Desarrollo de un sistema de recomendación de menús de comidas para dietas.

TRABAJO DE FIN DE GRADO

IVÁN LEDESMA CASADO y SANDRA RAMOS RAMOS

Director

Antonio Sarasa Cabezuelo

Doble Grado en Ingeniería Informática y Administración y Dirección de Empresas

Facultad de Informática

Universidad Complutense de Madrid

Junio 2023

Development of a food menu recommendation system for diets.

FINAL DEGREE PROJECT

IVÁN LEDESMA CASADO y SANDRA RAMOS RAMOS

Director

Antonio Sarasa Cabezuelo

Double Degree in Computer Engineering and Business Administration and Management

Faculty of Informatics

Complutense University of Madrid

June 2023

Resumen

El desarrollo de este trabajo se encuentra enfocado en el desarrollo de una aplicación de comidas. Facilitar un elemento tan cotidiano como la gestión de comidas a los usuarios resulta elemental y de gran interés fomentando la descarga de la aplicación y su recomendación a personas cercanas.

A través de la aplicación el usuario podrá visualizar las recetas que puede preparar en base a los alimentos que se encuentran en su despensa. En consecuencia, se proporciona la gestión de una despensa virtual en base a los alimentos que el usuario indica que posee en su domicilio. Derivada de su despensa virtual, el usuario tendrá acceso a una lista de la compra que le permitirá administrar sus próximas compras, así como visualizar fácilmente sus gastos en compras anteriores. Además, para aquellos usuarios más "atrevidos" se permite la creación de recetas que podrán ser visibles públicamente si así lo desean.

El sistema se basa en una aplicación móvil para dispositivos Android que implementa una base de datos para gestión de la información de los usuarios.

Palabras clave

Recetas, despensa virtual, registro de gastos, lista de la compra, menú, aplicación Android.

Abstract

The development of this work is focused on the development of a food application. Facilitating an element as daily as the management of meals to users is elementary and of great interest, encouraging the download of the application and its recommendation to people close to them.

Through the application, the user will be able to view the recipes that can be prepared based on the foods found in their pantry. Consequently, the management of a virtual pantry is provided based on the food that the user indicates that he has at home. Derived from his virtual pantry, the user will have access to a shopping list that will allow him to manage his upcoming purchases, as well as easily view his expenses on previous purchases. In addition, for those more "daring" users, the creation of recipes is allowed that may be publicly visible if they wish.

The system is based on a mobile application for Android devices that implements a database for managing user information.

Keywords

Recipes, virtual pantry, expense record, shopping list, menu, Android application.

Índice

Índice de figuras

1. Introducción

1.1. Motivación

La alimentación es un aspecto vital en la vida de las personas dado que es una necesidad básica de supervivencia incidiendo, por tanto, en la salud de estas. Una alimentación adecuada proporciona los nutrientes necesarios para el correcto funcionamiento del organismo, previniendo enfermedades y trastornos alimentarios.

La elección de los diversos alimentos y su consumo influyen en el estado de salud de las personas. Una dieta equilibrada debe incluir alimentos como frutas y verduras, grasas, proteínas y carbohidratos, en proporciones adecuadas. Además, es importante que los alimentos escogidos sean frescos y de calidad, limitando el consumo de alimentos procesados, ricos en grasas, azúcares, sal y aditivos.

Una dieta saludable tiene un impacto positivo a nivel emocional y mental, ya que una buena alimentación es capaz de conseguir una mejora en el estado de ánimo y reducir los riesgos de sufrir depresión o ansiedad entre otros trastornos.

La alimentación también tiene un impacto en el rendimiento físico e intelectual. La buena alimentación puede mejorar el rendimiento deportivo, la concentración y la memoria. Sin embargo, una mala alimentación es capaz de producir fatiga, falta de energía y dificultad para concentrarse.

En resumen, la alimentación es un aspecto esencial en la vida de las personas, ya que afecta a múltiples aspectos en la vida, por ello, es necesario prestar atención a la elección y el consumo de alimentos para garantizar una buena calidad de vida.

Debido a las razones que radican en la importancia de la alimentación, se ha llevado a cabo una aplicación para dispositivos Android con la que facilitar la gestión de comidas. Con soluciones entre las que se encuentra ofrecer recetas en función de los alimentos que disponga la persona en su despensa, en consecuencia, se facilita una gestión de la despensa. También, visualizar el gasto en alimentación, crear recetas de carácter público o privado y facilitar el proceso de la creación de la lista de la compra.

1.2. Objetivos

El desarrollo de la aplicación se centra en el principal objetivo de satisfacer las necesidades diarias de los usuarios facilitando su gestión tanto dentro como fuera de la cocina. La aplicación permitirá al usuario, en base a los elementos que se encuentran en su despensa, realizar recetas almacenadas en la base de datos.

Derivado de este objetivo surgen los siguientes objetivos más específicos:

- Desarrollar una funcionalidad que limite el acceso a la app únicamente a usuarios registrados, permitiendo a los usuarios registrarse en la aplicación.
- Creación de una funcionalidad que permita al usuario gestionar la información de su cuenta además de poder eliminarla.
- Implementación de una funcionalidad que permita a los administradores manejar, buscar, y eliminar si es necesario, a los usuarios registrados en la aplicación.
- Implantar una funcionalidad que facilite al usuario opciones para poder visualizar y cocinar recetas de su interés en base a los alimentos de su despensa.
- Creación de una funcionalidad que permita visualizar un historial de recetas ya preparadas anteriormente.
- Desarrollar una funcionalidad que proporciona las funcionalidades de crear, eliminar y modificar recetas únicamente pertenecientes al usuario.
- Implementación de una funcionalidad para poder visualizar y buscar recetas que se encuentran en la base de datos.
- Creación de una funcionalidad que facilite al usuario poder crear, modificar, eliminar y buscar alimentos en su despensa virtual.
- Implantar una funcionalidad que permita al usuario añadir un alimento a la lista de la compra al ser eliminado de la despensa virtual.

- Desarrollo de una funcionalidad para añadir, eliminar, modificar o buscar un alimento en la lista de la compra del usuario.
- Generación de una funcionalidad que ofrezca al usuario la opción de visualizar un historial de gastos de sus anteriores compras.

1.3. Metodología

El desarrollo del proyecto se ha realizado de acuerdo con un plan de proyecto y a una organización del trabajo que se explica a continuación.

1.3.1. Fases del proyecto

Para desarrollar el proyecto, se hizo una planificación que incluyó la investigación y toma de requisitos, desarrollo de cuentas de usuarios, desarrollo de gestor de despensa, desarrollo de gestor de recetas, desarrollo de gestor de lista de la compra, y en último lugar, desarrollo de la memoria. Además, en cada fase, se ha realizado un proceso de identificación y arreglo de soluciones.

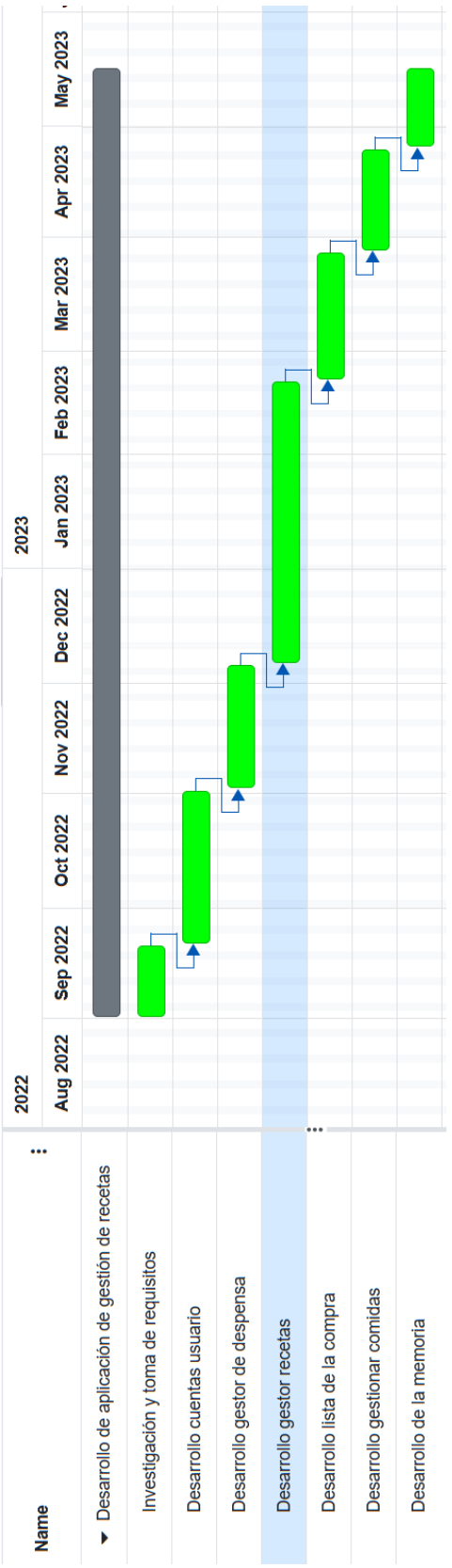


Figura 1.1: Diagrama de Gantt con las etapas del desarrollo del proyecto

Etapa	Duración
Investigación y toma de requisitos	2 Septiembre - 20 Septiembre
Desarrollo cuentas usuario	21 Septiembre - 14 Noviembre
Desarrollo gestor de despensa	15 Noviembre - 5 Diciembre
Desarrollo gestor de recetas	6 Diciembre - 20 Febrero
Desarrollo gestor lista de la compra	21 Febrero - 18 Abril
Desarrollo memoria	19 Abril - 16 Mayo

Tabla 1.1: Etapas del proyecto

En consiguiente se procede a explicar las distintas fases de desarrollo del proyecto.

Fase 1: Investigación y toma de requisitos

En esta primera fase se llevó a cabo la investigación de posibles necesidades relacionadas con la gestión de comidas que pudieran tener las personas. En consecuencia, se analizaron diversas aplicaciones alternativas que ofrecieran una visión general de cómo sería el proyecto una vez desarrollado. Se llevó a cabo un “brainstorming” y recopilando las características más interesantes de cada aplicación más aquellas que parecieran interesantes se realizaron los requisitos necesarios para la aplicación.

Fase 2: Desarrollo cuentas usuario

En esta segunda fase se llevó a cabo el desarrollo de la aplicación móvil destinada a gestionar las cuentas de usuario, abarcando la creación, la edición, el login y el borrado de estos. Para ello, se ha llevado a cabo un desarrollo destinado a gestionar los contenidos almacenados en la base de datos

pertenecientes a los usuarios. Esta fase se ha desarrollado por la necesidad de gestionar los datos de los usuarios, de cara a personalizar la aplicación según características individuales.

Fase 3: Desarrollo gestor de despensa

Durante el desarrollo de esta fase de la aplicación se llevó a cabo la creación de una despensa virtual disponible para los usuarios registrados en la aplicación. Para el desarrollo del sistema se tuvo en cuenta las necesidades del usuario para poder gestionar los alimentos que posee en su despensa. Por ello se pone a disposición del usuario una interfaz sencilla que le permita gestionar de manera cómoda su despensa. El usuario podrá revisar, actualizar, modificar todo lo relacionado con su despensa.

Fase 4: Desarrollo gestor de recetas

En la cuarta fase se ha desarrollado un gestor de recetas, siendo parte de la funcionalidad central de la aplicación. En base al desarrollo de esta cuarta fase, se ha llevado a cabo una gestión de contenidos de la información relativa a las recetas disponibles en la aplicación. Además, esta funcionalidad permite crear, editar, compartir, buscar y eliminar recetas. Esta funcionalidad se implementa con la idea de generar ideas de diversas recetas a los usuarios, con la finalidad de que éstos puedan llevar una dieta variada.

Fase 5: Desarrollo gestor lista de la compra

Para el quinto apartado se ha desarrollado una funcionalidad centrada en la gestión de la lista de la compra. Se implementa para facilitar al usuario la acción de realizar la compra, pudiendo añadir, eliminar y editar los diversos alimentos a comprar. Además, si el usuario ha procedido a eliminar un producto de la despensa por agotamiento del producto, se facilita su adición a la lista de la compra. Además, se ha tenido que gestionar la información relativa a la gestión de la lista de la compra en una base de datos.

1.3.2. Organización del trabajo

Durante la realización del trabajo se ha llevado a cabo una metodología de trabajo incremental. Las fases mencionadas han sido iteraciones entre uno y dos meses. Tras cada fase, se ha realizado un periodo de pruebas de usuario y solución de errores. Se ha finalizado este proceso mediante una reunión con el objetivo de confirmar que el trabajo se había realizado.

El trabajo ha sido realizado por dos personas, haciendo uso de herramientas como GitHub [1], para llevar a cabo un sistema de control de versiones, además, de permitir tener el trabajo en ambos equipos. Por otro lado, se ha hecho uso de Discord [2], para llevar a cabo las reuniones acontecidas durante el desarrollo del trabajo.

1.4. Estructura de la memoria

La memoria de trabajo consta de varios capítulos donde se detallan los detalles relevantes a la realización del proyecto. La memoria se estructura en los siguientes apartados:

- Estado del arte: durante este capítulo se describen aquellas aplicaciones que resultaron interesantes debido a la estética y funcionalidad que soportaban, siendo de gran utilidad y referencia para la realización del proyecto.
- Tecnología empleada: en este apartado se tratan las principales y más importantes herramientas y tecnologías necesitadas para el desarrollo del proyecto.
- Especificación de requisitos: en función de los diferentes actores que intervienen en la aplicación se desarrollan los requisitos necesarios para los módulos de la aplicación: cuentas de usuario, despensa virtual, gestionar recetas, lista de la compra y gestionar comidas.
- Arquitectura de la aplicación: a lo largo de este capítulo se detalla la estructura por la que está compuesta la aplicación y el modelo de datos relativo al proyecto.
- Implementación y diseño de la aplicación: en este capítulo se trata y visualiza la implementación y el diseño del proyecto por el que se ha optado durante el proyecto.

- Conclusiones y trabajo futuro: para este capítulo se explican las posibles líneas de desarrollo futuro y se hace una pequeña conclusión del fundamento de la aplicación.

- Aportes individuales: en base al trabajo realizado por cada uno de los integrantes del grupo se especifica en detalle el trabajo realizado por cada uno de los integrantes.

- Guía de uso: para finalizar la memoria se detalla un manual de uso para poder comprobar los resultados del proyecto así como entender mejor el funcionamiento de la aplicación.

Introduction

Motivation

Food is a vital aspect in people's lives since it is a basic need for survival, thus affecting their health. An adequate diet provides the necessary nutrients for the proper functioning of the body, preventing diseases and eating disorders.

The choice of various foods and their consumption influence the state of health of people. A balanced diet should include foods such as fruits and vegetables, fats, proteins, and carbohydrates, in proper proportions. In addition, it is important that the foods chosen are fresh and of good quality, limiting the consumption of processed foods, rich in fats, sugars, salt and additives.

A healthy diet has a positive impact on an emotional and mental level, since a good diet is capable of improving mood and reducing the risk of suffering from depression or anxiety, among other disorders.

Diet also has an impact on physical and intellectual performance. A good diet can improve sports performance, concentration and memory. However, a poor diet is capable of producing fatigue, lack of energy and difficulty concentrating.

In summary, food is an essential aspect in people's lives, since it affects multiple aspects of life, therefore, it is necessary to pay attention to the choice and consumption of food to guarantee a good quality of life.

Due to the reasons that lie in the importance of food, an application for Android devices has been developed to facilitate meal management. With solutions among which is offering recipes based on the food that the person has in their pantry, consequently, pantry management is facilitated. Also, visualize food spending, create public or private recipes and facilitate the process of creating the shopping list.

Objectives

The development of the application is focused on the main objective of satisfying the daily needs of users, facilitating their management both inside and outside the kitchen. The application will allow the user, based on the items found in his pantry, to make recipes stored in the database.

Derived from this objective, the following more specific objectives arise:

- Develop functionality that limits access to the app to registered users only, allowing users to register in the app.
- Creation of a functionality that allows the user to manage their account information as well as being able to delete it.
- Implementation of a functionality that allows administrators to manage, search, and delete if necessary, registered users in the application.
- Implement a functionality that provides the user with options to be able to view and cook recipes of their interest based on the food in their pantry.
- Creation of a functionality that allows viewing a history of previously prepared recipes.
- Develop a functionality that provides the functionalities of creating, deleting and modifying recipes uniquely belonging to the user.
- Implementation of a functionality to be able to view and search for recipes found in the database.
- Creation of a functionality that makes it easier for the user to create, modify, delete and search for food in their virtual pantry.
- Implement a functionality that allows the user to add a food to the shopping list when it is removed from the virtual pantry.
- Development of a functionality to add, delete, modify or search for a food in the user's shopping list.

- Generation of a functionality that offers the user the option of viewing a history of expenses of their previous purchases.

Metodology

The development of the project has been carried out in accordance with a project plan and a work organization that is explained below.

Project phases

To develop the project, a planning was made that included research and requirements, development of user accounts, pantry manager development, recipe manager development, shopping list manager development, and lastly, memory development. In addition, in each phase, a process of identifying and fixing solutions has been carried out.

Phase 1: Research and gathering of requirements

In this first phase, the investigation of possible needs related to the management of meals that people might have was carried out. Consequently, various alternative applications were analyzed to offer an overview of how the project would be once developed. A "brainstorming" was carried out and collecting the most interesting characteristics of each application plus those that seemed interesting, the necessary requirements for the application were made.

Phase 2: Development user accounts

In this second phase, the development of the mobile application for managing user accounts was carried out, covering the creation, edition, login and deletion of these. For this, a development has been carried out to manage the contents stored in the database belonging to the users. This phase has been developed due to the need to manage user data, in order to customize the application according to individual characteristics.

Phase 3: Pantry manager development

During the development of this phase of the application, the creation of a virtual pantry available to registered users of the application was carried out. For the development of the system, the needs of the user were taken into account to be able to manage the food they have in their pantry. For this reason, a simple interface is made available to the user that allows them to comfortably manage their pantry. The user will be able to review, update, modify everything related to his pantry.

Phase 4: Recipe manager development

In the fourth phase, a recipe manager has been developed, being part of the central functionality of the application. Based on the development of this fourth phase, a content management of the information related to the recipes available in the application has been carried out. In addition, this functionality allows you to create, edit, share, search and delete recipes. This functionality is implemented with the idea of generating ideas for various recipes for users, so that they can have a varied diet.

Phase 5: Development of the shopping list manager

For the fifth section, a functionality focused on the management of the shopping list has been developed. It is implemented to facilitate the user the action of making the purchase, being able to add, delete and edit the various foods to buy. In addition, if the user has proceeded to remove a product from the pantry due to exhaustion of the product, its addition to the shopping list is facilitated. In addition, it has had to manage the information related to the management of the shopping list in a database.

Work organization

During the realization of the work an incremental work methodology has been carried out. The mentioned phases have been iterations between one and two months. After each phase, a period of user testing and error solving has been carried out. This process has been finalized through a meeting with the aim of confirming that the work had been carried out.

The work has been carried out by two people, making use of tools such as GitHub [1], to carry out a version control system, as well as allowing the work to be carried out on both computers. On the other hand, Discord [2] has been used to carry out the meetings that occurred during the development of the work.

Memory structure

The working memory consists of several chapters where the details relevant to the realization of the project are detailed. The memory is structured in the following sections:

- State of the art: this chapter describes those applications that were interesting due to the aesthetics and functionality they supported, being very useful and a reference for carrying out the project.
- Technology used: this section deals with the main and most important tools and technologies needed for the development of the project.
- Specification of requirements: depending on the different actors involved in the application, the necessary requirements for the application modules are developed: user accounts, virtual pantry, manage recipes, shopping list and manage meals.
- Application architecture: throughout this chapter, the structure of the application and the data model related to the project are detailed.
- Implementation and design of the application: This chapter discusses and visualizes the implementation and design of the project that has been chosen during the project.

- Conclusions and future work: for this chapter the possible lines of future development are explained and a small conclusion is made on the foundation of the application.
- Individual contributions: based on the work carried out by each one of the members of the group, the work carried out by each one of the members is specified in detail.
- User guide: to finish the report, a user manual is detailed to be able to check the results of the project as well as to better understand how the application works.

2. Estado del arte

En este capítulo se desarrollan algunas aplicaciones con similitudes a la implementada en este proyecto.

2.1. Yuka

Yuka[3] es una aplicación disponible tanto para dispositivos iOS como Android. Yuka almacena una gran cantidad de datos para poder proporcionar las siguientes funciones:

- Evaluar la calidad de los productos alimentarios.
- Evaluar la calidad de los productos cosméticos.
- Recomendar mejores productos a los buscados.
- Mostrar una síntesis gráfica de los alimentos buscados en función de la calidad.

Todas estas funcionalidades se caracterizan por no tener influencias de marcas, no promocionar productos propios en la aplicación y gozar de una financiación limpia.

2.2. Cookpad

Cookpad[4] es tanto una aplicación móvil como web. Ofrece a los usuarios una plataforma para compartir recetas desde todas las partes del mundo. Se permite a los usuarios publicar sus propias recetas, buscar y guardar recetas de otros usuarios. Además, se podrá crear una colección personal de recetas.

Por otro lado, Cookpad permite la planificación de comidas, gestión de listas de la compra o compartir comentarios y fotos sobre las recetas.

2.3. TikTok

TikTok[5] es una aplicación basada en una red social con origen chino. Está disponible tanto para iOS como para Android. Permite realizar las siguientes funcionalidades:

- Compartir y visualizar vídeos de corta duración, además, de gestionar el perfil de los usuarios.
- Permite grabar y editar vídeos cortos en loop, añadiendo sonidos, filtros y efectos visuales.
- Permite estar en contacto con otros usuarios, ya sea por mensaje o por interacciones en las publicaciones.
- Permite realizar transmisiones en directo.

La UI, User Interface, de esta aplicación ha sido utilizada como idea principal para realizar este proyecto.

2.4. Spendee

Spendee[6] es una aplicación web y móvil que facilita a los usuarios la gestión financiera de sus presupuestos y gastos de una manera eficaz. A los usuarios que utilicen la aplicación se les permite gestionar sus datos diarios, establecer presupuestos mensuales, visualizar informes detallados de gatos y recibir alertas sobre posibles excesos de gatos.

Por otro lado, permite la sincronización con la cuenta bancaria para poder importar datos sobre las actividades de la cuenta de manera más cómoda. Posee una versión premium que aporta características adicionales, aunque únicamente complementan a la versión gratuita.

3. Tecnología empleada

En este apartado se describen las principales tecnologías y herramientas utilizadas durante el desarrollo del proyecto.

3.1. Android studio

Android Studio[7] es un entorno de desarrollo integrado (IDE). Android Studio se centra en el desarrollo de aplicaciones móviles Android. Es una aplicación gratuita de código abierto, desarrollada por Google. Proporciona un conjunto de recursos y herramientas para facilitar el trabajo de los desarrolladores.

Entre los recursos proporcionados, Android Studio, cuenta con un editor de código avanzado, herramientas de depuración, emuladores de dispositivos Android para probar las aplicaciones, integración con Git[11] para controlar las versiones, además, de una gran cantidad de plantillas y bibliotecas de código abierto.

Además, es compatible con lenguajes como Kotlin y Java[8], proporcionando flexibilidad a los desarrolladores para la elección del lenguaje.

3.2. Java

Java [8] es un lenguaje de programación orientado a objetos y de alto nivel. Se emplea para el desarrollo de aplicaciones y software. Java permite el desarrollo y ejecución de código en diversas plataformas sin tener que volver a recompilar el código. Debido a ello, su uso está muy extendido para el desarrollo de aplicaciones móviles, web, juegos y otros tipos de software.

Java ofrece una gran cantidad de bibliotecas, clases y métodos ya definidos. Gracias a esto, los programadores pueden llevar a cabo programas de manera más cómoda y rápida. Por otro lado, Java cuenta con un sistema de seguridad que protege el código volviendo a Java un lenguaje “seguro” frente a vulnerabilidades.

Cualquier plataforma que tenga una JVM (Java Virtual Machine) tiene la posibilidad de ejecutar código Java, volviéndolo un lenguaje de programación portable. En consecuencia, el código puede ser ejecutado en cualquier lugar y ser escrito una vez ahorrando esfuerzo y dinero.

3.3. XML

XML[9] es un lenguaje de marcado extensible. Permite almacenar y definir datos de forma compatible. Además, admite realizar intercambios de información entre sistemas de computación, bases de datos, sitios web, aplicaciones de terceros, etc.

XML es un lenguaje caracterizado por organizar los datos en elementos, utilizando etiquetas de inicio y de fin. Además, cada elemento dispone de atributos propios para su edición, como por ejemplo es el uso de la negrita.

Este lenguaje se utiliza principalmente en la industria de la tecnología y se puede utilizar para definir la estructura de datos de aplicaciones y sistemas. También, se usa para definir interfaces de usuario, tanto en aplicaciones móviles como en web, mediante archivos de diseño.

3.4. Firebase

Firebase[10] es una plataforma móvil y web propiedad de Google para desarrollar aplicaciones. Proporciona multitud de servicios en Cloud para construir, mejorar y escalar aplicaciones. Entre sus servicios se destacan:

- Autenticación de usuario: los usuarios se pueden autenticar a través de diferentes formas. Entre ellas se destaca el uso del correo electrónico, Google o Facebook.
- Almacenamiento en nube: ofreciendo a los desarrolladores guardar y recuperar datos de manera segura.
- Base de datos: funciona en tiempo real para permitir a los desarrolladores la sincronización de datos en tiempo real en cualquier instancia de la aplicación.
- Análisis: los desarrolladores pueden obtener información sobre el uso de la aplicación, acciones de los usuarios y rendimiento de la aplicación.

Firebase se puede integrar cómodamente con otras aplicaciones como Google Cloud o Google Analytics.

3.5. Git

Git[1] es un portal creado para alojar código, diseñado por Linus Torvalds y comprado por Microsoft en junio de 2018. Esta plataforma fue desarrollada con el objetivo de facilitar el desarrollo de código a los programadores, de tal forma que puedan subir el código de sus aplicaciones y herramientas. Además, permite la colaboración en el desarrollo del código almacenado.

Por tanto, Git es un sistema de control que permite comparar código de un archivo, visualizando las diversas versiones y restaurando las antiguas en caso necesario.

3.6. Charting

Charting[11] es una biblioteca de gráficos que se puede encontrar en github y está orientada a Android. Esta biblioteca ofrece una gran cantidad de gráficos con opción de personalización. Entre los gráficos más populares y comunes se encuentran los gráficos de líneas, de barras o de tarta. La biblioteca se mantiene por la aportación de “Phillpp Jahoda”, o “mikephill” en Github.

Dentro de la biblioteca se encuentran diversas clases que implementan diferentes tipos de gráficos. Estas clases permiten a los usuarios que incluyen la biblioteca desarrollar aplicaciones Android que muestran datos interactivos y personalizables.

4. Especificación de requisitos

Se han definido los siguientes tipos de usuario:

4.1. Actores

La aplicación desarrollada cuenta con los siguientes tipos de usuarios:

- Administrador: se identifica con el usuario encargado de administrar las cuentas de usuarios registrados en la aplicación móvil.

