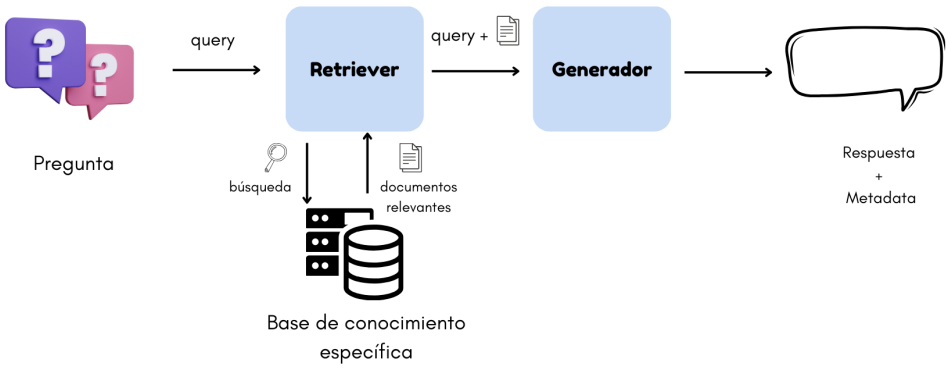


Figura 4.2: Modelo generativo

En la primera etapa, el recuperador es responsable de encontrar los documentos relevantes en la base de datos. Para ello, utiliza una consulta que especifica qué tipo de documentos se buscan. La consulta puede incluir palabras clave, frases o incluso conceptos más complejos. Una vez que el recuperador ha encontrado los documentos relevantes, los pasa al generador. El generador es responsable de generar una respuesta a la consulta utilizando los documentos recuperados. La respuesta puede ser un resumen, una lista de los documentos más relevantes o un nuevo documento que sintetice la información recuperada.

4.1.2. *Chunking*

El *chunking* es una técnica fundamental en el procesamiento del lenguaje natural. Su función consiste en dividir grandes volúmenes de datos en partes más pequeñas y manejables a las que llamaremos *chunks*. Estos *chunks* no tienen una extensión fija, pueden ser frases, cláusulas, palabras que estén relacionadas semánticamente o cualquier otra unidad lingüística que tenga un significado cohesivo dentro del contexto. Existen diferentes tipos de *chunks*, cada uno con aplicaciones diferentes.

Dentro de nuestro contexto de un asistente nutricional, el *chunking* se vuelve una tarea importante para poder comprender la información. Como se mencionó anteriormente, un *chunk* no tiene una extensión fija, pero sí debe tener un significado cohesivo. La aplicación usa un libro de recetas que aporta diferenciación frente a otras aplicaciones del mercado actual. A lo largo de esta sección se expondrán diferentes técnicas/extensiones de *chunk*, donde se determina cuál es la mejor para cada fuente de conocimiento que se emplea.

Podemos identificar estos cinco principales tipos de *chunks* [4] con sus respectivas aplicaciones:

1. **Fragmentación de ventana de token máximo sin superposición:** el documento se divide en ventanas de tamaño fijo, cada una representando un fragmento de documento separado. No hay superposición entre las ventanas.

    - **Aplicaciones:**
        - **Análisis de sentimientos:** dividir reseñas de productos en fragmentos para analizar opiniones de manera más enfocada.
        - **Procesamiento de lenguaje natural:** dividir documentos largos en partes manejables para tareas como clasificación de texto o extracción de información.

2. **Fragmentación de ventana de token máximo con superposición:** similar al primero, pero las ventanas tienen superposición, lo que significa que parte del contenido se duplica en fragmentos adyacentes.

    - **Aplicaciones:**
        - **Detección de temas:** al permitir superposición, se pueden capturar mejor las transiciones entre temas en un conjunto de documentos.
        - **Resumen de texto:** superponer fragmentos para asegurar que las partes más importantes del texto estén presentes en múltiples fragmentos, mejorando la calidad del resumen.

3. **Fragmentación de delimitadores naturales:** se utilizan los espacios en blanco naturales en el documento para determinar los límites de cada fragmento. Por ejemplo, los párrafos o las oraciones pueden ser los delimitadores naturales.

- **Aplicaciones:**
  - **Análisis de documentos estructurados:** utilizar los delimitadores naturales como párrafos o secciones para segmentar documentos técnicos o académicos.
  - **Extracción de información:** identificar y extraer fragmentos relevantes de documentos basados en patrones de delimitadores naturales como listas o tablas.

4. **Clustering para crear documentos semánticos:** en lugar de dividir el documento en fragmentos basados en características estructurales o de formato, se agrupan fragmentos similares para formar documentos semánticos más grandes. Esto se hace comúnmente mediante técnicas de clustering que identifican similitudes semánticas entre fragmentos.

   - **Aplicaciones:** tiene diversas aplicaciones, incluyendo la organización y resumen automático de grandes conjuntos de documentos, la identificación de temas principales en colecciones de textos, y la creación de índices temáticos para facilitar la recuperación de información.

5. **Documentos completos sin fragmentación:** el documento entero se trata como un único *chunk* y no se divide en fragmentos más pequeños. Esto puede ser útil en ciertos casos donde se requiere análisis del documento en su totalidad sin fragmentación.

   - **Aplicaciones:**
     - **Traducción automática:** al trabajar con documentos completos, se pueden aplicar modelos de traducción automática a textos enteros, en lugar de fragmentos individuales, para mantener la coherencia y el contexto.
     - **Análisis de documentos legales:** analizar documentos legales completos para identificar cláusulas específicas o relaciones complejas entre secciones.

Tras analizar las diferentes opciones de *chunks*, se determinó explorar la fragmentación de ventana de token máximo con superposición.

