UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

AppRecetas: Aplicación Android para compartir y conservar recetas de cocina canaria

TITULACIÓN: Graduado en Ingeniería en Tecnologías de la Telecomunicación
AUTOR: Oscar Medina Morillo
TUTORA: Dra. Carmen Nieves Ojeda Guerra

FECHA: Diciembre 2017

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

AppRecetas: Aplicación Android para compartir y conservar recetas de cocina canaria

HOJA DE FIRMAS

Autor:

Fdo: Oscar Medina Morillo

Tutora:

Fdo: Dra. Carmen Nieves Ojeda Guerra

Fecha: Diciembre 2017

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

AppRecetas: Aplicación Android para compartir y conservar recetas de cocina canaria

HOJA DE EVALUACIÓN

Calificación: _____

Presidente/a:

Fdo:

Vocal:

Fdo:

Secretario/a:

Fdo:

Fecha: Diciembre 2017

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Parte I

Memoria

Capítulo **1**

INTRODUCCIÓN

1.1 Introducción

El Trabajo Fin de Grado que se plantea en el presente documento consiste en el desarrollo de una aplicación móvil para terminales Android que permita a la población joven (y no tan joven) acceder a un almacenamiento de datos, en la que éstos, a través de sus dispositivos móviles, puedan consultar recetas de la cocina canaria que anteriormente han realizado nuestros mayores.

1.1.1 Antecedentes

A lo largo de los tiempos, las recetas para realizar cualquier tipo de comida, se transmitían de generación en generación, de padres a hijos. Con el paso del tiempo y con los cambios en nuestra sociedad, cada vez es más frecuente que los niños pasen la mayor parte del tiempo con los abuelos, con tiempo y capacidad para facilitarles un menú diario elaborado y equilibrado.

Hoy en día, los jóvenes emprenden su aventura estudiantil o laboral fuera de nuestra comunidad, con mayor frecuencia. Según los resultados de la Encuesta de Jóvenes de Canarias 2012 [1],en torno a un 30 % de jóvenes canarios se han trasladado fuera de su isla en algún momento de su vida por motivos de trabajo, estudios, voluntariado o por intercambios juveniles, tanto a otra isla distinta de la de residencia, bien a otra CCAA o al extranjero. Esta movilidad potencia la pérdida de nuestras costumbres, entre las que están las recetas de nuestra cocina y la añoranza de las comidas caseras. Por ello, surge el presente Trabajo Fin de Grado, con el objetivo de facilitar el acceso de nuestras recetas tal y como las prepararían nuestros mayores, en un entorno común.

Con la aparición de las tecnologías móviles en los últimos años, la sociedad ha visto su forma de interactuar con el mundo totalmente cambiada. Estos dispositivos, que antaño se usaban únicamente para llamar o mandar mensajes, han ampliado su funcionalidad sobremanera, permitiendo al usuario realizar cualquier tarea cotidiana. Esto se debe, fundamentalmente, a la existencia de múltiples y variadas aplicaciones que, ejecutadas sobre estos dispositivos, permiten a los usuarios mejorar su calidad de vida.

1.1.2 La tecnología móvil

La comunicación es un proceso vital para el desarrollo individual y de la sociedad. Actualmente se han multiplicado en gran número las posibilidades de comunicación a distancia, y uno de los medios más populares y con más difusión en el mundo es el teléfono móvil. Tanto es así, que desde finales de 2014 existen más terminales móviles en el mundo que personas hay en él.

Los dispositivos móviles han crecido exponencialmente en popularidad con el paso de los años, gracias a la amplia oferta de terminales, con un gran rango de precios. Su fácil e intuitivo uso favorece a su crecimiento. Así, y según *statista* (web de estadísticas que usa más de 18.000 fuentes diferentes), para 2017 se prevé que más de 4.770 millones de personas en el mundo tengan un teléfono móvil [2].

En línea con la aparición de los dispositivos móviles, diferentes compañías desarrollaron sistemas operativos que, implantados sobre estos terminales, gestionan de forma eficiente los recursos disponibles en ellos. A lo largo del tiempo se han diseñado numerosos sistemas operativos para terminales móviles, sin embargo, en la actualidad, los sistemas operativos dominantes en el mercado según Gartner [3], son: Google Android [4] y Apple iOS [5] con el 99.1 % del mercado entre ambos, siendo Android el sistema operativo que tiene mayor cuota de mercado con el 86.2 % (figura 1.1).

Operating System	2Q16 Units	2Q16 Market Share (%)	2Q15 Units	2Q15 Market Share (%)
Android	296,912.8	86.2	271,647.0	82.2
iOS	44,395.0	12.9	48,085.5	14.6
Windows	1,971.0	0.6	8,198.2	2.5
Blackberry	400.4	0.1	1,153.2	0.3
Others	680.6	0.2	1,229.0	0.4
Total	**344,359.7**	**100.0**	**330,312.9**	**100.0**

Figura 1.1: Ventas mundiales de smartphones por Sistemas Operativos en el 2º trimestre de 2016

A la hora de implementar una aplicación para terminal móvil, lo primero que hay que hacer es elegir el lenguaje que se va a usar. Al igual que en otros ámbitos de programación, en la programación de aplicaciones móviles hay distintas opciones

que elegir, siendo los más usados los lenguajes nativos de los sistemas operativos indicados anteriormente. Así, en el caso de Android se usa el lenguaje de programación Java [6], mientras que para el sistema operativo iOS se utilizan los lenguajes Objective-C [7] y Swift [8]. Estos lenguajes de programación presentan características y funcionalidades que permiten explotar al máximo las características de los terminales móviles.

Por otro lado, y una vez desarrollada la aplicación, el problema que se presenta es la distribución de la misma. Una distribución que tiene que ser tal, que pueda llegar al máximo número de personas en el mundo. Para ello, estas grandes compañías que han desarrollado los sistemas operativos móviles actuales, también disponen de plataformas de distribución, como son: Google Play [9] en el caso de Google y App Store [10] en el caso de Apple. Evidentemente, el subir una aplicación final para distribuirla, implica un costo que depende de la plataforma en sí (la plataforma de Google es mucho más barata que la de Apple). Asimismo, por cada venta de una copia de la aplicación, la compañía correspondiente se queda con un porcentaje del precio de la misma.

Finalmente, indicar que en la actualidad se pueden encontrar en ambas plataformas (Google y Apple) aplicaciones relacionadas con el mundo de la gastronomía, como, por ejemplo: "Recetas de cocina gratis" [11] o "Recetas de todo gratis" [12]. Pero, en el ámbito de compartir recetas como si de un *blog* se tratara, similar a lo que se plantea en este Trabajo Fin de Grado, el número de aplicaciones se reduce, destacando "Cookpad Recetas" [13].

Basándonos en los datos expuestos anteriormente, la aplicación a desarrollar en este trabajo se implementará en Java para Android, ya que, además de que este sistema operativo tiene el 86,2 % de cuota de mercado (y, por tanto, se puede llegar a un mayor público), el alta como desarrollador en su plataforma es más barato.

Por último, hay que tener en cuenta que los datos a utilizar en la aplicación se almacenarán en una base de datos, por lo que se utilizará un servicio de almacenamiento en la nube. Con el fin de almacenar datos sensibles de forma segura, se utilizará un servicio con mecanismos de protección de datos para cumplir con la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal [14].

1.2 **Objetivos**

El objetivo a conseguir con este Trabajo Fin de Grado será desarrollar una aplicación sobre dispositivos móviles Android que permita a usuarios compartir, de forma fácil y sencilla, recetas de la gastronomía canaria, posibilitando las siguientes acciones:

1. Registrarse/loguearse y desconectarse de la aplicación.
2. Añadir una nueva receta.
3. Administrar recetas añadidas por el usuario.
4. Consultar y editar tu perfil.
5. Visualizar todas las recetas compartidas.
6. Realizar un filtrado de recetas.
7. Añadir una receta a la lista de favoritos.

Inicialmente, la aplicación pedirá al usuario registrarse/loguearse. Dependiendo del tipo de cuenta tendrá acceso a diferentes características. Existen dos tipos de cuentas: "creador de contenido" y "consumidor del contenido". Ambas cuentas comparten estas funcionalidades: una vez logueado, se visualizará la pantalla principal de la aplicación como tal, pudiendo navegar por ella a través de un menú que cuenta con los apartados de la aplicación: "Recetas", "Favoritos" y "Perfil".

En el apartado "Recetas", el usuario visualizará todas las recetas compartidas ordenadas cronológicamente de más reciente a más antigua. En esta pantalla se podrá realizar una búsqueda/filtrado de las recetas deseadas, pudiendo añadirlas a la lista de recetas favoritas. Al acceder a "Favoritos" podrá visualizar todas las recetas que ha marcado previamente como tal. Y, por último, mediante el apartado "Perfil", el usuario podrá editar a la vez que observar sus últimas acciones en la aplicación.

Además de las funciones explicadas anteriormente, el tipo de cuenta "creador de contenidos" tendrá acceso al apartado exclusivo: "Crear nueva receta", accesible desde el menú principal, donde podrá añadir una nueva receta incluyendo el nombre de la misma, añadiendo una foto o haciendo referencia a un vídeo público de *YouTube*, explicando la realización de dicha receta.

En la figura 1.2 se representa, de manera gráfica, el objetivo propuesto en este Trabajo de Fin de Grado. Para alcanzar este objetivo se desarrollará una aplicación Android que sea fácil de usar y mantener, para lo cual se seguirán los siguientes objetivos parciales:

O1. Analizar los requisitos de usuario de la aplicación.

O2. Implementar una interfaz de usuario que siga los principios de usabilidad enunciados por Jakob Nielsen [15] y realizar un test de usabilidad que será probado por usuarios finales de la aplicación.