- Usuario registrado: este tipo de usuario podrá navegar libremente por la aplicación disfrutando de todos los privilegios. Podrán disponer de una serie de comidas adaptadas a sus despensas, pudiendo acceder a éstas de manera virtual encontrando los alimentos de los que disponen en casa, pudiendo así gestionar su propia lista de la compra.

- Usuario sin registrar: este tipo de usuario no podrá acceder a las funcionalidades que ofrece la aplicación, con la excepción de realizar su propio registro en la misma.

4.2. Módulos

4.2.1. Módulo cuentas de usuario

El Diagrama de Caso de Uso (DCU):

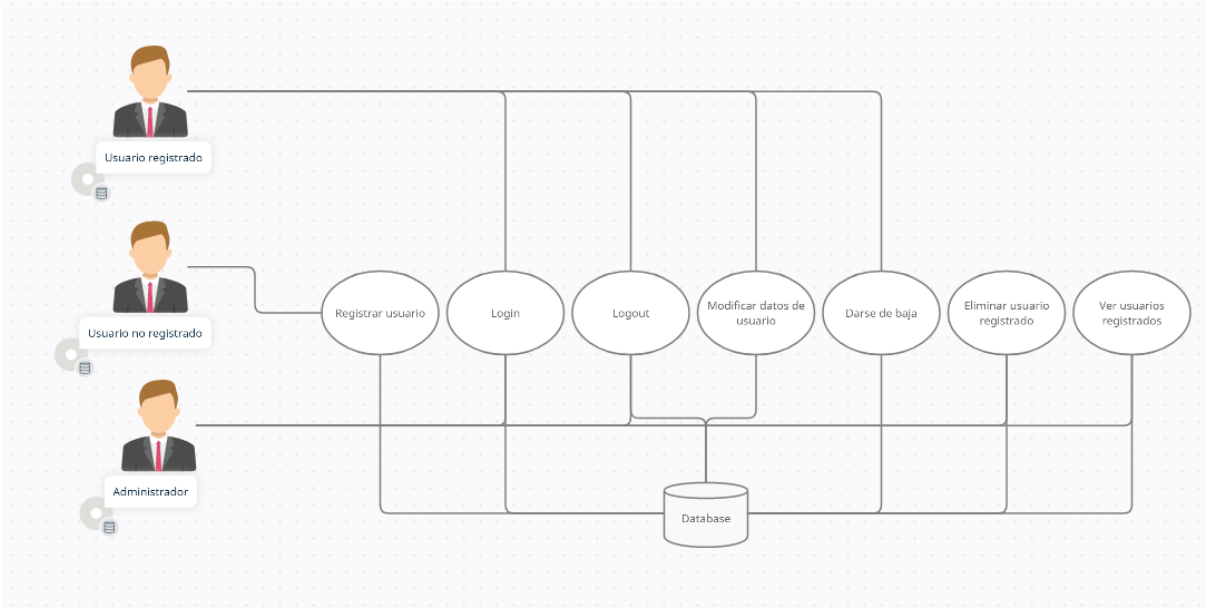


Figura 4.1: Diagrama de casos de uso del módulo cuentas de usuario

Requisito	**Registrar usuario**	
Identificador	1.1	
Precondición	NA	
Descripción	Los usuarios pueden crear una cuenta en la aplicación.	
Entrada	Nombre de usuario, sexo, fecha de nacimiento, correo electrónico y contraseña.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario inicia la aplicación donde el sistema le muestra la pantalla de login.
	2	El usuario pincha en “Registro”.
	3	El sistema muestra una pantalla en la que se indica toda la información a cumplimentar.
	4	El usuario rellena los campos pedidos y pincha en “Crear Cuenta”.
	5	El sistema valida los datos introducidos por el usuario.
	6	El sistema muestra la pantalla principal.
Postcondición	Se crea un nuevo usuario en el sistema.	
Excepciones	Paso	Acción
	4	Se muestra mensaje correspondiente a nombre de usuario, si ya existe otro igual.
	4	Se muestra mensaje correspondiente a contraseña si no cumple los requisitos de la creación de esta.
	4	Se muestra el mensaje correspondiente si algún dato no ha sido proporcionado por el usuario.
Comentarios	NA	
Actores	Usuario no registrado.	

Requisito	**Login**	
Identificador	1.2	
Precondición	El usuario tiene que estar registrado.	
Descripción	Los usuarios pueden loguearse en la aplicación (inicio de sesión).	
Entrada	Nombre de usuario y contraseña.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la página principal de la aplicación donde se pide la cumplimentación de los datos por parte del usuario.
	2	El usuario rellena los datos y hace click en “Login”.
	3	El sistema valida los datos introducidos por el usuario.
	4	El sistema muestra la pantalla principal.
Postcondición	El usuario inicia sesión con su cuenta.	
Excepciones	Paso	Acción
	3	Se muestra el mensaje correspondiente si algún dato requerido no ha sido rellenado.
	3	Se muestra el mensaje correspondiente si los datos no corresponden entre sí.
Comentarios	NA	
Actores	Usuario registrado	

Requisito	**Logout**	
Identificador	1.3	
Precondición	El usuario tiene que estar logueado.	
Descripción	Los usuarios pueden salir de la aplicación con el fin de iniciar una nueva sesión.	
Entrada	Nombre de usuario e ID usuario.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación.
	2	El usuario hace click en el botón de su “Perfil”.
	3	El sistema muestra al usuario la información sobre su perfil y el usuario hace click en el botón “Logout”.
	4	El sistema cierra la sesión y vuelve a la página de login.
Postcondición	Se ha cerrado la sesión del usuario.	
Excepciones	Paso	Acción
	4	Se muestra el mensaje correspondiente si no se ha podido cerrar la sesión.
Comentarios	NA	
Actores	Usuario registrado	

Requisito	**Modificar datos de usuario**	
Identificador	1.4	
Precondición	El usuario tiene que estar logueado.	
Descripción	Los usuarios pueden modificar datos sobre su perfil.	
Entrada	Nombre de usuario, ID usuario, contraseña, fecha de nacimiento y correo electrónico.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal.
	2	El usuario hace click en “Mi perfil”.
	3	El sistema muestra al usuario la información sobre su perfil, el usuario modifica los aspectos que considere y hace click en el botón “Guardar” para guardar los cambios.
	4	El sistema muestra un mensaje de confirmación de los cambios y el usuario hace click en ”Aceptar”.
	5	El sistema valida los datos introducidos por el usuario.
	6	El sistema muestra la pantalla perteneciente al perfil con los datos modificados.
Postcondición	Se han modificado los datos de usuario.	
Excepciones	Paso	Acción
	5	Se muestra el mensaje correspondiente si no se ha podido modificar algún dato del usuario.
Comentarios	NA	
Actores	Usuario registrado.	

Requisito	**Darse de baja**	
Identificador	1.5	
Precondición	El usuario tiene que estar logueado.	
Descripción	Los usuarios pueden darse de baja de la aplicación.	
Entrada	ID usuario.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal.
	2	El usuario hace click en “Mi perfil”.
	3	El sistema muestra al usuario la información sobre su perfil y el usuario hace click en “Eliminar cuenta”.
	4	El sistema muestra un mensaje de confirmación de eliminación de cuenta.
	5	El sistema elimina al usuario de la aplicación.
	6	El sistema muestra la pantalla para registrarse.
Postcondición	Se ha eliminado al usuario.	
Excepciones	Paso	Acción
	5	Se muestra el mensaje correspondiente si no se ha podido eliminar al usuario.
Comentarios	NA	
Actores	Usuario registrado.	

Requisito	**Eliminar usuario registrado**	
Identificador	1.6	
Precondición	El usuario debe estar registrado como administrador.	
Descripción	El administrador puede eliminar un usuario registrado de la aplicación.	
Entrada	ID usuario.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El administrador accede a la aplicación y el sistema le muestra la página principal.
	2	El sistema muestra el listado de usuarios registrados.
	3	El administrador hace click en el usuario a eliminar.
	4	El sistema muestra la información del usuario registrado.
	5	El administrador hace click en “Eliminar Usuario”.
	6	El sistema muestra un mensaje de confirmación.
	7	El administrador hace click en “Aceptar”.
	8	El sistema valida la eliminación del usuario.
	9	El sistema muestra el listado de usuarios actualizado.
Postcondición	Se ha eliminado al usuario.	
Excepciones	Paso	Acción
	2	El usuario a borrar no se encuentra dentro de la lista de usuarios registrados.
	8	Se muestra el mensaje correspondiente si no se ha podido eliminar al usuario.
Comentarios	NA	
Actores	Administrador.	

Requisito	**Ver usuarios registrados**	
Identificador	1.7	
Precondición	El usuario debe estar registrado como administrador.	
Descripción	El administrador puede ver los usuarios registrados en la aplicación.	
Entrada	Nombre de usuario.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El administrador accede a la aplicación y el sistema le muestra la página principal.
	2	El sistema muestra el listado de usuarios registrados.
	3	El administrador hace click sobre el usuario a visualizar.
	4	El sistema muestra una pantalla con toda la información disponible del usuario seleccionado.
Postcondición	NA	
Excepciones	Paso	Acción
	2	El usuario a visualizar no se encuentra dentro de la lista de usuarios registrados.
Comentarios	NA	
Actores	Administrador.	

4.2.2. Módulo despensa virtual

El Diagrama de Caso de Uso (DCU):

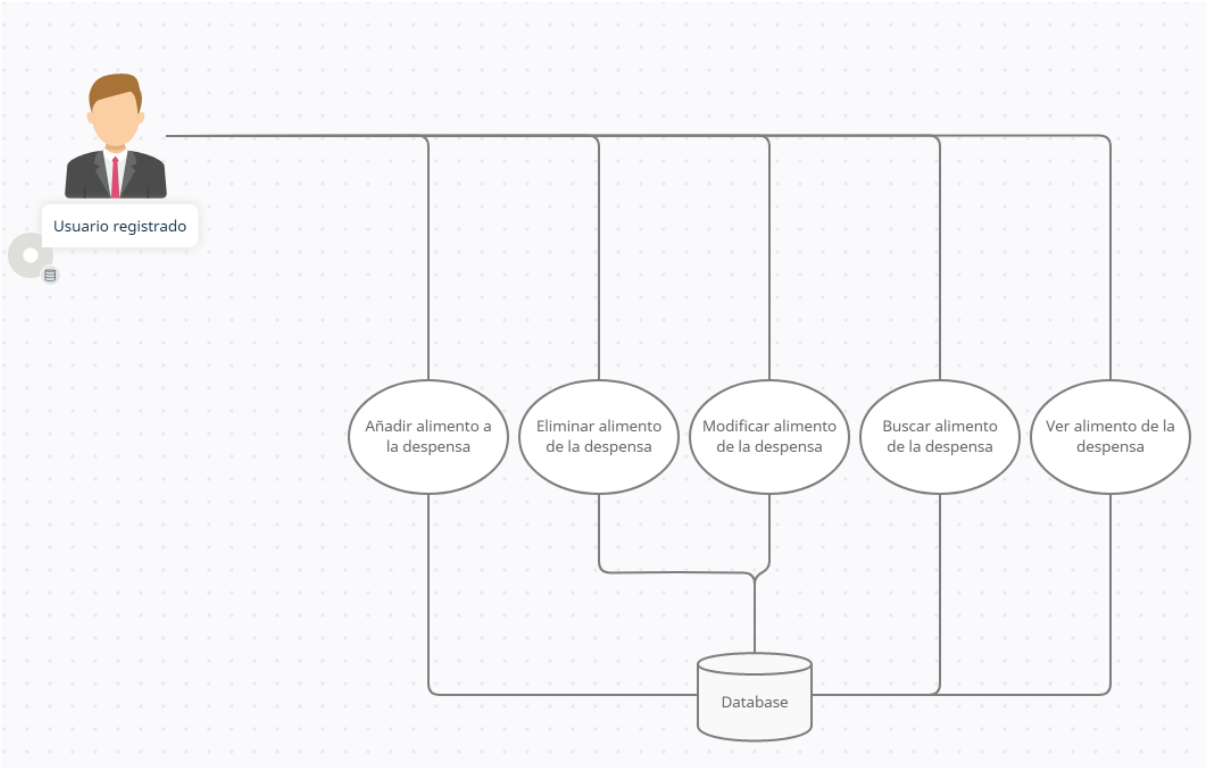


Figura 4.2: Diagrama de casos de uso del módulo despensa virtual

Requisito	**Añadir alimento a la despensa**	
Identificador	2.1	
Precondición	El usuario debe estar logueado.	
Descripción	Los usuarios pueden añadir alimentos de la aplicación a su despensa.	
Entrada	Nombre del alimento y cantidad.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal.
	2	El usuario hace click en “Mi despensa”.
	3	El sistema muestra el listado de alimentos que se encuentran en la despensa.
	4	El usuario hará click en “Añadir alimento”.
	5	El sistema le muestra una pantalla con los datos a introducir.
	6	El usuario rellena los campos y hace click en “Añadir”.
	7	El sistema muestra un mensaje de confirmación de los datos introducidos.
	8	El usuario hace click en “Aceptar”.
	9	El sistema valida los datos introducidos por el usuario.
	10	El sistema muestra al usuario la página de listado de alimentos de la despensa.
Postcondición	Se ha añadido un alimento a la despensa.	
Excepciones	Paso	Acción
	9	Se muestra el mensaje correspondiente si algún dato introducido no es el adecuado.
Comentarios	NA	
Actores	Usuario registrado.	

Requisito	**Eliminar alimento de la despensa**	
Identificador	2.2	
Precondición	El usuario debe estar logueado y tener al menos un alimento en la despensa.	
Descripción	Se procede a eliminar un alimento de la despensa virtual de un usuario.	
Entrada	ID alimento	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal.
	2	El usuario hace click en “Mi despensa”.
	3	El sistema le muestra una pantalla con el listado de alimentos.
	4	El usuario busca el alimento y lo selecciona manteniendo pulsado el alimento.
	5	El sistema muestra un icono de eliminar.
	6	El usuario hace click sobre el mismo.
	7	El sistema muestra un mensaje de confirmación para borrar el alimento.
	8	El usuario hace click en “Aceptar”.
	9	El sistema borra el alimento seleccionado.
	10	El sistema muestra al usuario la página principal con el listado de su despensa virtual actualizada.
Postcondición	Se ha eliminado el alimento	
Excepciones	Paso	Acción
	9	Se muestra el mensaje correspondiente si no se ha podido eliminar el alimento.
Comentarios	NA	
Actores	Usuario registrado.	

Requisito	**Modificar alimento de la despensa**	
Identificador	2.3	
Precondición	El usuario debe estar logueado y tener al menos un alimento en su despensa.	
Descripción	Los usuarios pueden modificar alimentos de la aplicación debido a algún error en su creación o la necesidad de modificar algún dato.	
Entrada	ID alimento, cantidad.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal.
	2	El usuario hace click en “Mi despensa”.
	3	El usuario busca el alimento y lo selecciona haciendo click en él.
	4	El sistema muestra la información del alimento y la posibilidad de modificar estos datos mediante campos.
	5	El usuario modifica los datos que correspondan y hace click en “Modificar”.
	6	El sistema muestra un mensaje de confirmación para modificar la comida y el usuario hace click en “Aceptar”.
	7	El sistema valida los datos introducidos por el usuario y modifica la comida seleccionada.
	8	El sistema muestra al usuario la página principal con el listado de su despensa actualizado.
Postcondición	Se ha modificado el alimento de la despensa.	
Excepciones	Paso	Acción
	7	El sistema muestra el mensaje correspondiente si no se ha podido guardar los cambios.
Comentarios	NA	
Actores	Usuario registrado	

Requisito	**Buscar alimento en la despensa**	
Identificador	2.4	
Precondición	El usuario debe estar logueado.	
Descripción	Los usuarios pueden buscar alimentos en su despensa virtual de entre los que dispone.	
Entrada	Nombre del alimento.	
Salida	Los alimentos que contengan los datos de búsqueda.	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal.
	2	El usuario hace click en “Mi despensa”.
	3	El sistema muestra el listado de alimentos.
	4	El usuario hace click en “Buscar”.
	5	El sistema muestra una barra de búsqueda donde el usuario introducirá el nombre del alimento a buscar.
	6	El sistema valida los datos introducidos por el usuario.
	7	El sistema muestra el listado de alimentos coincidentes con la búsqueda.
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	NA	
Actores	Usuario registrado.	

Requisito	**Ver alimento de la despensa**	
Identificador	2.5	
Precondición	El usuario debe estar logueado.	
Descripción	Los usuarios pueden visualizar los alimentos disponibles en su despensa virtual.	
Entrada	ID alimento.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal.
	2	El usuario hace click en “Mi despensa”.
	3	El sistema muestra el listado de alimentos.
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	NA	
Actores	Usuario registrado.	

4.2.3. Módulo gestionar recetas

El Diagrama de Caso de Uso (DCU):

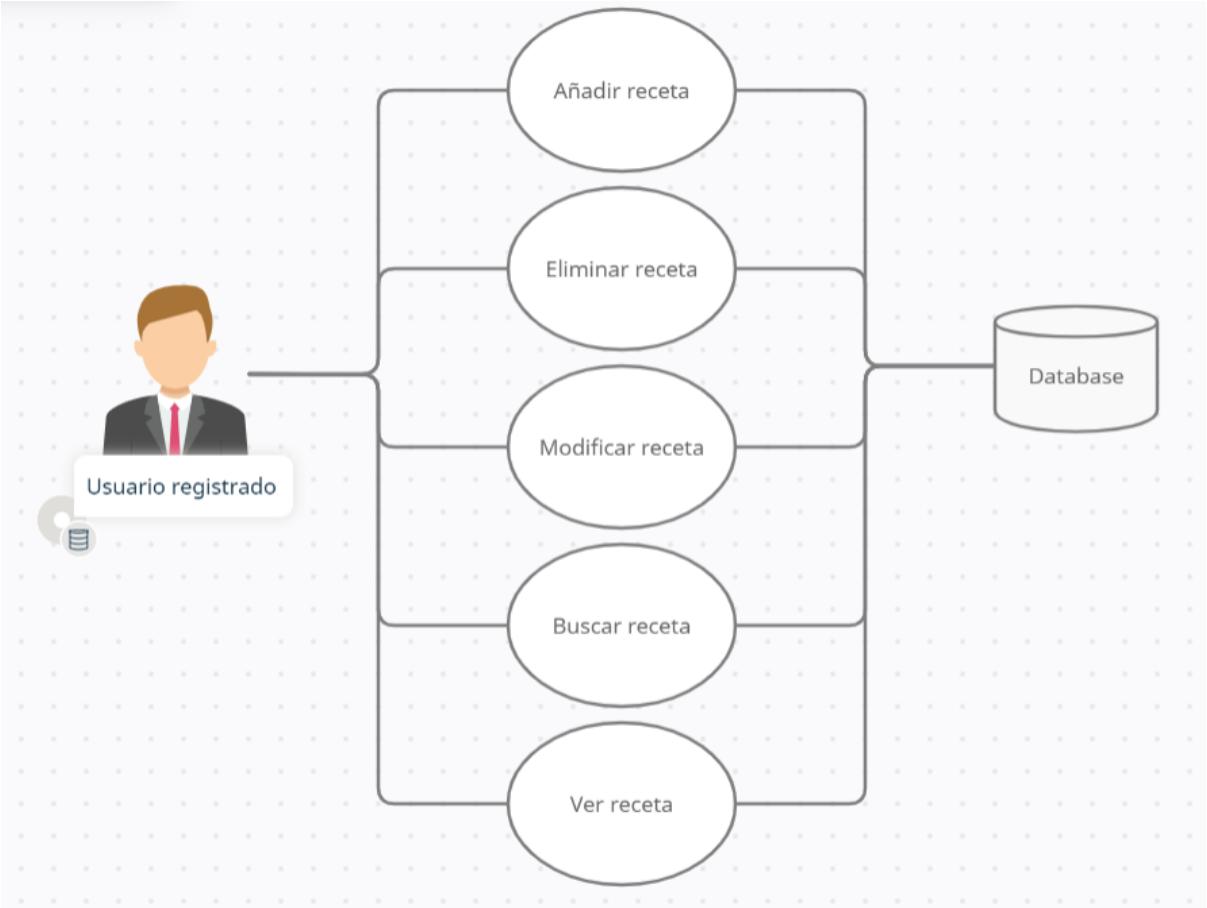


Figura 4.3: Diagrama de casos de uso del módulo gestionar recetas

Requisito	**Añadir receta**	
Identificador	3.1	
Precondición	El usuario debe estar logueado.	
Descripción	Los usuarios pueden añadir nuevas recetas a la aplicación que compartirán si desean con el resto de usuarios para poder generar las posibles recetas diarias.	
Entrada	Nombre de la receta, pares de ingredientes y cantidades, foto y descripción.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página con un listado de recetas.
	2	El usuario hace click en “Añadir receta”.
	3	El sistema le muestra una pantalla con los datos a rellenar para poder añadir una receta.
	4	El usuario rellena los campos. Selecciona si desea que sea pública o privada y hace click en “Añadir”.
	5	El sistema muestra un mensaje de confirmación de los datos.
	6	El usuario hace click en “Aceptar''.
	7	El sistema valida los datos introducidos por el usuario y añade la receta.
	8	El sistema muestra al usuario la página principal con el listado de recetas actualizado.
Postcondición	Se ha añadido una receta.	
Excepciones	Paso	Acción
	7	Se muestra el mensaje correspondiente si algún elemento introducido es incorrecto o se haya vacío cuando no debería.
Comentarios	NA	
Actores	Usuario registrado	

Requisito	**Eliminar receta**	
Identificador	3.2	
Precondición	El usuario logueado debe estar logueado.	
Descripción	Los usuarios pueden eliminar recetas de la aplicación que ya no resulten necesarias o supongan un problema.	
Entrada	ID receta	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página con un listado de recetas.
	2	El usuario busca la receta y la selecciona manteniendo pulsada la receta.
	3	El sistema muestra un icono de eliminar y el usuario hace click sobre el mismo.
	4	El sistema muestra un mensaje de confirmación para borrar la receta.
	5	El usuario hace click en “Aceptar”.
	6	El sistema borra la receta seleccionada.
	7	El sistema muestra al usuario la página principal con el listado de recetas actualizado.
Postcondición	Se ha eliminado una receta	
Excepciones	Paso	Acción
	4	Se muestra el mensaje correspondiente si no se ha podido eliminar la receta.
Comentarios	NA	
Actores	Usuario registrado	

Requisito	**Modificar receta**	
Identificador	3.3	
Precondición	El usuario debe estar logueado.	
Descripción	Los usuarios pueden modificar recetas de la aplicación debido a algún error en su creación o la necesidad de modificar algún dato.	
Entrada	ID receta, nombre de la receta, pares de ingredientes y cantidades, foto y descripción.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página con un listado de recetas.
	2	El usuario busca la receta y la selecciona haciendo click en ella.
	3	El sistema muestra la información de la receta y la posibilidad de modificar estos datos mediante campos.
	4	El usuario modifica los datos que correspondan y hace click en “Modificar”.
	5	El sistema muestra un mensaje de confirmación para modificar la receta.
	6	El usuario hace click en “Aceptar”.
	7	El sistema valida los datos introducidos por el usuario y modifica la receta seleccionada.
	8	El sistema muestra al usuario la página principal con el listado de recetas actualizado.
Postcondición	Se ha modificado la receta	
Excepciones	Paso	Acción
	7	Se muestra el mensaje correspondiente si el elemento no se ha modificado correctamente debido a que algún elemento modificado es invalido.
Comentarios	NA	
Actores	Usuario registrado	

Requisito	**Buscar receta**	
Identificador	3.4	
Precondición	El usuario debe estar logueado.	
Descripción	Los usuarios pueden buscar una determinada receta.	
Entrada	Nombre de la receta.	
Salida	Las recetas que contengan los datos de búsqueda.	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página con un listado de recetas.
	2	El usuario introduce el nombre de la receta a buscar en la barra de búsqueda.
	3	El usuario rellena el nombre de la receta y busca la receta.
	4	El sistema validará los datos introducidos por el usuario.
	5	El sistema muestra el listado de recetas coincidentes con la búsqueda.
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	NA	
Actores	Usuario registrado	

Requisito	**Ver receta**	
Identificador	3.5	
Precondición	El usuario debe logueado.	
Descripción	El usuario puede visualizar una receta almacenada en la base de datos.	
Entrada	ID receta.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página con un listado de recetas.
	2	El usuario selecciona la receta que desea visualizar.
	3	El sistema muestra la información de la receta seleccionada
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	NA	
Actores	Usuario registrado	

4.2.4. Módulo lista de la compra

El Diagrama de Caso de Uso (DCU):

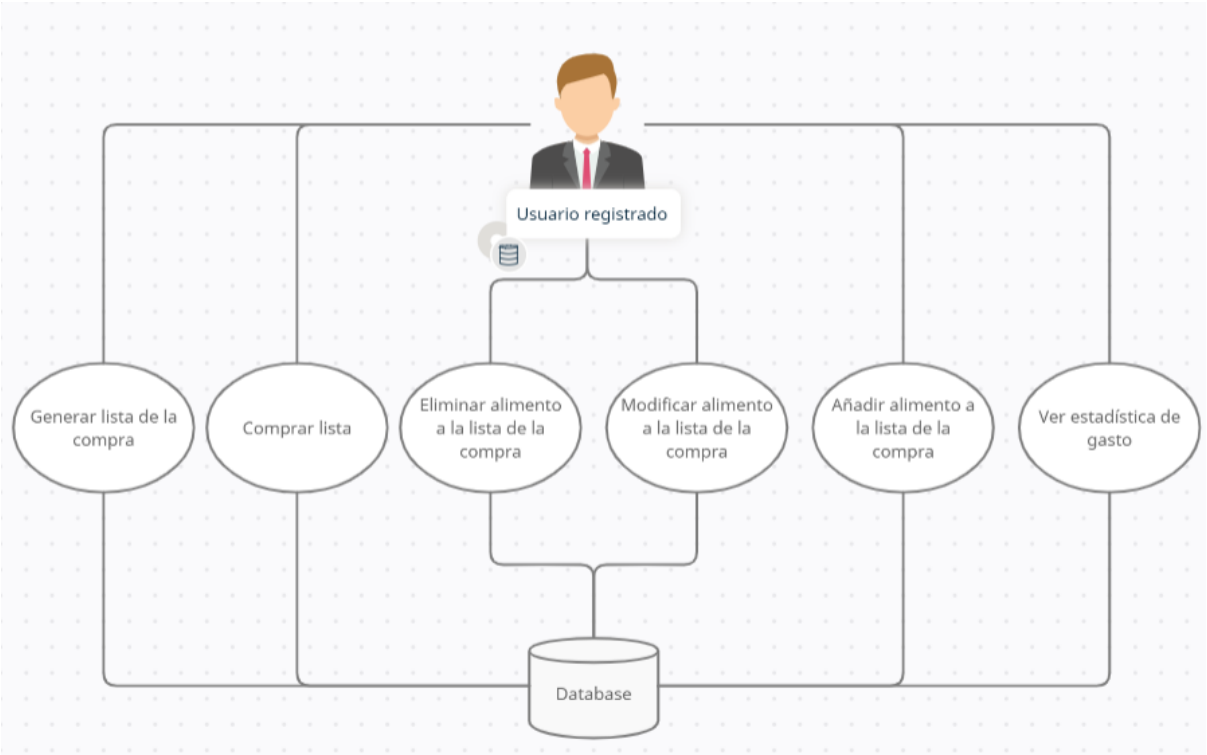


Figura 4.4: Diagrama de casos de uso del módulo lista de la compra

Requisito	**Añadir alimento a la lista de la compra**	
Identificador	4.1	
Precondición	El usuario debe estar logueado.	
Descripción	Los usuarios podrán añadir a su lista de la compra alimentos que deseen tener en su despensa.	
Entrada	Nombre de alimento y cantidad.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal.
	2	El usuario hace click en “Mi lista”.
	3	El sistema muestra la lista de la compra.
	4	El usuario hace click en “Añadir alimento” para añadirlo a la lista de la compra.
	5	El usuario introducirá el alimento en la lista con la correspondiente cantidad que desee.
	6	El sistema muestra un mensaje de confirmación de los datos introducidos.
	7	El usuario hace click en “Aceptar”.
	8	El sistema valida los datos introducidos por el usuario.
	9	El sistema muestra la lista de la compra actualizada.
Postcondición	Se ha añadido un alimento a la lista de la compra	
Excepciones	Paso	Acción
	8	Se muestra el mensaje correspondiente si algún dato introducido no es el adecuado.
Comentarios	NA	
Actores	Usuario registrado.	