Este tipo de fragmentación es la más sencilla, pues permite que se pueda dividir un documento en segmentos de tamaño fijado. Al tener segmentos o *chunks* de un tamaño igual o similar el sistema será más consistente. Sin embargo, pueden ocurrir pérdidas de contexto al dividir la información en *chunks* diferentes. Para ello, se introduce el termino *overlap* o ventana superpuesta. El *overlap* es la cantidad específica de contenido que dos fragmentos vecinos comparten. Para determinar la cantidad de *chunks* de forma teórica que se han generado se ha empleado la siguiente fórmula:

$$n_{\text{chunks}} = \left\lceil \frac{text_{\text{length}}}{chunk_{\text{size}} - chunk_{\text{overlap}}} \right\rceil$$

Los parámetros que se han establecido son un *chunk_size* de 800 caracteres y un *overlap* de 100 caracteres. El *chunk_size* de 800 caracteres permite capturar segmentos de texto lo suficientemente grandes como para contener información coherente y significativa, como párrafos completos o secciones temáticas. Por otro lado, el *overlap* de 100 caracteres asegura que los *chunks* adyacentes compartan una porción de texto, manteniendo la coherencia semántica entre ellos y garantizando que ninguna información se pierda en las divisiones.

Después de realizar el *chunking*, podemos observar que ambos *chunks* comparten contexto (*overlap*), lo que permite mantener la información sin pérdidas, ya que siguen el mismo formato.

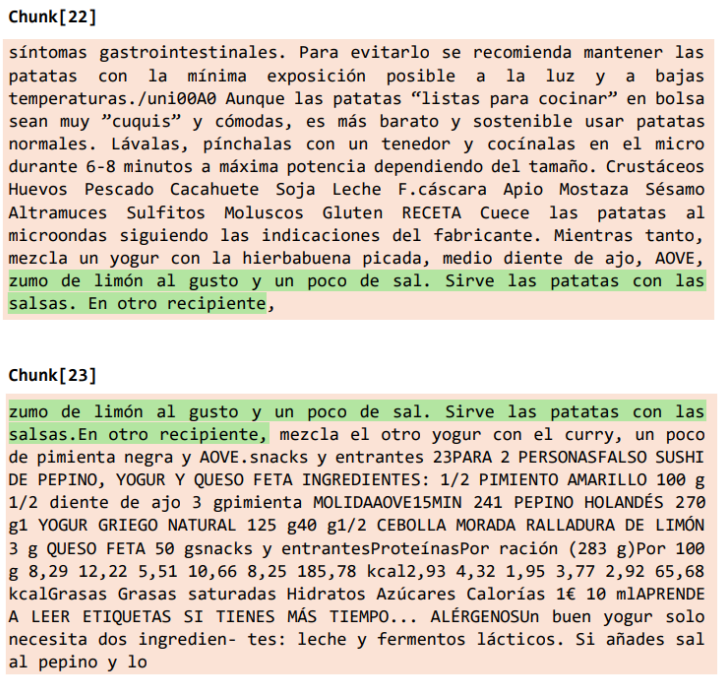


Figura 4.3: Chunking con overlap

4.1.3. *Embeddings*

Los *embeddings* son representaciones vectoriales de palabras o frases que capturan las relaciones semánticas y sintácticas entre ellas. La idea principal consiste en mapear cada unidad lingüística a un vector único en un espacio vectorial compartido. El uso de *embeddings* permite trabajar con textos de preparación de recetas para aprender estructuras y relaciones entre los alimentos. Se ha optado por el modelo *text-embedding-ada-002* [2], con el que se crean representaciones en un espacio semántico multidimensional. El motivo por el cual se ha elegido este modelo se debe a su rendimiento superior en la clasificación de textos y su capacidad para manejar documentos extensos gracias a una longitud de contexto ampliada a 8192 *tokens* (unidades básicas de texto). Además, produce *embeddings* más pequeños y eficientes en términos de cómputo, ocupando solo 1536 dimensiones. Finalmente, su capacidad para mapear palabras semánticamente similares a vectores cercanos mejora la precisión en la búsqueda y clasificación de recetas, facilitando la organización y acceso al contenido textual.

Para explicar de manera visual el concepto de *embeddings*, se ha generado la siguiente gráfica que representa diversas recetas proporcionadas por el Departamento de Salud Pública de California. En este gráfico, cada punto de color corresponde a una receta específica. La proximidad entre dos puntos refleja la similitud entre las recetas: cuanto más cercanos estén dos puntos, más parecidas serán las recetas; por el contrario, si la distancia entre ellos es mayor, las recetas serán menos parecidas entre sí.

Para facilitar la visualización y mantener las propiedades de estos *embeddings*, se utiliza el algoritmo t-SNE (Stochastic Neighbor Embedding t-distribuido) [6], que transforma los

datos originales en un espacio de dos dimensiones.

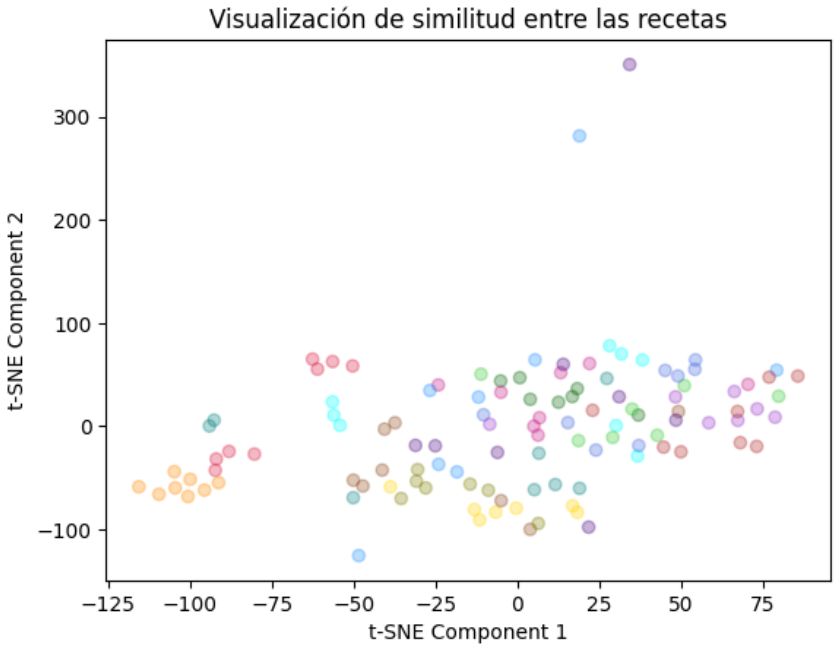


Figura 4.4: Visualización de vectores de recetas usando algoritmo de reducción de dimensionalidad