O3. Usar una metodología de desarrollo de software para implementar la aplicación en el lenguaje Java para Android, que utilice una arquitectura completamente desacoplada, de forma que cualquier cambio importante sólo afecte a la parte cambiada, no al resto de la aplicación. Este objetivo se subdivide en:

- Especificar el diseño del código.
- Especificar el modelo de datos.
- Implementar la aplicación.

O4. Implementar una batería de test del modelo de datos para comprobar el acceso correcto a los datos (que es la parte más crítica de la aplicación).

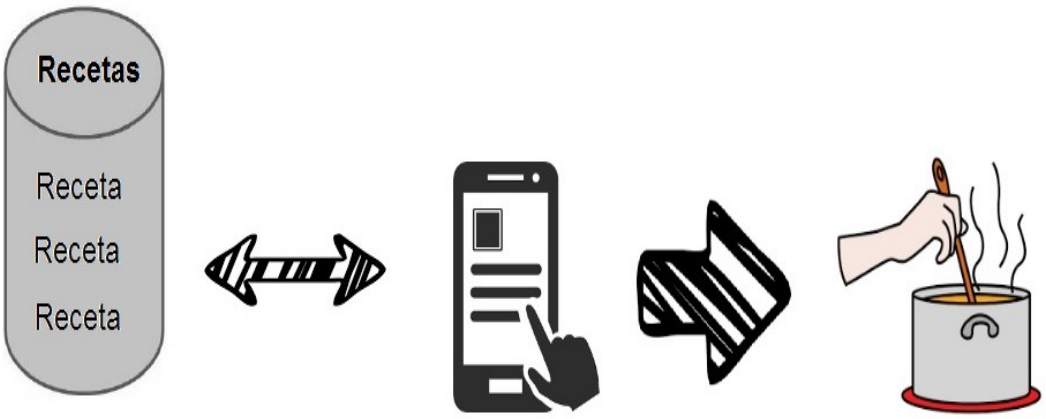


Figura 1.2: Gráfico del objetivo propuesto

1.3 Peticionario

El Trabajo de Fin de Grado presentado en esta memoria ha sido elaborado a petición de la Escuela de Ingeniería de Telecomunicación y Electrónica, para la obtención del título de Graduado en Ingeniería en Tecnologías de la Telecomunicación, mención Telemática, tal y como obliga el Real Decreto 1393/2007, de 29 de Octubre (BOE del 30) y su actualización en el Real Decreto 861/2010, de 2 de Julio en los que se indica que todas las enseñanzas oficiales concluirán con la elaboración y defensa de un Trabajo Fin de Título.

1.4 Organización de la memoria

La memoria que analiza la información y desarrollo de este trabajo se divide en seis capítulos, el pliego de condiciones y el presupuesto. A continuación, se comenta brevemente los capítulos incluidos:

- El capítulo 1 realiza una introducción a la tecnología propia de la programación de aplicaciones para móviles, los antecedentes del trabajo y especifica el objetivo principal del trabajo.
- El capítulo 2 amplía la información de la aplicación delimitando el sistema y buscando la funcionalidad que debe ofrecer desde la perspectiva del usuario final. Presenta los casos de uso y la interfaz de usuario de la aplicación.
- El capítulo 3 muestra las especificaciones detalladas de todos los objetos, incluyendo sus características y operaciones. Asimismo, en este capítulo se realiza el diseño del modelo de datos.
- El capítulo 4 utiliza el resultado del capítulo anterior para generar el código final en el lenguaje de programación elegido.
- El capítulo 5 verifica si el resultado es el que se quería, mediante una serie de pruebas sobre el modelo de la aplicación. Asimismo, en este capítulo se presentan los test de usabilidad del prototipo de la interfaz de usuario.
- El capítulo 6 incluye una serie de conclusiones del trabajo final y las posibles mejoras del mismo.
- Por último, se adjunta el pliego de condiciones que especifica qué condiciones y requisitos ha de satisfacer la aplicación; y el presupuesto, que detalla los costes totales del software generado.

Capítulo 2

MODELO DE REQUISITOS

2.1 Introducción

Un modelo, en el desarrollo de software, define cómo solucionar los problemas que aparecen en el desarrollo de una aplicación. Para desarrollar el software, existen diferentes metodologías, como la *Objectory* [16], que definen los distintos tipos de modelos que se pueden encontrar en este proceso. Así, los modelos básicos son: **de Requisitos, de Diseño, de Implementación y de Pruebas.** Asimismo, el desarrollo de software debe ser registrado a lo largo de todo el proceso, dando lugar a distintos documentos **(Documentación)**. De todos los modelos indicados, en este capítulo se tratará el de requisitos, como primer modelo a definir en el desarrollo de la aplicación objeto de este Trabajo de Fin de Grado.

El **modelo de requisitos** [17] tiene como objetivo delimitar el sistema y capturar la funcionalidad que debe ofrecer desde la perspectiva del usuario (es el contrato entre el desarrollador y el usuario final). El modelo de requisitos, que se va a presentar en este capítulo, está basado en *el modelo de casos de uso*. Este modelo es el primero en desarrollarse, ya que es la base para la formación del resto de los modelos.

El modelo de requisitos consiste, básicamente, de tres modelos principales:

- **Modelo de comportamiento:** se basa directamente en el modelo de casos de uso y especifica la funcionalidad, desde el punto de vista del usuario. Tiene dos conceptos claves:
  - *Actores*: representan los distintos papeles que los usuarios pueden jugar en el sistema.
  - *Casos de uso*: representa qué pueden hacer los actores con respecto al sistema.
- **Modelo de presentación o de interfaces o de borde:** especifica cómo interactúa el sistema con actores externos al ejecutar los casos de uso, es decir, especifica cómo se verán las interfaces gráficas y que función tiene cada una de ellas.
- **Modelo de información o del dominio del problema:** conceptualiza el sistema según los objetos que representan las entidades básicas de la aplicación.

Esta separación en tres modelos diferentes, dentro del modelo de requisitos, es importante para conseguir una mayor estabilidad en el desarrollo del sistema, permitiendo minimizar los efectos de cada uno sobre los otros dos. En este Trabajo Fin de Grado se presentan los dos primeros sub-modelos de requisitos: *Modelo de comportamiento* y *Modelo de interfaces*, para caracterizar la aplicación que se va a desarrollar. El tercer sub-modelo se sustituye por el estudio de los diagramas de secuencia que se hará en el capítulo siguiente (Modelo de diseño). Inicialmente, y antes de realizar estos sub-modelos, se realiza una descripción del problema, tal y como se le haría a un usuario de la aplicación.

2.2 Descripción del Problema

La descripción del problema es una definición muy básica de las necesidades de usuario, que sirve como punto de partida para comprender los requisitos del sistema, es decir, debe ser una descripción de lo que se necesita y no una propuesta de solución.

En este Trabajo Fin de Grado se pretende desarrollar una *Aplicación para realizar recetas de la gastronomía canaria*. Esta aplicación permite al usuario visualizar y/o añadir recetas de la amplia y variada cocina canaria.

La aplicación presentará en su página principal un fondo característico de la cultura canaria,(paisaje o comida), junto con los botones para poder *loguearse*, registrarse y acceder a la aplicación, pudiendo navegar en ella. Dependiendo del tipo de cuenta utilizada, (*creador de contenido* o *consumidor de contenido*), se tendrá acceso a distintas características.

Ambas cuentas comparten algunas funcionalidades a través de un menú con sus ítem: *Recetas*, *Visualizar Receta* e *Información Receta*. En el apartado *Recetas*, se podrá visualizar una lista de todas las recetas disponibles ordenadas alfabéticamente, de forma que el usuario pueda encontrar cualquiera de ellas de forma sencilla, separadas según el tipo de recetas (Carne, Pescado, Postres, Sopas y Potajes,...). En el apartado de *Información Receta* se mostrará en detalle la receta elegida mientras que en el apartado *Visualizar Receta* será posible ver, por medio de vídeo, la preparación de la receta.

Por otro lado, hay otras funcionalidades que sólo serán disponibles para los "Usuarios Registrados", como son: *Favoritos*, *Perfil* y *Crear nueva receta*. Así, en el apartado *Favoritos* se mostrará al usuario, por medio de una lista, todas las recetas que ha marcado como favoritas. Para poder editar la información referente a la cuenta del usuario se accederá al apartado *Perfil* y, por último, en la sección *Crear nueva receta*, el usuario registrado podrá añadir una nueva receta rellenando su título, una foto y el enlace a un video alojado en el servidor *Youtube*, explicando la realización de dicha receta.

2.3 Modelo de Comportamiento

El **modelo de comportamiento** [17], ilustra el modo en el que se comporta el software como consecuencia de eventos externos llamados *casos de uso*. Un caso de uso capta las interacciones que ocurren entre los productores o consumidores de la información y el sistema. Las distintas formas de usar un sistema dan lugar a los *actores*.Como se muestra en la figura 2.1 el actor y el caso de uso representan los dos elementos básicos de este modelo.

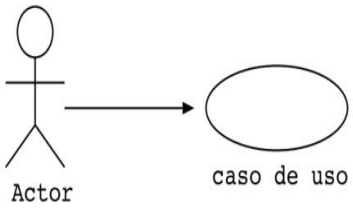


Figura 2.1: Representación de las entidades básicas para el modelo de comportamiento

2.3.1 Actores

Un **actor** [17], no es una persona específica sino el rol que desempeña ésta (o un dispositivo) en un contexto específico. Un actor "llama al sistema para que entregue uno de sus servicios". Al definir todos los actores y los casos de uso se define la funcionalidad completa del sistema.