Requisito	**Eliminar alimento de la lista de la compra**	
Identificador	4.2	
Precondición	El usuario debe estar logueado.	
Descripción	Los usuarios podrán eliminar productos de su lista de la compra.	
Entrada	ID alimento	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra en la página principal.
	2	El usuario hace click en “Mi lista”.
	3	El sistema muestra el listado de alimentos a comprar.
	4	El usuario hace click en el alimento a eliminar.
	5	El sistema muestra la información relativa al alimento seleccionado.
	6	El usuario pulsa “Eliminar”.
	7	El sistema muestra un mensaje de confirmación de los cambios.
	8	El usuario hace click en “Aceptar”.
	9	El sistema valida los datos.
	10	El sistema muestra la pantalla con todos los alimentos de la lista de la compra del usuario.
Postcondición	Se ha quitado el alimento de la compra	
Excepciones	Paso	Acción
	9	Se muestra el mensaje correspondiente si no se ha podido eliminar el alimento.
Comentarios	NA	
Actores	Usuario registrado.	

Requisito	**Modificar alimento de la lista de la compra**	
Identificador	4.3	
Precondición	El usuario debe estar logueado.	
Descripción	Los usuarios podrán modificar la cantidad de los alimentos de la lista de su compra.	
Entrada	Cantidad e ID alimento	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal.
	2	El usuario hace click en “Mi lista”.
	3	El sistema muestra el listado de alimentos de la lista de la compra.
	4	El usuario hace click en el alimento a modificar.
	5	El sistema muestra la información del alimento, que podrá modificarse en esta misma.
	6	El usuario rellena los datos y hace click en “Editar”.
	7	El sistema muestra un mensaje de confirmación de los cambios.
	8	El usuario hace click en “Aceptar”.
	9	El sistema valida los datos introducidos y muestra el alimento con los datos actualizados.
Postcondición	Se ha modificado la cantidad del alimento	
Excepciones	Paso	Acción
	9	Se muestra mensaje correspondiente de error si el dato introducido es incorrecto
Comentarios	NA	
Actores	Usuario registrado.	

Requisito	**Generar lista de la compra**	
Identificador	4.4	
Precondición	Hay nulo stock de algún alimento en la despensa.	
Descripción	El sistema revisará el contenido de la despensa virtual y al eliminarse un alimento porque ya no queda stock se añadirá automáticamente a la lista para su compra.	
Entrada	ID alimentos	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El sistema revisa los alimentos que había en la despensa.
	2	Cuando se elimina un alimento de la despensa por falta de stock el sistema lo añadirá a la lista de la compra.
Postcondición	Se ha añadido un alimento a la lista de la compra	
Excepciones	Paso	Acción
Comentarios	NA	
Actores	NA	

Requisito	**Comprar lista**	
Identificador	4.5	
Precondición	El usuario debe estar logueado.	
Descripción	Los usuarios añaden a su despensa virtual los alimentos que hayan comprado.	
Entrada	ID alimento y gasto.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal.
	2	El usuario hace click en “Mi lista”.
	3	El sistema muestra la lista de la compra.
	4	El usuario hace click en “Alimento adquirido” de aquellos que haya comprado.
	5	Tras haber seleccionado los alimentos que ha comprado el sistema le solicitará el gasto total de la compra para realizar estadísticas.
	6	El sistema añade los alimentos a la despensa y muestra la lista de la compra actualizada.
Postcondición	Se ha añadido el alimento en la despensa y se ha actualizado las estadísticas de gasto.	
Excepciones	Paso	Acción
	6	Se muestra el mensaje correspondiente si no se ha podido añadir el alimento a la despensa.
Comentarios	NA	
Actores	Usuario registrado.	

Requisito	**Ver estadísticas de gasto**	
Identificador	4.6	
Precondición	El usuario debe estar logueado.	
Descripción	Los usuarios pueden ver gráficas sobre su gasto en la compra de alimentos.	
Entrada	Fecha.	
Salida	Gráfico de gasto.	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal.
	2	El usuario hace click en “Mi lista”.
	3	El sistema muestra la lista de la compra.
	4	El usuario hace click en “Historial de gastos”.
	5	El sistema muestra una pantalla con el gráfico de gastos por semanas por defecto.
	6	El usuario puede elegir ver el gasto por semanas o meses.
Postcondición	NA	
Excepciones	Paso	Acción
	6	Se muestra el mensaje correspondiente si no se ha podido calcular el gasto por semanas o meses.
Comentarios	NA	
Actores	Usuario registrado.	

Requisito	**Buscar alimento de la lista de la compra**	
Identificador	4.7	
Precondición	El usuario debe estar logueado.	
Descripción	Los usuarios pueden buscar un determinado alimento de la lista de la compra.	
Entrada	Nombre del alimento de la lista de la compra.	
Salida	Los alimentos de la lista de la compra que contengan los datos de búsqueda.	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal.
	2	El usuario hace click en “Mi lista”.
	3	El sistema muestra la lista de la compra.
	4	El usuario rellena el nombre del alimento y busca el alimento.
	5	El sistema validará los datos introducidos por el usuario.
	6	El sistema muestra el listado de alimentos coincidentes con la búsqueda.
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	NA	
Actores	Usuario registrado	

4.2.5. Módulo gestionar comidas

El Diagrama de Caso de Uso (DCU):

Figura 4.5: Diagrama de casos de uso del módulo gestionar comidas

Requisito	**Elegir comida**	
Identificador	5.1	
Precondición	El usuario debe estar logueado.	
Descripción	El sistema en función de los alimentos disponibles en la despensa y los alimentos necesarios para las recetas muestra una lista de recetas que pueden ser cocinadas para que el usuario pueda elegir para visualizarla y así cocinarla. El sistema creará un registro en el historial donde se guardará la receta preparada una vez haya sido cocinada.	
Entrada	ID alimentos de la despensa.	
Salida	Lista de recetas posibles.	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal, en concreto se visualizarán las comidas posibles para cocinar en base a los alimentos de la despensa.
	2	El usuario selecciona la comida a visualizar.
	3	El sistema muestra al usuario la información de esa receta.
	4	El usuario al preparar la receta, hace click en “Comida preparada”.
	5	El sistema actualizará el historial de comidas.
Postcondición	NA	
Excepciones	Paso	Acción
	5	Se muestra el mensaje correspondiente si no se ha podido actualizar el historial de comidas.
Comentarios	NA	
Actores	Usuario registrado.	

Requisito	**Ver historial de comidas**	
Identificador	5.2	
Precondición	El usuario debe estar logueado y haber marcado una comida como realizada.	
Descripción	Los usuarios pueden visualizar las comidas que hayan realizado con anterioridad.	
Entrada	ID comida.	
Salida	NA	
Secuencia nominal	Paso	Acción
	1	El usuario accede a la aplicación y el sistema le muestra la página principal, en concreto se visualizarán las comidas del día actual.
	2	El usuario hace click en “Historial”.
	3	El sistema muestra un listado con las comidas que el usuario ha marcado como realizadas.
	4	El usuario puede seleccionar una comida haciendo click sobre la misma.
	5	El sistema muestra la información sobre la comida seleccionada.
Postcondición	NA.	
Excepciones	Paso	Acción
	3	Se muestra el mensaje correspondiente si la lista de comidas se encuentra vacía.
Comentarios	NA	
Actores	Usuario registrado.	

5. Arquitectura de la aplicación

En este apartado se define la estructura de la aplicación y su respectivo modelo de datos.

5.1. Estructura de la aplicación

Para el desarrollo de la aplicación se ha empleado la técnica de una arquitectura sin servidor denominada “Serverless”. Una arquitectura “Serverless” consiste en un modelo de desarrollo que permite el despliegue de aplicaciones donde el proveedor de servicios en la nube administra la infraestructura necesaria para el funcionamiento de la aplicación y se encarga del equilibrio dinámico de recursos necesarios para la gestión de los recursos necesarios. Este enfoque, permite a los desarrolladores de código centrarse en la creación de la lógica de la aplicación despreocupándose de la gestión de los servidores necesarios para el correcto funcionamiento de la aplicación, las bases de datos y los sistemas de almacenamiento.

Figura 5.1: Esquema arquitectura de la aplicación

5.2. Modelo de datos

La base de datos maneja información sobre:

- Menú: almacena información para cada usuario, con relación a los nombres de recetas cuyos ingredientes están disponibles en la despensa del usuario.

- Pantry: almacena listas de datos para cada usuario sobre alimentos que dispongan en sus hogares.

- Recipes: almacena para cada usuario información acerca de las recetas publicadas por los mismos, abarcando datos como nombre de receta, descripción y lista de ingredientes. Además, integra recetas que los usuarios comparten, es decir, que gozan de carácter público, almacenando para ello los mismos campos mencionados.

- Shopping: integra, para cada usuario, una lista de los gastos realizados en sus compras. Para ello, almacena tanto la fecha de la realización de compra como el importe. Además, integra una lista de ingredientes a comprar, abarcando nombre y cantidad.

- Users: para cada usuario, se almacena información relativa a su tipo de cuenta, es decir, si es administrador o usuario, la fecha de nacimiento, el email, nombre de usuario y género.

Mediante el uso de Firebase, con Cloud Firestore, se almacena de manera persistente la información necesaria para el correcto funcionamiento de la aplicación y el manejo de la información de los usuarios.

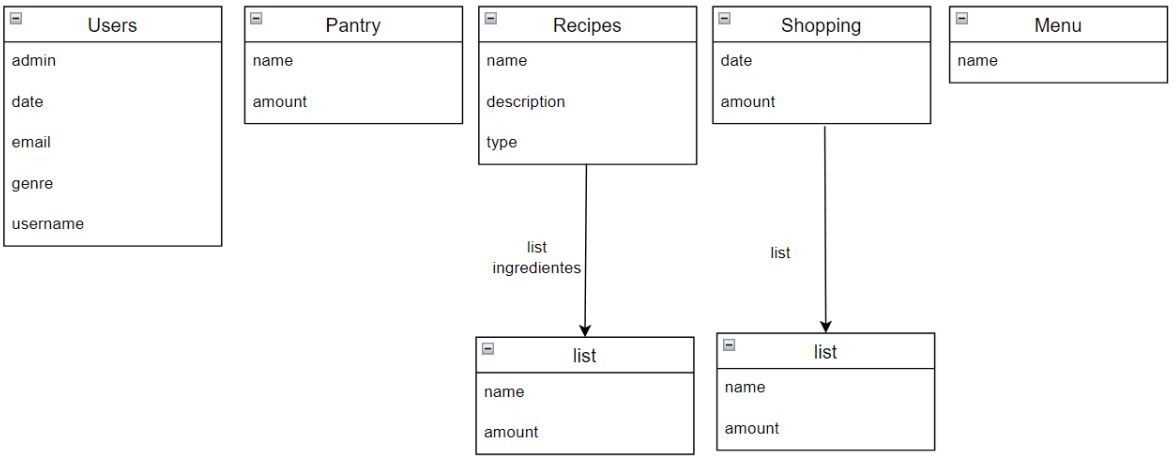


Figura 5.2: Esquema modelo de datos

5.2.1. Colección Menú

La colección menú es usada para almacenar comidas que puedan ser elaboradas por los usuarios, en función de los ingredientes que disponen en sus despensas. Almacena para cada usuario un listado con nombres de comidas.

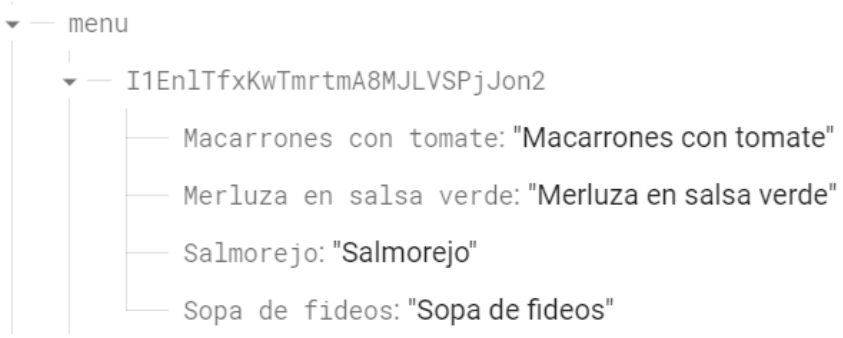


Figura 5.3: Colección Menú

5.2.2. Colección Pantry

La colección pantry es usada para almacenar alimentos que el usuario disponga en su despensa. Un ingrediente está caracterizado por una clave que hace referencia al nombre y un valor relativo a la cantidad disponible. Por tanto, esta colección almacena, para cada usuario, un listado de ingredientes disponibles en la despensa de este.

Figura 5.4: Colección Pantry

5.2.3. Colección Recipes

La colección recipes es usada para almacenar información sobre recetas. La información de la receta abarca campos como descripción, nombre, tipo (pública o privada) y una lista de pares donde se almacena el nombre y la cantidad de cada ingrediente. En el caso de las recetas privadas, esta colección almacena, para cada usuario, un conjunto de recetas identificadas por el nombre y que contienen información relativa a las mismas. En el caso de recetas públicas, esta colección almacena, por cada creador de receta pública, un listado de recetas abarcando información anteriormente mencionada.

Figura 5.5: Colección Recipes (receta privada)

Figura 5.6: Colección Recipes (receta pública)

5.2.4. Colección Shopping

La colección shopping almacena información acerca de los gastos relativos a las compras. Para su gestión se almacena como clave la fecha de la realización de compra y como valor el importe pagado. Además, esta colección gestiona información relativa a los productos que integran la lista de la compra, usando como clave el nombre del ingrediente y como valor la cantidad a comprar. Por tanto, la colección shopping almacena, para cada usuario, una lista de gastos y una lista de ingredientes.

Figura 5.7: Colección Shopping

5.2.5. Colección Users

La colección users integra información acerca de los usuarios. Almacena campos como fecha (fecha nacimiento), "admin" (tipo de usuario: identifica si se trata de un usuario o administrador), email, género y nombre de usuario. Por tanto, diversos usuarios, y por cada usuario, se almacena información relativa del mismo.

Figura 5.8: Colección Users

6. Implementación y diseño de la aplicación

En este capítulo se presenta la implementación y diseño de la aplicación. La información se presentará en base a los módulos existentes: módulo cuentas de usuario, módulo despensa virtual, módulo gestionar recetas, módulo lista de la compra y módulo gestionar comidas.

6.1. Módulo cuentas de usuario

6.1.1. Registrar usuario

Esta vista contiene los campos username (nombre de usuario), email, password (contraseña), birthdate (fecha nacimiento) y genre (género). Estos campos se ven reflejados en la figura 6.1, donde se pueden introducir las credenciales para realizar el registro de un usuario. En el caso de que haya algún campo vacío, no se puede llevar a cabo el registro ya que el botón "sign up" no estará activo para su uso. Se encuentran advertencias como la relativa al correo electrónico. Este campo debe tener al menos un carácter antes de un "@", debe contener un "@", un "." y contener de 2 a 4 caracteres tras el ".". También, existen advertencias relativas a la longitud de la contraseña y campos vacíos. Por último, en caso de que haya un usuario ya creado con el mismo correo se muestra fallo. En caso de no ocurrir errores, se almacenan los datos del usuario en Firebase[10] y se redirecciona a la página principal del usuario, ilustrada en la figura 6.2. Además, se puede parte del código asociado a este caso de uso en la figura 6.3.

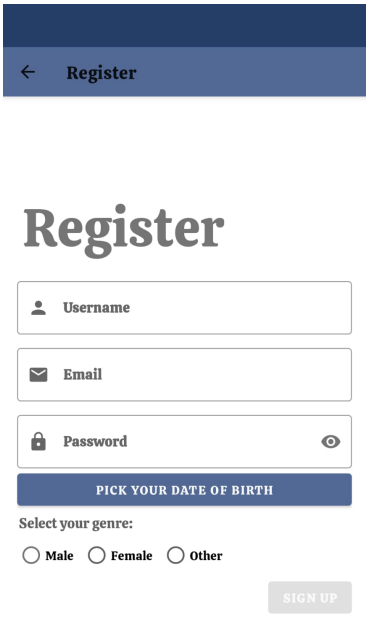


Figura 6.1: Pantalla de registro

Figura 6.2: Pantalla principal de la aplicación

```java
public void register(String username, String email, String password, String checked, String date) {
    mAuth = FirebaseAuth.getInstance();
    myRef = FirebaseDatabase.getInstance().getReference();

    mAuth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()) {
                        userF = mAuth.getCurrentUser();
                        User user = new User(username, email, checked, date);
                        myRef.child( pathString: "users").child(userF.getUid()).setValue(user);
                        registerResult.setValue(new RegisterResult(username));
                    } else {
                        registerResult.setValue(new RegisterResult("Error while creating your account"));
                    }
                }
            });
}
```

Figura 6.3: Fragmento de código (Registro)

6.1.2. Login

Esta vista contiene los campos email y password. Estos campos se ven reflejados en la figura 6.4, donde se pueden introducir las credenciales para realizar el login de un usuario o administrador. En el caso de que los campos estén vacíos no está activa la acción sign in. Existen advertencias en caso de que existan campos vacíos. También, hay advertencia sobre la longitud de la contraseña. Una vez introducidas las credenciales, se comprueba la existencia de esos datos en la base de datos. En caso de error, se muestra fallo de login, en caso contrario, se redirige a la página principal de usuario o administrador, tal y como refleja la imagen 6.2. Además, se muestra parte del código asociado a este caso de uso en la figura 6.5.

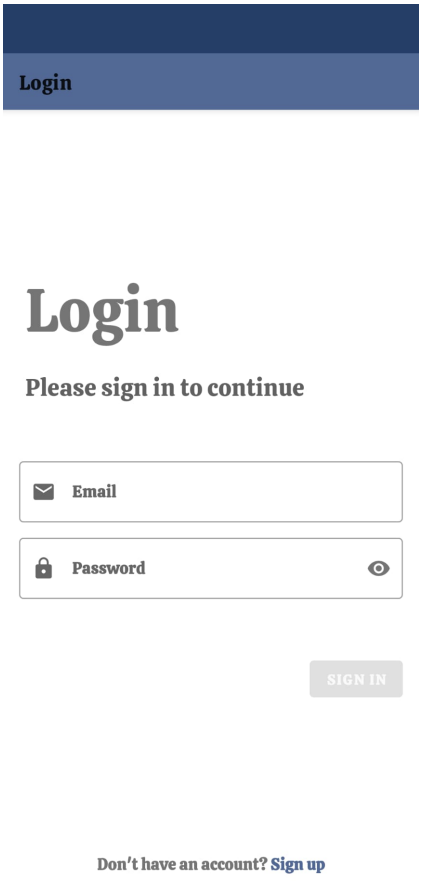


Figura 6.4: Pantalla de login

```
public void login(String email, String password) {
    mAuth = FirebaseAuth.getInstance();
    mAuth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
                    FirebaseUser user = mAuth.getCurrentUser();
                    myRef = FirebaseDatabase.getInstance().getReference();
                    assert user != null;
                    myRef.child( pathString: "users").child(user.getUid()).child( pathString: "username").get().addOnCompleteListener(task1 -> {
                        if (!task1.isSuccessful())
                            loginResult.setValue(new LoginResult("Login failed"));
                        else {
                            if (task1.getResult().getValue() != null)
                                loginResult.setValue(new LoginResult(String.valueOf(task1.getResult().getValue())));
                            else {
                                user.delete();
                                loginResult.setValue(new LoginResult("Your account has been delete"));
                            }
                        }
                    });
                } else {
                    loginResult.setValue(new LoginResult("Login failed"));
                }
            });
}
```

Figura 6.5: Fragmento de código (Login)

6.1.3. Logout

La acción relativa a logout se localiza en la vista relativa a datos de usuario. Para realizar la acción se localiza un icono en la esquina superior izquierda, tal y como muestra la figura 6.6. En este caso, al pulsar sobre el icono se muestra un diálogo para corroborar que el usuario o administrador quiere realizar un logout. Esto se ve reflejado en la figura 6.7. Una vez confirmado, se redirige a la página de Login 6.4. En caso de cancelar la operación, se mantiene en la vista relativa a datos de usuario 6.6.

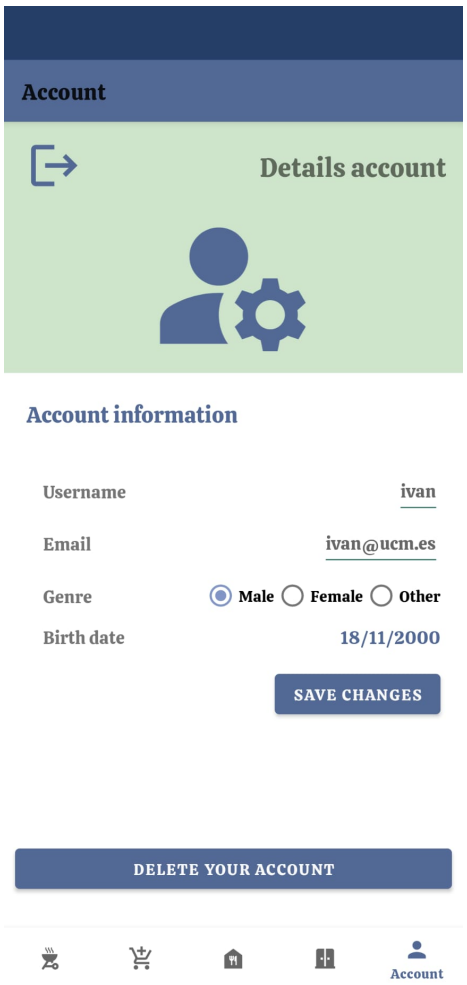


Figura 6.6: Pantalla de cuenta de usuario

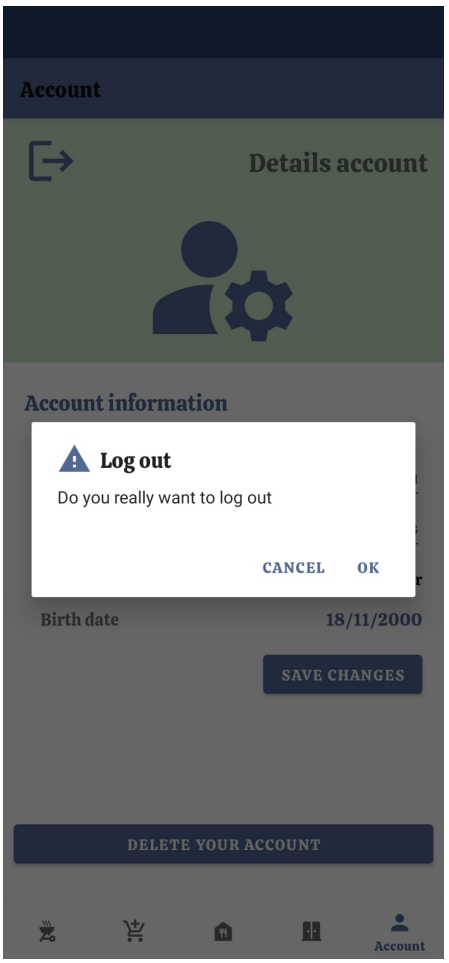


Figura 6.7: Diálogo logout

```
binding.iconLogout.setOnClickListener(v -> new AlertDialog.Builder(requireContext())
        .setTitle("Log out")
        .setMessage("Do you really want to log out")
        .setIcon(R.drawable.ic_warning)
        .setPositiveButton(android.R.string.yes, (dialog, whichButton) -> {
            mAuth.signOut();
            Toast.makeText(getContext(), text: "Log out.", Toast.LENGTH_LONG).show();
            Intent loginIntent = new Intent(getContext(), LoginActivity.class);
            startActivity(loginIntent);
        })
        .setNegativeButton(android.R.string.no, listener: null).show());
```

Figura 6.8: Fragmento de código (Logout)

6.1.4. Modificar datos de usuario

Esta vista ofrece al usuario la posibilidad tanto de observar la información relativa a su cuenta además de modificarla. Para realizar cualquier modificación el usuario podrá actualizar uno o varios de los campos que desee cambiar. Para registrar correctamente los datos, evitando modificaciones por error, el usuario pincha en un botón situado en la parte inferior del formulario tal y como muestra la figura 6.6. En este caso, al pulsar sobre el icono se muestra un diálogo para corroborar que el usuario quiere realizar una modificación de sus datos. Esto se ve reflejado en la figura 6.9.