En el gráfico de la Figura 4.4, cada punto de color representa una receta del libro de recetas. Los colores de los puntos se asignan según las categorías que se muestran en el Cuadro 4.1:

Cuadro 4.1: Categorías

Categoría
Desayunos balanceados
Almuerzos de bajas calorías
Cenas fáciles de preparar
Postres deliciosos
Bocadillos y bebidas prácticos
Análisis de productos
Información nutricional
Consejos y tips de cocina
Recetas especiales
Acompañamientos
Guarniciones
Técnicas de cocina
Recetas internacionales
Recetas festivas
Recetas saludables

La técnica t-SNE se encarga de reducir las dimensiones de los datos de *embeddings* a dos dimensiones, etiquetadas como *Component 1* y *Component 2*. Esta reducción de

dimensiones conserva las relaciones de similitud entre las recetas. En términos prácticos, esto significa que recetas similares estarán representadas por puntos más cercanos entre sí en el gráfico. Por ejemplo, si dos puntos comparten el mismo color, indica que ambas recetas la misma categoría en el conjunto de datos original. Como se puede observar en la imagen, hay dos puntos aislados. Estos puntos pueden corresponderse a fragmentos del libro de recetas relacionados con el índice y por tanto no se corresponde con ninguna receta. Una forma de eliminar estos puntos aislados sería hacer un preprocesado de la base de conocimiento para eliminar aquella información irrelevante que no necesitamos. También pueden sugerir que estas recetas contienen ingredientes o preparaciones únicas que no se encuentran en otras recetas del conjunto de datos.

El propósito de esta visualización es facilitar la comprensión de cómo las recetas están relacionadas entre sí, utilizando una representación bidimensional que retiene las relaciones de similitud presentes en el espacio de alta dimensión original.

4.2. Clasificación de imágenes

4.2.1. Fundamentos teóricos

La clasificación de imágenes es fundamental en la aplicación, ya que permite al usuario escanear cualquier tipo de alimento y obtener información sobre sus calorías. Para ello, se utiliza el modelo GPT-4 Vision Preview, que combina el procesamiento de imágenes y la comprensión avanzada del lenguaje para interpretar datos visuales junto con la entrada textual. GPT-4 Vision se basa en el aprendizaje profundo y emplea redes neuronales con múltiples capas de nodos interconectados, imitando la estructura del cerebro humano. Esta arquitectura le permite procesar y comprender con eficacia extensos conjuntos de datos de imágenes y texto. OpenAI no ha especificado de forma concreta qué modelos usa, sin embargo, diversos artículos e investigaciones [1] concluyen que los principales componentes de GPT-4 Vision son:

Arquitectura *Transformer*: GPT-4 Vision utiliza esta arquitectura, que se basa en el manejo de datos secuenciales. Esta estructura es ideal para procesar tanto información textual como visual. Fue propuesta en un artículo de Vaswani et al. en 2017 llamado *Attention Is All You Need* [5]. Consta de dos componentes principales: el codificador y el decodificador. El codificador se encarga de procesar la información de entrada $z = (z_1 \dots z_n)$ y transformarla en una representación vectorial que captura las relaciones entre las palabras o elementos, mientras que el decodificador es responsable de generar la secuencia de salida $y = (y_1 \dots y_n)$ a partir de la representación generada por el codificador. En la Figura 4.5 se muestra un esquema general de la arquitectura.

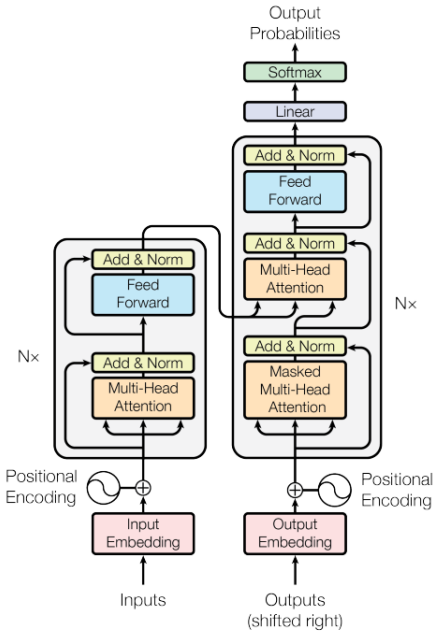


Figura 4.5: Arquitectura transformer [5]

El *encoder* (codificador), permite procesar cada palabra considerando su relación con las demás palabras de la secuencia, generando así un contexto específico para cada una. Por ejemplo, en una secuencia de datos como *"El paciente tiene una dieta alta en carbohidratos, lo cual podría afectar su nivel de glucosa"*, la capa de *self-attention* relaciona "lo cual" con "dieta alta en carbohidratos" en lugar de "paciente", proporcionando un contexto claro que identifica a qué información se refiere el pronombre. Esto permite a los *transformers* establecer relaciones entre los datos sin necesidad de procesarlos secuencialmente.