Los actores se identifican antes que los casos de uso, para que estos sean la herramienta principal para encontrar los casos de uso. Al definir todos los actores y los casos de uso se define la funcionalidad completa del sistema.

A la hora de definir los actores, se identifican primero aquellos que son la razón principal del sistema, conocidos como *actores primarios*, que son los que rigen la secuencia lógica de ejecución del sistema. Además existen otros actores que supervisan y mantienen el sistema, conocidos como *actores secundarios*. Estos corresponden, por lo general, a máquinas o sistemas externos.

Los actores principales interactúan para lograr la función requerida del sistema y obtienen el beneficio previsto de éste. Trabajan con el software en forma directa y con frecuencia mientras que los actores secundarios dan apoyo al sistema, de modo que los primarios puedan hacer su trabajo.

Para especificar los actores de la aplicación, se pueden diferenciar los siguientes: los actores primarios a los que se les define como **Usuario registrado** y **Usuario no registrado**, que son los encargados de introducir los datos necesarios en la

aplicación para que ésta le puedan proporcionar el servicio que proporciona, y un actor secundarios, como es: **Base de datos**, que será el responsable del manejo de las recetas a través de un servidor en la nube. En la figura 2.2 se presenta un diagrama representando al sistema como una caja cerrada y los diferentes actores como entidades externas a él.

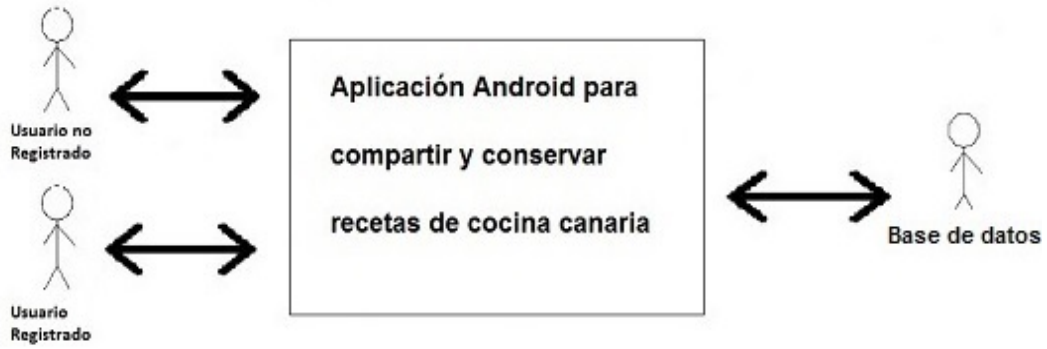


Figura 2.2: Delimitación del sistema para la Aplicación AppRecetas

2.3.2 Casos de uso

Después de haber definido los actores se define la funcionalidad del sistema a través de los casos de uso [17]. Cada caso de uso constituye un flujo completo de eventos que muestra la interacción entre los actores y el sistema (es decir, qué van a hacer los actores). El actor primario es el encargado de empezar esta interacción y los casos de uso se muestran como respuesta al evento anterior. La ejecución del caso de uso acaba cuando algún actor genera un evento que requiere un caso de uso nuevo.

Aunque la idea es mantener el modelo de casos de uso lo más sencillo posible, existen ocasiones en los que la funcionalidad debe ser puesta en casos de uso separados o bien realizar una subdivisión del caso de uso. Existen dos enfoques para expresar variantes:

1. Si las diferencias entre los casos de uso son pequeñas, se pueden definir subflujos separados dentro de un mismo caso de uso.
2. Si las diferencias entre los casos de uso son grandes, se deben describir como casos de uso separados. Para estos casos de uso se utilizan principalmente las relaciones de *inclusión*, *extensión* y *generalización*.

Para la aplicación de este Trabajo Fin de Grado, la figura 2.3 muestra cinco casos de usos principales (*Obtener Recetas*, *Manejo Recetas*, *Registro/Loguearse*, *Manejo Favoritos* y *Manejo Perfil*), y tres casos de uso secundarios: *Visualizar Receta*, *Información Receta* y *Manejo de Información*.

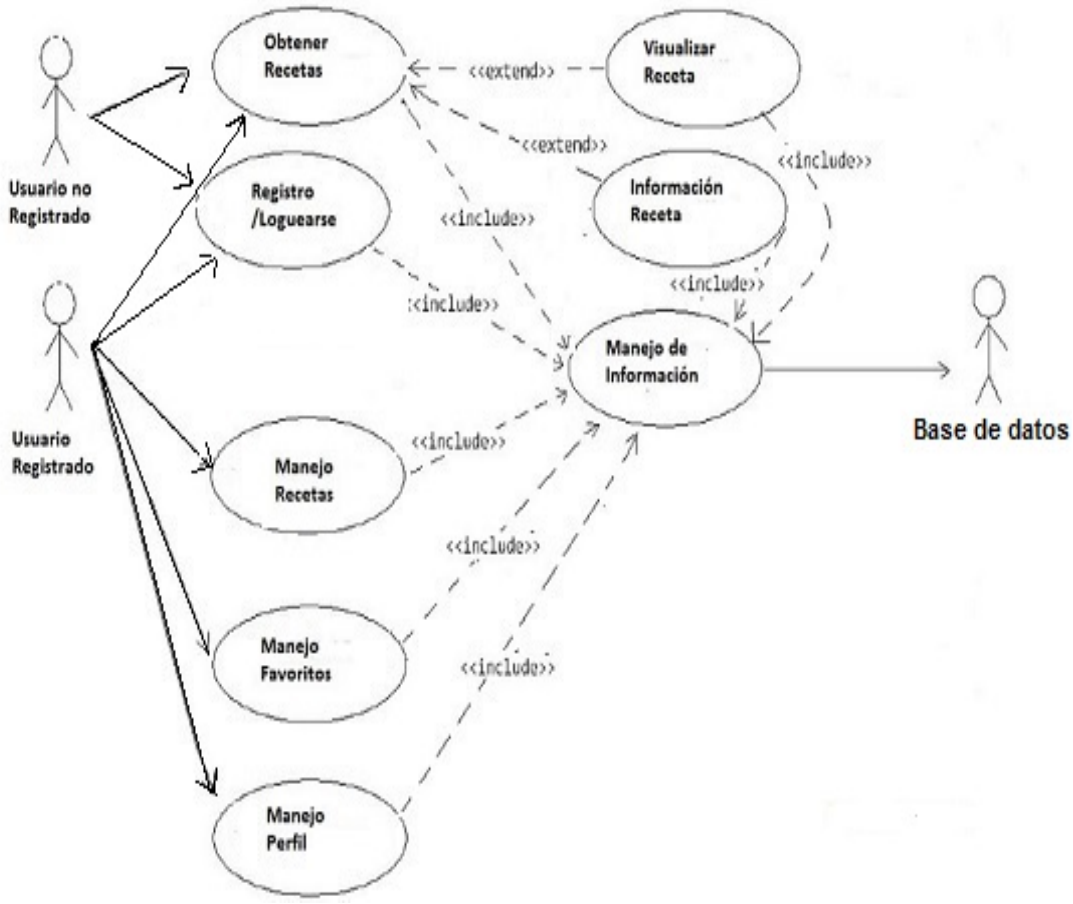


Figura 2.3: Diagrama de casos de uso para la Aplicación *AppRecetas* en Android

2.3.2.1. Inclusión

La **inclusión** se define como una sección de un caso de uso que es parte obligatoria del caso de uso básico. El caso de uso que se va a insertar depende del caso de uso anterior. Como se muestra en la figura 2.3, la notación para inclusión es la etiqueta «include».

En la aplicación a desarrollar (figura 2.3), el caso de uso *Manejo de Información* está incluido dentro de los casos *Visualizar Receta*, *Información Receta*, *Obtener Recetas*, *Manejo Recetas*, *Registro/Loguearse*, *Manejo Favoritos* y *Manejo Perfil* porque todos ellos necesitan cargar o modificar información incluida en la base de datos externa.

2.3.2.2. Extensión

La **extensión** se utiliza para modelar secuencias de eventos opcionales de casos de uso que, al manejarse de manera independiente, pueden ser insertados o eliminados. Especifica cómo un caso de uso puede insertarse en otro para extender la

función del anterior. El caso de uso donde se va a insertar la nueva funcionalidad debe ser independiente del caso de uso insertado. El caso de uso original se ejecuta hasta donde se inserta el nuevo caso de uso. Después de que este nuevo caso de uso haya terminado su función, el curso original de la secuencia continúa como si nada hubiera ocurrido.

Como se muestra en la figura 2.3, la notación para extensión es la etiqueta «extend». En esta figura se puede observar que el caso de uso *Visualizar Recetas* e *Información Receta* extienden del caso de uso *Obtener Recetas*, ya que una vez se obtenga la lista de las recetas, el usuario tiene la libertad de realizar alguna acción sobre ellas. (visualizar su realización o presentar su información).

2.3.2.3. Generalización

Una relación adicional entre casos de uso es la **generalización**, que apoya la reutilización de los casos de uso. Mediante esta relación es necesario describir las partes similares una sola vez, en lugar de repetirlas para todos los casos de uso de comportamiento común. En la generalización hay dos tipos de casos de uso:

- A los casos de uso que se extraen se les llama casos de uso *abstractos*, ya que sirven para describir partes que son comunes a otros casos de uso (no se instancian).
- A los casos de uso que realmente son instanciados se les conoce como casos de uso *concretos*.

En la aplicación desarrollada en este Trabajo Fin de Grado no se presenta ningún caso de uso de generalización.