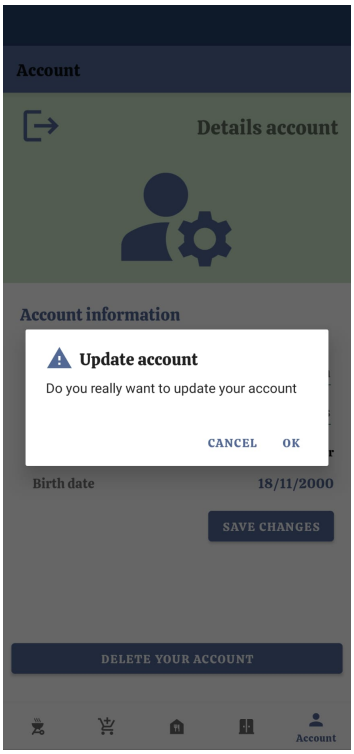


Figura 6.9: Diálogo actualizar cuenta

```
binding.save.setOnClickListener(v -> new AlertDialog.Builder(requireContext())
        .setTitle("Update account")
        .setMessage("Do you really want to update your account")
        .setIcon(R.drawable.ic_warning)
        .setPositiveButton(android.R.string.yes, (dialog, whichButton) -> {
            String username = Objects.requireNonNull(binding.editUsernameAccount.getText()).toString();
            String email = Objects.requireNonNull(binding.editEmailAccount.getText()).toString();
            String edit_date = date.getText().toString();
            if (!username.equals("")) myRef.child( pathString: "users").child(actual_user.getUid()).child( pathString: "username").setValue(username);
            if (!email.equals("")) {
                myRef.child( pathString: "users").child(actual_user.getUid()).child( pathString: "email").setValue(email);
                actual_user.updateEmail(email);
            }
            if (!checked.equals("")) myRef.child( pathString: "users").child(actual_user.getUid()).child( pathString: "genre").setValue(checked);
            if (!edit_date.equals("")) myRef.child( pathString: "users").child(actual_user.getUid()).child( pathString: "date").setValue(edit_date);
            binding.editUsernameAccount.setText("");
            binding.editEmailAccount.setText("");
            Toast.makeText(getContext(),  text: "Your account information has beed updated", Toast.LENGTH_SHORT).show();
        })
        .setNegativeButton(android.R.string.no,  listener: null).show());
```

Figura 6.10: Fragmento de código (actualizar cuenta)

6.1.5. Darse de baja

La acción relacionada con darse de baja de la aplicación se encuentra en la vista relativa a datos de usuario. Para realizar la acción, el usuario dispone de un botón en la parte inferior de la pantalla donde puede eliminar su cuenta, tal y como muestra la figura 6.6. Al pulsar sobre el botón se muestra un diálogo para corroborar el borrado de cuenta, podemos observar en la figura 6.11 el diálogo. Una vez confirmado, se redirige a la página de Login 6.4. En caso de cancelar la operación, se mantiene en la vista relativa a datos de usuario 6.6.

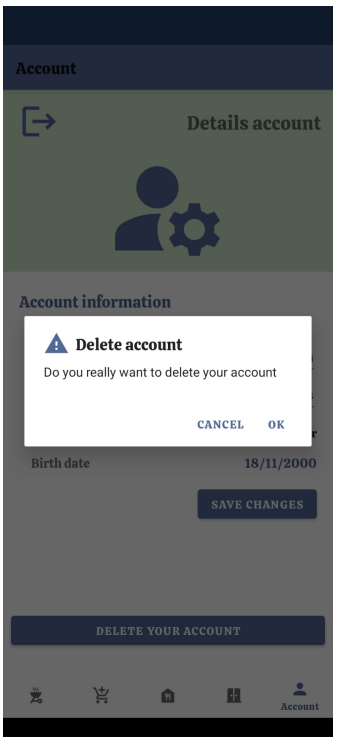


Figura 6.11: Diálogo eliminar cuenta

```
binding.deleteAccount.setOnClickListener(v -> new AlertDialog.Builder(requireContext())
        .setTitle("Delete account")
        .setMessage("Do you really want to delete your account")
        .setIcon(R.drawable.ic_warning)
        .setPositiveButton(android.R.string.yes, (dialog, whichButton) -> {
            myRef.child( pathString: "users").child(actual_user.getUid()).removeValue();
            myRef.child( pathString: "pantry").child(actual_user.getUid()).removeValue();
            myRef.child( pathString: "recipes").child(actual_user.getUid()).removeValue();
            actual_user.delete();
            FirebaseAuth.getInstance().signOut();
            Toast.makeText(getContext(),  text: "Account delete.", Toast.LENGTH_LONG).show();
            Intent loginIntent = new Intent(getContext(), LoginActivity.class);
            startActivity(loginIntent);
        })
        .setNegativeButton(android.R.string.no,  listener: null).show());
```

Figura 6.12: Fragmento de código (eliminar cuenta)

6.1.6. Eliminar usuario registrado

Únicamente un administrador puede eliminar un usuario de la aplicación. Para realizar la acción, el administrador buscará entre la lista de usuarios al usuario a eliminar, tal y como muestra la figura 6.16. Una vez seleccionado el usuario, se desplegará la información sobre él y el

administrador podrá eliminar su cuenta al pulsar el botón situado en la parte inferior de la pantalla, tal y como muestra la figura 6.13. Una vez pulsado el botón se muestra un diálogo para corroborar el borrado de cuenta, podemos observar en la figura 6.14 el diálogo. Una vez confirmado, se redirige a la página de usuarios registrados 6.16. En caso de cancelar la operación, se mantiene en la vista relativa a datos de usuario registrado 6.13.

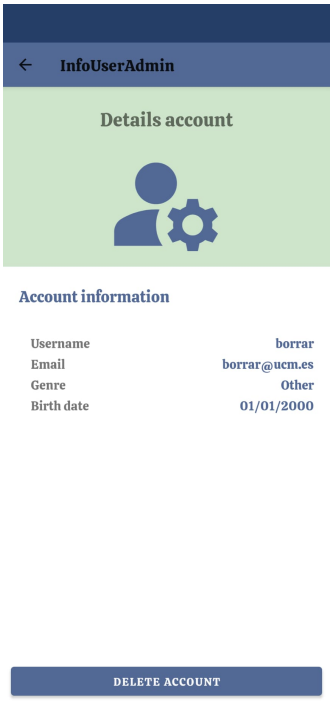


Figura 6.13: Pantalla información usuario registrado

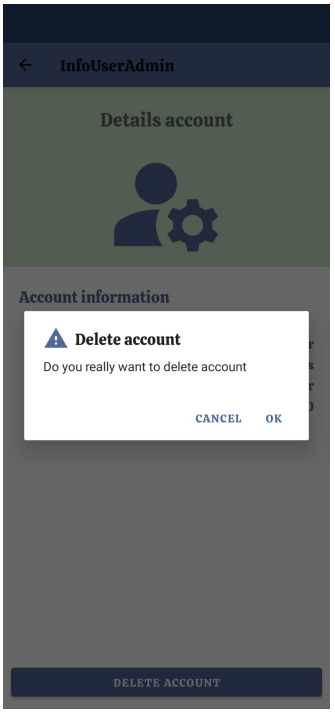


Figura 6.14: Diálogo eliminar cuenta usuario

```
binding.deleteAccountUser.setOnClickListener(v -> new AlertDialog.Builder( context: this)
        .setTitle("Delete account")
        .setMessage("Do you really want to delete account")
        .setIcon(R.drawable.ic_warning)
        .setPositiveButton(android.R.string.yes, (dialog, whichButton) -> {
            myRef.child( pathString: "users").child(uid).removeValue();
            Toast.makeText(getApplicationContext(),  text: "User has been delete", Toast.LENGTH_LONG).show();
            finish();
        })
        .setNegativeButton(android.R.string.no,  listener: null).show());
```

Figura 6.15: Fragmento de código (eliminar cuenta usuario)

6.1.7. Ver usuarios registrados

Para visualizar los usuarios registrados en la aplicación, el administrador dispone de un listado de usuarios, tal y como muestra la figura 6.16. Al pulsar sobre un usuario el administrador podrá visualizar la información relativa a este al igual que muestra la figura 6.13.

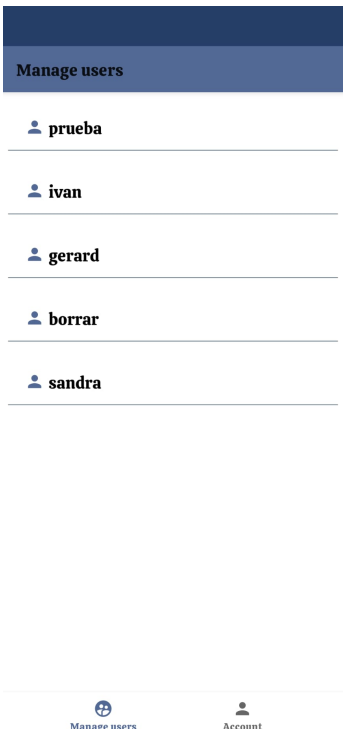


Figura 6.16: Pantalla usuarios registrados

```
myRef.child( pathString: "users").addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        userList.clear();
        for (DataSnapshot userSnapshot : snapshot.getChildren()) {
            User user = userSnapshot.getValue(User.class);
            assert user != null;
            if (!user.getAdmin() && !userList.contains(user)) {
                userList.addLast(user);
            }
        }
        recyclerView = binding.recyclerViewAdmin;
        mAdapter = new UserAdapter(getContext(), userList);
        recyclerView.setAdapter(mAdapter);
        recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));
    }
```

Figura 6.17: Fragmento de código (usuarios registrados)

6.2. Módulo despensa virtual

6.2.1. Añadir alimento a la despensa.

Para realizar la acción de añadir alimento se debe partir de la vista principal de la despensa, reflejada en la figura 6.18. El usuario debe seleccionar el icono situado en la esquina inferior derecha, caracterizado por un signo "+", para ser redirigido a la vista correspondiente a la acción añadir alimento a la despensa. Esta vista refleja la acción de añadir un alimento a la despensa. Únicamente el usuario registrado puede realizar esta acción. Se caracteriza por dos campos a cumplimentar, recayendo en el nombre y cantidad del ingrediente, tal y como se ilustra en la figura 6.19. Se ha desarrollado un control, de tal forma que la acción de añadir, no se vea activada hasta que los datos hayan sido rellenados. Una vez cumplimentados los datos, se almacena la información en la base de datos y se redirecciona al usuario a la vista principal de la despensa.

Figura 6.18: Pantalla despensa virtual

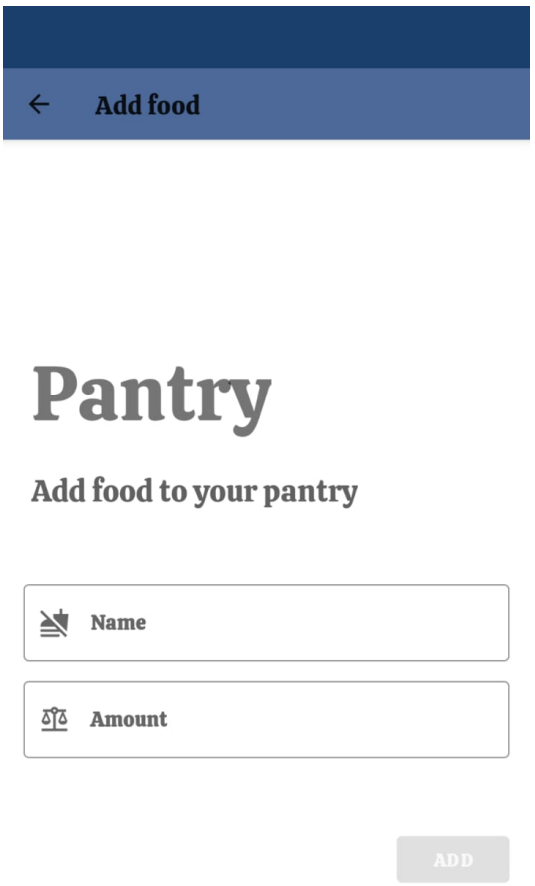


Figura 6.19: Pantalla añadir alimento

```
myRef.child( pathString: "pantry").child(user.getUid()).addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        foodList.clear();
        for (DataSnapshot foodSnapshot : snapshot.getChildren()) {
            String name = foodSnapshot.getKey();
            String amount = foodSnapshot.getValue(String.class);
            Food food = new Food(name, amount);
            if (!foodList.contains(food)) {
                foodList.addLast(food);
            }
        }
        mAdapter = new FoodAdapter(context, foodList);
        recyclerView.setAdapter(mAdapter);
        recyclerView.setLayoutManager(new LinearLayoutManager(context));
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
    }
});
```

Figura 6.20: Fragmento de código (despensa)

```
addButton.setOnClickListener(v -> {
    DatabaseReference myRef = FirebaseDatabase.getInstance().getReference();
    FirebaseUser user = mAuth.getCurrentUser();
    Food food = new Food(nameEditText.getText().toString(), amountEditText.getText().toString());
    assert user != null;
    myRef.child( pathString: "pantry").child(user.getUid()).child(food.getName()).setValue(food.getAmount());
    Toast.makeText(getApplicationContext(),  text: "Has been added " + food.getAmount() + " of " + food.getName(), Toast.LENGTH_SHORT).show();
    finish();
});
```

Figura 6.21: Fragmento de código (añadir alimento)

6.2.2. Eliminar alimento de la despensa.

Para realizar esta acción, el usuario debe estar situado en la vista principal de la despensa 6.18. La vista relativa a eliminar alimento de la despensa se caracteriza por un ícono correspondiente a una basura situado en el extremo derecho del ingrediente a eliminar, tal y como se ilustra en la figura 6.22. Para llegar a esta vista, el usuario ha mantenido pulsado el nombre del ingrediente a eliminar. Una vez seleccionada la acción de eliminar, se muestra un diálogo con el objetivo de confirmar que el usuario desea eliminar el ingrediente, tal y como se refleja en la imagen 6.23. En el caso de cancelar la acción, se redirecciona a la vista principal de la despensa. En el caso de eliminar el alimento, se mostrará otro diálogo. Este diálogo tiene el objetivo de facilitar la elaboración de la lista de la compra, por lo que se pregunta si el alimento eliminado debe ser

añadido a la lista de la compra, tal y como se muestra en la figura 6.24. En cualquier caso, se redirige al usuario a la vista principal de la despensa.

Figura 6.22: Pantalla eliminar alimento despensa virtual

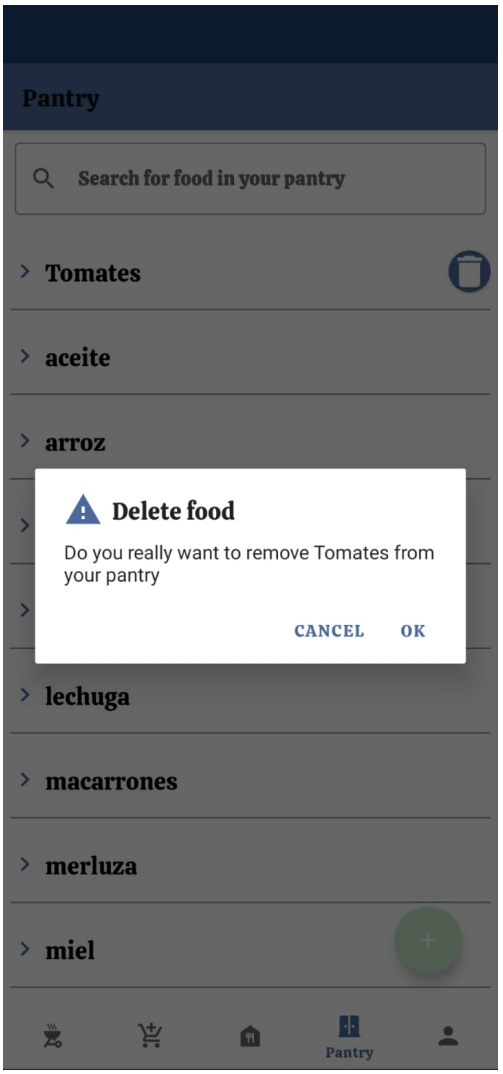


Figura 6.23: Diálogo eliminar alimento

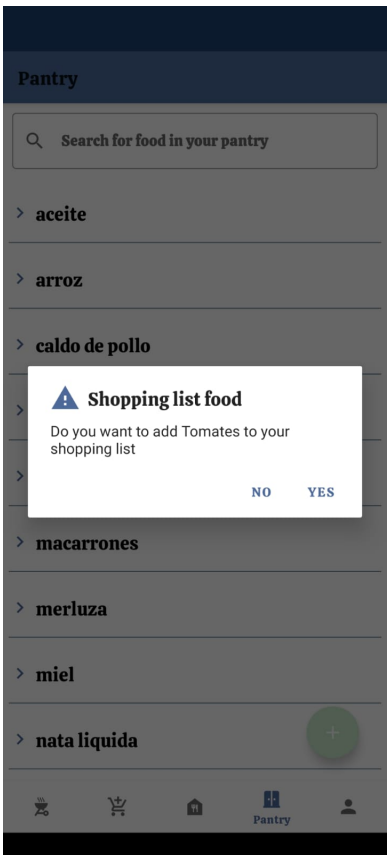


Figura 6.24: Diálogo añadir alimento lista de la compra

```
foodCardView.setOnClickListener(v -> new AlertDialog.Builder(context)
        .setTitle("Delete food")
        .setMessage("Do you really want to remove " + foodItemView.getText().toString() + " from your pantry")
        .setIcon(R.drawable.ic_warning)
        .setPositiveButton(android.R.string.yes, (dialog, whichButton) -> {
            assert user != null;
            myRef.child( pathString: "pantry").child(user.getUid()).child(name).removeValue();
            Toast.makeText(context,  text: foodItemView.getText().toString() + " has been removed from your pantry", Toast.LENGTH_SHORT).show();
            new AlertDialog.Builder(context)
                    .setTitle("Shopping list food")
                    .setMessage("Do you want to add " + foodItemView.getText().toString() + " to your shopping list")
                    .setIcon(R.drawable.ic_warning)
                    .setPositiveButton( text: "Yes", (dialog_shopping, whichButton_shopping) -> {
                        myRef.child( pathString: "shopping").child(user.getUid()).child( pathString: "list").child(name).setValue(amount);
                        Toast.makeText(context,  text: foodItemView.getText().toString() + " has been added to your shopping list", Toast.LENGTH_SHORT).show();
                    })
                    .setNegativeButton( text: "No",  listener: null).show();
        })
        .setNegativeButton(android.R.string.no,  listener: null).show());
```

Figura 6.25: Fragmento de código (eliminar alimento despensa y añadir a la lista)

6.2.3. Modificar alimento de la despensa.

El usuario es redirigido a esta vista una vez pulse sobre un ingrediente localizado en la vista principal de la despensa. La vista relativa a modificar alimento de la despensa se caracteriza por dos campos fundamentalmente, el nombre y la cantidad del ingrediente seleccionado, ilustrado en la

figura 6.26. El usuario puede modificar estos campos gracias a que son editables. Se han realizado comprobaciones para que los datos no puedan estar vacíos tras la modificación. Para completar la modificación del alimento, el usuario debe guardar la información mediante el botón situado en la parte inferior de la vista. Una vez pulsado, el sistema muestra un diálogo con la finalidad de confirmar el deseo de modificar los datos, tal y como refleja la figura 6.27. Si el usuario cancela la acción, se cierra el diálogo. Sin embargo, en caso de corroborar la modificación, el sistema muestra un mensaje confirmando los cambios efectuados, por tanto, se actualiza la información de la base de datos y se redirecciona al usuario a la vista principal de la despensa.

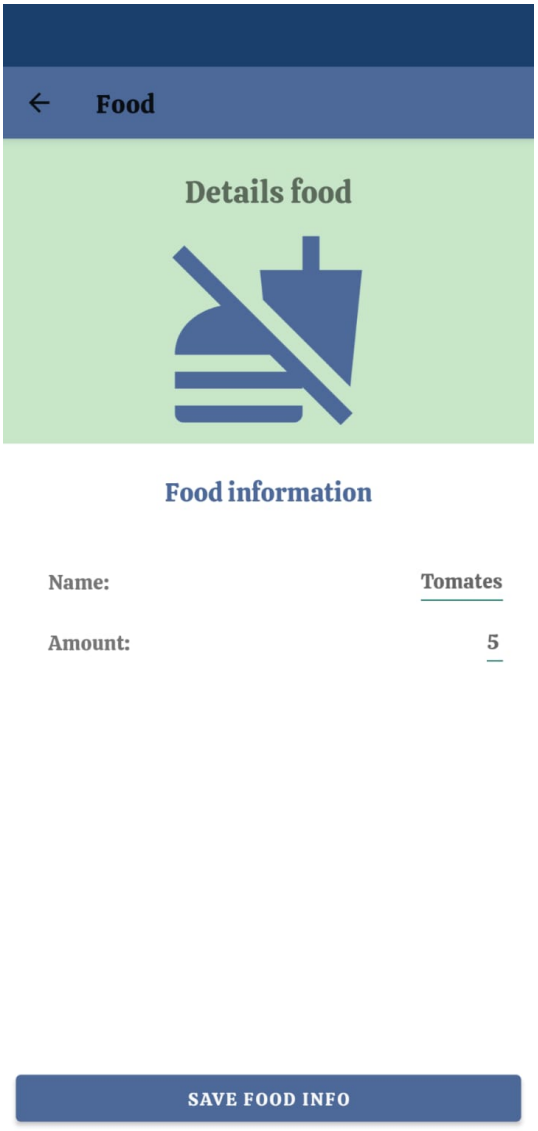


Figura 6.26: Pantalla informacion alimento

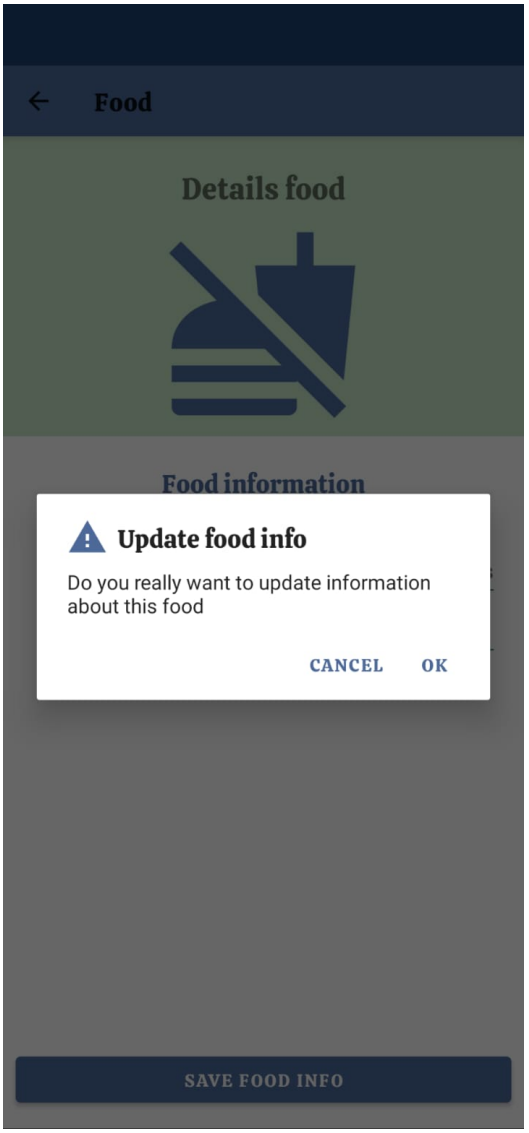


Figura 6.27: Modificar alimento de la despensa

```
binding.saveFood.setOnClickListener(v -> new AlertDialog.Builder( context: this)
        .setTitle("Update food info")
        .setMessage("Do you really want to update information about this food")
        .setIcon(R.drawable.ic_warning)
        .setPositiveButton(android.R.string.yes, (dialog, whichButton) -> {
            String uid = FirebaseAuth.getInstance().getCurrentUser().getUid();
            String food_name = binding.nameFoodFill.getText().toString();
            String food_amount = binding.amountFoodFill.getText().toString();
            if (!food_name.equals("") && !food_amount.equals("")) {
                myRef.child( pathString: "pantry").child(uid).child(name).removeValue();
                myRef.child( pathString: "pantry").child(uid).child(food_name).setValue(food_amount);
                Toast.makeText( context: this,  text: "Name and amount have been update", Toast.LENGTH_SHORT).show();
            } else if (!food_amount.equals("")) {
                myRef.child( pathString: "pantry").child(uid).child(name).setValue(food_amount);
                Toast.makeText( context: this,  text: "Amount has been update", Toast.LENGTH_SHORT).show();
            } else {
                myRef.child( pathString: "pantry").child(uid).child(name).removeValue();
                myRef.child( pathString: "pantry").child(uid).child(food_name).setValue(binding.amountFoodFill.getHint());
                Toast.makeText( context: this,  text: "Name has been update", Toast.LENGTH_SHORT).show();
            }
            finish();
        })
        .setNegativeButton(android.R.string.no,  listener: null).show());
```

Figura 6.28: Fragmento de código (modificar alimento de la despensa)

6.2.4. Buscar alimento en la despensa.

La vista principal de la despensa virtual se caracteriza por contar con un buscador en la parte superior de la vista, tal y como refleja la figura 6.18. El usuario puede realizar búsquedas escribiendo carácteres en dicho buscador. El sistema le muestra resultados coincidentes con la búsqueda, tal y como refleja la figura 6.29.

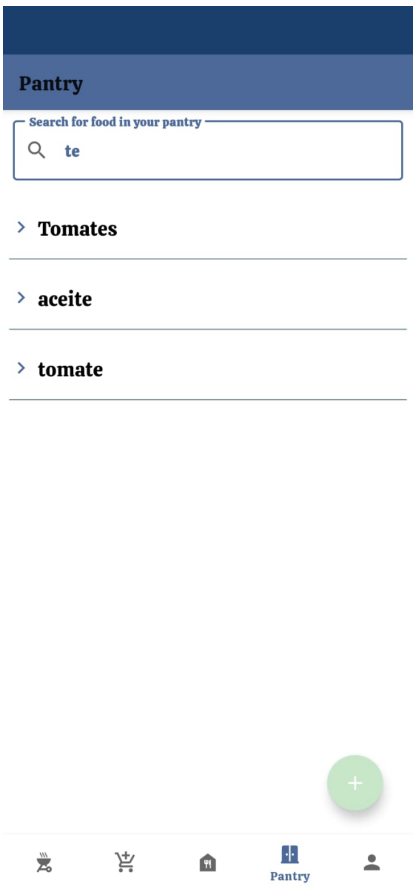


Figura 6.29: Pantalla buscar alimento

```
TextWatcher afterTextChangedListener = new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
    }

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
    }

    @Override
    public void afterTextChanged(Editable s) {
        myRef.child( pathString: "pantry").child(user.getUid()).get().addOnCompleteListener(new OnCompleteListener<DataSnapshot>() {
            @Override
            public void onComplete(@NonNull Task<DataSnapshot> task) {
                if (task.isSuccessful()) {
                    foodList.clear();
                    for (DataSnapshot foodSnapshot : task.getResult().getChildren()) {
                        String name = foodSnapshot.getKey();
                        String amount = foodSnapshot.getValue(String.class);
                        Food food = new Food(name, amount);
                        if (!foodList.contains(food) && food.getName().toLowerCase(Locale.ROOT).contains(Objects.requireNonNull(binding.editSearch.getText()).toString().toLowerCase(Locale.ROOT))) {
                            foodList.addLast(food);
                        }
                    }
                    mAdapter = new FoodAdapter(context, foodList);
                    recyclerView.setAdapter(mAdapter);
                    recyclerView.setLayoutManager(new LinearLayoutManager(context));
                }
            }
        });
    }
};

binding.editSearch.addTextChangedListener(afterTextChangedListener);
```

Figura 6.30: Fragmento de código (buscar alimento de la despensa)

6.2.5. Ver alimento de la despensa.

Para realizar la acción de añadir alimento se debe partir de la vista principal de la despensa, reflejada en la figura 6.18. La vista relativa a ver los alimentos de la despensa se caracteriza por dos campos fundamentalmente, el nombre y la cantidad del ingrediente seleccionado, ilustrado en la figura 6.26.

```
myRef.child( pathString: "pantry").child(FirebaseAuth.getInstance().getCurrentUser().getUid()).addValueEventListener(new ValueEventListener() {

    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        for (DataSnapshot foodSnapshot : snapshot.getChildren()) {
            String name_food = foodSnapshot.getKey();
            if (name_food.equals(name)) {
                binding.nameFoodFill.setHint(name);
                binding.amountFoodFill.setHint(foodSnapshot.getValue(String.class));
            }
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        Toast.makeText(getApplicationContext(),  text: "Error loading food information.", Toast.LENGTH_LONG).show();
    }
});
```

Figura 6.31: Fragmento de código (ver alimento de la despensa)

6.3. Módulo gestionar recetas

6.3.1. Añadir receta.

Para realizar la acción de añadir receta se debe partir de la vista principal de recetas, reflejada en la figura 6.32. El usuario debe seleccionar el icono situado en la esquina inferior derecha, caracterizado por un signo “+”, para ser redirigido a la vista correspondiente a la acción añadir receta. Únicamente el usuario registrado puede realizar esta acción. Se caracteriza por una serie de campos a cumplimentar, recayendo en el nombre, descripción, una lista de ingredientes caracterizados por nombre y cantidad, además, existen campos a cumplimentar relativos a la imagen de la receta y a su carácter público o privado, tal y como se ilustra en la figura 6.33. La lista de ingredientes puede ser incrementada o decrementada, tal y como se refleja en la figura 6.34. Se ha desarrollado un control, de tal forma que la acción de añadir, no se vea activada hasta que los datos

hayan sido rellenados. Una vez cumplimentados los datos, se almacena la información en la base de datos y se redirecciona al usuario a la vista principal de las recetas.