Un detalle importante es que, aunque los *decoders* pueden relacionar cada dato con los otros en la secuencia, no tenemos una manera de saber dónde está ubicado cada dato. Por eso, el artículo [5] también introduce los *positional encodings*. Los *positional encodings* tienen la misma dimensionalidad d que los *embeddings*, de esta forma pueden sumarse.

$$PE_{(\text{pos},2i)} = \sin\left(\frac{\text{pos}}{10000^{\frac{2i}{d}}}\right) \tag{4.1}$$

$$PE_{(\text{pos},2i+1)} = \cos\left(\frac{\text{pos}}{10000^{\frac{2i}{d}}}\right) \tag{4.2}$$

Estos forman una matriz donde cada vector corresponde a una posición en la frase. Así, todos los datos que entran al *transformer* se relacionan con una posición dentro de esta matriz, lo que permite que su *embedding* conserve información sobre su posición y su relación posicional con otros datos.

Esta matriz se genera usando funciones seno y coseno, lo cual es una gran ventaja ya que permite procesar secuencias de longitud variable, pudiendo aplicar el *transformer* a secuencias más largas que las del entrenamiento.

Una vez se obtienen estos *positional encodings* se suman a los *embeddings* de las palabras. Estos nuevos vectores enriquecidos con información de posición se pasan a través de las capas del *Transformer*.

Aprendizaje multimodal: la característica principal de GPT-4 Vision es su capacidad de aprendizaje multimodal. Esto significa que el modelo puede procesar simultáneamente tanto texto como imágenes. Esta capacidad le permite realizar una variedad de tareas complejas, como generar descripciones textuales detalladas de imágenes, responder preguntas específicas sobre contenido visual y crear imágenes a partir de descripciones textuales proporcionadas por el usuario. En este modelo, las imágenes se dividen en dos regiones principales y se obtienen sus *embeddings* tras combinar la salida de la red convolucional con *embeddings* posicionales.

Preentrenamiento y ajuste fino: GPT-4 Vision pasa por un proceso de entrenamiento en dos fases. Durante la fase de preentrenamiento, el modelo se entrena con grandes conjuntos de datos que incluyen tanto texto como imágenes. Este análisis le permite comprender y generar contenido en ambos formatos, estableciendo una base de conocimiento multimodal.

Una vez completado el preentrenamiento, GPT-4 Vision pasa por una fase de ajuste fino, donde el modelo se adapta específicamente al dominio de la aplicación. En el contexto de un asistente nutricional, por ejemplo, el ajuste fino se centra en perfeccionar las capacidades del modelo para tareas relacionadas con la nutrición, como la identificación de alimentos a partir de imágenes y la estimación de calorías basándose en descripciones textuales o visuales de los alimentos.

Las redes neuronales convolucionales (CNNs) se utilizan ampliamente en tareas de visión por computadora como la clasificación de imágenes debido a su capacidad para extraer características espaciales de las imágenes. El modelo GPT-4-Vision Preview emplea CNNs para extraer características relevantes de las imágenes de alimentos. En la Figura 4.7 se puede observar una representación detallada de este proceso. Las CNNs, con la ayuda de mecanismos de atención, son capaces de enfocarse en las partes más importantes de la imagen, lo que resulta esencial para un análisis preciso.

El proceso de extracción de características se divide en varias fases. En primer lugar, la imagen de alimentos se representa como una matriz de píxeles, tal como se ilustra en la Figura 4.6. Esta matriz actúa como la entrada para la red neuronal, permitiendo que la CNN procese y analice la imagen a nivel de píxel. A partir de esta matriz, la red neuronal convolucional realiza una serie de operaciones matemáticas que incluyen convoluciones, activaciones y agrupaciones, con el objetivo de detectar y realzar las características más relevantes de la imagen.

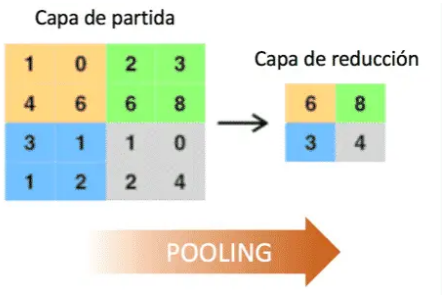


Figura 4.6: Matriz de píxeles [3]

Las convoluciones (CONV en la Figura 4.7) aplican filtros específicos a la matriz de píxeles para extraer patrones visuales, tales como bordes, texturas y formas. Cada capa de convolución en la CNN captura características de mayor complejidad a medida que se avanza en la red. Finalmente, las agrupaciones (o *poolings*) reducen la dimensionalidad de los datos, manteniendo las características más importantes y reduciendo la carga

computacional.

El uso de mecanismos de atención dentro de la CNN mejora aún más el proceso al permitir que la red neuronal se concentre en las áreas más significativas de la imagen. Estos mecanismos asignan diferentes pesos a distintas regiones de la imagen, priorizando aquellas que contienen información más relevante para la tarea de análisis de alimentos.

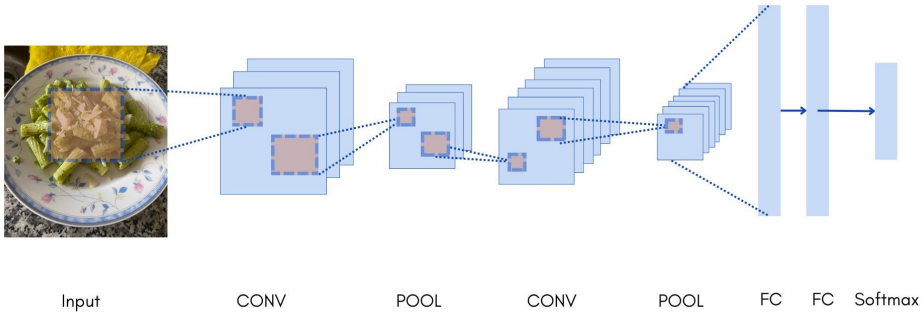


Figura 4.7: Funcionamiento red neuronal convolucional

Capítulo 5

Conclusiones y trabajos futuros

El nivel de desarrollo alcanzado en la aplicación ha permitido la creación de menús personalizados y la capacidad de tomar fotografías de platos de comida para determinar su contenido calórico.

