TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica Superior de Jaén

DESARROLLO DE UN PROTOTIPO DE APLICACIÓN MÓVIL PARA LA GESTIÓN DE RECETAS E INGREDIENTES CON COMUNIDAD EN LÍNEA

Autor: Juan Carlos Montilla Hernández

Grado: Ingeniería Informática

Director: Francisco de Asís Conde Rodríguez
Departamento del director: Departamento de Informática

Fecha: 03/07/2024

CREA

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Don Francisco de Asís Conde Rodríguez, tutor del Trabajo Fin de Grado titulado: '**Desarrollo de un prototipo de aplicación móvil para la gestión de recetas e ingredientes con comunidad en línea**', que presenta Don Juan Carlos Montilla Hernández, otorga el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, Julio de 2024

El alumno:

MONTILLA HERNANDEZ JUAN CARLOS - 78027507F

Firmado digitalmente por MONTILLA HERNANDEZ JUAN CARLOS - 78027507F
Fecha: 2024.07.03 13:33:24 +02'00'

Juan Carlos Montilla Hernández

El tutor:

CONDE RODRIGUEZ FRANCISCO DE ASIS - 24226298F

Firmado digitalmente por CONDE RODRIGUEZ FRANCISCO DE ASIS - 24226298F
Fecha: 2024.07.03 09:56:45 +02'00'

Francisco de Asís Conde Rodríguez

(Página intencionalmente en blanco)

Agradecimientos

Quisiera expresar mi más sincera gratitud a todas las personas que me han acompañado durante el desarrollo de este proyecto y a quienes siempre están a mi lado para apoyarme en lo que necesite.

En primer lugar, agradecer a mis padres y a toda mi familia por apoyarme incondicionalmente y ayudarme a convertirme en la persona que soy hoy.

A mi pareja y amigos, por acompañarme siempre, mostrándome su apoyo, animándome cuando era necesario y ayudándome a desconectar cuando era importante hacerlo.

También quiero agradecer al profesorado de la UJA, en especial a mi tutor Francisco, por brindarme la motivación para interesarme por el mundo del desarrollo de aplicaciones móviles y por su guía en este proyecto.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Proyecto de Ingeniería
Especialidad (solo TFG)	Tecnologías de la Información
Mención (solo TFG)	Sin mención
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	Juan Carlos Montilla Hernández
Fecha de asignación	22/11/2023
Descripción corta	Cada vez hay más gente preocupada por comer sano y barato, pero en ocasiones no dispone de los conocimientos necesarios ni de la motivación para dedicar el tiempo necesario. En este trabajo de fin de grado, se va a realizar un prototipo de aplicación móvil basada en Flutter y Firebase que permitirá a sus usuarios gestionar los ingredientes de su cocina de manera eficiente y acceder a una amplia base de datos de recetas compartidas por otros usuarios. Además, la app proporcionará una experiencia social, permitiendo a los usuarios compartir sus propias recetas, explorar las creaciones culinarias de otros y encontrar la motivación para dedicar tiempo a cocinar.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
Estas normas se han utilizado como base o referencia para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como proyecto la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.	

Contenido

Índice de ilustraciones

Índice de tablas

1 ESPECIFICACIÓN DEL TRABAJO

En este capítulo se presenta la especificación del trabajo, con una estructura y contenidos **inspirados** en los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparecen en el apartado 6 (Definiciones y abreviaturas).

1.1 Introducción

En este Trabajo de Fin de Grado se presenta una aplicación móvil desarrollada con Flutter que unifica múltiples funcionalidades de gestión de ingredientes y recetas en una sola plataforma. La motivación principal de este proyecto surge de la existencia de una gran cantidad de aplicaciones individuales que abordan diferentes aspectos de la gestión alimentaria, como la creación de listas de la compra, la búsqueda de recetas por ingredientes u otros filtros, el seguimiento de la caducidad de los alimentos y la obtención de información nutricional a través del escaneo de códigos de barras de los productos. Estas aplicaciones, aunque útiles, presentan el inconveniente de ser independientes, lo cual puede resultar tedioso para los usuarios que deseen usar varias funcionalidades y deban alternar entre ellas para hacerlo.

El objetivo de esta aplicación es simplificar y centralizar todas estas funcionalidades en una sola herramienta intuitiva y de fácil uso. Entre las características principales de la aplicación se incluyen la gestión de listas de la compra y de la despensa, con la posibilidad de mover ingredientes entre estas listas fácilmente, el seguimiento de la caducidad de los productos con alertas mediante notificaciones programables y personalizables, la búsqueda de recetas por ingredientes disponibles, nombre del plato o categorías, y la visualización de información nutricional mediante el escaneo de códigos de barras. Además, la aplicación permite a los usuarios compartir recetas, valorarlas, crear listas

personalizadas de recetas entre otras funcionalidades que ofrecen una experiencia completa y unificada.

1.2 Objetivos del trabajo

El principal objetivo de este trabajo es desarrollar una aplicación móvil integral que simplifique la gestión de la alimentación y la cocina diaria mediante la **unificación de múltiples funcionalidades** en una sola plataforma. Uno de los objetivos es crear un sistema de gestión de despensa que permita a los usuarios agregar y manejar los ingredientes disponibles en su hogar de forma fácil y versátil. Este sistema incluirá características como la ordenación de la lista por distintos criterios, la visualización de la fecha de caducidad, y la posibilidad de editar y borrar elementos.

Asimismo, se pretende desarrollar un sistema de gestión de listas de la compra que facilite la transferencia de ingredientes entre la despensa y la lista de compra, además de ofrecer la opción de compartir la lista a través de otras aplicaciones. Un objetivo adicional es implementar un sistema de lectura de códigos de barras que, conectándose a una API externa, permita **extraer información nutricional de los productos**. Esto agilizaría la adición de elementos a las listas ya que podría hacerse de forma mucho más rápida además de proporcionar una valoración nutricional rápida y un resumen de la información del producto.

Otro objetivo clave es incorporar un sistema de **seguimiento de la fecha de caducidad de los ingredientes**, proporcionando notificaciones programables y personalizables para alertar a los usuarios cuando un ingrediente esté próximo a caducarse, pudiendo elegir la hora de notificación y los días de antelación con los que se desea que se le avise. Además, se planea desarrollar una base de datos de ingredientes propia, con iconos asociados a cada ingrediente para mejorar el aspecto visual de la aplicación y facilitar el reconocimiento rápido y atractivo de los ingredientes.

La aplicación también incluirá una plataforma con algunas funcionalidades de red social, permitiendo a los usuarios **crear perfiles**, **subir recetas** y **valorar las recetas** de otros usuarios. Para mejorar la experiencia de búsqueda, se implementará

un sistema avanzado que permita buscar recetas mediante distintos métodos, **aplicar filtros por preferencias personales** y **clasificar automáticamente las recetas en categorías**. Los usuarios podrán **buscar recetas por ingredientes específicos**, ya sea ingresándolos manualmente o utilizando los ingredientes disponibles en la despensa con un solo clic. También se dispondrá de una pantalla con distintas categorías de recetas para explorar.

Acompañando a lo anterior, los usuarios podrán subir recetas a través de una pantalla de creación de recetas intuitiva, con múltiples opciones, etiquetas, enlaces a videos y diversa información, facilitando la posterior aplicación de filtros y clasificación en categorías. Se implementará un **sistema de filtros** basado en preferencias personales, incluyendo criterios de ordenación, filtrado de recetas con videos, tiempo de preparación y tipo de dieta.

Además, se permitirá a los usuarios **crear listas personalizadas de recetas**, similar a las listas de publicaciones en redes sociales, y desarrollar un sistema de perfil de usuario donde se **guarde online la información de las recetas con las que interactúan**, como recetas a las que se les ha dado me gusta o listas de recetas personales. Por último, se busca desarrollar una **interfaz agradable e intuitiva** para mejorar la experiencia del usuario.

1.3 Antecedentes y estado del arte

Actualmente, el mercado de aplicaciones móviles incluye una amplia variedad de herramientas diseñadas para diferentes aspectos de la gestión de alimentos y la cocina diaria. Es esencial evaluar estas aplicaciones existentes, ya que el proyecto que se trata de desarrollar tiene como objetivo consolidar todas estas funcionalidades en una sola plataforma.

Para un análisis más claro del estado del arte se han categorizado las aplicaciones actuales según sus funciones principales, de forma que se puedan tratar de forma individual.

1.3.1 Gestión de despensa y listas de la compra

En esta categoría es dónde más opciones podemos encontrar ya que es la funcionalidad más básica a la hora de pensar en aplicaciones para la gestión de alimentos. Precisamente por ser una funcionalidad tan básica, las aplicaciones que encontramos en el mercado tratan de **añadir características adicionales como valor añadido** para que al usuario final le merezca la pena su uso.

Entre las más famosas podemos encontrar algunas como *AnyList, Bring!*, *OurGroceries*, o *Listonic*:

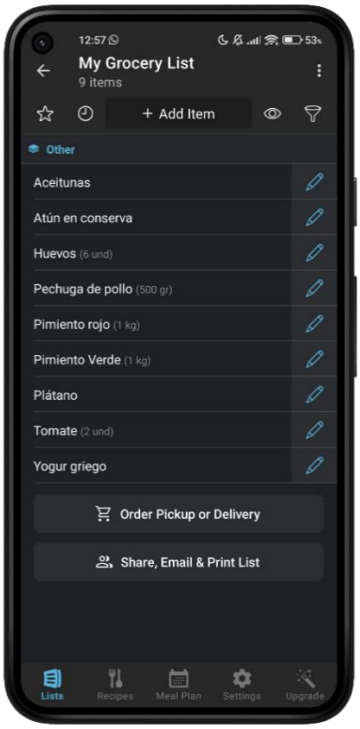

Ilustración 1.1: Captura de AnyList

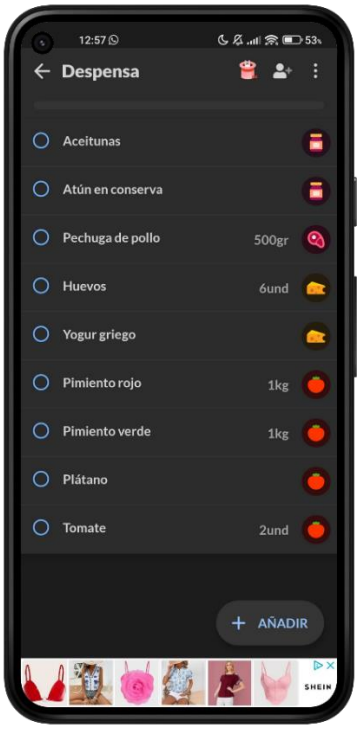

Ilustración 1.2: Captura de Listonic

Ilustración 1.3: Captura de OurGroceries

Ilustración 1.4: Captura de Bring!

Entre los mencionados valores añadidos podemos encontrar la clasificación de alimentos por categorías, posibilidad de creación de múltiples listas, ofrecer iconos asociados a las categorías de alimentos, compartir las listas o enviar las listas por mail entre otras muchas.

Uno de los problemas que encontramos es que, a menudo, se ofrecen tantas opciones y modos de clasificación que **pueden distraer al usuario de la tarea principal**, como sucede con AnyList. En otros casos, como Listonic, a pesar de tener una interfaz limpia y funciones centradas en su cometido principal, esto mismo limita otras funcionalidades clave, como mover alimentos entre distintas listas de forma sencilla o agregar fechas de caducidad. Además respecto a la adición de iconos respectivos a cada alimento vemos como es algo pobre ya que tan solo hay algunos iconos asociados a cada categoría, lo que hace que alimentos completamente distintos puedan llegar a compartir icono y puedan confundir al usuario. En este proyecto se desarrollará una base de ingredientes propias con iconos independientes asociados para cada uno de los ingredientes en lugar de aplicarlos de forma general para cada categoría.

Algo principal que se quiere tratar de solucionar en este proyecto y que podemos observar en la mayoría de casos disponibles es que todas las aplicaciones que encontramos **están centradas en una única función** de almacenar las listas de productos , explotando más o menos opciones, pero todas alrededor de la gestión de listas. De esta forma no se está aprovechando el potencial que puede ofrecer tener en una aplicación una lista de los alimentos que el usuario tiene en despensa para otras funcionalidades añadidas paralelas a la gestión de las propias listas, como son el desarrollar un sistema de búsqueda de recetas basado en los ingredientes disponibles en la despensa, la visualización de recetas con un contexto de los ingredientes que posee el usuario o no o la adición de funcionalidades de seguimiento y notificación de caducidad de los alimentos. Todas estas funciones tratarán de implementarse en este proyecto, tratando de lograr un sistema intuitivo, fácil de usar, no sobrecargado de opciones pero pudiendo explotar todo el potencial del sistema.

1.3.2 Seguimiento de caducidad de los alimentos

Respecto a la funcionalidad de gestionar la caducidad de los alimentos, aunque con menos alternativas que en la categoría anterior, también encontramos una gran variedad de aplicaciones en el mercado. En este caso **no encontramos grandes diferencias entre las distintas opciones**, en la mayoría de los casos se reduce a una lista dónde manualmente el usuario introduce los productos de su despensa y su respectiva fecha de caducidad y la aplicación se encarga de enviar notificaciones cuando esa fecha de caducidad se encuentre próxima.

Lo que sucede es que en todas estas aplicaciones se requiere que el usuario mantenga constantemente actualizada la lista de alimentos que tiene en despensa, registrando los que se van consumiendo y los que se adquieren nuevos. Este proceso es indispensable para poder controlar adecuadamente la función de seguimiento de caducidades. Sin embargo, mantener esta lista al día puede **suponer un trabajo considerable para el usuario**, especialmente si el beneficio se limita a recibir notificaciones sobre las fechas de caducidad.

Adicionalmente, si el usuario desea controlar los alimentos que necesita comprar y buscar recetas basadas en los ingredientes de su despensa, debe gestionar

listas en múltiples aplicaciones, lo que **aumenta significativamente el esfuerzo requerido**. Este proyecto busca solucionar estas deficiencias integrando el seguimiento de caducidades con la gestión de despensa, la creación de listas de la compra y la búsqueda de recetas. Así, el esfuerzo del usuario será más significativo y productivo al centralizar todas estas funcionalidades en una sola aplicación, optimizando el proceso de gestión de alimentos y mejorando la experiencia de usuario.

1.3.3 Análisis nutricional de los productos

En el ámbito del análisis nutricional de los productos alimenticios, existen varias aplicaciones destacadas que permiten a los usuarios escanear códigos de barras para obtener información detallada sobre los alimentos que consumen. Entre las más conocidas se encuentran *Yuka*, *Foodvisor* y *Coco*, las cuales han ganado popularidad por su capacidad para proporcionar datos nutricionales precisos y accesibles.

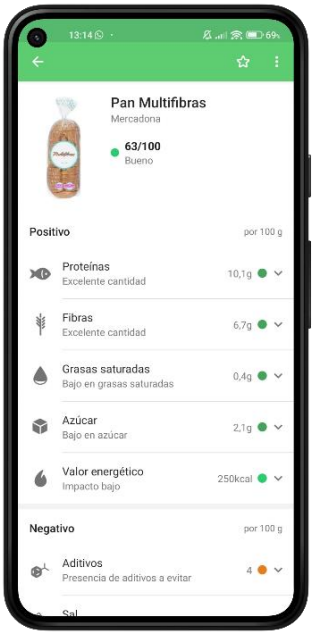

Ilustración 1.5: : Captura de Yuka

Ilustración 1.6: : Captura de Coco

Estas aplicaciones a menudo se basan en *Open Food Facts* [1], una base de datos abierta y colaborativa que recopila información sobre productos alimenticios de todo el mundo. *Open Food Facts* **permite a las aplicaciones acceder a una enorme**

cantidad de datos, facilitando así la tarea de evaluar la calidad nutricional de los alimentos. Muchas de las aplicaciones más populares utilizan directamente esta base de datos o implementan gran parte de su información en base a ella, lo que garantiza una cobertura amplia y actualizada de productos. Ya que se trata de una base de datos que cuenta con una API de calidad, abierta, gratuita y accesible para todo el mundo será la que se utilizará en este proyecto, de forma que se pueda ofrecer una valoración nutricional de los alimentos de una calidad similar a la mayoría de aplicaciones del mercado.

Un factor clave en la evaluación de alimentos en estas aplicaciones es el sistema *NutriScore*, una herramienta de clasificación que califica los productos alimenticios en función de su calidad nutricional. *NutriScore* asigna a cada producto una letra (de la A a la E), donde la 'A' representa la mejor calidad nutricional y la 'E' la peor. Este sistema proporciona una referencia rápida y fácil de entender para los consumidores, ayudándoles a tomar decisiones informadas sobre su alimentación. Encontramos una puntuación basada en estos criterios en la ya mencionada API, por lo que **será el principal criterio que se usará para calificar los alimentos** de forma general.

1.3.4 Búsqueda avanzada de recetas mediante ingredientes y filtros.

En el mercado actual, existe una gran variedad de aplicaciones dedicadas a la búsqueda de recetas. Sin embargo, la cantidad de aplicaciones que ofrecen una búsqueda avanzada, incluyendo la posibilidad de buscar recetas por ingredientes, tipo de dieta y otros filtros especializados, **es relativamente limitada**. Esta funcionalidad es especialmente valiosa, ya que permite a los usuarios encontrar recetas que se adapten específicamente a los ingredientes que tienen disponibles en su despensa, optimizando así el uso de recursos y reduciendo el desperdicio de alimentos.

Un aspecto relevante de la búsqueda de recetas por ingredientes es que requiere una base de datos bien estructurada, donde las recetas estén etiquetadas con precisión según los ingredientes. Esto garantiza que cuando un usuario realice

una búsqueda, las recetas mostradas coincidan con los ingredientes disponibles en su despensa.

Entre las aplicaciones más destacadas en este campo encontramos *Ekilu* (anteriormente conocida como *Nooddle*), *Cookpad* y *SuperCook*. Estas aplicaciones no solo permiten una búsqueda eficiente por ingredientes, sino que también integran otras funcionalidades avanzadas. *Ekilu*, en particular, se distingue por su interfaz intuitiva y su enfoque en ofrecer una experiencia de usuario cuidada y accesible que le ha llevado a una gran popularidad. Sin embargo, a pesar de su éxito, aún existen áreas de mejora, como la integración de múltiples filtros adicionales que no siempre están disponibles sin una suscripción.

Ilustración 1.7: Captura de Ekilu en la pantalla principal

Ilustración 1.8: Captura de Ekilu en la pantalla de selección de filtros

Además, la mayoría de las aplicaciones que permiten la búsqueda por ingredientes **no suelen incluir la gestión de una despensa de productos.** Esto es fundamental, ya que facilita al usuario ver qué recetas pueden preparar con los ingredientes que ya tienen, sin tener que introducir manualmente los ingredientes en cada búsqueda. *SuperCook* es una de las pocas aplicaciones que ha implementado

eficazmente un sistema de gestión de despensa, proporcionando una solución más integrada y eficiente para los usuarios, aunque carente de una interfaz tan intuitiva u otras funcionalidades como aplicar filtros de búsqueda, reduciendo su funcionalidad únicamente a la búsqueda por ingredientes de la despensa.

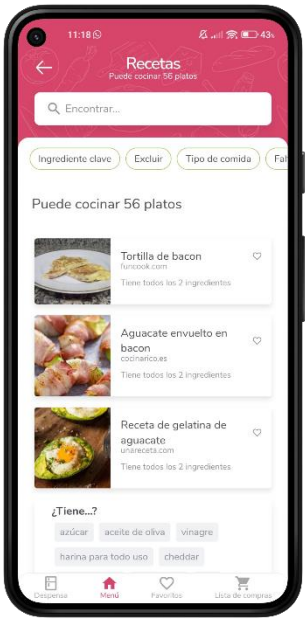

Ilustración 1.9: Captura de SuperCook

Otro aspecto clave a destacar es la conexión con redes sociales. En la actualidad, **las redes sociales se han convertido en el mayor escaparate para las recetas** subidas por usuarios [2], a menudo en formato de videos cortos que muestran la preparación de platos de manera rápida y visualmente atractiva. Integrar esta funcionalidad en una aplicación de recetas mejoraría la experiencia del usuario al permitir el acceso a videos explicativos. Esto beneficia tanto a los usuarios, que obtienen una mejor comprensión de las recetas, como a los creadores de contenido en redes sociales, quienes reciben mayor atención a sus publicaciones.

La aplicación que se desarrollará en este TFG pretende combinar las funcionalidades más destacables disponibles actualmente. Esto incluye la integración de múltiples filtros de búsqueda, la capacidad de buscar recetas por ingredientes disponibles en la despensa del usuario, y la conexión con redes sociales para permitir el acceso a contenido multimedia relacionado con las recetas. Al unificar estas

funciones en una sola aplicación, se busca proporcionar una herramienta más completa y útil para la gestión y planificación de comidas, optimizando el tiempo y los recursos de los usuarios de manera significativa.

1.4 Descripción de la situación de partida

En esta sección se describirá la situación de partida para proporcionar el contexto adecuado en el que se desarrollará este proyecto. Se detallarán los elementos condicionantes del trabajo, con el objetivo de comprender las razones y justificaciones de las características del software a desarrollar.

1.4.1 Descripción del entorno actual

A la hora de desarrollar una aplicación móvil se debe tener en cuenta tanto las condiciones de la temática de la aplicación a desarrollar en el mercado actual, tendencias actuales, las tecnologías que se usarán para la creación del propio software, las tecnologías backend en caso de ser necesarias como es este caso, costes o licencias asociadas a estas tecnologías y escalabilidad de estas. A continuación se realizará un análisis de cada una de estas características:

1.4.1.1 Fragmentación del mercado en aplicaciones sobre alimentación

Actualmente, **el mercado de aplicaciones móviles sobre alimentación está dividido**, con herramientas que se enfocan en funcionalidades específicas. Existen aplicaciones dedicadas exclusivamente a la gestión de despensa y listas de la compra, otras centradas en el seguimiento de la caducidad de los alimentos, y algunas más orientadas al análisis nutricional o la búsqueda de recetas por ingredientes. **Cada una de estas aplicaciones se ha especializado en su nicho,** ofreciendo características adicionales que buscan atraer a los usuarios, sin embargo, existen realmente pocas opciones que traten de abarcar una gestión más completa de los alimentos y recetas.

Muchas de estas aplicaciones gozan de gran popularidad, lo que evidencia que los usuarios encuentran útiles estas herramientas para gestionar sus necesidades diarias relacionadas con la alimentación. Esta especialización también significa que los usuarios a menudo deben utilizar múltiples aplicaciones para cubrir todas sus necesidades, lo que puede resultar en una experiencia fragmentada y menos eficiente.

La popularidad de estas aplicaciones demuestra su utilidad, por lo que una aplicación que consolide varias de estas funcionalidades en una sola plataforma **podría simplificar significativamente la gestión de alimentos para los usuarios**.

1.4.1.2 Uso de Flutter para el desarrollo de aplicaciones móviles en la actualidad

Flutter es un SDK (Software Development Kit) desarrollado por Google que permite la creación de aplicaciones móviles nativas de alta calidad para iOS y Android, entre otras plataformas, a partir de un único código fuente. Desde su lanzamiento, **Flutter ha experimentado un crecimiento significativo** y se ha consolidado como una de las principales herramientas para el desarrollo de aplicaciones móviles.

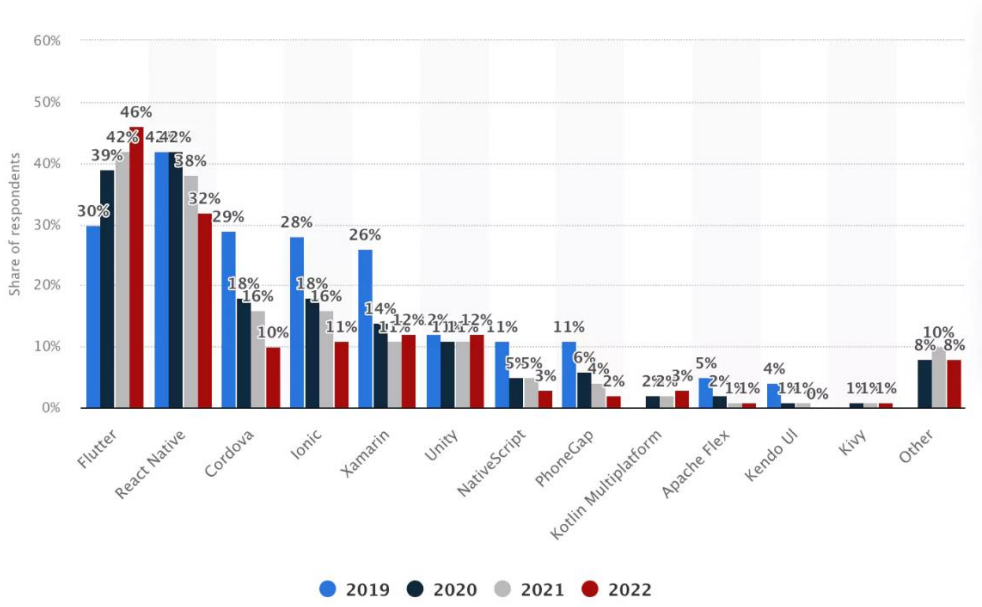

Ilustración 1.10: Marcos móviles multiplataforma utilizados por desarrolladores de software en todo el mundo de 2019 a 2022

Desde su lanzamiento en 2017, Flutter ha mostrado un crecimiento notable en su cuota de mercado. Según los datos recopilados [3], la adopción de Flutter ha aumentado de manera constante, superando a otros frameworks populares como *React Native*, *Cordova* e *Ionic*. En 2022, **Flutter se convirtió en el framework más utilizado por los desarrolladores**, alcanzando una cuota de mercado del 46%. Esta

evolución positiva se debe a varios factores clave que han impulsado su popularidad y aceptación en la comunidad de desarrollo.

Uno de los aspectos más importantes es el compromiso de Google con la actualización constante de esta tecnología. Las actualizaciones frecuentes no solo han introducido nuevas funcionalidades, sino que también han optimizado el rendimiento y la estabilidad del framework.

La comunidad de desarrolladores de Flutter ha crecido exponencialmente desde su lanzamiento. **Este crecimiento ha generado una gran cantidad de recursos**, incluyendo tutoriales, paquetes y bibliotecas, que facilitan el proceso de desarrollo. La colaboración y el intercambio de conocimientos dentro de esta comunidad han sido cruciales para el éxito de Flutter. Además, la disponibilidad de recursos comunitarios ha hecho que la curva de aprendizaje sea más accesible para los nuevos desarrolladores, fomentando una adopción aún mayor.

El respaldo de grandes empresas ha sido un factor determinante en la consolidación de Flutter como un framework de desarrollo móvil líder. Compañías como Google, Alibaba, eBay y Tencent han adoptado Flutter para el desarrollo de sus aplicaciones, lo que no solo ha aumentado su credibilidad, sino que también **ha demostrado su eficacia en aplicaciones de gran escala y alto rendimiento**. Esta adopción por parte de gigantes tecnológicos ha influido significativamente en la percepción y aceptación de Flutter en la industria.

Flutter ofrece múltiples **ventajas que lo diferencian de otros frameworks** de desarrollo móvil, destacándose por su capacidad de proporcionar un desarrollo rápido, productivo y de alto rendimiento. Una de las características más destacadas de Flutter es el Hot Reload, que permite a los desarrolladores ver los cambios en el código en tiempo real sin necesidad de reiniciar la aplicación. Esta funcionalidad acelera significativamente el proceso de desarrollo y depuración, permitiendo iterar y ajustar el código de manera eficiente.

Además, Flutter permite escribir **un único código fuente que se compila para múltiples plataformas**, tanto iOS como Android entre otras. Esto reduce considerablemente el tiempo y esfuerzo necesarios para desarrollar y mantener aplicaciones en ambas plataformas, optimizando los recursos y facilitando el manejo de versiones. La capacidad de Flutter para ofrecer un rendimiento nativo es otro de sus puntos fuertes ya que utiliza un motor de renderizado que dibuja widgets directamente en la pantalla, resultando en una experiencia de usuario fluida y responsiva. A diferencia de otros frameworks como React Native, que requieren puentes para comunicarse con componentes nativos, Flutter se compila directamente en código nativo, eliminando la sobrecarga de rendimiento y asegurando una ejecución más rápida y eficiente [4].

La flexibilidad y personalización de la interfaz de usuario es otro aspecto destacado de Flutter. Este ofrece una amplia gama de widgets personalizables, permitiendo a los desarrolladores crear interfaces de usuario atractivas y consistentes en varias plataformas de forma simultánea. Además del desarrollo móvil, Flutter ha extendido su soporte a plataformas web y de escritorio, permitiendo a los desarrolladores reutilizar el mismo código base para crear aplicaciones que funcionen en múltiples dispositivos y sistemas operativos, ampliando su alcance y eficiencia.

Flutter se basa en el lenguaje de programación Dart, también desarrollado por Google. Dart está optimizado para el desarrollo de interfaces de usuario, haciéndolo ideal para su uso en Flutter. La arquitectura de Flutter se centra en el uso de widgets, donde todo, desde los elementos de la interfaz de usuario hasta la estructura de la aplicación, se construye utilizando widgets. Estos pueden ser combinados y anidados para crear interfaces complejas y altamente personalizables.

Para integrar con servicios nativos, Flutter proporciona paquetes y bibliotecas que permiten la integración con APIs nativas de cada plataforma, como acceso a la cámara, geolocalización, almacenamiento local y más. Esto asegura que las aplicaciones desarrolladas con Flutter puedan aprovechar al máximo las capacidades de cada dispositivo.

1.4.1.3 Firebase como elección para backend online

Firebase, una plataforma desarrollada por Google, se ha convertido en una opción popular para el desarrollo de backend en aplicaciones móviles y web. La elección de Firebase como backend para una aplicación como la que se quiere desarrollar en este trabajo se justifica por varias razones clave, que van desde la facilidad de uso y la integración hasta la escalabilidad y el soporte de una amplia gama de servicios.

Uno de los principales atractivos de Firebase es su capacidad de **proporcionar una infraestructura robusta y escalable**. Para desarrolladores independientes y pequeños equipos, Firebase ofrece un plan gratuito con una generosa cantidad de recursos, lo que permite comenzar a desarrollar y desplegar aplicaciones sin incurrir en costos significativos. Este modelo de precios basado en el uso **se adapta perfectamente a proyectos en crecimiento**, permitiendo a los desarrolladores escalar sus aplicaciones sin preocuparse por costos iniciales elevados.

Firebase incluye una amplia gama de servicios que facilitan el desarrollo de aplicaciones complejas. Entre estos servicios se encuentran *Firebase Authentication*, que simplifica la gestión de usuarios y la autenticación segura, y *Cloud Firestore*, una base de datos NoSQL en tiempo real que permite sincronización en tiempo real y almacenamiento de datos escalable. Estas herramientas son especialmente útiles para aplicaciones de alimentación que requieren la gestión de perfiles de usuario, la personalización de contenido y la sincronización de datos entre múltiples dispositivos.

La documentación extensa y la comunidad activa de Firebase también juegan un papel crucial en su popularidad. Los desarrolladores pueden acceder a una gran cantidad de recursos, tutoriales y foros de discusión, lo que acelera el proceso de aprendizaje y resolución de problemas. La disponibilidad de estos recursos ayuda a los desarrolladores a **implementar rápidamente funcionalidades avanzadas**.

En términos de seguridad, Firebase proporciona una sólida infraestructura que incluye reglas de seguridad basadas en la nube y soporte para HTTPS, asegurando que **los datos de los usuarios estén protegidos contra accesos no autorizados**.

Además, el soporte para auditorías y monitoreo en tiempo real permite a los desarrolladores detectar y responder rápidamente a cualquier problema de seguridad.

1.4.2 Resumen de las deficiencias y carencias identificadas

La principal deficiencia identificada es la fragmentación de funcionalidades en múltiples aplicaciones. Esta multiplicidad de aplicaciones no solo incrementa la carga de trabajo del usuario, que debe navegar entre diferentes plataformas para realizar tareas relacionadas con la gestión de alimentos y la cocina, sino que también puede llevar a que muchos usuarios se priven de ciertas funcionalidades para evitar el esfuerzo extra. En lugar de utilizar varias aplicaciones para listas de la compra, seguimiento de caducidades y búsqueda de recetas, una plataforma unificada podría simplificar enormemente esta gestión.

Mejoras propuestas:

- **Unificación de funcionalidades**: Integrar la gestión de despensa, listas de la compra, seguimiento de caducidades y búsqueda de recetas en una sola plataforma.
- **Gestión integral de alimentos**: Utilizar la lista de alimentos disponible en la despensa para múltiples propósitos como la gestión de caducidad, la creación automática de listas de la compra y la búsqueda de recetas basadas en ingredientes disponibles.
- **Escaneo y análisis de productos**: Permitir a los usuarios escanear códigos de barras para analizar productos y añadirlos automáticamente a listas, combinando así el análisis nutricional con la gestión de despensa.
- **Facilidad de uso**: Proporcionar una interfaz intuitiva que simplifique la gestión de todas estas funciones, reduciendo el esfuerzo del usuario y mejorando la eficiencia del sistema.

1.5 Requisitos iniciales

Este proyecto trata de desarrollar una aplicación con múltiples funcionalidades que aunque se complementen entre sí, se tratarán como individuales para poder estructurar mejor la lista de requisitos iniciales:

1. Gestión de Despensa y Lista de la Compra
    - **Agregar y gestionar ingredientes**: La aplicación debe permitir a los usuarios añadir, editar y eliminar ingredientes de su despensa.
    - **Ordenación de la lista**: La aplicación debe permitir ordenar la lista de ingredientes por diferentes criterios como fecha de caducidad, alfabéticamente, o por cantidad.
    - **Visualización de fecha de caducidad**: La aplicación debe mostrar claramente la fecha de caducidad de cada ingrediente si el usuario asi lo quiere.
    - **Interacción entre despensa y listas de compra**: La aplicación debe permitir alternar ingredientes entre la despensa y la lista de la compra.
    - **Compartir listas**: La aplicación debería permitir la opción de compartir la lista de la compra con otras aplicaciones.

2. Lectura de Códigos de Barras e Información nutricional
    - **Conexión con API externa**: La aplicación debe integrar una API externa para extraer información nutricional de los productos escaneados.
    - **Agregar ingredientes rápidamente**: La aplicación debe permitir agregar ingredientes a las listas de despensa o compra mediante escaneo de códigos de barras.
    - **Valoración nutricional**: La aplicación debería mostrar una valoración nutricional y un resumen de la información nutricional de los alimentos escaneados siempre que el usuario lo desee.
3. Seguimiento de Fecha de Caducidad

- **Notificaciones programables**: La aplicación debe proporcionar notificaciones personalizables cuando un ingrediente esté próximo a caducarse.
- **Configuración**: La aplicación debe tener una interfaz que permita configurar fácilmente las notificaciones.

4. Base de Datos de Ingredientes
   - **Base de datos completa**: Se debe crear una base de datos de ingredientes propia que abarque la mayor parte de ingredientes y sus variantes posibles.
   - **Coherencia en la aplicación**: La base de datos de ingredientes debe ser usada en todas las funcionalidades (despensa, listas de compra, recetas) para mantener la coherencia.
   - **Visualización**: La base de datos debe asociar a cada uno de los ingredientes de los que dispone un icono de arte lineal para mejora de su interfaz.
5. Búsqueda y Gestión de Recetas
   - **Búsqueda avanzada**: La aplicación debe permitir distintos métodos de búsqueda y aplicar filtros por preferencias personales.
   - **Buscar por ingredientes**: La aplicación debe permitir la búsqueda de recetas por ingredientes específicos, ya sea manualmente o usando los ingredientes en la despensa.
   - **Clasificación automática**: La aplicación debe clasificar automáticamente las recetas en categorías basadas en ingredientes y otras características.
   - **Creación de recetas por usuarios**: La aplicación debe proporcionar múltiples opciones y etiquetas para ayudar a la creación de recetas detalladas, incluyendo enlaces a videos e información adicional.
6. Plataforma con algunas funcionalidades de red social
   - **Perfiles de usuario**: La aplicación debe permitir a los usuarios crear perfiles con nombre de usuario y foto de perfil.

- **Listas propias de recetas**: La aplicación debería permitir a los usuarios crear y gestionar listas de recetas personales, similar a las listas de publicaciones que se usan en redes sociales.
- **Interacción social**: La aplicación debe permitir dar me gustas a las recetas como forma de interacción con estas.

7. Interfaz de usuario

- **Interfaz agradable e intuitiva**: La aplicación debe tener una interfaz de usuario que sea visualmente atractiva y fácil de usar.
- **Consistencia visual**: La interfaz debe ser coherente en todas las funcionalidades de la aplicación.

1.6 Alcance

El alcance del proyecto incluye los siguientes entregables, tanto los productos finales como la documentación asociada:

1.6.1 Productos Finales

1. Prototipo de aplicación Móvil Funcional:

- **Archivos compilados (.apk)**: Una aplicación funcional en formato .apk lista para ser instalada en dispositivos móviles Android.
- **Funcionalidades incluidas**:
  - Gestión de despensa y listas de compra.
  - Lectura de códigos de barras y obtención de información nutricional.
  - Seguimiento de fechas de caducidad con notificaciones.
  - Base de datos de ingredientes con iconos asociados.
  - Plataforma de búsqueda y gestión de recetas.
  - Plataforma con perfiles de usuario y algunas funcionalidades de red social.
  - Interfaz de usuario intuitiva y agradable.

2. Código Fuente:

   - **Código fuente completo**: El código fuente de la aplicación desarrollada en Flutter, organizado y documentado adecuadamente para facilitar su comprensión y mantenimiento futuro.

1.6.2 Documentación de Gestión y Control

1. Documentación del Proyecto:

   - **Descripción completa de todo el proceso de desarrollo**, diseño del software, implementación y estructura del código.
   - **Instalación de la aplicación**: Instrucciones detalladas para la instalación de la aplicación móvil en dispositivos Android.
   - **Uso de la aplicación**: Guía completa para el uso de todas las funcionalidades de la aplicación, incluyendo la gestión de despensa, listas de compra, búsqueda de recetas y uso de la plataforma social.

1.7 Hipótesis y restricciones

El TFT se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento. Por consiguiente, la principal restricción aplicable es la limitación de la duración del trabajo.

Otra restricción significativa ha sido la infraestructura de backend. Dado que no se dispone de recursos para implementar y mantener un servidor propio, ni para costear un servicio externo, se optó por utilizar Firebase debido a su generosa cuota gratuita. Esta decisión condicionó el desarrollo de la aplicación, siendo necesario optimizar las lecturas, escrituras y almacenamiento en Firebase para evitar incurrir en costes adicionales. La aplicación se ha diseñado de manera que no suponga ningún coste mientras no escale más allá de ciertos límites establecidos por el plan gratuito de Firebase.

Además, el desarrollo para dispositivos iOS se ha visto limitado. Aunque Flutter permite crear aplicaciones para ambas plataformas con un único código base, la falta

de acceso a dispositivos MacOS e iOS impidió realizar pruebas en estos dispositivos. Por tanto, el desarrollo y las pruebas se realizaron exclusivamente en dispositivos Android, lo que restringe la garantía de funcionalidad y rendimiento de la aplicación en el ecosistema iOS.

1.8 Estudio de alternativas y viabilidad

Para el desarrollo de la aplicación móvil y su backend, se han considerado diversas alternativas con el objetivo de identificar la solución más adecuada que cumpla con los requisitos del proyecto en términos de rendimiento, facilidad de desarrollo, costos y mantenimiento. A continuación, se presentan las alternativas evaluadas para el desarrollo móvil y el servicio de backend.

Característica	Flutter	React Native	Desarrollo Nativo (IOS y Android)
Lenguaje Usado	Dart	JavaScript/TypeScript	Swift (iOS), Kotlin/Java (Android)
Rendimiento	Alto (cercano a nativo)	Bueno (ligeramente inferior por el puente JavaScript)	Excelente (optimizado para cada plataforma)
Consistencia de UI	Excelente (propio motor de renderizado)	Buena (depende de componentes nativos)	Excelente (acceso directo a controles UI nativos)
Velocidad de Desarrollo	Rápida (Hot Reload, código único para múltiples plataformas)	Rápida (Hot Reload, código compartido)	Moderada (desarrollo separado para cada plataforma)
Comunidad y Ecosistema	Rápido crecimiento, respaldado por Google	Amplia, establecida, respaldada por Facebook	Madura y estable, grandes comunidades de iOS y Android
Curva de Aprendizaje	Moderada (Dart es fácil de aprender pero menos conocido)	Fácil (JavaScript ampliamente conocido)	Moderada (conocimiento de Swift y Kotlin/Java)
Reutilización de Código	Alta	Alta	Baja (código separado para cada plataforma)
Integración con Funcionalidades Nativas	Buena (a través de plugins)	Excelente (acceso fácil a funcionalidades nativas)	Excelente (acceso nativo completo)

Cuota de Mercado	Crecimiento rápido	Estable, ampliamente utilizado	Estable, muy utilizado en aplicaciones de alto rendimiento
Facilidad de Pruebas y Depuración	Buena (herramientas de depuración incluidas)	Buena (herramientas de depuración)	Excelente (herramientas nativas de alto nivel)
Flexibilidad y Personalización	Alta (personalización a través de widgets)	Alta (personalización a través de componentes)	Alta (personalización completa nativa)
Soporte de Animaciones y Gráficos	Excelente (soporte integrado)	Bueno (depende de librerías adicionales)	Excelente (soporte nativo completo)
Coste de Desarrollo	Moderado (una sola base de código)	Moderado (una sola base de código)	Alto (dos equipos de desarrollo, uno para iOS y otro para Android)
Actualización y Mantenimiento	Fácil (un solo código base)	Fácil (un solo código base)	Complejo (dos bases de código separadas)
Accesibilidad a Recursos y Bibliotecas	Bueno (en crecimiento)	Excelente (amplia gama de bibliotecas)	Excelente (recursos nativos y bibliotecas maduras)

Tabla 1.1: Comparación de características entre Flutter, React Native y Lenguajes Nativos para desarrollo de aplicaciones móviles.

Flutter ha sido elegido para este proyecto debido a su alto rendimiento cercano al nativo, consistencia de interfaz de usuario, su flexibilidad, facilidad de uso, y su compatibilidad con otro tipo de servicios backend externos como Firebase. Para el tipo de aplicación que se trata de desarrollar Flutter cumple con todos los requerimientos por lo es la opción escogida.

Característica	Firebase	AWS Amplify	Supabase
Facilidad de Uso	Alta (fácil de integrar y usar)	Moderada (requiere configuración más compleja)	Alta (similar a Firebase, fácil de usar)
Costo	Generosa cuota gratuita, pago por uso	Pago por uso, puede ser costoso a gran escala	Generosa cuota gratuita, pago por uso
Escalabilidad	Alta (gestionada automáticamente por Google)	Alta (gestionada por AWS)	Alta (gestión manual requerida en algunos casos)

Soporte de Bases de Datos	Realtime Database, Firestore	DynamoDB, Aurora	PostgreSQL
Autenticación	Amplio soporte (Google, Facebook, Email, etc.)	Amplio soporte (OAuth, OpenID Connect)	Amplio soporte (similar a Firebase)
Funciones en la Nube	Cloud Functions (FaaS)	Lambda (FaaS)	Supabase Functions (basadas en PostgreSQL)
Integración con Herramientas de Desarrollo	Excelente (buenas herramientas y SDKs)	Buena (herramientas avanzadas pero más complejas)	Buena (herramientas amigables)
Documentación y Comunidad	Excelente (amplia documentación y comunidad)	Buena (extensa pero más técnica)	En crecimiento (documentación y comunidad activas)
Seguridad	Alta (gestión de seguridad avanzada)	Alta (gestión de seguridad avanzada)	Alta (basada en la seguridad de PostgreSQL)

Tabla 1.2: Comparación de características entre Firebase, AWS Amplify y Supabase

Firebase fue seleccionado como la plataforma backend para este proyecto debido a su facilidad de uso, generosa cuota gratuita, y capacidades escalables. Además, su excelente integración con Flutter y su soporte para diversas funcionalidades esenciales como autenticación, bases de datos NoSQL y funciones en la nube, lo hacen la mejor opción para este proyecto.

1.9 Descripción de la solución propuesta

La solución propuesta para este proyecto consiste en desarrollar una aplicación móvil integral utilizando Flutter y Firebase, con múltiples funcionalidades alrededor de la gestión de alimentos. Esta aplicación unificará características que actualmente se encuentran dispersas en diferentes aplicaciones, ofreciendo una plataforma completa y eficiente para los usuarios.

Las características más significativas de la solución propuesta son:

- **Gestión de Despensa**: Facilitará a los usuarios la adición, edición y eliminación de ingredientes en su despensa, permitiendo la ordenación

de la lista por diferentes criterios y visualización de las fechas de caducidad. Además, se incluirán notificaciones programables cuando los ingredientes estén próximos a caducar.

- **Listas de la Compra**: Permitirá alternar ingredientes entre la despensa y la lista de la compra, además de ofrecer la opción de compartir esta lista con otras aplicaciones.
- **Lectura de Códigos de Barras**: Integrará una API externa para la extracción rápida de información nutricional a través del escaneo de códigos de barras, facilitando la adición de productos a las listas.
- **Base de Datos de Ingredientes**: Contará con una base de datos propia, con iconos asociados a cada ingrediente para mejorar la coherencia y el reconocimiento visual dentro de la aplicación.
- **Características de red social**: Implementará funcionalidades sociales donde los usuarios podrán crear perfiles, subir recetas, y visualizar y valorar las recetas de otros usuarios, asi como crear listas personalizadas de recetas.
- **Búsqueda de Recetas**: Ofrecerá una búsqueda avanzada de recetas por ingredientes, filtros personalizados y clasificación automática en categorías.
- **Interfaz Intuitiva**: Diseñada para ser amigable y fácil de usar, mejorando significativamente la experiencia del usuario.

1.10 Tecnologías utilizadas

En este apartado se detallarán las tecnologías utilizadas para el desarrollo del proyecto. Debido al amplio número de dependencias y herramientas empleadas, se ha dividido en dos secciones: la primera se centrará en los softwares y recursos generales, mientras que la segunda abordará las dependencias específicas de Flutter utilizadas.

1.10.1 Softwares y recursos

Para el desarrollo de este proyecto, se han empleado diversas tecnologías y recursos, seleccionados por su capacidad de cumplir con los requisitos del proyecto y facilitar el proceso de desarrollo. A continuación, se describen las principales tecnologías utilizadas:

- **Flutter**: Este framework, desarrollado por Google, permite la creación de aplicaciones móviles nativas de alta calidad para iOS y Android a partir de un único código fuente. Flutter destaca por su capacidad para proporcionar una experiencia de usuario fluida y de alto rendimiento.
- **Firebase**: Plataforma backend elegida para este proyecto. Proporciona una infraestructura robusta y escalable con distintas herramientas que facilitan el desarrollo. Su integración sencilla con Flutter y su amplia cuota gratuita lo hacen una buena elección para este proyecto.
- **Dart**: El lenguaje de programación utilizado con Flutter. Dart es un lenguaje de alto nivel, moderno, fácil de aprender y proporciona un rendimiento excelente, ideal para el desarrollo de aplicaciones móviles. La arquitectura de Flutter se centra en el uso de widgets, donde todo, desde los elementos de la interfaz de usuario hasta la estructura de la aplicación, se construye utilizando estos componentes.
- **API de Open Food Facts**: Esta API externa se utiliza para la lectura de códigos de barras y la obtención de información nutricional detallada de los productos.
- **Android Studio**: Este IDE (Integrated Development Environment) es utilizado para la compilación y prueba de la aplicación en dispositivos Android. Android Studio ofrece herramientas avanzadas de depuración y emulación, facilitando el desarrollo y la optimización de la aplicación para la plataforma Android.
- **Git**: Utilizado para el control de versiones, Git permite un desarrollo colaborativo y seguro, así como la gestión eficaz del código fuente del proyecto. Es fundamental para mantener un historial de cambios y gestionar todo el repositorio de código.

- **Android**: Como plataforma objetivo, Android es fundamental en este proyecto. La aplicación se desarrollará y optimizará para funcionar de manera eficiente en dispositivos Android, garantizando una experiencia de usuario fluida.
- **OneNote**: Utilizado para realizar sketches de los diseños y organizar el desarrollo del proyecto.

1.10.2 Paquetes y dependencias externas

En el desarrollo de esta aplicación se han empleado un total de 40 paquetes de Flutter, cada uno de los cuales ha sido seleccionado por sus capacidades específicas y su contribución al funcionamiento general del proyecto. A continuación, se destacan algunos de los paquetes más relevantes, agrupados por su funcionalidad y su contribución a la interfaz de usuario:

1. Respectivos a la funcionalidad:
   - **sqflite**: Es una biblioteca que proporciona una implementación de SQLite para Flutter. Se utiliza como base de datos relacional para gestionar y almacenar localmente en el dispositivo la base de datos de ingredientes, asi como el almacenamiento de los productos en las listas.
   - **hive**: *Es* una base de datos NoSQL rápida y ligera para Flutter. Se emplea para almacenar información en caché, como datos de inicio de sesión, recetas que gustan al usuario o que ha guardado en listas de recetas, evitando consultas constantes al servidor.
   - **shared_preferences**: Esta biblioteca permite almacenar datos simples en formato clave-valor en el dispositivo. Se utiliza para guardar las preferencias y ajustes personales del usuario.
   - **uuid**: Genera identificadores únicos universales (UUID). Se emplea para generar automáticamente IDs únicos para los ingredientes o recetas creadas por el usuario.

- **fuzzy**: Proporciona un algoritmo de búsqueda difusa. Se utiliza para implementar una búsqueda más eficiente y flexible de ingredientes en la base de datos local.
- **provider**: Es una herramienta de gestión de estado para Flutter. Se utiliza para gestionar la lógica de agregar y mover elementos entre la despensa y la lista de la compra.
- **ai_barcode_scanner**: Permite la lectura de códigos de barras con opciones de personalización. Es la herramienta principal para leer códigos de barras y personalizar la pantalla de lectura.
- **http**: Proporciona funcionalidades para realizar peticiones HTTP. Se utiliza para realizar peticiones a la API de Open Food Facts.
- **url_launcher**: Permite lanzar URLs en el navegador o abrir otras aplicaciones. Se emplea para abrir la página de Open Food Facts correspondiente al producto escaneado o para lanzar aplicaciones de videos incluidos en las recetas.
- **permission_handler**: Gestiona las solicitudes de permisos en tiempo de ejecución. Se utiliza para gestionar las peticiones de permisos para el servicio de notificaciones.
- **flutter_local_notifications**: Proporciona una forma de mostrar notificaciones locales en Flutter. Se utiliza para enviar notificaciones a los usuarios.
- **firebase_core**, **firebase_auth**, **cloud_firestore**, **firebase_storage**: Conjunto de paquetes para la integración con Firebase. Gestionan toda la lógica relacionada con Firebase, incluyendo inicios de sesión, registro, acceso a bases de datos y almacenamiento de archivos multimedia.
- **crypt**: Proporciona métodos de criptografía para Flutter. Se utiliza para encriptar contraseñas, asegurando que solo se suban los hashes al servidor.

- **android_alarm_manager_plus**: Permite programar tareas en segundo plano en Android. Se utiliza para programar notificaciones a horas concretas en dispositivos Android.
- **share_plus**: Facilita la compartición de contenido con otras aplicaciones. Se utiliza para compartir la lista de la compra con otras aplicaciones.
- **logger**: Proporciona una herramienta de logging para Flutter. Mejora la depuración con mensajes más visibles y personalizables.

2. Respectivos a la interfaz:

- **modal_bottom_sheet**: Proporciona una implementación de modales. Se utiliza para mostrar información en la parte inferior de la pantalla.
- **flutter_expandable_fab**: Permite crear botones de acción flotantes expandibles. Se utiliza para ofrecer múltiples herramientas en un solo botón de acción flotante.
- **day_night_time_picker**: Selector de tiempo que cambia entre modos día y noche. Se utiliza para configurar las notificaciones.
- **google_fonts**: Facilita el uso de fuentes de *Google Fonts*. Implementa fuentes personalizadas en la aplicación.
- **password_strength_checker**: Proporciona información sobre la fortaleza de las contraseñas. Ofrece feedback al usuario sobre la seguridad de su contraseña durante el registro.
- **fluttertoast**: Muestra mensajes tipo toast. Se utiliza para mostrar avisos rápidos al usuario.
- **convex_bottom_bar**: Proporciona una barra inferior con diseño convexo. Se utiliza para el menú inferior de las pantallas principales.
- **font_awesome_flutter**: Permite el uso de iconos de *Font Awesome*. Implementa iconos adicionales en la interfaz.
- **flutter_svg**: Proporciona soporte para visualizar gráficos SVG. Se utiliza para mostrar imágenes SVG.

- **loading_indicator**: Proporciona indicadores de carga personalizados. Se utiliza para visualizar indicadores de carga más atractivos.
- **flutter_overboard**: Proporciona una pantalla inicial de bienvenida. Se utiliza para ofrecer una pantalla de introducción para nuevos usuarios.

1.11 Metodología de desarrollo de software

Una metodología de desarrollo de software es un conjunto de prácticas, técnicas y procedimientos que se utilizan para organizar, planificar y ejecutar proyectos de desarrollo de software. Su objetivo principal es mejorar la eficiencia y la calidad del proceso de desarrollo, asegurando que el software se entregue a tiempo, dentro del presupuesto y cumpla con los requisitos del cliente. Estas metodologías proporcionan un marco estructurado que guía a los equipos de desarrollo a través de las diferentes fases del ciclo de vida del software, desde la concepción y el diseño hasta la implementación, prueba y mantenimiento.

1.11.1 Scrum como elección de metodología

Dada la estructura del proyecto y sus dimensiones, se ha optado por utilizar la **metodología Scrum**. Scrum es una metodología ágil que se basa en la realización de incrementos pequeños y manejables, permitiendo así una mayor flexibilidad y adaptación a los cambios a lo largo del proceso de desarrollo.

Algunos de los componentes clave de Scrum son:

- **Roles**: Scrum define tres roles principales:
  - **Product Owner**: Responsable de maximizar el valor del producto y gestionar el backlog del producto.
  - **Scrum Master**: Facilita el proceso Scrum, ayuda al equipo a seguir las prácticas de Scrum y elimina impedimentos.
  - **Equipo de Desarrollo**: Grupo multifuncional que trabaja en la entrega de incrementos del producto.
- **Eventos**:

- **Sprint**: Periodo de tiempo fijo (generalmente de 1 a 4 semanas) durante el cual se realiza un incremento del producto.
- **Sprint Planning**: Reunión para planificar el trabajo que se realizará durante el sprint.
- **Daily Scrum**: Reunión diaria de 15 minutos para sincronizar el trabajo del equipo.
- **Sprint Review**: Reunión para revisar el incremento y adaptar el backlog del producto si es necesario.
- **Sprint Retrospective**: Reunión para reflexionar sobre el sprint y buscar mejoras continuas.

- **Artefactos**:
  - **Product Backlog**: Lista priorizada de todo el trabajo que se necesita en el producto.
  - **Sprint Backlog**: Lista de tareas seleccionadas del product backlog para completarse en el sprint actual.
  - **Increment**: El resultado de un sprint, que debe ser un producto utilizable y potencialmente desplegable.

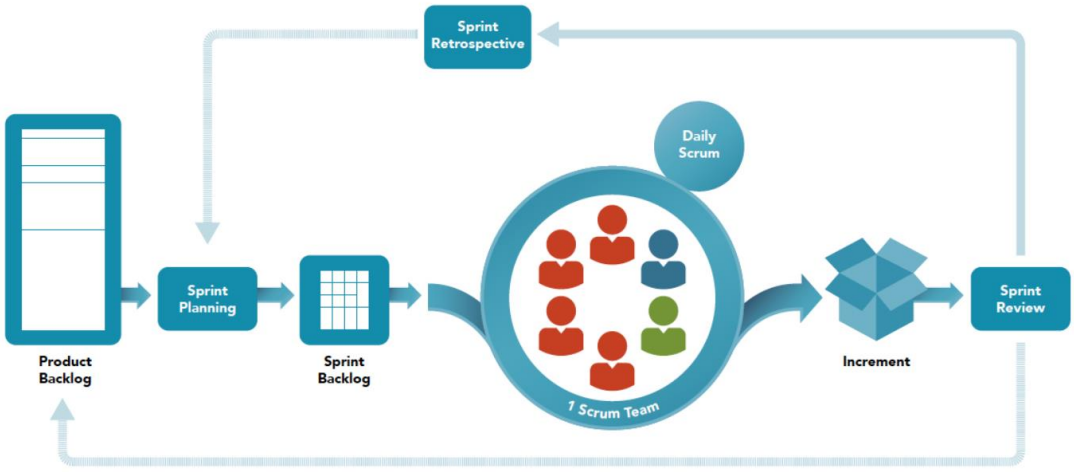

Ilustración 1.11: Esquema de desarrollo de Scrum

Scrum se caracteriza por su enfoque en la **transparencia, inspección y adaptación**, permitiendo a los equipos responder rápidamente a los cambios y mejorar continuamente.

1.11.2 Adaptación de Scrum para desarrolladores independientes

Aunque Scrum está diseñado originalmente para equipos, sus principios y estructura pueden adaptarse para el desarrollo por una sola persona. Basándonos en el artículo "Scrum for One" de *Lucidchart* [5] vamos a desarrollar el proceso de adaptación de esta metodología para una sola persona.

En un entorno de un solo desarrollador, la misma persona asume los roles de *Product Owner*, *Scrum Master* y Equipo de Desarrollo. Esto requiere una clara organización y autogestión. El desarrollador único debe gestionar el backlog del producto, facilitar su propio proceso de desarrollo y eliminar impedimentos de manera autónoma. Este enfoque puede ser beneficioso ya que el desarrollador tiene control total sobre las decisiones y priorizaciones, lo que puede agilizar el proceso.

Se pueden mantener los eventos clave de Scrum, pero de manera simplificada y flexible para una sola persona:

- **Sprint Planning**: Planificar los sprints sigue siendo esencial. El desarrollador único debe dedicar tiempo a establecer objetivos claros para cada sprint, definir las tareas necesarias y priorizarlas en el sprint backlog.
- **Daily Scrum**: Aunque no hay un equipo con el cual sincronizarse, el desarrollador puede realizar una auto-revisión diaria. Esto ayuda a mantener el enfoque y la disciplina, permitiendo reflexionar sobre el progreso y ajustar el plan si es necesario.
- **Sprint Review**: Al final de cada sprint, el desarrollador revisa el trabajo completado. Este evento puede incluir la evaluación de cómo se han alcanzado los objetivos del sprint y la identificación de cualquier ajuste necesario en el backlog del producto.

- **Sprint Retrospective**: Es crucial para la mejora continua. El desarrollador reflexiona sobre lo que funcionó bien y lo que se puede mejorar para los próximos sprints.

Los artefactos de Scrum también siguen siendo importantes ya que una vez vistos las modificaciones en los eventos se puede concluir que los artefactos se usarán de la misma forma que en un equipo con varias personas.

Además, la naturaleza iterativa de Scrum permite una gran flexibilidad y capacidad de adaptación. Esto significa que el desarrollador puede responder rápidamente a los cambios y **ajustar su enfoque según sea necesario**, lo cual es vital en un entorno de desarrollo dinámico. Las retrospectivas regulares fomentan una cultura de mejora continua, permitiendo identificar áreas de mejora y aplicar cambios en ciclos cortos, lo que facilita la evolución y optimización del proceso de desarrollo.

1.11.3 Sprints de desarrollo

Los sprints de desarrollo se han dividido en base a funcionalidades específicas de la aplicación, permitiendo un enfoque incremental y manejable para la implementación de cada componente clave. A continuación, se describen los sprints planificados y ejecutados durante el desarrollo del proyecto.

Sprint 1 - Diseño, Planificación y Configuración Inicial

En las primeras semanas, el enfoque está en el diseño y la planificación de la interfaz y los sistemas de la aplicación. Se realizan bocetos y se definen las funcionalidades principales, asegurando una base sólida para el desarrollo posterior. Además, se crea el proyecto inicial en Flutter, repositorio, y se configura el entorno de desarrollo.

Sprint 2 - Implementación de la Pantalla Principal y Lista de Ingredientes

Se centra en la creación de la pantalla principal, que incluye las listas de despensa y de la compra, y la integración de la lista de ingredientes con la pantalla principal. Se implementan todas las funcionalidades básicas asociadas, como la

adición, edición y eliminación de elementos en estas listas, y la posibilidad de añadir productos a partir de una base de datos de ingredientes predefinidos.

Sprint 3 - Lectura de Códigos de Barras y mejora de interfaz

Este sprint se dedica a la actualización y mejora de los elementos visuales de la aplicación, así como a la implementación de la funcionalidad de lectura de códigos de barras y su integración con la API de Open Food Facts, permitiendo a los usuarios escanear productos y obtener información nutricional.

Sprint 4 - Gestión de Notificaciones

En este sprint, se desarrollan las funcionalidades de notificaciones, permitiendo a los usuarios recibir alertas programables y personalizables sobre la caducidad de los productos en su despensa.