2.3.3 Documentación

Como se ha dicho anteriormente, el modelo de casos de uso debe estar documentado, por lo que una de las principales partes de este modelo es una descripción detallada de cada uno de los actores y casos de uso identificados. El formato de documentación para los actores se muestra a continuación:

Actor *Nombre del actor*

- **Casos de uso**: Nombre del caso de uso o casos de uso en los que actúa.
- **Tipo**: *Primario* o *Secundario*.

- **Descripción**: Razón de ser del actor.

Las descripciones de los casos de uso representan todas las interacciones del actor o actores con el sistema. En esta etapa no se incluyen eventos internos del sistema, ya que se trata en la parte de diseño (en el que se analizarán los diagramas de secuencias). El formato de documentación para los casos de uso se muestra a continuación:

Caso de uso *Nombre del caso de uso*

- **Actores**: Actores primarios y secundarios que actúan en el caso de uso.
- **Tipo**: *Básico, Inclusión, Extensión, Generalización.*
- **Propósito**: Razón de ser del caso de uso.
- **Resumen**: Resumen del caso de uso.
- **Precondiciones**: Condiciones que se tienen que satisfacer para poder ejecutar el caso de uso.
- **Flujo Principal**: El flujo de eventos más importante del caso de uso, donde dependiendo de las acciones de los actores se continuará con alguno de los subflujos.
    - Los flujos secundarios del caso de uso o subflujos, numerados como (S-1), (S-2), etc.
- **Excepciones**: Excepciones que se pueden ocurrir durante el caso.
    - Excepciones numeradas como (E-1), (E-2), etc.

En la sección 2.5 se detallan los actores y casos de uso de la aplicación usando el formato indicado anteriormente.

2.4 Modelo de Interfaces

El **modelo de interfaces** describe la presentación de información entre los actores y el sistema. Para ello, se especifica cómo se verán las interfaces de usuario al ejecutar cada uno de los casos de uso, lo cual ayuda al usuario a visualizarlos según sean mostrados por el sistema. Cuando se diseñan las interfaces de usuario, es esencial tener a los usuarios involucrados, siendo de máxima importancia que las interfaces reflejen la visión lógica del sistema. En la sección 2.5.2, se muestran las interfaces usadas en la aplicación diseñada en este Trabajo de Fin de Grado a la vez que se describen los casos de uso de dicha aplicación.

2.5 Actores y casos de uso de la aplicación AppRecetas

En esta sección se muestra la documentación de los actores y casos de uso, junto con el diseño de las interfaces, que serán usadas como prototipo del sistema.

2.5.1 Actores

Se describen un total de tres actores en la aplicación: *Usuario Registrado*, *Usuario no Registrado* y *Base de Datos*.

Actor *Usuario Registrado*

- **Casos de uso**: Obtener Recetas, Manejo Recetas, Registro/Loguearse, Manejo Favoritos, Manejo Perfil.
- **Tipo**: *Primario*.
- **Descripción**: Representa a cualquier persona registrada que desee utilizar la aplicación *AppRecetas*.

Actor *Usuario no Registrado*

- **Casos de uso**: Registro/Loguearse, Obtener Recetas.
- **Tipo**: *Primario*.
- **Descripción**: Representa a cualquier persona no registrada que desee utilizar la aplicación *AppRecetas*.

Actor *Base de Datos*

- **Casos de uso**: Manejo de Información.
- **Tipo**: *Secundario*.
- **Descripción**: Representa el manejo de la información relacionada con la aplicación *AppRecetas*.

2.5.2 **Casos de uso**

Como apoyo en la descripción de los casos de uso de la aplicación se muestran las vistas propuestas para ella. Estas vistas se han diseñado mediante una herramienta de prototipado que se usará para realizar un test a los usuarios potenciales de la aplicación, con el fin de probar la usabilidad de ésta. El test de usabilidad y los resultados del mismo se presentan en el capítulo *Modelo de pruebas*. En el diseño de la interfaz de usuario se han planteado una serie de vistas (que constituyen las distintas pantallas de la aplicación) que se pasan a presentar a continuación.

En primer lugar, una vez se acceda a la aplicación, se le presentará al usuario la vista principal (figura 2.4).

Figura 2.4: Pantalla principal propuesta para la aplicación

Como se puede observar en la figura 2.4, la vista principal de la aplicación solo presenta una imagen característica de las Islas Canarias acompañada por la imagen de un plato de comida. Para poder navegar por la aplicación se hará uso del menú desplegable de la parte superior izquierda, que contendrá todos los ítem que permitirá al usuario "moverse" entre las distintas vistas.

Así, la figura 2.5, presenta los ítem del menú para el *Usuario no Registrado* (imagen de la izquierda) y para el *Usuario Registrado* (imagen de la derecha), cuyo significado es fácilmente reconocible por el nombre del ítem. A través del menú de la aplicación , cualquier usuario, registrado o no, tendrá acceso a las vistas: *Recetas*, mediante el ítem *Recetas* del menú (figura 2.6, donde se aprecia en la imagen de la izquierda, la vista para el *Usuario no Registrado* y en la imagen de la derecha, la vista para el *Usuario Registrado*), *Información*, seleccionando una receta de la lista (figura 2.7, donde se aprecia en la imagen de la izquierda, la vista para el *Usuario*

no Registrado y en la imagen de la derecha, la vista para el *Usuario Registrado*) y *Visualizar*, seleccionando una receta de la lista y presionando el icono de visualización que está en la parte superior derecha de la información de la receta (figura 2.8).

Figura 2.5: Ítem de menú para el *Usuario no Registrado* y para el *Usuario Registrado*

Figura 2.6: Ejemplo de pantalla de visualización de la lista de recetas para el *Usuario no Registrado* y para el *Usuario Registrado*

Como se muestra en la figura 2.6, las recetas se presentan en una lista ordenada alfabéticamente y separadas por categorías (*Entrantes*, *Platos*, *Postres*, entre otros. En la figura se presenta la lista de la categoría *Platos*). En esta figura, se observa que en la lista de recetas para el *Usuario no Registrado* (imagen de la izquierda), la vista tiene un botón flotante de registro (parte inferior derecha) para permitir al usuario registrarse. Asimismo, en la lista de recetas para el *Usuario Registrado*

Figura 2.7: Ejemplo de pantalla de visualización de la información de la receta para el *Usuario no Registrado* y para el *Usuario Registrado*

Figura 2.8: Pantalla propuesta de visualización del video

(imagen de la derecha), la vista tiene un botón flotante para agregar nueva receta (parte inferior derecha).

Al elegir una receta concreta de la lista, se presenta el detalle de la receta seleccionada (figura 2.7), en la que se pueden ver todos los pasos para realizarla. Esta vista es diferente según el usuario esté o no registrado. Así, para el *Usuario no Registrado*, la vista tiene dos botones: un botón flotante de registro (parte inferior derecha) y un botón de visualización del vídeo de la receta (parte superior derecha). Por otro

lado, para el *Usuario Registrado*, la vista tiene tres botones: un icono de favorito (en la parte inferior izquierda) para marcar la receta como favorita, un botón flotante para editar la receta en la parte inferior derecha (sólo aparece en el caso de que la receta haya sido añadida por el usuario) y un botón de visualización del vídeo de la receta (parte superior derecha).

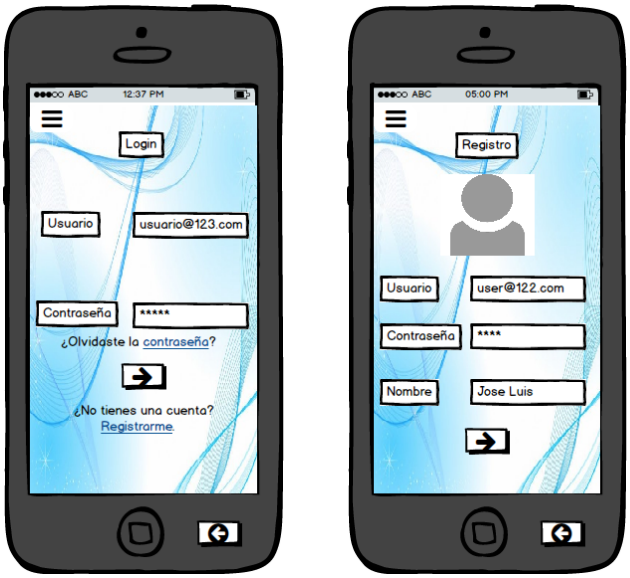


Figura 2.9: Pantallas propuestas para *loguearse* y registrarse

Si el usuario desea acceder a más características debe *loguearse* (imagen de la izquierda de la figura 2.9) o registrarse (imagen de la derecha de la figura 2.9), seleccionando el ítem de menú *Login* (figura 2.5) o presionando sobre el botón flotante para *loguearse*, que hay en algunas vistas en la parte inferior derecha (tal y como se muestra en la imagen de la izquierda de la figura 2.7). La imagen de la izquierda de la figura 2.9 muestra la pantalla donde el usuario debe introducir los datos de su cuenta para poder acceder a todas las características que ofrece la aplicación. Si el usuario es un *Usuario Registrado* y no recuerda su contraseña, puede seleccionar el botón "¿Olvidaste la contraseña?", y la aplicación le enviará un correo electrónico con la misma. En el caso de ser un *Usuario no Registrado* y querer realizar el registro como nuevo usuario, podrá hacerlo por medio del botón "Registrarme" de la vista de *login*. Una vez que el usuario haya seleccionado esta opción, pasará a visualizar la pantalla de registro (imagen de la derecha de la figura 2.9) en la cual se deberán rellenar todos los campos para proceder con el registro.