Figura 6.32: Pantalla principal de recetas

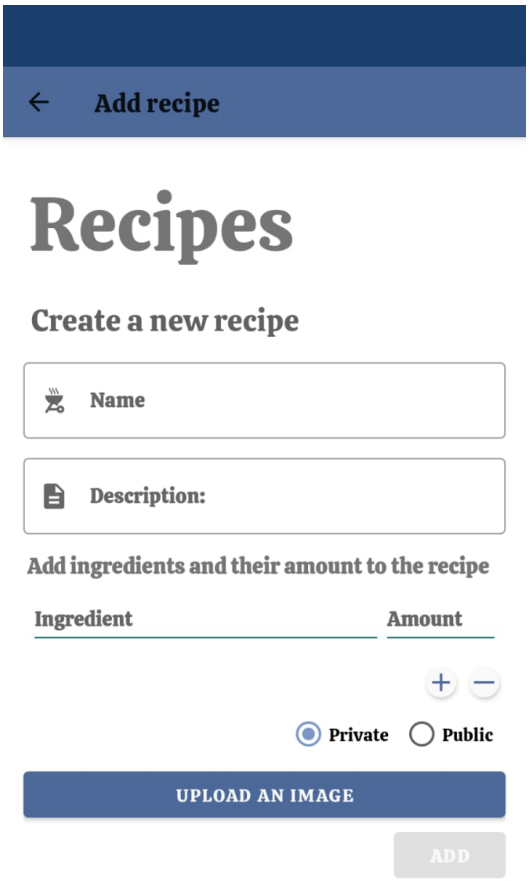


Figura 6.33: Pantalla de añadir receta

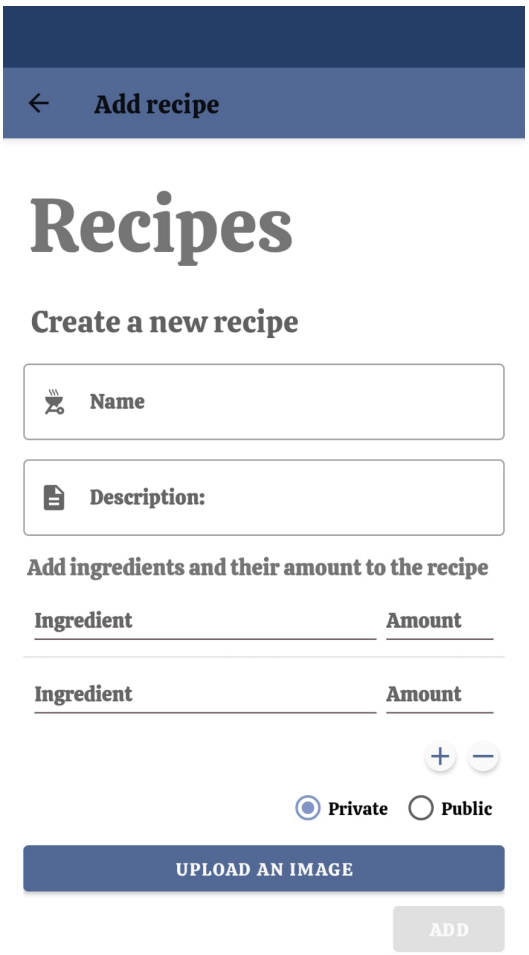


Figura 6.34: Pantalla añadir ingredientes a la receta

```
myRef.child( pathString: "recipes").addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        List<Recipe> list_aux = new ArrayList<>();
        for (DataSnapshot idSnapshot : snapshot.getChildren()) {
            if (Objects.equals(idSnapshot.getKey(),  b: "public"))
                for (DataSnapshot recipeSnapshot : idSnapshot.getChildren())
                    for (DataSnapshot r : recipeSnapshot.getChildren()) {
                        Recipe recipe = new Recipe(r.child( path: "name").getValue(String.class), r.child( path: "description").getValue(String.class),
                                (List<Pair<String, String>>) r.child( path: "list").getValue(), r.child( path: "type").getValue(String.class));
                        list_aux.add(recipe);
                    }
            else {
                assert user != null;
                if (Objects.equals(idSnapshot.getKey(), user.getUid()))
                    for (DataSnapshot r : idSnapshot.getChildren()) {
                        Recipe recipe = new Recipe(r.child( path: "name").getValue(String.class), r.child( path: "description").getValue(String.class),
                                (List<Pair<String, String>>) r.child( path: "list").getValue(), r.child( path: "type").getValue(String.class));
                        list_aux.add(recipe);
                    }
            }
        }

        for (Recipe r : list_aux)
            if (!recipeList.contains(r)) recipeList.add(r);

        Iterator<Recipe> iterator = recipeList.iterator();

        while (iterator.hasNext()) {
            Recipe r = iterator.next();
            if (r.getType().equals("private") && !list_aux.contains(r)) iterator.remove();
        }

        mAdapter = new RecipeAdapter(context, recipeList);
        recyclerView.setAdapter(mAdapter);
        recyclerView.setLayoutManager(new LinearLayoutManager(context));
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
    }
});
```

Figura 6.35: Fragmento de código (recetas)

```
addButton.setOnClickListener(v -> {

    Recipe recipe = new Recipe(nameEditText.getText().toString(), descriptionEditText.getText().toString(),
            adapter.getList(), checked.toLowerCase(Locale.ROOT));

    DatabaseReference myRef = FirebaseDatabase.getInstance().getReference();
    FirebaseUser user = mAuth.getCurrentUser();
    StorageReference storageRef = FirebaseStorage.getInstance().getReference();
    StorageReference recipeRef = storageRef.child(recipe.getName());

    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    selectedImageBitmap.compress(Bitmap.CompressFormat.JPEG,  quality: 100, baos);
    byte[] data = baos.toByteArray();

    if (checked.equals("Public")) {
        myRef.child( pathString: "recipes").child( pathString: "public").get().addOnCompleteListener(task -> {
            List<Recipe> recipesList = new ArrayList<>();
            for (DataSnapshot userSnapshot : task.getResult().getChildren()) {
                for (DataSnapshot recipeSnapshot : userSnapshot.getChildren()) {
                    Recipe recipe1 = new Recipe(recipeSnapshot.child( path: "name").getValue(String.class), recipeSnapshot.child( path: "description").getValue(String.class),
                            (List<Pair<String, String>>) recipeSnapshot.child( path: "list").getValue(), recipeSnapshot.child( path: "type").getValue(String.class));
                    recipesList.add(recipe1);
                }
            }
            if (!recipesList.contains(recipe.getName())) {
                UploadTask uploadTask = recipeRef.putBytes(data);
                uploadTask.addOnFailureListener(exception -> Toast.makeText(getApplicationContext(),  text: "An error has occurred while uploading the recipe ", Toast.LENGTH_SHORT).show()).addOnSuccessListener(taskSnapshot -> {
                    assert user != null;
                    myRef.child( pathString: "recipes").child( pathString: "public").child(user.getUid()).child(recipe.getName()).setValue(recipe);
                    Toast.makeText(getApplicationContext(),  text: "The recipe: " + recipe.getName() + " has been added successfully", Toast.LENGTH_SHORT).show();
                    finish();
                });

            } else
                Toast.makeText(getApplicationContext(),  text: "The recipe: " + recipe.getName() + " already exists", Toast.LENGTH_SHORT).show();
        });
    } else {
        assert user != null;
        myRef.child( pathString: "recipes").child(user.getUid()).get().addOnCompleteListener(task -> {
            List<Recipe> recipesList = new ArrayList<>();
            for (DataSnapshot recipeSnapshot : task.getResult().getChildren()) {
                Recipe recipe12 = new Recipe(recipeSnapshot.child( path: "name").getValue(String.class), recipeSnapshot.child( path: "description").getValue(String.class),
                        (List<Pair<String, String>>) recipeSnapshot.child( path: "list").getValue(), recipeSnapshot.child( path: "type").getValue(String.class));
                recipesList.add(recipe12);
            }
            if (!recipesList.contains(recipe.getName())) {
                UploadTask uploadTask = recipeRef.putBytes(data);
                uploadTask.addOnFailureListener(exception -> Toast.makeText(getApplicationContext(),  text: "An error has occurred while uploading the recipe ", Toast.LENGTH_SHORT).show()).addOnSuccessListener(taskSnapshot -> {
                    myRef.child( pathString: "recipes").child(user.getUid()).child(recipe.getName()).setValue(recipe);
                    Toast.makeText(getApplicationContext(),  text: "The recipe: " + recipe.getName() + " has been added successfully", Toast.LENGTH_SHORT).show();
                    finish();
                });
            } else
                Toast.makeText(getApplicationContext(),  text: "The recipe: " + recipe.getName() + " already exists", Toast.LENGTH_SHORT).show();
        });
    }
});
```

Figura 6.36: Fragmento de código (añadir receta)

```
binding.imageadd.setOnClickListener(v -> {
    list = adapter.getList();
    Pair<String, String> pair = new Pair<>("", "");
    list.add(pair);
    if (list.size() > 2) {
        ViewGroup.LayoutParams params = listView.getLayoutParams();
        params.height = 450;
        listView.setLayoutParams(params);
        listView.requestLayout();
    } else if (list.size() == 2) {
        ViewGroup.LayoutParams params = listView.getLayoutParams();
        params.height = 300;
        listView.setLayoutParams(params);
        listView.requestLayout();
    }
    adapter = new ListviewAdapter(getApplicationContext(), list);
    listView.setAdapter(adapter);
});

binding.imageremove.setOnClickListener(v -> {
    if (list.size() > 1) list.remove( index: list.size() - 1);
    if (list.size() > 2) {
        ViewGroup.LayoutParams params = listView.getLayoutParams();
        params.height = 450;
        listView.setLayoutParams(params);
        listView.requestLayout();
    } else if (list.size() == 2) {
        ViewGroup.LayoutParams params = listView.getLayoutParams();
        params.height = 300;
        listView.setLayoutParams(params);
        listView.requestLayout();
    } else {
        ViewGroup.LayoutParams params = listView.getLayoutParams();
        params.height = 150;
        listView.setLayoutParams(params);
        listView.requestLayout();
    }
    adapter = new ListviewAdapter(getApplicationContext(), list);
    listView.setAdapter(adapter);
});
```

Figura 6.37: Fragmento de código (añadir/eliminar ingrediente)

```java
@SuppressLint("SetTextI18n") ActivityResultLauncher<Intent> launchSomeActivity = registerForActivityResult(new ActivityResultContracts.StartActivityForResult(), result -> {
    if (result.getResultCode() == Activity.RESULT_OK) {
        Intent data = result.getData();
        if (data != null && data.getData() != null) {
            Uri selectedImageUri = data.getData();
            try {
                selectedImageBitmap = MediaStore.Images.Media.getBitmap(this.getContentResolver(), selectedImageUri);
                uploadButton.setText("Uploaded image");
                addButton.setEnabled(nameEditText.getText() != null && descriptionEditText.getText() != null &&
                        !nameEditText.getText().toString().equals("") && !descriptionEditText.getText().toString().equals("") && uploadButton.getText().toString().equals("Uploaded image")
                        && adapter.getEmpty());
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
});

uploadButton.setOnClickListener(v -> {
    Intent i = new Intent();
    i.setType("image/*");
    i.setAction(Intent.ACTION_GET_CONTENT);
    launchSomeActivity.launch(i);
});
```

Figura 6.38: Fragmento de código (cargar imagen)

6.3.2. Eliminar receta.

Para realizar esta acción, el usuario debe estar situado en la vista principal de las recetas. Para eliminar una receta se debe mantener pulsado sobre el nombre de esta, tal y como refleja la figura 6.39, apareciendo un icono relativo a una basura en el lateral derecho de la receta. Una vez el usuario haya pulsado el icono, el sistema muestra un diálogo de verificación, con el objetivo de corroborar que se desea eliminar la receta definitivamente, ilustrado en la figura 6.40. En el caso de cancelar la operación el usuario no es redirigido a ninguna otra vista. Si se procede con la operación, el sistema actualiza la base de datos y el usuario se mantiene situado en la vista principal de recetas.

Figura 6.39: Pantalla eliminar receta

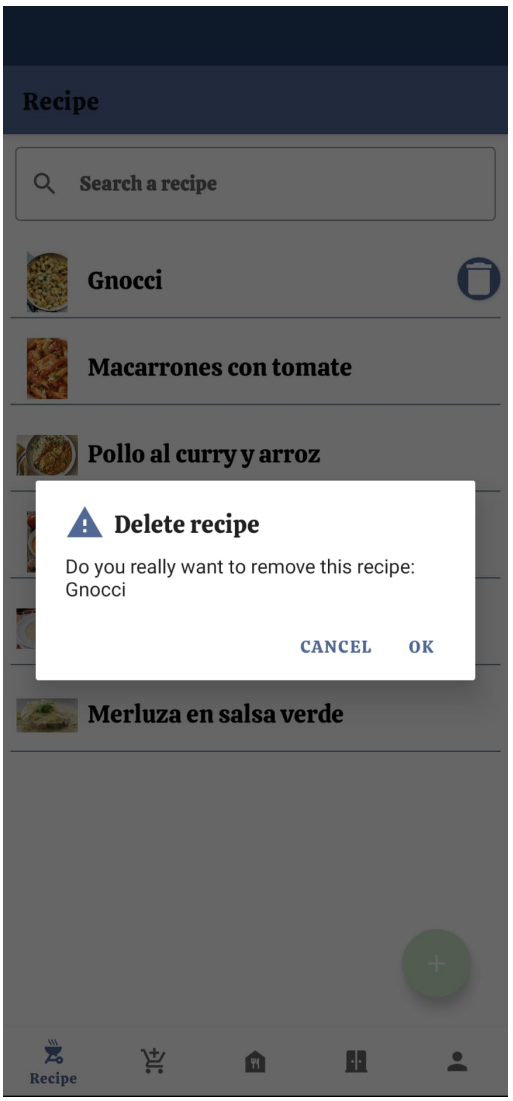


Figura 6.40: Diálogo eliminar receta

```
recipeCardView.setOnClickListener(v -> new AlertDialog.Builder(context)
        .setTitle("Delete recipe")
        .setMessage("Do you really want to remove this recipe: " + recipeItemView.getText().toString())
        .setIcon(R.drawable.ic_warning)
        .setPositiveButton(android.R.string.yes, (dialog, whichButton) -> {
            StorageReference storageRef = FirebaseStorage.getInstance().getReference().child(recipeItemView.getText().toString());
            storageRef.delete().addOnSuccessListener(new OnSuccessListener<Void>() {
                @Override
                public void onSuccess(Void aVoid) {
                    if (public_type)
                        myRef.child( pathString: "recipes").child( pathString: "public").child(user).child(recipeItemView.getText().toString()).removeValue();
                    else
                        myRef.child( pathString: "recipes").child(user).child(recipeItemView.getText().toString()).removeValue();
                    Toast.makeText(context,  text: recipeItemView.getText().toString() + " has been removed from the database", Toast.LENGTH_SHORT).show();
                }
            }).addOnFailureListener(exception -> Toast.makeText(context,  text: "An error has occurred while deleting the recipe", Toast.LENGTH_SHORT).show());

        })
        .setNegativeButton(android.R.string.no,  listener: null).show());
```

Figura 6.41: Fragmento de código (eliminar receta)

6.3.3. Modificar receta.

El usuario es redirigido a esta vista una vez pulse sobre la receta localizada en la vista principal de recetas. La vista relativa a modificar receta se caracteriza por una serie de campos, tales como el nombre, la descripción, la lista de ingredientes, el carácter o tipo de la receta y la imagen. Además, cuenta con un botón en la parte inferior que permite realizar la acción de modificar la receta, tal y como muestra la figura 6.42. Esta vista solo se muestra en caso de que la receta haya sido creada por el usuario. Se puede incrementar y decrementar el número de filas de la lista como se muestra en la figura 6.34. Una vez haya pulsado el usuario el botón relativo a modificar receta, el sistema muestra un diálogo que confirme la acción, tal y como se ilustra en la figura 6.43. En el caso de cancelar la acción, el usuario no es redirigido. En caso contrario, se actualiza la información en la base de datos y se redirige al usuario a la vista principal de recetas.

Figura 6.42: Pantalla receta privada

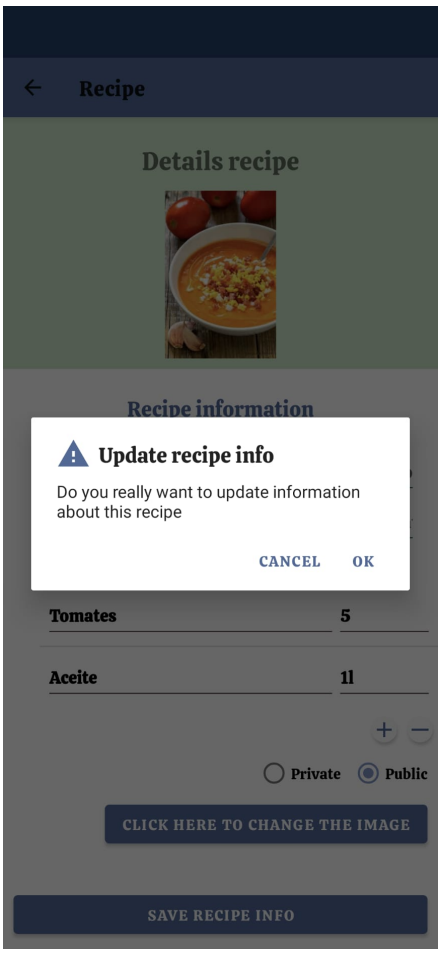


Figura 6.43: Diálogo modificar receta

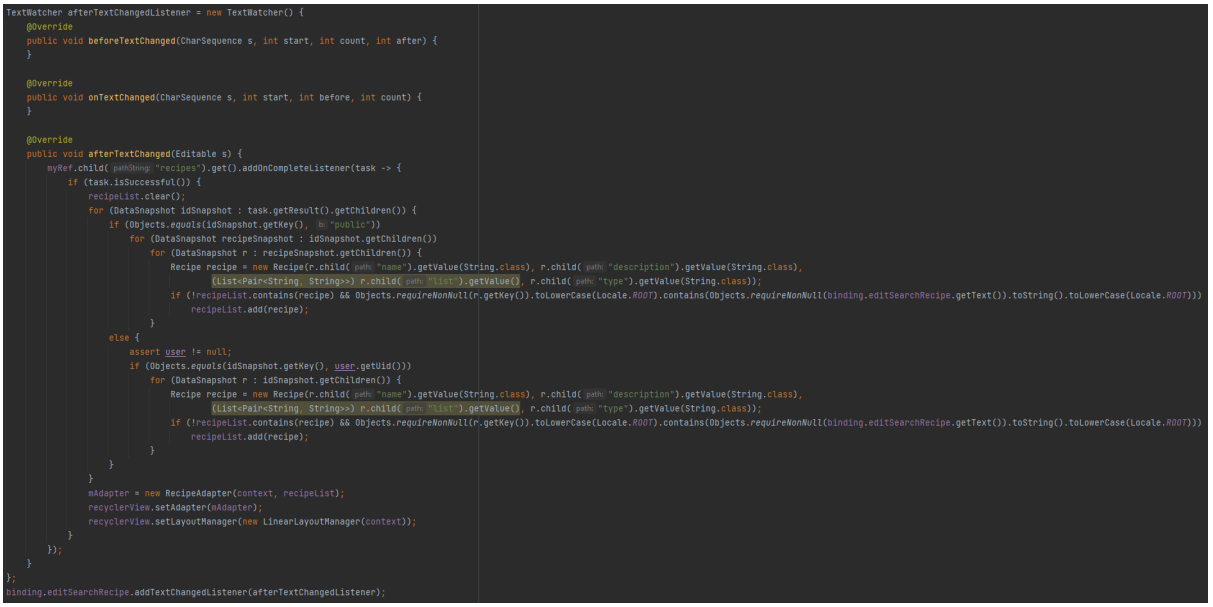
```
binding.saveRecipe.setOnClickListener(v -> new AlertDialog.Builder( context: this)
        .setTitle("Update recipe info")
        .setMessage("Do you really want to update information about this recipe")
        .setIcon(R.drawable.ic_warning)
        .setPositiveButton(android.R.string.yes, (dialog, whichButton) -> {
            Recipe recipe = new Recipe(Objects.requireNonNull(binding.nameRecipeFill.getText()).toString().equals("") ? Objects.requireNonNull(binding.nameRecipeFill.getHint()).toString() : binding.nameRecipeFill.getText().toString(),
                    Objects.requireNonNull(binding.descriptionRecipeFill.getText()).toString().equals("") ? Objects.requireNonNull(binding.descriptionRecipeFill.getHint()).toString() : binding.descriptionRecipeFill.getText().toString()
                    , adapter.getList(), checked.toLowerCase(Locale.ROOT));

            DatabaseReference myRef = FirebaseDatabase.getInstance().getReference();
            StorageReference storageRef = FirebaseStorage.getInstance().getReference();
            StorageReference recipeRef = storageRef.child(recipe.getName());

            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            selectedImageBitmap.compress(Bitmap.CompressFormat.JPEG,  quality: 100, baos);
            byte[] data = baos.toByteArray();


            UploadTask uploadTask = recipeRef.putBytes(data);
            if (type.equals("public")) {
                uploadTask.addOnFailureListener(exception -> Toast.makeText(getApplicationContext(),  text: "An error has occurred while updating the recipe ", Toast.LENGTH_SHORT).show()).addOnSuccessListener(taskSnapshot -> {
                    if (checked.equals("public") && recipe.getName().equals(name)) {
                        myRef.child( pathString: "recipes").child( pathString: "public").child(uid).child(recipe.getName()).setValue(recipe);
                    } else if (checked.equals("public")) {
                        myRef.child( pathString: "recipes").child( pathString: "public").child(uid).child(name).removeValue();
                        myRef.child( pathString: "recipes").child( pathString: "public").child(uid).child(recipe.getName()).setValue(recipe);
                        storageRef.child(name).delete();
                    } else {
                        myRef.child( pathString: "recipes").child( pathString: "public").child(uid).child(name).removeValue();
                        myRef.child( pathString: "recipes").child(uid).child(recipe.getName()).setValue(recipe);
                        if (!recipe.getName().equals(name))
                            storageRef.child(name).delete();
                    }
                });
            } else {
                uploadTask.addOnFailureListener(exception -> Toast.makeText(getApplicationContext(),  text: "An error has occurred while updating the recipe ", Toast.LENGTH_SHORT).show()).addOnSuccessListener(taskSnapshot -> {
                    if (checked.equals("private") && recipe.getName().equals(name)) {
                        myRef.child( pathString: "recipes").child(uid).child(recipe.getName()).setValue(recipe);
                    } else if (checked.equals("private")) {
                        myRef.child( pathString: "recipes").child(uid).child(name).removeValue();
                        myRef.child( pathString: "recipes").child(uid).child(recipe.getName()).setValue(recipe);
                        storageRef.child(name).delete();
                    } else {
                        myRef.child( pathString: "recipes").child(uid).child(name).removeValue();
                        myRef.child( pathString: "recipes").child( pathString: "public").child(uid).child(recipe.getName()).setValue(recipe);
                        if (!recipe.getName().equals(name))
                            storageRef.child(name).delete();
                    }
                });
            }
            finish();
        })
        .setNegativeButton(android.R.string.no,  listener: null).show());
```

Figura 6.44: Fragmento de código (modificar receta)

6.3.4. Buscar receta.

La vista principal de recetas se caracteriza por contar con un buscador en la parte superior de la vista, tal y como refleja la figura 6.32. El usuario puede realizar búsquedas escribiendo caracteres en dicho buscador. El sistema le muestra resultados coincidentes con la búsqueda, tal y como refleja la figura 6.45.

Figura 6.45: Pantalla buscar receta



```
TextWatcher afterTextChangedListener = new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
    }

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
    }

    @Override
    public void afterTextChanged(Editable s) {
        myRef.child( pathString: "recipes").get().addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
                recipeList.clear();
                for (DataSnapshot idSnapshot : task.getResult().getChildren()) {
                    if (Objects.equals(idSnapshot.getKey(),  b: "public"))
                        for (DataSnapshot recipeSnapshot : idSnapshot.getChildren())
                            for (DataSnapshot r : recipeSnapshot.getChildren()) {
                                Recipe recipe = new Recipe(r.child( path: "name").getValue(String.class), r.child( path: "description").getValue(String.class),
                                        (List<Pair<String, String>>) r.child( path: "list").getValue(), r.child( path: "type").getValue(String.class));
                                if (!recipeList.contains(recipe) && Objects.requireNonNull(r.getKey()).toLowerCase(Locale.ROOT).contains(Objects.requireNonNull(binding.editSearchRecipe.getText()).toString().toLowerCase(Locale.ROOT)))
                                    recipeList.add(recipe);
                            }
                    else {
                        assert user != null;
                        if (Objects.equals(idSnapshot.getKey(), user.getUid()))
                            for (DataSnapshot r : idSnapshot.getChildren()) {
                                Recipe recipe = new Recipe(r.child( path: "name").getValue(String.class), r.child( path: "description").getValue(String.class),
                                        (List<Pair<String, String>>) r.child( path: "list").getValue(), r.child( path: "type").getValue(String.class));
                                if (!recipeList.contains(recipe) && Objects.requireNonNull(r.getKey()).toLowerCase(Locale.ROOT).contains(Objects.requireNonNull(binding.editSearchRecipe.getText()).toString().toLowerCase(Locale.ROOT)))
                                    recipeList.add(recipe);
                            }
                    }
                }
                mAdapter = new RecipeAdapter(context, recipeList);
                recyclerView.setAdapter(mAdapter);
                recyclerView.setLayoutManager(new LinearLayoutManager(context));
            }
        });
    }
};
binding.editSearchRecipe.addTextChangedListener(afterTextChangedListener);
```

Figura 6.46: Fragmento de código (buscar receta)

6.3.5. Ver receta.

En el caso de que la receta haya sido creada por el usuario, la vista relativa a ver las recetas se caracteriza por el nombre, la descripción, la lista de ingredientes, el carácter o tipo de la receta y la imagen, ilustrado en la figura 6.42. En caso contrario, la vista se caracteriza por los campos relativos al nombre, descripción, lista de ingredientes y el carácter público de la receta, tal y como se ve reflejado en la figura 6.47.