Sprint 5 - Integración con Firebase y Gestión de Cuentas de Usuario

Este sprint se centra en la integración del proyecto con Firebase para la gestión de usuarios, almacenamiento de datos y archivos multimedia, implementando las pantallas de login y registro.

Sprint 6 - Visualización y Creación de Recetas

Durante este sprint, se desarrolla el modelo de las recetas en la aplicación junto con la pantalla de visualización de recetas. También se dedica a la elaboración de la pantalla para la creación de recetas, permitiendo a los usuarios subir sus propias recetas a Firebase.

Sprint 7 - Implementación de la Búsqueda y Filtrado de Recetas

Se desarrolla una funcionalidad de búsqueda básica de recetas, permitiendo a los usuarios buscar recetas por nombre o ingredientes. Este sprint también se centra en la implementación de la clasificación de recetas en categorías y en la optimización de la búsqueda, añadiendo filtros y mejorando tanto la búsqueda por ingredientes como la búsqueda por nombre.

Sprint 8 - Gestión de Listas de Recetas y Sistema de Valoración

En este sprint, se implementa la funcionalidad para que los usuarios puedan crear y gestionar listas de recetas, sincronizándolas online en Firebase. Además, se desarrolla un sistema de valoración de recetas basado en "me gustas" de los usuarios.

Sprint 9 - Optimización Final y solución de errores

El sprint final se dedica a la corrección de errores, optimización general del rendimiento de la aplicación y gestión de distintos errores que se hayan podido dejar atrás.

1.12 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFT, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas). Por consiguiente, los cálculos de tamaño del proyecto están supeditados al tiempo disponible. En cuanto al esfuerzo, se dispone de tan un solo efectivo (la persona autora del trabajo).

Para realizar una estimación del tamaño y esfuerzo del proyecto, se utilizarán historias de usuario y puntos de historia. Las historias de usuario son descripciones breves y sencillas de una funcionalidad o característica desde la perspectiva del usuario final. Estas historias permiten entender qué es lo que se debe desarrollar y cuál es el valor que aporta al usuario. Por otro lado, los puntos de historia son una unidad de medida que ayuda a estimar el esfuerzo relativo requerido para implementar cada historia de usuario. Esta estimación no se basa en tiempo absoluto, sino en la complejidad, el riesgo y la cantidad de trabajo necesario.

Historia	Puntos de historia
Diseño de la interfaz, planificación y configuración inicial	6
Gestión de despensa y cesta	6
Desarrollar lista de ingredientes	4
Funcionalidad para escanear productos mediante el código de barras y consulta a la API	5
Pantalla de usuario	3
Menús y mejora de la interfaz	2

Sistema de control de la caducidad en alimentos	3
Notificaciones programables de caducidad	8
Implementación de Firebase al proyecto	5
Creación de cuentas en Firebase	5
Gestión de cuentas de usuarios	6
Implementación y visualización de recetas	4
Creación de recetas	5
Subida de recetas	3
Gestión de distintos roles de usuario y moderación	2
Búsqueda por ingredientes	6
Búsqueda por nombre	6
Búsqueda por categoría	4
Aplicar filtros de búsqueda	6
Sistema de valoración basado en "me gustas" de los usuarios	5
Creación y gestión de listas de recetas personales de los usuarios	4
Optimización	2
Arreglo de errores	3
Total	**103**

Tabla 1.3: Historias de usuario y puntos de historia

Es importante destacar que esta estimación inicial puede ser modificada durante el desarrollo de la aplicación. A medida que se avanza en los sprints y se obtiene una mejor comprensión de los requerimientos y desafíos técnicos, es común ajustar los puntos de historia para reflejar con mayor precisión el esfuerzo real involucrado. Esto asegura que la planificación sea flexible y se pueda adaptar a cambios y nuevas informaciones.

1.13 Planificación temporal

A continuación se detalla el cronograma del proyecto, incluyendo los resultados parciales, hitos intermedios y la duración prevista del trabajo desde la fecha de inicio. Se ha utilizado un diagrama de Gantt para representar visualmente esta planificación.

La realización de las tareas del proyecto se combina con la actividad académica pertinente, lo que puede resultar en periodos de inactividad, así como en la ampliación o compresión de otras actividades en función del tiempo disponible.

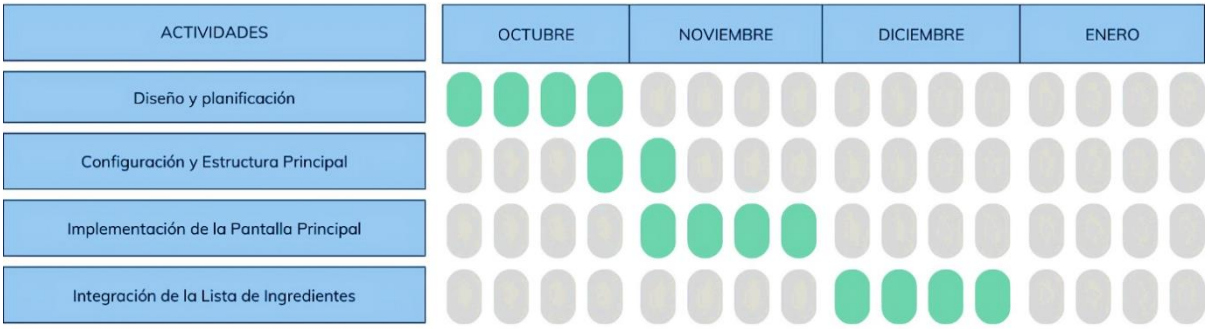

Ilustración 1.12: Diagrama de Gantt de los meses de octubre a enero

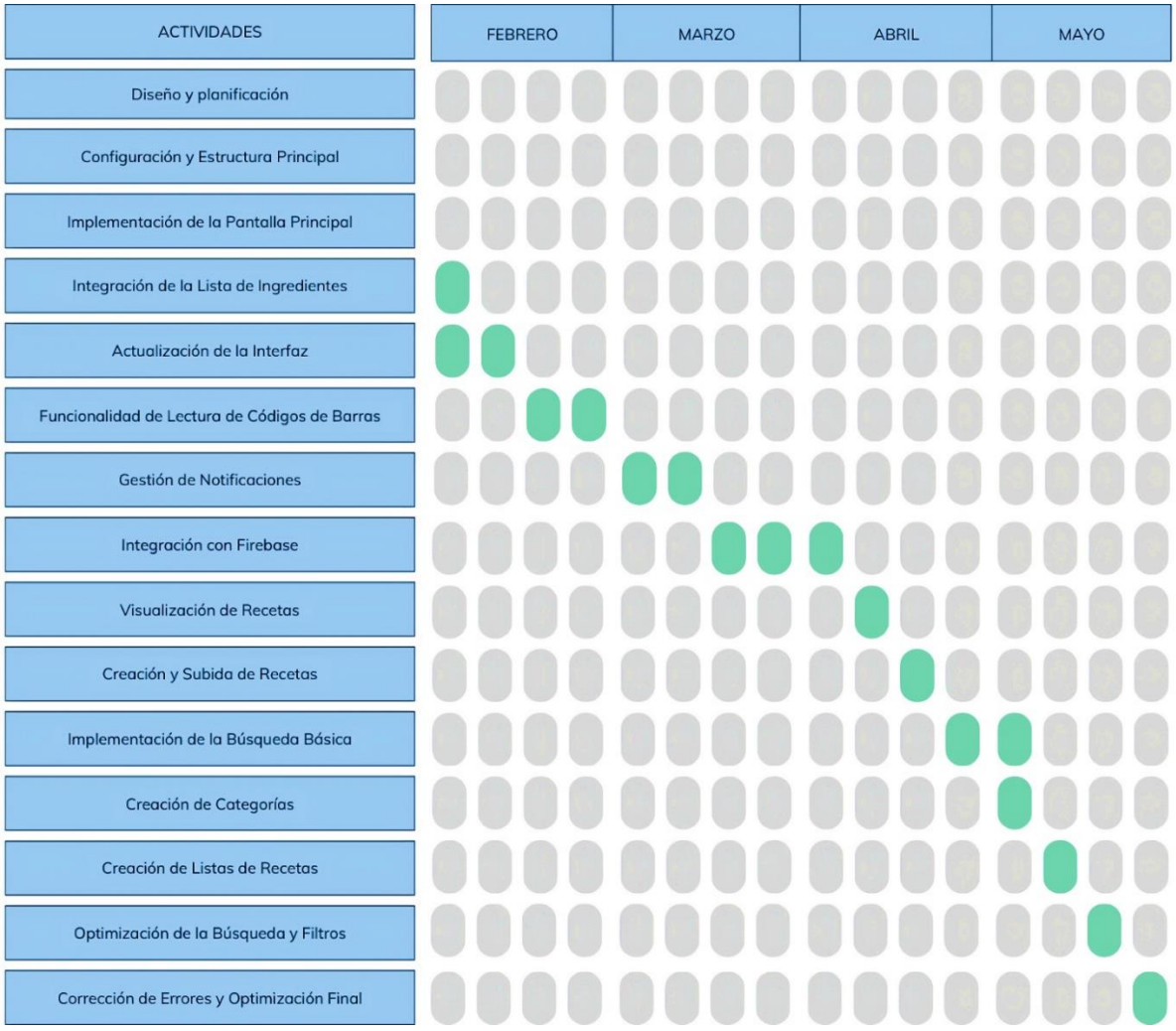

Ilustración 1.13: Diagrama de Gantt de los meses de febrero a mayo

1.14 Presupuesto

Para elaborar el presupuesto desglosaremos los diferentes elementos que han intervenido en el desarrolle y calcularemos sus costes asociados aproximados para los 8 meses de duración que ha tenido.

1.14.1 Recursos humanos

Este proyecto ha sido realizado por una sola persona que ha asumido los roles de jefe de proyecto, diseñador y programador, pero en este caso consideraremos que se contrata a un programador junior, quién asumiría todos los costes esenciales. Según el XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública, el salario anual de un programador junior es de aproximadamente 14.800,66 euros. Desglosado para el periodo de 8 meses que ha durado el proyecto, el coste es el siguiente:

$$Salario\ base\ anual = 14.800{,}66\ euros$$

$$Salario\ mensual = \frac{14.800{,}66}{12} = 1.233{,}39\ euros/mes$$

$$Salario\ mensual = 1.233{,}39 \times 8\ meses = 9{,}867.12\ euros$$

1.14.2 Recursos materiales

El desarrollo del proyecto ha requerido el uso de hardware tanto para el desarrollo como para la realización de pruebas y test. En particular se ha usado un ordenador portátil *PFS Pavilion 16-a0003N HP* con un coste actual de 899,57 euros y un smartphone *Xiaomi Mi 10 5G* con un coste actual de 300 euros. Desglosado para su amortización en los 8 meses de desarrollo:

$$Costo\ del\ ordenador = 899{,}57\ euros$$
$$Vida\ útil\ estimada = 4\ años\ (48\ meses)$$

$$Amortización\ mensual = \frac{899{,}57}{48} = 18{,}74\ euros/mes$$

$$Amortización\ para\ 8\ meses = 18{,}74 \times 8 = 149{,}92\ euros/mes$$

$$Costo\ del\ smartphone = 300\ euros$$

$$Vida\ útil\ estimada = 3\ años\ (36\ meses)$$

$$Amortización\ mensual = \frac{300}{36} = 8{,}33\ euros/mes$$

$$Amortización\ para\ 8\ meses = 8{,}33 \times 8 = 66{,}64\ euros/mes$$

1.14.3 Recursos Tecnológicos

Para la realización de este proyecto se han utilizado diversas herramientas de software de uso gratuito como *Android Studio* para su uso como entorno de desarrollo, *GitHub Desktop* para mantener un control de versiones o softwares de edición de imágenes para algunos de los iconos o gráficos de la aplicación entre otros programas para utilidades particulares.

Respecto a las tecnologías backend se ha usado Firebase, el cual aunque sí que tiene un coste de mantenimiento asociado, aunque durante este proyecto se ha estado usando la cuota gratuita que ofrece sin llegar a excederse en ninguna de sus funcionalidades que impliquen un pago.

Por lo tanto no se considerarán gastos asociados a softwares ni licencias de uso de ningún tipo.

1.14.4 Coste total

Finalmente, sumando todos los costes anteriores, se obtiene el coste total del proyecto. Este presupuesto completo asegura que todos los elementos necesarios para la realización del trabajo han sido considerados y justificados, presentando un desglose claro y detallado de cada partida.

Concepto	Costo Unitario	Cantidad	Amortización Mensual	Amortización en 8 meses	Coste Total
Ordenador	899.57 €	1	18.74 €	149.92 €	149.92 €
Smartphone	300.00 €	1	8.33 €	66.64 €	66.64 €
Salario Programador Junior	14,800.66 €/año	1	1,233.39 €/mes	9,867.12 €	9,867.12 €
Android Studio	0.00 €				0.00 €
Firebase	0.00 €				0.00 €
Total Directo					**10,083.68 €**

Tabla 1.4: Presupuesto respecto a los costes directos

Además de los costes directos, se deben considerar los costes indirectos que surgen del uso de instalaciones, servicios y otros gastos generales asociados al proyecto. Para simplificar el cálculo, se ha estimado que los costes indirectos **representan un 10%** del total de los costes directos. Estos incluyen gastos como alquiler, electricidad, agua, gestión y otros servicios necesarios para el desarrollo del proyecto.

Concepto	Coste Total
Total Directo	10,083.68 €
Gastos Indirectos (10%)	1,008.37 €
Coste Total Proyecto	**11,092.05 €**

Tabla 1.5: Presupuesto total final

El coste total de este proyecto asciende a **11,092.05 €**, incluyendo los gastos indirectos y todos los recursos mencionados. Este presupuesto determina y justifica el coste económico de la elaboración del trabajo.

2 DISEÑO INICIAL

A continuación, se explicará la arquitectura del sistema propuesto y los modelos de diseño correspondientes a la funcionalidad, interfaces y datos con el objetivo de **definir completamente el funcionamiento de la aplicación**, todos sus requisitos y las bases de la aplicación que marcarán el desarrollo de la misma.

2.1 Requisitos del sistema

Los requisitos funcionales y no funcionales del sistema son fundamentales para definir qué debe hacer la aplicación y cómo debe comportarse. Los requisitos funcionales describen las funcionalidades específicas que debe tener la aplicación para cumplir con los objetivos del proyecto, mientras que los requisitos no funcionales **especifican los criterios de calidad que deben ser satisfechos**, tales como seguridad, rendimiento y usabilidad.

2.1.1 Requisitos funcionales

Los requisitos funcionales se centran en las **funcionalidades específicas que la aplicación debe proporcionar** para cumplir con las necesidades de los usuarios y los objetivos del proyecto. En este subapartado, se describen detalladamente las acciones que los usuarios podrán realizar con la aplicación, como la gestión de despensa, la creación y búsqueda de recetas, y la interacción social entre usuarios. Cada funcionalidad se presenta con una breve descripción y su prioridad relativa, indicando la importancia y una descripción del mismo.

Requisito	Descripción	Prioridad
Gestión de Despensa	Permitir a los usuarios añadir, borrar y editar productos en una lista de productos de despensa.	Alta
Gestión de Cesta	Permitir a los usuarios añadir, borrar y editar productos en una lista de productos de la compra.	Alta
Orden de Listas	Permitir que las listas sean ordenadas según distintas preferencias del usuario y recordar estas preferencias.	Media
Compartir Cesta	Permitir que el usuario comparta su lista de la compra fácilmente con otras aplicaciones.	Baja

Cantidades	Permitir que los usuarios especifiquen una cantidad y una unidad al añadir un producto.	Alta
Caducidad	Permitir que los usuarios especifiquen una fecha de caducidad en cada producto de la despensa.	Alta
Visualizar Caducidad	Permitir al usuario elegir si quiere visualizar o no las fechas de caducidad en los alimentos.	Media
Lista de Ingredientes	Crear una lista de ingredientes local completa con nombre, icono y unidad de medida para cada ingrediente.	Alta
Iconos de Ingredientes	Incluir suficientes iconos asociados a los ingredientes para que la mayoría puedan ser fácilmente identificados.	Alta
Sugerencias de Ingredientes	Ofrecer sugerencias basadas en la lista de ingredientes cuando un usuario introduce manualmente un producto.	Media
Mover Productos	Permitir que el usuario mueva productos fácilmente entre la lista de despensa y la lista de la compra.	Alta
Escaneo de Productos	Permitir el escaneo de productos mediante su código de barras y obtener información mediante una API externa.	Alta
Escaneo como Adición	Usar el escaneo de productos como una forma alternativa de adición rápida de productos.	Alta
Información de Escaneo	Visualizar la información principal del producto en la pantalla de adición antes de ser guardado.	Alta
Sugerencias de Escaneo	Ofrecer sugerencias para emparejar productos escaneados con ingredientes de la base de datos.	Media
Escaneo en Adición Manual	Permitir el uso del escáner de códigos de barras desde la adición manual de productos.	Baja
Notificaciones	Notificar al usuario sobre productos que van a caducar, si así lo desea.	Alta
Personalización de Notificaciones	Permitir la personalización de notificaciones: activar/desactivar, hora de notificación, días de antelación.	Media
Manejo de la Cuenta de Usuario	Permitir al usuario cerrar sesión, editar o borrar su cuenta en cualquier momento.	Alta
Configuración de Cuenta de Usuario	Proveer una pantalla de configuración de cuenta donde se pueda editar nombre de usuario y foto de perfil.	Media
Pantalla de Ajustes	Incluir una pantalla de ajustes con acceso a las principales configuraciones de la aplicación.	Media

Creación de Recetas	Proveer una pantalla completa para la creación y subida de recetas, con todos los campos y comprobaciones necesarias.	Alta
Previsualización en Creación	Ofrecer una previsualización de la receta durante el proceso de creación.	Media
Borradores	Permitir guardar recetas en una lista de borradores para continuar después.	Media
Gestión de Borradores	Mostrar un menú para ver, borrar o continuar recetas guardadas como borradores.	Media
Pantalla de Búsqueda	Incluir una pantalla de búsqueda dividida en búsqueda por ingredientes y búsqueda por nombre de receta.	Alta
Búsqueda por Ingredientes	Proveer una interfaz sencilla para buscar recetas por ingredientes, permitiendo añadir ingredientes a la búsqueda.	Alta
Usar Despensa en Búsqueda	Permitir el uso de ingredientes de la despensa en la búsqueda con un solo botón.	Alta
Pantalla de Resultados	Mostrar la información de las recetas encontradas y ofrecer filtros rápidos y avanzados en la pantalla de resultados.	Alta
Orden en Búsqueda	Permitir ordenar los resultados de búsqueda por recetas más recientes, mejor valoradas y más rápidas de elaborar.	Media
Aplicación de Filtros	Permitir aplicar filtros en función del tiempo de preparación, tipo de dieta y otros parámetros.	Media
Previsualización de Recetas	Mostrar nombre, foto, principales ingredientes, tiempo de preparación y número de ingredientes en la previsualización.	Alta
Categorías	Incluir una pantalla para buscar recetas por categorías, con clasificación automática basada en ingredientes y etiquetas.	Media
Visualización de Recetas	Proveer una pantalla completa y atractiva para la visualización de recetas, con todas las opciones e información.	Alta
Ingredientes en Recetas	Mostrar ingredientes con icono, cantidades y unidades, y marcar claramente los que el usuario tiene en despensa.	Alta
Modificación de Cantidades	Permitir modificar las cantidades de recetas en función del número de raciones.	Media
Instrucciones	Mostrar todas las instrucciones de la receta en una pestaña propia en forma de lista.	Alta
Videos de Recetas	Mostrar enlaces a videos asociados a la receta en una pestaña específica.	Media

Fotografía de Receta	Permitir visualizar y ampliar la fotografía de la receta.	Media
Me Gustas	Permitir a los usuarios dar "me gusta" a una receta y registrar este cambio en el servidor.	Alta
Lista de Me Gustas	Añadir automáticamente las recetas que han recibido "me gusta" a una lista de recetas del usuario.	Media
Listas de Recetas	Permitir guardar recetas en una o más listas de recetas, con la opción de crear nuevas listas desde la misma pantalla.	Alta
Creación de Listas de Recetas	Permitir crear listas de recetas con un nombre e icono asociado y borrarlas cuando sea necesario.	Alta
Subida de Listas	Subir las listas de recetas al servidor y asociarlas a la cuenta del usuario.	Alta
Roles	Implementar un sistema de roles para otorgar distintos permisos a los usuarios (normal, admin, superadmin).	Alta
Usuarios Administradores	Proveer un menú adicional para los usuarios con roles superiores, otorgando permisos de moderación y visualización.	Media
Moderación de Recetas	Permitir que las recetas subidas al servidor sean aprobadas por un administrador antes de hacerse públicas.	Media
Eliminación por Moderación	Permitir a los moderadores eliminar recetas inapropiadas o vacías.	Media

Tabla 2.1: Requisitos funcionales de la aplicación

2.1.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos que **definen los criterios de calidad que debe cumplir la aplicación** para ser considerada eficiente, segura y fácil de usar. Estos requisitos incluyen aspectos como la seguridad de los datos del usuario, la usabilidad de la interfaz, el rendimiento del sistema y la capacidad de escalar para soportar un gran número de usuarios. En este subapartado, se detallan estos requisitos, proporcionando una visión clara de los estándares de calidad que se espera que la aplicación cumpla.

Requisito	Descripción	Prioridad
Proyecto de Firebase	Tener un proyecto de Firebase asociado como servidor para la gestión de cuentas de usuario, recetas, búsquedas y listas de recetas.	Alta
Gestión de Estados de Usuarios	Controlar los estados de sesión de usuario y comportarse distinto en función de si es la primera vez que el usuario abre la aplicación.	Media
Pantallas de Login y Registro	Mostrar una pantalla de inicio de sesión y registro si el usuario no tiene cuenta iniciada.	Alta
Overboard	Mostrar una pantalla de bienvenida o "overboard" la primera vez que el usuario usa la aplicación.	Baja
Registro	Permitir un registro completo mediante correo electrónico, introduciendo correo, nombre de usuario, foto y contraseña sin errores.	Alta
Comprobaciones de Utilización	Comprobar que el nombre de usuario o correo no estén ya en uso durante el registro.	Alta
Nombres Ofensivos	Ofrecer un mecanismo para impedir la introducción de nombres de usuario ofensivos mediante una API externa.	Baja
Verificación por Correo	Utilizar la verificación de cuenta por correo mediante Firebase, no permitiendo iniciar sesión hasta que el usuario lo verifique.	Alta
Encriptación	Encriptar las contraseñas localmente en el dispositivo, mandando al servidor solo el hash de las contraseñas.	Alta
Feedback en Registro	Proveer pantallas de registro y login claras e intuitivas, ofreciendo información al usuario sobre el progreso durante la carga.	Baja
Permisos en Notificaciones	Gestionar correctamente los permisos de notificaciones, guiando al usuario para otorgar los permisos necesarios.	Alta
Seguridad	Asegurar la protección de datos de usuario en todas las interacciones de la aplicación.	Alta
Usabilidad	Diseñar una interfaz de usuario intuitiva y fácil de usar para mejorar la experiencia del usuario.	Alta
Rendimiento	Asegurar que la aplicación responde rápidamente a las interacciones del usuario.	Alta
Escalabilidad	Diseñar la aplicación para soportar muchos usuarios y grandes volúmenes de datos.	Media

Tabla 2.2: Requisitos no funcionales de la aplicación

2.2 Casos de Uso

Un caso de uso es una secuencia de acciones que un usuario realiza para completar una tarea específica dentro del sistema. Es fundamental definir los casos de uso en el desarrollo de sistemas ya que **proporciona una descripción detallada de cómo interactuarán los usuarios con la aplicación**. Estos ayudan a entender y documentar los requisitos funcionales y no funcionales desde la perspectiva del usuario, lo que facilita el diseño y la posterior implementación.

Los casos de uso deberían:

- Identificar y aclarar las funcionalidades que debe tener la aplicación.
- Describir de manera clara y comprensible las interacciones entre los usuarios y el sistema.
- Servir como base para la creación de diagramas de casos de uso y diagramas de secuencia, que ayudan a visualizar y planificar la arquitectura y el flujo del sistema.
- Facilitar la comunicación entre los desarrolladores, diseñadores y demás partes interesadas en el proyecto.

A continuación, se presentan los casos de uso de la aplicación, describiendo las principales interacciones que los usuarios tendrán con las diferentes funcionalidades del sistema.

2.2.1 Gestión de despensa y de lista de la compra

Nombre	Descripción	Actores	Precondiciones
Gestión de Despensa y Lista de la Compra	Permitir a los usuarios gestionar productos en su despensa y lista de la compra, incluyendo la adición manual, escaneo de productos, solicitud de información nutricional, ordenación de listas, visualización de fechas de caducidad, y movimiento de productos entre listas.	Usuario	El usuario debe estar autenticado.

Tabla 2.3: Descripción del caso de uso para la gestión de las listas de productos

Flujo principal:

1. El usuario accede a la sección de despensa o lista de la compra.

2. El usuario puede:
    - Añadir un producto manualmente.
    - Escanear un producto para añadirlo.
    - Solicitar información nutricional del producto a través de una API.
    - Ordenar la lista según distintos parámetros.
    - Visualizar o no las fechas de caducidad.
    - Mover productos entre la despensa y la lista de la compra.
    - Compartir la lista de la compra.
3. El sistema guarda los cambios en la base de datos.

Diagrama de caso de uso:

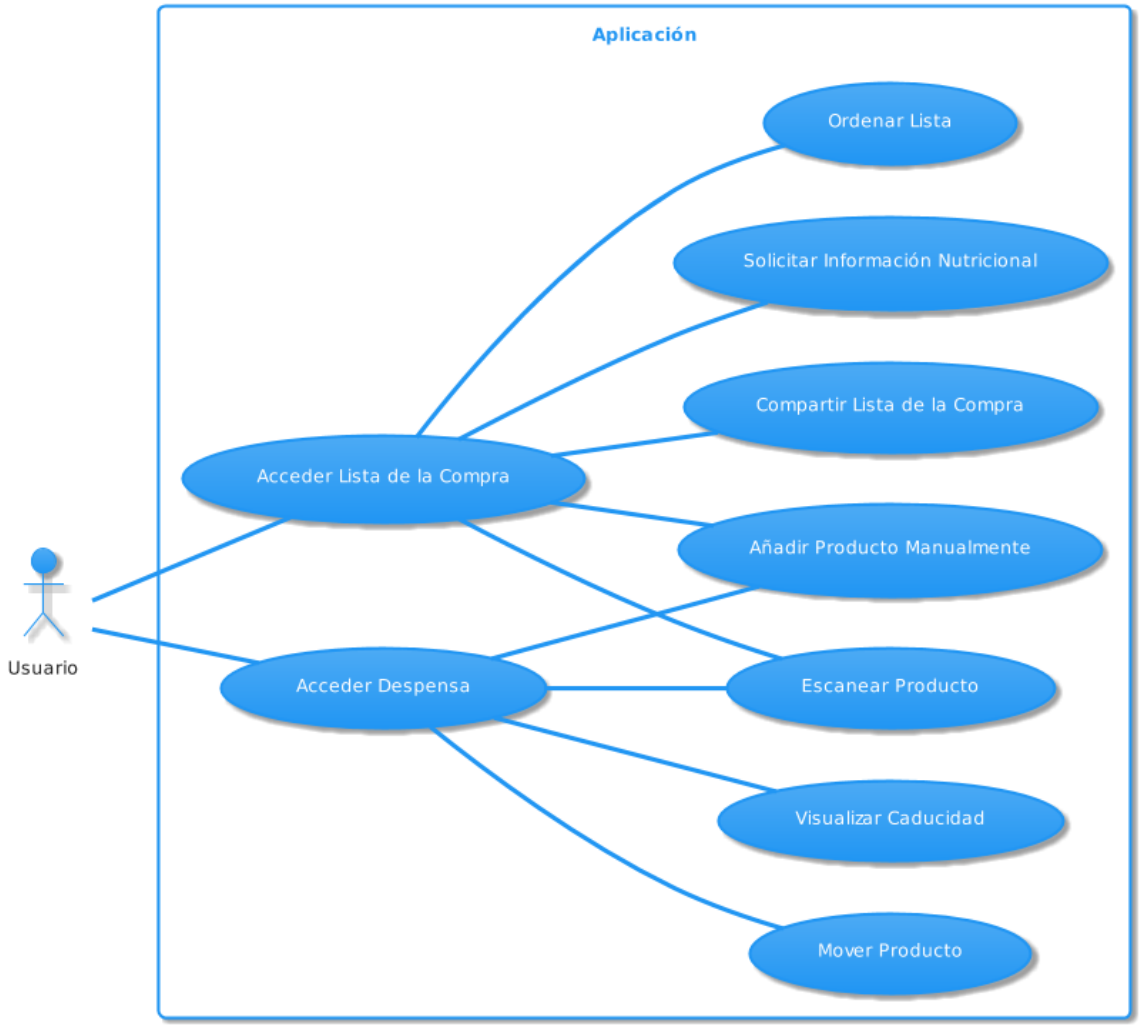

Ilustración 2.1: Diagrama de caso de uso para la gestión de las listas de productos

2.2.2 Búsqueda de recetas por ingredientes y nombre

Nombre	Descripción	Actores	Precondiciones
Búsqueda de Recetas por Ingredientes y Nombre	Permitir a los usuarios buscar recetas por ingredientes introducidos manualmente o seleccionados de la despensa, o por nombre de receta.	Usuario	El usuario debe estar autenticado.

Tabla 2.4: Descripción del caso de uso para la búsqueda de recetas

Flujo principal:

1. El usuario accede a la pantalla de creación de recetas.
2. El usuario puede:
   - Rellenar campos de la receta (nombre, ingredientes, etc.).
   - Guardar la receta en borradores.
   - Previsualizar la receta.
   - Subir la receta (si se han completado los campos obligatorios).
3. • El sistema guarda o sube la receta según corresponda.

Diagrama de caso de uso:

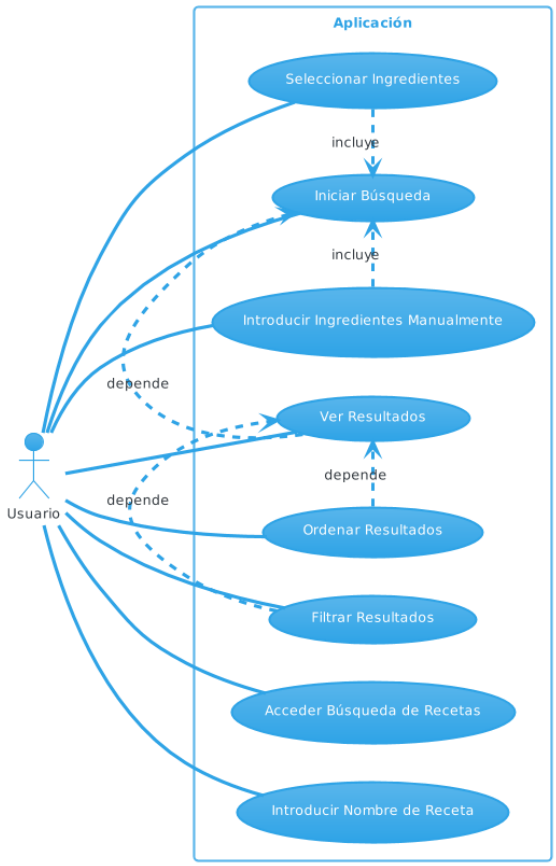

Ilustración 2.2: Diagrama de caso de uso para la búsqueda de recetas

2.2.3 Búsqueda de recetas por categoría

Nombre	Descripción	Actores	Precondiciones
Búsqueda de Recetas por Categoría	Permitir a los usuarios buscar recetas por categorías predefinidas.	Usuario	El usuario debe estar autenticado.

Tabla 2.5: Descripción del caso de uso para buscar recetas por categoría

Flujo principal:

1. El usuario accede a la pantalla de categorías.
2. El usuario selecciona una categoría.
3. El sistema muestra los resultados de recetas en esa categoría y permite:
    - Ordenar por valoración, tiempo o recientes.
    - Filtrar por recetas veganas o vegetarianas.
    - Filtrar por tiempo de preparación máximo.
    - Filtrar por recetas que contengan video.

Diagrama de caso de uso:

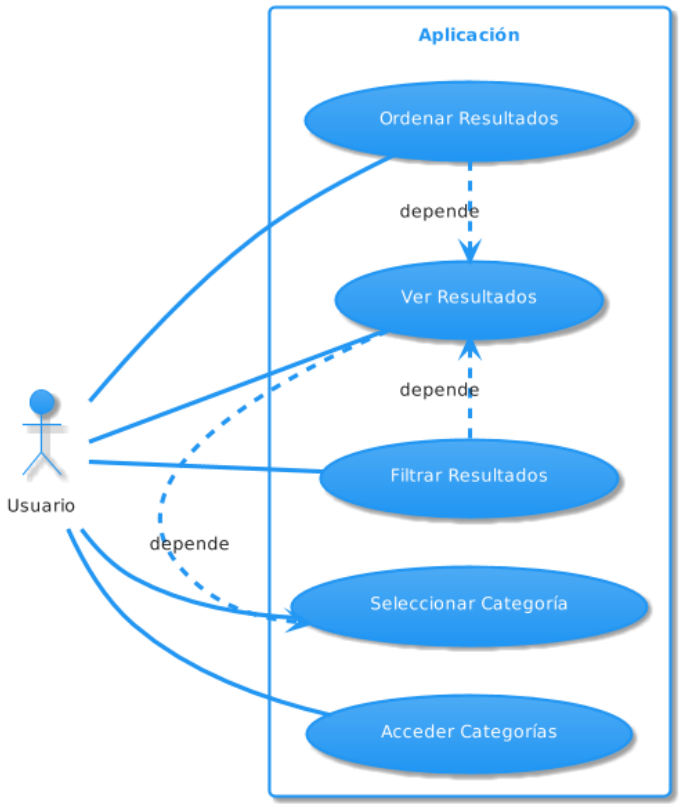

Ilustración 2.3: Diagrama de caso de uso para buscar recetas por categoría

2.2.4 Creación y subida de recetas

Nombre	Descripción	Actores	Precondiciones
Gestión de Recetas	Permitir a los usuarios crear, editar y borrar recetas, incluyendo la opción de guardar borradores y previsualizar la receta.	Usuario	El usuario debe estar autenticado y en caso de acceder a un borrador debe tener una receta guardada previamente en esta lista

Tabla 2.6: Descripción del caso de uso para crear y subir recetas

Flujo principal:

1. El usuario accede a la pantalla de creación de una receta nueva o acceder a una receta guardada en sus borradores.
2. El usuario puede:
   - Rellenar campos de la receta (nombre, ingredientes, instrucciones, etc.).
   - Guardar la receta en borradores.
   - Previsualizar la receta.
   - Subir la receta (si se han completado los campos obligatorios).
3. • El sistema guarda o sube la receta según corresponda.

Diagrama de caso de uso:

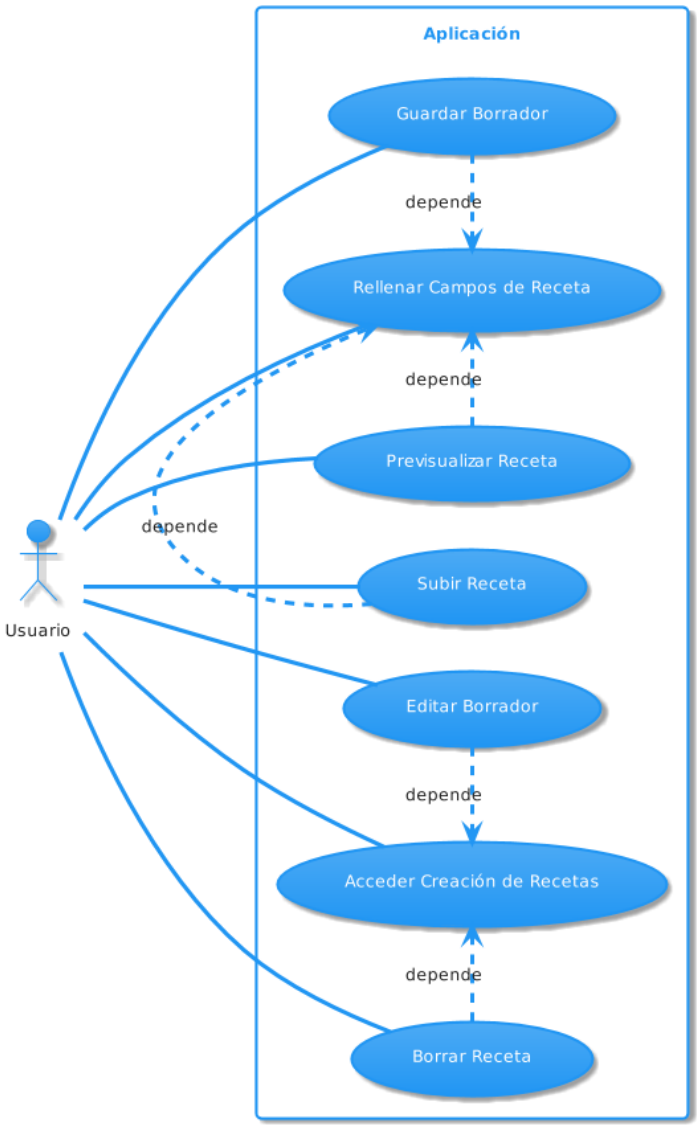

Ilustración 2.4: Diagrama de caso de uso para crear y subir recetas

2.2.5 Notificaciones de caducidad

Nombre	Descripción	Actores	Precondiciones
Notificaciones de Caducidad	Notificar a los usuarios sobre productos que están próximos a caducar y permitir la configuración de estas notificaciones.	Usuario	El usuario debe estar autenticado y tener productos con fechas de caducidad especificadas en su despensa.

Tabla 2.7: Descripción del caso de uso para la gestión de notificaciones

Flujo principal:

1. El usuario accede a la pantalla de configuración de notificaciones.

2. El usuario puede:
   - Activar o desactivar las notificaciones.
   - Configurar la hora y el día de las notificaciones.
3. El sistema verifica periódicamente las fechas de caducidad de los productos en la despensa del usuario.
4. Si algún producto está próximo a caducar y las notificaciones están activadas, el sistema envía una notificación al usuario.
5. El usuario puede ver la notificación y acceder a la despensa.

Diagrama de caso de uso:

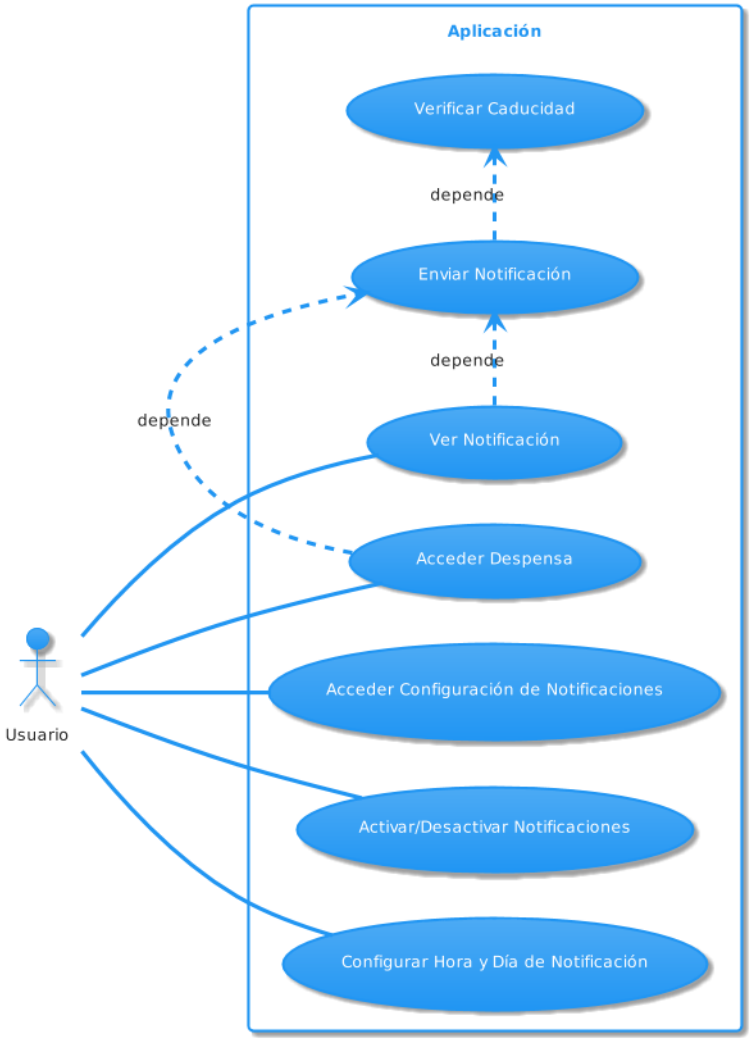

Ilustración 2.5: Diagrama de caso de uso para la gestión de notificaciones

2.2.6 Registro e Inicio de Sesión

Nombre	Descripción	Actores	Precondiciones
Registro e Inicio de Sesión	Permitir a los usuarios registrarse e iniciar sesión en la aplicación.	Usuario	El usuario debe tener acceso a una cuenta de correo electrónico verificable.

Tabla 2.8: Descripción del caso de uso para la interacción con las recetas

Flujo principal:

1. El usuario accede a la pantalla de inicio de sesión o registro.
2. Si el usuario ya tiene una cuenta, introduce su correo electrónico y contraseña.
3. El sistema verifica las credenciales y autentica al usuario.
4. Si el usuario no tiene una cuenta, selecciona la opción de registro.
5. El usuario introduce sus datos (correo electrónico, nombre de usuario, foto, contraseña).
6. El sistema verifica que el correo y el nombre de usuario no estén en uso.
7. El sistema envía un correo de verificación al usuario.
8. El usuario verifica su cuenta a través del correo recibido.
9. El sistema permite al usuario iniciar sesión tras la verificación.

Diagrama de caso de uso:

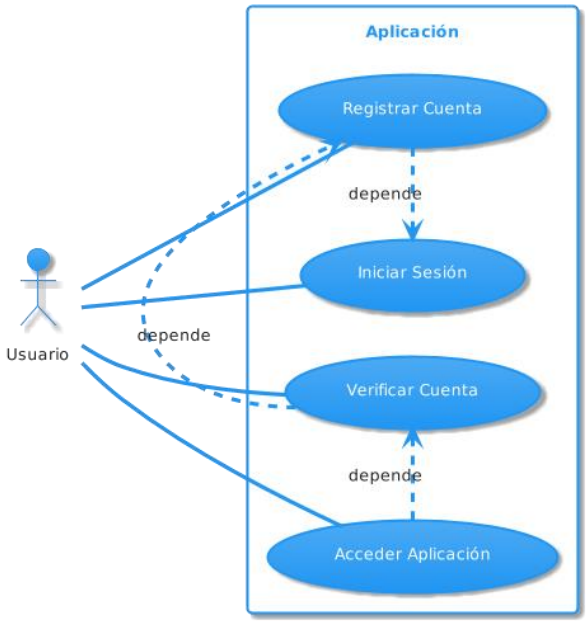

Ilustración 2.6: Diagrama de caso de uso para la interacción con las recetas

2.2.7 Interacción con recetas

Nombre	Descripción	Actores	Precondiciones
Interacción con Recetas	Permitir a los usuarios interactuar con las recetas, incluyendo dar me gusta, modificar cantidades, acceder a fotografías y videos, y guardar en listas de recetas.	Usuario	El usuario debe estar autenticado.

Tabla 2.9: Descripción del caso de uso para la interacción con las recetas

Flujo principal:

1. El usuario accede a la pantalla de visualización de recetas.
2. El usuario puede:
   - Dar me gusta a la receta.
   - Modificar las cantidades en función del número de raciones.
   - Acceder a la fotografía en pantalla completa.
   - Acceder a videos asociados.
   - Guardar la receta en una o varias listas de recetas.

- Crear una nueva lista de recetas y añadir la receta a esta lista.

3. El sistema guarda los cambios y actualiza la base de datos.

Diagrama de caso de uso:

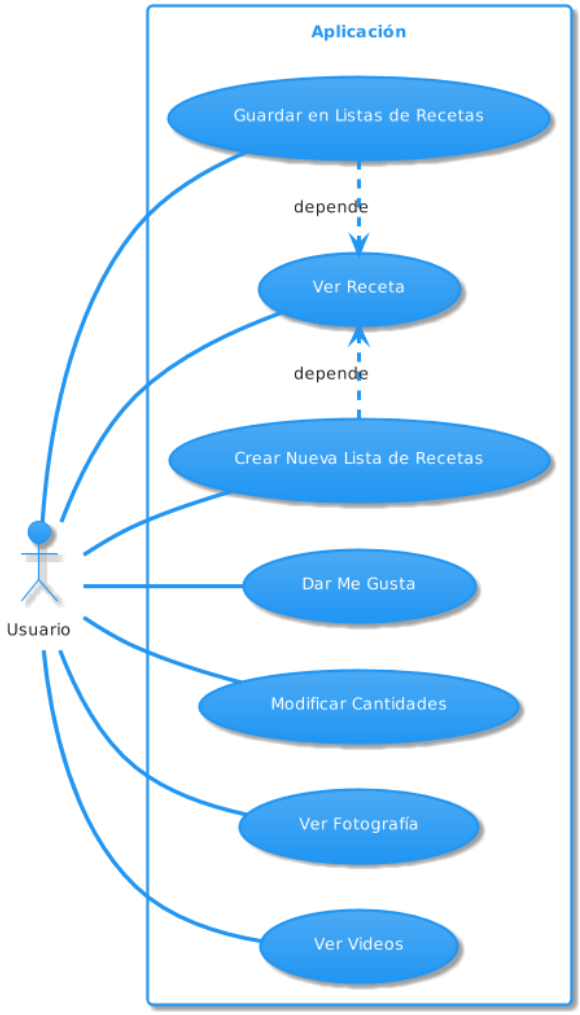

Ilustración 2.7: Diagrama de caso de uso para la interacción con las recetas

2.3 Diagramas de secuencia

Los diagramas de secuencia son una herramienta visual que se utiliza para ilustrar cómo interactúan los distintos componentes de un sistema a lo largo del tiempo. Estos diagramas muestran el flujo de mensajes entre los actores (usuarios y sistemas) y los objetos dentro del sistema, permitiendo una comprensión clara y detallada de la dinámica del sistema y sus procesos internos. La finalidad de los diagramas de secuencia es **proporcionar una representación precisa de las**

interacciones temporales en el sistema, facilitando la identificación de posibles problemas y la optimización de los procesos.

Los diagramas de secuencia se basarán en los casos de uso descritos anteriormente. Se centrarán en aquellos procesos que resulten más complejos, obviando las funcionalidades más simples para mantener la claridad y relevancia de los diagramas. Los participantes en estos incluirán, además del usuario, los siguientes componentes:

- **La aplicación**: El cliente que interactúa con el usuario.
- **Firestore o Cloud Firestore**: La base de datos de Firebase donde se almacenarán las recetas, usuarios y listas de recetas.
- **Firebase Storage**: Donde se almacenarán las fotos de recetas y de perfil del usuario.
- **Firebase Authentication**: Para la autenticación de usuarios.
- **API de Open Food Facts:** Para consultar información nutricional basada en códigos de barras.
- **API Perspective de Google**: Para evaluar la potencial ofensividad de nombres de usuario durante el registro o cambio de nombre.

2.3.1 Registro e Inicio de Sesión

Para mayor claridad y organización en el manejo de los procesos de inicio de sesión y registro, en los diagramas de secuencia pertenecientes a estos procesos, la aplicación se divide en los procesos:

- **Screen**: Este componente representa la interfaz de usuario con la que los usuarios interactúan directamente. En las pantallas de login (LoginScreen) y registro (RegisterScreen), los usuarios ingresan sus datos y realizan acciones como enviar formularios o seleccionar imágenes. El *Screen* se encarga de capturar estas interacciones, validar los datos ingresados y mostrar mensajes de error o éxito según corresponda.

- **Provider**: Este componente maneja la lógica de negocio y las interacciones con los servicios backend. En el caso de login (LoginProvider) y registro (RegisterProvider), el Provider se encarga de procesar los datos ingresados por el usuario, realizar comprobaciones (como verificar la disponibilidad de nombres de usuario y correos electrónicos), interactuar con servicios externos, y actualizar los datos según sea necesario. El Provider actúa como intermediario entre el *Screen* y los servicios backend, asegurando que las operaciones se realicen correctamente y de manera eficiente.

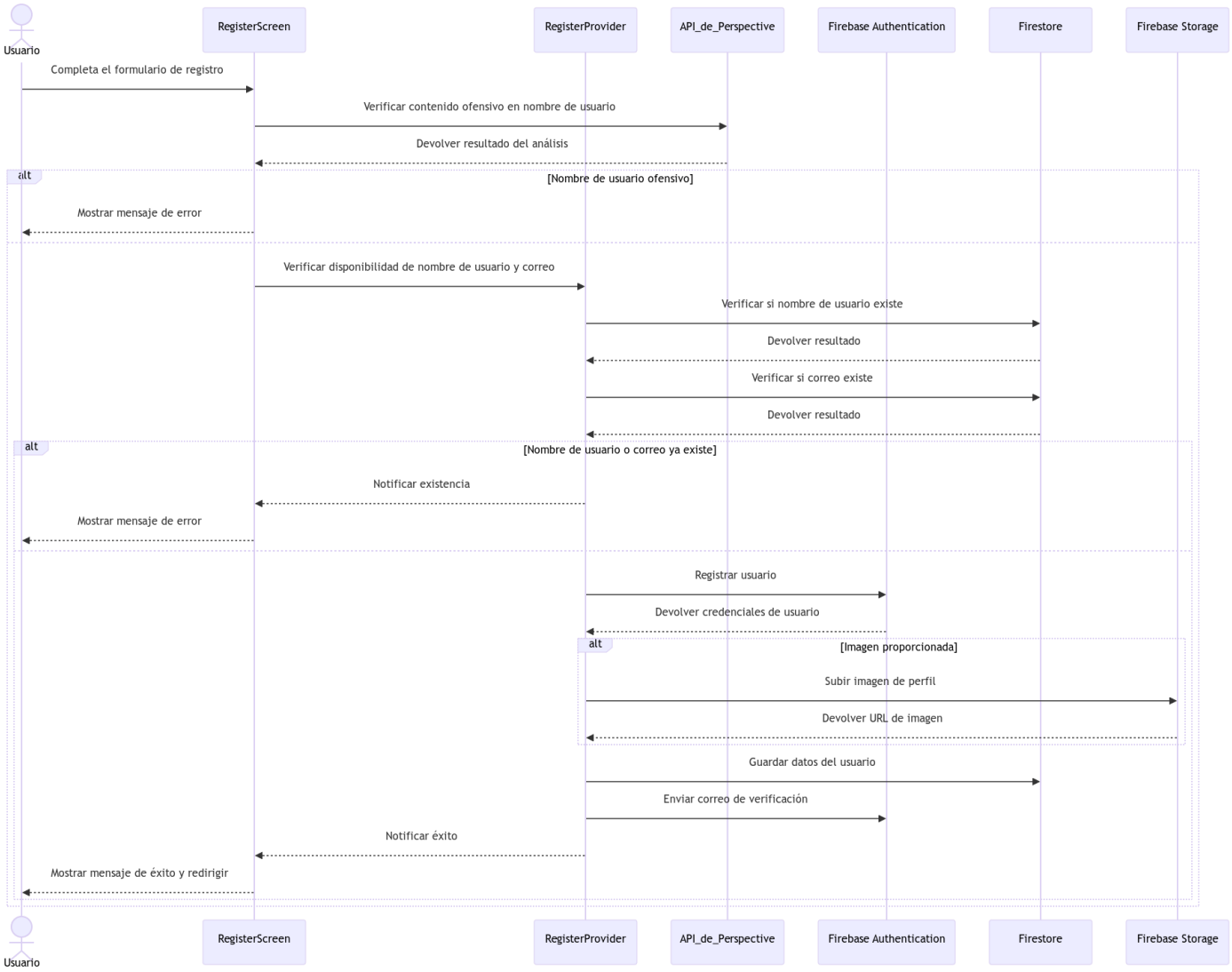

Ilustración 2.8: Diagrama de secuencia para el proceso de registro

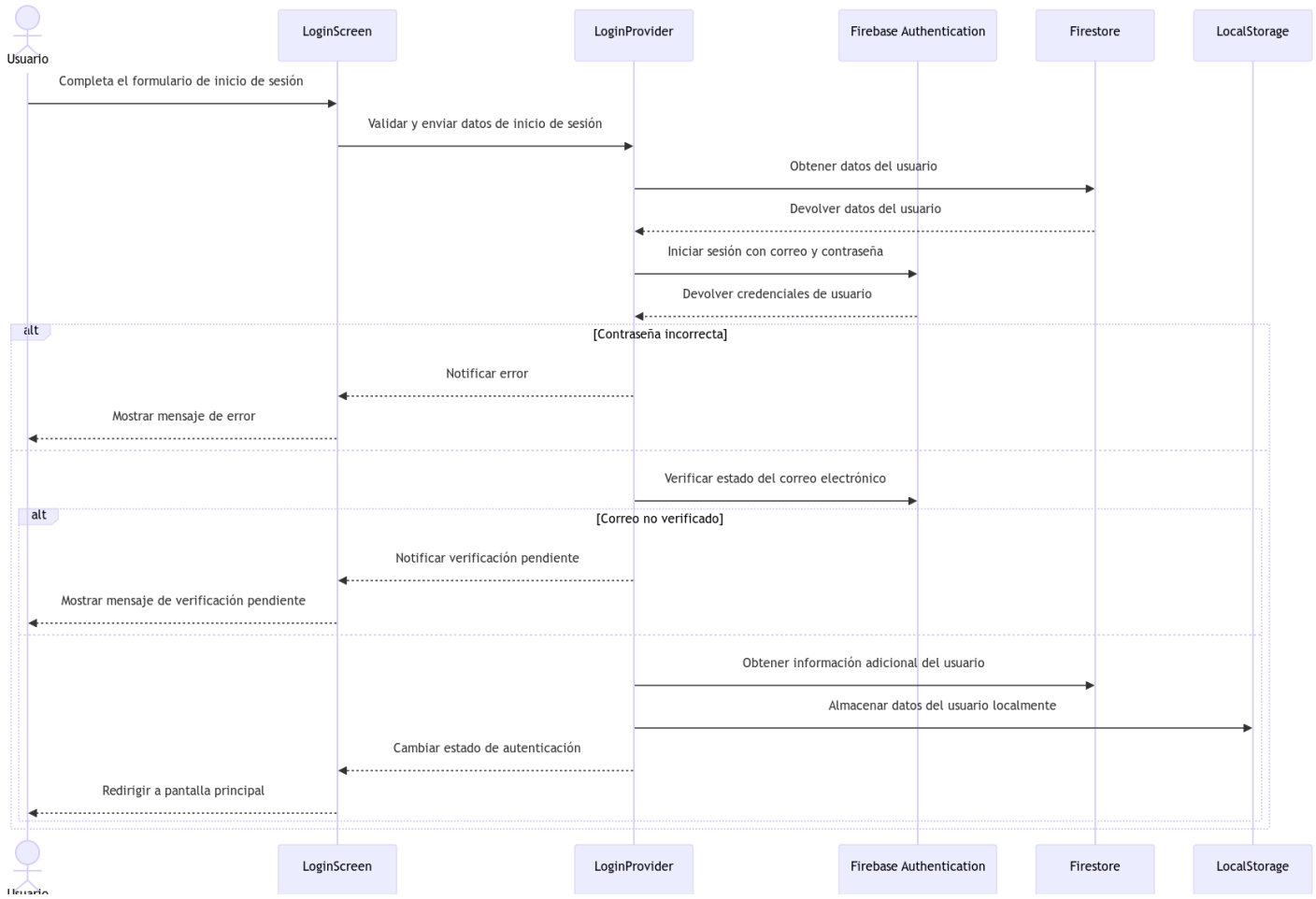

Ilustración 2.9: Diagrama de secuencia para el proceso de inicio de sesión

Para el proceso de registro el usuario accede a la pantalla de registro y completa un formulario con su nombre de usuario, correo electrónico, contraseña, repetición de la contraseña y, opcionalmente, una imagen de perfil. La aplicación valida localmente estos datos para asegurar que todos los campos obligatorios están completos y cumplen con las restricciones especificadas, como la longitud mínima de la contraseña y la estructura del correo electrónico.

Una vez que el formulario está validado, la aplicación procede a verificar si el nombre de usuario contiene contenido ofensivo utilizando la API de Perspective. Esta API analiza el texto del nombre de usuario en busca de posibles contenidos tóxicos. Si se detecta contenido ofensivo, se notifica al usuario y se le solicita que elija un nombre de usuario diferente.

Si el nombre de usuario es aceptable, la aplicación **verifica la disponibilidad** del nombre de usuario y del correo electrónico en la base de datos de Firestore. Si cualquiera de ellos ya existe, **se notifica al usuario** con un mensaje de error, y se le pide que proporcione credenciales diferentes.

Cuando se confirma que tanto el nombre de usuario como el correo electrónico están disponibles, la aplicación **registra al usuario en *Firebase Authentication***, creando una nueva cuenta con el correo electrónico y la contraseña proporcionados. Si el usuario ha seleccionado una imagen de perfil, esta **se sube a *Firebase Storage*** y la URL de la imagen **se almacena en *Firestore*** junto con otros datos del usuario, como el nombre de usuario, la contraseña encriptada, el rol del usuario, el token y la fecha de creación de la cuenta.

Finalmente, Firebase Authentication **envía un correo electrónico de verificación** al usuario. La aplicación notifica al usuario que revise su correo electrónico para completar la verificación de la cuenta. Si el registro es exitoso, el usuario es redirigido a la pantalla de inicio de sesión, donde podrá iniciar sesión una vez que haya verificado su correo electrónico.

Para el inicio de sesión el proceso es similar, el usuario accede a la pantalla de inicio de sesión e introduce su correo electrónico y contraseña en el formulario correspondiente. La aplicación valida estos datos para asegurar que ambos campos no estén vacíos y cumplan con las restricciones básicas.

Después de la validación, la aplicación **obtiene los datos del usuario desde *Firestore*** utilizando el correo electrónico proporcionado. Esta información incluye la contraseña encriptada, que se compara con la contraseña ingresada por el usuario para verificar su autenticidad.

Si las credenciales son correctas, *Firebase Authentication* inicia sesión al usuario. La aplicación entonces **verifica si el usuario ha completado la verificación** de su correo electrónico. Si el correo electrónico no está verificado, se muestra un

mensaje al usuario solicitándole que complete la verificación antes de poder iniciar sesión.

Una vez que se verifica que el correo electrónico está confirmado, la aplicación procede a **obtener información adicional del usuario** desde *Firestore*, como el nombre de usuario, el rol del usuario y la URL de la imagen de perfil. Esta información se almacena localmente para facilitar el acceso y mejorar el rendimiento de la aplicación.

Además, la aplicación descarga las recetas favoritas del usuario y las recetas creadas por el usuario desde Firestore. También se obtienen y almacenan localmente las listas de recetas del usuario, asegurando que toda la información relevante esté disponible sin necesidad de múltiples consultas a la base de datos.

Finalmente, el estado de autenticación del usuario se actualiza, y la aplicación redirige al usuario a la pantalla principal. Si en cualquier punto del proceso de inicio de sesión ocurre un error, como credenciales incorrectas o problemas de red, **se muestra un mensaje de error** adecuado al usuario, permitiéndole corregir la información ingresada y volver a intentarlo.

2.3.2 Gestión de listas de productos y escáner de productos

En el proceso de gestión de listas de productos incluiremos las principales acciones que se pueden realizar en esta, dónde además del usuario, encontraremos los siguientes actores:

1. **Interfaz de Usuario (Pantalla Despensa/Cesta):** Actúa como el punto de interacción entre el usuario y la lógica subyacente de la aplicación.
2. **Provider:** Se encargan de gestionar las listas de productos en la despensa y la cesta. Estos Providers reciben las solicitudes de la interfaz de usuario, realizan las operaciones necesarias (añadir, eliminar, editar, mover productos) y actualizan la base de datos local en consecuencia.

3. **Base de Datos Local (IngredientDao):** Es el componente que almacena persistentemente los datos de los productos. Cada vez que se realiza una operación de gestión de productos, los Providers interactúan con IngredientDao para guardar, actualizar o eliminar los datos de la base de datos local.

4. **API de Open Food Facts**: Cuando el usuario escanea un producto, la interfaz de usuario envía el código de barras a esta API, que devuelve datos como el nombre del producto, su valor nutricional y otras características relevantes. Esta información se muestra al usuario, quien puede decidir si desea guardar el producto en su lista.