Desde que el usuario se haya registrado o *logueado* (*Usuario Registrado*), podrá navegar a la pantalla *Mi Perfil* (figura 2.10), a través del ítem del menú *Mi Perfil* (imagen de la derecha de la figura 2.5, donde se muestra el menú del *Usuario Registrado*). En la pantalla del perfil, el usuario podrá editar los datos de su cuenta o eliminar su cuenta.

Por otro lado, el *Usuario Registrado* podrá añadir una nueva receta a la aplicación (figura 2.11), seleccionando el botón flotante +, que aparece sobre la lista de recetas

Figura 2.10: Pantalla propuesta para el perfil

(imagen de la derecha de la figura 2.6). Una vez añadida, la receta estará disponible para que el resto de usuarios pueda verla.

Figura 2.11: Pantalla propuesta para crear una nueva receta

Asimismo, a través del ítem de menú *Mis Favoritos* se accede a la vista de recetas favoritas (imagen de la derecha de la figura 2.12), que es una pantalla a la que solo tienen acceso el *Usuario Registrado*, donde se mostrará una lista con las recetas favoritas del usuario. Cuando un *Usuario Registrado* presiona el icono **estrella** en

la vista del detalle de una receta (imagen de la derecha de la figura 2.7), la receta pasa a formar parte de la lista de favoritos del usuario, y el icono estrella aparece marcado (imagen de la izquierda de la figura 2.12). A través de la lista de favoritos, el usuario también tiene la posibilidad de eliminar recetas de esa lista.

Figura 2.12: Pantallas propuestas para tratar los favoritos

Figura 2.13: Pantalla propuesta para manejar la lista de recetas agregadas por el *Usuario Registrado*

Asimismo, el *Usuario Registrado* puede visualizar las recetas que él o ella ha introducido en la aplicación (figura 2.13) o las recetas que están a medio definir (todavía no están a la vista de todos los usuarios porque el usuario no las ha subido a la nube), seleccionando el ítem *Mis Recetas* del menú. En la vista que presenta

la lista de recetas del usuario, éste puede añadir nuevas recetas (seleccionando el botón flotante), editar o eliminar recetas ya existentes (a través de los iconos lápiz y papelera de la parte superior derecha).

Por último, tanto el *Usuario Registrado* con el *Usuario no Registrado* pueden realizar búsquedas de recetas mediante un filtro. En la figura 2.14 se muestran todas las pantallas que verá el usuario al realizar una búsqueda de recetas. En la última de ellas, se presenta una lista con las recetas que se ajusten a la información de filtrado.

Figura 2.14: Pantallas propuestas para la búsqueda de recetas

2.5.2.1. Caso de uso Registro/Loguearse

En el caso de uso *Registro/Lgouearse*, el usuario puede introducir sus datos para *loguearse* y poder acceder así a la aplicación o registrarse creando una cuenta, con la que tendrá acceso a más características dentro de la aplicación. Una explicación más detallada del caso de uso se presenta a continuación:

Caso de uso *Registro/Loguearse*

- **Actores**: Usuario Registrado, Usuario no Registrado.
- **Tipo**: *Básico.*
- **Propósito**: Permite al usuario no registrado, registrarse y al usuario registrado, "loguearse", para poder acceder a la aplicación.

- **Resumen**: Este caso de uso comienza cuando el usuario accede al apartado *Login* situado en el menú de la aplicación.
- **Precondiciones**: El usuario debe acceder al apartado *Login* del menú.
- **Flujo Principal**: Al acceder a la aplicación, se le presenta al usuario la pantalla principal de ésta (figura 2.4). Al acceder al apartado "Login" del menú (figura 2.5), se le presentará al usuario una pantalla donde poder introducir sus datos de acceso (imagen de la izquierda de la figura 2.9).
  - (S-1) Recuperar la contraseña. Cuando el usuario está registrado, puede recuperar su contraseña olvidada seleccionado el botón "¿Olvidaste la contraseña?". La aplicación, cuando esto ocurre, envía un correo electrónico al usuario para recordársela.
  - (S-2) Registrarse. Cuando el usuario no está registrado, puede seleccionar el botón "Registrarme", presentándosele una pantalla para realizar el registro como nuevo usuario (imagen de la derecha de la figura 2.9).
- **Excepciones**:
  - (E-1) Error de usuario no registrado: se produce cuando un usuario que no está registrado intenta acceder a la aplicación.

2.5.2.2. Caso de uso Obtener Recetas

Este caso de uso está relacionado con la obtención de las recetas almacenadas en la aplicación, presentándose una lista con todas ellas. De este caso se extienden los casos de uso *Visualizar Receta* e *Información Receta*. A continuación, se muestra este caso de uso en detalle:

Caso de uso *Obtener Recetas*

- **Actores**: Usuario Registrado y Usuario no Registrado.
- **Tipo**: *Básico*.
- **Propósito**: Presentar al usuario, la lista de todas las recetas disponibles.
- **Resumen**: Este caso de uso es iniciado cuando el usuario accede a la aplicación, independientemente de si está o no registrado en ella, y selecciona el ítem *Recetas* del menú. Una vez realizada esta acción, se presenta una lista con las recetas disponibles y una serie de iconos relacionados con ella.
- **Precondiciones**: El usuario debe acceder al ítem *Recetas* del menú y seleccionar un tipo de plato específico.

- **Flujo Principal**: El usuario, después de seleccionar el ítem *Recetas* del menú y el tipo de plato (figura 2.5), accede a la vista donde se le muestra la lista con todas las recetas. En función de que el usuario esté o no registrado, se presentan iconos diferentes: de registro o de agregación de nueva receta (figura 2.6). Este proceso se realiza a través del caso de uso *Registro/Loguearse* y del caso de uso *Manejo Recetas*, respectivamente. Asimismo, si el usuario quiere analizar una determinada receta de la lista, presiona sobre ella pasando al caso de uso *Visualizar Receta* o *Información Receta*.
    - (S-1) Filtrar las recetas. Si el usuario presiona sobre la barra de búsqueda, e introduce un texto a buscar, se presenta una lista con las recetas que se ajusten a los criterios de filtrado (figura 2.14).
- **Excepciones**: Ninguna.

2.5.2.3. Caso de uso Visualizar Recetas

Este caso de uso permite el usuario visualizar el video sobre la realización de la receta elegida. El detalle es como sigue:

Caso de uso *Visualizar Recetas*

- **Actores**: Usuario Registrado y Usuario no Registrado.
- **Tipo**: *Extensión*.
- **Propósito**: Presenta al usuario un vídeo sobre la realización de la receta.
- **Resumen**: Este caso de uso es iniciado por alguno de los usuarios principales, donde podrá visualizar el vídeo de la realización de una receta seleccionada previamente de la lista de recetas.
- **Precondiciones**: El usuario debe haber seleccionado una receta de la lista de recetas.
- **Flujo Principal**: El usuario accederá a esta caso de uso si previamente hace una elección de alguna receta de la lista mostrada la figura 2.6. Una vez se encuentre en la vista que contiene la información de la receta (figura 2.7), el usuario podrá presionar sobre el botón de la esquina superior derecha de la pantalla, que le dará acceso al vídeo sobre dicha receta (figura 2.8). Una vez terminada la presentación del vídeo, el usuario podrá volver a la vista de la información de la receta.
- **Excepciones**:
    - (E-1) Error de acceso al vídeo buscado: el vídeo seleccionado no se encuentra en la nube o el enlace al vídeo es erróneo.

2.5.2.4. Caso de uso Información Receta

En este caso de uso, el usuario analiza la descripción detallada sobre los ingredientes y preparación de la receta seleccionada con anterioridad. El detalle de este caso de uso, se muestra a continuación:

Caso de uso *Información Receta*

- **Actores**: Usuario Registrado y Usuario no Registrado.
- **Tipo**: *Extensión*.
- **Propósito**: Permite visualizar toda la información perteneciente a la receta.
- **Resumen**: Este caso de uso se inicia en el momento en el que el usuario selecciona una receta, presentándose la información relacionada a ella e iconos de manejo de la misma.
- **Precondiciones**: El usuario debe haber seleccionado una receta de la lista de recetas.
- **Flujo Principal**: Después de que el usuario selecciona una receta de la lista de recetas, se presenta una vista con toda la información referente a la receta seleccionada (figura 2.7). En función de que el usuario esté o no registrado, se presentan iconos diferentes: de registro o de edición de la receta. Este proceso se realiza a través del caso de uso *Registro/Loguearse* y del caso de uso *Manejo Recetas*, respectivamente.
- **Excepciones**:
  - (E-1) Error de acceso a la receta buscada: no existe información de la receta buscada en la nube.

2.5.2.5. Caso de uso Manejo Recetas

Este caso de uso contempla la creación de nuevas recetas, la edición y la eliminación de las recetas publicadas anteriormente. En detalle, se tiene:

Caso de uso *Manejo Recetas*

- **Actores**: Usuario Registrado.
- **Tipo**: *Básico*.