Sin embargo, durante la fase de captura de requisitos, se identificaron varias características adicionales que aún están pendientes de implementación. Una de las funciones principales sugeridas fue la generación automática de listas de la compra basadas en los ingredientes de cada plato seleccionado. Esta característica adicional sería importante para facilitar la planificación de compras y la preparación de comidas.

Además, se ha planificado el desarrollo de botones de preparación de recetas dentro de Salvia. Estos botones tendrán la función de proporcionar a los usuarios instrucciones detalladas sobre cómo preparar cada plato de manera efectiva y precisa.

El desarrollo de Salvia ha supuesto un reto personal significativo, ya que ha implicado abordar múltiples campos como el desarrollo de *backend*, *frontend* y la implementación de funcionalidades con Inteligencia Artificial Generativa. Además, la realización del TFG en el entorno de una empresa me ha proporcionado una experiencia en trabajo en equipo, permitiéndome explorar diversas opciones de modelos de IA en un ambiente de investigación y compartir nuevos conocimientos dentro de la organización.

Durante la implementación, enfrenté diversos desafíos, especialmente en relación con la cantidad de tokens generados al utilizar varios libros de recetas. Este problema me llevó a concluir que, por cuestiones de costos, sería más viable usar inicialmente un solo libro. Sin embargo, se prevé que en el futuro se desarrollará una solución que permita ampliar la base de conocimientos.

Bibliografía

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Nilesh B Korade, Mahendra B Salunke, Amol A Bhosle, Prashant B Kumbharkar, Gayatri G Asalkar, and Rutuja G Khedkar. Strengthening sentence similarity identification through openai embeddings and deep learning. *International Journal of Advanced Computer Science & Applications*, 15(4), 2024.

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] Sinan Ozdemir. *Quick start guide to large language models: strategies and best practices for using ChatGPT and other LLMs*. Addison-Wesley Professional, 2023.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[6] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. *Distill*, 1(10):e2, 2016.