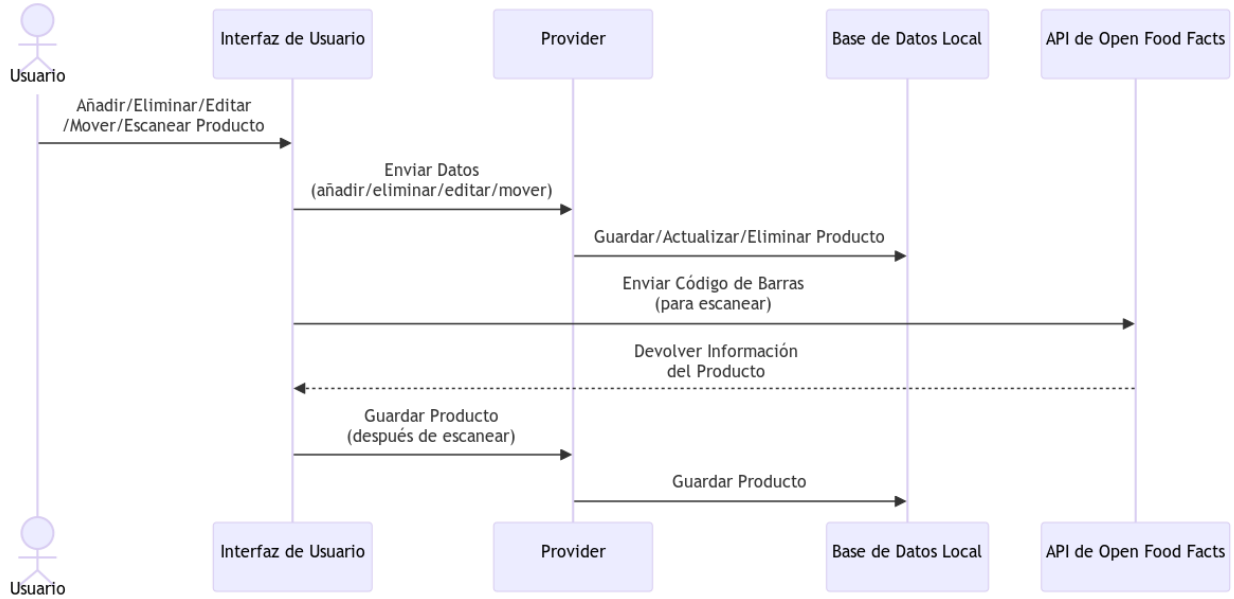

Ilustración 2.10: Diagrama de secuencia para la interacción con las listas de recetas

El proceso de gestión de listas de productos **permite al usuario añadir, eliminar, editar y mover productos** entre la Despensa y la Cesta de manera unificada. Al realizar cualquier de estas acciones, la interfaz de usuario envía la solicitud al Provider correspondiente, que a su vez actualiza la base de datos local a través de IngredientDao.

Para escanear productos, el usuario accede a la pantalla de escaneo y captura el código de barras. Este código se envía a la API de *Open Food Facts*, que devuelve

la información del producto. **La información se muestra al usuario**, quien puede optar por guardar el producto en la lista. Si decide hacerlo, la interfaz de usuario envía los datos al Provider, que guarda el producto en la base de datos local.

2.3.3 Subir una receta

Los actores para realizar este diagrama de secuencia serán similares a los ya usados en los diagramas de inicio de sesión y registro, puesto que también se interactuará con *Firestore* y *Firebase Storage*. También se hará uso del almacenamiento local de la aplicación o *LocalStorage* para guardar la receta en borradores si el usuario asi lo quiere.

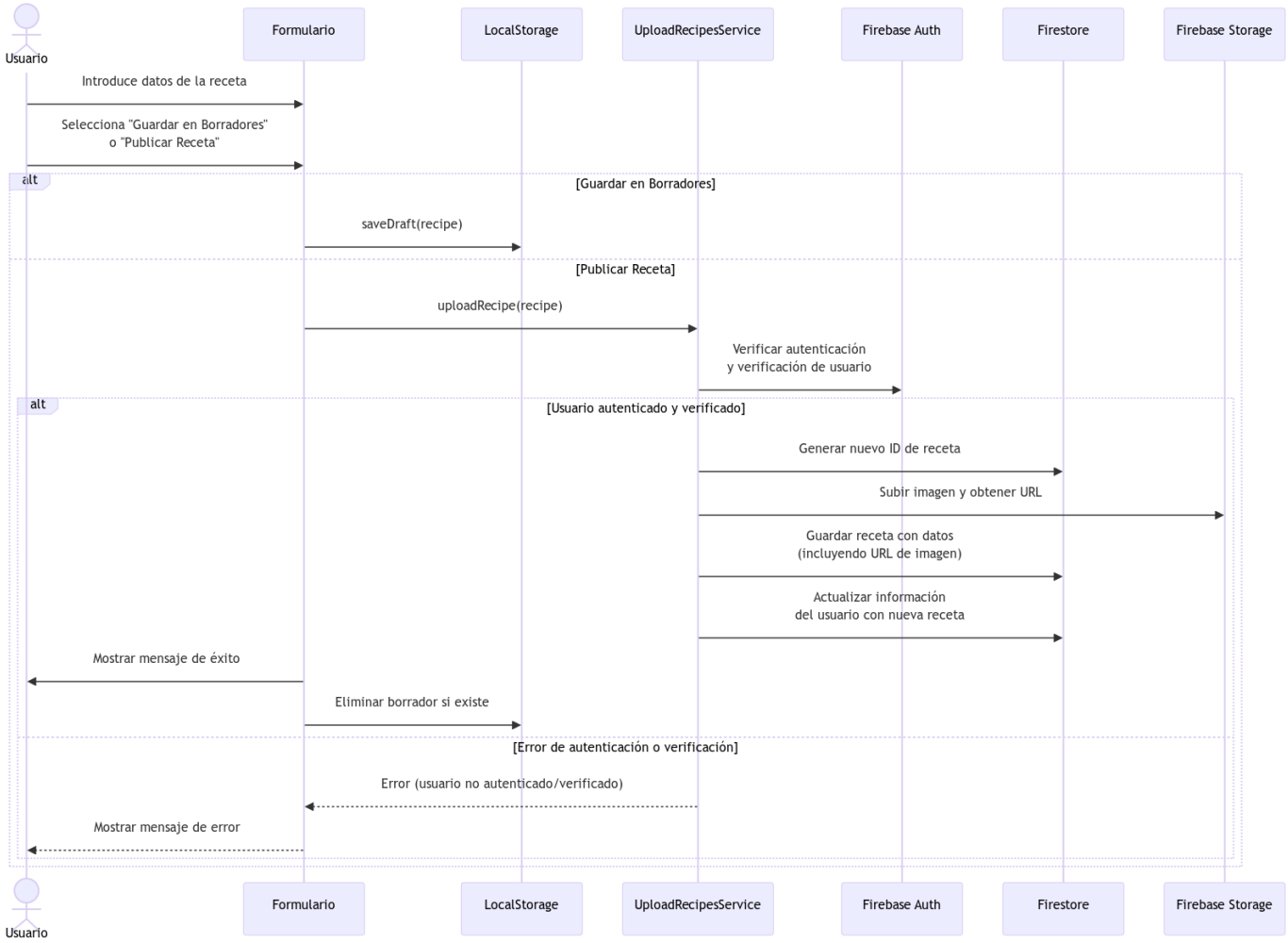

Ilustración 2.11: Diagrama de secuencia para la subida de recetas

El proceso de subida de recetas comienza cuando el usuario accede a la pantalla del formulario de recetas. En esta pantalla, el usuario puede introducir los detalles de la receta, como el título, la descripción, los ingredientes, las instrucciones, la región, las etiquetas, los enlaces de video y la foto de la receta.

El usuario **tiene la opción de guardar la receta en borradores** o **publicarla** directamente. Si elige guardar en borradores, la receta se almacena localmente en el dispositivo del usuario utilizando LocalStorage. Esta acción permite al usuario **continuar editando la receta más tarde**.

Si el usuario decide publicar la receta, se realiza una validación de los campos obligatorios. Si todos los campos requeridos están completos y correctos, el formulario crea un objeto Recipe con la información proporcionada.

La aplicación luego **se comunica con el servicio *UploadRecipesService*** para subir la receta a Firebase. Primero, se verifica si el usuario está registrado y ha verificado su correo electrónico. Si el usuario cumple con estos requisitos, se genera un nuevo ID para la receta y se sube la imagen a *Firebase Storage*. Una vez que la imagen se ha subido correctamente, se obtiene la URL de la imagen.

La información de la receta, junto con la URL de la imagen, se prepara en un mapa de datos. Estos datos se envían a *Firestore*, donde se guarda la receta en la colección correspondiente.

Finalmente, la aplicación **actualiza la información del usuario en *Firestore*** para incluir el ID de la nueva receta en la lista de recetas del usuario. Si la receta se sube correctamente, se muestra un mensaje de éxito al usuario y se elimina la receta de los borradores locales, si existía. Si ocurre algún error durante el proceso, se muestra un mensaje de error y la aplicación no cambia el estado de la receta.

2.3.4 Búsqueda e interacción con recetas

Para el siguiente diagrama de secuencia se han querido abarcar los casos de uso 2.2.2, 2.2.3 y 2.2.7 respectivos a los modos de búsqueda de recetas y a la interacción con las mismas. Estos tres **procesos tienen un funcionamiento muy similar**, en el que tan solo se interactúa con la parte del servidor correspondiente a Firestore, además de que el proceso de interacción con las recetas **ocurre secuencialmente** después del proceso de búsqueda, por lo que unir estas funcionalidades en un solo diagrama de secuencia resulta más intuitivo.

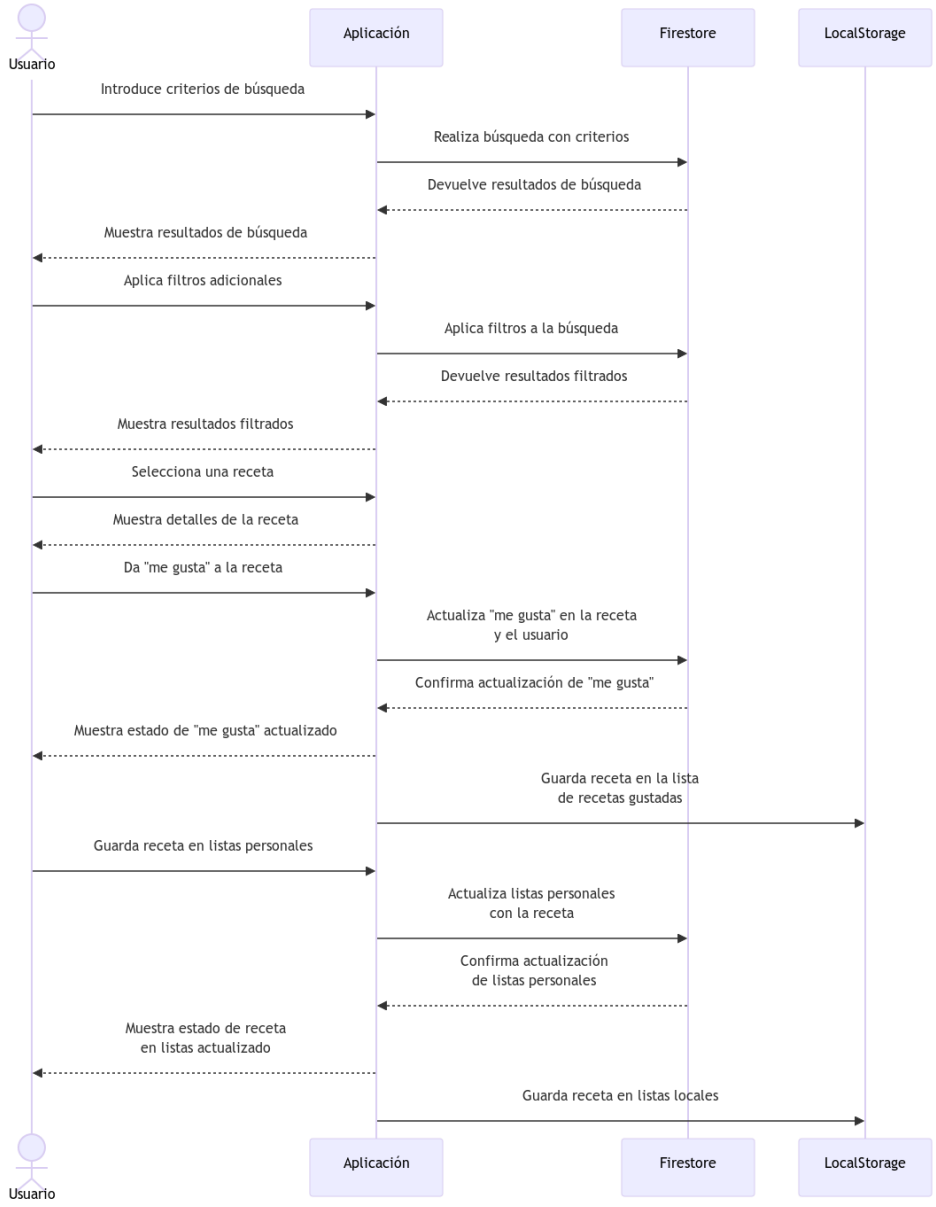

Ilustración 2.12: Diagrama de secuencia para la búsqueda e interacción con las recetas

El proceso de búsqueda e interacción con recetas comienza cuando el usuario introduce criterios de búsqueda en la aplicación. La aplicación **envía estos criterios a *Firestore***, que devuelve los resultados correspondientes. El usuario puede aplicar filtros adicionales, y la aplicación vuelve a consultar *Firestore* para obtener los resultados filtrados. Una vez que el usuario selecciona una receta, puede interactuar con ella dando "me gusta" o guardándola en listas personales. Ambos tipos de interacción actualizan tanto *Firestore* como el almacenamiento local de la aplicación para reflejar los cambios realizados por el usuario.

2.4 Diseño de la base de datos

En el diseño de la base de datos se ha necesitado pensar principalmente en cómo definir las entidades clave que serán fundamentales para su funcionamiento: **un ingrediente, una receta, un usuario y una lista de recetas**. Estas entidades marcan el comportamiento del sistema, ya que definirán la forma en la que actúan los distintos sistemas.

Dado que la estructura de la base de datos que se va a manejar en el servidor es *Firestore* [6], que es una base de datos no relacional, **se utilizará una relación de muchos a muchos**. Este enfoque es particularmente adecuado para este proyecto porque permite una mayor flexibilidad y escalabilidad en la gestión de datos, eliminando la necesidad de tablas adicionales de intersección, típicas en las bases de datos relacionales.

En una relación de muchos a muchos, un documento en una colección puede estar relacionado con múltiples documentos en otra colección, y viceversa. Esto se logra mediante el uso de identificadores o referencias entre documentos de diferentes colecciones. Por ejemplo, en la aplicación, **un usuario puede tener varias recetas guardadas en sus listas**, y **una receta puede pertenecer a múltiples listas de diferentes usuarios**. Firestore facilita este tipo de relaciones a través de su estructura de colecciones y documentos, permitiendo manejar eficientemente estos enlaces.

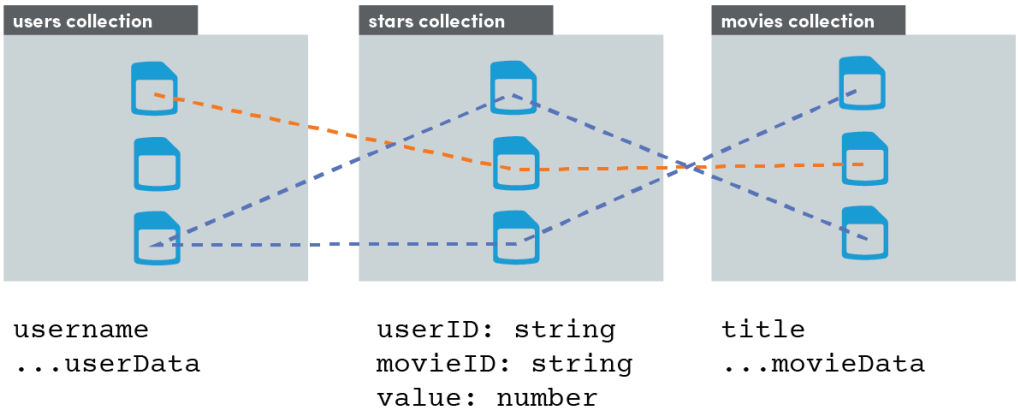

Ilustración 2.13: Ejemplo de una relación de muchos a muchos

Las ventajas de utilizar relaciones de muchos a muchos en *Firestore* incluyen la simplificación del diseño y la gestión de los datos. En lugar de tener que crear y mantener tablas de intersección, *Firestore* **permite almacenar referencias directamente en los documentos**. Esto no solo facilita las consultas y las actualizaciones, sino que también mejora el rendimiento al reducir la complejidad de las operaciones de base de datos.

Además, *Firestore* proporciona una capacidad de consulta que permite realizar búsquedas avanzadas y operaciones complejas directamente sobre los documentos. Esto es crucial para la funcionalidad de la aplicación, donde se necesita realizar búsquedas eficientes de recetas, optimizando tanto el rendimiento como la experiencia del usuario.

A continuación, se presenta una tabla con los campos de cada una de estas entidades, detallando los atributos que las componen y que son esenciales para su correcta implementación.

Campo	Tipo de Dato	Descripción
id	String	Identificador único del ingrediente.
idIngredient	String	Identificador del ingrediente en las listas.
name	String	Nombre del ingrediente.

icon	String	Icono asociado al ingrediente.
amount	int	Cantidad del ingrediente.
unit	String	Unidad de medida del ingrediente.
date	DateTime	Fecha de caducidad asociada.
barcode	String	Código de barras del ingrediente.
head	bool	Indica si el ingrediente tiene subcategorías o es único.

Tabla 2.10: Campos de la tabla Ingrediente

Los ingredientes son **el único tipo de objeto que se gestionará exclusivamente de forma local** y no en el servidor, ya que todas las funciones relacionadas con las listas de productos y notificaciones se manejan localmente. Este objeto se utilizará tanto para la lista de ingredientes definida en la aplicación como para las listas de productos de despensa y cesta.

Por esta razón, cada ingrediente tiene dos identificadores: el *id* y el *idIngredient*. El identificador principal *id* **identifica el tipo de ingrediente** y se utilizará para buscar recetas por ingredientes. Este id es el mismo para todos los productos del mismo tipo, asegurando la coherencia y permitiendo realizar búsquedas eficientes.

Por otro lado, el *idIngredient* **es un identificador único para cada producto específico en la despensa o cesta**. Esto permite que un usuario pueda tener múltiples instancias del mismo tipo de ingrediente en su despensa, con cada instancia teniendo su propio *idIngredient*. Sin embargo, cuando se realizan búsquedas de recetas, solo se utiliza el id principal, de modo que se mantiene la consistencia y se evita la duplicación en los resultados de búsqueda.

Campo	Tipo de Dato	Descripción
id	String	Identificador único de la receta.
emailUser	String	Correo del usuario creador.
name	String	Nombre de la receta.

description	String	Descripción de la receta.
createdAt	String	Fecha de creación.
instructions	List<String>	Instrucciones de preparación.
quantities	List<double>	Cantidades de ingredientes.
ingredients	List<String>	Lista de identificadores de ingredientes.
units	List<String>	Unidades de los ingredientes.
numPersons	int	Número de personas de los ingredientes.
image	Uint8List	Imagen de la receta.
likes	List<String>	Usuarios que dieron "me gusta".
minutes	int	Tiempo de preparación (minutos).
mealtime	List<String>	Momento del día (p.ej., desayuno, cena).
tags	List<String>	Etiquetas (p.ej., vegano, saludable).
region	String	Región de la receta (opcional).
visible	bool	Visibilidad de la receta.
youtubeUrl	String	Enlace a video de YouTube (opcional).
tiktokUrl	String	Enlace a video de TikTok (opcional).
instagramUrl	String	Enlace a video de Instagram (opcional).
otherUrl	String	Otro enlace a video (opcional).

Tabla 2.11: Campos de la tabla Receta

Cada receta contiene una lista de identificadores de ingredientes, lo que permite **asociar fácilmente los ingredientes con las recetas**. Esto facilita la búsqueda y la gestión de recetas basadas en los ingredientes disponibles.

Para fines de moderación, cada receta tiene un campo booleano *visible*, que por defecto está en valor *false* cuando la receta es creada y subida por un usuario.

Los moderadores tendrán acceso a una lista de recetas pendientes de aprobación, que incluirá todas aquellas recetas con el campo *visible* en *false*. Una vez que los moderadores aprueban una receta, este campo se establece en valor *true*, **permitiendo que la receta aparezca en los resultados de búsqueda**.

La tabla *Recipe* también incluye varios campos opcionales, proporcionando versatilidad al usuario en la creación de recetas. Estos campos opcionales permiten al usuario agregar enlaces a videos (YouTube, TikTok, Instagram, etc.), etiquetas específicas (como vegano, vegetariano, saludable, etc.), y la región de la receta, entre otros. En cuanto a las imágenes de las recetas, estas se subirán a *Firebase Storage* y en la base de datos solo se guardará la ruta a dichas imágenes, **optimizando así el almacenamiento y la gestión de las mismas**.

Campo	Tipo de Dato	Descripción
createdAt	String	Fecha de creación de la cuenta.
email	String	Correo electrónico del usuario.
image	String	Imagen de perfil del usuario.
password	String	Contraseña del usuario.
rol	String	Rol del usuario (p.ej., admin, user).
token	String	Token de autenticación.
username	String	Nombre de usuario.

Tabla 2.12: Campos de la tabla Usuario

La tabla de usuarios maneja la información de los usuarios, incluyendo un campo *token* para la autenticación. Al crear un usuario, se envía un correo de verificación y el rol por defecto es *user* (usuario normal sin permisos extra), a menos que un administrador lo cambie. El **identificador principal** de la cuenta **será el correo electrónico**, aunque también se cuenta con un identificador propio gestionado por *Firestore*.

Similar a las recetas, la foto del usuario se almacena en *Firebase Storage* y en la base de datos solo se guarda la ruta hacia esa foto. Para mayor seguridad, la contraseña del usuario se almacenará como un hash en lugar de la contraseña en texto claro.

Campo	Tipo de Dato	Descripción
name	String	Nombre de la lista de recetas.
descripcion	String	Descripción de la lista.
emailUser	String	Correo del usuario creador.
iconId	String	Identificador del icono de la lista.
idList	String	Identificador único de la lista.
recipes	List<String>	Lista de identificadores de recetas.
visible	bool	Visibilidad de la lista.
likes	List<String>	Usuarios que dieron “me gusta”.

Tabla 2.13: Campos de la tabla Lista de Recetas

La tabla para las listas de recetas está diseñada para gestionar las listas de recetas creadas por los usuarios. En el diseño actual, se utilizan los campos *name*, *idList*, *iconId* y *recipes*, ya que las listas de recetas están pensadas para ser asociadas al usuario que las creó y ser públicas solo para él. Sin embargo, **el diseño está preparado para una futura implementación de listas públicas**, añadiendo campos para indicar su visibilidad, descripción o lista de “me gusta” recibidos.

2.5 Diseño de la interfaz

Para el diseño de la interfaz se ha hecho uso principalmente de sketchs ya que proporcionan una visión general del diseño que se busca en cada parte de la aplicación, además de ofrecer una mayor flexibilidad para diseñar elementos más complejos y personalizables de la interfaz.

El diseño de la interfaz se ha dividido en los distintos mecanismos principales de la aplicación, con el objetivo de **unificar el diseño de las pantallas que están más directamente relacionadas**. En este diseño se han obviado aspectos más planos, como el formulario de creación de recetas o de agregación manual de producto, ya que estos están compuestos principalmente por una sucesión de campos rellenables que no requieren un diseño gráfico complejo. De igual manera, algunas pantallas de ajustes no tendrán ningún diseño previo debido a su naturaleza limitada a unas pocas opciones, lo que reduce la necesidad de personalización y flexibilidad en estos casos específicos.

2.5.1 Estructura principal

La aplicación que se trata de desarrollar dispone de una gran cantidad de opciones y funcionalidades independientes que deben combinarse entre sí a la misma vez que puedan ser percibidos como apartados independientes. Dado estas necesidades se ha decidido **establecer 5 secciones** o pantallas principales que serán accesibles en todo momento desde un menú inferior:

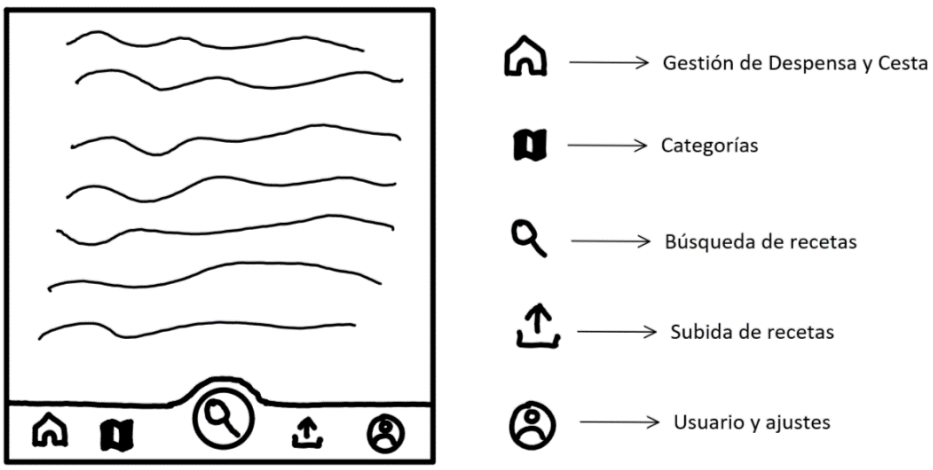

Ilustración 2.14: Sketch del menú principal

En estas secciones se agruparían el resto de funcionalidades principales:

- **Sección de gestión de despensa y cesta**: Está dedicada a la gestión de las listas de productos del usuario, tanto la lista correspondiente a la despensa como la correspondiente a la lista de compra o cesta. Además debe incluir todas las funcionalidades asociadas como creación manual de alimentos, escaneo de códigos de barras de productos, inclusión de fechas de caducidad, cantidades y traspaso de productos entre ambas listas de forma sencilla. Esta sería la pantalla por defecto al abrir la aplicación ya que proporciona los aspectos más básicos de la aplicación.
- **Sección de categorías**: Esta sección sería la más simple de entre todas ya que tan solo contaría con el apartado de búsqueda de recetas por categoría, incluyendo todas las categorías posibles entre las que se pudiesen dividir las recetas.
- **Sección de búsqueda de recetas**: Dentro de esta encontraríamos la búsqueda más específica de recetas, es decir, la búsqueda por los ingredientes que contienen y por nombre de receta. Esta pantalla debería de implementar ambos tipos de búsqueda y para el caso de ser por ingredientes implementar de alguna forma un mecanismo para usar los ingredientes de la despensa en la búsqueda a la misma vez que otro para poder introducir los ingredientes manualmente.
- **Sección de subida de recetas**: Esta pantalla estaría destinada tanto a la subida de recetas, como visualización de las listas de recetas que ya tiene publicadas del usuario y (en caso de existir) la lista de recetas en borradores.
- **Sección de usuario y ajustes**: La última sección como suele ser habitual en otras aplicaciones y redes sociales está dedicada a los ajustes de la aplicación, aspectos relacionados con la cuenta de usuario y también englobaría las listas de recetas creadas por el usuario.

Uno de los aspectos a resaltar puede ser el hecho de que la **búsqueda por categoría sea una de las secciones principales**, es importante que esto sea asi ya que si se quiere incluir una gran cantidad de categorías en las que clasificar las recetas y que el usuario pueda acceder a ellas de forma simple no se podrían colocar en ningún otro lugar debido a que saturarían cualquier otra pantalla de opciones. De esta forma si el usuario desea buscar recetas por requisitos específicos como nombre o los ingredientes que contiene se dirigirá a la sección de búsqueda y si quiere buscar recetas que ya estén previamente clasificadas en categorías más generales accedería a la sección de categorías.

Otro punto a destacar es que ya que se tratan de las secciones principales sobre las que se desarrollarán el resto de funcionalidades, como se ha comentado antes, la pantalla por defecto al abrir la aplicación será la de gestión de despensa y cesta pero esta preferencia contará con un apartado en ajustes desde el que podrá cambiarse para que el usuario seleccione cual quiere que sea su sección de apertura.

El diseño de las secciones principales es el observado en el sketch anterior, sin embargo, se han barajado otras opciones que han sido descartadas:

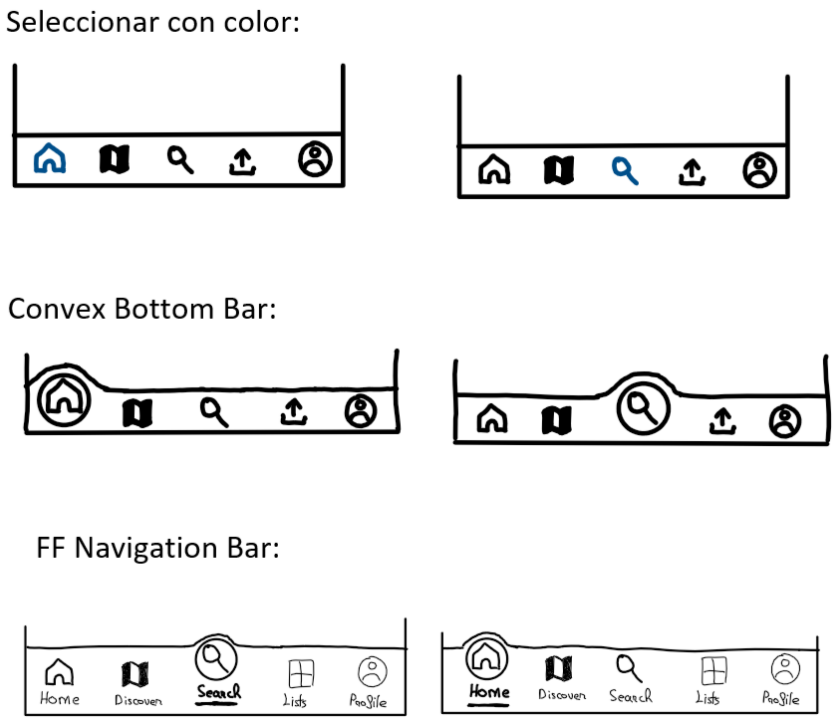

Ilustración 2.15: Sketchs descartados para el diseño del menú principal

También **se barajaron otras opciones para las secciones principales del menú**, como pueden ser el hecho de que la cesta y despensa fueran dos opciones independientes o de que hubiese un apartado específico para las listas de recetas del usuario. Para el primer caso se decidió unir ambas listas de productos en una sola sección debido a que las opciones en ambas listas son muy similares, además de que se quiere hacer percibir al usuario la gestión de productos en un solo lugar de la aplicación, para separarlo de las funcionalidades relacionadas con las recetas, notificaciones o ajustes.

2.5.2 Diseño de despensa y cesta

En esta pantalla se gestionarán dos listas de productos (despensa y lista de la compra o cesta), las cuáles contarán con características muy similares, por lo que deben diseñarse de forma que se **capte claramente en que pantalla de la aplicación nos encontramos**.

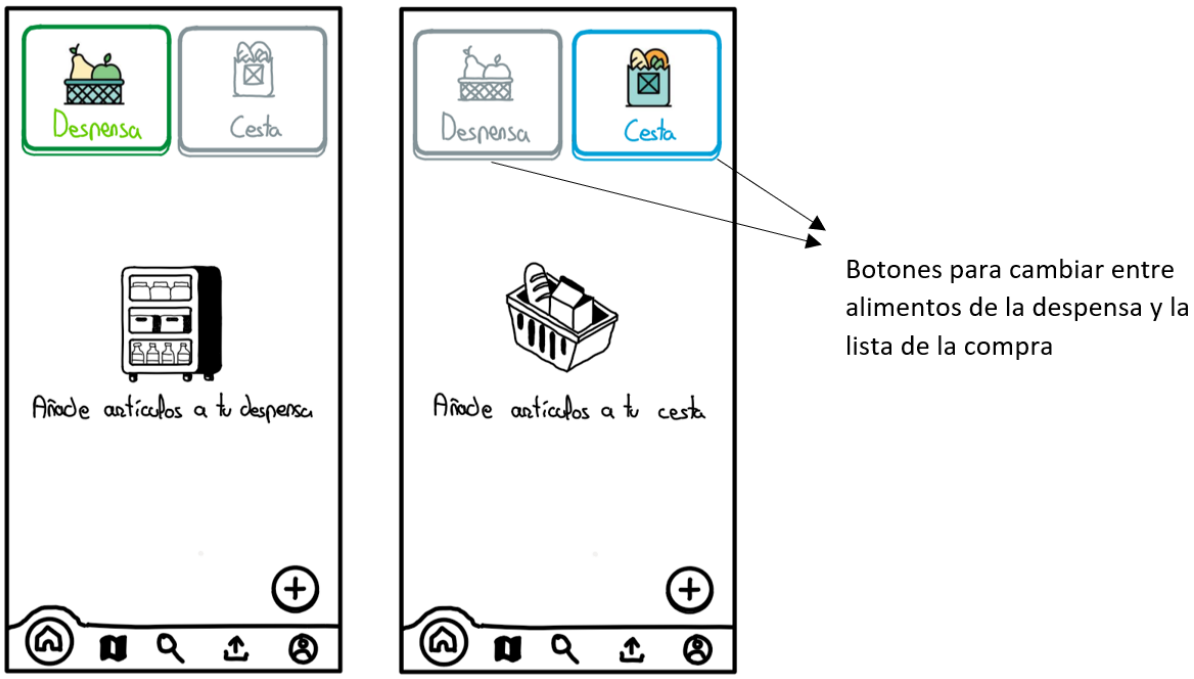

Ilustración 2.16: Sketch de la pantalla de gestión de despensa y cesta

Se ha optado por establecer **dos botones superiores principales** con un título e iconos asociados, que identifiquen rápidamente la sección a la que se refieren y que mediante el color indicarán en que lista nos encontramos, mostrando con color y

sombreada la lista actual y en color gris y menos sombreada la sección en la que no nos encontramos.

Un punto clave para esta pantalla es que el **usuario pueda transferir elementos fácilmente** y de forma intuitiva entre las dos listas, es decir, si tiene un artículo en la lista de la compra y ya lo ha comprado, con un gesto **debería poder pasarlo a la despensa ya que dispondría de él** tras la compra. De la misma forma si tiene un artículo en su despensa que se haya gastado, **debe poder pasarlo con un gesto a la lista de la compra para marcarlo como pendiente de compra**. Para establecer esta funcionalidad se ha decidido colocar una lista a cada lado y que la forma de cambiar productos de lista sea deslizando de un lado a otro:

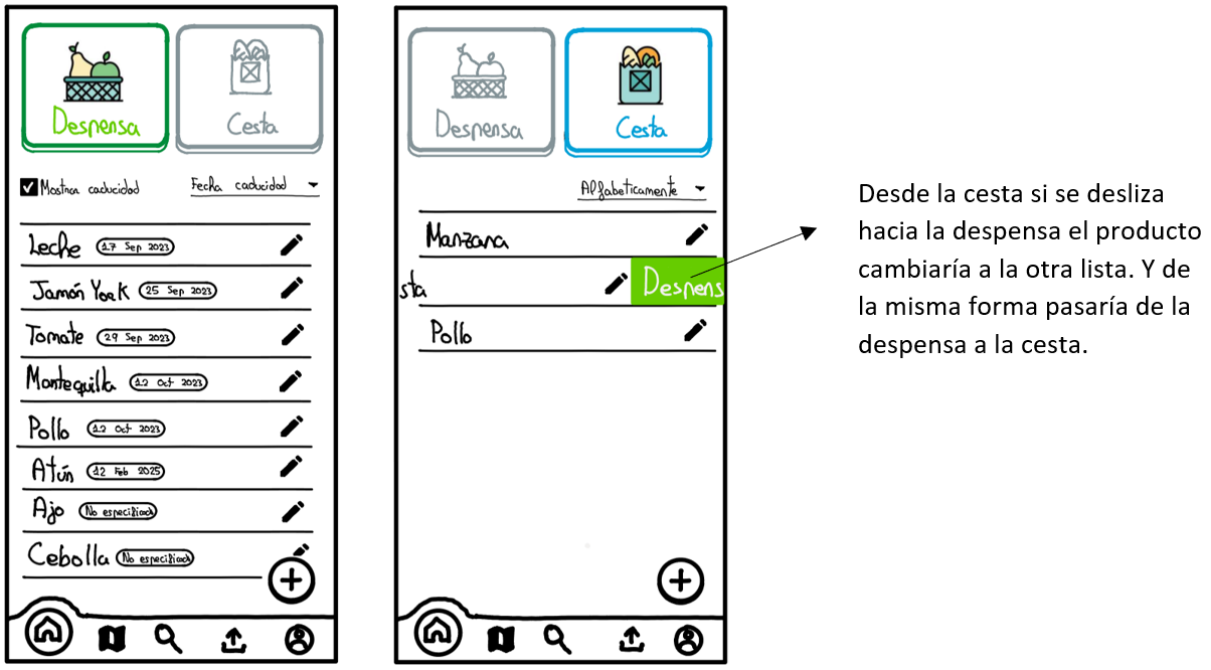

Ilustración 2.17: Sketch de la pantalla de gestión de despensa y cesta con elementos introducidos

Ya que la despensa está a la izquierda y la cesta está a la derecha, si el usuario está en la cesta y quiere pasar un artículo a la despensa, tan solo debe **deslizar el artículo hacia la despensa** para que se cambie de lista. De la misma forma si el usuario se encuentra en la despensa y desea mover un elemento a la cesta **lo deslizaría hacía la cesta**. En caso de querer eliminar un elemento deberá deslizar el

producto hacia la parte dónde no haya ninguna lista, es decir, si está en la despensa hacia la izquierda y si está en la cesta hacia la derecha.

En un primer vistazo puede parecer que este sistema de direcciones no es tan intuitivo pero al tratarse de una funcionalidad que se usará de forma muy recurrente, una vez que el usuario **tenga el modelo mental establecido** de que la izquierda se refiere a la despensa y la derecha se refiere a la cesta, lo más idóneo es que los gestos para mover o eliminar productos acompañen estas direcciones.

A continuación se detallarán los elementos de los que se compone esta pantalla.

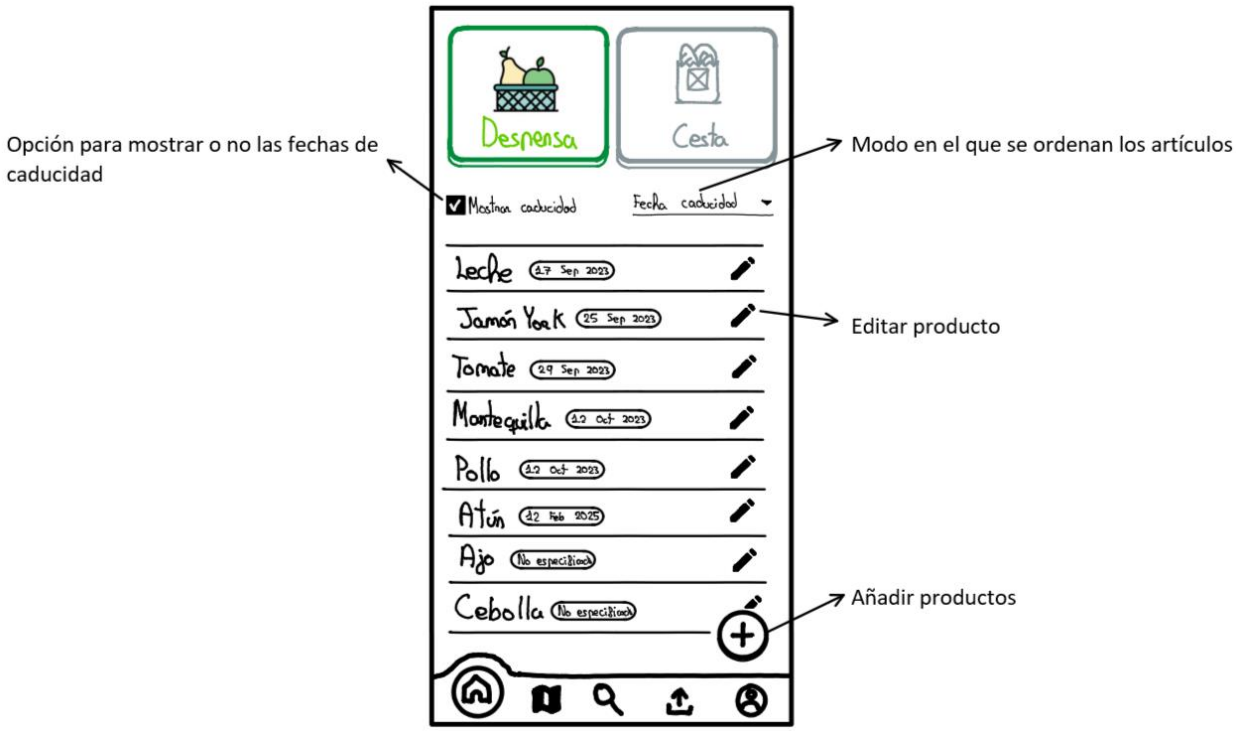

Ilustración 2.18: Sketch de la pantalla de despensa

2.5.3 Diseño del escaneo de productos

Previo al diseño de la propia pantalla de escaneo de códigos de barras, hay que decidir cómo incorporar la opción de añadir productos mediante el escáner en la pantalla de gestión de despensa y cesta. Para esto se ha optado por **modificar el comportamiento del botón de añadir productos**, haciendo que este sea un menú

desplegable dónde se pueda seleccionar entre la adición manual de producto y la adición mediante el uso del escáner.

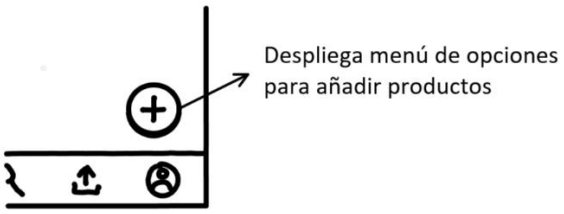

Ilustración 2.19: Sketch del menú de opciones para añadir producto plegado

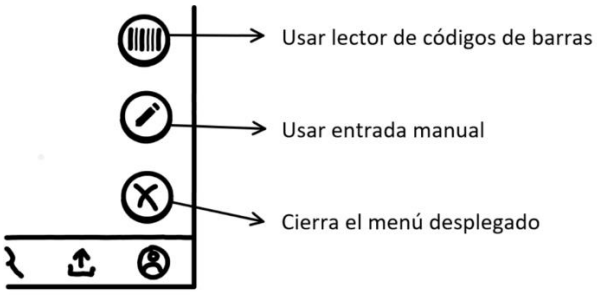

Ilustración 2.20: Sketch del menú de opciones para añadir producto desplegado

De esta forma el usuario tendría tan accesible cualquiera de las dos opciones para añadir un producto.

Respecto al diseño de la propia interfaz, para el escáner del código de barras se ha buscado conseguir una interfaz lo más limpia posible, contando solo con las opciones necesarias para cambiar entre cámaras y activar o desactivar el flash. Una vez se abre esta pantalla ya estaría comenzando a escanear y en cuando detectase un código de barras **automáticamente desplegaría una pestaña inferior con la información** de este producto consultada a la API de *Open Food Facts*, mostrando principalmente su nombre, fotografía y calificación nutricional general.

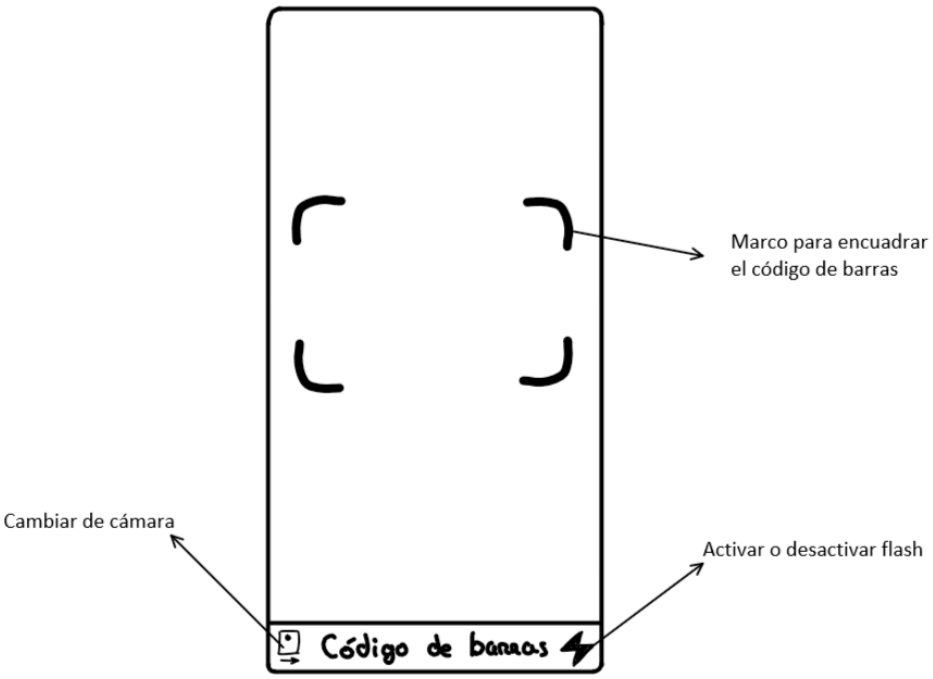

Ilustración 2.21: Sketch del escáner de códigos de barras por defecto

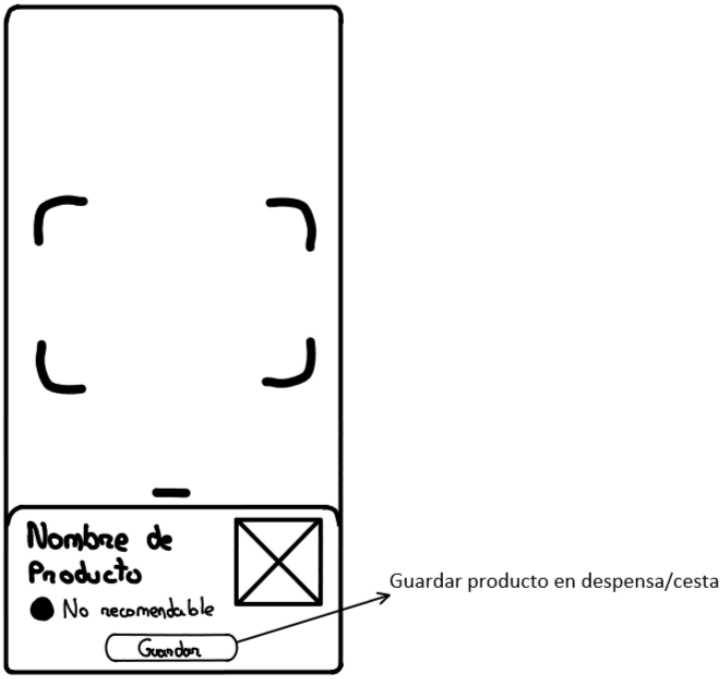

Ilustración 2.22: Sketch del escáner de códigos de barras al reconocer un producto

La pestaña contaría con un botón para guardar el producto en la lista de productos correspondiente, que cerraría el escáner e introduciría el producto automáticamente. En el caso de que se haya reconocido y mostrado un producto y el lector detecte otro código de barras distinto al del producto actual, significaría que el

usuario ha apuntado a otro producto, por lo que se cerraría la pestaña con la información del producto anterior, abriendo una pestaña con la nueva información.

2.5.4 Diseño de categorías

El diseño de la pantalla dónde se muestran las categorías es simple ya que el cometido principal de esta es mostrar una lista con las distintas categorías en las que se podrían clasificar automáticamente las recetas.

Ilustración 2.23: Sketch de la sección de categorías

Las categorías disponibles **estarán formadas en base a los distintos campos que pueden contar los ingredientes**, por lo que cuando un usuario seleccione una, no se estaría accediendo a una lista preestablecida de recetas, si no que estaría realizando una búsqueda de todas las recetas subidas que cumplan con unas determinadas características, es decir, se estaría realizando una búsqueda normal en el servidor.

Ya que cada receta contaría con unos ingredientes reconocidos asociados sobre los que podemos buscar, las categorías principales usarán estos ingredientes para clasificarse. Por ejemplo, si se buscase en la categoría *Pescados y Mariscos* se realizaría una búsqueda de todas las recetas que contengan ingredientes pertenecientes a los pescados o mariscos. De la misma forma, las recetas también

cuentan con una gran cantidad de información asociada que nos puede permitir elaborar una gran cantidad de categorías utilizando estos datos o combinación de los mismos. El diseño actual permitiría ser extensible a ampliar la selección de categorías todo lo posible.

2.5.5 Diseño de las pantallas de búsqueda

La interfaz de la pantalla de búsqueda es un aspecto clave en el diseño de la aplicación, ya que es una de las principales funcionalidades, que **debe tener toda la capacidad de búsqueda posible a la misma vez que debe resultar intuitiva para el usuario**. Se buscaba que el usuario fuese capaz de realizar búsquedas de recetas basadas en el nombre de la receta o basada en los ingredientes por los que se compone. Respecto a la búsqueda por ingredientes además, el usuario debe ser capaz de introducir una lista de ingredientes para la búsqueda de forma manual o usar los ingredientes de los que dispone en su despensa.

Aquí surgen diferentes cuestiones en torno a la organización de estas funcionalidades ya que podríamos optar por **realizar varias pantallas para dividir los modos de búsqueda** y que estas pantallas fuesen iterables entre sí de forma sencilla u optar por **condesar de alguna forma todos los modos de búsqueda en una única pantalla.** Además en caso de querer condensar los modos de búsqueda en una sola pantalla hay múltiples posibilidades de combinar estos elementos entre sí.

Para evaluar las distintas opciones a la hora de realizar esta pantalla se realizaron **varios sketchs con los distintos planteamientos**. Las primeras 3 alternativas corresponden a un planteamiento en el que se condensan los distintos modos de búsqueda en una sola pantalla.

Ilustración 2.24: Sketch de la primera alternativa como la pantalla de búsqueda

Ilustración 2.25: Sketch de la segunda alternativa como la pantalla de búsqueda

Al visualizar estas interfaces podemos observar que resultan funcionales, pero pueden no ser la forma más óptima e intuitiva para el usuario.

Para el caso de las primeras alternativas, en las Ilustración 2.24 y 2.25, el usuario dispondría de una sección para introducir el nombre y otra para introducir los ingredientes, dividida en un botón superior para incluir la despensa, un campo de introducción de texto para buscar ingredientes manualmente y una sección donde aparecerían los ingredientes seleccionados. En la parte inferior de la pantalla se

encontraría el propio botón para iniciar la búsqueda en función de los parámetros introducidos en los campos superiores. Esta alternativa, aunque viable, **puede saturar al usuario**.

La acción de búsqueda **debe de ser una acción fácil y rápida de realizar** ya que es una funcionalidad muy recurrente en la aplicación, por lo que **no debería de ser percibida por el usuario como un formulario a rellenar**. Saturar una pantalla con más de un campo de introducción de texto además de otros botones dan la sensación de tratarse de un formulario o de algo más tedioso que una búsqueda normal.

La aplicación puede contener formularios para funcionalidades más complejas, pero estos deben ser reservados para acciones que se realicen con menos frecuencia, como una creación de cuenta de usuario o la creación de una receta para subir, pero no para algo tan recurrente como una búsqueda.

Ilustración 2.26: Sketch de la tercera alternativa como pantalla de búsqueda

En el sketch de la Ilustración 2.26 se observa como ahora la interfaz resulta mucho más minimalista y simplificada que la anterior, lo que transmite una mejor sensación para el usuario. Sin embargo, **resulta menos intuitiva**. En este diseño se ha tratado de que en un mismo campo de búsqueda se pudiera buscar tanto el nombre de la receta a buscar como los ingredientes a usar. Aunque se podría indicar al usuario

con algún tutorial o indicación el funcionamiento del campo de búsqueda, no resulta del todo intuitivo.

Esto se debe a que los campos de introducción de texto se espera que **funcionen distinto en función de su funcionalidad o modo de búsqueda**. Si el campo de búsqueda está destinado a buscar algo por nombre, el usuario espera que baste con escribir lo que quiere para realizar la búsqueda sin más. Por otro lado, si se quieren introducir distintas palabras como términos de búsqueda, se espera que el campo de introducción de texto funcione como únicamente para buscar esas palabras como criterios de búsqueda.

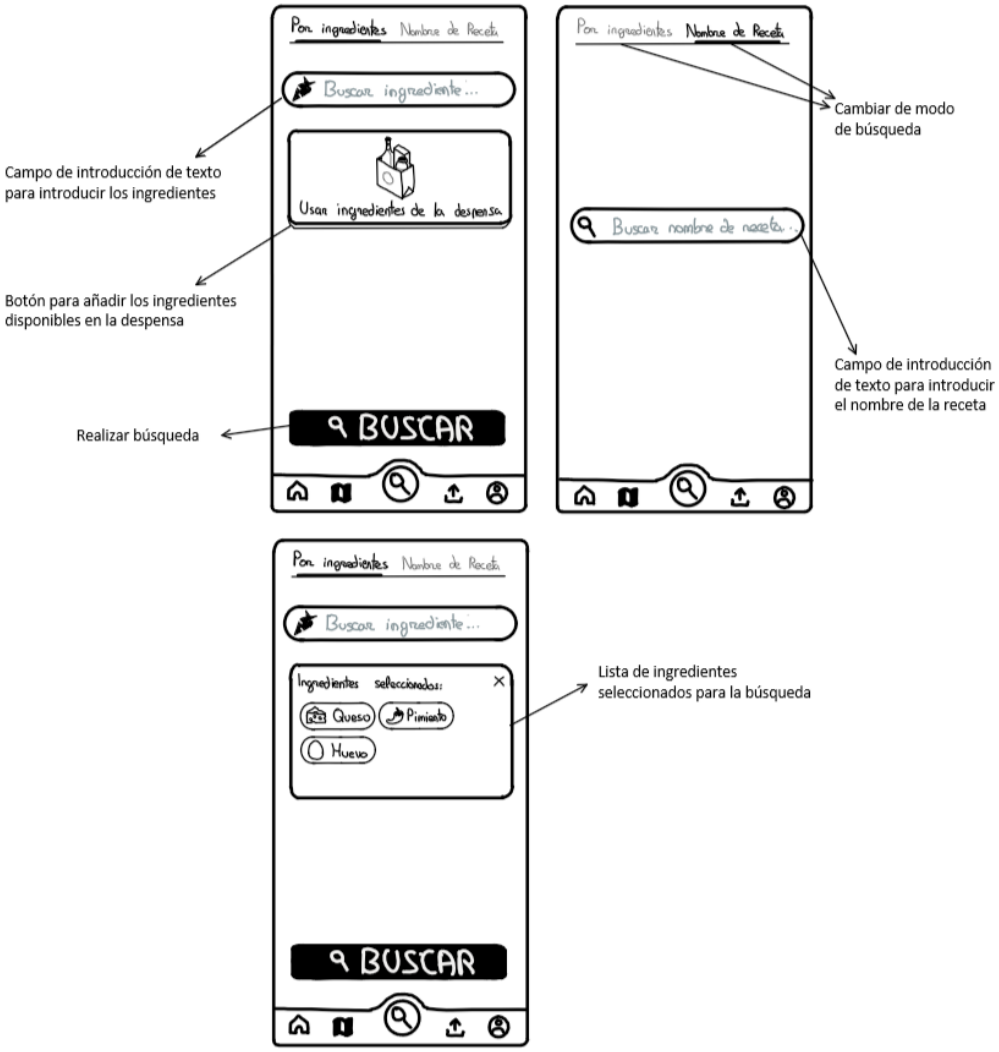

Ilustración 2.27: Sketch de la cuarta alternativa como pantalla de búsqueda

Para solucionar estos problemas, finalmente se ha optado por **dividir la pantalla de búsqueda** en dos pantallas deslizables en función del modo de búsqueda que desearía utilizar el usuario, como se puede observar en la ilustración 2.27.

En esta pantalla contaríamos con dos pantallas independientes para cada modo de búsqueda pero deslizables entre sí, para dar la sensación de que se está en la sección de búsqueda en lugar de en dos secciones distintas de la aplicación. Cada interfaz para cada modo de búsqueda contaría con los elementos justos y necesarios para realizar la búsqueda sin saturar al usuario.

Para el modo de nombre de receta se ha simplificado al máximo, dejando tan solo un cuadro de introducción de texto, dónde el usuario introduciría el nombre de la receta y ya pasaría a visualizar los resultados.

Para el modo de búsqueda por ingredientes se contaría con un campo para buscar los ingredientes a utilizar y con un botón de usar los ingredientes de la despensa. Además, en el momento en el que se introducen ingredientes, el botón de usar ingredientes de la despensa desaparece para dar paso a la lista de ingredientes seleccionados, lo que mejora la identificación de a que corresponde cada elemento.

Otro aspecto a destacar de este diseño es que se ha eliminado el botón de acceso a los filtros que poseían los otros sketchs realizados. Esto es debido a que este pasará a mostrarse en la propia pantalla de resultados de búsqueda, para simplificar más la pantalla de búsqueda principal.

2.5.6 Diseño de resultados de búsqueda y visualización de receta

En la pantalla de resultados se busca mostrar para cada receta la mayor cantidad de información posible sin llegar a saturar al usuario, siendo deseable también incluir opciones para organizar los resultados o aplicar filtros sobre estos de forma sencilla.

Hay que tener en cuenta que como se han definido anteriormente las recetas, estas cuentan con una gran cantidad de datos e información asociadas, por lo que hay que decidir hasta qué punto debería o no mostrarse en esta pantalla.

Lo principal para cada recta es mostrar su nombre y foto de perfil, dado esto, los dos formatos en los que se pueden organizar son horizontal con la foto a un lado y la información al otro o vertical con la foto arriba y la información abajo. En un primer intento se probó a realizar un primer diseño horizontal, como se puede observar a continuación en la Ilustración 2.28.

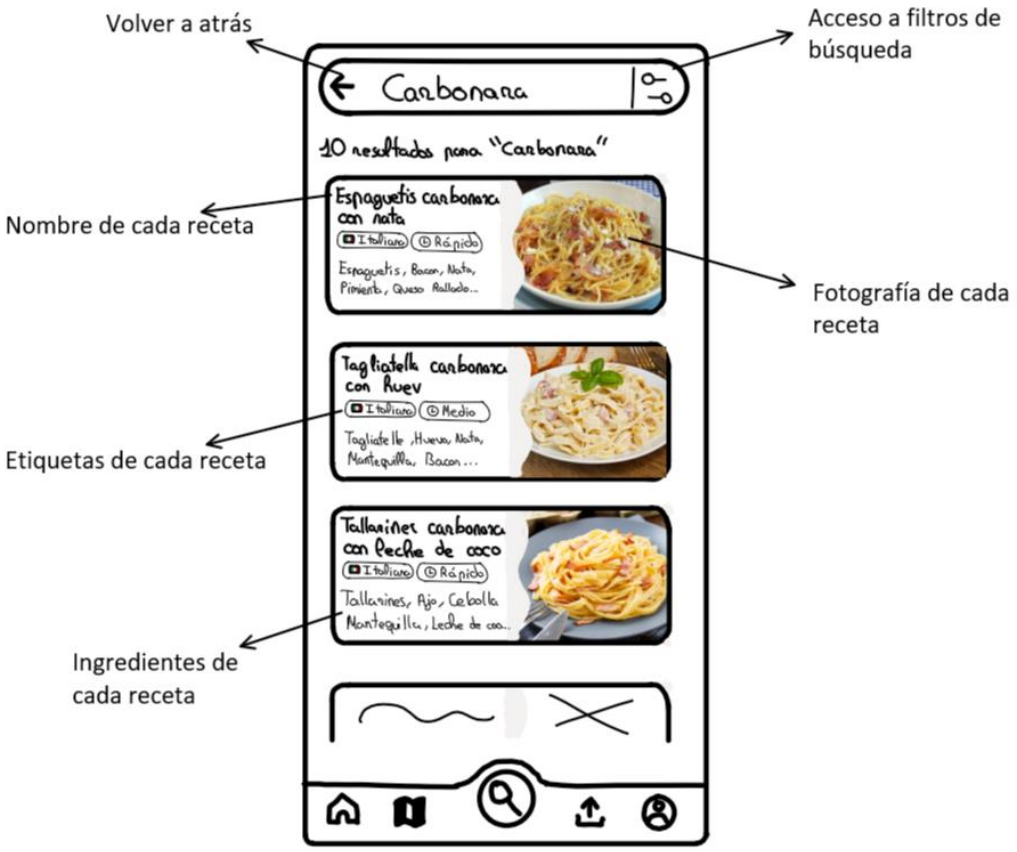

Ilustración 2.28: Sketch de una primera opción como pantalla de resultados de búsqueda

Esta interfaz aunque en un principio cumple con lo que se buscaba, finalmente fue descartada debido a distintos inconvenientes encontrados. En un primer lugar, mostrar varias etiquetas de la receta junto con ingredientes en tan poco espacio no acaba dando una visión clara de los elementos más representativos de la receta a mostrar. **El usuario acaba percibiendo dos columnas de información principales**, una columna a la izquierda donde hay una sucesión de texto y una derecha que se trata con una sucesión de imágenes.