- **Propósito**: Realizar diversas tareas de manipulación de las recetas.
- **Resumen**: Este caso de uso se inicia por el usuario cuando quiera interactuar con las recetas propias o crear una nueva receta.
- **Precondiciones**: El usuario debe ser estar registrado y debe acceder a la lista de recetas o a la información de una receta particular suya o de otro usuario.
- **Flujo Principal**: Desde la lista de recetas disponibles, el usuario puede agregar una nueva usando el botón flotante + (imagen de la derecha de la figura 2.6). Asimismo, en la vista de información de una receta propia seleccionada de la lista, el usuario podrá editarla, seleccionando el botón lápiz (imagen de la derecha de la figura 2.7). Por otro lado, desde la lista de recetas propias (obtenida al seleccionar el botón ítem *Mis Recetas* del menú (figura 2.5), el usuario podrá: agregar nueva receta, editar o eliminar la receta que desee (figura 2.13).
    - (S-1) Crear nueva receta. El usuario puede crear una nueva receta desde la lista de recetas de la aplicación (imagen de la derecha de la figura 2.6) o desde su propia lista de recetas (figura 2.13). Cuando el usuario presione el botón +, se abrirá una nueva vista (figura 2.11), donde el usuario tendrá que rellenar un formulario. Después de rellenar el formulario, el usuario puede decidir guardar la receta sin compartir (porque la receta no está completa) seleccionando el botón de la izquierda de la esquina superior derecha de la figura 2.11 o puede decidir guardarla para compartir seleccionando el botón de la derecha de la esquina superior derecha de la figura 2.11.
    - (S-2) Editar receta. El usuario puede cambiar la información que introdujo al crear una de sus recetas (desde su propia lista de recetas, mostrada en la figura 2.13, marcando la receta y presionando el icono lápiz; o desde la vista de información de una receta (imagen de la derecha de la figura 2.7) seleccionando el icono de lápiz. Cuando se presiona el icono lápiz, se muestra una vista con la información de la receta preparada para ser modificada (figura 2.15). Como se puede observar, esta vista es similar a la vista para agregar una nueva receta (figura 2.11).
    - (S-3) Eliminar Receta. El usuario puede eliminar una receta creada por él con anterioridad. Para ello, en la lista de recetas propias (figura 2.13) se marca la receta y se presiona el icono papelera. Al presionar este icono, la receta queda borrada de la aplicación.
- **Excepciones**:
    - (E-1) Error de receta no propia: La receta, que el usuario quiere editar no fue añadida por el usuario.
    - (E-2) Error de formulario: Los campos del formulario de agregación o edición se encuentran vacíos.

Figura 2.15: Pantalla propuesta para la edición de una receta

2.5.2.6. Caso de uso Manejo Favoritos

El caso de uso *Manejo Favoritos* hace posible que el usuario pueda seleccionar y añadir las recetas que desee a una lista de favoritos, para acceder rápidamente a ellas. El detalle del caso de uso es el siguiente:

Caso de uso *Manejo Favoritos*

- **Actores**: Usuario Registrado.
- **Tipo**: *Básico*.
- **Propósito**: Presenta al usuario la posibilidad de incluir en un grupo sus recetas favoritas.
- **Resumen**: El caso de uso se inicia cuando el usuario presiona el botón añadir a favoritos presente en la pantalla donde se muestra la información de la receta.
- **Precondiciones**: El usuario debe estar registrado y previamente debe acceder a la información de una receta.
- **Flujo Principal**: En la vista de información de una receta, el usuario puede añadir recetas nuevas a su lista de favoritos. Al añadir la receta a esta lista se consigue un acceso más rápido a ella. Una vez marcada alguna receta como favorita, se le mostrará al usuario en la pantalla al acceder al ítem "Mis Favoritos" del menú (figura 2.5).

  - (S-1) Listar recetas favoritas. Seleccionando el ítem de menú *Mis Favoritos*, el usuario accede a una lista con las recetas favoritas (imagen de la derecha de la figura 2.12).
  - (S-2) Eliminar Favorito. El usuario puede borrar de su lista de favoritos, una receta previamente añadida. Para ello, presiona sobre el icono papelera correspondiente a esa receta (imagen de la derecha de la figura 2.12).
  - (S-3) Agregar recetas favoritas. En la vista de información de una receta (imagen de la derecha de la figura 2.7), el usuario presiona el botón estrella (parte inferior izquierda de la pantalla), para indicar que la receta se debe añadir a su lista de recetas favoritas. Una vez seleccionada la receta como favorita, se le informa al usuario del proceso realizado (imagen de la izquierda de la figura 2.12).

- **Excepciones**: Ninguna.

2.5.2.7. Caso de uso Manejo Perfil

Este caso de uso permite al usuario registrado poder visualizar y manejar los datos de su perfil dentro de la aplicación. El detalle es el siguiente:

Caso de uso *Manejo Perfil*

- **Actores**: Usuario Registrado.
- **Tipo**: *Básico*.
- **Propósito**: Se presenta al usuario la información perteneciente a su perfil para manejarla.
- **Resumen**: Este caso de uso se inicia cuando el usuario presiona el ítem "Mi Perfil" del menú (imagen de la derecha de la figura 2.5).
- **Precondiciones**: El usuario debe estar registrado en la aplicación.
- **Flujo Principal**: Se presenta al usuario una pantalla (figura 2.10) con toda la información correspondiente a la información de su perfil.
  - (S-1) Editar perfil. El usuario puede editar todos los datos de su perfil presionando el icono lápiz de la parte superior derecha de la vista del perfil (figura 2.10).
  - (S-2) Eliminar perfil. El usuario puede eliminar su cuenta, presionando el icono papelera de la parte superior derecha de la vista del perfil (figura 2.10).

- **Excepciones**:
  - (E-1) Error de formulario: Los campos del formulario del perfil se encuentran vacíos.

2.5.2.8. Caso de uso Manejo de información

El caso de uso *Manejo de Información*, está relacionado a todos los casos de uso. Es el encargado de guardar y cargar datos que se visualizarán en el resto de casos.

Caso de uso *Manejo de Información*

- **Actores**: Base de Datos.
- **Tipo**: *Inclusión.*
- **Propósito**: Manejar todos los datos que se almacenan y se recuperan en y desde la nube.
- **Resumen**: Se inicia cuando el resto de casos de uso necesita guardar o recuperar cualquier tipo de información.
- **Precondiciones**: Tener acceso a Internet.
- **Flujo Principal**: El acceso a la base de datos se realiza en todos los procesos de la aplicación.
  - (S-1) Registro/Loguearse. Se añade un nuevo usuario o concede el acceso de un usuario antiguo a la aplicación.
  - (S-2) Mostrar lista de recetas. Recupera la lista de todas las recetas existentes para ser presentada al usuario.
  - (S-3) Mostrar vídeo. Accede a la dirección URL de un vídeo para ser mostrado al usuario.
  - (S-4) Mostrar información de la receta. Recupera la información de una receta seleccionada para ser presentada al usuario.
  - (S-5) Manejar receta. Recupera la información de una receta, agrega una nueva, edita o elimina una ya existen.
  - (S-6) Mostrar favoritos. Recupera la lista de recetas marcadas como favoritas para ser presentada al usuario.

- (S-7) Mostrar perfil. Recupera la información sobre el perfil del usuario para presentarla al usuario.

- **Excepciones**:
  - (E-1) Error de acceso a Internet: error en el acceso a la información en la nube.

Capítulo **3**

MODELO DE DISEÑO

3.1 Descripción de la arquitectura MVP

La arquitectura MVP o *Modelo-Vista-Presentador* [18] es una arquitectura que tiene como objetivo separar la interfaz de usuario de la lógica de las aplicaciones.Idealmente el patrón MVP permitiría conseguir que una misma lógica pudiera tener vistas totalmente diferentes e intercambiables. Básicamente, esta arquitectura consiste en tres componentes (figura 3.1).

La idea básica es que la clase Presentador haga de intermediario entre la Vista (la interfaz gráfica de usuario) y el modelo de datos.

- La *vista*. Compuesta de las ventanas y controles que forman la interfaz de usuario de la aplicación.
- El *modelo*. Donde se lleva a cabo toda la lógica de negocio.
- El *presentador*. Escucha los eventos que se producen en la vista y ejecuta las acciones necesarias a través del modelo. Además, puede tener acceso a las vistas a través de las interfaces que la vista debe implementar.

La explicación de esta arquitectura es bastante sencilla. Por un lado, se tiene la vista, que se encarga de mostrar la información al usuario y de interactuar con él para hacer ciertas operaciones. Por otro lado, está el modelo que, ignorante de cómo la información es mostrada al usuario, realiza toda la lógica de negocio usando las entidades del dominio. Y por último, se tiene al presentador que es el que "presenta" a ambos componentes sin que haya ningún tipo de dependencia entre ellos.

La vista en la arquitectura MVP debe ser pasiva, es decir, cuando el usuario provoque un evento sobre la interfaz de usuario, la vista debe delegar en el presentador el tratamiento de ese evento. A su vez, el modelo realiza las acciones que le solicita el presentador cuando corresponda (acciones como recuperar información de una base de datos interna o externa, entre otros). Cuando la información solicitada está disponible, el modelo "notifica" que los datos están accesibles y el presentador, que está a la espera, "observando" al modelo (objeto "observado") responderá a la

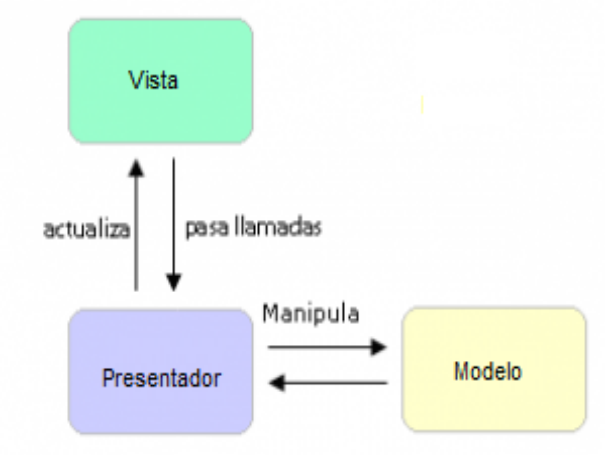


Figura 3.1: Figura representa la arquitectura MVP

notificación como corresponda a la aplicación desarrollada. Es importante indicar que en la arquitectura MVP, el modelo "no conoce" al presentador ni a la vista.