Figura 6.47: Pantalla receta pública

```
myRef.child( pathString: "recipes").addValueEventListener(new ValueEventListener() {

    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        for (DataSnapshot idSnapshot : snapshot.getChildren()) {
            if (Objects.equals(idSnapshot.getKey(),  b: "public")) {
                for (DataSnapshot recipeSnapshot : idSnapshot.getChildren())
                    for (DataSnapshot r : recipeSnapshot.getChildren()) {
                        if (Objects.equals(r.child( path: "name").getValue(String.class), name)) {
                            binding.nameRecipeFill.setHint(r.child( path: "name").getValue(String.class));
                            binding.descriptionRecipeFill.setHint(r.child( path: "description").getValue(String.class));
                            if (Objects.equals(r.child( path: "type").getValue(String.class),  b: "public")) {
                                binding.radioPublic.setChecked(true);
                                binding.radioPublic.setVisibility(View.VISIBLE);
                            } else {
                                binding.radioPrivate.setChecked(true);
                                binding.radioPrivate.setVisibility(View.VISIBLE);
                            }
                            for (HashMap<String, String> m : (ArrayList<HashMap<String, String>>) Objects.requireNonNull(r.child( path: "list").getValue())) {
                                String n, a;
                                n = a = "";
                                boolean x = false;
                                for (Map.Entry<String, String> aux : m.entrySet()) {
                                    if (x) a = aux.getValue();
                                    else {
                                        n = aux.getValue();
                                        x = true;
                                    }
                                }
                                Pair<String, String> p = new Pair<>(n, a);
                                list.add(p);

                            }
                            if (list.size() > 2) {
                                ViewGroup.LayoutParams params = listView.getLayoutParams();
                                params.height = 300;
                                listView.setLayoutParams(params);
                                listView.requestLayout();
                            } else if (list.size() == 2) {
                                ViewGroup.LayoutParams params = listView.getLayoutParams();
                                params.height = 200;
                                listView.setLayoutParams(params);
                                listView.requestLayout();
                            } else {
                                ViewGroup.LayoutParams params = listView.getLayoutParams();
                                params.height = 100;
                                listView.setLayoutParams(params);
                                listView.requestLayout();
                            }
                            adapter = new ListviewPublicRecipeAdapter(getApplicationContext(), list);
                            listView.setAdapter(adapter);
```

Figura 6.48: Fragmento de código (ver receta 1)

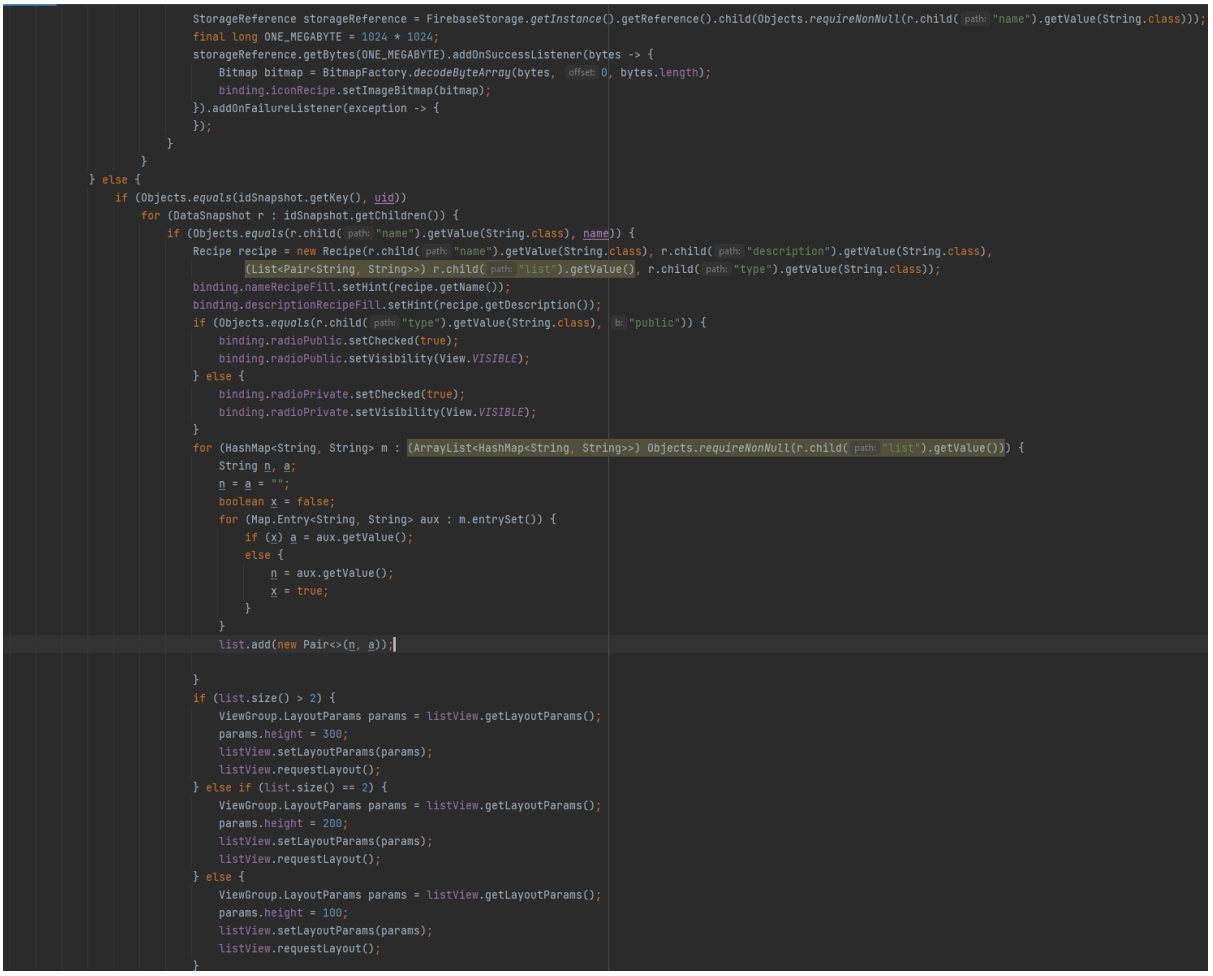


Figura 6.49: Fragmento de código (ver receta 2)

Figura 6.50: Fragmento de código (ver receta 3)

6.4. Módulo lista de la compra

6.4.1. Añadir alimento a la lista de la compra.

Para realizar la acción de añadir un alimento a la lista de la compra se debe partir de la vista principal de la lista de la compra, reflejada en la figura 6.51. El usuario debe seleccionar el icono situado en la esquina inferior derecha, caracterizado por un signo “+”, para ser redirigido a la vista correspondiente a la acción añadir alimento a la lista de la compra. Únicamente el usuario registrado puede realizar esta acción. Se caracteriza por dos campos a cumplimentar, recayendo en el nombre y cantidad del ingrediente, tal y como se ilustra en la figura 6.52. Se ha desarrollado un control, de tal forma que la acción de añadir, no se vea activada hasta que los datos hayan sido rellenados. Una vez cumplimentados los datos, se almacena la información en la base de datos y se redirecciona al usuario a la vista principal de la lista de la compra.

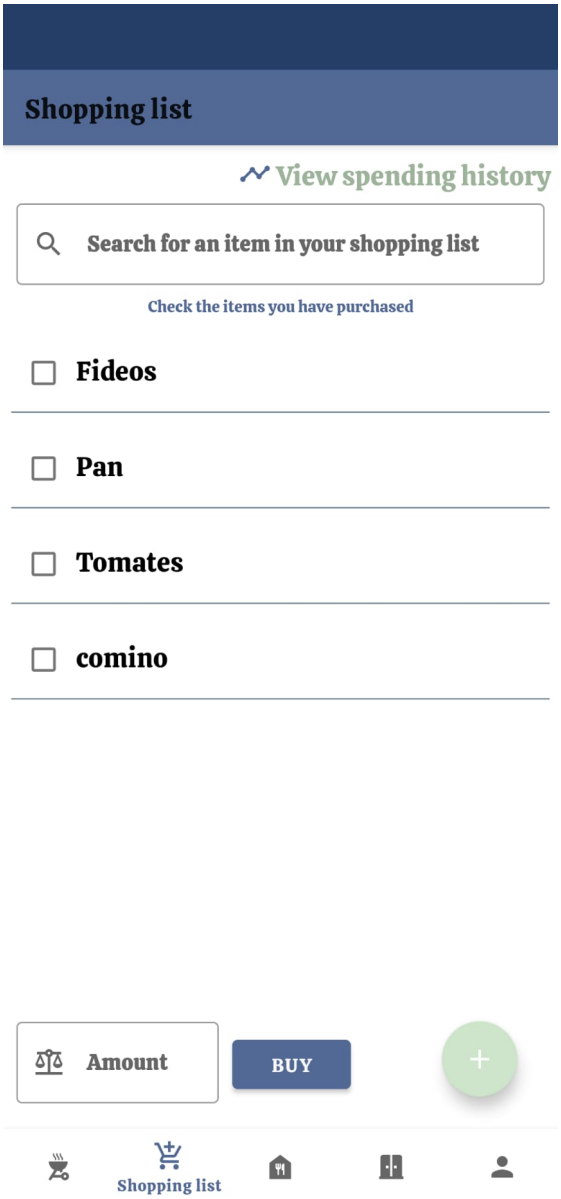


Figura 6.51: Pantalla lista de la compra

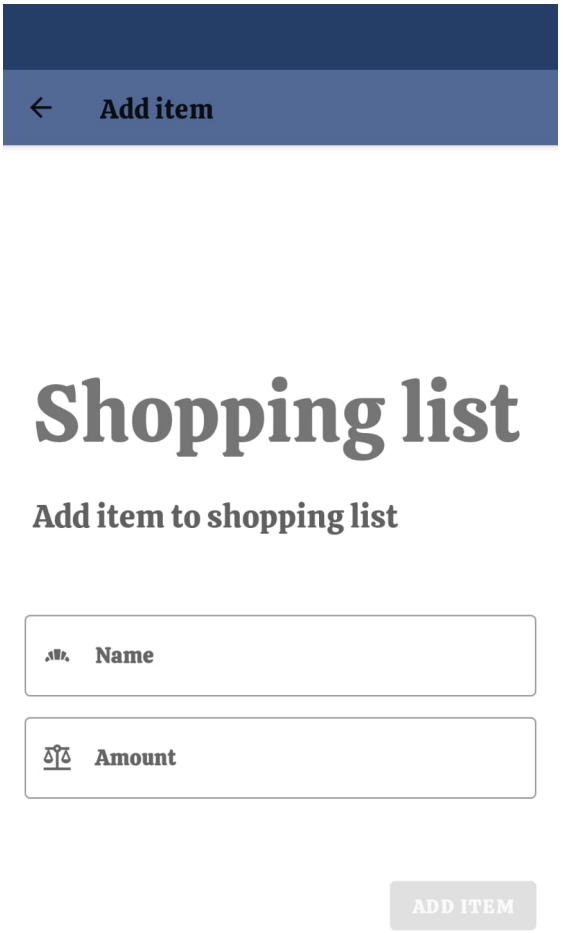


Figura 6.52: Pantalla añadir alimento a la lista de la compra

```
myRef.child( pathString: "shopping").child(user.getUid()).child( pathString: "list").addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        itemList.clear();
        for (DataSnapshot itemSnapshot : snapshot.getChildren()) {
            String name = itemSnapshot.getKey();
            String amount = itemSnapshot.getValue(String.class);
            Item item = new Item(name, amount);
            if (!itemList.contains(item)) {
                itemList.addLast(item);
            }
        }
        mAdapter = new ItemAdapter(context, itemList);
        recyclerView.setAdapter(mAdapter);
        recyclerView.setLayoutManager(new LinearLayoutManager(context));
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
    }
});
```

Figura 6.53: Fragmento de código (lista de la compra)

```
addButton.setOnClickListener(v -> {
    DatabaseReference myRef = FirebaseDatabase.getInstance().getReference();
    FirebaseUser user = mAuth.getCurrentUser();
    Item item = new Item(nameEditText.getText().toString(), amountEditText.getText().toString());
    assert user != null;
    myRef.child( pathString: "shopping").child(user.getUid()).child( pathString: "list").child(item.getName()).setValue(item.getAmount());
    Toast.makeText(getApplicationContext(),  text: "Se ha añadido " + item.getAmount() + " de " + item.getName(), Toast.LENGTH_SHORT).show();
    finish();
});
```

Figura 6.54: Fragmento de código (añadir alimento a la lista)

6.4.2. Eliminar alimento de la lista de la compra.

Para llegar a la vista de eliminar alimento de la lista de la compra, debe partirse de la vista principal de la lista de la compra. Una vez el usuario haya mantenido seleccionado un ingrediente se muestra la vista. La vista destaca de la principal por la aparición de un icono en el lateral derecho del nombre del ingrediente, tal y como se muestra en la figura 6.55. Una vez el usuario realiza la acción de eliminar, se abre un diálogo para corroborar que se desea eliminar definitivamente el alimento de la lista de la compra, reflejado en la figura 6.56. En caso de cancelar la acción, se cierra el diálogo y se mantiene al usuario en la misma vista. En caso de confirmar la eliminación, se actualiza la información en la base de datos y se redirige al usuario a la página principal de la lista de la compra.

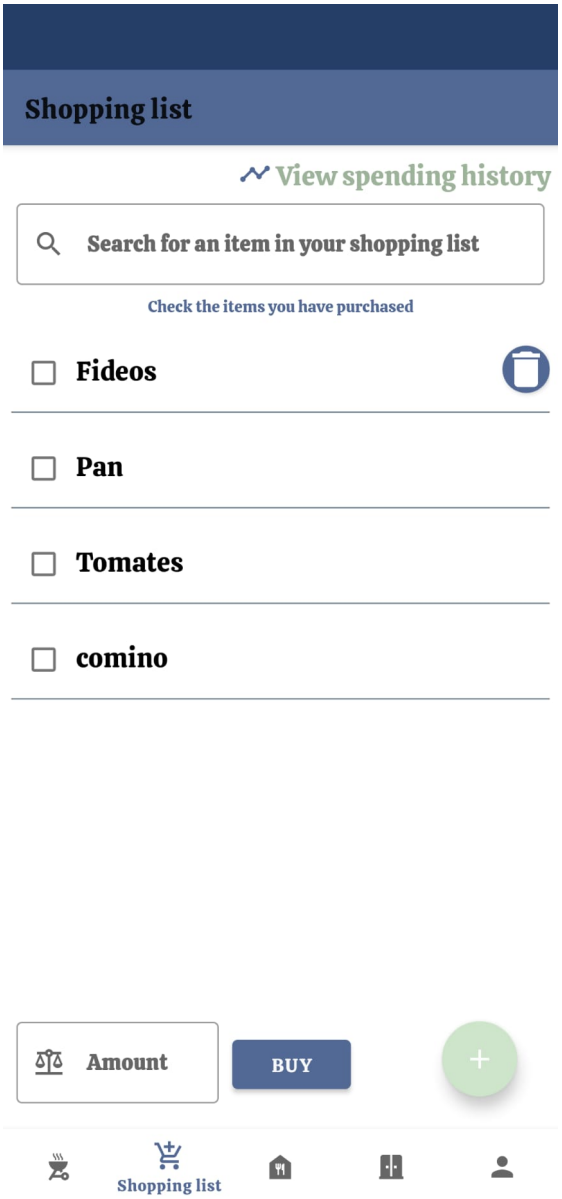


Figura 6.55: Pantalla eliminar alimentos de la lista

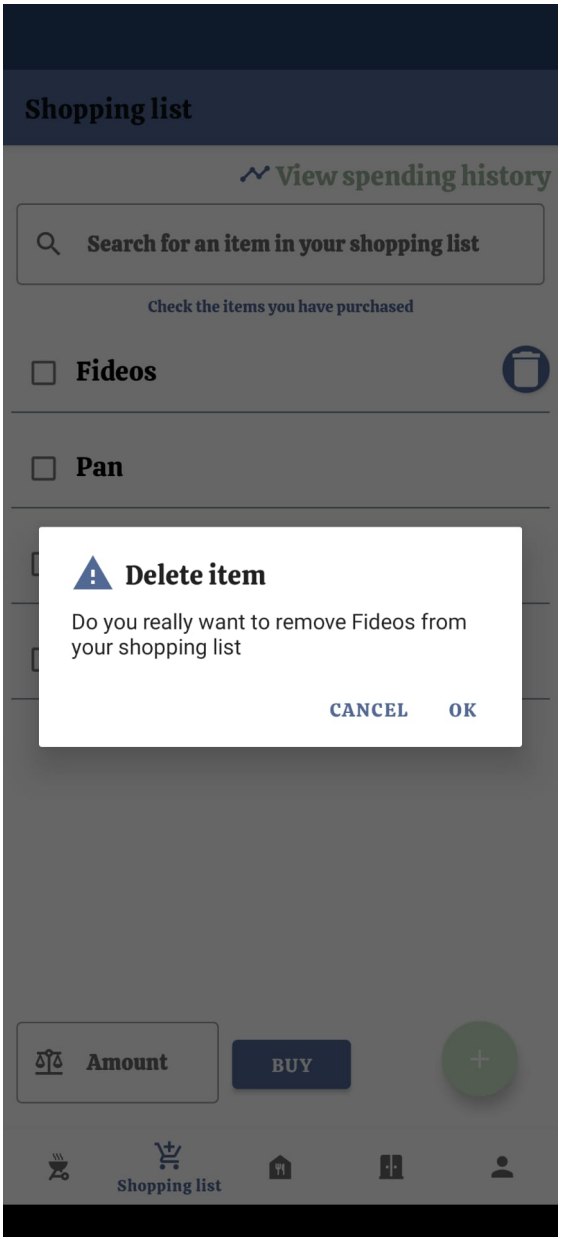


Figura 6.56: Diálogo eliminar alimento de la lista

```
itemCardView.setOnClickListener(v -> new AlertDialog.Builder(context)
        .setTitle("Delete item")
        .setMessage("Do you really want to remove " + itemView.getText().toString() + " from your shopping list")
        .setIcon(R.drawable.ic_warning)
        .setPositiveButton(android.R.string.yes, (dialog, whichButton) -> {
            DatabaseReference myRef = FirebaseDatabase.getInstance().getReference();
            myRef.child( pathString: "shopping").child(Objects.requireNonNull(FirebaseAuth.getInstance().getCurrentUser()).getUid()).child( pathString: "list").child(itemView.getText().toString()).removeValue();
            Toast.makeText(context, text: itemView.getText().toString() + " has been remove from your shopping list", Toast.LENGTH_SHORT).show();
        })
        .setNegativeButton(android.R.string.no, listener: null).show());
```

Figura 6.57: Fragmento de código (eliminar alimento de la lista)

6.4.3. Modificar alimento de la lista de la compra.

El usuario es redirigido a esta vista una vez pulse sobre el ingrediente localizado en la vista principal de lista de la compra. La vista relativa a modificar alimento se caracteriza por una serie de campos, tales como el nombre y cantidad. Además, cuenta con un botón en la parte inferior que permite realizar la acción de modificar el alimento, tal y como muestra la figura 6.58. Una vez haya pulsado el usuario el botón relativo a modificar alimento, el sistema muestra un diálogo que confirme la acción, tal y como se ilustra en la figura 6.59. En el caso de cancelar la acción, el usuario no es redirigido. En caso contrario, se actualiza la información en la base de datos y se redirige al usuario a la vista principal de lista de la compra.

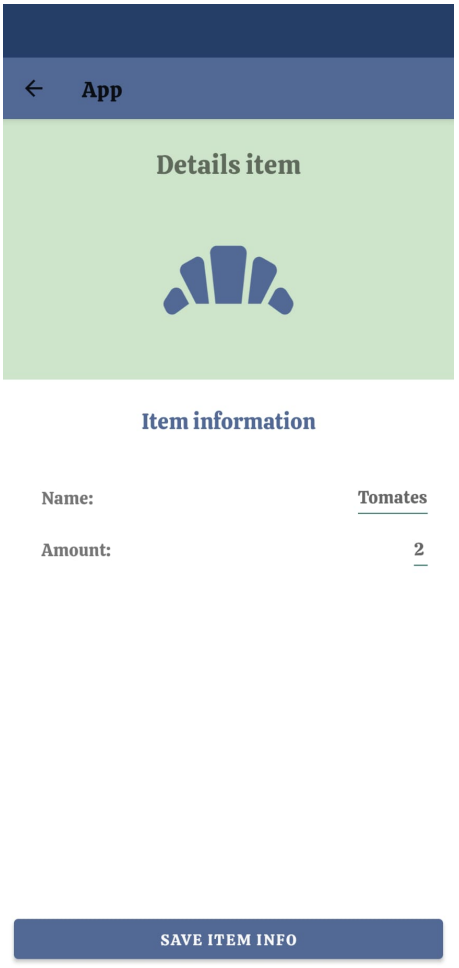


Figura 6.58: Pantalla modificar alimentos de la lista

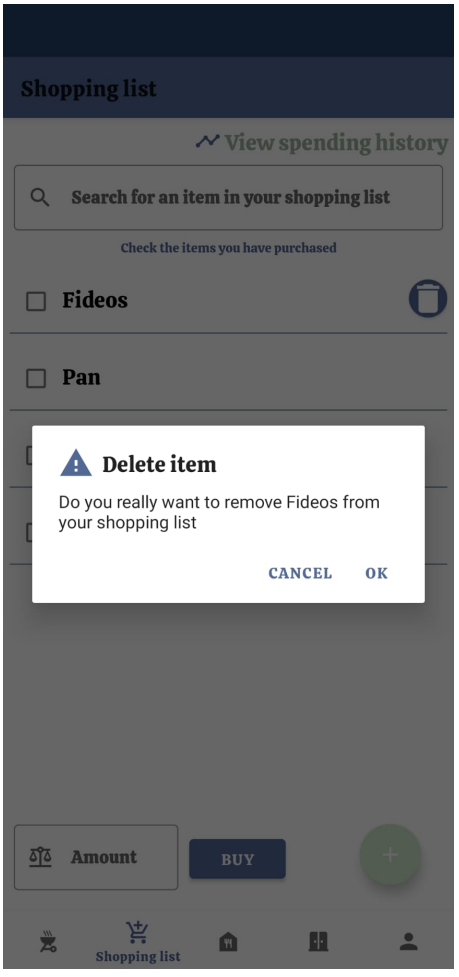


Figura 6.59: Diálogo modificar alimento de la lista

```
binding.saveItem.setOnClickListener(v -> new AlertDialog.Builder( context: this)
        .setTitle("Update item info")
        .setMessage("Do you really want to update information about this item")
        .setIcon(R.drawable.ic_warning)
        .setPositiveButton(android.R.string.yes, (dialog, whichButton) -> {
            String uid = FirebaseAuth.getInstance().getCurrentUser().getUid();
            String item_name = Objects.requireNonNull(binding.nameItemFill.getText()).toString();
            String item_amount = Objects.requireNonNull(binding.amountItemFill.getText()).toString();
            if (!item_name.equals("") && !item_amount.equals("")) {
                myRef.child( pathString: "shopping").child(uid).child( pathString: "list").child(name).removeValue();
                myRef.child( pathString: "shopping").child(uid).child( pathString: "list").child(item_name).setValue(item_amount);
                Toast.makeText( context: this,  text: "Name and amount have been update", Toast.LENGTH_SHORT).show();
            } else if (!item_amount.equals("")) {
                myRef.child( pathString: "shopping").child(uid).child( pathString: "list").child(name).setValue(item_amount);
                Toast.makeText( context: this,  text: "Amount has been update", Toast.LENGTH_SHORT).show();
            } else {
                myRef.child( pathString: "shopping").child(uid).child( pathString: "list").child(name).removeValue();
                myRef.child( pathString: "shopping").child(uid).child( pathString: "list").child(item_name).setValue(binding.amountItemFill.getHint());
                Toast.makeText( context: this,  text: "Name has been update", Toast.LENGTH_SHORT).show();
            }
            finish();
        })
        .setNegativeButton(android.R.string.no,  listener: null).show());
```

Figura 6.60: Fragmento de código (modificar alimento de la lista)

6.4.4. Generar lista de la compra.

Esta vista se produce cuando el usuario ha eliminado un alimento de su despensa. Muestra un diálogo con la finalidad de generar o no la lista de la compra incluyendo el alimento, tal y como se refleja en la figura 6.24. En caso de que el usuario quiera añadirlo a la lista de la compra, el sistema actualiza la información. En ambos casos, se redirige al usuario a la página principal de despensa. El código asociado a esto se ve reflejado en la figura 6.25.

6.4.5. Comprar lista.

La vista comprar lista coincide con la vista principal de lista de la compra 6.61. Se caracteriza por estar conformada por un conjunto de ingredientes pudiendo ser marcados como comprados. Además, se localiza un campo editable correspondiente al importe de la compra. La acción de comprar se realiza gracias a un botón. Se han implementado controles en caso de no haber seleccionado alimentos y no introducir ningún importe, mostrando el correspondiente error.