No se trata de ningún problema que impida su cometido principal, pero si de un diseño que podría **mejorarse significativamente si las recetas fuesen más fácilmente distinguibles**. Por eso, se ha elaborado un segundo diseño para esta pantalla que será el que se utilizará como definitivo.

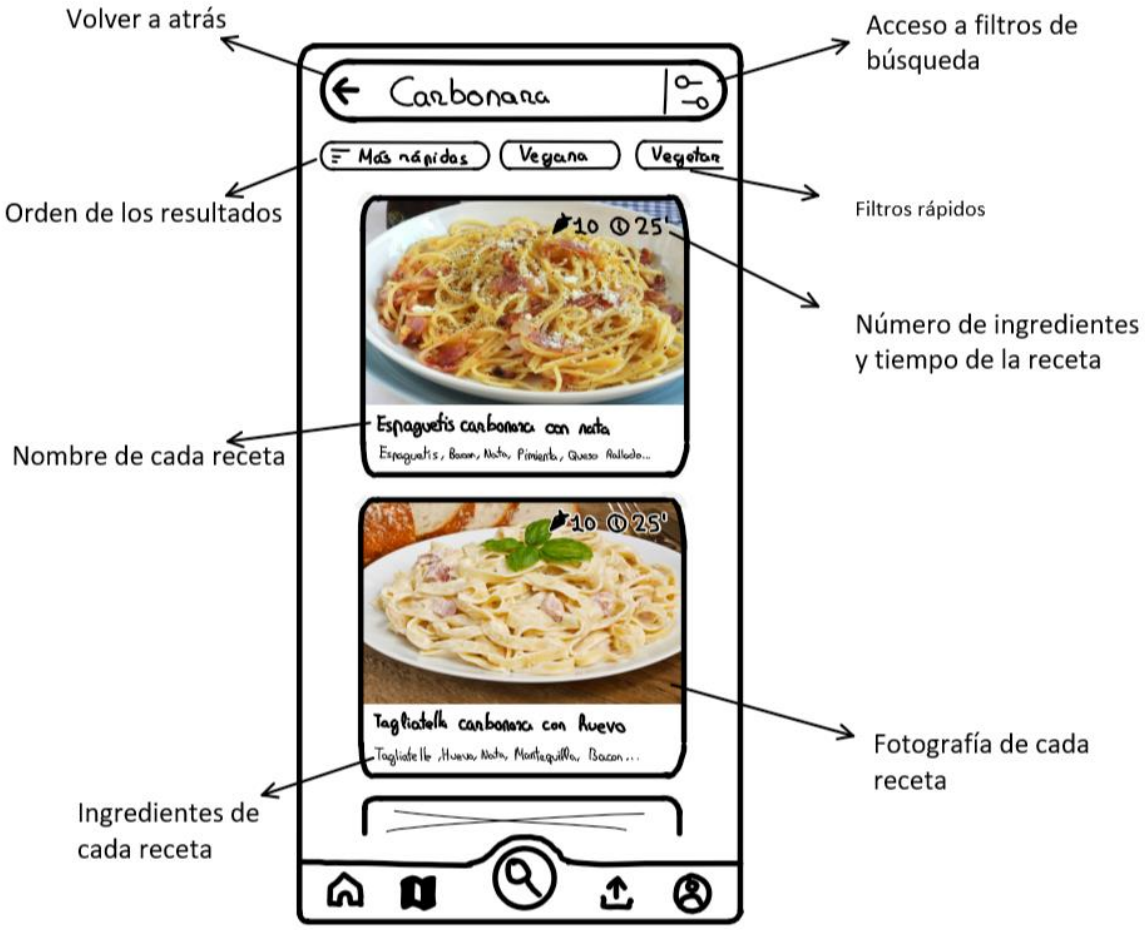

Ilustración 2.29: Sketch de la pantalla de resultados de búsqueda

En este nuevo diseño para la pantalla de resultados observamos como las recetas son mucho más distinguibles entre sí, en lugar de tener todo el texto agrupado a la izquierda y fotos a la derecha, tenemos una columna dónde están todos los elementos en vertical, apreciando más fácilmente la distinción entre elementos. Además, observamos como no toda la información está agrupada en una sola parte del cuadro, si no que se divide en nombre e ingredientes en la parte inferior y número de ingredientes y tiempo en una esquina dentro de la propia fotografía.

Además para mejorar la experiencia de usuario se ha decidido incluir un pequeño menú de ajustes rápidos que hagan que no sea necesario navegar hasta la pantalla de filtros a menos que el usuario busque un uso avanzado de los filtros.

Ilustración 2.30: Sketch de la pantalla de filtros

Debido a que la pantalla de resultados ya muestra unos filtros básicos, en el diseño de la propia pantalla de filtros, que podemos observar en la ilustración 2.30, se ha buscado incluir algunos aspectos mucho más específicos como establecer un tiempo máximo específico de preparación para las recetas o establecer una región a la que pertenezca el plato.

Estos filtros permiten a los usuarios refinar su búsqueda de manera más detallada, **asegurándose de encontrar recetas** que se ajusten perfectamente a sus necesidades y preferencias.

Además, estos **serían extensibles a cualquiera de las características y etiquetas que poseen las recetas**, permitiendo una personalización avanzada en la búsqueda, como filtrar por recetas que contengan vídeo explicativo o por momento del día.

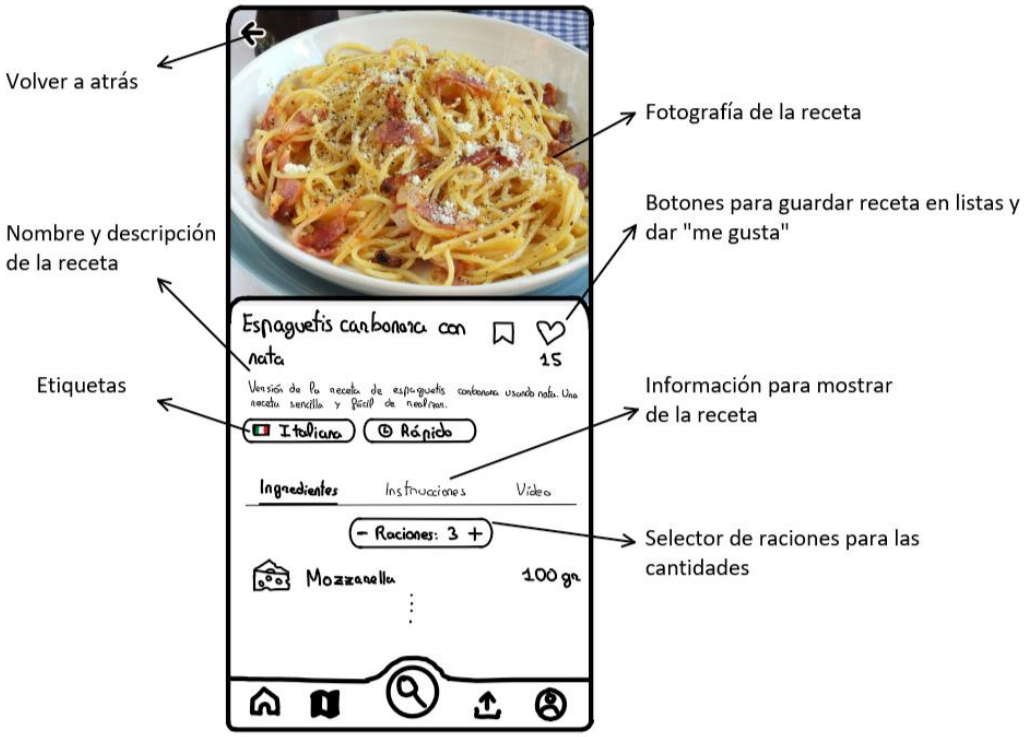

Ilustración 2.31: Sketch de la pantalla de visualización de receta

Para la pantalla de visualización de receta el diseño se ha estructurado para **hacer accesibles todos los detalles** sin que el usuario tenga que deslizar entre una gran lista de características. La pantalla se encontraría dividida en dos partes, dónde en la parte superior encontraría los elementos principales como la fotografía de la receta, el nombre, descripción y etiquetas, junto con las acciones principales para guardar la receta en listas o dar “me gusta”. Por otro lado, **en la parte inferior se encontraría el resto de elementos más específicos** como la lista de ingredientes y sus cantidades, lista de instrucciones y los enlaces a los vídeos. Estas categorías han sido ordenadas en orden de importancia y de consulta, de forma que los ingredientes sean la sección marcada por defecto. Además se han incluido algunos elementos interactivos como un selector para las raciones que modificaría las cantidades asociadas.

2.5.7 Diseño de menús y ajustes

Como se ha mencionado anteriormente la aplicación contendrá multitud de pantallas de ajustes, menús o formularios por lo que en este apartado comentaremos las que resulten de mayor relevancia para el proyecto.

Comenzaremos tratando el diseño de la pantalla de usuario, es una de las pantallas más importantes de la aplicación porque contendrá toda la información personal asociada al usuario, por lo que puede resultar de acceso frecuente. Entre las funciones que debe desempeñar esta pantalla se encuentran la de mostrar toda la información de la cuenta personal, acceso a ajustes de la aplicación, acceso a recetas favoritas del usuario (recetas a las que se le ha dado “me gusta”) y acceso a las listas de recetas creadas por el usuario, asi como el acceso a la creación de las mismas.

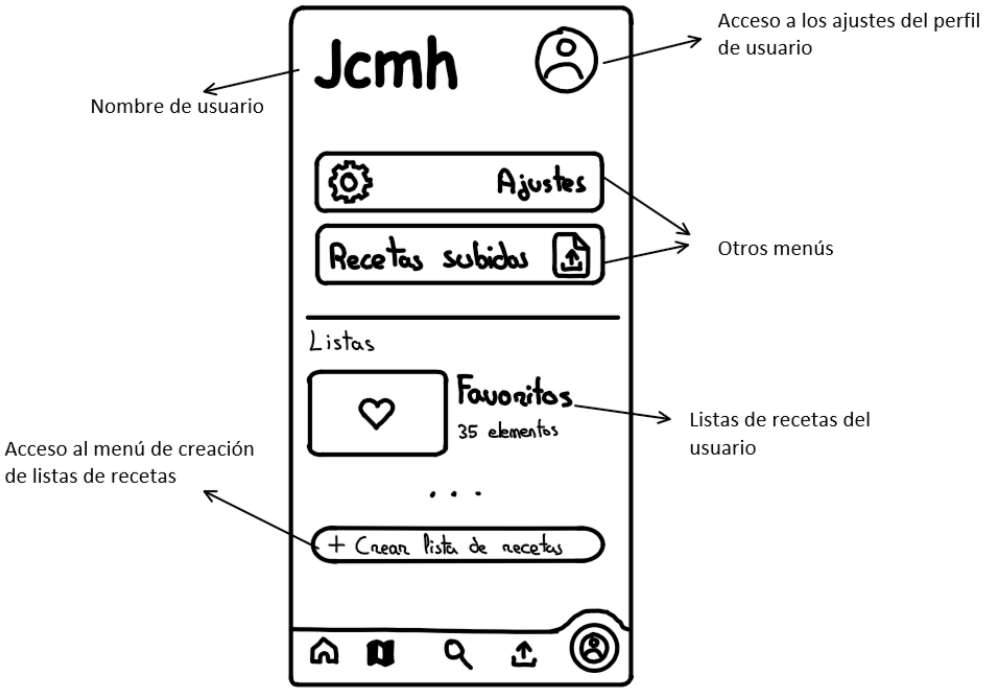

Ilustración 2.32: Sketch de la pantalla de usuario

Cómo se observa en la ilustración 2.32 en la parte superior se dispondrá el nombre y foto de perfil de la cuenta, que a la misma vez servirá como un botón de acceso a los ajustes de la cuenta, comunicando al usuario que esta pantalla está destinada a aspectos relacionados con la cuenta personal al mostrarse elementos propios. Justo debajo encontramos el botón de ajustes y se dispondrá de algún botón

más que corresponda a algún menú o ajuste recurrente para que no se tenga que redirigir siempre a la pantalla de ajustes general para acceder.

Debajo de los elementos relacionados con cuenta y ajustes nos encontraríamos con las listas de recetas creadas personales, dónde para receta se mostrará un nombre, icono asociado y número de elementos que contiene. Justo en la parte inferior a las recetas encontraremos un botón destinado a crear listas de recetas, el cual desplegará un menú para esto como se puede observar en la ilustración 2.33.

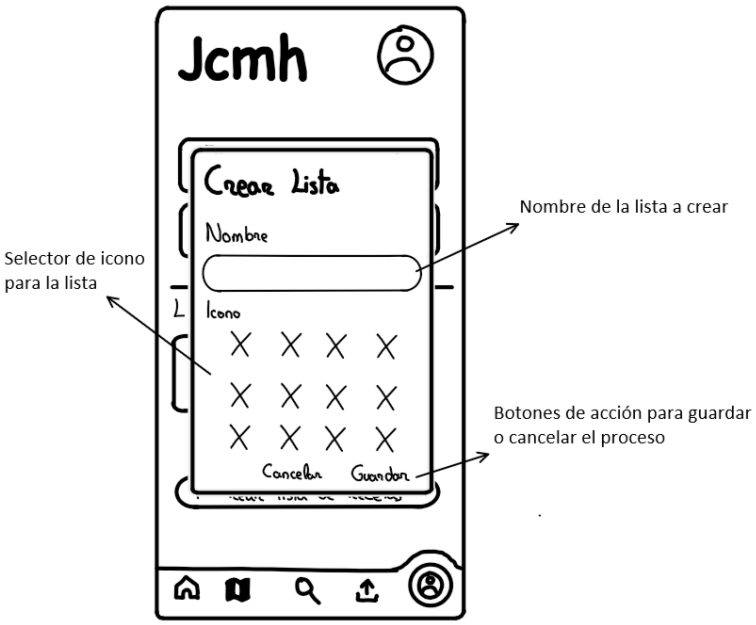

Ilustración 2.33: Sketch del menú de creación de listas de recetas

Otro elemento a destacar sería el previamente mencionado botón para acceder al perfil del usuario, cuyo menú podemos ver en la Ilustración 2.34.

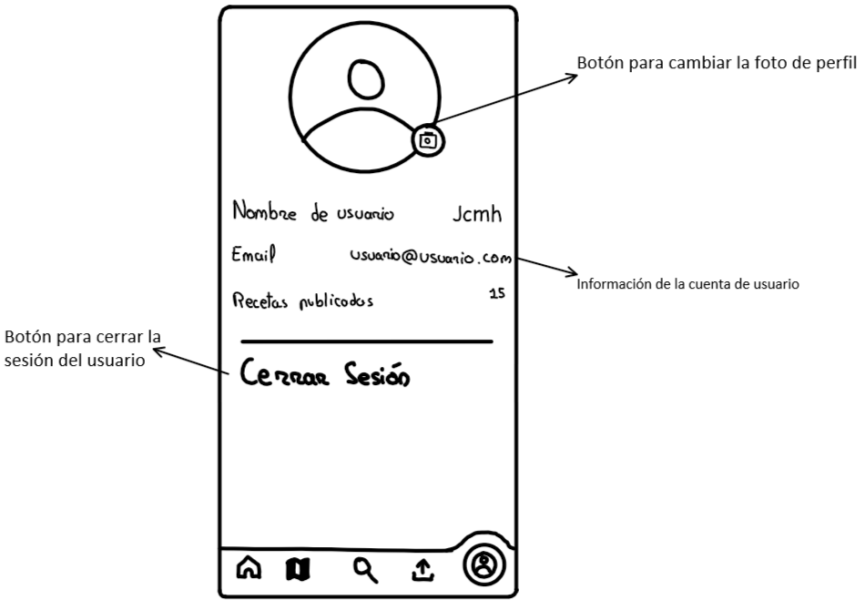

Ilustración 2.34: Sketch de la pantalla de perfil

Por último, se ha realizado el diseño de los formularios de inicio de sesión y registro en la aplicación, que podemos observar en las ilustraciones 2.35 y 2.36.

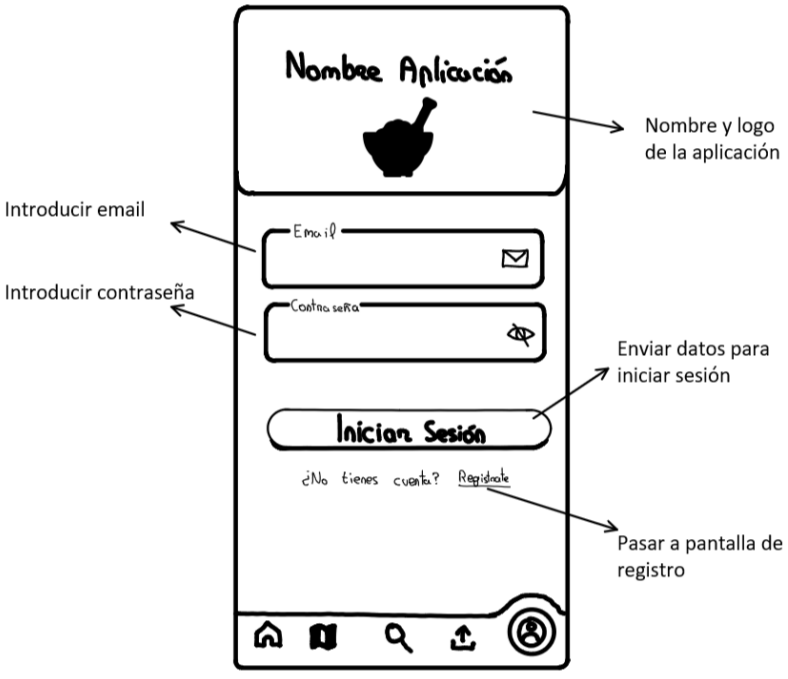

Ilustración 2.35: Sketch de la pantalla de inicio de sesión

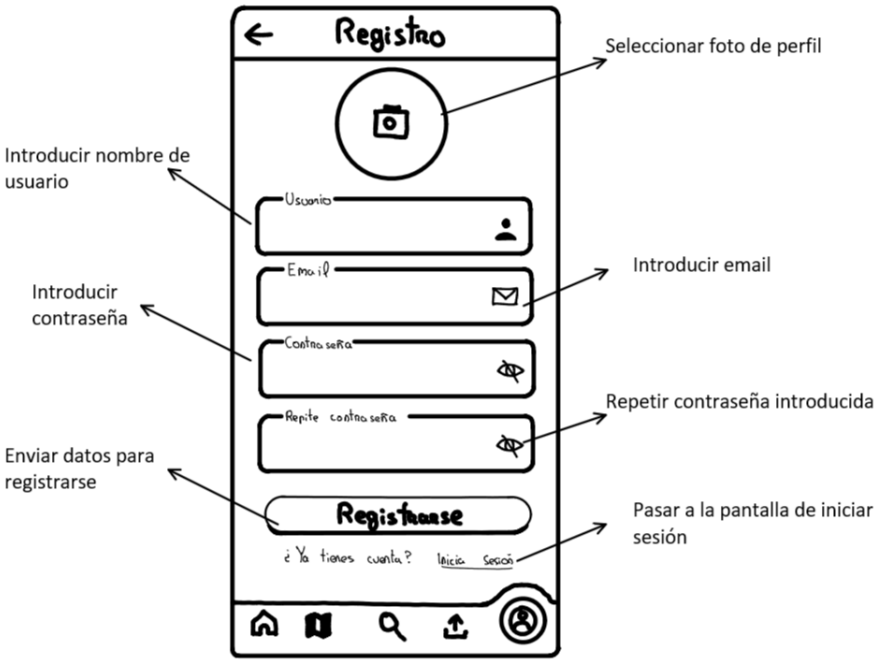

Ilustración 2.36: Sketch de la pantalla de registro

3 DESARROLLO

Como se desarrolló en el apartado 1.11 (Metodología de desarrollo de software), la metodología a utilizar en este proyecto es Scrum. Esta metodología ágil permite dividir el desarrollo en iteraciones llamadas sprints, cada una de las cuales se centra en entregar incrementos funcionales del producto.

El proceso de desarrollo **se documentará organizándose en dos grandes secciones**: la primera narrará la ejecución del desarrollo en iteraciones o sprints, mientras que la segunda detallará las funcionalidades presentes en la aplicación en profundidad.

La primera parte, en el apartado 3.1, se centrará en la ejecución del desarrollo mediante la metodología Scrum. A lo largo de esta sección, se describirá cómo el proyecto fue evolucionando poco a poco a través de desarrollos incrementales. Las funcionalidades desarrolladas se tratarán de forma más genérica sin profundizar demasiado ya que se pretende que en este apartado se centre en la propia evolución del proyecto.

La segunda parte, en el apartado 3.2, estará dedicada a las funcionalidades desarrolladas en la aplicación. En esta sección se detallará cada funcionalidad, explicando su desarrollo en profundidad y su implementación en la aplicación. Además, se evaluarán los resultados obtenidos para mostrar cómo se garantiza el rendimiento y la calidad de la aplicación.

3.1 Ejecución del desarrollo en sprints

A continuación pasarán a desarrollarse la ejecución de los distintos sprints que fueron definidos previamente en el apartado 1.11.3 (Sprints de Desarrollo). Para llevar más claramente las tareas asociadas a cada sprint, se hará uso de la tabla de historias de usuario que fue planteada inicialmente en el apartado 1.12 (Estimación del tamaño y esfuerzo). En dicha tabla se indicarán las **historias de usuario a desarrollar en naranja** y las **historias ya desarrolladas en sprints anteriores en verde**, junto con sus puntos de usuario, para poder visualizar el trabajo ya realizado y por realizar en cada incremento

Historia	Puntos de historia
Historia de usuario ya desarrollada	x
→ Historia de usuario por desarrollar	x
→ Historia de usuario por desarrollar	x
…	

Tabla 3.1: Ejemplo de tabla que se usará en cada sprint

Junto con esta tabla, en cada incremento **se mostrará un diagrama de Burndown**. El diagrama de Burndown es una herramienta visual utilizada en Scrum para hacer seguimiento del progreso del sprint. Muestra gráficamente la cantidad de trabajo pendiente en relación con el tiempo restante del sprint.

3.1.1 Sprint 1: Diseño, planificación y configuración inicial

Durante este primer sprint se pretende dar un comienzo al proyecto, realizando el diseño del proyecto, definición de objetos principales e implementación de la base del proyecto para la aplicación. Todo esto está recogido en una sola actividad con una puntuación de 6 puntos de historia como se puede observar en la tabla 3.2.

Historia	Puntos de historia
→ Diseño de la interfaz, planificación y configuración inicial	6
Gestión de despensa y cesta	6
Desarrollar lista de ingredientes	4
…	

Tabla 3.2: Puntos de historia a desarrollar en el sprint 1

Aunque no supuse un esfuerzo considerable respecto a otras actividades del desarrollo, la etapa de diseño ocupó casi todo el primer mes debido a que se busca que la aplicación a desarrollar tenga una gran cantidad de pantallas para cumplir con todas las especificaciones deseadas. El proceso de diseño implicó realizar el diseño de cada una de estas pantallas, relacionarlas entre sí para su posterior y plantear múltiples alternativas de diseño para las pantallas más importantes y su evaluación.

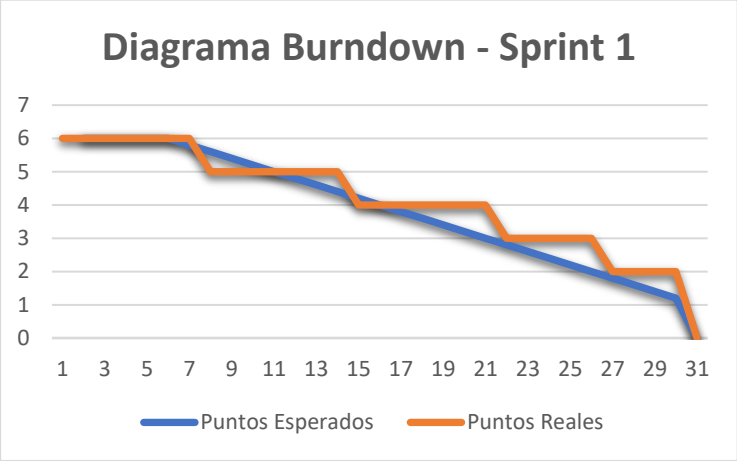

Ilustración 3.1: Diagrama Burndown para el sprint 1

En el diagrama de Burndown de la ilustración 3.1 se puede observar el desarrollo temporal, dónde conforme se iban completando el diseño de un conjunto de pantallas se marcaban puntos de historia completados, resultado que se aprecia en la línea escalonada de puntos reales.

Como resultado de este sprint podemos observar los principales storyboards de la aplicación en las ilustraciones 3.2, 3.3, 3.4 y 3.5 que se muestran a continuación, resultado del proceso de diseño.

Ilustración 3.2: Storyboard de las principales secciones

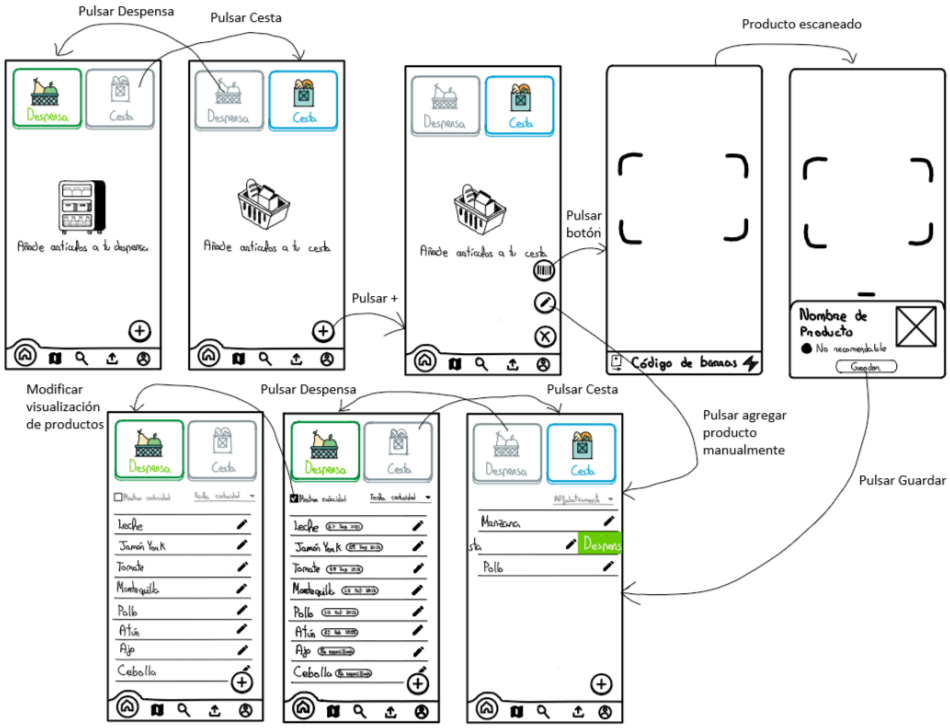

Ilustración 3.3: Storyboard de las pantallas de gestión de listas

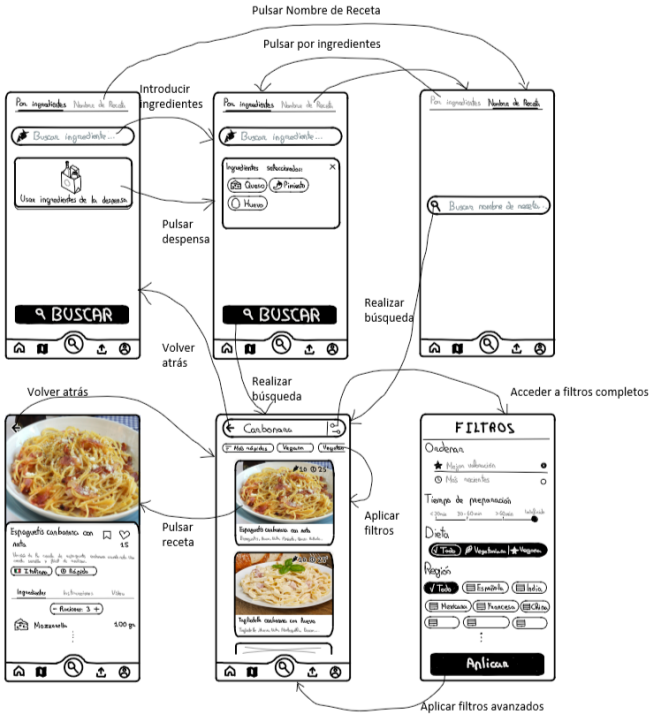

Ilustración 3.4: Storyboard del proceso de búsqueda y resultados

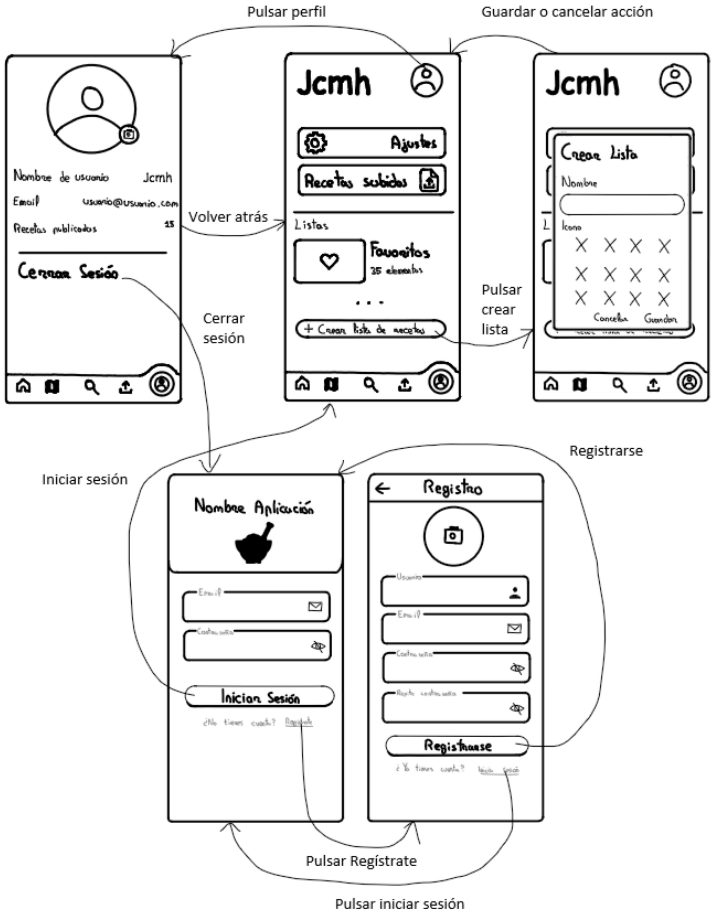

Ilustración 3.5: Storyboard de la gestión de cuenta de usuario

La última parte de este sprint se corresponde a la creación del proyecto y base de la aplicación, repositorio en GitHub y preparación del entorno de desarrollo.

3.1.2 Sprint 2: Implementación de la pantalla principal y lista de Ingredientes

En esta iteración se tratará de elaborar la funcionalidad de gestión de listas principal, obteniendo un sistema usable para la gestión de lista de productos en cesta, lista de la compra con operaciones de adición, edición y eliminación de productos. Además para la adición correcta de productos es necesaria la elaboración de una base de datos o lista de ingredientes predefinidos que cuenten con un nombre, identificador e icono asociados, de forma que cuando el usuario agregue productos seleccione un producto de esta lista y tenga siempre un mismo identificador, como ya se comentó en el apartado 2.4 (Diseño de la base de datos).

Podemos observar los puntos de historia a realizar en esta iteración en la tabla 3.3 mostrada a continuación, dónde encontramos un total de 10 puntos de historia para este sprint.

Historia	Puntos de historia
Diseño de la interfaz, planificación y configuración inicial	6
→ Gestión de despensa y cesta	6
→ Desarrollar lista de ingredientes	4
Funcionalidad para escanear productos mediante el código de barras y consulta a la API	5
…	

Tabla 3.3: Puntos de historia a desarrollar en el sprint 2

Sobre el desarrollo temporal, cabe destacar que mientras la implementación de toda la funcionalidad de gestión de listas de productos y sus pantallas asociadas tienen un mayor esfuerzo asociado, estas se desarrollaron con normalidad durante el tiempo estimado. Sin embargo, la creación de una lista de ingredientes completa con toda su información asociada fue un proceso que se alargó mucho en el tiempo debido a querer realizar una base de datos de ingredientes lo más completa posible.

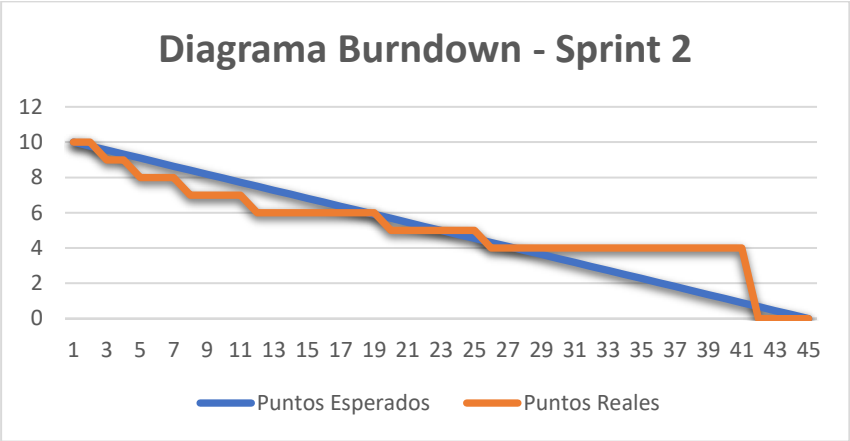

Ilustración 3.6: Diagrama Burndown para el sprint 2

En el diagrama Burndown de la ilustración 3.6 observamos lo mencionado, apreciando como los puntos de historia correspondientes al desarrollo de la funcionalidad se desarrollaron con normalidad en un tiempo similar al esperado, mientras que los puntos correspondientes a la lista de ingredientes se alargaron en el tiempo bastante hasta su finalización.

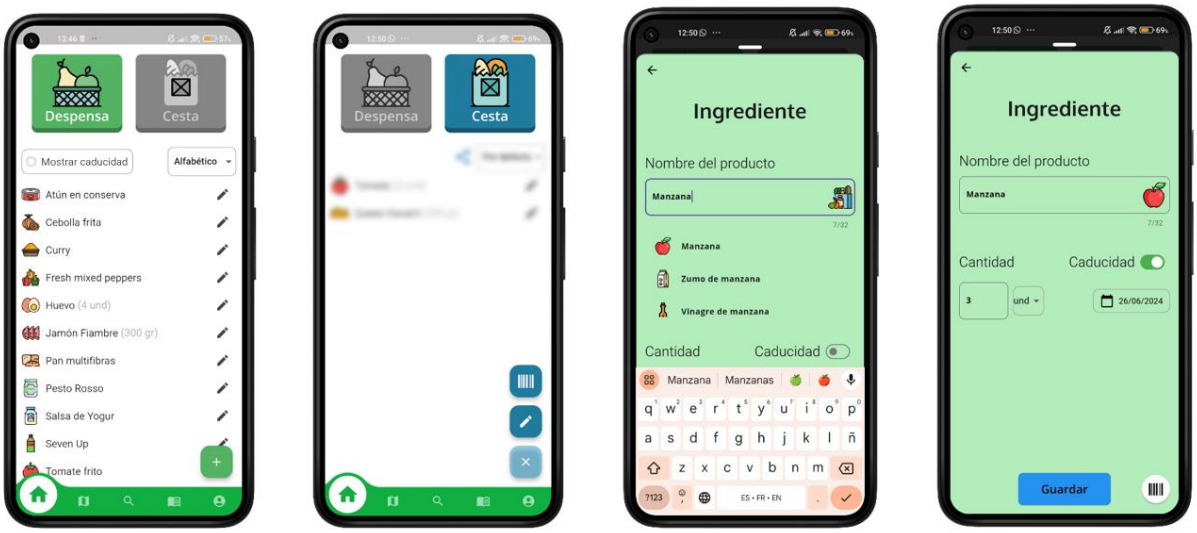

Ilustración 3.7: Algunas capturas de las pantallas resultado del sprint 2

El resultado de esta iteración fue un sistema funcional de gestión de listas de productos, como ser se puede ver en la ilustración 3.7, cuyo funcionamiento se detallará en profundidad en apartados futuros. Además se creó una base de datos de

unos 1000 ingredientes con sus iconos asociados que cubren la mayoría de categorías en alimentación para añadir ingredientes en base a estas sugerencias.

3.1.3 Sprint 3: Lectura de Códigos de Barras y mejora de interfaz

Para esta tercera etapa se espera implementar un sistema que permita a la aplicación leer códigos de barras de productos para analizar su información nutricional y usarse como una forma de agregar productos más rápida y eficaz. Además se espera añadir una pantalla de usuario y realizar algunas mejoras en la interfaz. Estas actividades suponen un total de 10 puntos de historia (5 + 3 + 2), que se desglosan en la tabla 3.4.

Historia	Puntos de historia
Diseño de la interfaz, planificación y configuración inicial	6
Gestión de despensa y cesta	6
Desarrollar lista de ingredientes	4
→ Funcionalidad para escanear productos mediante el código de barras y consulta a la API	5
→ Pantalla de usuario	3
→ Menús y mejora de la interfaz	2
Sistema de control de la caducidad en alimentos	3
…	

Tabla 3.4: Puntos de historia a desarrollar en el sprint 3

La mayor parte del tiempo empleado en esta iteración se utilizó en completar la funcionalidad de escaneo de productos correctamente, ya que se llegó a probar con diferentes paquetes para lograr una función de escaneo lo más fluida posible y se tuvo que investigar sobre el funcionamiento de la API a utilizar.

Se realizaron las actividades previstas por completo con unos tiempos similares a los esperados, con un breve adelanto en el tiempo para el desarrollo de la pantalla de usuario que se comentará a continuación, lo que dejó más tiempo para la implementación de la mejora de interfaz. Observamos estos cambios en el diagrama de Burndown de la ilustración 3.8.

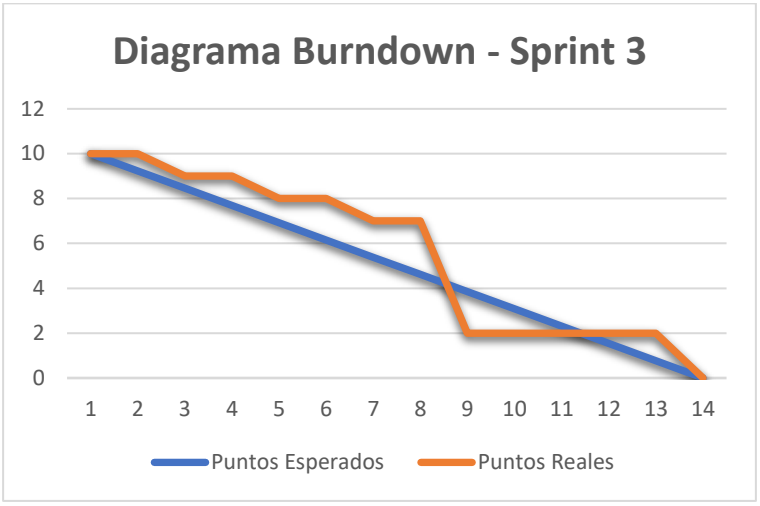

Ilustración 3.8: Diagrama Burndown para el sprint 3

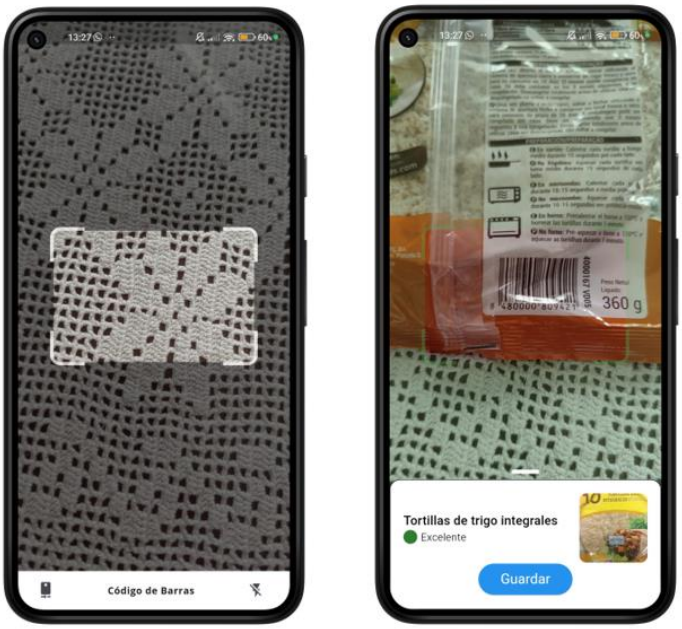

Ilustración 3.9: Capturas del modo de escaneo de productos tras el sprint 3

Además se realizó una pantalla para la sección de usuario que, aunque debería incluir aspectos relativos al perfil del usuario y listas de recetas, estas son funcionalidades que se desarrollarán en etapas futuras, por lo que tan solo se desarolló un esqueleto de la interfaz dónde deberían ser incluidos los elementos restantes conforme fuesen desarrollados.

Al haber ocupado menos tiempo de lo esperado se empleó el tiempo restante en realizar algunas mejoras de la interfaz. Se estableció un tema para la aplicación, definiendo fuentes, iconos, paleta de colores y un modo oscuro general para las pantallas que ya estaban implentadas y para usarse de aquí en adelante en todas las pantallas que se creasen.

Ilustración 3.10: Modo claro y oscuro de la aplicación tras el sprint 3

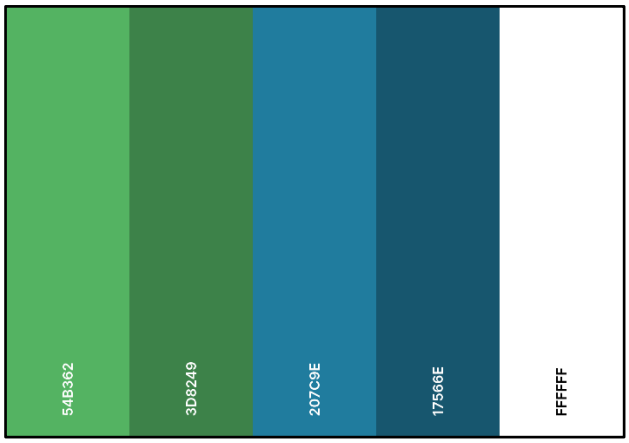

Ilustración 3.11: Paleta de color

3.1.4 Sprint 4: Gestión de notificaciones

El resultado esperado de este sprint es añadir una funcionalidad para permitir establecer una fecha de caducidad a los alimentos de la despensa y establecer notificaciones personalizables y programables para notificar al usuario de la caducidad próxima de estos. Para ello se usarán 11 puntos de historia (8 + 3) como se puede ver en la tabla 3.5.

Historia	Puntos de historia
Diseño de la interfaz, planificación y configuración inicial	6
Gestión de despensa y cesta	6
Desarrollar lista de ingredientes	4
Funcionalidad para escanear productos mediante el código de barras y consulta a la API	5
Pantalla de usuario	3
Menús y mejora de la interfaz	2
→ Sistema de control de la caducidad en alimentos	3
→ Notificaciones programables de caducidad	8
Implementación de Firebase al proyecto	5
…	

Tabla 3.5: Puntos de historia a desarrollar en el sprint 4

La asignación de una cantidad significativa de puntos de historia a la esta implementación se justifica por la complejidad inherente al desarrollo de un sistema de notificaciones locales eficiente. Esta complejidad se deriva principalmente de los requisitos técnicos específicos de la plataforma Android.

El sistema requiere la capacidad de ejecutar tareas en segundo plano en momentos específicos programados por el usuario para verificar la existencia de productos próximos a caducar. En Android, nos enfrentamos a una notable variabilidad en permisos y restricciones que difieren significativamente entre versiones del sistema operativo. Esta diversidad se ve aún más acentuada por las capas de personalización que implementan los distintos fabricantes de dispositivos.

Como resultado, aspectos como los modos de ahorro de batería, las configuraciones de "No molestar" y las limitaciones en la ejecución de procesos en segundo plano pueden interferir tanto en la ejecución de los procesos necesarios como en la visualización de las notificaciones.

Considerando estos factores, fue necesario un período de desarrollo extenso para probar distintos sistemas de notificaciones locales para que fuera compatible y funcional en la mayoría de las versiones de Android y capas de personalización.

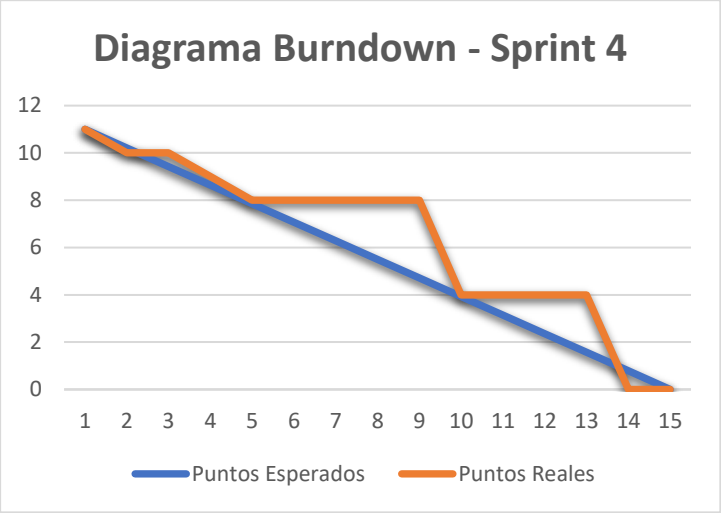

Ilustración 3.12: Diagrama Burndown para el sprint 4

Esta inversión de tiempo se refleja en el diagrama de Burndown de la ilustración 3.12 mostrada a continuación, dónde los “escalones” que encontramos en la línea de puntos reales se ven mucho más acentuados.

Ilustración 3.13: Ejemplo de notificación implementada en el sprint 4

3.1.5 Sprint 5: Integración con Firebase y gestión de cuentas de usuario

Esta iteración es importante ya que supone el comienzo de la implementación de todas las funcionalidades basadas en la interacción con un servidor, cambiando la aplicación de un funcionamiento exclusivamente en local, a pasar a estar coordinada con Firebase, que es el servicio de backend a implementar.

Ya que el resultado de la iteración debe ser un producto usable, no solo basta con la creación del proyecto en firebase y su inclusión en la aplicación, también hay que hay que realizar unas funcionalidades mínimas para que incluirlo en el proyecto

tenga un impacto significativo. Es por esto que en esta iteración además de la configuración de firebase se busca añadir un sistema de creación y gestión de cuentas de usuario, lo que daría a este sprint una puntuación total de 16 puntos de historia (5 + 5 + 6), situándola como la iteración con mayor puntuación hasta ahora.

Historia	Puntos de historia
Diseño de la interfaz, planificación y configuración inicial	6
Gestión de despensa y cesta	6
Desarrollar lista de ingredientes	4
Funcionalidad para escanear productos mediante el código de barras y consulta a la API	5
Pantalla de usuario	3
Menús y mejora de la interfaz	2
Sistema de control de la caducidad en alimentos	3
Notificaciones programables de caducidad	8
→ Implementación de Firebase al proyecto	5
→ Creación de cuentas en Firebase	5
→ Gestión de cuentas de usuarios	6
Implementación y visualización de recetas	4
…	

Tabla 3.6: Puntos de historia a desarrollar en el sprint 5

A pesar de tratarse de la iteración con más puntos de historia hasta ahora, el desarrollo transcurrió con normalidad en el tiempo previsto, como se puede observar en el diagrama Burndown de la ilustración 3.14 mostrado a continuación.

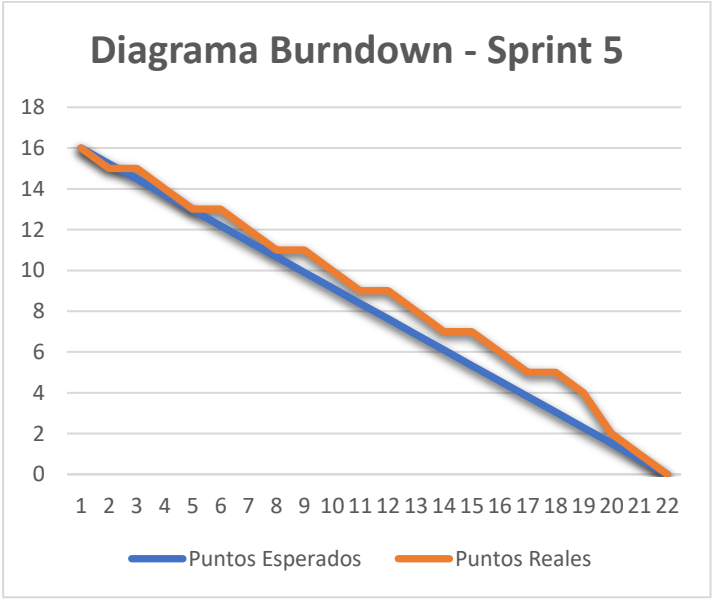

Ilustración 3.14: Diagrama Burndown para el sprint 5

Sin entrar demasiado en detalle de la implementación de Firebase en el proyecto, ya que eso se desarrollará en un apartado independiente dentro del apartado de funcionalidades desarrolladas (3.2), se muestran a continuación capturas de las pantallas para el inicio de sesión y registro creadas en este sprint.

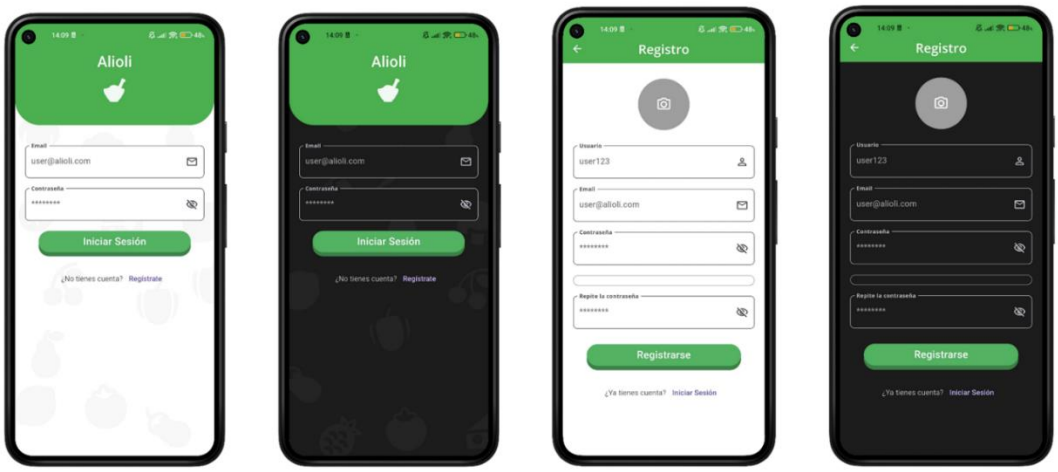

Ilustración 3.15: Capturas de Inicio de sesión y registro tras el sprint 5

Algo a mencionar es que para este punto ya contamos con un sistema de cuentas de usuario funcional en el proyecto, por lo que podemos actualizar la sección de usuario creada en el sprint anterior, para añadir las partes faltantes. En la ilustración 3.16 se puede apreciar la evolución de esta pantalla, mostrando **en la primera captura su estado en el sprint 4** y **en las dos capturas siguientes su estado tras el sprint 5**.

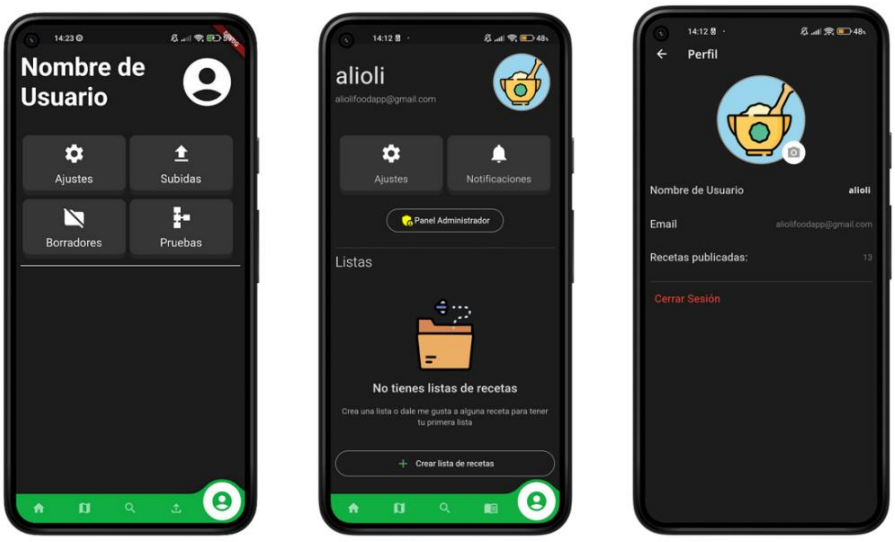

Ilustración 3.16: Evolución de la sección de usuario entre el sprint 4 y 5

3.1.6 Sprint 6: Creación, subida y visualización de recetas

Ya que la aplicación tiene un sistema de cuentas de usuario implementada, durante esta iteración se puede llevar a cabo la funcionalidad que permita a los usuarios crear recetas y subirlas a la plataforma. Junto a esto se debe de implementar la pantalla para la visualización de las recetas que apoye la funcionalidad anterior.

También a la hora de subir recetas surge una necesidad de moderación de contenido, ya que todo lo que suban los usuarios pasará posteriormente a formar parte de los resultados de búsqueda e inclusión en las categorías disponibles, por lo que sería conveniente asegurar de alguna forma que el contenido subido por los usuarios cumpliese algunos estándares antes de pasar a ser público. Para esto hay que añadir a la implementación anterior de cuentas de usuario, un sistema de roles, dónde cada usuario tenga un rol asociado que le otorgue más o menos permisos. Los usuarios nombrados como administrador serían los encargados de visualizar y aprobar las recetas subidas por los usuarios para que pasen a ser públicas.

Las mencionadas actividades a desarrollar se pueden observar en la tabla 3.7 mostrada a continuación, dónde vemos una puntuación total de 14 puntos de historia (4 + 5 + 3 + 2).

Historia	Puntos de historia
Diseño de la interfaz, planificación y configuración inicial	6
Gestión de despensa y cesta	6
Desarrollar lista de ingredientes	4
Funcionalidad para escanear productos mediante el código de barras y consulta a la API	5
Pantalla de usuario	3
Menús y mejora de la interfaz	2
Sistema de control de la caducidad en alimentos	3
Notificaciones programables de caducidad	8
Implementación de Firebase al proyecto	5
Creación de cuentas en Firebase	5
Gestión de cuentas de usuarios	6
→ Implementación y visualización de recetas	4
→ Creación de recetas	5
→ Subida de recetas	3
→ Gestión de distintos roles de usuario y moderación	2

Búsqueda por ingredientes	6
…	

Tabla 3.7: Puntos de historia a desarrollar en el sprint 6

El desarrollo de las actividades a lo largo del tiempo se llevó a cabo según lo esperado como se puede ver en el diagrama Burndown de la ilustración 3.17

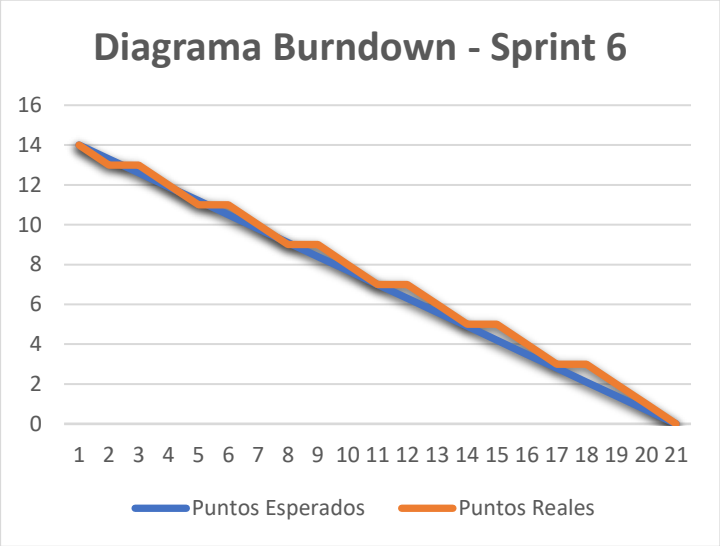

Ilustración 3.17: Diagrama Burndown para el sprint 6

El resultado de esta iteración es un sistema funcional dónde los usuarios pueden crear sus propias recetas con una interfaz intuitiva y subirlas al servidor. Se han añadido los mecanismos necesarios a esta acción, como que estas recetas se queden asociadas a la cuenta del usuario que las ha creado, de forma que si el usuario inicia sesión en otro dispositivo automaticamente se descargarían las recetas asociadas a su cuenta para que aparezca en el apartado de 'mis recetas'.

También se ha incluido la funcionalidad de una lista de borradores, dónde los usuarios pueden optar por guardar la receta creada en borradores para continuar después en lugar de subirla.

Además, cómo se mencionó anteriormente, se ha creado el sistema de roles de usuario. A partir de esta iteración cada usuario tiene un rol asociado y los que estén marcados como usuarios administradores verán en la aplicación un menú adicional

dónde podrán visualizar las recetas subidas por los usuarios pendientes de aprobación y decidir si deben ser publicadas o no. Este menú de administrador además ya que está implementado abre la puerta a desarrollar más funciones de gestión de la plataforma directamente desde la propia aplicación.

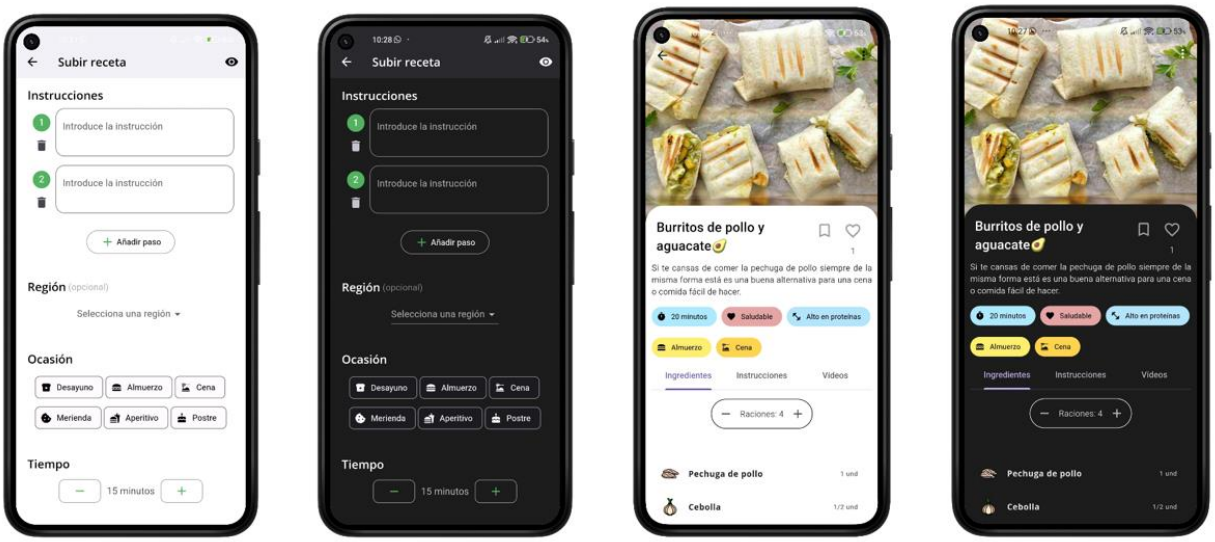

Ilustración 3.18: Capturas de creación y visualización de recetas tras el sprint 6

3.1.7 Sprint 7: Búsqueda y filtrado de recetas

Ya que tenemos una forma de subir recetas al servidor y visualizarlas, el siguiente paso es desarrollar el sistema de búsqueda de recetas completo. Durante este sprint se desarrollarán estas características, incluyendo algunas adicionales como la implementación eficiente de filtros basados en las etiquetas y características de cada receta.

Además una vez desarrollado el sistema de búsqueda y filtros, es conveniente incluir en esta iteración el desarrollo de una búsqueda por categorías ya que esta aprovechará las funciones ya desarrolladas de filtrado y búsqueda, por lo que también lo incluiremos como una actividad a desarrollar.

Todas estas especificaciones suman un total de 22 puntos de historia (6 + 6 + 4 + 6) como se puede ver en la tabla 3.8 mostrada a continuación.