Además de los tres componentes indicados, la aplicación a desarrollar contará con un componente extra, al cual se llamará, *Mediador*, que realizará aquel proceso que no es particular de ningún presentador en concreto, sino que se encargará de tareas que todos los presentadores tendrán que realizar y que por conveniencia, se concentrarán en un único punto del código. Normalmente, estas operaciones tienen que ver con el sistema operativo del terminal.

Por último, en la descripción de esta arquitectura se puede ver que todos los componentes están perfectamente desacoplados, lo que implica que cualquier cambio sobre cualquier componente, no afecta al resto. Asimismo, el código final puede ser probado con relativamente poco esfuerzo y de forma simultánea a la implementación de los diferentes componentes.

3.1.1 Clases identificadas en la aplicación

Una vez completado el *Modelo de requisitos* se pueden hacer un primer boceto de las clases identificadas en la aplicación. Así, por cada pantalla de la aplicación (según el prototipo realizado), habrá una clase relacionada (se denominan clases de la vista). Asimismo, con cada clase de la vista habrá también una clase relacionada (denominadas clases del presentador). Por último, habrá una clase para el modelo que se encargará del manejo de los datos. Según este criterio, las clases identificadas en esta aplicación son:

- Clase *VistaPrincipal* con su interfaz *IVistaPrincipal* (relacionada con la figura 2.4).

- Clase *VistaLogin* con su interfaz *IVistaLogin* (relacionada con la imagen de la izquierda de la figura 2.9).
- Clase *VistaRegistro* con su interfaz *IVistaRegistro* (relacionada con la imagen de la derecha de la figura 2.9).
- Clase *VistaListaRecetas* con su interfaz *IVistaListaRecetas* (relacionada con la vista mostrada en la figura 2.6).
- Clase *VistaReceta* con su interfaz *IVistaReceta* (relacionada con la vista mostrada en la figura 2.7).
- Clase *VistaVideo* con su interfaz *IVistaVideo* (relacionada con la vista mostrada en la figura 2.8).
- Clase *VistaAgregarEditarReceta* con su interfaz *IVistaAgregarEditarReceta* (relacionada con la vista mostrada en la figura 2.11 y 2.15).
- Clase *VistaPerfil* con su interfaz *IVistaPerfil* (relacionada con la vista mostrada en la figura 2.10).
- Clase *VistaFavoritos* con su interfaz *IVistaFavoritos* (relacionada con la imagen de la derecha de la la figura 2.12).
- Clase *VistaMisRecetas* con su interfaz *IVistaMisRecetas* (relacionada con la vista mostrada en la figura 2.13).
- Clase *PresentadorPrincipal* con su interfaz *IPresentadorPrincipal* (relacionada con la vista *VistaPrincipal*).
- Clase *PresentadorLogin* con su interfaz *IPresentadorLogin* (relacionada con la vista *VistaLogin*).
- Clase *PresentadorRegistro* con su interfaz *IPresentadorRegistro* (relacionada con la vista *VistaRegistro*).
- Clase *PresentadorListaRecetas* con su interfaz *IPresentadorListaRecetas* (relacionada con la vista *VistaListaRecetas*).
- Clase *PresentadorReceta* con su interfaz *IPresentadorReceta* (relacionada con la vista *VistaReceta*).
- Clase *PresentadorVideo* con su interfaz *IPresentadorVideo* (relacionada con la vista *VistaVideo*).
- Clase *PresentadorAgregarEditarReceta* con su interfaz *IPresentadorAgregarEditarReceta* (relacionada con la vista *VistaAgregarEditarReceta*).
- Clase *PresentadorPerfil* con su interfaz *IPresentadorPerfil* (relacionada con la vista *VistaPerfil*).

- Clase *PresentadorFavoritos* con su interfaz *IPresentadorFavoritos* (relacionada con la vista *VistaFavoritos*).
- Clase *PresentadorMisRecetas* con su interfaz *IPresentadorMisRecetas* (relacionada con la vista *VistaMisRecetas*).
- Clase *Modelo* con su interfaz *IModelo*.

3.2 Diagramas de secuencia

Partiendo del prototipo realizado en el *Modelo de requisitos*, se tratará de descubrir los métodos a implementar en cada clase identificada en el apartado anterior. Para ello, se hará uso de los diagramas de secuencia [19] realizados con **Draw.io** [20]. Draw es una increíble herramienta que nos permite elaborar diagramas en linea sin necesidad de instalar absolutamente nada en nuestro PC. Su interfaz es bastante sencilla y fácil de utilizar, además es tan completa que nada tiene que envidiarle a cualquier software de pago para escritorio.

Un diagrama de secuencias muestra la interacción de un conjunto de objetos de una aplicación a través del tiempo, en el cual se indicaran los módulos o clases que formaran parte del programa y las llamadas que se hacen cada uno de ellos para realizar una tarea determinada, por esta razón permite observar la perspectiva cronológica de las interacciones. Es importante recordar que el diagrama de secuencias se realiza a partir de la descripción de un caso de uso.

El eje vertical representa el tiempo, y en el eje horizontal se colocan los objetos y actores participantes en la interacción, sin un orden prefijado. Cada objeto o actor tiene una línea vertical, y los mensajes se representan mediante flechas entre los distintos objetos. El tiempo fluye de arriba abajo.

3.2.1 Diagrama de secuencia para Loguearse

La figura 3.2, representa el flujo de mensajes intercambiados entre la vista, el modelo y el presentador. Desde esta pantalla, el usuario puede loguearse, acceder a la pantalla destinada a realizar el registro o recuperar su contraseña en el caso de de haberla olvidado.

3.2.2 Diagrama de secuencia para Registrarse

En la figura 3.3 se muestran los mensajes intercambiados entre modelo, vista y presentador de la pantalla de registro en la que el usuario puede registrarse en la aplicación.