Apéndice

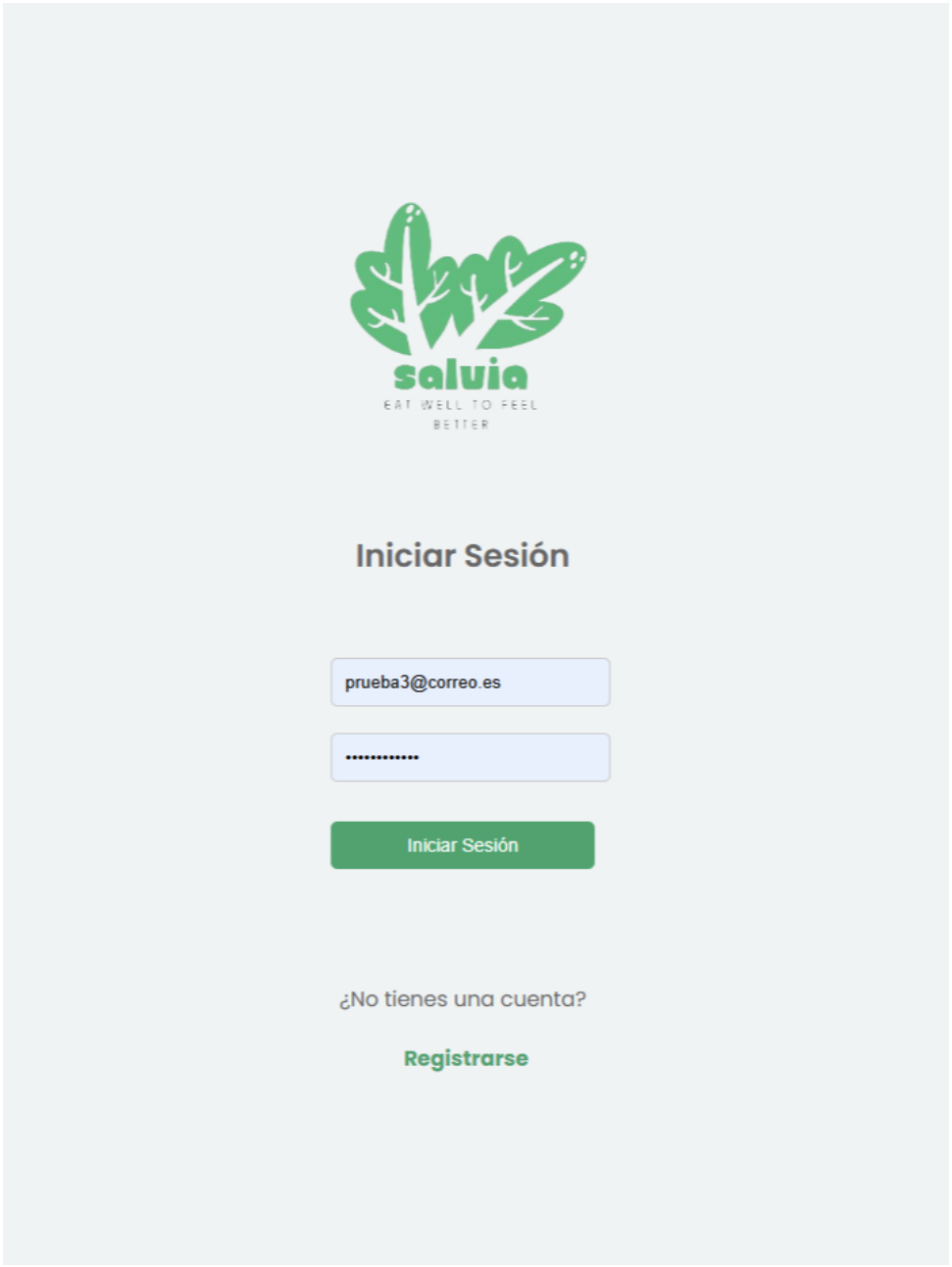


Figura 5.1: Inicio de sesión

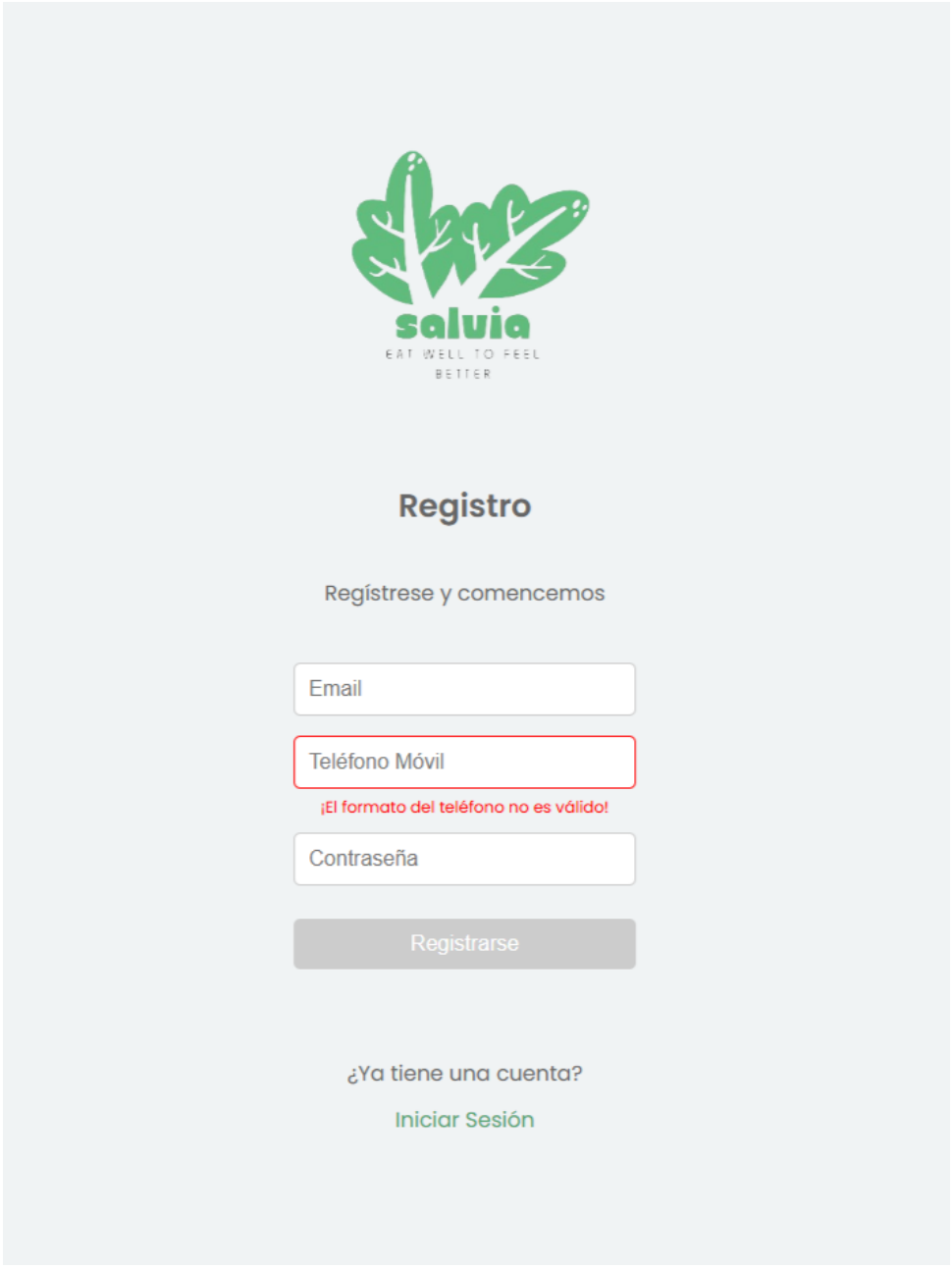


Figura 5.2: Registro

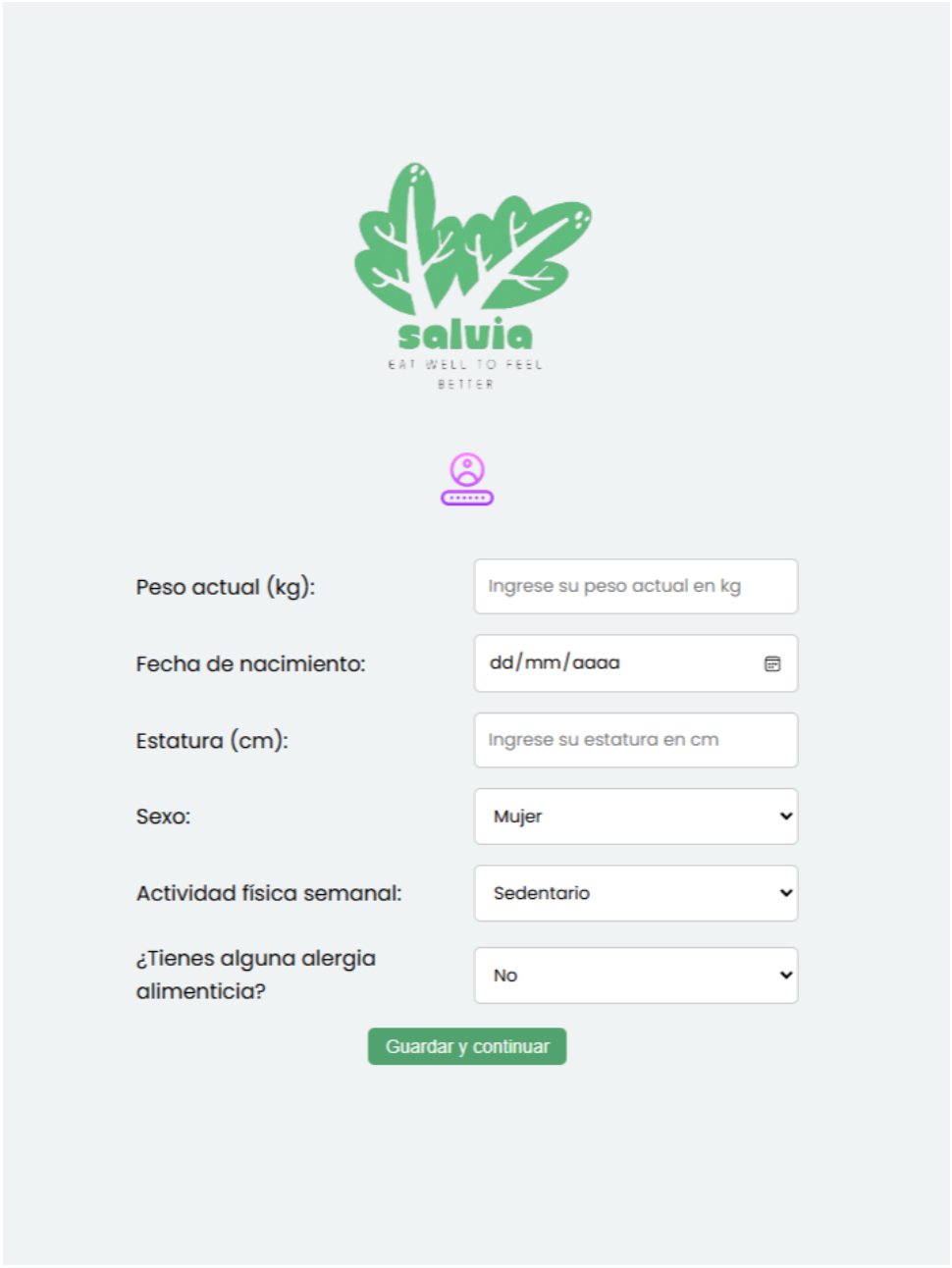


Figura 5.3: Ficha

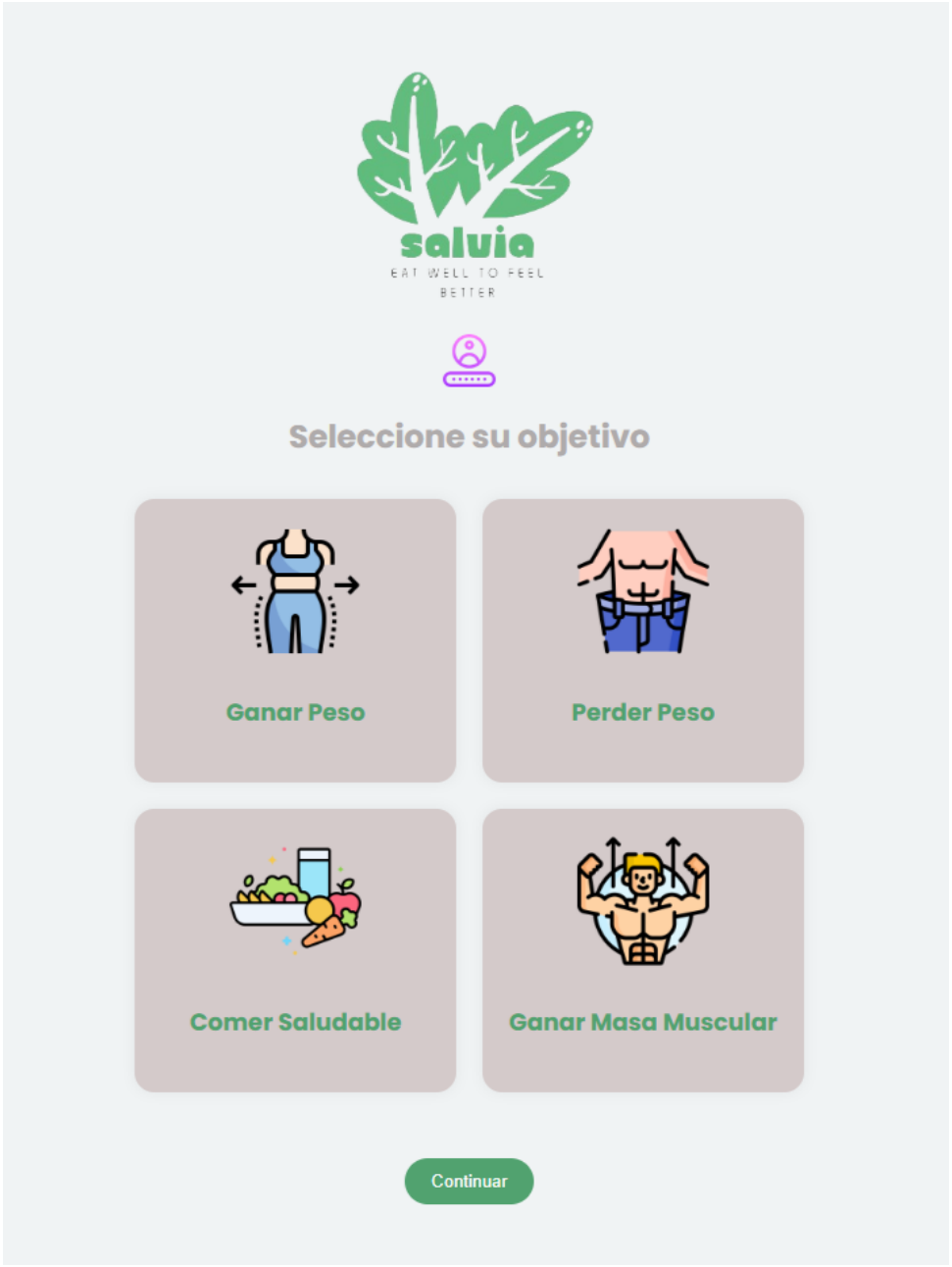


Figura 5.4: Selección de metas

Figura 5.5: Menú semanal

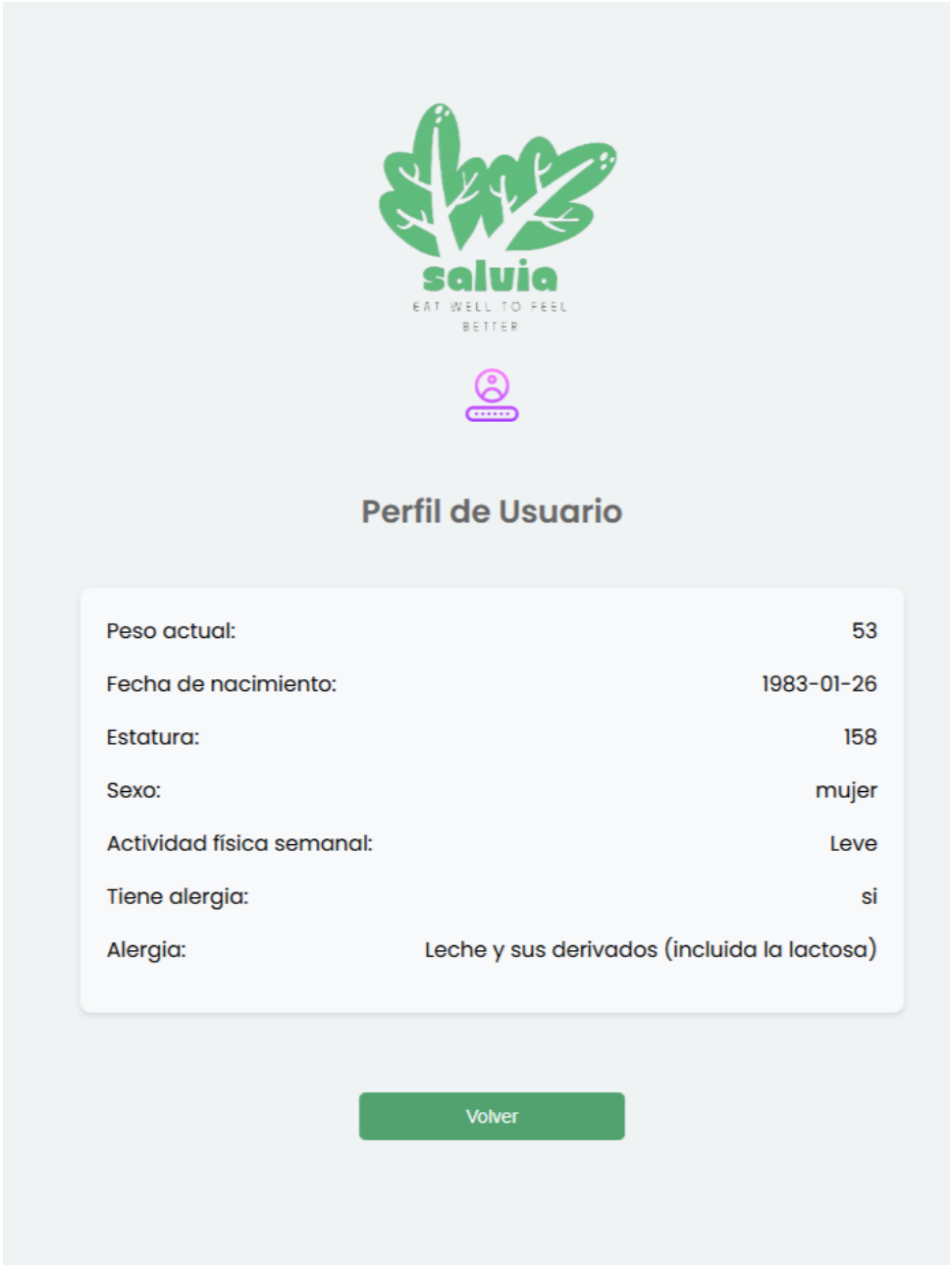


Figura 5.6: Ficha personal

Figura 5.7: Pizza parmesana con rúcula

```
1     'content': 'Pizza con rucula: aproximadamente 1100-1200
          calorias.'
```

Figura 5.8: Hamburguesa de ternera con queso cheddar y patatas fritas

```
1     'content': 'Hamburguesa con queso y papas fritas:
          aproximadamente 1000-1200 calorias'
```

Figura 5.9: Pops de pollo, patatas fritas y bolitas de queso

```
1       'content': 'Pops de pollo, papas fritas, cornballs de queso -
            aproximadamente 1000-1200 calorias.'
```

Figura 5.10: Nachos con guacamole y crema salvadoreña

```
1       'content': 'Nachos con guacamole y crema: alrededor de
            1050-1200 calorias.'
```

Figura 5.11: Espárragos a la vinagreta y mayonesa

```
'content': 'Esparragos con mayonesa y salsa pimiento -
    aproximadamente 250-300 calorias'
```

Figura 5.12: Pollo frito con pan y patatas fritas

```
'content': 'Pollo frito con pan y papas fritas: 1200-2000
    calorias aproximadamente.'
```