Historia	Puntos de historia
Diseño de la interfaz, planificación y configuración inicial	6
Gestión de despensa y cesta	6
Desarrollar lista de ingredientes	4
Funcionalidad para escanear productos mediante el código de barras y consulta a la API	5
Pantalla de usuario	3
Menús y mejora de la interfaz	2
Sistema de control de la caducidad en alimentos	3
Notificaciones programables de caducidad	8
Implementación de Firebase al proyecto	5
Creación de cuentas en Firebase	5
Gestión de cuentas de usuarios	6
Implementación y visualización de recetas	4
Creación de recetas	5
Subida de recetas	3
Gestión de distintos roles de usuario y moderación	2
→ Búsqueda por ingredientes	6
→ Búsqueda por nombre	6
→ Búsqueda por categoría	4
→ Aplicar filtros de búsqueda	6
Sistema de valoración basado en "me gustas" de los usuarios	5
Creación y gestión de listas de recetas personales de los usuarios	4

Tabla 3.8: Puntos de historia a desarrollar en el sprint 7

Esta se trata de la iteración con un mayor esfuerzo en todo el desarrollo del proyecto al tratar de desarrollar una búsqueda eficiente con multitud de características y toda la flexibilidad posible. Siempre es posible tratar de optimizar un poco más la búsqueda buscando resultados más precisos, reducir las lecturas en la base de datos a las mínimas posibles para reducir el coste o reducir tiempos de carga en los resultados. Por tanto el límite viene dado por el propio tiempo asignado para esta iteración

Durante esta iteración se estuvieron probando diferentes alternativas en cuanto a los mecanismos de búsqueda, descartando mecanismos ya implementados si se encontraba alguno que diese mejores resultados. Se comenzó realizando las interfaces para las pantallas de búsqueda, resultados, categorías y filtros, que era lo más lineal y posteriormente se pasó a la implementación de las funcionalidades. Esto

es apreciable en el diagrama Burndown mostrado en la ilustración 3.19 dónde se puede observar cómo los primeros días el desarrollo avanza con normalidad hasta que se comienzan a desarrollar las funcionalidades y hay un estancamiento hasta que se finalmente se dieron por finalizadas.

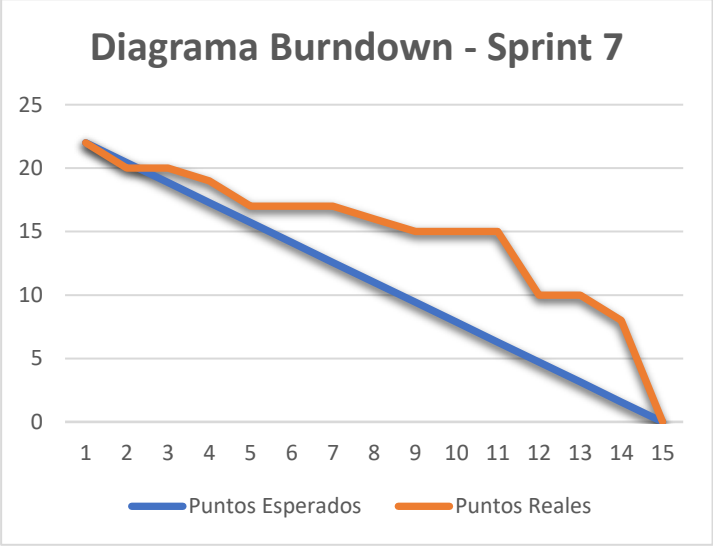

Ilustración 3.19: Diagrama Burndown para el sprint 7

Cabe a destacar que a pesar de ser la iteración con más puntos de historia asociados tan solo se disponía de unas dos semanas para llevarla a cabo, por lo que hizo falta que en los días de desarrollo se usarán más horas de las previstas para poder cumplir con los plazos establecidos.

El funcionamiento de los mecanismos implementados pasará a explicarse en profundidad en el apartado de funcionalidades que se verá más adelante, pero podemos observar en la ilustración 3.20 algunas capturas de los resultados de interfaces y funcionalidades tras este sprint.

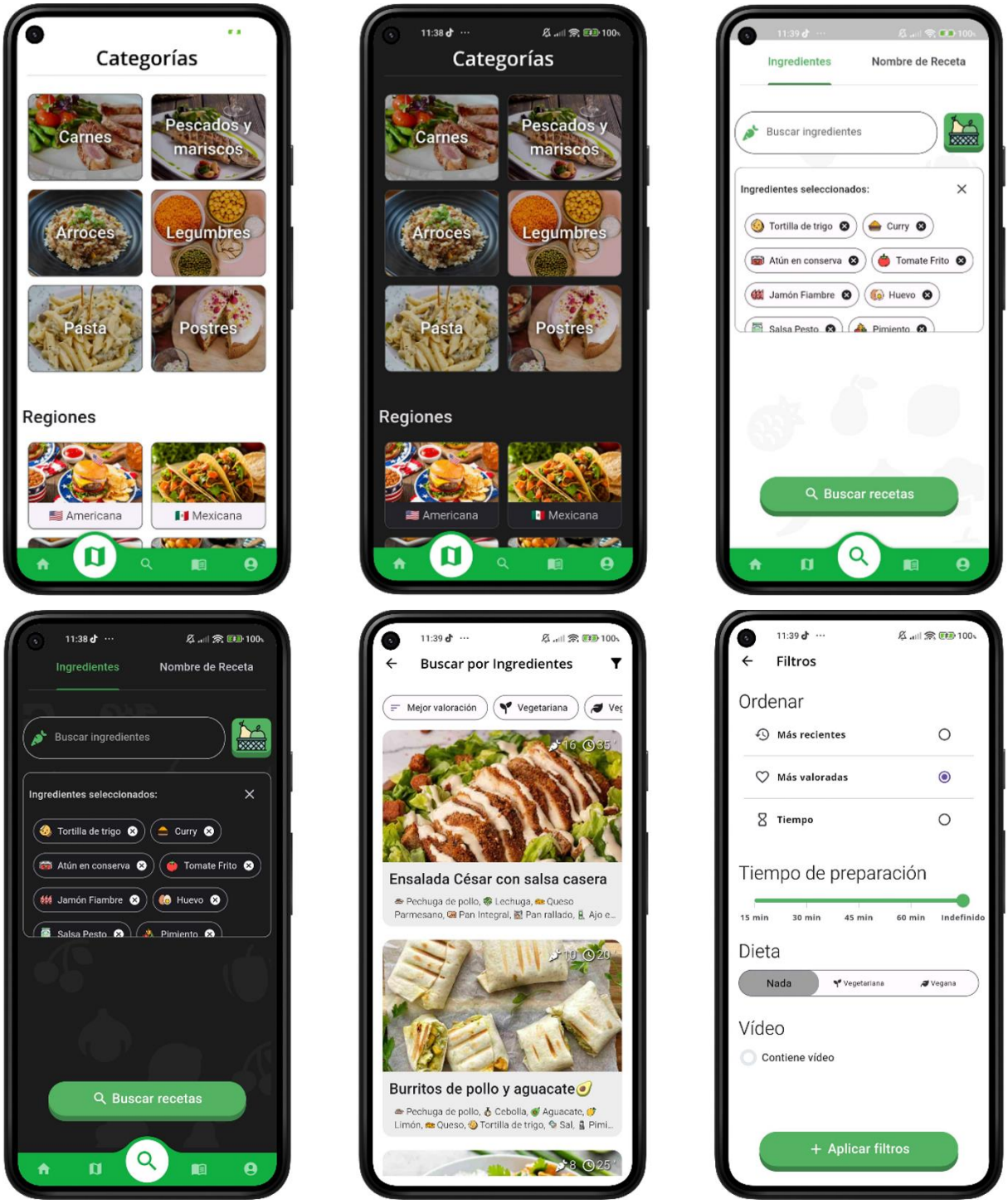

Ilustración 3.20: Capturas de búsqueda, categorías y filtros tras el sprint 7

3.1.8 Sprint 8: Gestión de listas de recetas y sistemas de valoración

En esta iteración se tratan de cumplir con los últimos requisitos requeridos para la aplicación, los cuales serían el sistema de creación y gestión de listas de recetas por parte de los usuarios y la posibilidad de dar una valoración a las recetas. En la

tabla 3.9 podemos observar cómo serían las últimas actividades por realizar antes de la revisión de errores, sumando un total de 9 puntos de historia (5 + 4).

Historia	Puntos de historia
Diseño de la interfaz, planificación y configuración inicial	6
Gestión de despensa y cesta	6
Desarrollar lista de ingredientes	4
Funcionalidad para escanear productos mediante el código de barras y consulta a la API	5
Pantalla de usuario	3
Menús y mejora de la interfaz	2
Sistema de control de la caducidad en alimentos	3
Notificaciones programables de caducidad	8
Implementación de Firebase al proyecto	5
Creación de cuentas en Firebase	5
Gestión de cuentas de usuarios	6
Implementación y visualización de recetas	4
Creación de recetas	5
Subida de recetas	3
Gestión de distintos roles de usuario y moderación	2
Búsqueda por ingredientes	6
Búsqueda por nombre	6
Búsqueda por categoría	4
Aplicar filtros de búsqueda	6
→ Sistema de valoración basado en "me gustas" de los usuarios	5
→ Creación y gestión de listas de recetas personales de los usuarios	4

Tabla 3.9: Puntos de historia a desarrollar en el sprint 8

El desarrollo de estas funcionalidades transcurrió según los tiempos previstos. Para este momento el proyecto ya ofrece la base para realizar todas estas funcionalidades, es decir, ya hay un sistema de recetas que se suben a un servidor con sus respectivos botones para dar “me gusta” o guardar en listas de recetas aunque sin funcionalidad implementada. En la pantalla de usuario también había una parte que estaba reservada para mostrar las listas de recetas que le habían gustado al usuario o listas de recetas personalizadas.

Por tanto, gracias al avance realizado en sprints anteriores, la actividad se limitó a implementar la funcionalidad restante, actualizando en el servidor las recetas con

los “me gusta” dados, asegurando que estas operaciones sean atómicas entre los múltiples usuarios, guardar las recetas valoradas en una lista asociada a la cuenta del usuario y realizar un funcionamiento similar para la creación de listas de recetas. Las historias de usuario se realizando algo más rápido que los tiempos esperados, como se pueden observar en el diagrama Burndown en la ilustración 3.21.

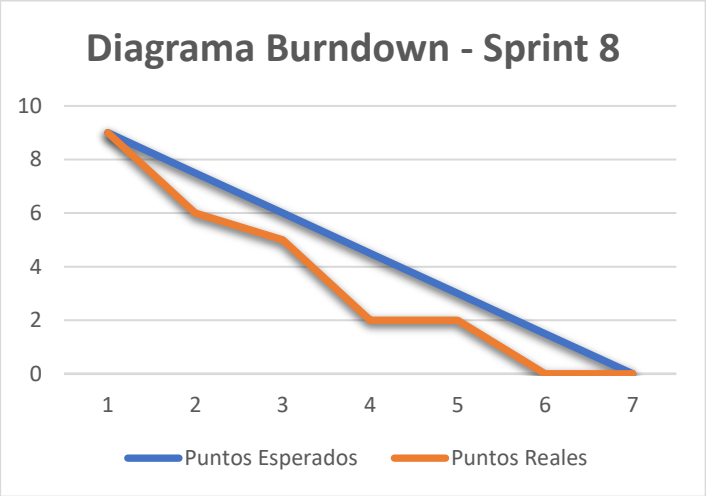

Ilustración 3.21: Diagrama Burndown para el sprint 8

Cabe destacar que para el caso de las listas de recetas se trató de ampliar un poco más la funcionalidad principal desarrollada. En un principio se tenía planteado que un usuario pudiera crear todas las listas de recetas que quisiera y guardar una receta en todas las listas que requiriera, lo cual se ha cumplido, pero al querer guardar estos cambios en el servidor para que el usuario descargara sus listas de recetas al iniciar sesión en otro dispositivo, se decidió incluir algunos cambios.

Las listas de recetas en lugar de guardarse en el propio documento del usuario en el servidor, pasaron a ser guardadas como una colección independiente en la base de datos y guardar en el documento del usuario tan solo el identificador de la lista. Además estas listas contarían con campos como un contador de “me gusta”, estado booleano para situarlas como privadas o públicas y una descripción. Al incluir estas características adicionales en un futuro con poco esfuerzo se podría añadir un sistema dónde los usuarios hicieran públicas sus listas de recetas y estas fueran accesibles por otros usuarios que pudieran verlas, valorarlas y guardarlas.

Como resultado de la iteración podemos ver a continuación, en la ilustración 3.22, capturas de la pantalla con listas de recetas creadas y su pantalla de creación y guardado.

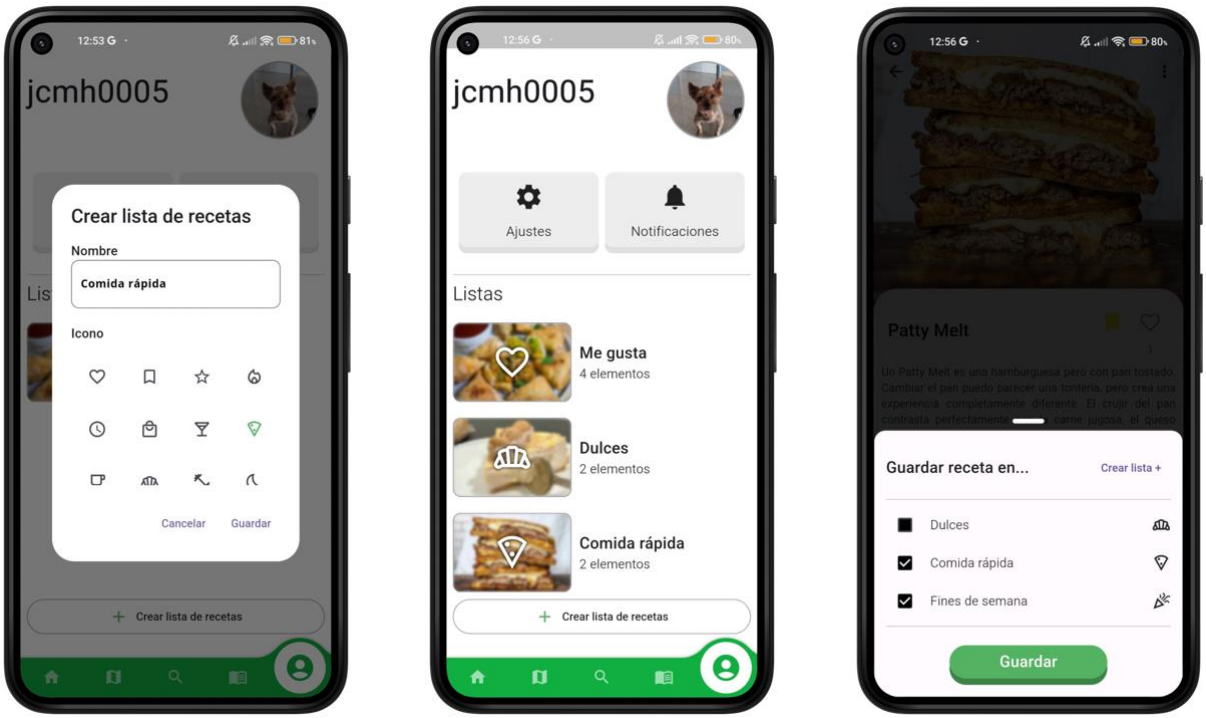

Ilustración 3.22: Capturas de creación y vista de listas de recetas tras el sprint 8

3.1.9 Sprint 9: Optimización

La última iteración del proyecto se ha reservado para realizar procesos de optimización y solución de errores. Era esperado que tras desarrollar tantas funcionalidades y algunas en tiempos relativamente reducidos pudiesen quedar algunos errores sin tratar o procesos con un margen de mejora esperable. Esta iteración tan solo cuenta con 5 puntos de historia ya que cuando se planeó no se esperaba que fuese a haber una gran cantidad de aspectos por corregir.

Historia	Puntos de historia
Diseño de la interfaz, planificación y configuración inicial	6
Gestión de despensa y cesta	6
Desarrollar lista de ingredientes	4
Funcionalidad para escanear productos mediante el código de barras y consulta a la API	5

Pantalla de usuario	3
Menús y mejora de la interfaz	2
Sistema de control de la caducidad en alimentos	3
Notificaciones programables de caducidad	8
Implementación de Firebase al proyecto	5
Creación de cuentas en Firebase	5
Gestión de cuentas de usuarios	6
Implementación y visualización de recetas	4
Creación de recetas	5
Subida de recetas	3
Gestión de distintos roles de usuario y moderación	2
Búsqueda por ingredientes	6
Búsqueda por nombre	6
Búsqueda por categoría	4
Aplicar filtros de búsqueda	6
Sistema de valoración basado en "me gustas" de los usuarios	5
Creación y gestión de listas de recetas personales de los usuarios	4
→ Optimización	2
→ Arreglo de errores	3

Tabla 3.10: Puntos de historia a desarrollar en el sprint 9

Los procesos de optimización y arreglo de errores se han realizado algo por debajo del tiempo esperado como se puede observar en el diagrama Burndown de la ilustración 3.23.

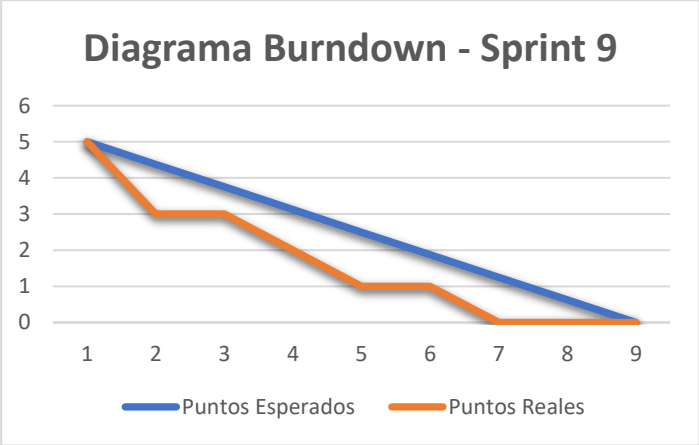

Ilustración 3.23: Diagrama Burndown para el sprint 9

En el transcurso de este sprint, se llevaron a cabo diversas mejoras y correcciones en la interfaz de usuario. Se abordaron problemas relacionados con la

actualización de elementos en pantalla tras la aplicar cambios, así como inconsistencias en la visualización al alternar entre los modos claro y oscuro. Adicionalmente, se implementaron restricciones, como límites de texto en los campos que eran introducidos por el usuario, para prevenir desbordamientos en la pantalla.

Un aspecto destacado del proceso de optimización fue en el sistema de adición de productos a listas mediante el escáner de códigos de barras. Anteriormente, al añadir un producto por este método, se extraía el nombre del producto de la API y se le asignaba la categoría de alimento cuya denominación fuera más similar. Sin embargo, esta aproximación resultaba ineficiente debido a la imprecisión de los nombres proporcionados por la API, lo que podía resultar en categorías erróneas.

Para solventar esto, ahora al escanear un producto, se despliega una ventana de selección de categorías. Esta interfaz presenta tres sugerencias que la aplicación considera potencialmente adecuadas para que el usuario elija. En caso de que ninguna de estas opciones sea apropiada, se habilita un campo de búsqueda para que el usuario pueda localizar manualmente la categoría correcta. Esta mejora asegura que todos los ingredientes se almacenen con su categoría correspondiente, aumentando así la precisión y utilidad de la base de datos de productos.

3.2 Funcionalidades desarrolladas

En este apartado se explicarán en profundidad las distintas funcionalidades de la aplicación y todos los aspectos relevantes del desarrollo del proyecto. Para mayor comprensión de la implementación se usarán imágenes, esquemas y pseudocódigos siempre que sea necesario. Este apartado está estructurado entre las distintas secciones y funciones principales de la aplicación, dónde en cada una se explicará tanto su funcionamiento en particular cómo su integración con las demás secciones de la aplicación.

Cabe destacar que uno de los objetivos del proyecto era desarrollar una interfaz cuidada que fuese lo más minimalista e intuitiva para los usuarios, por lo que también se incluirán explicaciones en torno a la estructura de la interfaz y su funcionamiento siempre que se considere necesario.

3.2.1 Gestión de listas de productos

La funcionalidad de gestión de la despensa y la lista de la compra en la aplicación se desarrolla a través de diversas operaciones que permiten a los usuarios añadir, editar, eliminar y mover productos entre ambas listas. Para ello **utiliza el patrón Provider para la gestión del estado**, concretamente *ChangeNotifierProvider*, que facilita la notificación de cambios en el estado a los widgets suscritos en él. Primero vamos a pasar a ver en que consiste este patrón.

El patrón Provider es una implementación del patrón de diseño de software Observador. El patrón Observador es un patrón de diseño en el que un objeto, conocido como el sujeto, mantiene una lista de sus dependencias, llamadas observadores, y les notifica automáticamente cualquier cambio de estado, usualmente llamando a uno de sus métodos. Este patrón es útil para implementar sistemas en los que se desea que un cambio en un objeto se refleje automáticamente en otros objetos sin que estos estén estrechamente acoplados. En el contexto de Flutter, Provider actúa como el sujeto que gestiona el estado y notifica a los widgets observadores.

Provider [7] es una solución ampliamente utilizada en Flutter para la gestión del estado. Proporciona una manera eficiente y fácil de **manejar y compartir el estado**

entre diferentes widgets en la aplicación, en este caso, para sincronizar las operaciones que hagamos sobre cada una de la lista de productos. *ChangeNotifierProvider* [8] es una implementación específica de Provider que utiliza la clase *ChangeNotifier* para notificar a los widgets suscritos sobre los cambios en el estado.

Con *ChangeNotifier* cuando se produce un cambio en el estado gestionado por ChangeNotifier, se llama a la función *notifyListeners(),* lo que actualiza automáticamente todos los widgets suscritos. Este patrón es especialmente útil para aplicaciones que requieren actualizaciones frecuentes de la interfaz de usuario basadas en cambios de estado.

Provider también se encarga de asegurar que los widgets hijos se reconstruyan correctamente cuando el estado cambia, evitando así inconsistencias visuales y errores en la interfaz de usuario. En el caso de nuestra aplicación, cada vez que un usuario añade, edita o elimina un ingrediente en la despensa o en la lista de la compra, *ChangeNotifierProvider* se asegura de que todos los widgets que dependen de esa información se actualicen automáticamente para reflejar el nuevo estado. En este caso, hay dos proveedores principales:

- **PantryListProvider**: Gestiona la lista de productos en la despensa. Inicializa la lista de ingredientes desde la base de datos y proporciona métodos para añadir, actualizar, eliminar y mover ingredientes.
- **BasketListProvider**: Gestiona la lista de productos en la cesta de la compra, de manera similar a PantryListProvider.

Estos proveedores además de gestionar las operaciones independientes para cada lista también funcionan conjuntamente para mover productos entre ambas listas. Cuando un usuario decide mover un ingrediente de la despensa a la lista de la compra, *PantryListProvider* elimina el ingrediente de la despensa y *BasketListProvider* lo añade a la lista de la compra, notificando los cambios a cada lista.

En el esquema de la ilustración 3.24 se observa como interactúan estos proveedores en la aplicación. La sección *Home* contendría las pantallas específicas para la despensa (*PantryScreen)* y cesta (*BasketScreen)*. Cada una de estas pantallas se comunicaría con su proveedor específico para actualizar cada lista. Por su parte cada proveedor interactuaría directamente con las clases referentes a la base de datos, que pasarán a comentarse a continuación.

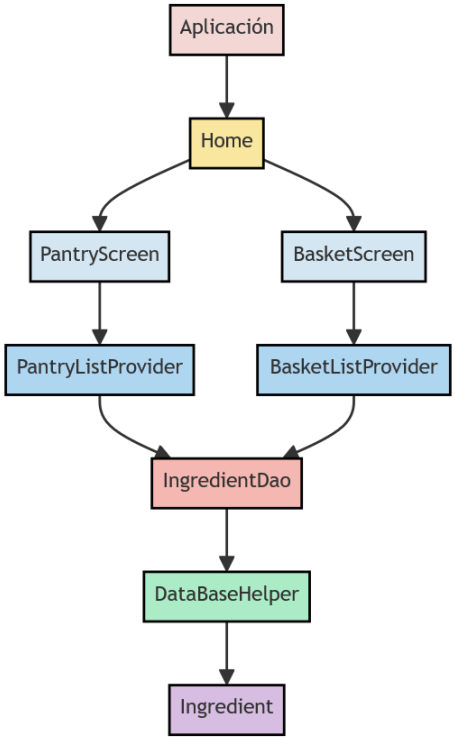

Ilustración 3.24: Esquema de funcionamiento para la interacción con listas

Los datos se guardan una base de datos SQLite, la cual se inicializa y gestiona mediante la clase *DateBaseHelper* visible en el esquema anterior. Durante la inicialización, la primera vez que se abre la aplicación, se crean las tablas necesarias para cada lista de ingredientes.

Por su parte, la clase *IngredientDao* actúa como el **intermediario entre la aplicación y la base de datos**. Este patrón de diseño, conocido como DAO (Data Access Object), es crucial para mantener una separación clara entre la lógica de negocio y el acceso a los datos. Este maneja las operaciones CRUD (crear, leer, actualizar, eliminar) para los ingredientes en la base de datos. Proporciona métodos para leer todos los ingredientes, insertar nuevos, actualizar existentes y eliminarlos.

Respecto a la interfaz construida para esta sección, podemos ver un resumen de su estructura en la ilustración 3.25 dónde se muestra la parte principal del árbol de widgets y mediante la ilustración 3.36 podemos ver los límites de diseño de cada una de las partes de la pantalla.

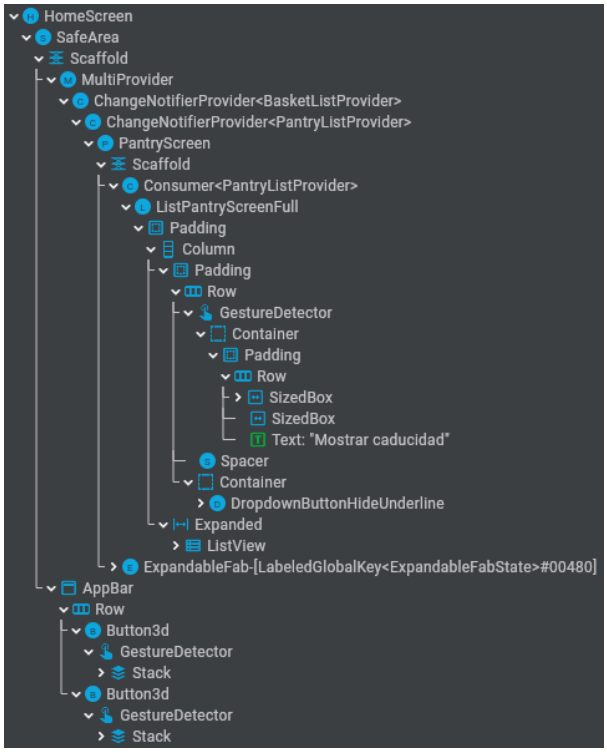

Ilustración 3.25: Árbol de Widgets de la sección Home

Ilustración 3.26: Límites de diseño de la sección Home

Ya que debido a la gran cantidad de elementos por los que está compuesta la interfaz no sería posible mencionarlos todos ni hacer un esquema completo que los etiquetase, vamos a pasar a nombrar los elementos más importantes. La pantalla *HomeScreen* es el punto central desde donde el usuario puede alternar entre la vista de la despensa y la vista de la cesta de la compra. Esta utiliza una barra de aplicación (*AppBar*) personalizada que contiene dos botones grandes, cada uno representando una de las vistas. Estos botones están diseñados usando un widget personalizado llamado *Button3d* [9], que proporciona un efecto tridimensional y cambia de color dependiendo de la vista seleccionada.

El contenido de *HomeScreen* son las pantallas respectivas a la despensa *PantryScreen* y a la cesta *BasketScreen*, las cuáles se van alternando en función de

los botones que pulse el usuario. Cada una de estas pantallas tiene un botón inferior desplegable [10] con las opciones para añadir el producto manualmente o mediante el escáner de códigos de barras como se puede ver en la ilustración 3.27.

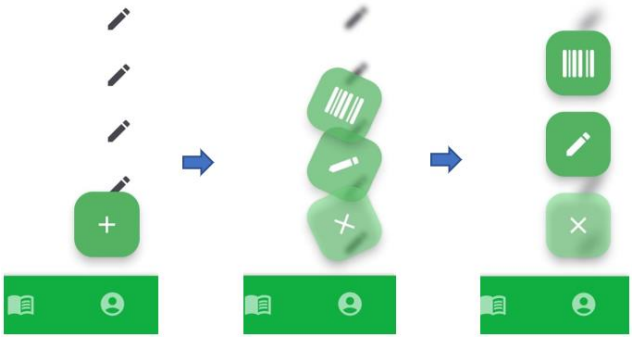

Ilustración 3.27: Botón flotante para añadir productos

Respecto a la funcionalidad de añadir productos mediante códigos de barras se tratará en apartado 3.2.3 (Escaneo y obtención de información de productos). Por otra parte cuando se selecciona añadirlo de forma manual, se muestra una ventana *ModalBottonSheet* [11] que presenta la pantalla de adición de ingredientes, la cual se puede ver en la Ilustración 3.28. No se profundizará en su estructura ya que está compuesta por un formulario simple dónde el usuario puede introducir los campos que desee para añadir el producto.

Ilustración 3.28: Agregar producto manualmente

Una vez que los productos son añadidos a la despensa o cesta el usuario puede realizar las acciones de eliminar los productos, moverlos entre ambas listas, editarlos, mostrar o no su fecha de caducidad, ordenarlos y en el caso de la lista de la compra también puede compartirlos. Respecto a esta funcionalidad la lista de la compra se convierte a una lista en texto plano y mediante el paquete *share_plus* [12] se puede compartir con otras aplicaciones como se ve en la ilustración 3.29.

Ilustración 3.29: Compartir lista con otras aplicaciones

Para mover productos entre ambas listas, la funcionalidad se ha diseñado como tal como se planteó en el diseño. Cada producto de la lista se ha implementado como un deslizable, de forma que si está en la cesta y se desliza hacia la despensa se moverá a esa lista e igualmente si está en la despensa y se desliza hacia la cesta cambiará a esta lista. Si el producto se desliza hacia el lugar opuesto a dónde esté la otra lista se eliminará. El usuario obtendrá retroalimentación de la acción que está realizando ya que mientras desliza cada producto podrá observar con un icono y nombre hacia dónde está moviendo el producto

En la ilustración 3.30 observamos 3 capturas de pantalla dónde se aprecia esta funcionalidad. En la primera, desde la despensa, se está deslizando un producto hacia la cesta situada a la derecha. En la segunda, desde la cesta, se está deslizando un producto hacia la despensa situada a la izquierda. La tercera, desde la despensa, trata

de deslizar un producto hacia la izquierda, dónde no hay ninguna lista, por lo que eliminaría el producto.

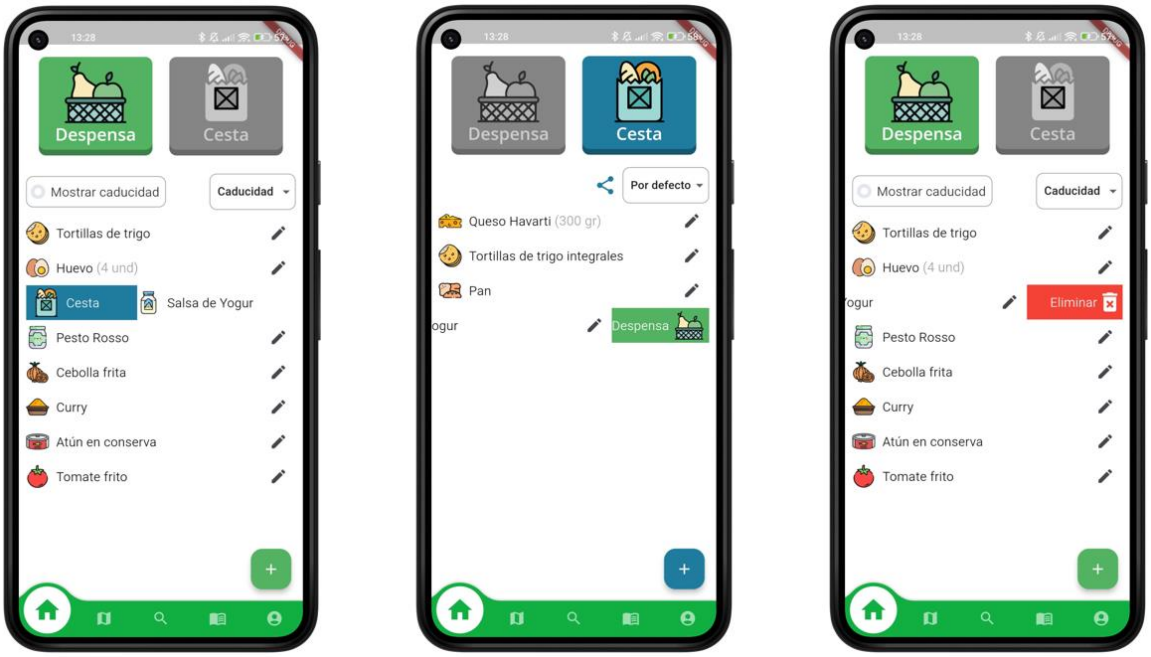

Ilustración 3.30: Mover y descartar productos

3.2.1.1 Pruebas

A continuación se mostrarán los resultados de los test realizados para la gestión de listas de productos.

T-1 Test de añadir productos	
Objetivo	Los menús deben permitir añadir productos con diferentes características sin ningún tipo de problema
Resultado	Los usuarios pueden agregar productos a las diferentes listas sin problemas especificando el nivel de detalle en cada producto que deseen
Observaciones	Funciona de acuerdo a lo esperado

Tabla 3.11: Test de añadir productos

T-2 Test de deslizar elementos	
Objetivo	Se deben permitir deslizar productos en las listas para moverlos de lista o eliminarlos
Resultado	Las listas permiten deslizar los productos correctamente para realizar las acciones esperadas, ofreciendo además retroalimentación visual de la acción que se está haciendo

Observaciones	El usuario puede llegar a tener en cuenta en su primera interacción la posibilidad de deslizar los productos. Podría ser interesante incluir una pantalla de tutorial la primera vez que se acceda a esta sección

Tabla 3.12: Test de deslizar elementos

T-3 Test de retroalimentación en listas	
Objetivo	La interfaz debe proporcionar retroalimentación que haga saber al usuario en que lista de productos se encuentra
Resultado	Se asigna un color a cada sección, verde a la despensa y azul a la cesta, de forma que el botón de añadir productos y el menú de adición se encuentran del color correspondiente. Además en la parte superior, aparece destacado con color el botón de la lista seleccionada y con un tono de gris apagado la lista que no está seleccionada
Observaciones	Ningún usuario que probara la pantalla llegó a tener dudas sobre en qué pantalla se encontraba ni llegó a confundirse con la lista con la que estaba interactuando

Tabla 3.13: Test de retroalimentación en listas

T-4 Test de visibilidad de la información	
Objetivo	La información de los productos debe ser completamente visible y flexible ante cualquier situacion
Resultado	El sistema no permite que se desborde ninguna parte de la pantalla ni se queden elementos ocultos
Observaciones	Se añadió un límite de caracteres para el nombre de los productos. También para que el botón de añadir productos no tapase información relevante, se colocó al final de cada lista un elemento invisible de la altura del botón para que siempre pueda llegar a ser visible el último elemento de cada lista

Tabla 3.14: Test de visibilidad de la información

3.2.2 Lista de ingredientes

La aplicación requiere una base de datos de alimentos clasificados para implementar todas las funcionalidades previstas. Este requisito se deriva principalmente de la función de búsqueda de recetas por ingredientes. Cada receta contendrá un listado de ingredientes, y cada ingrediente estará asociado a un identificador único. Este sistema garantiza que el mismo ingrediente en diferentes recetas mantenga el mismo identificador.

Cuando un usuario desee buscar recetas con ingredientes específicos, la aplicación deberá localizar las recetas que contengan los identificadores correspondientes a los ingredientes seleccionados por el usuario. Para facilitar este proceso, es necesaria una base de datos de ingredientes donde cada elemento tenga un identificador único para la búsqueda y un nombre para la visualización del usuario. Adicionalmente, cada ingrediente tendrá asociado un icono para mejorar su identificación visual y contribuir a una interfaz más atractiva, así como la unidad de medida correspondiente.

Además, la funcionalidad que permite al usuario buscar recetas basadas en los productos disponibles en su despensa también requiere que todos los alimentos gestionados en esta lista tengan su identificador correspondiente, basado en la lista de ingredientes. Para cumplir con esto, cuando un usuario añada un producto a su despensa o cesta, se le solicitará que seleccione el ingrediente de la lista predefinida. Durante la introducción manual, se mostrarán sugerencias que coincidan con el nombre que el usuario está escribiendo, permitiéndole seleccionar el ingrediente deseado. Este enfoque asegura la correcta etiquetación de todos los productos en las listas y agiliza el proceso de introducción de alimentos.

En cuanto a la adición de productos mediante el escaneo de códigos de barras, el proceso será similar. Se buscarán en la base de datos los ingredientes cuyo nombre sea más similar al del producto escaneado. Para evitar errores, el usuario deberá confirmar la selección del producto escaneado antes de guardarlo. Este proceso se detallará más exhaustivamente en la sección 3.2.3 (Escaneo y obtención de información de productos).

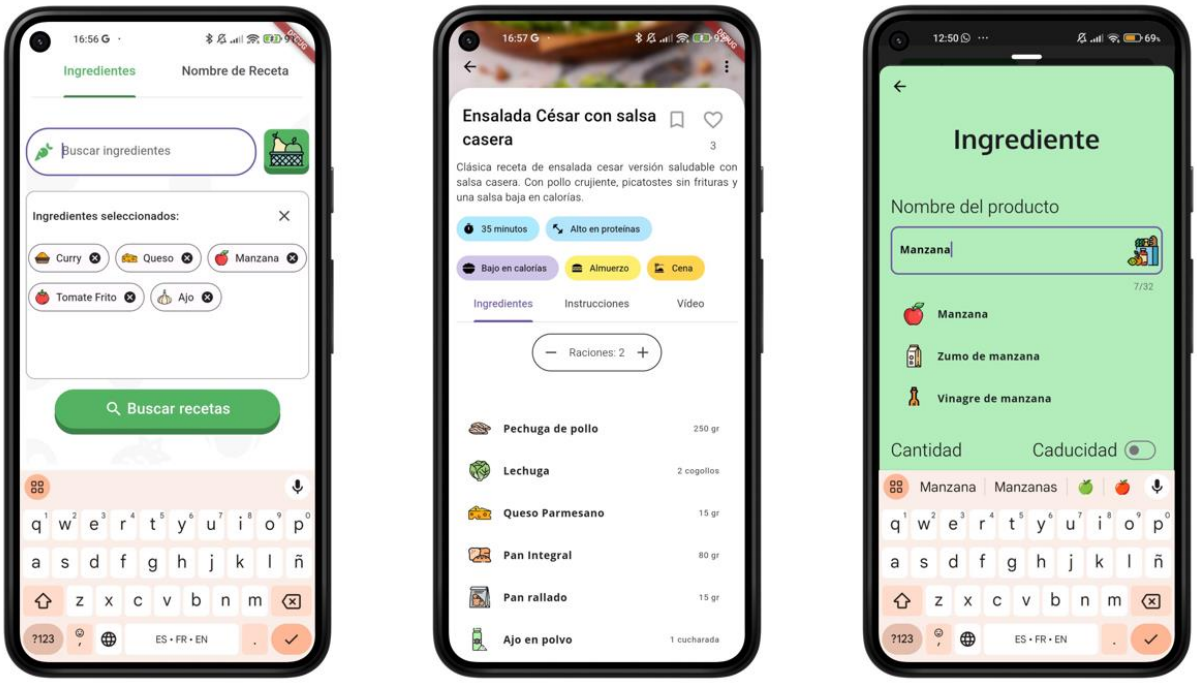

Ilustración 3.31: Ejemplos de uso de la lista de ingredientes en la aplicación

La implementación de una lista de ingredientes conlleva ventajas adicionales ya que al tener cada ingrediente un icono asociado, se mejora la coherencia visual de la interfaz, ya que los mismos iconos aparecerán en todas las secciones donde se mencionen ingredientes (despensa, cesta, búsqueda de recetas, visualización de recetas, etc.). Esto proporciona una experiencia más unificada y facilita la identificación rápida de ingredientes mediante sus iconos.

Otra ventaja se refleja en la adición manual de ingredientes. Al seleccionar un ingrediente de las sugerencias, no solo se establecerá automáticamente su icono, sino también la unidad de medida registrada para ese tipo de producto en la base de datos. Estas funcionalidades son extensibles, permitiendo la incorporación de características adicionales a los ingredientes en la base de datos, lo que podría enriquecer la experiencia del usuario mediante nuevas funcionalidades.

La base de datos de ingredientes desarrollada cuenta con aproximadamente 1000 ingredientes, dónde se cubren la mayoría de los productos relacionados con alimentación y sus variantes. Esta ha sido realizada manualmente, ayudándose con de otras bases de datos de productos de alimentación extraídas de supermercados.

La base de datos se ha realizado de forma que pueda resultar extensible en cualquier momento para ser aplicada mediante una actualización a la aplicación. Los ingredientes y sus elementos asociados se almacenan en un archivo JSON con un formato similar al siguiente:

```
                                                ...
442. {"id": "443", "name": "Manzana", "icon": "manzana", "unit": "kg", "head": 0},
443. {"id": "444", "name": "Maracuyá", "icon": "maracuya", "unit": "kg", "head": 0},
444. {"id": "445", "name": "Margarina", "icon": "margarina", "unit": "gr", "head": 0},
445. {"id": "446", "name": "Mayonesa", "icon": "mayonesa", "unit": "und", "head": 0},
446. {"id": "447", "name": "Maíz", "icon": "maiz", "unit": "und", "head": 0},
447. {"id": "448", "name": "Mejillones", "icon": "mejillon", "unit": "kg", "head": 0},
448. {"id": "449", "name": "Melocotón", "icon": "melocoton", "unit": "und", "head": 0},
449. {"id": "450", "name": "Melva", "icon": "melva", "unit": "kg", "head": 0},
450. {"id": "451", "name": "Melón", "icon": "melon", "unit": "und", "head": 0},
451. {"id": "452", "name": "Membrillo", "icon": "membrillo", "unit": "kg", "head": 0},
452. {"id": "453", "name": "Merluza", "icon": "merluza", "unit": "kg", "head": 0},
453. {"id": "454", "name": "Mermelada", "icon": "mermelada", "unit": "gr", "head": 1},
                                                ...
```

Fragmento del archivo JSON dónde se listan los ingredientes

Cuando se inicia la aplicación por primera vez, se carga la lista de ingredientes del archivo en la base de datos SQLite que contiene el sistema, de forma que sea fácilmente accesible en cualquier momento. La operación de cargar los ingredientes en la base tan solo se realizaría una vez, sin embargo, se ha implementado un sistema de control de versiones dónde si se modifica algo en el archivo se actualizaría la versión de la aplicación y cuando la aplicación detectase que hay una nueva versión, actualizaría la base de datos con los nuevos ingredientes que hubiera.

Respecto a los iconos utilizados para los ingredientes, ya que hacía falta utilizar cientos de iconos para cubrir todos los ingredientes, se han buscado que fuesen imágenes vectoriales o SVG las cuáles ocupan mucho menos espacio y pueden escalarse y adaptarse a cualquier tamaño de forma perfecta, además de que flutter cuenta con el paquete *flutter_svg* [13] para mostrarlos de forma nativa.

Ilustración 3.32: Fragmento de los iconos utilizados para los ingredientes

Los iconos debían de tener un estilo similar, que en este caso se buscaba que fuese “Flat Design” o diseño plano, caracterizados por usar colores sólidos, formas simples y minimalistas sin efectos tridimensionales ni degradados. Han sido extraídos de *Flaticon* [14], la cual es una base de datos de iconos gratuita que ofrece una gran cantidad de conos que pueden usarse de forma personal o comercial con atribución.

Otro aspecto a mencionar es la forma para buscar dentro de esta lista de ingredientes. Inicialmente, se implementó un sistema de búsqueda convencional que comparaba el texto introducido por el usuario con el nombre de cada ingrediente, ofreciendo como resultado las coincidencias exactas. Sin embargo, este enfoque demostró no ser óptimo, ya que variaciones menores como tildes o cambios en la estructura del nombre de cada producto podían resultar en no obtener los resultados deseados.

Para abordar esta limitación, se optó por implementar un sistema de búsqueda difusa, también conocido como "Fuzzy Search" . La búsqueda difusa [15] es una técnica avanzada que utiliza algoritmos específicos para identificar cadenas de texto que coinciden de manera aproximada con un patrón dado. Este método es particularmente útil cuando se buscan coincidencias que pueden contener errores tipográficos menores, variaciones ortográficas o diferencias en la estructura del texto.

El algoritmo de búsqueda difusa evalúa la similitud entre el término de búsqueda y los elementos en la base de datos, permitiendo un cierto grado de flexibilidad en las coincidencias. Esto significa que puede encontrar resultados relevantes incluso cuando la entrada del usuario no coincide exactamente con los términos almacenados, lo que mejora significativamente la eficacia y la experiencia del usuario en el proceso de búsqueda.

Ilustración 3.33: Uso de búsqueda difusa para encontrar ingredientes

Esta funcionalidad de búsqueda difusa se ha implementado en todas las secciones de la aplicación donde se requiere la búsqueda de ingredientes basada en la entrada de texto por parte del usuario. La efectividad de esta implementación se puede observar en la ilustración 3.33, que muestra ejemplos concretos de los resultados obtenidos mediante este método de búsqueda. Para su correcta implementación se ha usado el paquete *fuzzy* [16] de flutter.

3.2.2.1 Pruebas

A continuación se mostrarán los resultados de los test realizados para la lista de ingredientes.

T-1 Test de magnitud de la lista	
Objetivo	La lista de ingredientes debe ser lo suficiente completa para que abarque la mayoría de categorías de ingredientes posible
Resultado	La lista es significativamente grande, cubriendo la mayoría de categorías de alimentos que podemos encontrar y muchas de las subcategorías de los mismos

Observaciones	Se ha elaborado esta lista en torno a bases de datos de supermercados españoles, por lo que puede haber alimentos localizados de otras nacionalidades que no se encuentren disponibles

Tabla 3.15: Test de magnitud de la lista

T-2 Test de iconos de los ingredientes	
Objetivo	Cada ingrediente debe tener asociado su propio icono, creando una identidad visual
Resultado	Todos los ingredientes tienen asociado su icono de un estilo similar de forma que facilite su identificación
Observaciones	Se ha cumplido según lo esperado

Tabla 3.16: Test de iconos de los ingredientes

T-3 Test de escalabilidad de la lista	
Objetivo	La lista debe ser fácilmente extensible
Resultado	La lista permite añadir nuevos productos sin mayor esfuerzo
Observaciones	Se ha establecido un control de versiones para que los usuarios puedan actualizar con la respectiva actualización de la aplicación de forma automática

Tabla 3.17: Test de escalabilidad la lista

T-4 Test de búsqueda de ingredientes	
Objetivo	Se deben poder buscar ingredientes de forma fácil
Resultado	Los ingredientes se encuentran rápidamente sin ninguna complicación
Observaciones	Se ha usado una búsqueda difusa para mejorar este proceso

Tabla 3.18: Test de búsqueda de alimentos

3.2.3 Escaneo y obtención de información de productos

La funcionalidad de escaneo y obtención de información de productos en la aplicación es una característica clave que permite a los usuarios **añadir productos a**

su despensa o cesta de manera eficiente a la misma vez que les permite **consultar información nutricional** sobre los productos escaneados.

La pantalla *BarcodeScreen* es la encargada de gestionar el escaneo de productos. Esta utiliza *AiBarcodeScanner* [17], un widget que forma parte de un paquete que facilita la integración de la funcionalidad de escaneo de códigos de barras en aplicaciones Flutter. *AiBarcodeScanner* proporciona una interfaz de usuario para la cámara del dispositivo, permitiendo a los usuarios escanear códigos de barras en tiempo real.

Respecto a la obtención de la información de los productos basado en su código de barras, como se ha comentado en apartados anteriores se va a utilizar *Open Food Facts* [18] que es la mayor base de datos colaborativa, abierta y gratuita que ofrece información estructurada para infinidad de productos. Esta ofrece una API gratuita dónde para realizar peticiones para extraer todo tipo de características de los productos, incluyendo **nombre, fotografía, información de sus componentes y una valoración nutricional**, que son los valores que utilizaremos en la aplicación.

Para *BarcodeScreen* se han definido dos modos de funcionamiento distintos, diseñados para diferentes contextos de uso: el **modo dinámico** y el **modo no dinámico**. Este diseño permite una flexibilidad significativa en la forma en que los usuarios interactúan con la funcionalidad de escaneo de códigos de barras, dependiendo de si están añadiendo un producto nuevo o simplemente agregando un código de barras a un producto ya existente.

En el modo dinámico, que es el comportamiento principal que va a usar el usuario, la pantalla está configurada para escanear continuamente. Cuando detecta un nuevo código de barras, guarda el valor del código escaneado y muestra una ventana o modal con la información del producto. Este modal (nombrado *ProductModal*) solicita los detalles del producto a la API de Open Food Facts y, una vez obtenida la información, actualiza su contenido para mostrar detalles como el nombre del producto, la imagen y la puntuación nutricional. El usuario tiene la opción de guardar el producto en su despensa o cesta pulsando un botón. Guardar el

producto cierra tanto el modal como la pantalla de escaneo, asegurando que el producto se añada correctamente a la lista correspondiente.

Cómo se mencionó en el apartado anterior, cuando se guarda un producto en una lista es conveniente categorizarlo dentro de uno de los ingredientes existentes en la lista de ingredientes, por lo que al cerrar la pantalla de escaneo se abrirá una pantalla de sugerencias que mostrarán los productos de los que se cree que se trata el alimento escaneado basado en una búsqueda difusa a partir del nombre del producto. En caso de no corresponderse con ninguna de las sugerencias, hay una barra de búsqueda para que el usuario busque manualmente la categoría. En la ilustración 3.34 se puede observar más claramente el proceso de escaneo y guardado de productos al completo.

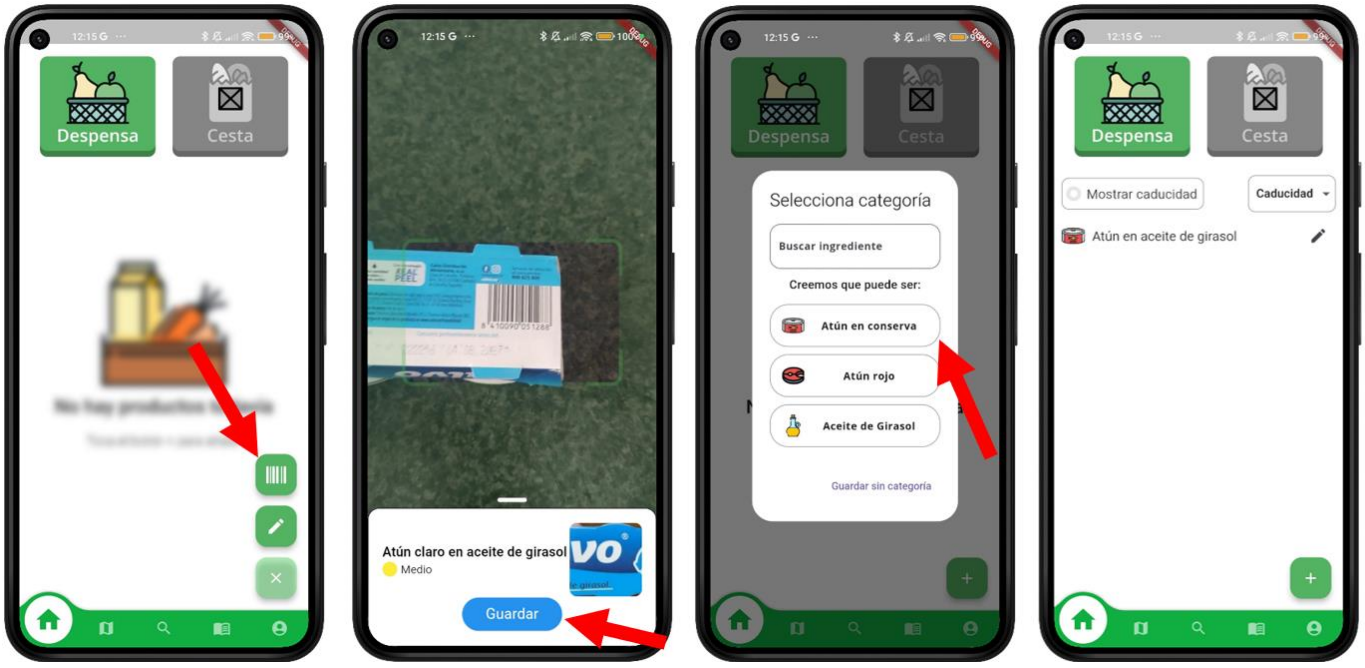

Ilustración 3.34: Proceso de escaneo y guardado de productos

Por otro lado, en el modo no dinámico, la funcionalidad es más directa. Cuando la BarcodeScreen detecta un código de barras, simplemente guarda el valor y cierra la pantalla, devolviendo el código escaneado. Este modo es útil cuando el usuario está en el proceso de añadir un producto manualmente y solo desea añadir el código de barras para obtener información nutricional. Al no requerir una confirmación adicional

o la visualización de información detallada del producto, este modo agiliza el proceso de entrada de datos.

A continuación se pasa a detallar el funcionamiento de la pantalla de escaneo haciendo uso de un diagrama de flujo mostrado en la ilustración 3.35 y de una explicación en detalle.

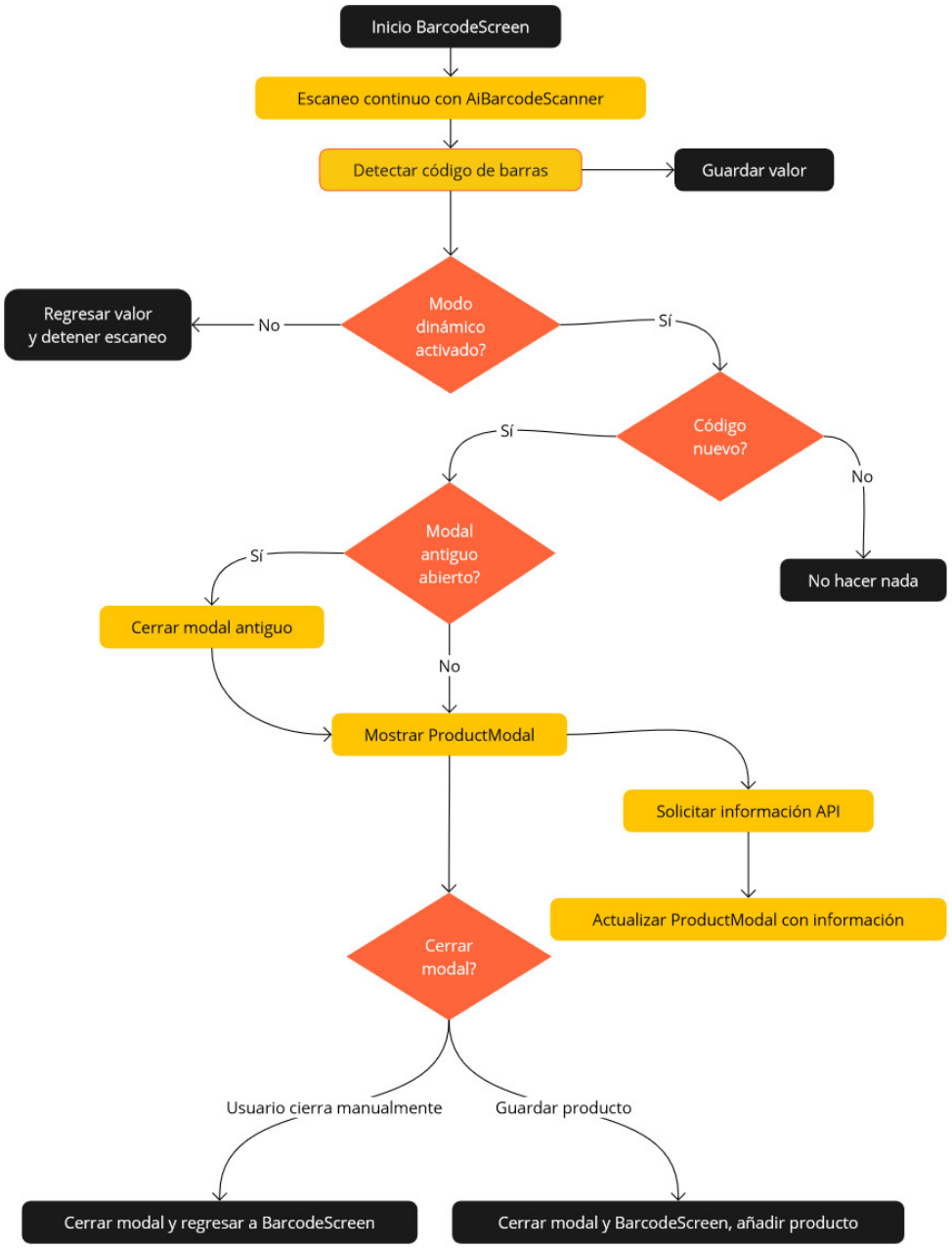

Ilustración 3.35: Diagrama de flujo de la pantalla de escaneo

BarcodeScreen escanea continuamente el entorno en busca de códigos de barras. Cuando se detecta un nuevo código, se guarda su valor. Si el modo dinámico está activado, se verifica si ya hay un modal abierto. Si es así, este se cierra para evitar superposiciones. Luego, se abre un nuevo modal con la información del producto escaneado. Este modal se encarga de solicitar los detalles del producto a la API y actualizar su contenido una vez que la información está disponible. El modal puede cerrarse manualmente por el usuario en cualquier momento, o automáticamente si se detecta un nuevo código de barras. Cuando el usuario decide guardar el producto, se cierra tanto el modal como la pantalla de escaneo, asegurando que el flujo de trabajo sea fluido y eficiente.

3.2.3.1 Obtención y evaluación de información nutricional

Ya se ha observado que en la pantalla de escaneo se muestra el nombre de cada producto, fotografía y una valoración nutricional que ofrece la API, sin embargo, también se trataba de permitir acceder a la información nutricional en detalle de cada producto escaneado.

Para realizar esto, cuando se escanea un producto se almacena su código de barras leído, de forma que cuando el usuario pulse cualquier producto que tenga en su lista de la compra o despensa, se use ese código de barras para solicitar la información nutricional de este a *Open Food Facts* y se abra una ventana con un resumen de esta.

En la ilustración 3.36 mostrada a continuación, se pueden observar algunos ejemplos de consulta de información de productos que han sido escaneados. En esta ventana se ofrece el nombre del producto, su marca, fotografía, una valoración basada en el sistema *NutriScore*, que es el que utiliza *Open Food Facts* para evaluar a los alimentos y un resumen de algunos de sus componentes principales.

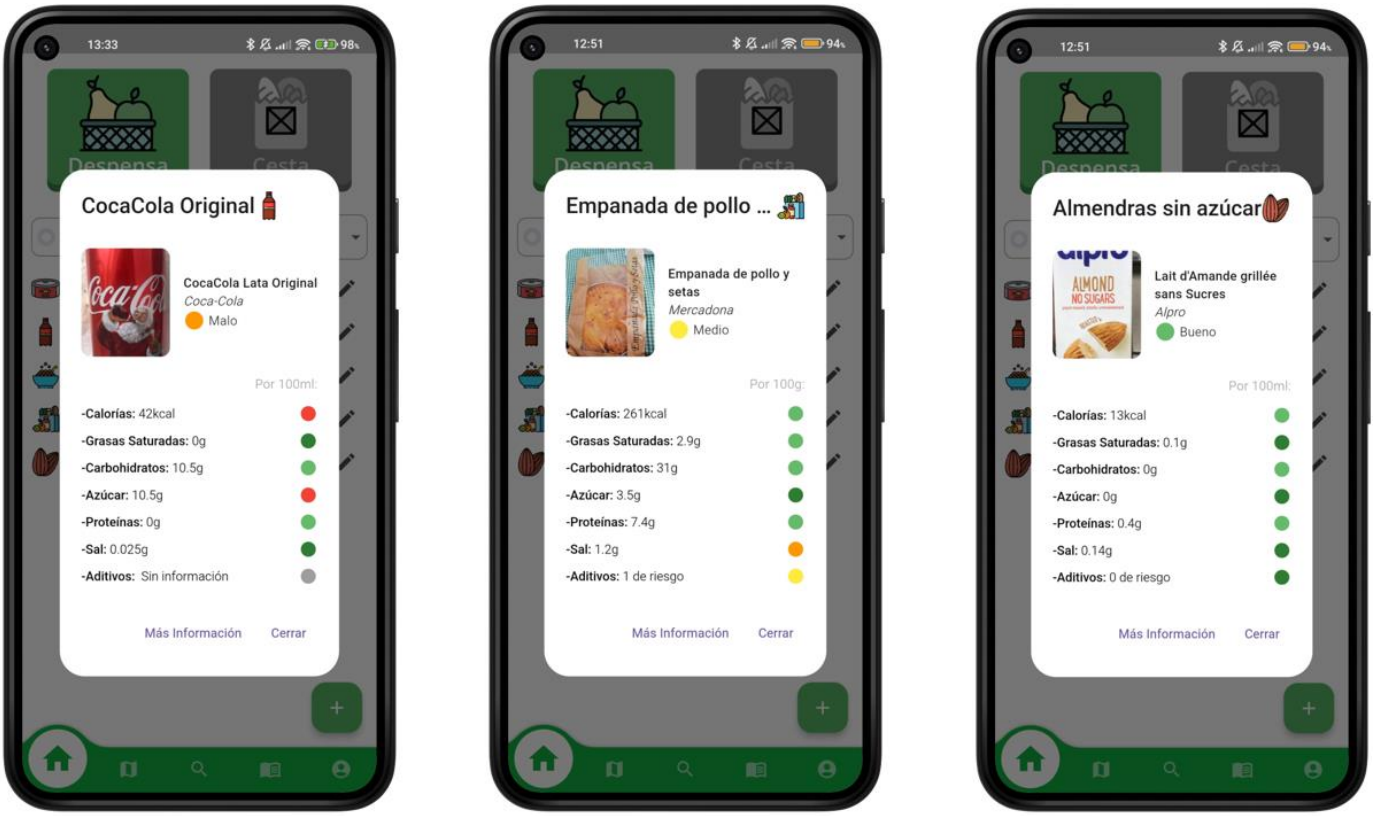

Ilustración 3.36: Ejemplos de información nutricional

Junto a cada valor nutricional (Calorías, Grasas Saturadas, Carbohidratos, etc.) se ofrece una evaluación sobre lo elevado o no que es cada elemento en función de su proporción.