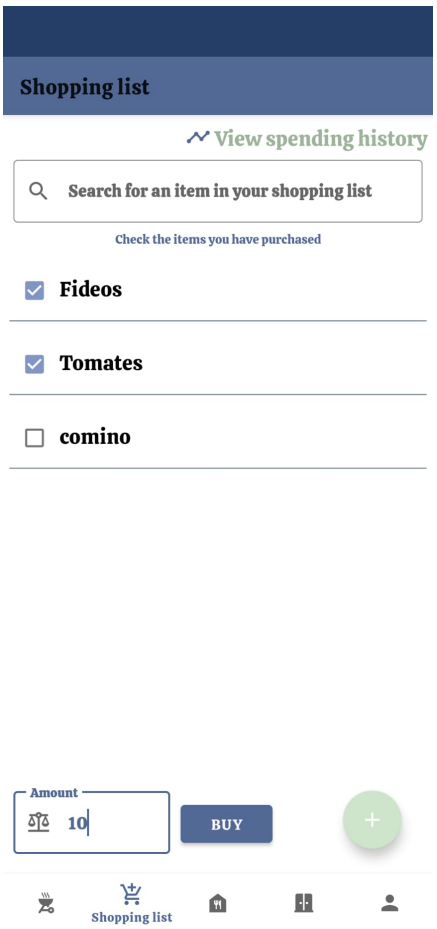


Figura 6.61: Pantalla comprar lista

```
binding.buyButton.setOnClickListener(v -> {
    String amount = Objects.requireNonNull(binding.editAmount.getText()).toString();
    if (amount.isEmpty())
        Toast.makeText(context, text: "Please enter the corresponding amount", Toast.LENGTH_SHORT).show();
    else {
        LinkedList<Item> list = mAdapter.getList();
        boolean any = false;
        for (Item i : list) {
            if (i.getCheck()) {
                any = true;
                myRef.child( pathString: "shopping").child(user.getUid()).child( pathString: "list").child(i.getName()).removeValue();
                myRef.child( pathString: "pantry").child(user.getUid()).child(i.getName()).setValue(i.getAmount());
            }
        }
        if (!any)
            Toast.makeText(context, text: "Please select at least 1 item", Toast.LENGTH_SHORT).show();
        else {
            Calendar calendar = Calendar.getInstance();
            String date = calendar.get(Calendar.DAY_OF_MONTH) + "-" + (calendar.get(Calendar.MONTH) + 1) + "-" + calendar.get(Calendar.YEAR);
            myRef.child( pathString: "shopping").child(user.getUid()).child( pathString: "expenses").child(date).get().addOnCompleteListener(task -> {
                if (task.getResult().getValue() != null)
                    myRef.child( pathString: "shopping").child(user.getUid()).child( pathString: "expenses").child(date).setValue(Integer.parseInt(amount) + task.getResult().getValue(Float.class));
                else
                    myRef.child( pathString: "shopping").child(user.getUid()).child( pathString: "expenses").child(date).setValue(Integer.parseInt(amount));
            });
            Toast.makeText(context, text: "The shopping list has been bought", Toast.LENGTH_SHORT).show();
            binding.editAmount.setText("");
        }
    }
});
```

Figura 6.62: Fragmento de código (comprar lista)

6.4.6. Ver estadísticas de gasto.

Para acceder a la vista ver estadísticas de gasto, el usuario debe haber pulsado en “view spending history” en la página principal de lista de la compra. Esta vista se caracteriza por mostrar un gráfico. Este gráfico se crea en función de un rango temporal elegido por el usuario, tal y como se muestra en la figura 6.63.

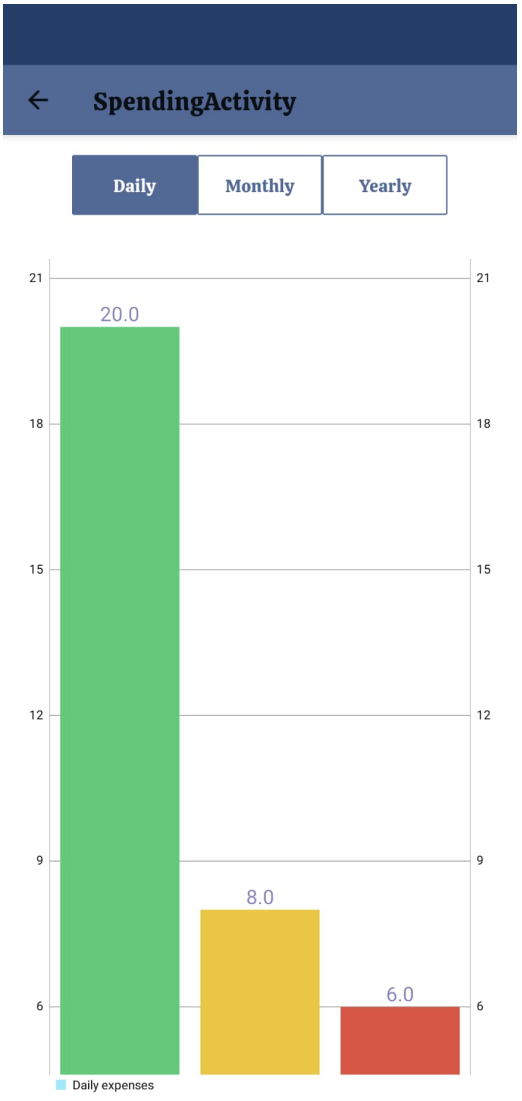


Figura 6.63: Pantalla historial de gasto

```
barEntriesArrayList = new ArrayList<>();
myRef.child( pathString: "shopping").child(uid).child( pathString: "expenses").get().addOnCompleteListener(task -> {
    String name = "";
    if (task.isSuccessful()) {
        if (interval.equals("Daily")) {
            float pos_x = 1;
            for (DataSnapshot dataSnapshot : task.getResult().getChildren()) {
                barEntriesArrayList.add(new BarEntry(pos_x, dataSnapshot.getValue(Float.class)));
                pos_x++;
            }
            name = "Daily expenses";
        } else if (interval.equals("Monthly")) {
            float pos_x = 1;
            float sum = 0;
            String month = Objects.requireNonNull(task.getResult().getChildren().iterator().next().getKey()).split( regex: "-")[1];
            for (DataSnapshot dataSnapshot : task.getResult().getChildren()) {
                String[] date = Objects.requireNonNull(dataSnapshot.getKey()).split( regex: "-");
                if (month.equals(date[1])) sum += dataSnapshot.getValue(Float.class);
                else {
                    barEntriesArrayList.add(new BarEntry(pos_x, sum));
                    pos_x++;
                    sum = dataSnapshot.getValue(Float.class);
                    month = dataSnapshot.getKey().split( regex: "-")[1];
                }
            }
            barEntriesArrayList.add(new BarEntry(pos_x, sum));
            name = "Monthly expenses";
        } else {
            float pos_x = 1;
            float sum = 0;
            String month = Objects.requireNonNull(task.getResult().getChildren().iterator().next().getKey()).split( regex: "-")[2];
            for (DataSnapshot dataSnapshot : task.getResult().getChildren()) {
                String[] date = Objects.requireNonNull(dataSnapshot.getKey()).split( regex: "-");
                if (month.equals(date[2])) sum += dataSnapshot.getValue(Float.class);
                else {
                    barEntriesArrayList.add(new BarEntry(pos_x, sum));
                    pos_x++;
                    sum = dataSnapshot.getValue(Float.class);
                    month = dataSnapshot.getKey().split( regex: "-")[2];
                }
            }
            barEntriesArrayList.add(new BarEntry(pos_x, sum));
            name = "Yearly expenses";
        }
```

Figura 6.64: Fragmento de código (historial de gastos 1)

```
            barDataSet = new BarDataSet(barEntriesArrayList, label: "Data user");
            barData = new BarData(barDataSet);
            barChart.setData(barData);
            barDataSet.setColors(ColorTemplate.MATERIAL_COLORS);
            barDataSet.setValueTextColor(R.color.primary_blue);
            barDataSet.setValueTextSize(16f);
            barChart.getDescription().setEnabled(false);
        } else Toast.makeText( context: SpendingActivity.this, text: "No expenses", Toast.LENGTH_SHORT).show();
        barDataSet = new BarDataSet(barEntriesArrayList, name);
        barData = new BarData(barDataSet);
        barChart.setData(barData);
        barDataSet.setColors(ColorTemplate.MATERIAL_COLORS);
        barDataSet.setValueTextColor(R.color.primary_blue);
        barDataSet.setValueTextSize(16f);
        barChart.getDescription().setEnabled(false);
        barChart.invalidate();
    });
}
```

Figura 6.65: Fragmento de código (historial de gastos 2)

6.5. Módulo gestionar comidas

6.5.1. Elegir comida.

Previamente a la vista de elegir comida, se encuentra la vista principal de menú. La vista menú contiene tanto el nombre de la receta como la imagen, tal y como se representa en la figura 6.66. Una vez se selecciona la receta a realizar, se redirige al usuario a la vista elegir comida. En esta vista se localiza información sobre la receta como son el nombre, la descripción, la lista de ingredientes y el carácter o tipo de la receta, tal y como se refleja en la figura 6.47. Esta vista se caracteriza por localizar un botón en la parte inferior, utilizado para marcar como realizada. Una vez se haya marcado como realizada, el sistema actualiza la información de la base de datos y redirige al usuario a la página principal de menú.

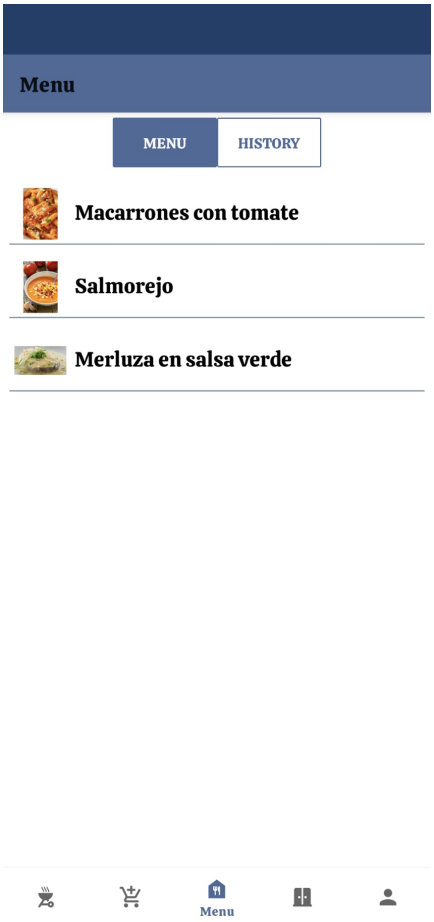


Figura 6.66: Pantalla menú de comidas

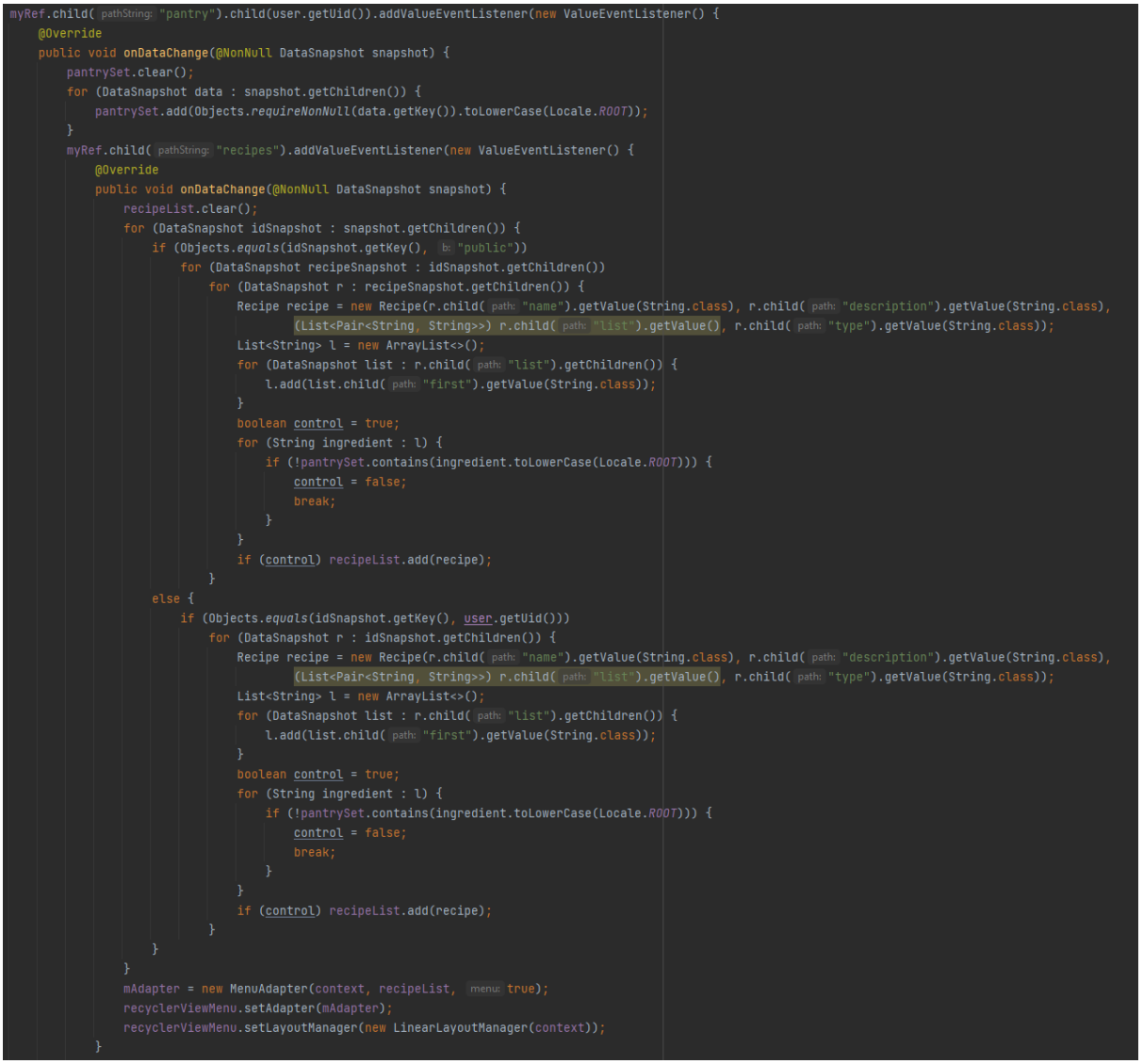


Figura 6.67: Fragmento de código (menú)

6.5.2. Ver historial de comidas.

Para acceder a esta vista debe haberse seleccionado la pestaña “history” en la vista principal del menú. Esta vista se caracteriza por estar conformada por un conjunto de recetas ya realizadas, tal y como se muestra en la figura 6.68. Se conforma por un conjunto de nombres e imágenes de dichas recetas.

Figura 6.68: Pantalla menú de comidas

```
myRef.child( pathString: "menu").child(user.getUid()).addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot _snapshot) {
        myRef.child( pathString: "recipes").addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                recipeHistory.clear();
                List<Recipe> recipes = new ArrayList<>();
                for (DataSnapshot idSnapshot : snapshot.getChildren()) {
                    if (Objects.equals(idSnapshot.getKey(),  b: "public"))
                        for (DataSnapshot recipeSnapshot : idSnapshot.getChildren())
                            for (DataSnapshot r : recipeSnapshot.getChildren()) {
                                Recipe recipe = new Recipe(r.child( path: "name").getValue(String.class), r.child( path: "description").getValue(String.class),
                                        (List<Pair<String, String>>) r.child( path: "list").getValue(), r.child( path: "type").getValue(String.class));
                                recipes.add(recipe);
                            }
                    else {
                        if (Objects.equals(idSnapshot.getKey(), user.getUid()))
                            for (DataSnapshot r : idSnapshot.getChildren()) {
                                Recipe recipe = new Recipe(r.child( path: "name").getValue(String.class), r.child( path: "description").getValue(String.class),
                                        (List<Pair<String, String>>) r.child( path: "list").getValue(), r.child( path: "type").getValue(String.class));
                                recipes.add(recipe);

                            }
                    }
                }
                mAdapter = new MenuAdapter(context, recipeHistory,  menu: false);
                recyclerViewHistory.setAdapter(mAdapter);
                recyclerViewHistory.setLayoutManager(new LinearLayoutManager(context));
                for (DataSnapshot name : _snapshot.getChildren()) {
                    for (Recipe recipe : recipes) {
                        if (recipe.getName().equals(name.getValue()))
                            recipeHistory.add(recipe);
                    }
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {

    }
});
```

Figura 6.69: Fragmento de código (historial)

7. Conclusiones y trabajo futuro

7.1. Conclusiones

Para este trabajo se ha desarrollado una aplicación enfocada en facilitar la gestión de comidas de las personas, con el objetivo de ofrecer una dieta variada en función de los alimentos disponibles. Para ello, se han implementado un conjunto de funcionalidades como es lista de la compra, despensa, recetas, historial de gasto y menú. Todas estas funcionalidades permiten al usuario un mejor manejo de sus comidas con previsión a futuro. Además, al mantener esta información digitalizada el usuario podrá acceder a un montón de contenido en línea que otros usuarios compartan, enfocado al ámbito de las recetas. También al llevar un registro de las comidas realizadas el usuario podrá repetir aquellas recetas que tanto le han gustado en otras ocasiones. Por otro lado, la gestión de un historial de precios sobre las compras realizadas anteriormente permite al usuario una mejor planificación futura de sus compras. El usuario podrá anticiparse a sus necesidades y observar sus principales fuentes de gastos.

Además, se ha implementado la funcionalidad de gestionar la información del propio usuario, registrando diversa información personal del usuario que de cara a futuro sería muy interesante para el desarrollo de nuevas funcionalidades. Esta información permitiría a la aplicación ofrecer unos servicios más especializados en base a las necesidades de los usuarios.

Todos los ficheros fuente del proyecto se encuentran alojados en el siguiente repositorio de GitHub: https://github.com/Ivanlcrv/TFG.git

7.2. Trabajo futuro

Todos los objetivos planteados durante la especificación de requisitos han sido alcanzados, sin embargo, aún quedan multitud de líneas de mejora y ampliación de la aplicación para ofrecer una experiencia más diversa y enriquecedora al usuario final. Entre ellas podemos destacar las siguientes:

- Medición nutricional: con el fin de ofrecer mejores recomendaciones a los usuarios, en base a los datos recogidos en el registro, se ofrecerán recomendaciones de menús basados no solo en los alimentos disponibles en la despensa sino también en los requerimientos nutricionales de cada usuario. Se manejarían variables como: aporte calórico, necesidades de proteínas, hidratos de carbono y grasas. En base a los objetivos que pudiese tener cada usuario se distinguirían tres casos: pérdida de peso, peso estable o aumento de peso.

- Historial de gastos más detallado: para poder ofrecer unos gastos más detallados, ofreciendo una mejor planificación financiera, el usuario podría especificar el coste de cada artículo. Además, el histórico de datos mostraría datos más extensos donde poder ver de manera más detallada cada componente del gasto y su aportación al total.

- Mayor variedad de registros en la base de datos: facilitar el registro al usuario resulta una opción interesante para fomentar el uso entre más usuarios de la aplicación. Permitir el registro a un usuario a través de cuentas como Google o Facebook que agilicen el proceso de rellenar el formulario indicando los datos necesarios para generar una cuenta en la aplicación.

- Mejora en la facilidad de crear recetas para los creadores de contenido: ofrecer una interfaz más intuitiva y cómoda para los creadores de contenido fomentaría la creación de nuevas recetas. Para ello los formularios de creación de recetas deberían permitir una mejor visualización de los datos a introducir especialmente focalizados en apartados como la descripción de una receta. Además, facilitar una sección de pasos a seguir durante la receta resultaría más cómodo para los consumidores de las recetas y a los creadores de contenido les permitiría ilustrar de mejor manera la información.

- Optimización gestión de las unidades: enfocado en la mejora de la gestión de la despensa virtual y los alimentos de la lista de la compra, sería interesante añadir un manejo de unidades establecido y diverso para las distintas opciones de alimentos.

Evitando así dualidades a la hora de representar la cantidad de un alimento. Esto influiría de manera directa en la gestión de recetas que permitiría a un usuario alterar las combinaciones de alimentos en su despensa de manera más rápida y eficaz.

Conclusions and future work

Conclusions

For this work, an application has been developed focused on facilitating the management of people's meals, with the aim of offering a varied diet based on the available foods. For this, a set of functionalities have been implemented, such as the shopping list, pantry, recipes, spending history and menu. All these functionalities allow the user to better manage their meals with a forecast for the future. In addition, by keeping this information digitized, the user will be able to access a lot of online content that other users share, focused on the field of recipes. Also, by keeping a record of the meals made, the user will be able to repeat those recipes that they have liked so much on other occasions. On the other hand, the management of a price history on the purchases made previously allows the user to better plan future purchases. The user will be able to anticipate their needs and observe their main sources of expenses.

In addition, the functionality of managing the user's own information has been implemented, registering various personal information of the user that in the future would be very interesting for the development of new functionalities. This information would allow the application to offer more specialized services based on the needs of the users.

All the project source files are hosted in the following GitHub repository: https://github.com/Ivanlcrv/TFG.git

Future work

All the objectives set during the requirements specification have been achieved, however, there are still many lines of improvement and expansion of the application to offer a more diverse and enriching experience to the end user. Among them we can highlight the following:

- Nutritional measurement: in order to offer better recommendations to users, based on the data collected in the registry, menu recommendations will be offered based not only on the foods available in the pantry but also on the nutritional

requirements of each user. Variables such as: caloric intake, protein needs, carbohydrates and fats would be handled. Based on the objectives that each user could have, three cases would be distinguished: weight loss, stable weight or weight gain.

- More detailed expense history: In order to offer more detailed expenses, offering better financial planning, the user could specify the cost of each item. In addition, the historical data would show more extensive data where you can see in more detail each component of the expense and its contribution to the total.
- Greater variety of records in the database: facilitating user registration is an interesting option to encourage use among more users of the application. Allow a user to register through accounts such as Google or Facebook that speed up the process of filling out the form, indicating the necessary data to generate an account in the application.
- Improved ease of creating recipes for content creators: Providing a more intuitive and comfortable interface for content creators would encourage the creation of new recipes. For this, the recipe creation forms should allow a better visualization of the data to be entered, especially focused on sections such as the description of a recipe. In addition, providing a section of steps to follow during the recipe would be more comfortable for consumers of the recipes and would allow content creators to better illustrate the information.
- Unit management optimization: focused on improving the management of the virtual pantry and the foods on the shopping list, it would be interesting to add an established and diverse unit management for the different food options. Thus avoiding dualities when representing the amount of a food. This would directly influence recipe management allowing a user to alter food combinations in their pantry more quickly and efficiently.

8. Aportes individuales

Durante este capítulo se expondrán las diferentes tareas realizadas por cada desarrollador durante la realización del proyecto.

8.1. Iván Ledesma Casado

Toma de requisitos

En el inicio del desarrollo resultó fundamental la búsqueda de requisitos en los que se iba a envolver la aplicación. Para ello, se llevó a cabo un análisis de otras aplicaciones semejantes con el fin de obtener un resultado final útil y de interés para el usuario. Por otro lado, también se exploraron aplicaciones multimedia que ofrecían una estética atractiva e interesante que lograse captar la atención del público. Para determinar los límites y áreas de interés del trabajo fue necesario mantener reuniones periódicas entre los miembros del grupo para plasmar luego las ideas en reuniones finales con el tutor.

Selección de tecnologías

Una vez establecidos los requisitos de la aplicación ambos miembros del grupo debieron formarse sobre las tecnologías que iban a permitir el desarrollo de la aplicación móvil. Previó al estudio de las tecnologías se llevó a cabo una reunión entre los miembros para decidir el material mínimo y necesario para el desarrollo de la aplicación en el entorno de desarrollo acordado. Cada miembro realizó por separado la investigación necesaria para adquirir los nuevos conocimientos en la materia que se les había presentado y así lograr abarcar un mayor punto de vista en un trabajo que posteriormente se realizaría de manera conjunta.

Creación y gestión de la base de datos

Para la realización de la base de datos se han hecho reuniones periódicas para estudiar de forma conjunta la estructura e información a almacenar. Buscando una eficiencia en la gestión de acceso a los datos y una implementación homogénea y coherente de las estructuras de datos que se

presentaban en el proyecto. Estas reuniones se realizaban una vez comentado el contenido de la funcionalidad a implementar.

Diseño e implementación de la aplicación móvil

La implementación se ha realizado de manera conjunta, mediante reuniones presenciales y en formato online. Tanto la implementación como el diseño de la aplicación ha sido desarrollada por ambos miembros del equipo, consiguiendo crear funcionalidades de la mejor manera posible. La primera iteración se caracterizó por una reunión previa donde se comentaron posibles diseños aplicables de carácter global. Individualmente, se aportaron ideas de diseño, basadas en otras aplicaciones o imágenes obtenidas de internet, además de gamas de colores. Ambos miembros hemos estado presentes en cada sesión de programación, de tal manera que se producía alternancia del que programaba. El miembro que no programaba se dedicaba a la resolución de dudas mediante búsquedas en internet y a la aportación de nuevas ideas, tanto para optimizar código como diseños. Esta colaboración ha permitido crear una aplicación con estilo homogéneo y mantener informados a ambos miembros sobre cada etapa del proceso.

Pruebas y corrección de errores

Durante el desarrollo del trabajo se han desarrollado pruebas y correcciones de errores para lograr una aplicación más depurada y optimizada. Este proceso se hizo de forma conjunta por ambos integrantes del grupo. Estas pruebas se han producido a medida que se desarrollaban las diversas etapas del trabajo evitando una congestión de la carga en las fases finales del proyecto. Al final de este, se han desarrollado pruebas finales con el objetivo de corregir cualquier posible fallo que no se hubiese detectado con anterioridad.

Redacción de memoria

La redacción de la memoria ha sido realizada entre ambos miembros del equipo bajo la supervisión del tutor del Trabajo de Fin de Grado. En busca de un enfoque integrador para mantener la coherencia durante todo el desarrollo de la memoria ambos integrantes del grupo realizaron el desarrollo de la memoria.

8.2. Sandra Ramos Ramos

Toma de requisitos

En el inicio del desarrollo resultó fundamental la búsqueda de requisitos en los que se iba a envolver la aplicación. Para ello, se llevó a cabo un análisis de otras aplicaciones semejantes con el fin de obtener un resultado final útil y de interés para el usuario. Por otro lado, también se exploraron aplicaciones multimedia que ofrecían una estética atractiva e interesante que lograse captar la atención del público. Para determinar los límites y áreas de interés del trabajo fue necesario mantener reuniones periódicas entre los miembros del grupo para plasmar luego las ideas en reuniones finales con el tutor.

Selección de tecnologías

Una vez establecidos los requisitos de la aplicación ambos miembros del grupo debieron formarse sobre las tecnologías que iban a permitir el desarrollo de la aplicación móvil. Previo al estudio de las tecnologías, se llevó a cabo una reunión entre los miembros para decidir el material mínimo y necesario. Cada miembro realizó por separado la investigación necesaria para adquirir los nuevos conocimientos en la materia que se les había presentado.

Creación y gestión de la base de datos

Se ha realizado un trabajo continuo y conjunto de la estructura e información a almacenar en la base de datos. Se han ido realizando reuniones a medida que se implementaron las funcionalidades, con el objetivo de crear estructuras óptimas al tener alto conocimiento de las necesidades de la funcionalidad desarrollada en cada etapa.

Diseño e implementación de la aplicación móvil

La implementación se ha realizado en mayor parte de forma conjunta, mediante reuniones presenciales y en formato online. Ambos miembros del equipo hemos participado en el diseño e implementación de cada funcionalidad ofrecida en la aplicación, con el objetivo de crear lluvias de ideas y crear así la mejor funcionalidad posible. Durante el desarrollo de la primera iteración se llevó a cabo una primera reunión para comentar posibles diseños generales de la aplicación. Ambos

miembros aportamos opiniones individuales, obtenidas de fuentes como internet o pinterest y diversas gamas de colores. Durante las sesiones dedicadas a la parte correspondiente con programación, se ha producido alternancia de la persona dedicada a programar, de tal forma que el otro miembro del grupo se encargaba de búsqueda de dudas y aportación de ideas, tanto de código como de diseños de vistas. Esta colaboración ha permitido crear un estilo homogéneo en la aplicación, optimizar la realización de código al poder comparar ideas y mantener, a ambos miembros, informados sobre todas las funcionalidades.