Para evaluar los valores nutricionales de los alimentos **existen múltiples recursos y normativas que guían esta evaluación**, incluyendo las recomendaciones de la Organización Mundial de la Salud o OMS [19], la Administración de Alimentos y Medicamentos de Estados Unidos o FDA [20] y el Reglamento de la Unión Europea o UE [21] sobre información alimentaria facilitada al consumidor. Estos criterios pueden ser subjetivos debido a variaciones individuales en necesidades dietéticas, pero se establecen umbrales generales para orientar a los consumidores.

En base a las recomendaciones mencionadas y otros criterios se han elaborado ciertos umbrales para que la aplicación **califique automáticamente cada uno de los**

valores nutricionales de cada producto como más o menos recomendables. Estos umbrales están divididos entre líquidos y sólidos, ya que los líquidos se asimilan inmediatamente por el organismo, por lo que elementos como las calorías o los azucares deben penalizarse más duramente para estos.

Nutriente	Líquido	Rango de valores	Color	Descripción
kcal	Sí	<= 1	Verde oscuro	Perfecto
		<= 14	Verde claro	Bueno
		<= 35	Naranja	Algo alto
		> 35	Rojo	Muy alto
	No	<= 160	Verde oscuro	Perfecto
		<= 360	Verde claro	Bueno
		<= 560	Naranja	Algo alto
		> 560	Rojo	Muy alto
Grasa saturada	Sí	<= 1	Verde oscuro	Perfecto
		<= 3	Verde claro	Bueno
		<= 6	Naranja	Moderado
		> 6	Rojo	Muy malo
	No	<= 2	Verde oscuro	Perfecto
		<= 4	Verde claro	Bueno
		<= 7	Naranja	Moderado
		> 7	Rojo	Muy malo
Carbohidratos		<= 40	Verde claro	Bueno
		<= 60	Naranja	Moderado
		> 60	Rojo	Alto
Azúcares	Sí	< 1.5	Verde oscuro	Perfecto
		<= 3	Verde	Bueno
		<= 7	Naranja	Moderado
		> 7	Rojo	Alto
	No	< 9	Verde oscuro	Perfecto
		<= 18	Verde claro	Bueno
		<= 31	Naranja	Moderado
		> 31	Rojo	Alto
Proteínas		>= 8	Verde oscuro	Perfecto
		< 8	Verde claro	Bueno
Sal	Sí	<= 0.23	Verde oscuro	Perfecto
		<= 0.7	Verde claro	Bueno
		<= 1.4	Naranja	Moderado
		> 1.4	Rojo	Muy alto
	No	<= 0.46	Verde oscuro	Perfecto
		<= 0.92	Verde claro	Bueno
		<= 1.62	Naranja	Moderado
		> 1.62	Rojo	Muy alto

Ilustración 3.37: Umbrales para evaluación de valores nutricionales

A continuación se pasa a desglosar los criterios establecidos en función de las fuentes consultadas [19] [20] [21] [22] para cada uno de los valores nutricionales evaluados:

- **Energía (kcal)**: Se ha decidido establecer que un alimento líquido con más de 35 kcal y un alimento sólido con más de 560 kcal se consideren de alta densidad calórica. La ingesta calórica diaria recomendada varía entre 1,800 y 3,000 kcal, dependiendo de la edad, sexo y nivel de actividad física. El Reglamento de la UE sobre información alimentaria establece que los alimentos bajos en energía deben tener ≤ 40 kcal por 100 g para sólidos y ≤ 20 kcal por 100 ml para líquidos.
- **Grasas Saturadas**: Los valores para grasas saturadas se han establecido considerando que más de 6 g para líquidos y más de 7 g para sólidos se clasifiquen como "muy malos". La OMS recomienda que las grasas saturadas no excedan el 10% de la ingesta calórica diaria, equivalente a menos de 20 g en una dieta de 2,000 kcal. La UE define un alimento bajo en grasas saturadas como aquel que contiene ≤ 1.5 g por 100 g para sólidos y ≤ 0.75 g por 100 ml para líquidos (FDA).
- **Carbohidratos y Azúcares**: Para carbohidratos, se ha determinado que un valor moderado sea hasta 60 g por 100 g o ml, y alto más de 60 g. En cuanto a azúcares, se ha clasificado como alto más de 7 g para líquidos y más de 31 g para sólidos. Las guías alimentarias sugieren que los carbohidratos deben constituir entre el 45% y el 65% de la ingesta calórica total, y que los azúcares libres deben ser menos del 10% de esta ingesta. Estas recomendaciones están diseñadas para prevenir problemas como la obesidad y la caries dental. Según la normativa de la UE, un alimento es bajo en azúcares si contiene ≤ 5 g por 100 g para sólidos y ≤ 2.5 g por 100 ml para líquidos (FDA).
- **Proteínas**: Para proteínas, se ha considerado "perfecto" un valor mayor o igual a 8 g. Las recomendaciones diarias de proteínas son de aproximadamente 50 g en una dieta de 2,000 kcal. La FDA no

proporciona un valor específico por 100 g o ml, pero establece que el valor diario de referencia para proteínas es de 50 g.

- **Sal**: La OMS recomienda una ingesta de sal inferior a 5 g por día. Por tanto, se ha decidido que valores superiores a 1.4 g para líquidos y 1.62 g para sólidos se clasifiquen como "muy altos". El Reglamento de la UE define un alimento bajo en sal como aquel que contiene ≤ 0.12 g de sodio por 100 g o ml (FDA).

Como se ha mencionado anteriormente, estos umbrales no podrían llegar a ser completamente objetivos ya que los valores recomendados varían en función de las fuentes consultadas y múltiples criterios, por lo que la aplicación se limitará a ofrecer una evaluación base genérica a partir de la cuál orientarse pero no para ser tomada como un criterio definitivo.

Para los usuarios que deseen conocer más aspectos acerca de los productos consultados, se ha establecido un botón mediante el cual serán redirigidos a la página de *Open Food Facts* correspondiente al producto escaneado. Como se aprecia en la ilustración 3.38 mediante el botón “Más Información” se redirigía a la página del producto.

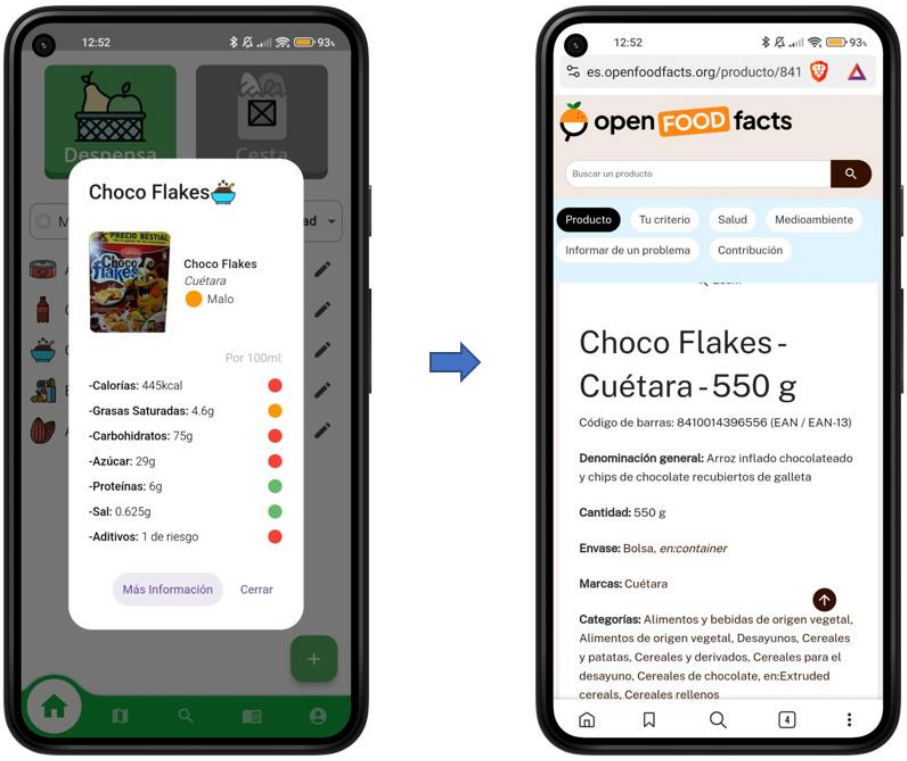

Ilustración 3.38: Mostrar más información de un producto escaneado

3.2.3.2 Pruebas

A continuación se mostrarán los resultados de los test realizados para el escaneo y obtención de la información nutricional de los productos.

T-1 Test de añadir productos mediante el escáner	
Objetivo	El sistema debe ser permitir escanear códigos de barras de distintos productos y añadir a la lista los productos escaneados
Resultado	Se permite escanear productos y agregarlos a las listas como se esperaba
Observaciones	Si la API de la que se hace uso se encuentra saturada puede llegar a causar una espera demasiado prolongadas en el proceso

Tabla 3.19: Test de añadir productos mediante el escáner

T-2 Test de flexibilidad de escaneo	
Objetivo	Durante el escaneo deben poder leerse varios productos secuencialmente
Resultado	Si ya se ha escaneado un producto y se apunta hacia otro, la ventana emergente del producto actual se cerraría y se abriría otra con la información del nuevo producto
Observaciones	Se ha necesitado implementar un tiempo de espera entre escaneo y escaneo para evitar que se abra y se cierre la ventana emergente con el mismo producto

Tabla 3.20: Test de flexibilidad de escaneo

T-3 Test de información nutricional accesible	
Objetivo	Debe poder consultarse la información nutricional de todos los alimentos escaneados siempre que se requiera
Resultado	Al pulsar sobre cualquier producto escaneado se accede a su información nutricional sin problemas
Observaciones	Al igual que durante el escaneo, la consulta de esta información depende de que el servidor de dónde se extrae la información no se encuentre saturado y de que el usuario tenga conexión a internet

Tabla 3.21: Test de información nutricional accesible

T-4 Test de evaluación nutricional	
Objetivo	Debe poder visualizarse la una evaluación tanto del producto como de los elementos que lo componen

Resultado	Junto con su información nutricional se puede observar una evaluación del mismo y sus valores nutricionales mediante mensajes y colores
Observaciones	Esta evaluación no siempre es acertada ya que se no existe una clara especificación de umbrales para cada valor nutricional y estos varían en función de las fuentes consultadas, por lo que se ha tratado de realizar una media

Tabla 3.22: Test de evaluación nutricional

3.2.4 Sistema de notificaciones locales

La finalidad de este sistema es ofrecer a los usuarios notificaciones personalizables para alertar sobre la caducidad de los alimentos que poseen en la despensa. Para desarrollar el mecanismo hay que diferenciar el proceso de mandar la notificación en sí del de programar una acción en segundo plano a una hora determinada que se encargue de comprobar si hay alimentos próximos a caducar.

El proceso de mandar una notificación se realiza de forma sencilla haciendo uso del paquete *flutter_local_notification* [23] que proporciona una interfaz para mostrar notificaciones en dispositivos Android e IOS y una forma accesible de manejar la interacción con las mismas.

Por otro lado, el proceso de programar una acción en segundo plano que se ejecute a una hora determinada es algo más complejo. Las tareas en segundo plano, son operaciones que se ejecutan sin la intervención del usuario y pueden continuar funcionando incluso cuando la aplicación está cerrada, esto puede suponer un aumento del gasto de batería en caso de realizar actividades pesadas, por lo que los sistemas operativos móviles tratan de limitar estas acciones de forma nativa y con el paso del tiempo aumentan estas restricciones. Para nuestro caso necesitamos ejecutar esta tarea en segundo plano para comprobar si hay alimentos con una fecha de caducidad próxima o no, lo cual no es una tarea pesada, sin embargo, las restricciones de los sistemas operativos le afectan de igual forma.

En el caso del sistema operativo Android, este ofrece distintas restricciones en cuanto a la ejecución de tareas en segundo plano en función de la versión del sistema que se utilice, el fabricante que lo implemente y la capa de personalización utilizada.

Además las políticas de ahorro de batería o modos de “No molestar” pueden afectar al comportamiento de las mismas.

Inicialmente, se consideró el uso del paquete *work_manager* [24] para programar las tareas de comprobación de caducidad. Sin embargo, al realizar pruebas se observó que su funcionamiento era inconsistente entre diferentes dispositivos y versiones de Android, especialmente en presencia de políticas agresivas de ahorro de batería. Por esta razón, se optó por utilizar *android_alarm_manager_plus* [25], un paquete de Flutter que ofrece una mayor versatilidad al permitir la ejecución de tareas en segundo plano de manera más consistente, similar a cómo funcionan las alarmas en Android.

Para personalizar las notificaciones se ha implementado una pantalla a modo de ajustes para activar o desactivar las notificaciones ,asi como poder personalizar aspectos como la hora de notificación o los días de antelación con los que quiere el usuario que se le avise. Estos ajustes se guardan en la memoria de la aplicación utilizando el paquete *shared_preferences* [26].

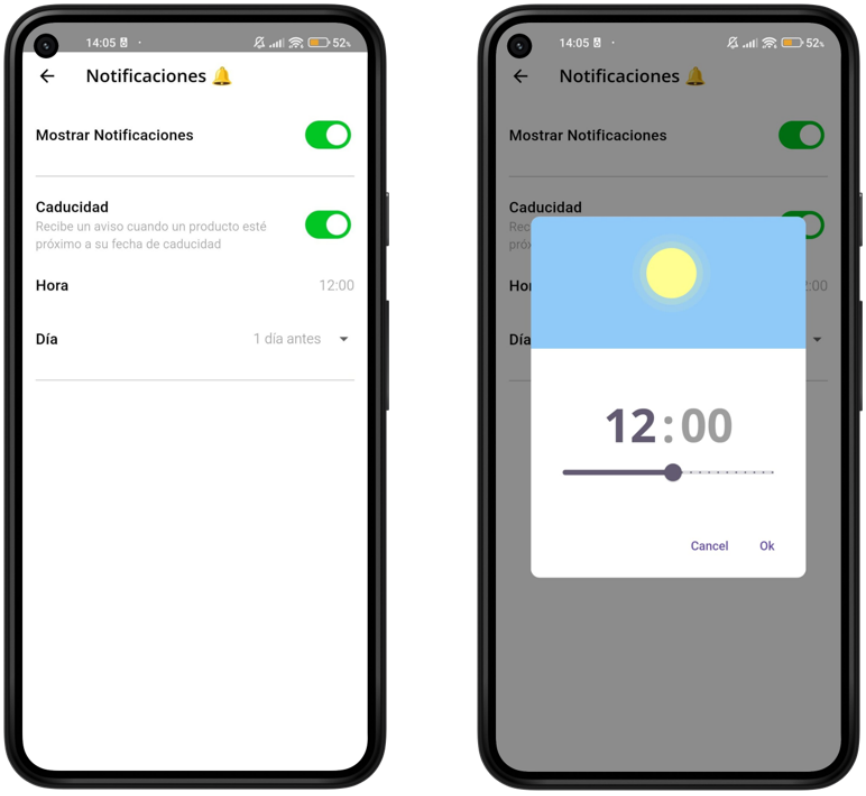

Ilustración 3.39: Pantalla de configuración de notificaciones

Respecto al funcionamiento del sistema, se ha definido una clase *AlarmManagerService* que se encarga de programar una alarma diaria que ejecuta una tarea en segundo plano para comprobar las fechas de caducidad de los productos en la despensa. Cuando llega la hora programada, el servicio comprueba, si están las notificaciones activadas, si hay algún producto próximo a caducar.

Si se encuentra algún producto que caduca en el período especificado por el usuario, se envía una notificación local utilizando *flutter_local_notifications*. Este proceso garantiza que las notificaciones se envíen a tiempo, independientemente del estado de la aplicación o las configuraciones del dispositivo. Durante el proceso se accede a distintos recursos como *DateTime* para consultar la hora, la memoria del dispositivo para verificar si están activadas las notificaciones y obtener los días de antelación con los que desea ser activado el usuario o la base de datos para acceder a la despensa. Podemos ver su funcionamiento más en detalle en el siguiente código.

```dart
@pragma('vm:entry-point')
  static Future<void> callback() async {
    final DateTime now = DateTime.now();
    print("Alarm fired at $now");

    SharedPreferences prefs = await SharedPreferences.getInstance();
    bool activeNotifications = prefs.getBool('activeNotifications') ?? false;
    bool expiryDateNotifications = prefs.getBool("expDateNotifications") ?? true;

    if (activeNotifications && expiryDateNotifications) {
      await DataBaseHelper.instance.init();
      List<Ingredient> pantryItems = await IngredientDao().readAll('pantry')?? [];
      List<Ingredient> alerts = [];

      final int daysBefore = prefs.getInt("dayNotification") ?? 1;

      for (var item in pantryItems) {
        DateTime expiryDate = item.date.subtract(Duration(days: daysBefore));
        if (sameDay(expiryDate, DateTime.now())) {
          alerts.add(item);
        }
      }

      if (alerts.isNotEmpty) {
        String alertBody = generateAlertBody(alerts, daysBefore);
        await NotificationService().showNotification(expiryTe, alertBody);
      }
    }
  }
```

Para el correcto funcionamiento del servicio de notificaciones, a partir de Android 13, se pide que el usuario de forma explícita acepte recibir notificaciones por parte de la aplicación. Esto es gestionado desde la propia pantalla de configuración

de las aplicaciones: Cuando un usuario activa las notificaciones, se utiliza el paquete *permission_handler* [27] para solicitar los permisos al usuario, controlando que en caso de que el usuario no los otorgue no permita activar el servicio de notificaciones y en caso de bloquear la petición del permiso, cosa que ocurre si se cancela la petición de permisos 3 veces, cuando se intenten activar las notificaciones en lugar de solicitar permisos se abrirá la pantalla de configuración de la aplicación en Android indicando al usuario que active los permisos manualmente.

3.2.4.1 Pruebas

A continuación se mostrarán los resultados de los test realizados para

T-1 Test de notificación de productos	
Objetivo	La aplicación debe notificar a los usuarios que hayan activado las notificaciones sobre los productos que vayan a caducar sin problemas
Resultado	Se reciben notificaciones sin problema, independientemente de si el dispositivo se encuentra en modo ahorro de batería o distintas limitaciones impuestas sobre la ejecución en segundo plano
Observaciones	Se han tenido en cuenta las distintas restricciones entre diferentes versiones del sistema operativo

Tabla 3.23: Test de notificación de productos

T-2 Test de personalización de notificaciones	
Objetivo	Las notificaciones deben poder ser personalizables
Resultado	A través de una pantalla de ajustes se pueden decidir si activar o desactivar las notificaciones, los días de antelación para recibir la alerta y la hora a la que recibirla
Observaciones	Funciona según lo esperado

Tabla 3.24: Test de personalización de las notificaciones

T-2 Test de control de permisos de notificaciones	
Objetivo	Deben gestionarse correctamente los permisos para notificaciones
Resultado	Se controla que no se puedan activar las notificaciones a menos que el usuario haya activado el permiso para eso. De la misma forma se guía al usuario en todo momento para activar las notificaciones teniendo en cuenta distintos escenarios

Observaciones	Funciona según lo esperado

Tabla 3.25: Test de control de permisos de notificaciones

3.2.5 Creación y gestión de cuentas de usuario

Firebase es la plataforma backend seleccionada para actuar como servidor de la aplicación. Los servicios que usaremos para este proyecto serán *Cloud Firestore*, una base de datos NoSQL en la nube, *Firebase Storage*, un sistema de almacenamiento de archivos multimedia con cargas y descargas seguras sin importar la calidad de la red y *Firebase Auth*, que es un servicio que puede autenticar a los usuarios utilizando únicamente código del lado del cliente [28].

3.2.5.1 Proceso de registro

El proceso de registro consta de las clases *RegisterScreen* para la interfaz y *RegisterProvider* para manejar la lógica de conexión con el servidor. La pantalla de registro, *RegisterScreen*, es el punto de entrada para los nuevos usuarios. Aquí, los usuarios ingresan sus datos, como el nombre de usuario, imagen de perfil, el correo electrónico y la contraseña, a través de un formulario. Una vez que el formulario se ha completado y validado, el proceso de registro se inicia al enviar los datos al *RegisterProvider*.

El ***RegisterProvider* maneja toda la lógica de interacción con *Firebase***. Al recibir los datos del usuario, primero verifica si el nombre de usuario o el correo electrónico ya existen en la base de datos de *Firebase Firestore*. Si cualquiera de ellos ya está en uso, se notifica al usuario y se detiene el proceso de registro. Si los datos son únicos, se procede a crear una nueva cuenta de usuario en *Firebase Authentication* utilizando el correo electrónico y la contraseña proporcionada.

Una vez creada la cuenta, si el usuario ha subido una imagen de perfil, esta se almacena en *Firebase Storage*, y se obtiene la URL pública de la imagen. Esta URL, junto con otros detalles del usuario, como el nombre de usuario, el correo electrónico, el rol del usuario, el token de notificación y la fecha de creación, se guardan en *Firebase Firestore*. Este paso asegura que toda la información del usuario esté centralizada y sea fácilmente accesible para otras partes de la aplicación.

El *RegisterProvider* también maneja la lógica para actualizar el perfil del usuario. Los usuarios pueden cambiar su nombre de usuario y su imagen de perfil en cualquier momento. Al igual que en el proceso de registro, primero se verifica que el nuevo nombre de usuario no esté en uso antes de actualizar la base de datos. Si se proporciona una nueva imagen, esta se sube a *Firebase Storage*, eliminando la fotografía antigua, y la URL de la imagen se actualiza en *Firestore*. Estos cambios se reflejan inmediatamente en el perfil del usuario, garantizando que la información esté siempre actualizada.

Como se ha mencionado anteriormente, un objetivo a seguir durante todo el proyecto es optimizar el uso de los recursos de *Firebase* todo lo posible. En este caso, para la subida de la fotografía de perfil, se realiza una compresión para reducir un 20% la calidad de la misma, esto es un cambio que no es apreciable en algo como una fotografía de perfil, pero si acaba resultando en una notable reducción de su tamaño en el servidor. Para realizar esta y otras reducciones del tamaño de las imágenes en la aplicación, se ha utilizado *image_picker* [29], este es el propio paquete que se utiliza para seleccionar las imágenes de la galería, que también ofrece la posibilidad de realizar una reducción en la calidad de las mismas para reducir su tamaño.

3.2.5.2 Proceso de inicio de sesión

El proceso de inicio de sesión **funciona de forma muy similar al de registro**, se divide en una clase *LoginScreen* para manejar la interfaz y el formulario y otra clase *LoginProvider* para controlar la interacción con el servidor. Durante este proceso hay que comprobar que el usuario exista, tanto en la base de datos como en la herramienta de autentificación de la plataforma, verificar que esté autentificado y que finalmente la contraseña introducida es correcta.

La principal diferencia con el registro del usuario reside en la **descarga de información**. El servidor además de los principales datos básicos de un perfil como nombre de usuario o fotografía, almacena otros datos del usuario que deben ser descargados y almacenados en la aplicación para evitar ser pedidos constantemente al servidor.

Un ejemplo de esto son las recetas creadas por el usuario. En la aplicación existe una sección de subida de recetas dónde, en caso de que se hayan subido recetas, aparece un listado de las recetas creadas y subidas. **Sería muy costoso que estas recetas se encontrasen siempre en el servidor** y se descargasen cada vez que se accediera a esta pantalla, por lo que, se ha optado por establecer una memoria en el dispositivo a modo de caché, para almacenar este tipo de datos.

En la parte inferior se muestra una de las funciones encargadas de descargar información del servidor durante el proceso de inicio de sesión que, siguiendo con el ejemplo anterior, es la función encargada de descargar las recetas publicadas por el usuario, convertirlas a objetos *Recipe* manejables por la aplicación y devolver este listado.

```
1. Future<List<Recipe>> getRecipes() async {
 2.   try {
 3.     User? currentUser = _auth.currentUser;
 4.     if (currentUser == null || !currentUser.emailVerified) return [];
 5.
 6.     DocumentSnapshot userDoc = await _firestore.collection('users')
.doc(currentUser.uid).get();
 7.     Map<String, dynamic>? userData = userDoc.data() as Map<String, dynamic>?;
 8.     if (!userDoc.exists || !(userData?.containsKey('recipes') ??false)) return [];
 9.
10.     List<String> recipeIds = List<String>.from(userData?['recipes'] ?? []);
11.     List<Recipe> recipes = [];
12.
13.     for (String id in recipeIds) {
14.       DocumentSnapshot recipeDoc = await _firestore.collection('recipes').doc(id).
get();
15.       Recipe recipeData = await convertDocToRecipe(recipeDoc);
16.       recipes.add(recipeData);
17.     }
18.
19.     return recipes;
20.   } catch (e) {
21.     return [];
22.   }
23. }
```

Para almacenar este tipo de datos en la aplicación **se descartó utilizar la base de datos SQLite ya existente** en el proyecto debido a que este tipo de almacenamiento implica trabajar con datos *Future*. Los datos *Future* es un valor o un error que estará disponible en el futuro, utilizan operaciones asíncronas y hay que esperar a su finalización. Actualmente **SQLite se utiliza para datos pequeños como son la base de datos de ingredientes y las listas de productos**, por lo que no supone tiempos de espera. Sin embargo, si queremos almacenar objetos más

complejos como una lista de recetas, no nos resulta conveniente usar SQLite debido a que supondría unos tiempos de espera considerablemente notables**.**

En su lugar **se ha optado por la utilización de *Hive*** [30]. Hive es una base de datos NoSQL rápida y ligera diseñada específicamente para Flutter y Dart. Proporciona almacenamiento local de datos de manera eficiente, permitiendo persistencia y recuperación rápida de la información en aplicaciones móviles. Una de las principales ventajas de Hive es que permite realizar operaciones de base de datos de forma síncrona. Esto significa que puedes acceder y modificar los datos directamente **sin tener que esperar a que una operación asíncrona se complete**. Esto simplifica el código y reduce la necesidad de manejar datos *Future* y usar operaciones *async/await*.

En términos de rendimiento, Hive está diseñado para ser extremadamente rápido y eficiente. Al ser una base de datos en memoria, muchas de sus operaciones son significativamente más rápidas en comparación con bases de datos tradicionales como SQLite. Al ser una base de datos NoSQL su gestión no necesita de tablas, en su lugar se utilizan “cajas” o “boxes”, los cuales almacenarán datos con una estructura de clave-valor.

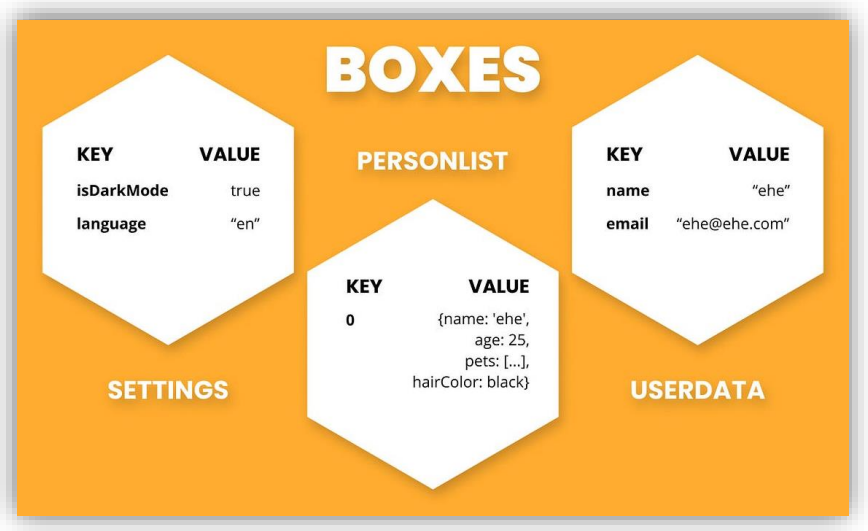

Ilustración 3.40: Ejemplo de almacenamiento usando Hive

Al poder acceder a los datos de forma instantánea, se ofrece como una opción perfecta para almacenar listados de recetas en memoria, ya que son objetos pesados con muchos campos y contenido multimedia como la fotografías.

Continuando con el ejemplo anterior de obtener las recetas publicadas por el usuario, en la parte inferior observamos las funciones para guardar y obtener las recetas de la memoria del dispositivo mediante *Hive*, dónde podemos apreciar la facilidad que implica la utilización de esta base de datos *NoSQL* además de observar como la operación *get* para obtener las recetas es inmediata sin tener necesidad de utilizar elementos *Future* ni funciones asíncronas.

```
1. Future<void> addPublishedRecipe(Recipe recipe) async {
2.     await _publishedRecipesBox.put(recipe.id, jsonEncode(recipe.toMap()));
3.   }
4.
5. List<Recipe> getPublishedRecipes() {
6.   return _publishedRecipesBox.values.map((recipe)
7.   => Recipe.fromMap(jsonDecode(recipe))).toList();
8. }
9.
```

3.2.5.3 Seguridad y autentificación

El método de autentificación para las cuentas usado es la verificación por correo electrónico. Después de que un usuario se registra, **se envía un correo electrónico de verificación a la dirección proporcionada**. *Firebase Auth* facilita este proceso, proporcionando una plantilla personalizable y herramientas para verificar si el usuario ha utilizado el enlace proporcionado o no.

Esto asegura que la dirección de correo electrónico es válida y que el usuario tiene acceso a ella. Los usuarios no podrán iniciar sesión hasta que hayan verificado su correo electrónico, lo que añade una capa adicional de seguridad al sistema. Si se intenta iniciar sesión sin haber realizado esto se mostrará un mensaje como se puede observar en la ilustración 3.41.

Ilustración 3.41: Solicitud de verificación por correo

La gestión de contraseñas es otro aspecto crucial, para este proyecto se ha realizado una encriptación de las contraseñas proporcionadas. Antes de almacenar una contraseña en la base de datos, **se cifra utilizando el algoritmo SHA-256**. Este es un algoritmo de hashing criptográfico que convierte la contraseña en una cadena de texto irreversible de longitud fija. Esto significa que, incluso si un atacante obtiene acceso a la base de datos, no podrá recuperar las contraseñas originales.

El algoritmo SHA-256 es parte de la familia de funciones de hash SHA-2 [31], funciona tomando una entrada, en este caso, la contraseña del usuario, y procesándola para generar una salida de 256 bits (32 bytes). Esta salida es única para cada entrada específica y no puede revertirse para obtener la entrada original, lo que se conoce como unidireccionalidad. Además, cualquier pequeño cambio en la entrada producirá un hash completamente diferente, lo que garantiza la integridad y la seguridad de los datos.

En el contexto del almacenamiento en el servidor, el uso de SHA-256 asegura que las contraseñas nunca se almacenan en texto plano. En su lugar, se almacena el hash de la contraseña. Durante el proceso de registro, cuando un usuario crea una cuenta, su contraseña se pasa por el algoritmo SHA-256 y el hash resultante se guarda en la base de datos. Este enfoque significa que, incluso si un atacante obtiene acceso a la base de datos, todo lo que tendrá son hashes, que son inútiles sin la capacidad de revertir el proceso de hash (lo cual es computacionalmente inviable).

Para la verificación durante el inicio de sesión, el sistema sigue un proceso similar. Cuando un usuario intenta iniciar sesión, la contraseña que ingresa se procesa con SHA-256 para generar un hash. Este hash se compara entonces con el hash almacenado en la base de datos. Si los hashes coinciden, se considera que la contraseña es correcta y se concede el acceso al usuario. Este método asegura que las contraseñas nunca se transmiten ni se almacenan en texto plano en ningún momento del proceso.

Para implementar esta funcionalidad en Flutter, se ha utilizado el paquete *crypt* [32]. Este paquete proporciona una interfaz sencilla para aplicar el algoritmo SHA-256 a las contraseñas.

Respecto a la seguridad en el acceso al servidor, esta se gestiona mediante las reglas de acceso configurables en *Firebase*. Las reglas de acceso de *Firebase* son un conjunto de instrucciones que definen quién puede leer y escribir datos en la base de datos *Firestore* y el almacenamiento *Firestorage*. Para el servicio de base de datos, se han especificado reglas para la colección de usuarios (*users*). En esta colección, se permite que ciertos campos de los documentos, como username y email, sean accesibles para usuarios sin identificar para los casos de registro de usuarios, ya que es necesario comprobar la disponibilidad de los mismos. El resto de campos de la colección solo serán accesibles para los usuarios autenticados correspondientes.

3.2.5.4 Moderación y uso de Perspective

Todos los usuarios registrados en la base de datos cuentan con un rol asignado a su cuenta. Los roles en cuentas de usuario son etiquetas asignadas a usuarios que determinan sus permisos y niveles de acceso dentro de una aplicación. Se utilizan

para controlar qué acciones puede realizar cada usuario, garantizando seguridad y organización según las funciones específicas de cada uno.

Los roles establecidos para este proyecto son ***user***, que sería el rol por defecto asignado a todos los usuarios, ***admin***, que le otorgaría permisos de moderación de contenido y ***superadmin***, que sería el rol asignado al gestor principal de la aplicación ya que también otorgaría permisos para visualizar y editar información de la base de datos.

Por defecto en la creación de la cuenta será asignado el rol *user* y tan solo desde la propia plataforma de Firebase se podrá editar un usuario para poder asignar un rol superior, asegurando que nadie sin acceso a la plataforma pueda editar sus permisos.

Otra característica que se ha querido añadir es para la moderación en la creación de los nombres de usuario. La aplicación está pensada para establecer una comunidad de usuarios que suban recetas a la plataforma, por lo que en algún momento, los perfiles de los usuarios deben de pasar a ser visibles por otras personas. Es deseable no encontrar contenido ofensivo en el nombre de usuario de estos perfiles, por lo que una práctica habitual es establecer algún tipo de filtrado que impida introducir este tipo de contenido.

Para realizar la detección de contenido ofensivo se ha usado *Perspective,* una API gratuita proporcionada por Google. Esta se especializa en el análisis de comentarios en línea para detectar contenido tóxico. Utiliza modelos de aprendizaje automático entrenados en grandes conjuntos de datos para identificar y puntuar varios atributos de toxicidad, tales como insultos, amenazas y lenguaje inapropiado. Los principales atributos que se pueden solicitar incluyen "TOXICITY" y "SEVERE_TOXICITY", entre otros.

En el código inferior se observa la función que gestiona su funcionamiento en la aplicación.

```
 1. Future<bool> containsOffensiveContent(String text) async {
 2.   var apiKey = getApiKey();
 3.   var url =
Uri.parse('https://commentanalyzer.googleapis.com/v1alpha1/comments:analyze?key=$apiKe
y');
 4.
 5.   var body = jsonEncode({
 6.     "comment": { "text": text },
 7.     "requestedAttributes": { "TOXICITY": {}, "SEVERE_TOXICITY": {} },
 8.     "languages": ["es", "en"],
 9.   });
10.
11.   var response = await http.post(url, body: body, headers: {'Content-Type':
'application/json'});
12.
13.   if (response.statusCode == 200) {
14.     var responseJson = json.decode(response.body);
15.     double toxicity =
responseJson['attributeScores']['TOXICITY']['summaryScore']['value'];
16.     double severeToxicity =
responseJson['attributeScores']['SEVERE_TOXICITY']['summaryScore']['value'];
17.     return toxicity > 0.1 || severeToxicity > 0.4;
18.   } else {
19.     print('API error: ${response.statusCode}');
20.     return false;  // Devuelve false en caso de error de la API
21.   }
22. }
23.
```

En la aplicación, la función *containsOffensiveContent()* se emplea para verificar si un nombre de usuario contiene contenido ofensivo antes de permitir el registro del usuario. Esta función envía una solicitud *HTTP POST* a la API de *Perspective* de Google, incluyendo el nombre de usuario como texto a analizar. La solicitud se configura con la URL de la API y una clave de API, especificando en el cuerpo de la solicitud los atributos de toxicidad que deben ser evaluados ("TOXICITY" y "SEVERE_TOXICITY"), así como los idiomas relevantes (español e inglés).

Al recibir una respuesta de la API, la función analiza el resultado. Si la respuesta es exitosa (código de estado 200), se extraen las puntuaciones de toxicidad del JSON de respuesta. La función evalúa estas puntuaciones comparándolas con umbrales predefinidos: una puntuación de toxicidad mayor a 0.1 o una puntuación de severa toxicidad mayor a 0.4 se considera indicativa de contenido ofensivo. Basado en esta evaluación, la función devuelve true si se detecta contenido ofensivo, y false en caso contrario. En caso de que la API falle, por ejemplo, debido a un error de red, la función también devuelve false para evitar bloquear el registro de usuarios debido a problemas técnicos.

3.2.5.5 Feedback durante el inicio de sesión y registro

Durante los procesos de inicio de sesión y registro, como se ha explicado, deben realizarse numerosos procedimientos secundarios, entre ellos, concretamente las descargas de recetas del servidor para completar listas de recetas personales del usuario, lista de recetas valoradas o listas de recetas subidas, pueden llegar a consumir un tiempo relativamente elevado si estas son de un tamaño considerable.

Mostrar una pantalla de carga sin ningún tipo de información adicional durante un tiempo elevado puede conducir al usuario a pensar que la aplicación se ha bloqueado o ha sufrido algún error. Para evitar estos posibles problemas, se ha añadido una pantalla de carga que incorpora un texto actualizable que va informando al usuario sobre los procesos que están siendo desarrollados mientras espera, eliminando la sensación de estancamiento para mostrar el progreso.

Ilustración 3.42: Ejemplos de textos de carga informativos

También, para mejorar la seguridad en la complejidad de las contraseñas se ha implementado en la interfaz de registro un medidor para proporcionar a los usuarios retroalimentación sobre la seguridad de las contraseñas introducidas. No se impone ningún nivel de seguridad necesario, pero sin embargo ayuda a los usuarios a ser conscientes de la seguridad de la contraseña que están creando.

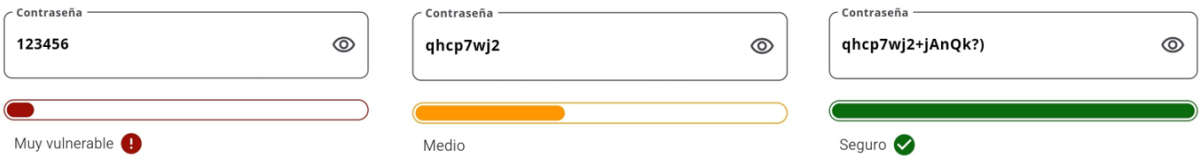

Ilustración 3.43: Ejemplos de retroalimentación con distintas contraseñas

3.2.5.6 Pruebas

A continuación se mostrarán los resultados de los test realizados para la creación y gestión de cuentas de usuario.

T-1 Test de conexión con Firebase	
Objetivo	La aplicación debe permitir realizar sus comunicaciones sin errores con el servidor
Resultado	La aplicación se comunica correctamente con el servidor
Observaciones	Funciona según lo esperado

Tabla 3.26: Test de conexión con Firebase

T-2 Test de registrar una cuenta	
Objetivo	Se debe permitir registrar un usuario correctamente
Resultado	Se permite realizar un registro teniendo en cuenta las restricciones correspondientes y realizando los procedimientos asociados correctamente
Observaciones	Funciona según lo esperado

Tabla 3.27: Test de registrar una cuenta

T-3 Test de verificación por correo	
Objetivo	El sistema debe de tener un sistema de verificación por correo
Resultado	Cuando un usuario realiza un registro, recibe un correo con un enlace de verificación al que debe acceder para poder iniciar sesión
Observaciones	El correo solo se recibe una vez sin ser posible solicitar un segundo enlace de verificación

Tabla 3.28: Test de verificación por correo

T-4 Test de iniciar sesión	
Objetivo	Se debe permitir iniciar sesión correctamente
Resultado	Se permite realizar un inicio de sesión teniendo en cuenta las restricciones correspondientes y realizando los procedimientos asociados correctamente
Observaciones	Funciona según lo esperado

Tabla 3.29: Test de iniciar sesión

T-5 Test de encriptación de contraseña	
Objetivo	El servidor no debe almacenar la contraseña en claro
Resultado	La aplicación utiliza una encriptación de la contraseña, generando un hash que es el que se almacena en el servidor
Observaciones	La contraseña se encripta desde la aplicación, no permitiendo que se exponga en ningún momento.

Tabla 3.30: Test de encriptación de contraseña

T-6 Test de estado de sesión	
Objetivo	La aplicación debe guardar y manejar el estado de sesión del usuario
Resultado	Se gestiona correctamente la información del estado de sesión en la memoria local del dispositivo, así como los cambios o borrados que pueda tener
Observaciones	Funciona según lo esperado

Tabla 3.31: Test de estado de sesión

T-7 Test de moderación de contenido en el registro	
Objetivo	No debe permitir usarse nombre de usuario con contenido ofensivo durante el registro
Resultado	Se rechazan los nombres de usuario potencialmente ofensivos, denegando el registro hasta que se modifique
Observaciones	Esta función es dependiente de una API gratuita de Google, por lo que si esta dejase de funcionar podría inhabilitarse la moderación.

Tabla 3.32: Test de moderación de contenido en el registro

3.2.6 Búsqueda de recetas

La búsqueda de recetas constituye una de las funcionalidades principales de la aplicación, siendo utilizada tanto en las pantallas de búsqueda como en las categorías de recetas, por lo que se prevé que sea una de las funciones más frecuentemente utilizadas por los usuarios. Dada su frecuencia de uso, es especialmente importante que la implementación de esta funcionalidad trate de **minimizar el número de lecturas en el servidor** todo lo posible para que no suponga costes innecesarios.

La búsqueda debe ser lo más flexible posible, permitiendo abarcar todos los parámetros definidos para las recetas. Además, se necesita que esta funcionalidad **permita ordenar los resultados en tres modos distintos**: por orden de likes descendente, por orden de creación descendente y por minutos de preparación ascendente. Junto con estos modos de ordenamiento, los usuarios deben poder **aplicar diversos filtros**, tales como tipo de dieta, duración máxima de preparación y presencia de videos en las recetas. La combinación de todos estos aspectos debe ser manejada de manera **que no se interfieran entre sí**, manteniendo al mismo tiempo la mayor eficiencia posible.

Existen diferentes modos de búsqueda en la aplicación: por ingredientes, por nombre de receta y por categorías. Para gestionar todos estos tipos de búsqueda y realizar las consultas según los parámetros proporcionados, se ha desarrollado una sola clase, denominada *SearchResults*. Esta clase centraliza la funcionalidad de búsqueda, permitiendo realizar consultas eficientes y flexibles. Por ejemplo, si a *SearchResults* se le pasan las etiquetas "postre" y los ingredientes "harina" y "huevo", devolverá las recetas etiquetadas como postre y que contengan harina y huevo entre sus ingredientes. Además, esta clase incorpora los modos de ordenamiento y filtros necesarios para refinar y modificar la consulta según las preferencias del usuario.

Parámetro	Descripción
recipeName	Nombre de la receta a buscar
ingredientsId	Lista de identificadores de ingredientes a incluir en la búsqueda
mealtime	Tiempo de comida (por ejemplo, desayuno, almuerzo, cena)
tags	Etiquetas asociadas a las recetas
region	Región geográfica de las recetas

hasVideo	Indica si la receta contiene un video
minMinutes	Tiempo mínimo de preparación
maxMinutes	Tiempo máximo de preparación
isVegetarian	Indica si la receta es vegetariana
isVegan	Indica si la receta es vegana

Tabla 3.33: Parámetros de búsqueda admisibles

Modo	Descripción
likesCount	Ordenar por número de likes en orden descendente
createdAt	Ordenar por fecha de creación en orden descendente
minutes	Ordenar por tiempo de preparación en orden ascendente

Tabla 3.34: Modos de orden utilizables

3.2.6.1 Paginación como técnica de optimización

El proceso de carga de recetas se realiza de manera incremental utilizando la paginación. La clase *SearchResults* mantiene una lista de recetas cargadas y una variable de control que indica si hay más recetas disponibles para cargar (*hasMoreRecipes*). La carga inicial y cada carga adicional de recetas se gestionan mediante el método *loadRecipes* encargado de realizar las consultas en la plataforma.

1. **Inicio de la Consulta:** Al iniciar la búsqueda, se configura una consulta básica a la colección recipes en Firestore. Esta consulta incluye el criterio de ordenamiento seleccionado (*orderBy*) y el límite de recetas a cargar por página (*pageSize*).
2. **Aplicación de parámetros de búsqueda:** Los parámetros de búsqueda especificados se aplican a esta consulta básica. En los apartados siguientes 3.2.6.2 , 3.2.6.3 y 3.2.6.4 se tratarán más en profundidad cada uno de los modos de búsqueda y la aplicación de los distintos parámetros que lo conforman.
3. **Punto de Inicio de la Carga:** Para manejar la paginación, si ya se han cargado recetas previamente, la consulta se modifica para empezar a cargar a partir del último documento cargado (*recipes.last*). Esto se hace utilizando el método *startAfterDocument* que viene predefinido para el objeto utilizado para las consultas en *Cloud Firestore*.

4. **Ejecución de la Consulta:** La consulta construida se ejecuta y se obtienen los resultados mediante el método get.
5. **Procesamiento de Resultados:** Los documentos resultantes se procesan y en función de los parámetros de búsqueda se realizan diferentes comportamientos.
6. **Actualización del Estado:** Los documentos ordenados se añaden a la lista de resultados y se actualiza el estado de carga (*isLoading* es establece en valor *false*). Si el número de recetas cargadas es menor que _pageSize, se marca que no hay más recetas disponibles (*hasMoreRecipes* se establece en valor *false*).
7. **Scroll y Carga Adicional:** El controlador de scroll (*scrollController*) supervisa la posición del deslizamiento sobre los resultados o *scroll*. Cuando el usuario alcanza el final de la lista y _isLoading es false, se llama nuevamente a loadRecipes para cargar la siguiente página de recetas.

En la siguiente ilustración se muestra un diagrama de flujo para visualizar más claramente el proceso que se sigue.

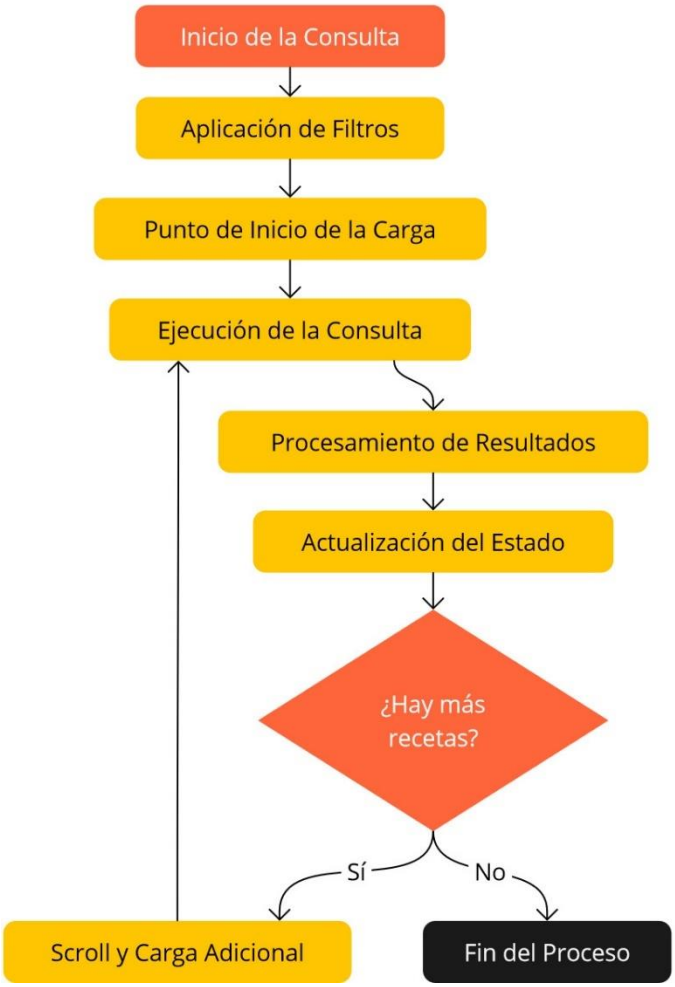

Ilustración 3.44: Diagrama de flujo del proceso de paginación

Por tanto solo se leen los documentos necesarios para cada página, en lugar de cargar todos los documentos de la colección. La mayoría de los filtros y criterios de ordenación se aplican directamente en la consulta a Firestore, reduciendo la cantidad de datos transferidos y procesados en el cliente. La carga incremental mediante paginación asegura que solo se carguen los datos necesarios en cada momento, mejorando el rendimiento y reduciendo el uso de ancho de banda.

3.2.6.2 Búsqueda por ingredientes

La clase encargada de la búsqueda de recetas recibe una lista de identificadores que corresponden a los ingredientes que se desean buscar. Estos identificadores son los mismos que poseen las recetas almacenadas en la base de datos. Para garantizar la precisión y eficiencia de la búsqueda, se implementa una verificación inicial que asegura que el número de ingredientes no supere los 30, dado

que este es el límite máximo que *Firebase* permite para búsquedas de tipo *arrayContainsAny*, que es el tipo de búsqueda que utilizaremos. Si la lista de ingredientes excede este límite, se reduce a los primeros 30 elementos, y se informa al usuario sobre esta restricción para evitar posibles errores en la consulta.

En la búsqueda propiamente dicha, **se utiliza el operador *arrayContainsAny***, el cual se utiliza para filtrar documentos donde un campo de tipo arreglo contiene al menos uno de los valores especificados. Es decir, **devuelve documentos donde el campo array contiene al menos uno de los elementos en la lista proporcionada**. Este tipo de búsqueda permite que se devuelvan todas las recetas que contengan cualquiera de los ingredientes especificados en la lista de búsqueda.

Sin embargo este método es que la búsqueda **devuelve todas las recetas que contienen al menos uno de los ingredientes**, lo que no necesariamente refleja las recetas más relevantes para el usuario. Por ejemplo, si se buscan recetas que contengan pollo, harina y huevo, el resultado incluirá todas las recetas que contengan cualquiera de estos ingredientes. Esto significa que tanto las recetas con solo uno de los ingredientes como las que contienen todos aparecerán en los resultados.

Para mejorar la relevancia de los resultados, **es fundamental que las recetas con mayor número de coincidencias de ingredientes aparezcan primero**. Es decir, si un usuario busca recetas que incluyan pollo, harina y huevo, las recetas que contengan estos tres ingredientes deben aparecer antes que las que contengan solo dos o uno de ellos.

Formalmente, si se buscan recetas con ingredientes $I_1, I_2, \ldots I_n$ las recetas que contengan todos los ingredientes $I_1 \cap I_2 \cap \cdots \cap I_n$ deben ser priorizadas, seguidas por las recetas que contengan combinaciones de $I_1 \cap I_2 \cap \cdots \cap I_{n-1}$, y así sucesivamente.

Para lograr esta ordenación por coincidencias, los resultados deben ser ordenados no solo por el número de coincidencias de ingredientes, sino también por otros criterios de ordenamiento especificados por el usuario. Los tres criterios de ordenamiento adicionales son: número de likes descendente, fecha de creación

descendente y tiempo de preparación ascendente. La solución consiste en **combinar estos dos tipos de ordenamiento** de manera que se mantenga la relevancia de las coincidencias de ingredientes junto con los criterios de preferencia del usuario.

El proceso de ordenación se realiza en dos pasos. Primero, se agrupan las recetas por el número de coincidencias de ingredientes. Luego, dentro de cada grupo, se ordenan las recetas según uno de los tres criterios adicionales especificados por el usuario. De este modo, los resultados finales se presentan en el siguiente orden:

1. **Recetas con el mayor número de coincidencias de ingredientes**, ordenadas según el criterio de “me gusta”, fecha de creación o tiempo de preparación.
2. **Recetas con un número intermedio de coincidencias de ingredientes**, nuevamente ordenadas según uno de los criterios adicionales.
3. **Recetas con el menor número de coincidencias de ingredientes**, ordenadas de acuerdo al criterio seleccionado.

En la ilustración inferior podemos ver un ejemplo de búsqueda, en el cual se muestran primero las recetas que contienen el mayor número de coincidencias y dentro de ese grupo es dónde se aplican los criterios de ordenación establecidos por el usuario.

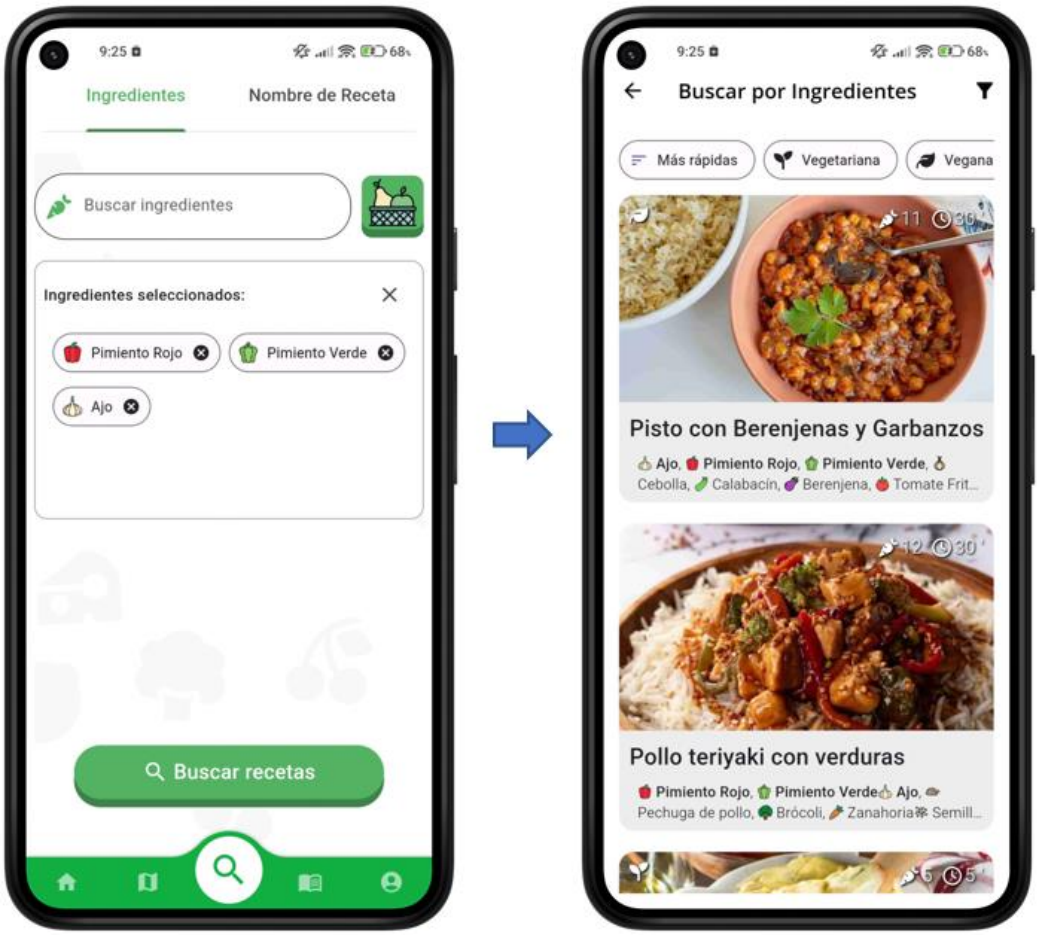

Ilustración 3.45: Ejemplo de búsqueda por ingredientes

La función _loadRecipes se encarga de gestionar la búsqueda y carga de recetas en la aplicación, asegurando que el proceso sea eficiente y que los resultados sean relevantes para el usuario. Para poder explicar más en detalle su funcionamiento se irá partes del código de la función junto con su explicación.

```
 1. Future<void> _loadRecipes() async {
 2.     if (!_hasMoreRecipes || _isLoading) return;
 3.
 4.     setState(() {
 5.       _isLoading = true;
 6.     });
 7.
 8.     Query query = FirebaseFirestore.instance.collection('recipes')
 9.         .orderBy(_orderBy, descending: !_ascending)
10.         .limit(_pageSize);
11.
```

Al inicio, la función verifica si existen más recetas por cargar y si no se está llevando a cabo otra carga en ese momento. Si alguna de estas condiciones no se

cumple, **la función se detiene para evitar sobrecargas y redundancias en la carga de datos**.

Una vez que se comprueba que es posible proceder, el estado de la aplicación se actualiza para indicar que el proceso de carga de recetas está en curso. Esto se realiza mediante la activación de un indicador de carga, lo que proporciona retroalimentación visual al usuario de que se está realizando una operación en segundo plano.

Se inicia una consulta básica a la colección de recetas, configurada para ordenar los resultados según el criterio especificado por el usuario, ya sea por número de likes, fecha de creación o tiempo de preparación, y limitando el número de resultados por página para manejar la paginación.

```
12.      ...otros métodos de búsqueda...
13.
14.     if (ingredientsId != null && ingredientsId!.isNotEmpty) {
15.       query = query.where('ingredients', arrayContainsAny: ingredientsId!);
16.     }
17.     if (widget.mealtime != null) {
18.       query = query.where('mealtime', arrayContains: widget.mealtime);
19.     }
20.     if (widget.tags != null && widget.tags!.isNotEmpty) {
21.       query = query.where('tags', arrayContainsAny: widget.tags!);
22.     }
23.
24.      ...comprobaciones del resto de filtros...
```

Si se proporciona una lista de ingredientes, la consulta se ajusta para incluir únicamente las recetas que contengan cualquiera de estos ingredientes, utilizando el operador ya mencionado arrayContainsAny. Esta técnica permite que la consulta devuelva todas las recetas que contengan al menos uno de los ingredientes especificados, aumentando así la relevancia de los resultados.

Además, se pueden aplicar otros filtros a la consulta, tales como el tiempo de comida (desayuno, almuerzo, cena), etiquetas específicas, región geográfica, y opciones dietéticas como recetas vegetarianas o veganas.

```
34.     if (_recipes.isNotEmpty) {
35.       query = query.startAfterDocument(_recipes.last);
36.     }
37.
38.     var snapshot = await query.get();
39.
40.     // Crear una lista para almacenar las recetas con el número de coincidencias
41.     List<Map<String, dynamic>> recipesWithMatches = [];
42.
43.     for (var doc in snapshot.docs) {
44.       int matches = 0;
45.       for (var ingredient in doc['ingredients']) {
46.         if (ingredientsId != null && ingredientsId!.contains(ingredient)) {
47.           matches++;
48.         }
49.       }
50.       recipesWithMatches.add({
51.         'doc': doc,
52.         'matches': matches,
53.       });
54.     }
```

Para implementar la paginación, la función verifica si ya se han cargado recetas previamente. Si es así, la consulta se ajusta para comenzar a cargar **a partir del último documento cargado**, evitando así la duplicación de resultados y mejorando la eficiencia de la carga.

Una vez configurada la consulta con todos los filtros y parámetros necesarios, se ejecuta para obtener los resultados. Estos resultados se procesan para **contar cuántos de los ingredientes especificados coinciden** con los ingredientes de cada receta. Este conteo de coincidencias es esencial para ordenar las recetas no solo por los criterios de ordenamiento especificados, sino también por la relevancia en términos de coincidencia de ingredientes.

```
56.     // Ordenar por coincidencias y parámetro de ordenación seleccionado
57.     recipesWithMatches.sort((a, b) {
58.       int compare = b['matches'].compareTo(a['matches']);
59.       if (compare != 0) {
60.         return compare;
61.       } else {
62.         return _ascending ? a['doc'][_orderBy].compareTo(b['doc'][_orderBy]) :
b['doc'][_orderBy].compareTo(a['doc'][_orderBy]);
63.       }
64.     });
65.
```

Después de procesar los resultados y calcular el número de coincidencias, las recetas se ordenan en dos pasos: **primero por el número de coincidencias de**

ingredientes y luego por el criterio de ordenamiento especificado por el usuario, ya sea por número de “me gusta”, fecha de creación o tiempo de preparación. Esta doble ordenación asegura que las recetas más relevantes y populares se presenten primero.

```
68.     List<DocumentSnapshot<Object?>> orderedRecipes = recipesWithMatches.map((item)
=> item['doc'] as DocumentSnapshot<Object?>).toList();
69.     if (orderedRecipes.length < _pageSize) {
70.       _hasMoreRecipes = false;  // No hay más recetas para cargar
71.     }
72.
73.     setState(() {
74.       _recipes.addAll(orderedRecipes);
75.       _isLoading = false;  // Desactiva el indicador de carga
76.     });
77.   }
78.
```

Finalmente, los documentos de recetas ordenados se añaden a la lista existente de recetas en el estado de la aplicación. Si el número de recetas cargadas es menor que el tamaño de la página, se marca que no hay más recetas disponibles para cargar, indicando que se ha alcanzado el final de los resultados posibles. El estado de la aplicación se actualiza para incluir las nuevas recetas y se desactiva el indicador de carga, permitiendo al usuario interactuar con los nuevos resultados de manera fluida.