Pruebas y corrección de errores

Durante el desarrollo del trabajo se han desarrollado pruebas y correcciones de errores. Este proceso se hizo de forma conjunta, entre ambos miembros del equipo. Estas pruebas se han producido a medida que se desarrollaban las diversas etapas del trabajo, con el objetivo de no acumular fallos. Al final de este, se han desarrollado pruebas finales para asegurar el correcto funcionamiento de la aplicación.

Redacción de memoria

La redacción de la memoria ha sido realizada entre ambos miembros del equipo bajo la supervisión del tutor del Trabajo de Fin de Grado. Buscando un enfoque unificado el trabajo realizado por los dos miembros del equipo ha sido conjunto durante este apartado del proyecto.

Bibliografía

[1] Github: https://github.com/

[2] Discord: https://discord.com/

[3] Yuka: https://yuka.io/es/

[4] Cookpad: https://cookpad.com/es/home

[5] TikTok: https://www.tiktok.com/

[6] Spendee: https://www.spendee.com/

[7] Android studio: https://developer.android.com/studio

[8] Java: https://www.java.com/es/

[9] XML: https://developer.mozilla.org/es/docs/Web/XML/XML_introduction

[10] Firebase: https://firebase.google.com/?hl=es-419

[11] Charting: https://github.com/PhilJay/MPAndroidChart/tree/master/MPChartLib/src/main/java/com/github/mikephil/charting

Anexos A

Guía de uso

En este anexo se procede a explicar cómo hacer uso de la aplicación de gestión de comidas para administrador y, en segundo lugar, se explica cómo usar la aplicación móvil para el resto de los usuarios.

A.1. Vista principal administrador

En la figura A.1 se ilustra la vista principal del administrador una vez haya iniciado sesión en la aplicación. Existen dos opciones de navegación, situadas en la parte inferior de la vista. Puede utilizar la opción mostrada por defecto, es decir, la correspondiente a la gestión de usuarios, o en caso contrario, la referente a la información de cuenta del propio administrador.

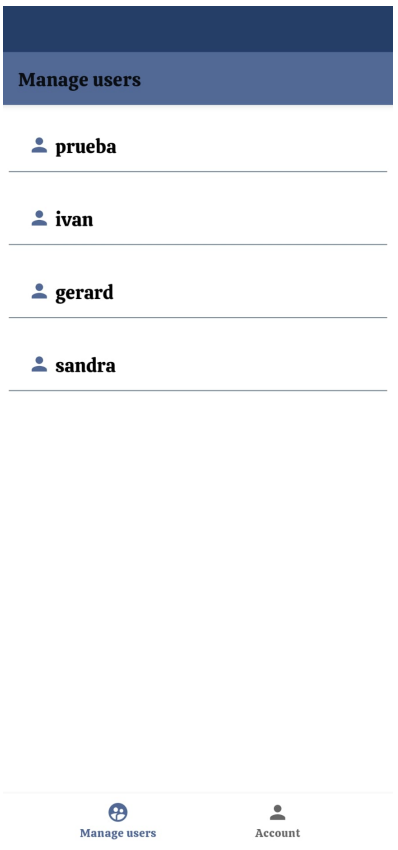


Figura A.1: Vista administrador

A.2. Gestión de usuarios

Esta gestión de usuarios cuenta con dos vistas para realizar una función como es eliminar usuario registrado. La primera de ellas es la previamente ilustrada en la figura A.1. Una vez situada en la vista, al seleccionar un usuario se redirige al administrador a una nueva vista, tal y como se refleja en A.2. La nueva vista refleja la información del usuario seleccionado, y mediante un botón situado en la parte inferior “DELETE ACCOUNT” se puede realizar la eliminación del usuario seleccionado.

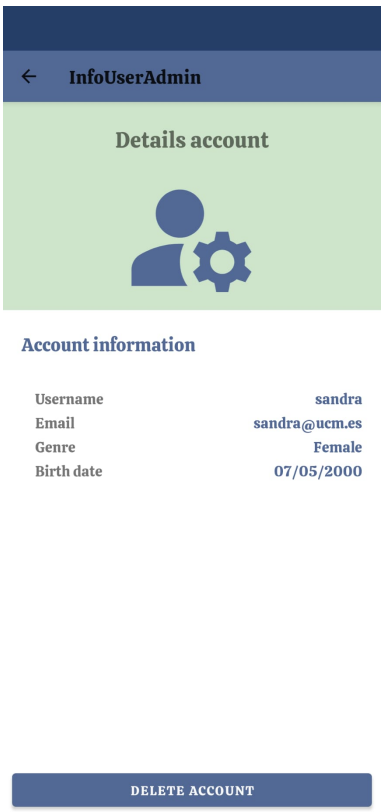


Figura A.2: Gestión usuarios

A.3. Gestión de cuenta

Esta gestión de cuenta refleja la información relativa al usuario o administrador. En esta vista se pueden realizar múltiples acciones como es modificar información, eliminar la cuenta y realizar un logout. En primer lugar, si se desea modificar la información, el usuario debe situarse sobre el campo a modificar, cumplimentando el mismo con la nueva información. Una vez haya cumplimentado los datos deseados, debe salvar los cambios mediante el botón "SAVE CHANGES". En segundo lugar, para eliminar la cuenta de usuario, basta con hacer uso del botón situado en la parte inferior de la vista, "DELETE YOUR ACCOUNT". En último lugar, para realizar un logout, debe pulsar sobre el icono de salida situado en la parte superior izquierda. Gestión de cuentas se ilustra en la figura A.3.

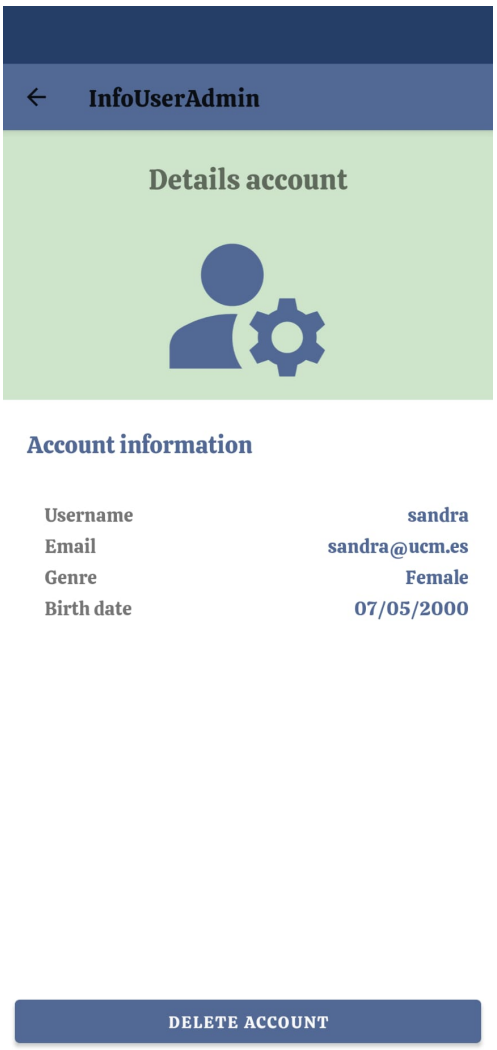


Figura A.3: Gestión de cuenta

A.4. Vista principal usuario

La vista principal del usuario recae con la funcionalidad de recetas. Esta vista aparece tras el inicio de sesión. Podrá encontrar un menú de navegación en la parte inferior de la vista, tal y como se refleja en A.4. Esta vista refleja un listado de recetas con sus correspondientes fotografías. Las posibles acciones a realizar son las relacionadas con recetas, lista de la compra, menú, despensa e información de cuenta.

Figura A.4: Vista usuario

A.5. Gestión de recetas

La gestión de recetas coincide con la vista principal, mostrada por defecto, tras el inicio de sesión de un usuario, ilustrada en la figura A.4. En ella se pueden realizar varias acciones, se puede ver, modificar, añadir, buscar y eliminar recetas, y para ello, se hace uso de diversas vistas. En primer lugar, para visualizar una receta, basta con seleccionar la receta deseada, de tal forma que se redirige al usuario a una nueva vista. En esta vista ilustrada en la A.5, se puede visualizar la receta y modificar la información relativa a la misma. En el caso de modificar la receta, basta con situarse en el campo a modificar e introducir la nueva información, a continuación, se debe pulsar sobre el botón "SAVE RECIPE INFO" para actualizar la información en el sistema. Partiendo de nuevo de la figura A.4,

se puede añadir una receta haciendo uso de un icono situado en la parte inferior derecha, caracterizado por el signo "+". Una vez pulsado, se redirige a una nueva vista ilustrada en A.6, el usuario debe cumplimentar el formulario, y una vez relleno se activa el botón "ADD". Para añadir la receta debe hacerse uso de dicho botón. La eliminación de una receta parte de la vista reflejada en A.4, se debe mantener seleccionado durante unos segundos la receta a eliminar, apareciendo un icono en el lateral derecho de la receta, tal y como se ilustra en A.7. Se debe pulsar dicho icono para procesar la eliminación. La acción de buscar se realiza gracias al buscador situado en la parte superior de la figura A.4. El usuario debe escribir los caracteres a buscar, mientras el sistema le muestra resultados coincidentes.

Figura A.5: Ver/modificar receta

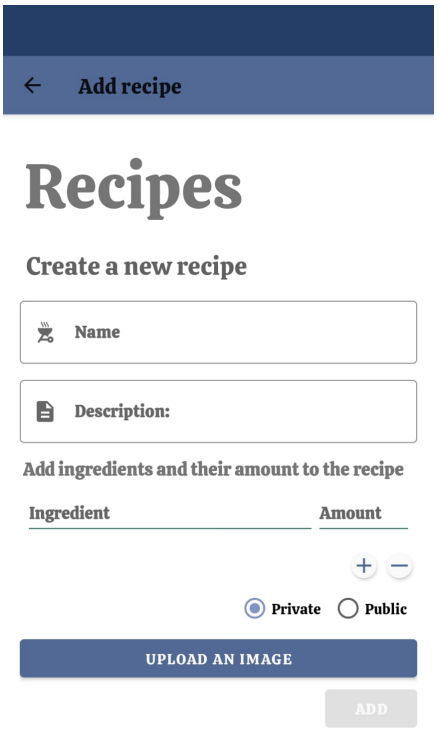


Figura A.6: Añadir receta

Figura A.7: Eliminar receta

A.6. Gestión de lista de la compra

Para llegar a esta funcionalidad debe seleccionarse la segunda opción del navegador inferior de la figura A.4. El usuario será redirigido a una nueva vista, tal y como se muestra en la figura A.8. Esta vista se caracteriza fundamentalmente por contener un listado con nombres de ingredientes. Se pueden realizar varias acciones como son las relativas a añadir alimento a la lista, eliminar alimento, ver historial de gato, marcar como comprada la lista añadiendo el gasto relativo a la compra, buscar alimento y modificar alimento. En primer lugar, para añadir un alimento a la lista de la compra, el usuario debe pulsar sobre el icono "+", situado en la parte inferior derecha. Una vez pulsado, se redirige al usuario a un formulario, tal y como se muestra en A.9. El usuario debe

cumplimentar el formulario, y una vez rellenado, se activa la acción de añadir el alimento. Para realizar la acción debe hacer uso del botón “ADD ITEM”. En segundo lugar, para eliminar un alimento se parte de la vista principal de lista de la compra, A.8. Debe mantenerse pulsado durante unos segundos el ingrediente a eliminar, esto produce que aparezca un icono de basura en la parte lateral derecha del ingrediente, ilustrado en A.10. El usuario debe pulsar sobre dicho icono para proceder con la eliminación. En tercer lugar, si el usuario desea visualizar el histórico de gastos, debe partir de la página principal de gestión de lista de la compra, reflejada en la figura A.8. Debe pulsar “View spending history” situado en la parte superior. El usuario será redirigido a una nueva vista, A.11. En ella, existen tres opciones temporales y un gráfico de gastos en función de la opción temporal seleccionada. En cuarto lugar, para marcar como comprado un ingrediente y su correspondiente gasto de la compra, se hace uso de la vista principal de lista de la compra, A.8. Tal y como refleja la figura, los ingredientes pueden ser seleccionados en la parte izquierda del nombre de este. El usuario debe seleccionar aquellos ingredientes que ha comprado. Una vez se haya realizado la selección, debe introducir el importe de la compra en el formulario situado en la parte inferior izquierda de la vista. Para finalizar esta acción, debe hacer uso del botón “BUY”. También, se puede realizar la acción de buscar gracias al buscador situado en la parte superior de la vista principal de la lista de la compra, A.8. El usuario debe rellenar caracteres a buscar y el sistema muestra coincidencias. Por último, se puede modificar el alimento. Esta acción permite al usuario modificar los datos, sobre todo, tiene el objetivo de ajustar la cantidad comprada del ingrediente. A partir de la vista principal de la lista de la compra, A.8, se debe pulsar sobre el ingrediente a modificar. El usuario es redirigido a una nueva vista, donde se muestra la información del ingrediente, tal y como refleja la figura A.12. Para modificar la información, basta con situarse en el campo a modificar e introducir la nueva información. Para finalizar la acción debe hacer uso del botón “SAVE ITEM INFO”.

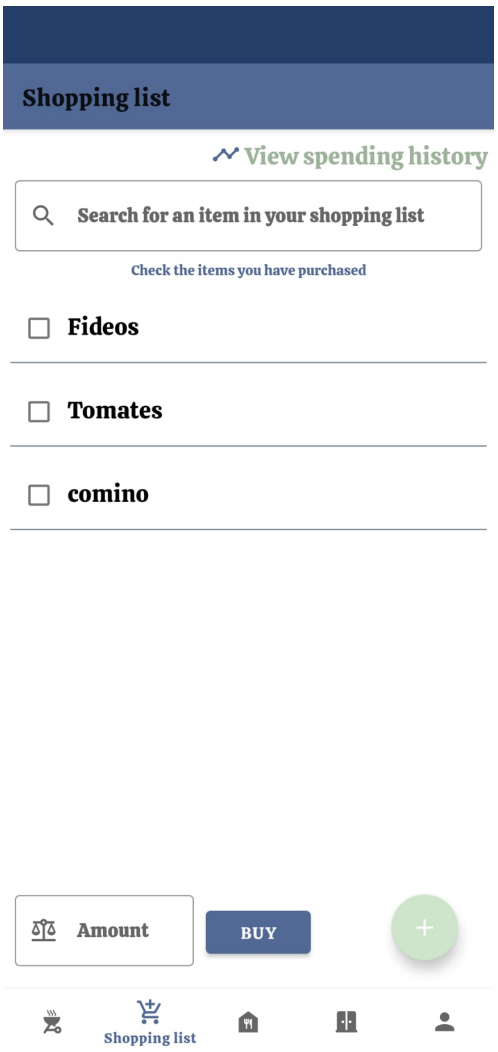


Figura A.8: Lista de la compra

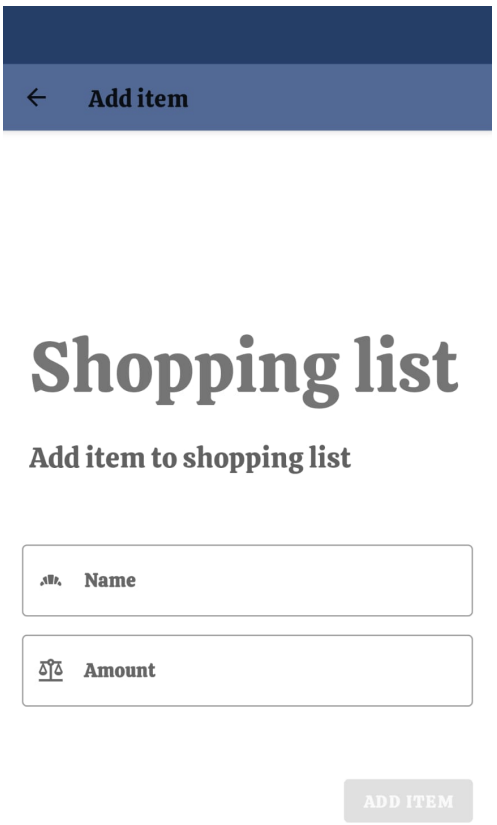


Figura A.9: Añadir elemento a la lista de la compra

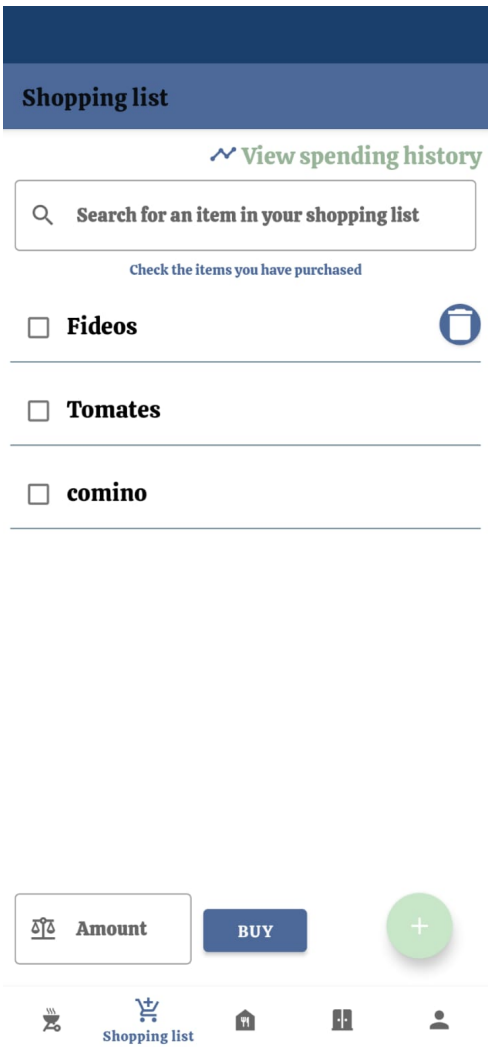


Figura A.10: Eliminar elemento de la lista de la compra

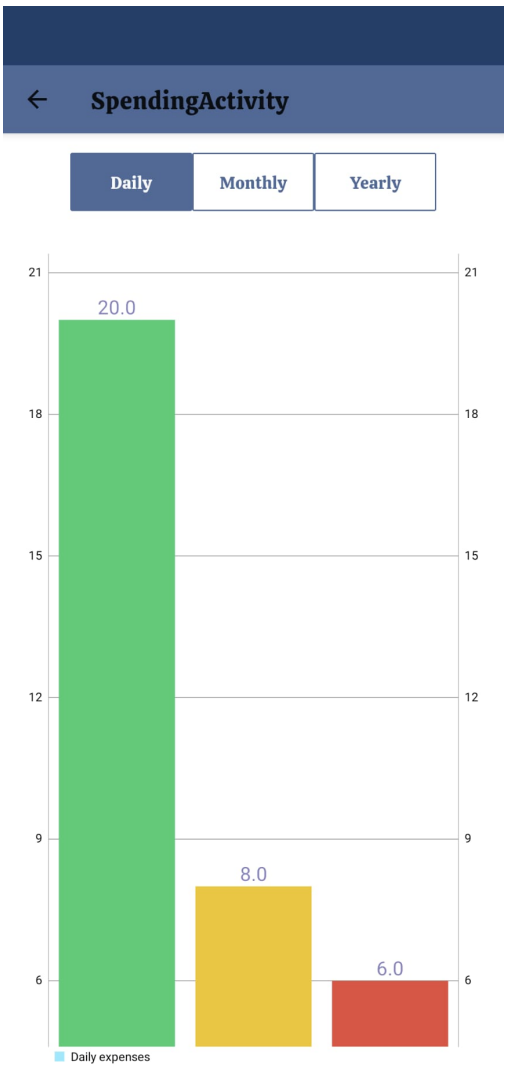


Figura A.11: Gráfico de gastos

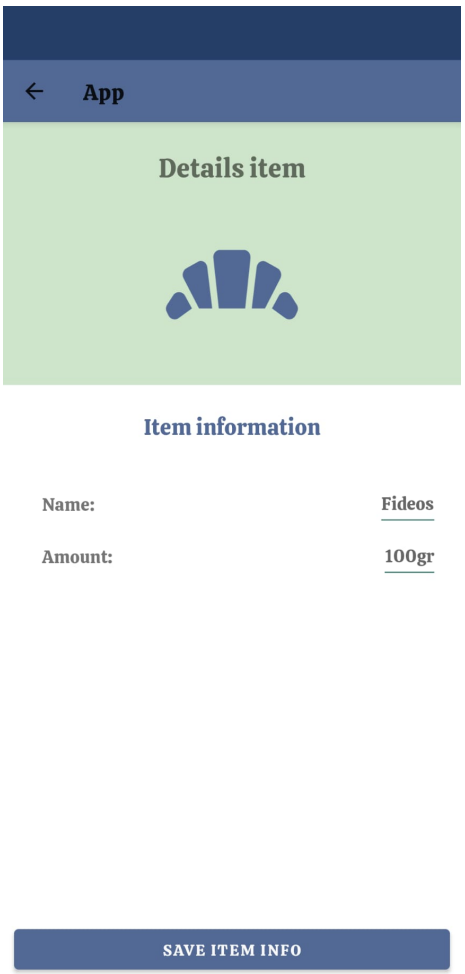


Figura A.12: Información elemento lista de la compra

A.7. Gestión de menú

Para llegar a esta funcionalidad debe seleccionarse la tercera opción del navegador inferior de la figura A.4. Fundamentalmente, esta vista se caracteriza por contar con un listado de recetas, tal y como se refleja en A.13. Se pueden realizar dos acciones, ver el conjunto de recetas que puede realizarse según la disponibilidad de ingredientes, coincidiendo con la figura A.13, visualizar una receta, marcar como realizada una receta y ver el historial de recetas. Para visualizar una receta, el usuario debe pulsar la receta deseada, siendo redirigido a una nueva vista, A.14, donde se ilustra la información relativa a la misma. Además, en dicha vista, puede marcar como realizada la receta. Para ello debe hacer uso del botón situado en la parte inferior "CHECK AS DONE!". Por otra parte, el historial de recetas se puede visualizar seleccionando la opción "HISTORY" situado en la

parte superior de la vista principal de menú, A.13. En ella, el usuario puede visualizar el listado de recetas previamente elaboradas, tal y como se refleja en A.15.

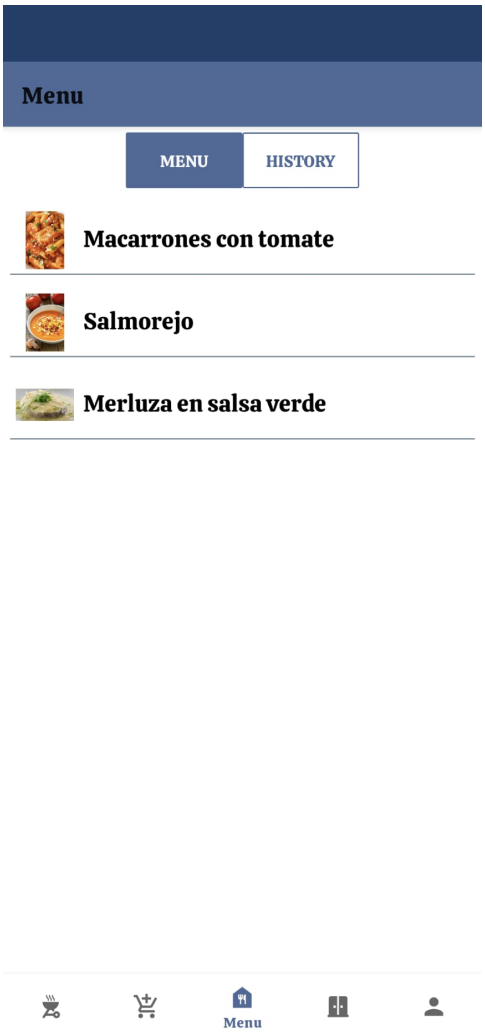


Figura A.13: Menú

Figura A.14: Marcar comida

Figura A.15: Historial comidas

A.8. Gestión de la despensa

Para llegar a esta funcionalidad debe seleccionarse la cuarta opción del navegador inferior de la figura A.4. El usuario será redirigido a una nueva vista, A.16. Se caracteriza por un listado de ingredientes. Se pueden hacer múltiples acciones relativas a añadir, eliminar, buscar, visualizar y modificar alimentos. En primer lugar, se puede añadir recetas haciendo uso del icono situado en la parte inferior derecha, caracterizado por un signo "+". Una vez pulsado, se redirige al usuario a una nueva vista, A.17, donde se ilustra un formulario a cumplimentar. Una vez cumplimentado, se activa la opción de añadir. Para finalizar la acción el usuario debe hacer uso del botón "ADD". La eliminación de ingredientes es una acción que parte de la vista principal de la despensa, A.16. El usuario debe mantener pulsado el ingrediente a eliminar, produciendo la aparición de un icono en el lateral derecho del ingrediente, reflejado en A.18. El usuario debe hacer uso de dicho icono para

proceder con la eliminación. Buscar un alimento se realiza gracias al buscador situado en la parte superior de la vista principal de la despensa, tal y como se muestra en A.16. El usuario debe introducir la serie de caracteres a buscar y el sistema muestra las coincidencias. Por último, para visualizar y modificar un alimento, se debe haber pulsado sobre el ingrediente a visualizar y/o modificar en la vista principal de la despensa. El usuario es redirigido a un formulario, tal y como ilustra la figura A.19. El formulario está cumplimentado con la información relativa del alimento. Para realizar modificaciones, basta con situarse en el campo a modificar e introducir la nueva información. Para finalizar la acción, el usuario debe hacer uso del botón inferior "SAVE FOOD INFO".

Figura A.16: Despensa virtual

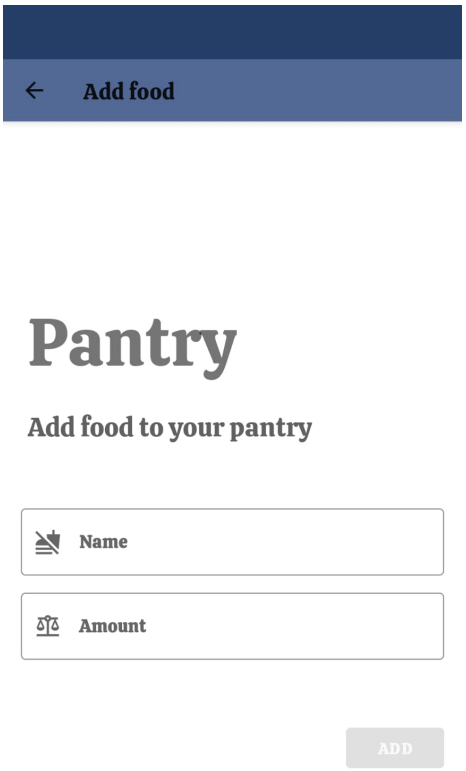


Figura A.17: Añadir alimento a la despensa

Figura A.18: Eliminar alimento de la despensa

Figura A.19: Editar alimento de la despensa