3.2.6.3 Búsqueda por nombre

Para buscar recetas por su nombre en la base de datos, es importante entender las capacidades y limitaciones que ofrece *Firebase* en términos de búsqueda de texto. *Firebase Firestore* permite realizar búsquedas básicas de texto mediante consultas exactas y consultas de coincidencia parcial usando operadores como ***equals*** y ***arrayContains***, pero no proporciona una funcionalidad nativa para implementar búsquedas de texto completas o más complejas en torno a un campo específico.

En el contexto de la búsqueda por nombre de receta, ***Firestore* no ofrece una forma directa de buscar de manera eficiente dentro de un campo de texto**. Por ejemplo, si deseamos encontrar todas las recetas que contienen ciertas palabras clave en su nombre, *Firestore* no tiene una capacidad nativa para realizar una búsqueda de texto completo que abarquen diversas combinaciones y coincidencias parciales.

Para solucionar esta limitación, se ha decidido utilizar una estrategia basada en listas de palabras clave. Cada receta almacenada en *Firestore* **incluye un campo adicional que contendrá una lista de palabras clave** (*keywords*). Estas palabras clave son extraídas del nombre de la receta cuando se sube al servidor mediante una función llamada *extractKeywords().* Esta función analiza el nombre de la receta, identifica las palabras relevantes y las convierte en una lista de palabras clave asociadas a la receta.

El proceso para extraer las palabras clave consiste en convertir toda la cadena a minúscula, realizar un mapa de caracteres diacríticos a caracteres base para eliminar las tildes, eliminar emojis o signos de puntuación que puedan contener las recetas y por último se divide todo el nombre de la receta en palabras separadas y de ella se eliminan las palabras vacías. Las palabras vacías o *stopwords* son palabras que no tienen un valor significativo para el cometido buscado, como algunas preposiciones, artículos. En la ilustración inferior se puede observar un ejemplo de esto.

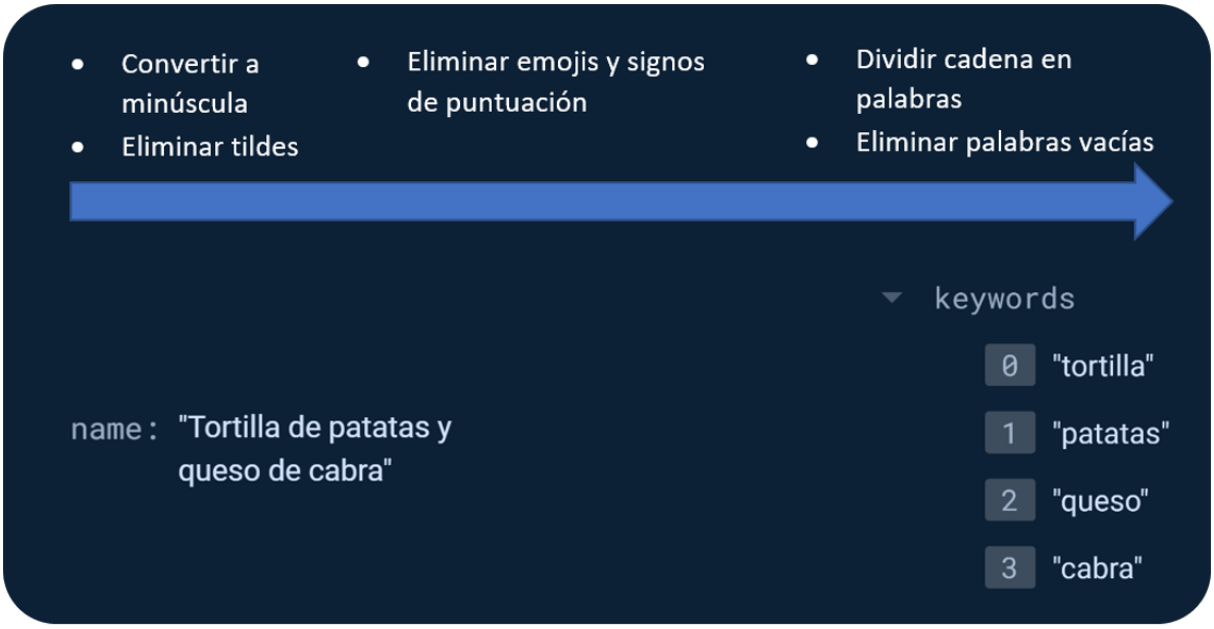

Ilustración 3.46: Esquema de ejemplo de extracción de palabras clave

Cuando un usuario realiza una búsqueda, los términos de búsqueda ingresados se procesan utilizando la misma función *extractKeywords().* Esto transforma los

términos de búsqueda en una lista de palabras clave que **pueden compararse con las listas de palabras clave almacenadas** en las recetas. El resultado es una búsqueda que compara listas de cadenas de texto, buscando coincidencias entre los términos de búsqueda y las palabras clave de las recetas.

Una vez transformados los términos de búsqueda, la consulta **se realiza utilizando el operador *arrayContainsAny***, mencionado en el apartado anterior. Similar a la búsqueda por ingredientes, los resultados deben ser ordenados primero por el número de coincidencias de palabras clave y luego por otros criterios de ordenamiento, como el número de likes, la fecha de creación o el tiempo de preparación. Así, los resultados se presentan en tres niveles de coincidencias: recetas con el mayor número de coincidencias de palabras clave, seguidas de recetas con coincidencias intermedias, y finalmente recetas con el menor número de coincidencias. Dentro de cada nivel de coincidencias, los resultados se ordenan según el criterio adicional especificado por el usuario.

Como se comentó previamente, todas las búsquedas se realizan desde una misma función que se adapta su comportamiento para realizar búsquedas distintas o combinadas en función de los parámetros que reciba la clase de búsqueda. En este caso como ya se contaba con una funcionalidad similar para la búsqueda por ingredientes, **se ha combinado con esta para aplicar el orden de los resultado**s.

A continuación se pasa a mostrar las partes del código de la función referentes a la búsqueda por nombre. Para el caso del código del orden de los resultados, se ha modificado el código dedicado a ordenar los resultados en función de los ingredientes mostrado en el apartado anterior para añadirle con la búsqueda de palabras clave.

```
Query query = FirebaseFirestore.instance.collection('recipes')
        .orderBy(_orderBy, descending: !_ascending)
        .limit(_pageSize);

if (widget.recipeName != null) {
  String recipeNameSearchTerm = widget.recipeName!.toLowerCase();
  List<String> recipeNameSearchTerms = extractKeywords(recipeNameSearchTerm);

  for (String term in recipeNameSearchTerms) {
    var tempQuery = query.where('keywords', arrayContains: term);


```

```
13.     var results = await tempQuery.get();
14.     if (results.size > 0) {
15.       query = tempQuery;
16.       break;
17.     }
18.   }
19. }
20.
```

Se comienza verificando si el usuario ha especificado un nombre de receta para buscar. Si es así, el término de búsqueda se pasa por *extractKeywords()* para generar una lista de palabras clave. A continuación, **se realizan subconsultas para verificar si estas palabras clave coinciden con las palabras clave almacenadas** en las recetas. Si alguna de estas subconsultas devuelve resultados, se actualiza la consulta principal para incluir solo esos resultados, afinando así la búsqueda según el nombre de la receta.

```
 1. List<Map<String, dynamic>> recipesWithMatches = [];
 2.
 3. List<String>? recipeNameSearchTerms;
 4. if (widget.recipeName != null) {
 5.   String recipeNameSearchTerm = widget.recipeName!.toLowerCase();
 6.   recipeNameSearchTerms = extractKeywords(recipeNameSearchTerm);
 7. }
 8.
 9. for (var doc in snapshot.docs) {
10.   int matches = 0;
11.   if (ingredientsId != null) {
12.     for (var ingredient in doc['ingredients']) {
13.       if (ingredientsId!.contains(ingredient)) {
14.         matches++;
15.       }
16.     }
17.   } else if (recipeNameSearchTerms != null) {
18.     for (var keyword in doc['keywords']) {
19.       if (recipeNameSearchTerms!.contains(keyword)) {
20.         matches++;
21.       }
22.     }
23.   }
24.   recipesWithMatches.add({
25.     'doc': doc,
26.     'matches': matches,
27.   });
28. }
29.
```

Una vez obtenidos los resultados de la consulta, se crea una lista para almacenar las recetas junto con el número de coincidencias de palabras clave. Este conteo de coincidencias es esencial para ordenar las recetas de manera que las más relevantes aparezcan primero. Si el usuario ha especificado ingredientes, se cuenta

cuántos de esos ingredientes coinciden con los ingredientes de cada receta. Si se ha especificado un nombre de receta, se cuenta **cuántas de las palabras clave de búsqueda coinciden** con las palabras clave de las recetas.

```
recipesWithMatches.sort((a, b) {
  int compare = b['matches'].compareTo(a['matches']);
  if (compare != 0) {
    return compare;
  } else {
    return _ascending ? a['doc'][_orderBy].compareTo(b['doc'][_orderBy]) :
b['doc'][_orderBy].compareTo(a['doc'][_orderBy]);
  }
});

List<DocumentSnapshot<Object?>> orderedRecipes = recipesWithMatches.map((item) =>
item['doc'] as DocumentSnapshot<Object?>).toList();
if (orderedRecipes.length < _pageSize) {
  _hasMoreRecipes = false;  // No hay más recetas para cargar
}

setState(() {
  _recipes.addAll(orderedRecipes);
  _isLoading = false;  // Desactiva el indicador de carga
});

```

El siguiente paso es ordenar las recetas en función del número de coincidencias de palabras clave y luego por los criterios adicionales especificados por el usuario. Este paso permanece igual que como había desarrolló para implementar la búsqueda por ingredientes. Por la estructura de la aplicación **nunca se llegarán a poder combinar una búsqueda por ingredientes con una búsqueda por nombre**, por lo que **el modo de ordenar de una búsqueda nunca colisionará** con el modo de ordenar recetas por el otro modo de búsqueda.

Finalmente, los documentos de recetas ordenados se agregan a la lista existente de recetas y se actualiza el estado de la aplicación para reflejar las nuevas recetas cargadas, desactivando el indicador de carga.

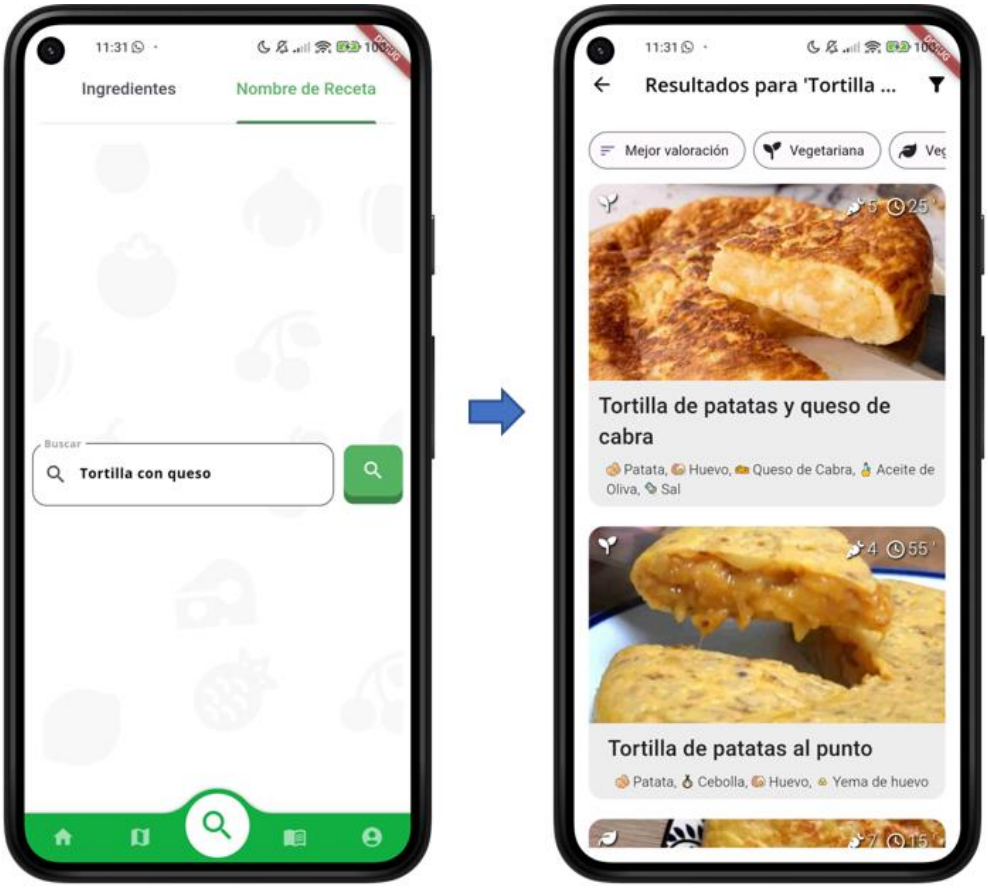

Ilustración 3.47: Ejemplo de búsqueda por nombre

En la ilustración 3.47 podemos observar el resultado de todo este proceso con un ejemplo de búsqueda por nombre. En este ejemplo vemos como se busca por el término “Tortilla con queso”, esto tras pasarse por la función de extracción de palabras clave buscaría por las palabras clave “tortilla” y “queso”. Vemos como en los resultados aparecen varias recetas coincidentes con el nombre de “tortilla”, sin embargo, la que aparece como primer resultado es una receta que contiene ambos resultados de búsqueda, “tortilla” y “queso”.

3.2.6.4 Búsqueda por categoría

El modo de búsqueda por categorías en la aplicación se ha diseñado para **aprovechar la infraestructura existente de otros modos de búsqueda**, asegurando así una implementación coherente y eficiente. Como se ha mencionado previamente, la clase *SearchResults* es responsable de procesar las consultas de búsqueda, aplicar filtros y ordenar los resultados. Esta clase ha sido diseñada para

ser extremadamente flexible, permitiendo el uso de una amplia gama de parámetros para ejecutar búsquedas de manera efectiva.

Gracias a la estructura versátil de la mencionada clase, es posible utilizarla para establecer y gestionar una variedad de categorías. Las recetas en la base de datos poseen diversos parámetros y etiquetas que permiten clasificarlas en categorías de manera intuitiva y eficiente. Dada a la multitud de categorías que se pueden llegar a realizar combinando las distintas etiquetas y campos que poseen las recetas, se ha optado por seleccionar un conjunto específico de categorías para mostrar en la pantalla de categorías, aunque la implementación está diseñada de manera que **pueda ampliarse fácilmente para incluir nuevas en el futuro**.

Para las categorías que se refieren a un solo parámetro, como "Postres", "Desayuno", "Mexicana", "Japonesa", entre otras, la clase *SearchResults* utiliza los campos pertinentes disponibles en los parámetros de la receta. Estos campos incluyen la ocasión, la región o etiquetas específicas. Al invocar la clase *SearchResults* con estos parámetros, se añaden a la consulta correspondiente y se ejecutan sin problemas, proporcionando los resultados esperados.

No obstante, hay ciertas categorías que no corresponden directamente a un parámetro específico en la base de datos, tales como "Carnes", "Pescados y Mariscos", "Arroces", "Legumbres" y "Pasta". Estas categorías se refieren más bien a los ingredientes que las recetas contienen en lugar de etiquetas predefinidas. Para manejar estas categorías, se ha hecho uso de la base de datos fija de ingredientes disponibles.

Para establecer estas categorías basadas en ingredientes, **se crean listas predefinidas de ingredientes que corresponden a cada categoría**. Por ejemplo, para la categoría de "Legumbres", se compila una lista con todos los identificadores de ingredientes que pertenecen a este grupo. Así, cuando un usuario selecciona la categoría "Legumbres", la aplicación busca recetas que contengan alguno de los ingredientes de esta lista específica.

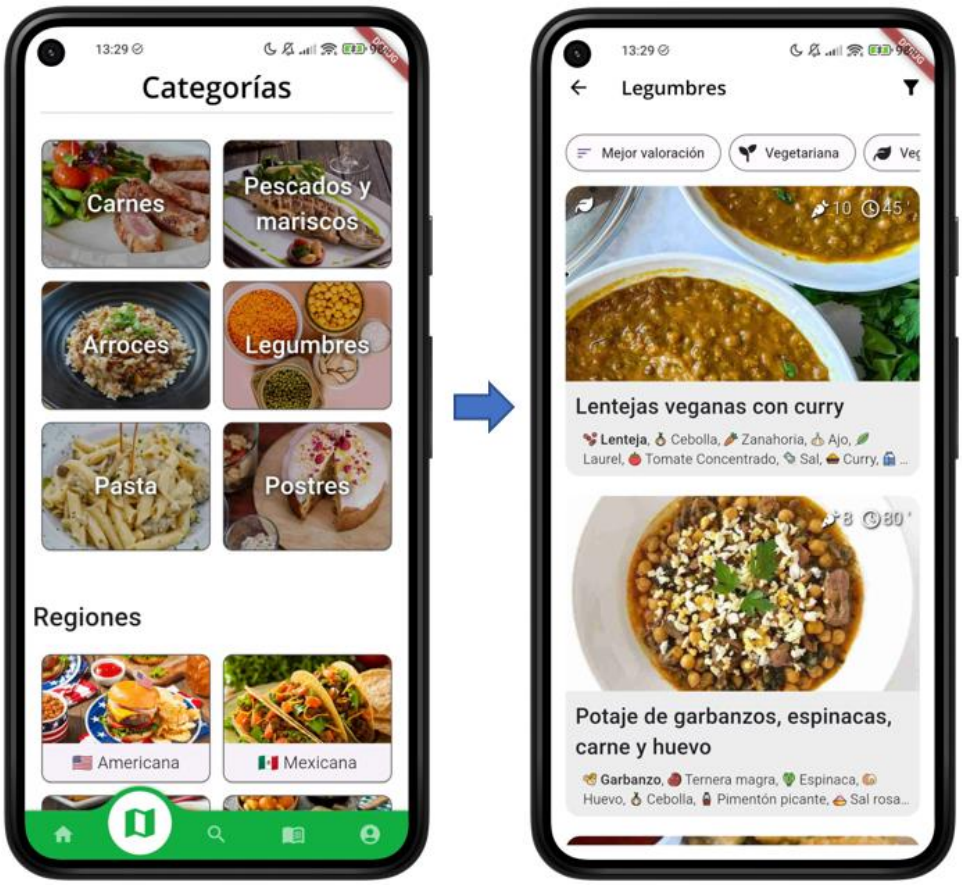

Ilustración 3.48: Ejemplo de búsqueda por categorías

Este enfoque ofrece al usuario la posibilidad de acceder rápidamente a categorías generales de recetas sin necesidad de especificar todos los detalles. El usuario puede consultar estas categorías generales de forma ágil, encontrando recetas que se ajusten a sus intereses sin tener que refinar al máximo los parámetros de búsqueda.

3.2.6.5 Filtros de búsqueda

La funcionalidad de filtros de búsqueda en la aplicación está diseñada para ofrecer a los usuarios una experiencia de búsqueda personalizada y precisa. Esto se logra mediante dos interfaces principales: los filtros rápidos presentados como chips interactivos en la pantalla de resultados y una pantalla de filtros avanzados accesible desde la misma.

En la pantalla de resultados, los filtros rápidos permiten a los usuarios aplicar ajustes básicos de manera inmediata. Estos chips interactivos incluyen opciones para

ordenar las recetas y aplicar filtros dietéticos. Por ejemplo, los usuarios pueden seleccionar filtros dietéticos para elegir si desean visualizar solo las recetas que se correspondan con una dieta vegana o con una dieta vegetariana. Al interactuar con estos chips, el estado del filtro correspondiente se actualiza dinámicamente. Por ejemplo, si un usuario selecciona el filtro vegetariano, se actualiza el estado *isVegetarian*, y se vuelve a cargar la lista de recetas para incluir solo aquellas que cumplen con este criterio. Este proceso se maneja internamente mediante la función *updateFilter*, que ajusta el estado y vuelve a ejecutar la consulta de búsqueda con los nuevos parámetros.

En la parte inferior se muestra una función de ejemplo solo para actualizar los campos vegetariano y vegano para mostrar el proceso de aplicación de filtros rápidos.

```
1. void updateFilter(String filterType, bool selected) {
2.     setState(() {
3.       switch (filterType) {
4.         case 'isVegetarian':
5.           _isVegetarian = selected;
6.           if (selected) _isVegan = false;
7.           break;
8.         case 'isVegan':
9.           _isVegan = selected;
10.          if (selected) _isVegetarian = false;
11.          break;
12.      }
13.      _recipes.clear(); // Limpiar la lista de recetas actual
14.      _hasMoreRecipes = true; // Resetear paginación
15.      _loadRecipes(); // Recargar recetas con nuevos filtros
16.    });
17.  }
18.
```

Para una personalización más detallada de los resultados de búsqueda, los usuarios pueden acceder a la pantalla de filtros avanzados, denominada *FilterScreen*. Esta pantalla ofrece una interfaz más completa para ajustar varios parámetros de búsqueda y aplicar filtros específicos. Los usuarios pueden seleccionar el criterio de ordenamiento a través de un menú con distintas opciones. La selección se refleja en el parámetro de ordenamiento de la búsqueda, y al confirmar los filtros, se actualiza el estado correspondiente en *SearchResults*.

Ilustración 3.49: Pantalla de filtros avanzados

Uno de estos ajustes a destacar es el de tiempo mínimo de preparación. Los intervalos disponibles van desde 15 minutos hasta un tiempo indefinido, y al ajustar este control deslizante, se actualiza el estado del filtro de tiempo de preparación. Este filtro se aplica mediante la función *applyFilters*, que ajusta el estado *maxMinutes* y desencadena una nueva búsqueda con este parámetro.

Un filtro adicional permite a los usuarios especificar si desean que las recetas incluyan un video. Este filtro se activa o desactiva mediante un control de selección y actualiza el estado *hasVideo*. Al ajustar este filtro, se vuelve a ejecutar la consulta de búsqueda para reflejar las preferencias del usuario.

Cuando los usuarios configuran y aplican filtros desde *FilterScreen*, la función *applyFilters* se encarga de recibir los parámetros seleccionados. Estos parámetros se utilizan para actualizar los estados correspondientes en *SearchResults*. Una vez actualizados los estados, se limpia la lista actual de recetas y se restablece la

paginación, asegurando que la siguiente búsqueda refleje los filtros aplicados y que los resultados sean precisos y relevantes.

A nivel técnico, la consulta a Firestore se ajusta para incluir los nuevos filtros mediante cláusulas *where* para los parámetros dietéticos y de video, así como ajustando el criterio de ordenamiento y el límite de tiempo de preparación. La consulta se ejecuta y los resultados se procesan para reflejar los filtros aplicados. Por ejemplo, si el filtro de tiempo máximo de preparación está activo, se incluye una condición en la consulta que filtra las recetas cuyo tiempo de preparación es menor o igual al especificado.

3.2.6.6 Índices compuestos

Firestore gestiona las búsquedas de una manera particular que requiere una planificación cuidadosa para consultas complejas. A diferencia de otras bases de datos que permiten realizar búsquedas sobre múltiples campos sin configuración adicional, *Firestore* exige la **creación de índices** para soportar estas consultas.

Estos índices son estructuras de datos diseñadas para mejorar la velocidad de las operaciones de consulta. Cada vez que se realiza una consulta, Firestore busca en estos índices para localizar rápidamente los documentos que cumplen con los criterios especificados. Por defecto, Firestore crea índices simples para cada campo en una colección, permitiendo búsquedas rápidas sobre un solo campo.

Sin embargo, cuando una consulta **involucra múltiples campos** o **requiere un orden específico**, **se necesitan índices compuestos**. Un índice compuesto en *Firestore* incluye dos o más campos de un documento, permitiendo realizar búsquedas que combinen filtros y ordenamientos en varios campos. Por ejemplo, si se necesita buscar recetas veganas, ordenarlas por fecha de creación y filtrar por ingredientes específicos, se requiere un índice compuesto que incluya los campos *isVegan*, *createdAt* e *ingredients*. La creación de estos índices permite a Firestore ejecutar consultas complejas de manera eficiente sin tener que escanear todos los documentos en una colección.

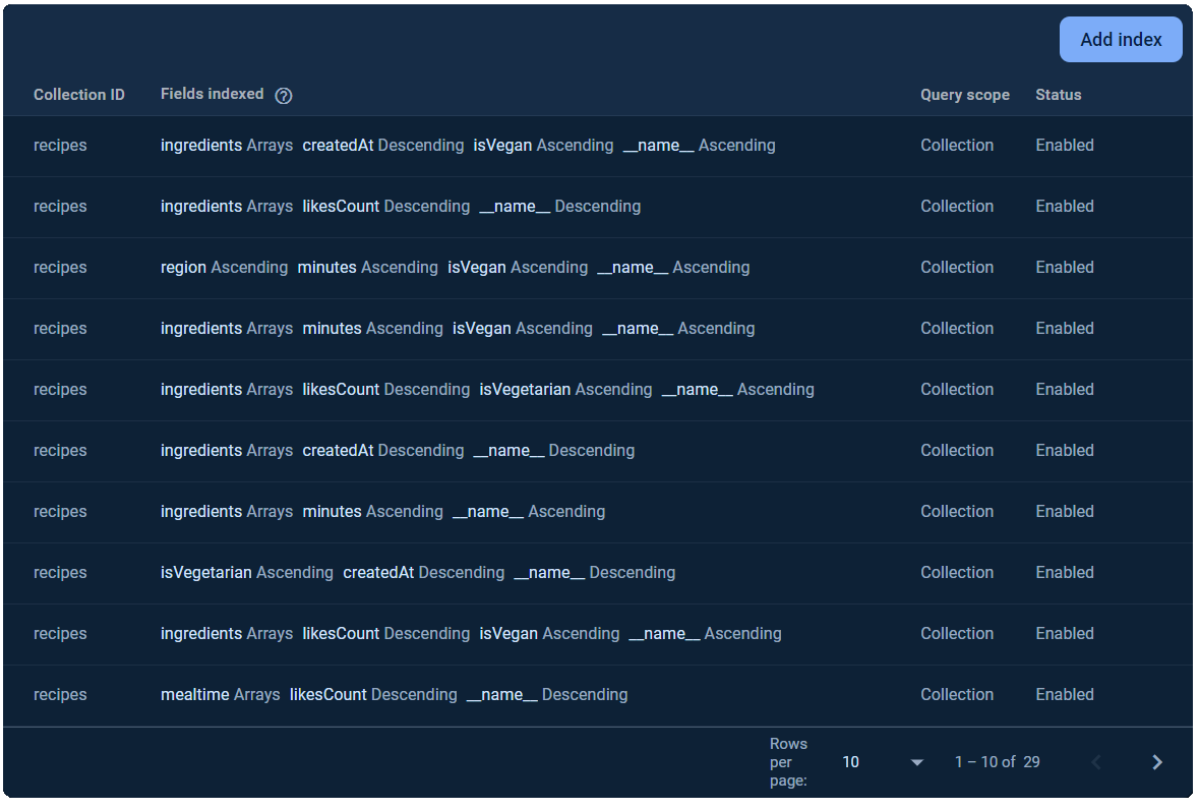

Ilustración 3.50: Ejemplo de algunos índices compuestos creados

En nuestro proyecto, los índices compuestos son esenciales debido a las diversas funcionalidades de búsqueda avanzada que ofrece la aplicación de recetas. Las búsquedas a menudo combinan múltiples filtros y criterios de ordenamiento, como encontrar recetas que contengan ciertos ingredientes, que contengan una etiqueta en concreto y estén ordenadas por un determinado parámetro. Para soportar esta complejidad, se han creado índices compuestos específicos para cada combinación de filtros y ordenamientos posibles dados los modos de búsqueda y filtros presentes.

3.2.6.7 Pruebas

A continuación se mostrarán los resultados de los test realizados para la búsqueda de recetas.

T-1 Test de búsqueda por ingredientes	
Objetivo	Deben poder buscarse recetas en base a los ingredientes que los componen

Resultado	Se devuelven los resultados esperados, mostrando las recetas en un orden de relevancia, de forma descendente a las coincidencias de ingredientes
Observaciones	Al haber varios modos de orden se ha priorizado el orden de coincidencia de los ingredientes antes que el resto de modos de orden especificables por el usuario, por lo que en algunos casos estos modos de orden pueden dar la sensación de no funcionar como se espera

Tabla 3.35: Test de búsqueda por ingredientes

T-2 Test de búsqueda por categorías	
Objetivo	Las recetas deben poder agruparse automaticamente por categorías accesibles por los usuarios
Resultado	Las categorías devuelven las recetas correctamente clasificadas
Observaciones	Funciona según lo esperado, permitiendo además que estas categorías sean extensibles de forma sencilla

Tabla 3.36: Test de búsqueda por categorías

T-3 Test de búsqueda por nombre	
Objetivo	Deben poder buscarse recetas en base a su nombre
Resultado	Se devuelven los resultados esperados, mostrando los nombres con más coincidencia primero
Observaciones	La búsqueda se realiza mediante palabras clave, por lo que hay veces que si el término de búsqueda difiere mucho del nombre buscado no se muestre el resultado esperado

Tabla 3.37: Test de búsqueda por nombre

T-4 Test de optimización de recursos	
Objetivo	Todas las consultas realizadas al servidor deben de tratar de utilizar la menor cantidad de lecturas posibles
Resultado	Se ha implementado un sistema de paginación para que tan solo se lean las recetas que el usuario haya visualizado en el momento en lugar de leer todos los resultados posibles para cada consulta
Observaciones	Se consigue una reducción notable en el número de consultas

Tabla 3.38: Test de optimización de recursos

T-5 Test de filtrado de resultados

Objetivo	Deben poder aplicarse filtros a todos los resultados de búsquedas de recetas, independientemente del modo de búsqueda utilizado
Resultado	Hay una variedad de filtros suficiente que funcionan correctamente en todos los modos de búsqueda
Observaciones	Funciona según lo esperado

Tabla 3.39: Test de filtrado de resultados

3.2.7 Interacción con las recetas del servidor

Antes de profundizar en las mecánicas de interacción con las recetas, es fundamental comprender cómo se ha estructurado el acceso a las mismas. Al buscar recetas en la aplicación, se obtiene una lista de objetos tipo *Recipe*, que representan a las recetas dentro de la aplicación. Cada objeto *Recipe* corresponde a un documento en la base de datos del servidor, y, por lo tanto, cada receta mostrada en la lista de resultados cuenta como una lectura.

El objetivo es reducir el número de lecturas al servidor para optimizar los costos de la plataforma. Aunque las lecturas para mostrar los resultados son inevitables, ya que se espera que la lista de resultados muestre el nombre, la fotografía y algunos datos adicionales para cada receta, se busca minimizar las lecturas adicionales.

La estrategia consiste en utilizar el objeto *Recipe* ya leído para la visualización de la lista de resultados, de manera que, cuando un usuario selecciona una receta para verla en detalle, **no sea necesario realizar una nueva lectura en el servidor**. En lugar de esto, se aprovecha el objeto previamente cargado en la memoria del dispositivo. Solo en el caso de que el usuario interactúe con el documento, se procederá a realizar lecturas o escrituras adicionales en el servidor.

3.2.7.1 Valoración de las recetas

El sistema de valoración de recetas consiste en la acción de dar “me gusta” por los usuarios, similar a la que se utiliza en redes sociales. Cada receta cuenta con un atributo llamado *likesCount*, que sirve para contabilizar el número de "me gusta" que ha recibido, para ordenar las búsquedas en función de la popularidad de las recetas. Además, cada receta tiene un atributo likes, que es una lista de identificadores de usuarios que han dado "me gusta", permitiendo llevar un control de quiénes han

interactuado con la receta. En paralelo, cada usuario también tiene un atributo *likesRecipes* en su documento, que contiene los identificadores de todas las recetas que les han gustado, facilitando así la asociación de cada usuario con las recetas que han marcado como favoritas.

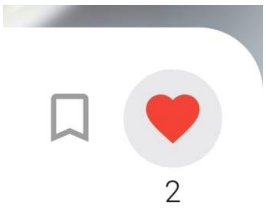

Ilustración 3.51: Botón para dar "me gusta" a una receta

En la pantalla de visualización de la receta, denominada *RecipeScreen*, existe un atributo booleano *isLiked* que controla si el usuario ha dado "me gusta" a una receta o no. Al cargar la pantalla, se verifica si el identificador del usuario actual se encuentra en la lista de usuarios que han dado "me gusta" a la receta. Si es así, la pantalla se carga con *isLiked* en valor true, mostrando visualmente que se ha dado "me gusta"; de lo contrario, se carga en false.

Para optimizar la interacción con el servidor, no se registra el "me gusta" inmediatamente cuando se pulsa el botón, sino que se hace al finalizar la interacción con la receta. Esta tarea la realiza una función llamada *updateLikes*(), que determina si se debe dar o quitar un "me gusta" o no hacer nada. Para más detalle se puede observar su código en la parte inferior:

```
 1. Future<void> updateLikes() async {
 2.     // Comprueba si el id de la receta está en LocalStorage().getLikeRecipes()
 3.     bool isLikedInLocalStorage = LocalStorage().getLikeRecipe(widget.recipe.id) !=
null;
 4.
 5.     if (!(isLikedInLocalStorage && !_isLiked) && !(!isLikedInLocalStorage &&
_isLiked)) {
 6.       Log.i('No changes to perform on likes.');
 7.       return;
 8.     }
 9.
10.     String email = LocalStorage().getEmail();
11.     String userId = FirebaseAuth.instance.currentUser?.uid ?? '';
12.
13.     try {
14.       await FirebaseFirestore.instance.runTransaction((transaction) async {
15.         final recipeDocumentRef = FirebaseFirestore.instance.collection('recipes')
.doc(widget.recipe.id);
16.
```

```
17.         final userDocumentRef = FirebaseFirestore.instance.collection('users')
.doc(userId);
18.
19.
20.         final recipeSnapshot = await transaction.get(recipeDocumentRef);
21.         final userSnapshot = await transaction.get(userDocumentRef);
22.
23.         int currentLikesCount = recipeSnapshot.data()?['likesCount'] ?? 0;
24.
25.         // Verifica si necesitamos remover el email del usuario de los likes
26.         if (isLikedInLocalStorage && !_isLiked) {
27.           Log.i('Removing like...');
28.           transaction.update(recipeDocumentRef, {
29.             'likes': FieldValue.arrayRemove([email]),
30.             'likesCount': currentLikesCount > 0 ? currentLikesCount - 1 : 0
31.           });
32.           transaction.update(userDocumentRef, {
33.             'likesRecipes': FieldValue.arrayRemove([widget.recipe.id])
34.           });
35.           await LocalStorage().deleteLikeRecipe(widget.recipe.id); // Elimina la
receta de _likesRecipesBox
36.           Log.i('Like removed successfully.');
37.         }
38.         // Verifica si necesitamos añadir el email del usuario a los likes
39.         else if (!isLikedInLocalStorage && _isLiked) {
40.           Log.i('Adding like...');
41.           transaction.update(recipeDocumentRef, {
42.             'likes': FieldValue.arrayUnion([email]),
43.             'likesCount': currentLikesCount + 1
44.           });
45.           transaction.update(userDocumentRef, {
46.             'likesRecipes': FieldValue.arrayUnion([widget.recipe.id])
47.           });
48.           await LocalStorage().addLikeRecipe(widget.recipe); // Añade la receta a
_likesRecipesBox
49.           Log.i('Like added successfully.');
50.         }
51.       });
52.     } catch (e) {
53.       Log.e('Error updating likes: $e');
54.     }
55.   }
```

La función *updateLikes()* sigue los siguientes pasos:

- **Comprobación Inicial**: Verifica si la receta ya está marcada como "me gusta" en el almacenamiento local (*LocalStorage*). Si no hay cambios necesarios (es decir, si el estado actual coincide con el estado en el almacenamiento local), no realiza ninguna acción y termina.
- **Obtención de Datos del Usuario**: Recupera el email y el identificador del usuario actual mediante *FirebaseAuth*.
- **Transacción en Firestore**: Ejecuta una transacción en Firebase Firestore para garantizar la consistencia de los datos y tras eso, recupera los documentos de la receta y del usuario. Es importante

obtener las listas actuales de "me gusta" y el conteo de "me gusta" de la receta por si la receta ha recibido cambios desde su obtención inicial en la búsqueda

- Actualizar Estado de "Me Gusta":
  - **Eliminar "Me Gusta"**: Si el usuario había dado "me gusta" anteriormente pero ahora desea quitarlo, la función elimina el identificador del usuario de la lista de likes de la receta y decrece el likesCount. También elimina el identificador de la receta de la lista likesRecipes del usuario en Firestore y del almacenamiento local.
  - **Agregar "Me Gusta"**: Si el usuario no había dado "me gusta" anteriormente y ahora desea hacerlo, la función añade el identificador del usuario a la lista de likes de la receta y aumenta el likesCount. Asimismo, agrega el identificador de la receta a la lista likesRecipes del usuario en Firestore y en el almacenamiento local.
- **Gestión de Errores**: En caso de errores durante la transacción, se registra el error para su posterior análisis.

Una vez se ha indicado un “me gusta” para una receta, además de guardarse en el servidor, también se guarda la receta en la memoria del dispositivo. De esta forma en la pantalla usuario se podrá mostrar una lista accesible a las recetas preferidas del usuario, como se puede ver en la ilustración inferior.

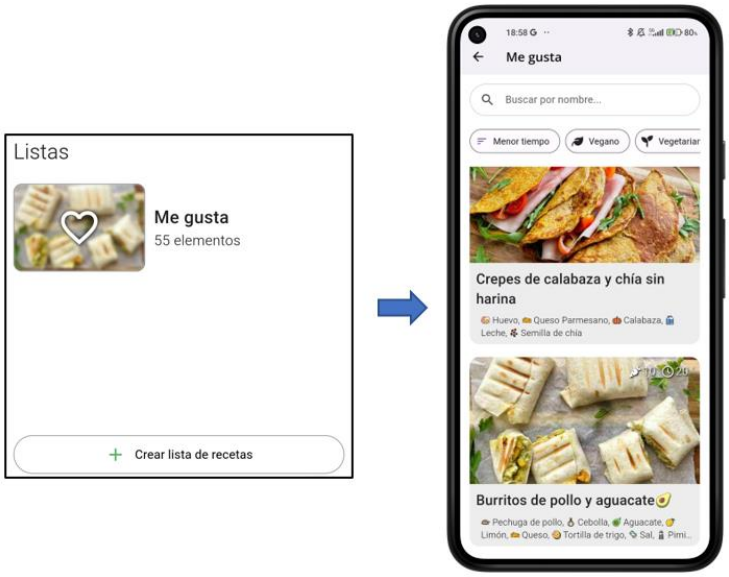

Ilustración 3.52: Lista de recetas valoradas por el usuario

Como se explicó en el apartado 3.2.5.2 (Proceso de Inicio de Sesión), las recetas que se guarden en memoria lo harán mediante Hive, mapeando las recetas en texto entendible por la base de datos. Además, de la misma forma que en el inicio de sesión se descargaban al almacenamiento del dispositivo las recetas que habían sido publicadas por el usuario, **también se descargan las recetas a las que el usuario ha dado "me gusta"**, para mantenerla en caso de que el usuario inicie sesión en otro dispositivo en algún momento. Por tanto, la lista de recetas valoradas por el usuario se almacena tanto en el servidor como en la memoria del dispositivo (para ahorrar lecturas frecuentes en la base de datos) aunque en todo momento se mantiene la coherencia de los cambios realizados en ambos sitios.

3.2.7.2 Listas de recetas

Las listas de recetas creadas por los usuarios tienen una interacción similar a la de valorar las recetas con un "me gusta", con la diferencia de que en este caso, en lugar de guardar la receta en una lista única, cada receta podrá ser guardada en varias listas de recetas a la vez, **añadiendo una capa extra de complejidad al tener que gestionar una lista de listas de recetas**.

Los usuarios pueden crear tantas listas como deseen, eligiendo nombre e icono para cada una. Estas listas no solo se almacenan en el servidor, sino también

localmente en el dispositivo. Este doble almacenamiento asegura que las listas sean accesibles sin necesidad de consultar constantemente al servidor, optimizando así las lecturas en Firestore o Firestorage.

Cada lista de recetas se almacena en la colección *recipes_list* del servidor, donde se asocia un email a cada lista para identificar a qué usuario pertenece, entre otros parámetros como se comentó en el apartado 2.4 (Diseño de la base de datos). Adicionalmente, en la colección users, cada documento de usuario contiene un atributo que es una lista de identificadores de las listas creadas por el usuario. Esto permite una fácil asociación entre los usuarios y sus listas de recetas.

En la pantalla de visualización de cada receta, denominada RecipeScreen, hay un botón que permite al usuario guardar la receta en una o varias de sus listas personales. Al pulsar este botón, se despliega un menú que ofrece la opción de guardar la receta en listas existentes o crear nuevas listas.

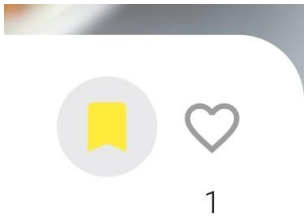

Ilustración 3.53: Botón para guardar una receta en listas de recetas

Ilustración 3.54: Menú desplegado para guardar la receta en listas

Al abrir este menú, la aplicación carga las listas de recetas almacenadas localmente y verifica si la receta actual ya está guardada en alguna de ellas. Esta verificación inicial se hace para actualizar visualmente el estado del botón, indicando si la receta ya ha sido guardada.

El usuario puede interactuar con el menú para seleccionar las listas en las que desea guardar la receta. También se ofrece la opción de crear una nueva lista en este mismo menú. Una vez que el usuario ha finalizado la selección, se presiona el botón de "Guardar", lo que desencadena el proceso de actualización en Firebase y en el almacenamiento local.

La aplicación optimiza esta interacción al diferir las operaciones de escritura en el servidor hasta que el usuario confirma su selección con el botón de guardado. Cuando se guarda una receta en una lista, se actualiza tanto en Firebase como localmente, añadiendo el identificador de la receta a la lista correspondiente. Si el usuario decide eliminar una receta de una lista, se realiza el mismo proceso, pero eliminando el identificador de la receta.

Al tratar con una lista de listas, la complejidad de las operaciones de guardado y recuperación se incrementa notablemente. Normalmente, para guardar y recuperar recetas en la memoria del dispositivo, se requiere mapear los valores de las recetas a un formato textual y luego convertir este formato textual de nuevo a objetos de recetas. Sin embargo, al manejar una lista de objetos que a su vez son listas de recetas, es necesario realizar un mapeo doble: primero mapear las listas de recetas a texto y luego, dentro de cada lista, mapear cada receta individualmente

Una operación dónde se puede apreciar esto es al tratar de recuperar las recetas del servidor en el inicio de sesión. El objetivo es que las listas de recetas del usuario se descarguen para que estén disponibles inmediatamente. **Según la estructura del servidor, este proceso requiere varios pasos**. Primero, se debe **acceder al documento** del usuario en la base de datos para obtener una lista de identificadores de listas de recetas, que representan las listas creadas por el usuario.

Cada uno de estos identificadores debe ser utilizado para **acceder a la colección *recipe_list* y recuperar el documento correspondiente** a cada lista.

Dentro de cada documento de lista de recetas, hay una lista de identificadores de recetas que la componen. Para recuperar las recetas individuales, es necesario **acceder a la colección recipes y obtener cada receta utilizando su identificador**. En la ilustración inferior se puede observar más claramente la estructura de cómo se almacenan estas listas.

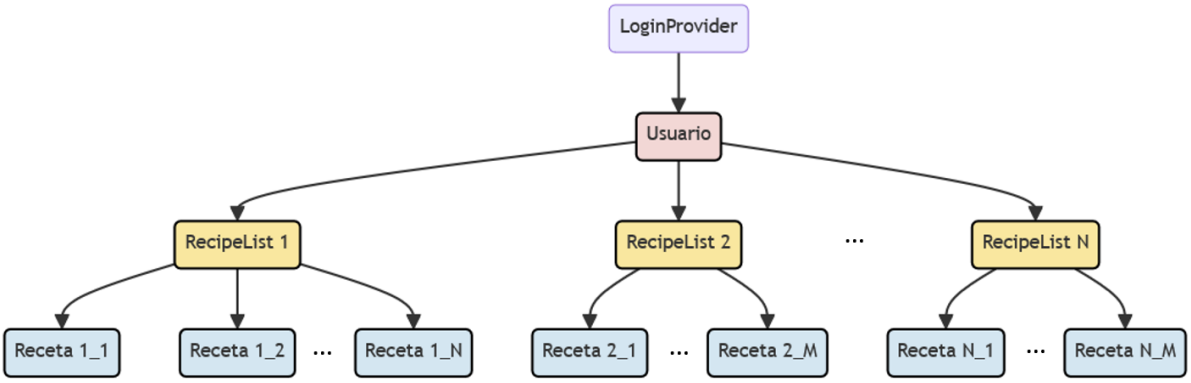

Ilustración 3.55: Esquema de almacenamiento de listas

Una vez que todas las recetas han sido recuperadas, **se deben ensamblar los objetos *recipe_list* en la aplicación**, integrando las recetas descargadas en sus respectivas listas. Este proceso implica múltiples consultas a la base de datos y una gestión cuidadosa de los datos para asegurar que se recuperen y estructuren correctamente.

En la función encargada de esta tarea, *getUserRecipeLists()*, se verifica primero si el usuario está registrado y ha verificado su correo electrónico. Luego, se obtiene el documento del usuario desde Firestore y se verifica si contiene identificadores de listas de recetas. Si es así, se extraen estos identificadores y se utilizan para acceder a la colección *recipe_list*, recuperando cada documento de lista. Para cada lista, se obtienen los identificadores de recetas que contiene y se procede a recuperar cada receta desde la colección recipes.

Cada receta recuperada se añade a su correspondiente objeto *recipe_list*, y una vez que todas las recetas han sido recuperadas y añadidas, se devuelve la lista

completa de objetos *recipe_list*. Para más detalle en la parte inferior podemos observar el funcionamiento de la función al completo.

```
Future<List<RecipeList>> getUserRecipeLists() async {
    try {
      Log.i('Obteniendo RecipeList del usuario...');

      // Verificar si el usuario está registrado y verificado
      User? currentUser = _auth.currentUser;
      if (currentUser == null || !currentUser.emailVerified) {
        Log.e('El usuario que trata de obtener las RecipeList no está registrado o verificado');
        return [];
      }

      // Obtener el documento del usuario
      DocumentSnapshot userDoc = await _firestore.collection('users').doc(currentUser.uid).get();

      // Verificar si el usuario tiene listas creadas
      Map<String, dynamic>? userData = userDoc.data() as Map<String, dynamic>?;
      if (!userDoc.exists || !(userData?.containsKey('recipeList') ?? false)) {
        Log.i('El usuario no tiene listas que recuperar');
        return [];
      }

      // Obtener los IDs de las RecipeList del usuario
      List<String> recipeListIds = List<String>.from(userData?['recipeList'] ?? []);

      // Obtener las RecipeList correspondientes a los IDs
      List<RecipeList> recipeLists = [];
      for (String id in recipeListIds) {
        DocumentSnapshot recipeListDoc = await _firestore.collection('recipe_list').doc(id).get();
        Map<String, dynamic> recipeListData = recipeListDoc.data() as Map<String, dynamic>;
        List<String> recipeIds = List<String>.from(recipeListData['recipes'] ?? []);

        recipeListData['recipes'] = [];
        RecipeList recipeList = RecipeList.fromMap(recipeListData);

        // Obtener las recetas correspondientes a los IDs
        for (String recipeId in recipeIds) {
          try {
            DocumentSnapshot recipeDoc = await _firestore.collection('recipes').doc(recipeId).get();
            Recipe recipeData = await convertDocToRecipe(recipeDoc);
            recipeList.addRecipe(recipeData);
          } catch (e) {
            Log.e('Error al obtener la receta con ID $recipeId: $e');
          }
        }

        recipeLists.add(recipeList);
      }

      return recipeLists;
    } catch (e) {
      Log.e('Error al intentar obtener las RecipeList: $e');
      return [];
    }
```

```
54.    }
55.
```

3.2.7.3 Pruebas

A continuación se mostrarán los resultados de los test realizados para la interacción con las recetas del servidor.

T-1 Test de valoración de recetas	
Objetivo	Los usuarios deben poder dar un "me gusta" a una receta como acción de valoración
Resultado	Se puede dar "me gusta" de forma correcta. Las recetas valoradas se guardarán en una lista accesible por el usuario
Observaciones	Funciona según lo esperado

Tabla 3.40: Test de valoración de recetas

T-2 Test de creación y uso de listas de recetas	
Objetivo	Los usuarios deben poder crear tantas listas de recetas como quieran e introducir las recetas que quieran
Resultado	El sistema de listas de recetas funciona correctamente, permitiendo al usuario crear listas con elementos personalizados y guardar las recetas en las listas que desee
Observaciones	Se han desarrollado algunas funciones extra para poder implementar un sistema de listas de recetas públicas en una futura actualización

Tabla 3.41: Test de creación y uso de listas de recetas

T-3 Test de descarga de recetas en caché	
Objetivo	Las recetas de todas las listas del usuario deberán descargarse y almacenarse en una caché para reducir las consultas al servidor
Resultado	Tanto las recetas valoradas como las incluidas en listas de recetas se almacenan en la memoria del dispositivo para evitar consultas innecesarias
Observaciones	Funciona correctamente aunque para mantener la coherencia de datos del usuario se ha implementado un sistema para descargar todas las recetas asociadas a una cuenta de usuario durante el inicio de sesión, de forma que no se pierda el estado al cambiar de dispositivo o cuenta

Tabla 3.42: Test de descarga de recetas en caché

T-4 Test de optimización de recursos durante la interacción	
Objetivo	Reducir las consultas al servidor durante la interacción con la receta
Resultado	Tan solo se interactúa con el servidor para realizar correctamente las acciones de dar "me gusta" o guardar la receta en listas de recetas
Observaciones	Funciona según lo esperado

Tabla 3.43: Test de optimización de recursos durante la interacción

3.2.8 Interfaz

3.2.8.1 Miniaturas de recetas

Las recetas son uno de los elementos principales en la aplicación, siendo además, de las más complejas de visualizar debido a la cantidad de datos, etiquetas e información multimedia que mostrar. Durante la búsqueda de recetas, asi como en las listas que se puedan encontrar, los elementos que las representarán son las miniaturas de recetas.

Las miniaturas de las recetas son elementos clave ya que van a aparecer en los resultados de búsqueda representando a las recetas y deben ofrecer un buen resumen de la información que contiene cada una, sin saturar al usuario y teniendo a su vez un aspecto agradable. Para construir las miniaturas, se ha creado una clase *RecipeCard()* que contendrá dos modos de visualización para poder adaptarse a distintas pantallas en función de sus necesidades.

Ilustración 3.56: Miniatura en modo 1

Ilustración 3.57: Miniatura en modo 2

El modo 1 será usado para mostrar las recetas en algunas listas de recetas como es el caso de la lista de publicaciones. Este modo muestra una tarjeta de receta más comprimida, de forma que pueda ser apropiada para un listado al poder incluir más elementos. Además muestra todas las etiquetas que tenga asociadas las recetas dentro de ella. Este inicialmente fue el único modo que iba a estar presente para mostrar recetas, sin embargo, más tarde se acabó descartando ya que no se daba la relevancia necesaria a la foto, llegando a no verse bien si la foto tenía un formato horizontal. Además aunque era capaz de mostrar todas las etiquetas de la receta, estas podían llegar a saturar al usuario cuando se mostraba en una lista junto a otras recetas.

Para resolver los problemas anteriormente mencionados se creó el modo 2 para la miniatura, que sería el usado en los resultados de búsqueda, que es dónde más relevancia debe de tener su apariencia. Para este se dio más importancia a la fotografía y para poder incluir las etiquetas se utilizó la propia fotografía, situando estos elementos en las esquinas con un sombreado. Además para el caso de la búsqueda por ingredientes, se añadió una opción para resaltar las coincidencias de ingredientes buscados como se puede observar en la ilustración 3.58.

Ilustración 3.58: Ejemplo de coincidencias de ingredientes señaladas

3.2.8.2 Generación de un estampado dinámico

La pantalla de inicio de sesión únicamente cuenta con dos campos de texto y un botón, lo cual le confiere un aspecto plano y vacío. Esta misma situación se observa en otras pantallas con funcionalidades limitadas, como las pantallas de búsqueda de

recetas. Por este motivo, se decidió **incluir un estampado** en el fondo de estas pantallas con el objetivo de mejorar su apariencia.

Inicialmente, se probó a crear un estampado como una imagen fija que solo se mostraría en el fondo. Sin embargo, esto generó problemas debido a las distintas resoluciones posibles, ya que en algunos casos el estampado no brindaba el aspecto deseado, dejando zonas vacías sin estampado, o presentando una densidad excesiva con elementos demasiado grandes o demasiado pequeños, entre otros inconvenientes.

Ante esta situación, se optó por crear un **estampado de manera manual**. Este método consiste en seleccionar una serie de imágenes vectoriales, de modo que la aplicación, al construir la pantalla, se encargue de generar un estampado con dichas imágenes. Además, al realizarse manualmente, es posible ajustar diversos aspectos, como dibujar los gráficos en diferentes colores dependiendo de si se está en modo oscuro o no, y adaptarse a los límites de cada pantalla. Asimismo, cada vez que se accede a una de estas pantallas, la generación del estampado será distinta, proporcionando un mayor dinamismo.

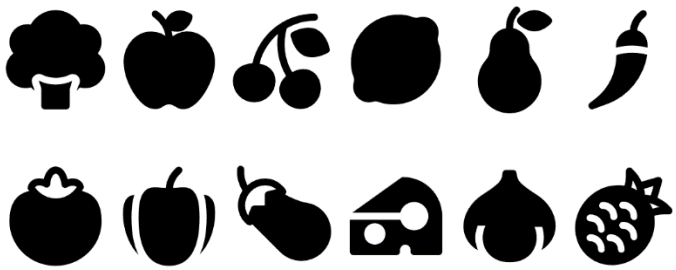

Ilustración 3.59: Siluetas utilizadas para formar el estampado

La función *generateSvgWidgets()* se encarga de crear un estampado visual dinámico utilizando imágenes SVG, adaptándose al tamaño y resolución de la pantalla en la que se despliega. A continuación, se describe su funcionamiento paso a paso:

En primer lugar, la función obtiene las dimensiones de la pantalla, almacenando el tamaño en una variable. A continuación, se calcula el número de filas (*rowCount*) que compondrán la cuadrícula del estampado, **utilizando la raíz cuadrada de la**

densidad multiplicada por sí misma, y redondeando el resultado hacia abajo. El espaciamiento base entre las imágenes SVG se determina dividiendo la menor dimensión de la pantalla (ancho o alto) por el número de filas.

La lista *svgWidgets* se limpia para asegurarse de que no haya elementos previos. Se inicializa un objeto de tipo *Random* para **generar posiciones aleatorias para las imágenes SVG**. Seguidamente, se crea una lista llamada positions que almacenará las coordenadas donde se colocarán las imágenes.

El siguiente paso es llenar la lista de posiciones con coordenadas aleatorias. Mientras la **longitud de las posiciones sea menor que el cuadrado de la densidad**, se generan coordenadas aleatorias *dx* y *dy* dentro de los límites del ancho y alto de la pantalla. Para evitar que las posiciones estén demasiado cerca unas de otras, se verifica que **la distancia entre una nueva posición y las ya existentes sea mayor que el espaciamiento base**. Si la nueva posición cumple con este criterio, se añade a la lista.

Una vez que se han generado todas las posiciones, se procede a crear y posicionar los widgets SVG. Para cada posición en la lista, se determina el índice del activo SVG que se utilizará, asegurando una rotación cíclica entre los activos disponibles la lista de siluetas. Luego, se calcula una rotación aleatoria para cada imagen, con un rango entre -0.2 y 0.2 radianes, para añadir variedad al estampado.

Finalmente, se añade cada imagen SVG a la lista *svgWidgets* como un widget de tipo *Positioned*, que coloca la imagen en las coordenadas dx y dy, centrada horizontal y verticalmente. La imagen se rota utilizando Transform.rotate, y se carga el activo SVG correspondiente, adaptando su color en función del brillo del tema de la aplicación (oscuro o claro).

En la parte inferior se observa el funcionamiento descrito en detalle.

```
1. void generateSvgWidgets() {
2.     final size = MediaQuery.of(context).size;
3.     final int rowCount = sqrt(density * density).floor();
4.     final double baseSpacing = min(size.width, size.height) / rowCount;
```

```
 5.
 6.      svgWidgets.clear();
 7.      final random = Random();
 8.
 9.      List<Offset> positions = [];
10.
11.      while (positions.length < density * density) {
12.        double dx = random.nextDouble() * size.width;
13.        double dy = random.nextDouble() * size.height;
14.
15.        // Asegurarse de que no esté demasiado cerca de otras posiciones
16.        bool tooClose = positions.any((pos) => (pos.dx - dx).abs() < baseSpacing &&
(pos.dy - dy).abs() < baseSpacing);
17.        if (!tooClose) {
18.          positions.add(Offset(dx, dy));
19.        }
20.      }
21.
22.      for (int i = 0; i < positions.length; i++) {
23.        final dx = positions[i].dx;
24.        final dy = positions[i].dy;
25.        final assetIndex = i % svgAssets.length;
26.        final asset = svgAssets[assetIndex];
27.        final rotation = (random.nextDouble() * 0.4) - 0.2; // Rotación aleatoria
entre -0.2 y 0.2 radianes
28.
29.        svgWidgets.add(Positioned(
30.          left: dx - width / 2, // Centra el SVG horizontalmente en dx
31.          top: dy - height / 2, // Centra el SVG verticalmente en dy
32.          child: Transform.rotate(
33.            angle: rotation,
34.            child: SvgPicture.asset(
35.              asset, width: width, height: height,
36.              color: MediaQuery.of(context).platformBrightness == Brightness.dark ?
Color(0xFF252424) : Color(0xFFF8F8F8),
37.            ),
38.          ),
39.        ));
40.      }
41.    }
42.
```

El resultado de esta función es un estampado dinámico que se genera de forma aleatoria cada vez que se cargue una pantalla, que se adapta a los modos claro y oscuro de la aplicación y a las dimensiones de cada pantalla.

En las ilustraciones mostradas a continuación se pueden observar los cambios de las pantallas a las que se les ha aplicado este estampado y ejemplos de la generación aleatoria de las mismas.

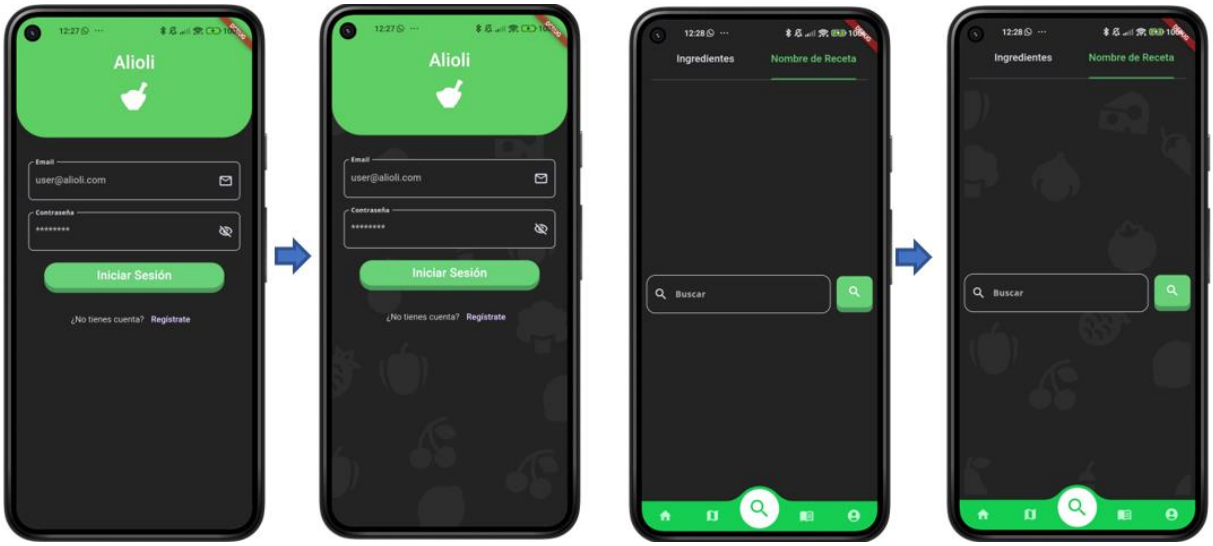

Ilustración 3.60: Pantallas antes y después del estampado

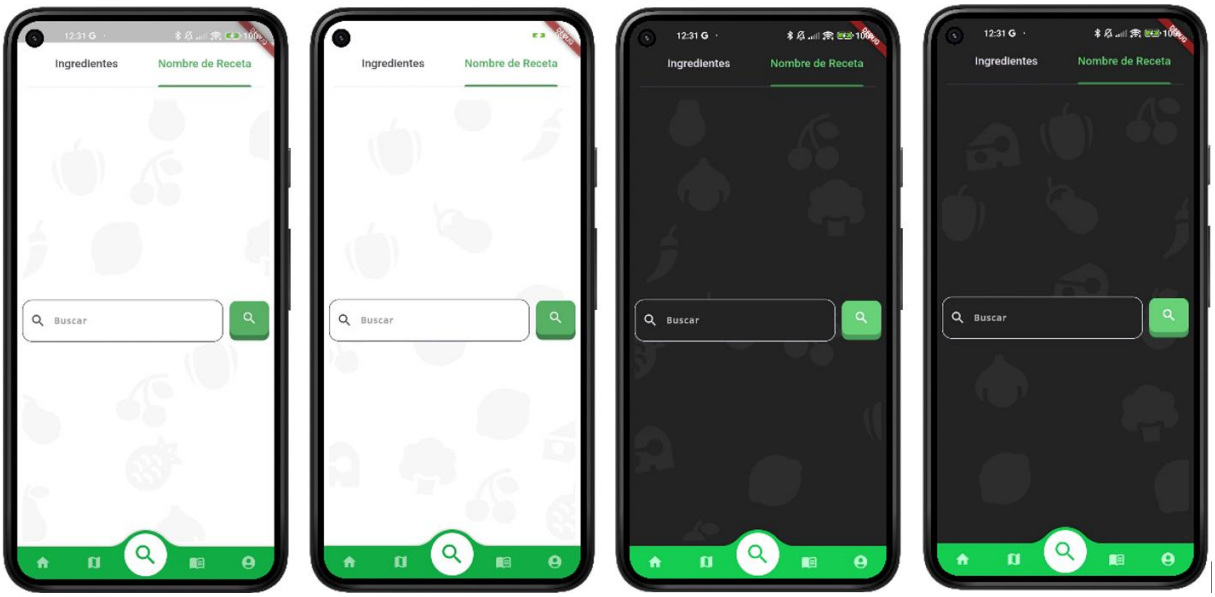

Ilustración 3.61: Ejemplos de estampados aleatorios

3.2.8.3 Pruebas

A continuación se mostrarán los resultados del test realizado para la interfaz. Para este apartado tan solo se ha realizado un test debido a que las pruebas de la interfaz se limitan a probar con los usuarios que la interfaz deja bien definida su funcionalidad y esta es minimalista y agradable para los usuarios, que era su cometido principal.

T-1 Test de funcionalidad de la interfaz	
Objetivo	Los elementos desarrollados en la interfaz son intuitivos y tienen una interfaz minimalista y amigable
Resultado	La interfaz resulta clara y limpia.
Observaciones	Los usuarios que probaron la aplicación no tuvieron problema para identificar la funcionalidad de cada elemento y les pareció que la interfaz era limpia y funcional

Tabla 3.44: Test de funcionalidad de la interfaz

4 RESULTADOS Y CONCLUSIONES

4.1 Resultados

El resultado de este proyecto es, además de una aplicación móvil multiplataforma que unifica múltiples funcionalidades de gestión de ingredientes y alimentación, una plataforma online con una comunidad de usuarios y base de datos de recetas.

Se ha conseguido el objetivo planteado, que era centralizar en un solo lugar las características ya existentes de otras aplicaciones de alimentación, de forma que además de conseguir unificar estas funcionalidades, también se consiga que se complementen entre sí para conseguir ese valor añadido.

Al tratarse de una aplicación móvil multipropósito alrededor de la alimentación, cuenta con un gran potencial ya que un usuario podría acudir a esta para gestionar todo tipo de aspectos relacionados con la gestión de su nutrición y gestión de alimentos. El proyecto desarrollado incluye todas las funcionalidades que ha dado tiempo a desarrollar en el tiempo planteado, pero se ha construido para que esté preparado para ser escalable a todo tipo de funcionalidades de forma sencilla y con poco esfuerzo.

Además, a pesar de que en un principio tan solo se buscaba desarrollar un prototipo, el proyecto ha resultado en una aplicación completamente funcional con gestión de errores y características completamente funcionales.

4.2 Conclusiones

La aplicación desarrollada en este proyecto cumple con la idea que se pretendía llevar a cabo, incluyendo características adicionales que han sido incorporadas durante el desarrollo del mismo, como es el caso de la función de escanear alimentos y consultar su información nutricional, funcionalidad que no estaba incluida en la idea principal para la aplicación. Para llevar a cabo este trabajo he tenido que profundizar en todas las etapas del desarrollo de una aplicación móvil

además de aprender sobre el desarrollo de una plataforma online con gestión de comunidad de usuarios.

Para conseguir llevarlo a cabo he necesitado profundizar en tecnologías como Flutter, dónde, gracias a la multitud y variedad de funcionalidades que se han incluido en la aplicación, me ha permitido desarrollar más flexibilidad en cuanto a su uso. De esta forma el conocimiento adquirido en cuanto a esta tecnología sería de gran ayuda para el desarrollo de otras aplicaciones móviles en un futuro.

He aprendido sobre el uso de Firebase y el desarrollo de una plataforma online basada en un servicio BaaS (Backend as a service), asi como de la optimización de los recursos e interacciones con el servidor incluyendo sistema de paginación y memoria caché. Además me ha permitido aprender a implementar un sistema de gestión de cuentas de usuario funcional y completo.

4.3 Posibles mejoras futuras

La aplicación ha sido desarrollada con un enfoque en la extensibilidad, permitiendo que pueda adaptarse y crecer con futuras mejoras. Como aplicación móvil multipropósito centrada en la alimentación, ofrece un amplio margen para añadir nuevas funcionalidades o expandir las existentes. A continuación, se presentan algunas de las posibles mejoras futuras que podrían implementarse:

- Las listas de la compra ya cumplen su función básica en la aplicación, pero una funcionalidad adicional útil sería permitir la compartición de estas listas entre varios usuarios. De esta manera, los miembros de una misma familia o grupo de convivencia podrían compartir y sincronizar los ingredientes de la despensa y la cesta de la compra.
- Actualmente, las listas de recetas permiten a los usuarios crear y gestionar sus propias colecciones. Sin embargo, dado que estas listas se almacenan en el servidor, una mejora interesante sería la opción de hacerlas públicas. Esto permitiría que los usuarios compartan y busquen listas de recetas creadas por otros, además de poder valorarlas.

- Otra mejora significativa sería la inclusión de información nutricional detallada para cada receta. Esto permitiría a los usuarios conocer los valores nutricionales de los platos o planificar sus comidas diarias o semanales en función de esta información. La posibilidad de calcular y seguir una dieta equilibrada directamente desde la aplicación aportaría un nuevo propósito a la misma.
- En las recetas se podría ampliar la interactividad incluyendo una sección de comentarios. Los usuarios podrían dejar opiniones y sugerencias, generando una mayor interacción entre la comunidad de usuarios.
- Aunque la aplicación ya cuenta con un sistema de perfiles de usuario y una plataforma dónde estos pueden subir recetas, ver las recetas de otros y valorarlas, existe un potencial significativo para aumentar la interacción social. Una mejora propuesta es que los perfiles sean públicos, de forma que un usuario pudiera visitar el perfil del usuario que ha subido una receta y realizar acciones como seguirlo o valorarlo.

5 APÉNDICES

En este apartado se incluirá la documentación adicional que pueda resultar de utilidad para la completa comprensión del proyecto.

5.1 Guía original del Trabajo Fin de Título

A continuación se incluirá la propuesta original del TFG publicada en la web de la EPS en el momento de la convocatoria, estructurada en las distintas secciones entre las que se compone.

5.1.1 Descripción corta del TFG

Cada vez hay más gente preocupada por comer sano y barato, pero en ocasiones no dispone de los conocimientos necesarios ni de la motivación para dedicar el tiempo necesario.

En este trabajo de fin de grado, se va a realizar un prototipo de aplicación móvil basada en Flutter y Firebase que permitirá a sus usuarios gestionar los ingredientes de su cocina de manera eficiente y acceder a una amplia base de datos de recetas compartidas por otros usuarios. Además, la app proporcionará una experiencia social, permitiendo a los usuarios compartir sus propias recetas, explorar las creaciones culinarias de otros y encontrar la motivación para dedicar tiempo a cocinar.

5.1.2 Datos del trabajo de fin de grado

- **Modalidad del TFG**: Proyecto de Ingeniería
- **Tipo de TFG**: TFG Específico
- **TFG en equipo**: No
- **Número de estudiantes**: 1

5.1.3 Tutor del trabajo de fin de grado

- **Nombre**: Francisco de Asís Conde Rodríguez
- **Departamento**: Informática

- **Área de conocimiento**: Lenguajes y sistemas informáticos

5.1.4 Conocimientos o Requisitos previos recomendados

Haber cursado la asignatura Desarrollo de Software para Dispositivos Móviles.

5.1.5 Objetivos del TFG

- Desarrollar un sistema de gestión de despensa que permita a los usuarios agregar y gestionar los ingredientes disponibles en su hogar.
- Implementar una plataforma de red social en la aplicación, permitiendo a los usuarios crear perfiles, subir/compartir recetas e interactuar con otros usuarios.
- Diseñar e implementar un algoritmo de recomendación que sugiera recetas a los usuarios en base a los ingredientes de su despensa.
- Proporcionar opciones de búsqueda avanzada en la aplicación, permitiendo a los usuarios filtrar recetas por ingredientes específicos y preferencias personales.
- Incorporar un sistema de seguimiento de la fecha de caducidad de los ingredientes ingresados por el usuario, y proporcionar notificaciones cuando un ingrediente esté próximo a caducarse, junto con sugerencias de recetas para utilizar dicho ingrediente antes de su vencimiento.
- Permitir la creación de listas de la compra: Proporcionar una funcionalidad para que los usuarios puedan crear y gestionar listas de la compra directamente desde la aplicación, pudiendo añadir directamente los ingredientes de recetas seleccionadas.
- Implementar un sistema de valoración y comentarios: Permitir a los usuarios calificar y dejar comentarios en las recetas compartidas por otros usuarios, fomentando la interacción y retroalimentación entre la comunidad culinaria.

5.1.6 Metodología a desarrollar

- Estudio de las necesidades de los usuarios interesados en gestionar los ingredientes de que disponen, encontrar recetas variadas y poder compartir experiencias sobre cocina con otros usuarios.
- Análisis del sistema, incluyendo requisitos funcionales y no funcionales.
- Estimación de costes y planificación temporal usando metodologías ágiles.
- Diseño de la interacción y prueba con usuarios reales.
- Diseño de la arquitectura del sistema incluyendo la base de datos y su integración en Firebase.
- Implementación de la solución.
- Pruebas de software.
- Redacción de la documentación final.

5.1.7 Documentos y formatos de entrega

- Documentación generada en el proceso, de acuerdo a las buenas prácticas de Ingeniería del Software, en formato digital.
- Memoria del trabajo, incluyendo anexos y manuales de usuario e instalación, en formato digital.
- Código fuente y ejecutables.
- Documentos y archivos adicionales a los que se haga mención expresa en la memoria.
- Vídeo demostración del sistema.
- Informe favorable del tutor.
- Autorización de publicación, si procede.

5.2 Instalación y configuración del sistema

Para instalar la aplicación en un dispositivo Android **únicamente nos hará falta el archivo instable (.apk)**. Nos dirigiremos a la ruta dónde se encuentre el archivo y al ejecutarlo se abrirá un sencillo menú para instalarlo como se visualiza en la imagen inferior.

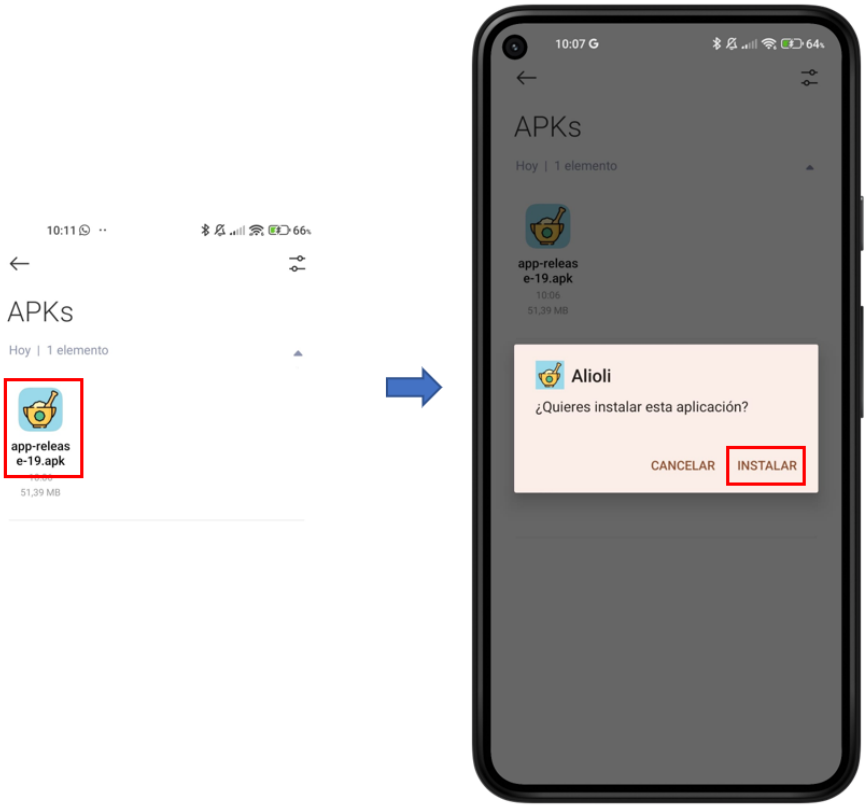

Ilustración 5.1: Instalación de la aplicación

Ya que la aplicación no proviene directamente de Google Play, **es posible que antes de instalar se solicite la activación de orígenes desconocidos**. Esta es una opción de Android para permitir instalar aplicaciones que no provengan de su tienda oficial de aplicaciones. El procedimiento para activar esta opción puede variar ligeramente dependiendo de la versión de Android, pero en general consiste en ir a la configuración del dispositivo, dirigirse al apartado de seguridad y buscar una opción de orígenes desconocidos similar a la ilustración inferior.

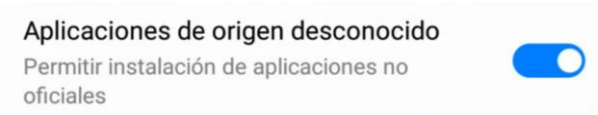

Ilustración 5.2: Opción para activar orígenes desconocidos

Para instalar la aplicación en un dispositivo IOS no se ofrece directamente un archivo instable ya que el desarrollo se ha seguido en Android al carecer de dispositivos Mac, sin embargo, al estar realizada en Flutter **fácilmente se podría exportar a este sistema**. Para ello habría que configurar Xcode, Flutter y Dart SDK en un dispositivo Mac, abrir el proyecto de la aplicación, conectar un dispositivo IOS (con la depuración activada) y ejecutar la aplicación en el dispositivo.

5.3 Manual de usuario

En este apartado se desarrollará un manual de usuario dónde se expondrán claramente las instrucciones para probar todas las funcionalidades desarrolladas así como los distintos aspectos que puedan precisar de una guía.

5.3.1 Primer uso de la aplicación

La primera vez que se abre la aplicación lo primero que se realizará es una petición de permisos para el servicio de notificaciones. Esta petición solo se hará en la primera apertura de la aplicación independientemente de que se permita o se rechace. En caso de rechazar el permiso de notificaciones si el usuario se dirige hacia los ajustes de notificaciones dentro de la aplicación se puede volver a solicitar este permiso sin problemas para activar las notificaciones de la aplicación.

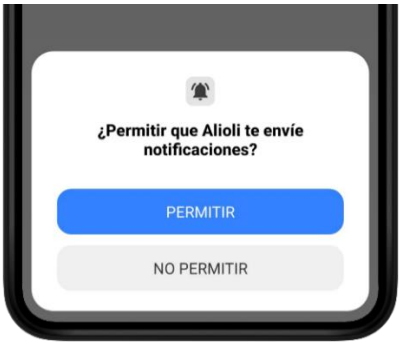

Ilustración 5.3: Solicitud de permiso de notificaciones

Tras esto, la primera vez que se inicie la aplicación se abrirá una pantalla de bienvenida en el que se expondrán las principales funcionalidades de la aplicación. Esta pantalla podrá omitirse en cualquier momento desde un botón que se encuentra en la parte inferior izquierda de la pantalla.

Ilustración 5.4: Saltar pantalla de bienvenida

Una vez finalizado u omitido este menú, se abrirá la pantalla de inicio de sesión y registro. Si **ya se posee una cuenta en la aplicación bastará con introducir el correo y la contraseña** para iniciar sesión. En caso de no poseer una **cuenta deberá registrarse en la plataforma**. Para registrarse hará falta un correo electrónico al que se tenga acceso, ya que posteriormente se enviará un correo de verificación al que se necesita acceder, un nombre de usuario y una contraseña. Tras registrarse, se recibirá un correo en la dirección especificada al que se tendrá que acceder para verificar la cuenta de usuario. Cuando la cuenta ya esté creada y el correo verificado se podrá iniciar sesión.

Si se ha iniciado sesión correctamente, las credenciales de usuario se guardarán y no hará falta volver a iniciar sesión a menos que se borren los datos de la aplicación o se haya desinstalado la misma.

5.3.2 Uso de las listas de productos

Las listas de alimentos se encuentran en el primer botón del menú inferior, una vez pulsado se accederá a las listas de productos de despensa y cesta.

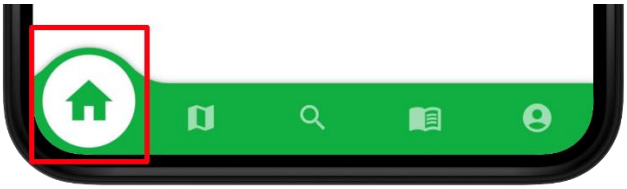

Ilustración 5.5: Botón para acceder a la sección de listas de alimentos

La sección se compone de la lista de despensa y la lista de cesta. Podremos alternar entre ambas listas mediante los botones superiores. Se mostrará con color la lista en la que nos encontramos y en blanco y negro y más apagado la lista en la que no nos encontramos (como se puede observar en la ilustración inferior). En ambas listas habrá un botón en la parte inferior derecha para añadir productos a cada respectiva lista, este se mostrará del mismo color que la sección en la nos encontramos para ofrecer mayor claridad (verde para la despensa y azul para la cesta de la compra).

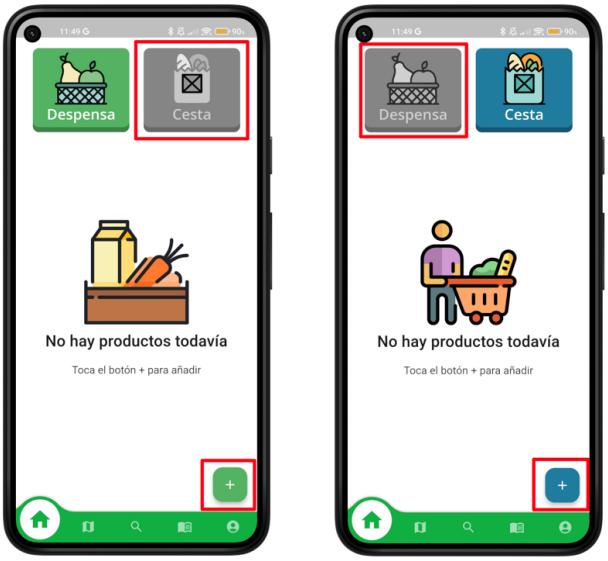

Ilustración 5.6: Botones para cambiar entre listas y añadir productos

Al pulsar el botón de añadir productos se abrirá un menú dónde se da a elegir entre la opción de añadir productos manualmente o añadirlos mediante el escáner del código de barras. La opción de añadirlos mediante el escáner se tratará en el siguiente apartado, por lo que a continuación se explicará el uso del modo de adición manual.

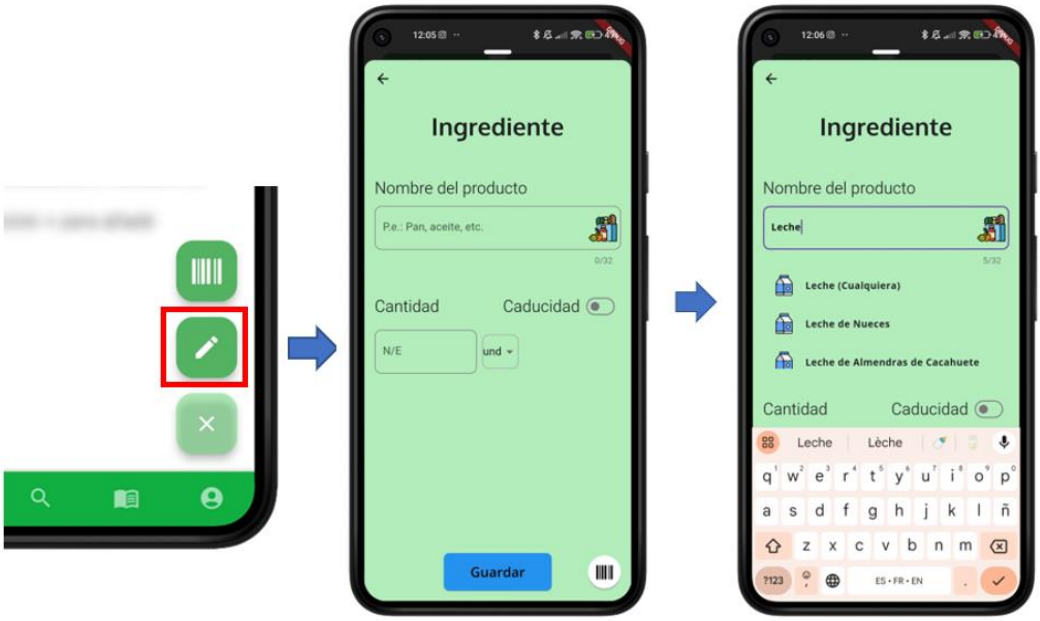

Ilustración 5.7: Botón para añadir productos manualmente

Al pulsar este botón se abrirá una pantalla para introducir la información del producto que deseamos agregar. Aquí se introduce tanta información como se desea y para guardar el producto se pulsa sobre el botón de la parte inferior con el texto “Guardar”. Durante la adición manual de productos se puede introducir el nombre del producto, sobre el que se mostrará una lista de sugerencias y se seleccionará un producto pulsando sobre él. Además habrá campos para especificar cantidad, unidad y fecha de caducidad.

Una vez añadidos los productos, estos podrán editarse, moverse de lista o eliminarse. Para editarse, cada producto cuenta con un botón en la parte izquierda con un icono de un lápiz. Al pulsar sobre este se abrirá la anterior pantalla de adición manual de productos pero esta vez con la información del producto seleccionado. Para editar la información del producto basta con modificar los campos de esta pantalla y darle al botón de guardar cuando se termine para preservar los cambios.

Para mover los productos de lista o eliminarlos se realizarán movimientos de deslizamiento. Cada producto en cada lista es deslizable y en función de hacia dónde se seleccione se realizará una acción u otra.

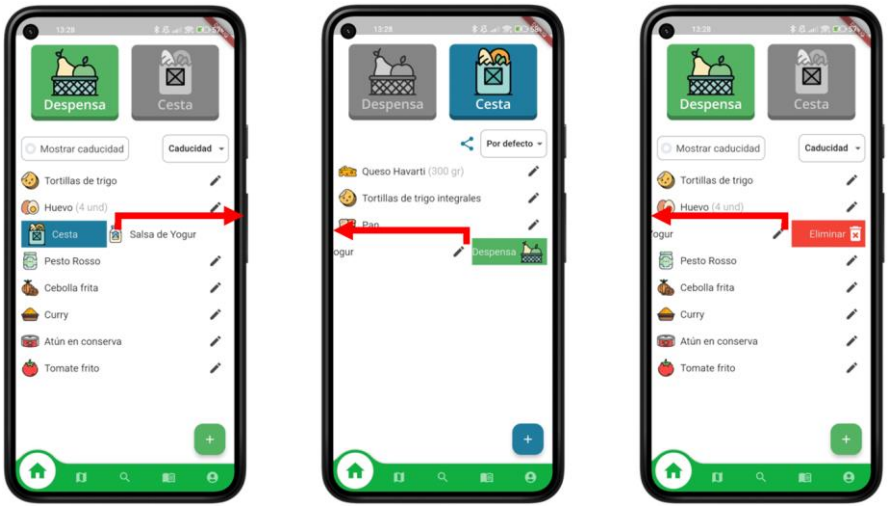

Ilustración 5.8: Deslizar productos para mover o eliminar

Cuando un producto se encuentre en la despensa (izquierda) podrá moverse a el producto deslizándolo hacía la despensa (derecha). Cuando se encuentre en la cesta (derecha) podrá moverlo hacia la otra lista deslizándolo hacia la despensa (izquierda). Si se desliza hacía un lugar dónde no haya lista se eliminará. Para verificar la acción que se está realizando, cuando se deslice cada producto se podrá observar un texto, icono y color que indique hacía dónde se está moviendo el producto.

5.3.3 Escaneo de alimentos y obtención de información nutricional

Para añadir productos mediante el escáner de códigos de barras, desde la sección de usuario habrá que pulsar el botón de añadir producto tal y como se describe en el apartado anterior. En lugar de usar el botón para añadir el producto manualmente se pulsará la opción para añadirlo mediante el escáner, la cual se encuentra ilustrada con un icono de un código de barras tal y cómo se puede observar en la imagen inferior.

Ilustración 5.9: Botones para acceder a la pantalla de escaneo y guardar un producto

Para escanear un producto bastará con apuntar la cámara hacia cualquier código de barras y ahí se procederá a obtener su información. Cuando se obtenga la información del producto se mostrará mediante una ventana emergente que contiene un marcado botón para guardar el producto escaneado en su lista.

Cabe la posibilidad de que si el usuario no disponga buena de conexión a internet o bien el servidor utilizado para hacer la consulta esté saturado, la consulta de información tarde tiempo de más. En este caso se informa al usuario mediante una pantalla de carga y un mensaje.

Una vez el producto esté en nuestra lista, para consultar información nutricional más detallada bastará con pulsar sobre este para que se abra una venta emergente con toda la información.

5.3.4 Gestión de usuarios

Una vez que la sesión esté iniciada en la aplicación se podrán realizar distintas acciones sobre la cuenta de usuario como modificar parámetros de la cuenta, cerrar sesión o borrarla. Para realizar cualquiera de estas acciones primero deberá dirigirse a la sección de usuario en la aplicación pulsando el quinto botón del menú inferior como se puede observar en la siguiente ilustración.

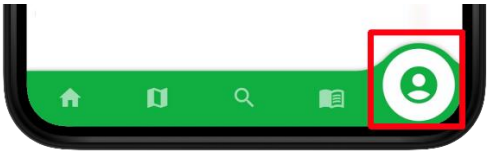

Ilustración 5.10: Botón para acceder a la sección de usuario

Para realizar cambios en el perfil de usuario o cerrar sesión encontraremos las siguientes opciones.

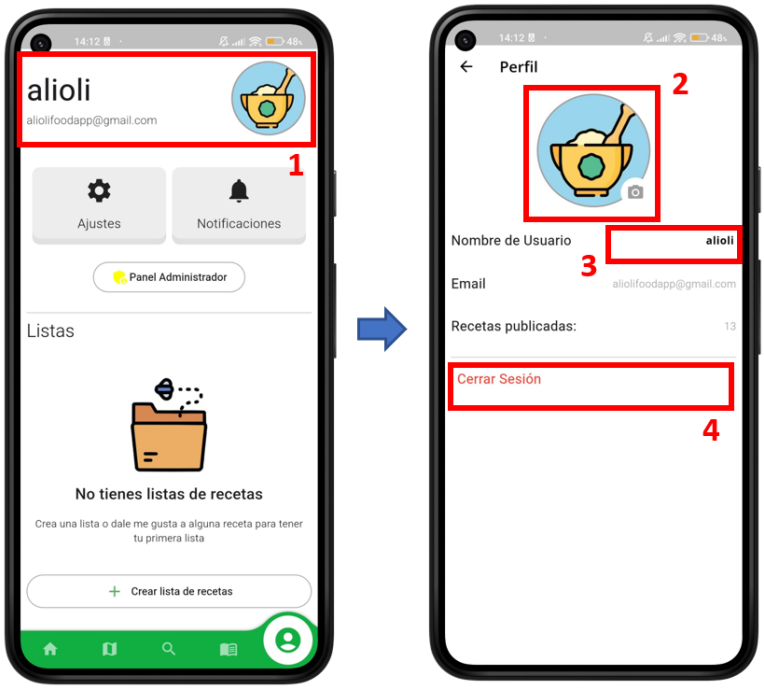

Ilustración 5.11: Opciones para modificar el usuario y cerrar sesión

En la ilustración anterior encontramos numeradas las siguientes opciones:

- 1: Acceder a los ajustes del perfil desde la sección de usuario.
- 2: **Cambiar fotografía** de la cuenta.
- 3: Campo de texto para **modificar el nombre de usuario**.
- 4: Botón para **cerrar sesión**.

Los cambios realizados sobre la sesión de usuario se realizarán también sobre los datos almacenados en el servidor, por lo que hará falta conexión a internet para realizarlos.

Para **borrar la cuenta** tendremos que acceder a los ajustes, ahí pulsar el botón de borrar cuenta, se abrirá una ventana emergente para introducir la contraseña de la cuenta, necesaria para verificar la identidad del usuario y poder realizar la acción.

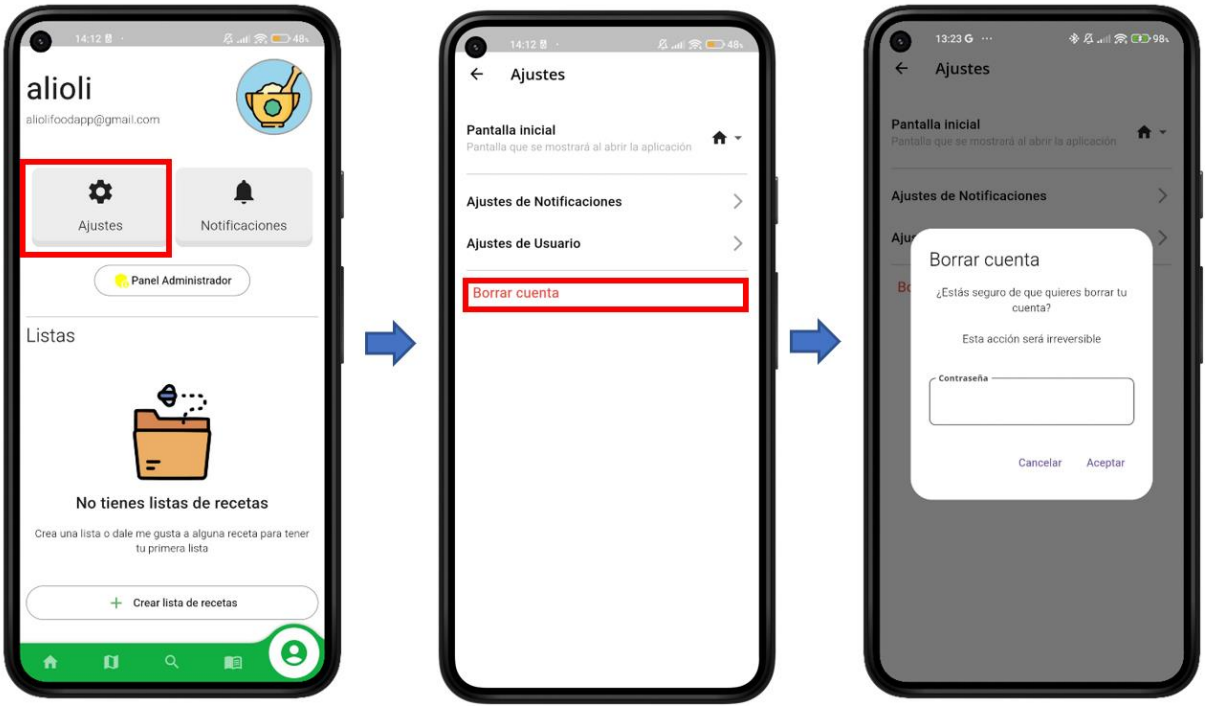

Ilustración 5.12: Proceso a seguir para borrar la cuenta

Una vez borrada la cuenta se borrarán todos sus datos asociados tanto de la memoria del dispositivo como del servidor y se redirigirá al usuario a la pantalla de inicio de sesión o registro.

5.3.5 Programar notificaciones de caducidad

Para activar las notificaciones hay que dirigirse a la sección de usuario como se ha explicado en el apartado anterior y se puede ver en la ilustración 5.10. Desde esta sección encontraremos un botón especialmente dedicado para acceder a los ajustes de notificaciones. Aunque también se puede acceder dentro de los propios ajustes.

Cuando se accede a los ajustes de notificaciones encontramos múltiples opciones, como se puede observar en la siguiente ilustración.

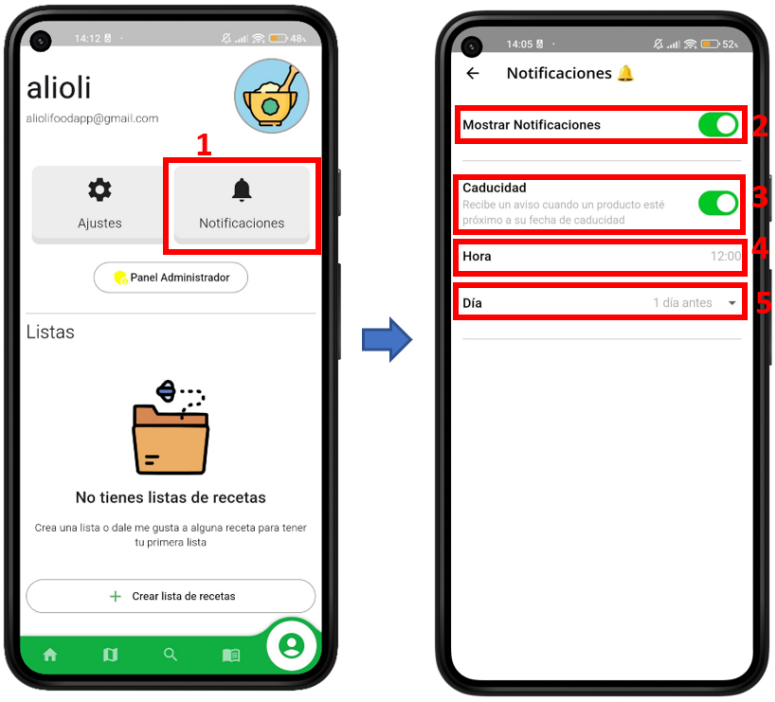

Ilustración 5.13: Botones para programar notificaciones

En la ilustración anterior encontramos numeradas las siguientes opciones:

- 1: Acceder a los ajustes de notificaciones.
- 2: Activar notificaciones generales.
- 3: Activar notificaciones específicas para la caducidad de los alimentos.
- 4: Seleccionar hora para recibir la notificación.
- 5: Seleccionar días de antelación con los que se quiere ser notificado.

Si no se otorgaron los permisos de notificación que se solicitaron durante la primera apertura de la aplicación, cuando se intente activar el botón de mostrar notificaciones se volverán a solicitar o bien se redirigirá hacia la pantalla de ajustes de la aplicación del sistema operativo para activar los permisos manualmente, indicando al usuario de esto.

Una vez activadas las notificaciones cuando llegue la hora de notificación si hay productos cuya fecha de caducidad coincida con los días de antelación programados se recibirá un aviso con el nombre del producto a caducar.

5.3.6 Crear y publicar una receta

La creación de recetas para publicarlas así como la gestión de las recetas subidas o en borradores se realiza mediante la sección de “Mis recetas”. Se accede a esta sección desde el cuarto botón del menú, visible en la ilustración inferior.

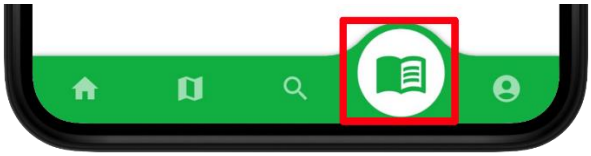

Ilustración 5.14: Botón para acceder a la sección Mis Recetas

Dentro de esta sección encontramos las opciones disponibles en la siguiente ilustración.

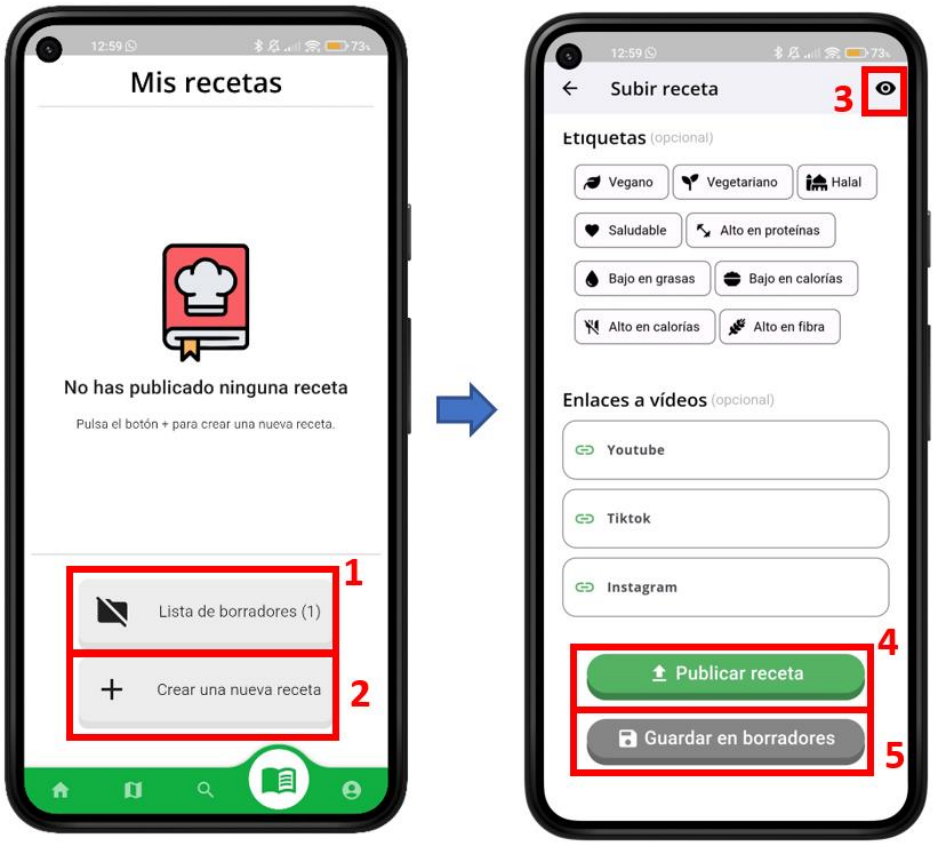

Ilustración 5.15: Botones para la creación, subida y gestión de recetas

En la ilustración anterior encontramos numeradas las siguientes opciones:

- 1: Acceder a la lista de borradores. Este botón solo será visible si existen borradores creados.
- 2: Acceder a la pantalla de creación de recetas.

- 3: Previsualizar una receta con los datos introducidos hasta ese momento.
- 4: Publicar una receta en la plataforma.
- 5: Guardar una receta en la lista de borradores.

Una vez publicada la receta esta no podrá editarse ni sufrir cambios más allá de ser eliminada si se solicita. Durante la creación de la receta podrá guardarse el estado de creación en un borrador, de forma que se pueda acceder a su estado previo en cualquier momento o bien borrar el borrador.

Ilustración 5.16: Lista de recetas publicadas

Las recetas publicadas podrán visualizarse desde la propia sección de Mis recetas como se puede observar en la ilustración anterior.

5.3.7 Buscar recetas

Para buscar recetas existen dos secciones en la aplicación, una para hacer búsquedas específicas de recetas por ingredientes o nombre y otra sección para buscar recetas en función de distintas categorías. **Para usar el modo principal de búsqueda mediante ingredientes y nombre**, se accede a la sección de búsqueda, al que se accede pulsando el tercer botón del menú inferior.

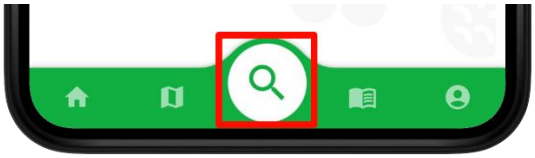

Ilustración 5.17: Botón para acceder a la sección de búsqueda

Esta sección estará compuesta de una pantalla para buscar por ingredientes y otra para buscar por nombre de receta. A continuación se presenta una ilustración con indicaciones de como navegar entre estas pantallas y realizar las búsquedas.

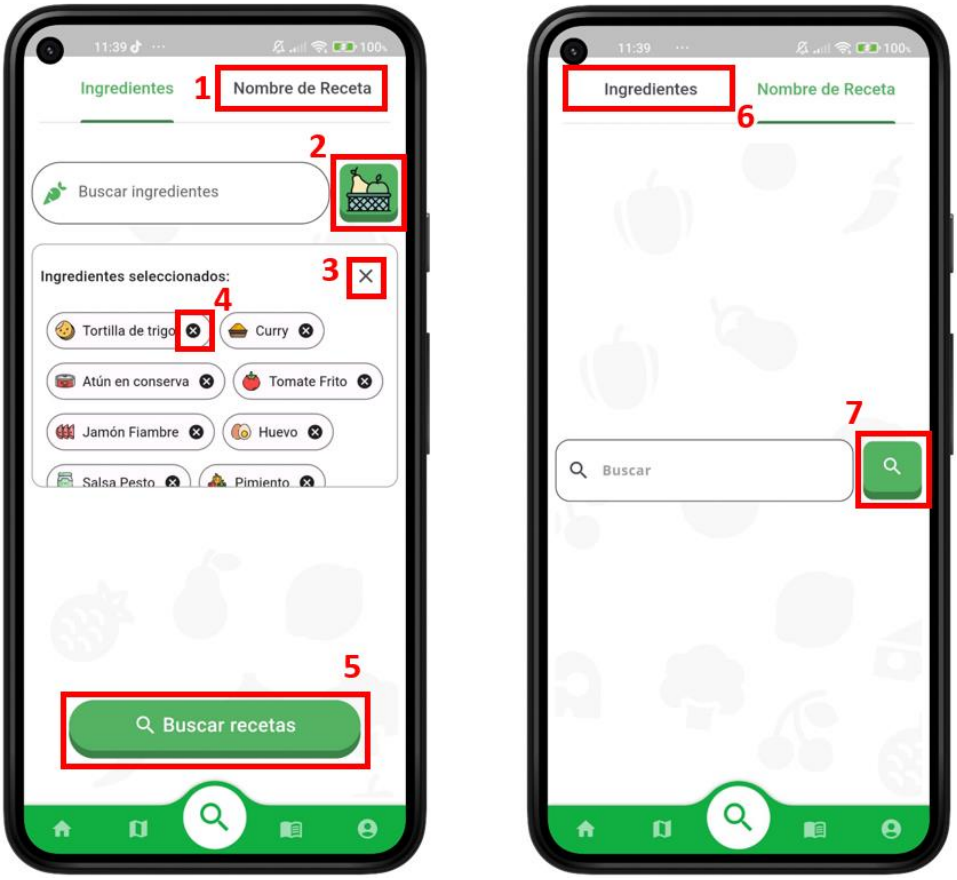

Ilustración 5.18: Botones para la búsqueda de recetas

En la ilustración anterior encontramos numeradas las siguientes opciones:

- 1: Cambiar a modo de búsqueda por nombre de receta.
- 2: Incluir ingredientes de la despensa en la búsqueda.
- 3: Borrar todos los ingredientes seleccionados.
- 4: Borrar un ingrediente seleccionado en particular.
- 5: Buscar recetas con los ingredientes seleccionados.

- 6: Cambiar a modo de búsqueda por ingredientes
- 7: Buscar recetas con el nombre introducido.

Por otra parte, para buscar recetas clasificadas en distintas categorías accederemos al segundo botón del menú inferior como se puede ver en la ilustración inferior. Desde esta sección encontraremos un botón destinado para cada categoría. Bastará con pulsar sobre cada categoría para buscar las recetas que pertenezcan a ella.

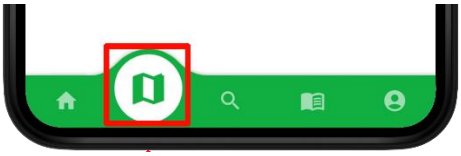

Ilustración 5.19: Botón para acceder a la sección categorías

Una vez realizada la búsqueda se accederá a una pantalla dónde se mostrarán los resultados de búsqueda junto con algunas opciones para **aplicar filtros**.

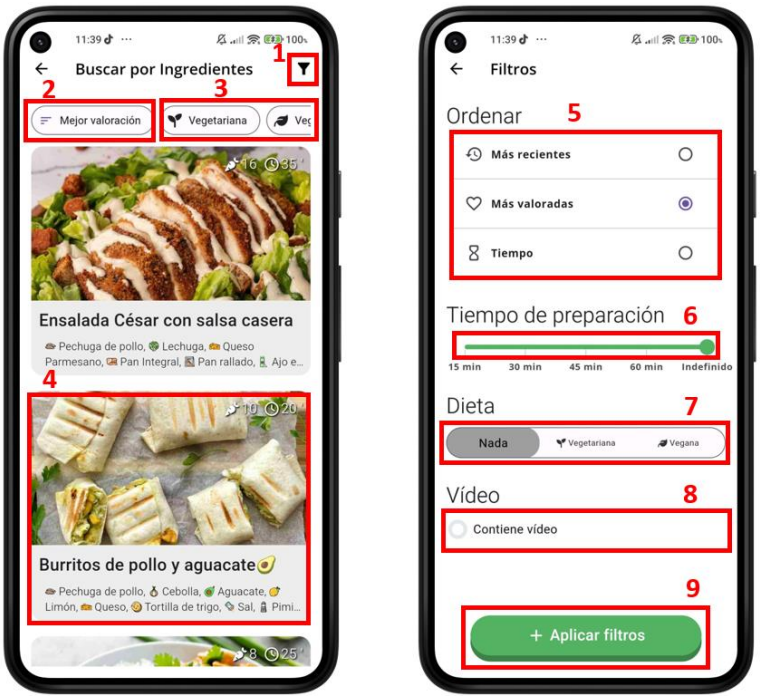

Ilustración 5.20: Botones para filtrar y ordenar resultados de búsqueda

En la ilustración anterior encontramos numeradas las siguientes opciones:

- 1: Acceder a la pantalla de filtros avanzados.
- 2: Cambiar modo de orden de las recetas.

- 3: Activar o desactivar filtros rápidos.
- 4: Abrir una receta en particular.
- 5: Modo de orden de los resultados.
- 6: Establecer un tiempo máximo de preparación.
- 7: Establecer tipo de dieta para las recetas.
- 8: Filtrar por recetas que contengan vídeo.
- 9: Aplicar filtros seleccionados

5.3.8 Crear y gestionar listas de recetas

Las listas de recetas creadas podrán ser visualizadas y gestionadas desde la sección de usuario, para acceder a esta sección se utiliza el quinto del botón del menú inferior como se ha explicado en los apartados anterior y visto en la ilustración 5.10.

En esta sección encontraremos tanto la lista de recetas formada por las recetas a las que se le hayan dado “me gusta”, como las listas de recetas propias que haya creado el usuario. Además podrán también podrán **crearse** y **eliminarse**. A continuación se mostrarán las distintas opciones.

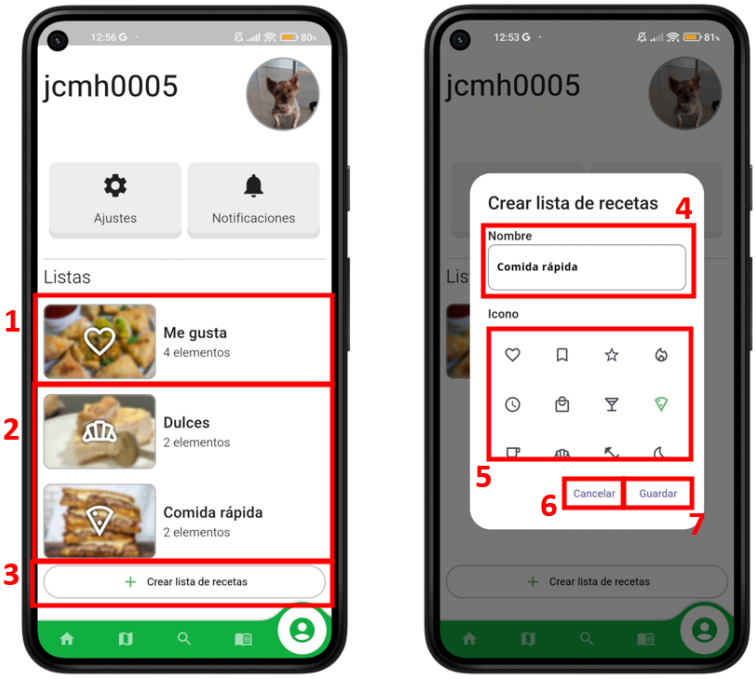

Ilustración 5.21: Botones para gestionar listas de recetas

En la ilustración anterior encontramos numeradas las siguientes opciones:

- 1: Lista de recetas referente a las recetas que se les haya dado “me gusta”.
- 2: Listas de recetas creadas por el usuario.
- 3: Botón para abrir ventana de creación de listas.
- 4: Introducción del nombre de la lista.
- 5: Selección de icono para la lista.
- 6: Cancelar lista.
- 7: Guardar lista.

Para borrar las listas habría que mantener pulsada la lista del listado que deseemos borrar y se abrirá una ventana para confirmar si se desea eliminar.

Para **añadir** o **eliminar una receta** en concreto a cada lista se hace desde la propia vista de la receta, en ella encontraremos un botón que abrirá un desplegable para seleccionar las listas en las que queremos guardar la receta, además de contar con otro botón para crear una lista de recetas desde ahí mismo

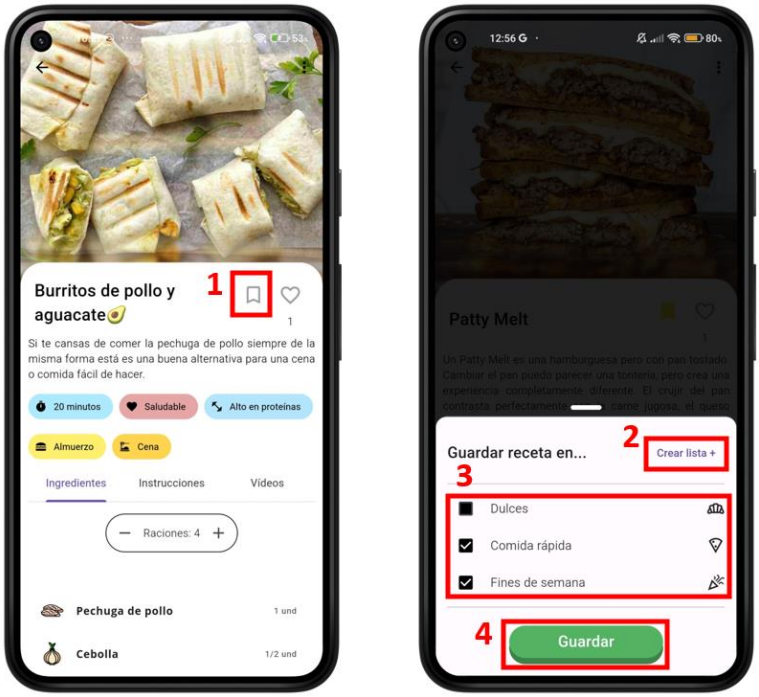

Ilustración 5.22: Botones para guardar elementos en listas

En la ilustración anterior encontramos numeradas las siguientes opciones:

- 1: Abrir desplegable de selección de listas.
- 2: Crear lista de recetas.
- 3: Marcar o desmarcar listas dónde guardar la receta.
- 4: Guardar cambios.

Cabe destacar que tanto para las acciones de crear o borrar una lista como del guardado de recetas en las mismas, se necesitará de una conexión a internet ya que se necesita actualizar y guardar los cambios en el servidor.

6 DEFINICIONES Y ABREVIATURAS

- **API:** Application Programming Interface (Interfaz de programación de aplicaciones)**:** Conjunto de herramientas y definiciones que permite a las aplicaciones comunicarse entre sí.
- **SDK:** Software Development Kit (Kit de desarrollo de software)**:** Conjunto de herramientas, bibliotecas y documentación para desarrollar aplicaciones para un sistema específico.
- **Framework**: Estructura de soporte reutilizable sobre la cual se puede desarrollar software de manera más eficiente.
- **Flutter**: Kit de desarrollo de software de código abierto de Google para crear aplicaciones multiplataforma desde una única base de código.
- **Widget**: Los widgets en Flutter son componentes reutilizables de la interfaz de usuario. Son los bloques básicos para construir aplicaciones, desde elementos simples como textos y botones hasta estructuras complejas como listas y formularios. Cada elemento visual en una app de Flutter es un widget, y estos se pueden combinar y anidar para crear interfaces más elaboradas.
- **Hot Reload**: Función que permite ver los cambios realizados en el código de una aplicación en tiempo real sin necesidad de reiniciar la aplicación.
- **Dart**: Lenguaje de programación desarrollado por Google, utilizado principalmente para desarrollar aplicaciones en Flutter.
- **Backend**: Parte de una aplicación que maneja la lógica del servidor, bases de datos y la gestión de usuarios, que no es visible para los usuarios finales.
- **Firebase**: Plataforma de Google para el desarrollo de aplicaciones móviles y web, que ofrece servicios como bases de datos, autenticación, y hosting.

- **Base de datos NoSQL:** Tipo de base de datos que permite almacenar y recuperar datos no estructurados y semi-estructurados de manera flexible, diferente a las bases de datos relacionales.
- **HTTPS:** Hypertext Transfer Protocol Secure (Protocolo seguro de transferencia de hipertexto) es una versión segura del HTTP, que utiliza cifrado SSL/TLS para proteger la integridad y la privacidad de los datos transferidos entre el dispositivo y el servidor.
- **IDE:** Un IDE (Entorno de Desarrollo Integrado) es un software que proporciona herramientas integrales para facilitar el desarrollo de aplicaciones, incluyendo editor de código, depurador y compilador.
- **SQLite:** Es un sistema de gestión de bases de datos relacional ligero, autónomo y sin servidor. Se integra directamente en las aplicaciones y no requiere configuración.
- **Patrón de Diseño**: Solución reutilizable y probada para un problema común en el desarrollo de software. Sirve como guía para resolver problemas de diseño en diferentes contextos, facilitando la creación de código más eficiente, mantenible y escalable.
- **DAO**: DAO (Data Access Object) es un patrón de diseño que proporciona una abstracción para las operaciones de acceso a la base de datos.
- **CRUD**: Acrónimo de Create, Read, Update, Delete (Crear, Leer, Actualizar, Eliminar). Representa las cuatro operaciones básicas de persistencia en bases de datos.
- **NutriScore**: Sistema de etiquetado nutricional que califica los alimentos de A a E, basándose en su composición nutricional, para ayudar a los consumidores a tomar decisiones alimentarias más saludables.
- **SVG**: Formato de imagen basado en XML para describir gráficos vectoriales bidimensionales. Permite crear imágenes que pueden escalarse a cualquier tamaño sin pérdida de calidad, y es ampliamente utilizado para gráficos, íconos, diagramas y animaciones.

- **Sprint**: Periodos cortos y definidos de tiempo en los que se debe completar una cierta cantidad de trabajo en un proyecto Scrum.
- **Diagrama de Gantt**: Herramienta de gestión de proyectos que muestra el cronograma de las tareas.
- **Hash**: Función que convierte datos en una cadena de longitud fija, usada para verificación y seguridad.

7 BIBLIOGRAFÍA

[1] E. Pérez, «Xataka,» 23 Septiembre 2019. [En línea]. Available: https://www.xataka.com/aplicaciones/yuka-que-se-basan-que-criterios-siguen-recomendaciones-aplicacion-nutricional-moda.

[2] S. Stockdale, «Exantediet,» 9 Julio 2021. [En línea]. Available: https://www.exantediet.com/blog/long-read/the-social-food-guide/.

[3] L. S. Vailshery, «Statista,» 3 Junio 2024. [En línea]. Available: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/.

[4] D. Dziurzyński, «applover,» 24 Noviembre 2023. [En línea]. Available: https://applover.com/blog/flutter-in-2024-maximizing-roi-in-cross-platform-app-development/.

[5] Lucidchart, «Lucidchart,» 30 Septiembre 2020. [En línea]. Available: https://www.lucidchart.com/blog/scrum-for-one.

[6] Google, «Firebase Documentation,» Google, [En línea]. Available: https://firebase.google.com/docs/firestore. [Último acceso: 2024].

[7] «State management - Flutter Documentation,» Google LLC, 2023. [En línea]. Available: https://docs.flutter.dev/data-and-backend/state-mgmt/simple.

[8] G. LLC, «ChangeNotifierProvider - Flutter Documentation,» Google LLC, 2023. [En línea]. Available: https://pub.dev/documentation/provider/latest/provider/ChangeNotifierProvider-class.html.

[9] nguyenvanduocit, «GitHub - Simple 3D button for Flutter,» [En línea]. Available: https://github.com/nguyenvanduocit/button3d.

[10] «Pub.dev,» Google LLC, Junio 2024. [En línea]. Available: https://pub.dev/packages/flutter_expandable_fab.

[11] «Pub.dev,» Google LLC, [En línea]. Available: https://pub.dev/packages/modal_bottom_sheet.

[12] «pub.dev,» Google LLC, [En línea]. Available: https://pub.dev/packages/share_plus.

[13] «pub.dev,» Google LLC, 2024. [En línea]. Available: https://pub.dev/packages/flutter_svg.

[14] «Flaticon,» [En línea]. Available: https://www.flaticon.es/.

[15] P. Brans, «What is a fuzzy search?,» TechTarget, Agosto 2022. [En línea]. Available: https://www.techtarget.com/whatis/definition/fuzzy-search.

[16] «pub.dev,» Google LLC, 2023. [En línea]. Available: https://pub.dev/packages/fuzzy.

[17] «pub.dev,» Google LLC, 2024. [En línea]. Available: https://pub.dev/packages/ai_barcode_scanner.

[18] «Open Food Facts,» Open Food Facts, [En línea]. Available: https://es.openfoodfacts.org/.

[19] OMS, «World Health Organization,» World Health Organization, 29 Abril 2020. [En línea]. Available: https://www.who.int/news-room/fact-sheets/detail/healthy-diet.

[20] U. F. a. D. Administration, «U.S. Food and Drug Administration,» U.S. Department of Health and Human Services, 14 Marzo 2024. [En línea]. Available: https://www.fda.gov/food/food-labeling-nutrition.

[21] «EUR-Lex,» Oficina de Publicaciones de la Unión Europea, [En línea]. Available: https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX%3A32011R1169.

[22] C. Vercelletto, «Livestrong,» 4 Diciembre 2023. [En línea]. Available: https://www.livestrong.com/article/288657-the-recommended-daily-intake-of-calories-carbs-fat-sodium-protein/.

[23] «pub.dev,» [En línea]. Available: https://pub.dev/packages/flutter_local_notifications.

[24] A. Pamnani, «Running Background Tasks in Flutter with WorkManager,» Medium, 3 Septiembre 2023. [En línea]. Available: https://aakashpp.medium.com/running-background-tasks-in-flutter-with-workmanager-fdb81e8244b1.

[25] «pub.dev,» [En línea]. Available: https://pub.dev/packages/android_alarm_manager_plus.

[26] «pub.dev,» [En línea]. Available: https://pub.dev/packages/shared_preferences.

[27] «pub.dev,» [En línea]. Available: https://pub.dev/packages/permission_handler.

[28] «Firebase Authentication,» Google, [En línea]. Available: https://firebase.google.com/docs/auth?hl=es-419.

[29] «pub.dev,» [En línea]. Available: https://pub.dev/packages/image_picker.

[30] «Hive Documentation,» [En línea]. Available: https://docs.hivedb.dev.

[31] B. K. Jena, «simplilearn,» 29 Agosto 2023. [En línea]. Available: https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm.

[32] «pub.dev,» [En línea]. Available: https://pub.dev/packages/crypt.

[33] [En línea]. Available: https://www.myfooddata.com/.