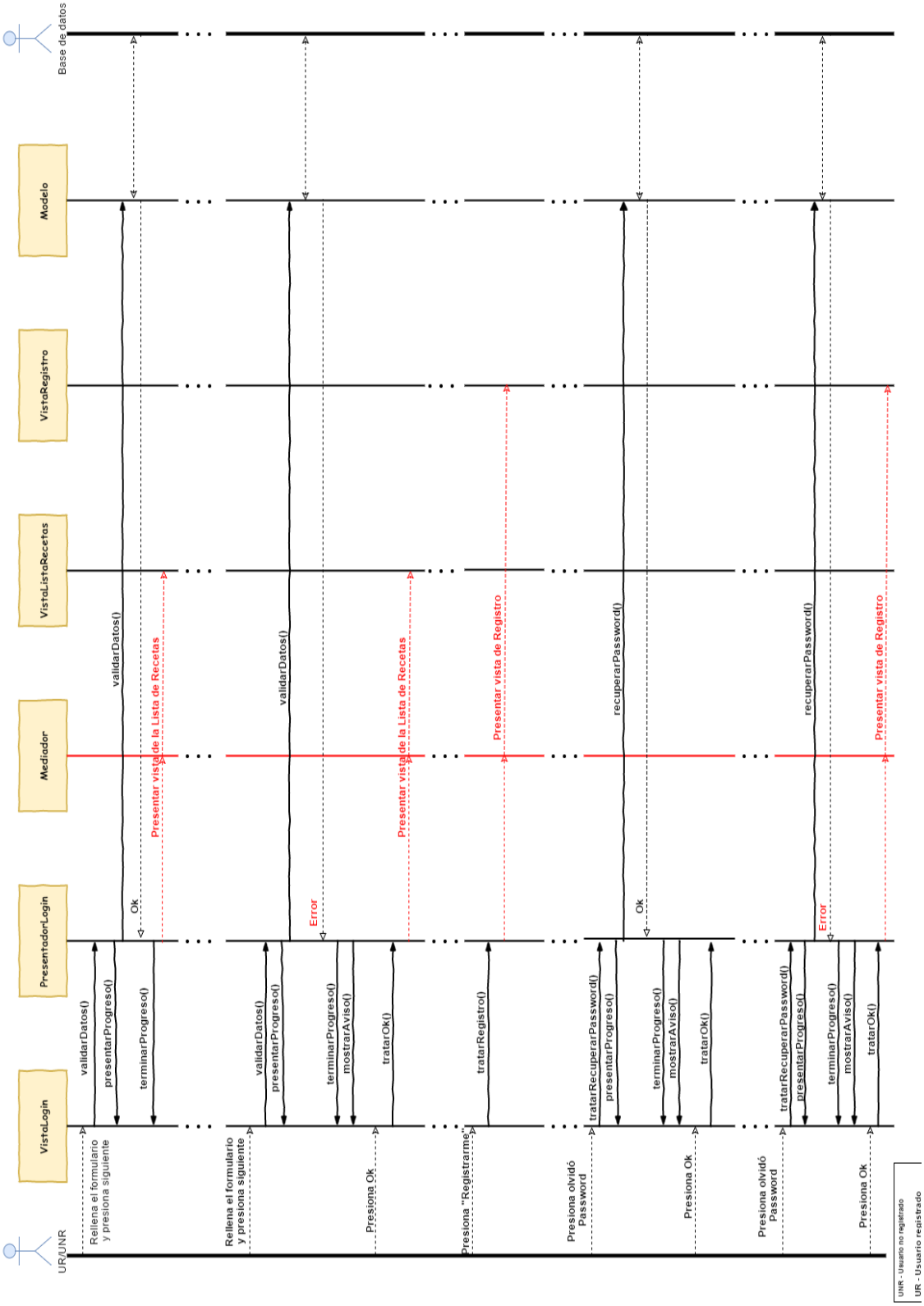


Figura 3.2: Diagrama de secuencia *Login*

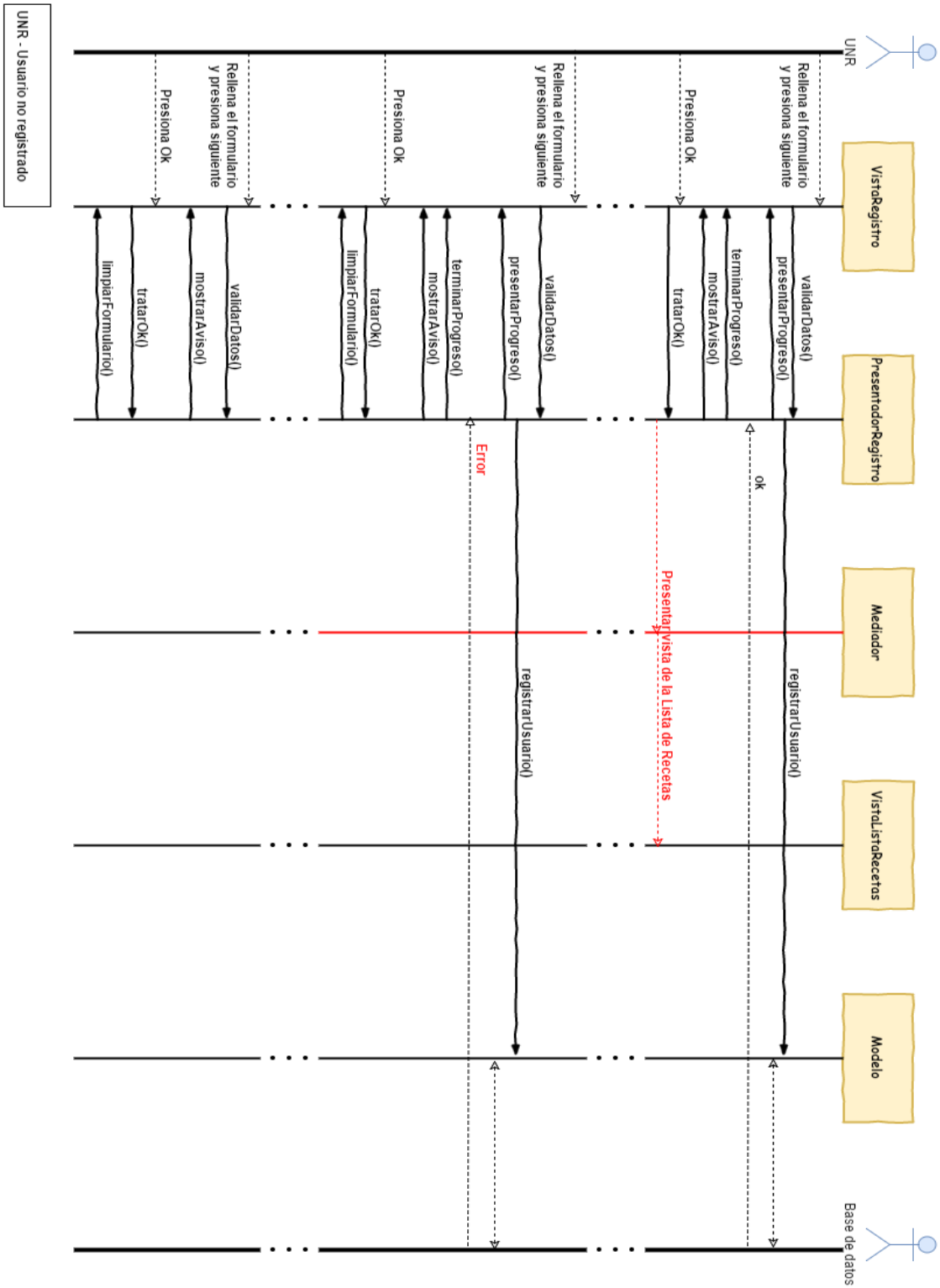


Figura 3.3: Diagrama de secuencia *Registro*

3.2.3 Diagrama de secuencia para la Lista de Recetas

En la figura 3.4 se muestra el flujo de mensajes intercambiado entre el modelo, la vista y el presentador. En primer lugar se debe cargar la categoría elegida con las recetas pertenecientes a dicha categoría.

Los usuarios no registrados podrán acceder a la información específica de una receta elegida, buscar una receta dentro de la categoría o registrarse en la aplicación accediendo a la pantalla para dicho fin.

Los usuarios registrados pueden acceder a la información específica de una receta, buscar una receta dentro de la categoría y acceder a la vista dedicada a la creación de una nueva receta.

3.2.4 Diagrama de secuencia para la Receta

La figura 3.5 representa el flujo de mensajes intercambiados entre el modelo, vista y presentador. El usuario, tanto registrado como no registrado, llega a esta pantalla al pulsar sobre una receta de alguna de las categorías. En ese momento, se debe cargar en la pantalla toda la información detallada sobre la receta en cuestión.

Desde esa pantalla, el usuario no registrado puede acceder a la pantalla para visualizar un video sobre la receta o a la pantalla para registrarse como nuevo usuario en la aplicación.

El usuario registrado puede acceder a la vista para visualizar un video sobre la receta, añadir o eliminar la receta de su lista de recetas favoritas o en el caso de que sea una receta creada por el usuario, puede acceder a la vista para *agregar editar recetas*.

3.2.5 Diagrama de secuencia para el Video

La figura 3.6 representa el flujo de mensajes intercambiados entre el modelo, la vista y presentador para que tanto el usuario no registrado como el usuario registrado, puedan visualizar el video sobre la receta seleccionada.

3.2.6 Diagrama de secuencia para el Perfil

La figura 3.7 representa el flujo de mensajes intercambiados entre el modelo, la vista y el presentador. Esta pantalla sólo está disponible para los usuarios registrados a los que se les presentará su perfil. Desde esta pantalla el usuario puede editar su perfil o eliminar su cuenta de usuario.

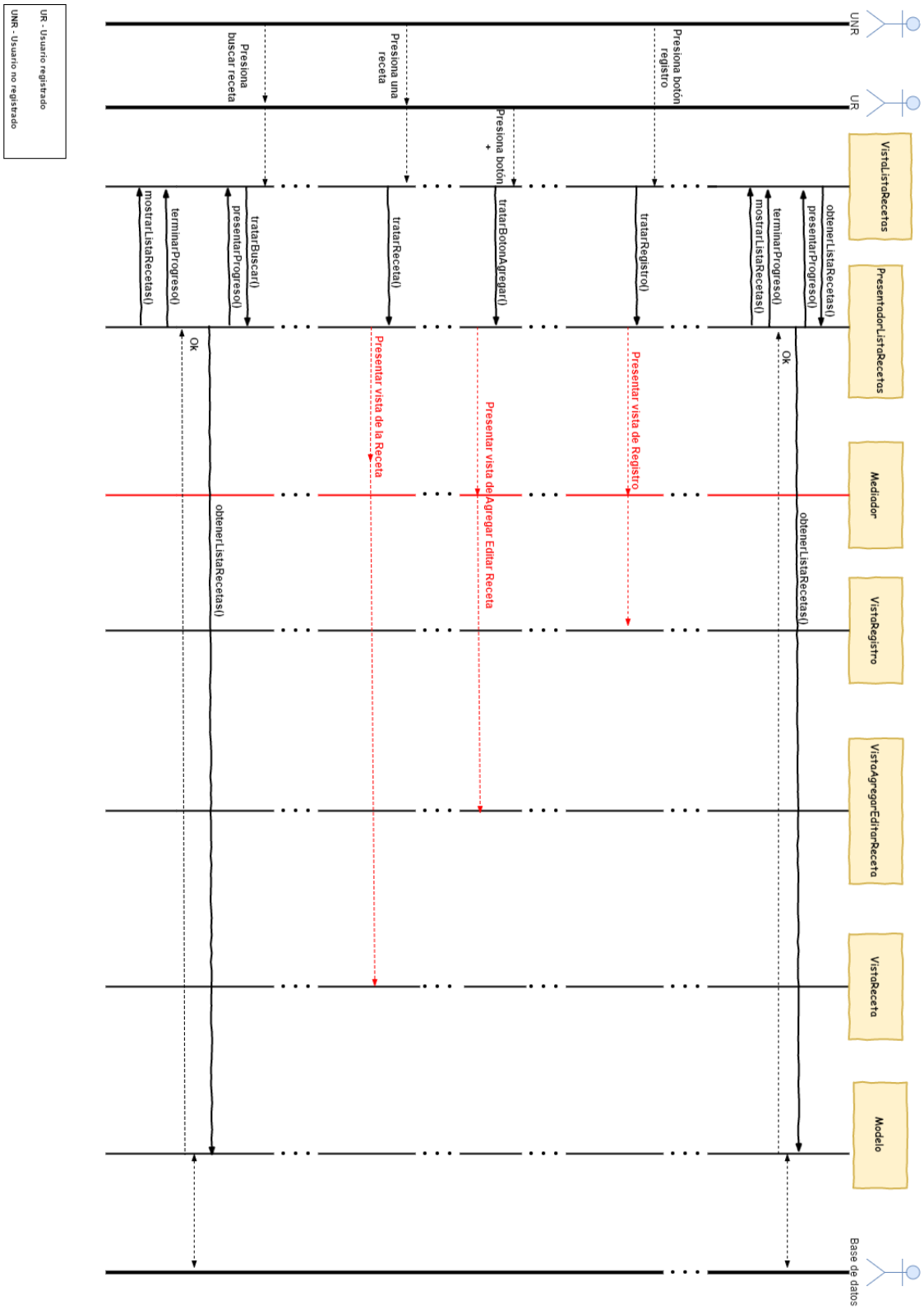


Figura 3.4: Diagrama de secuencia *Lista de Recetas*

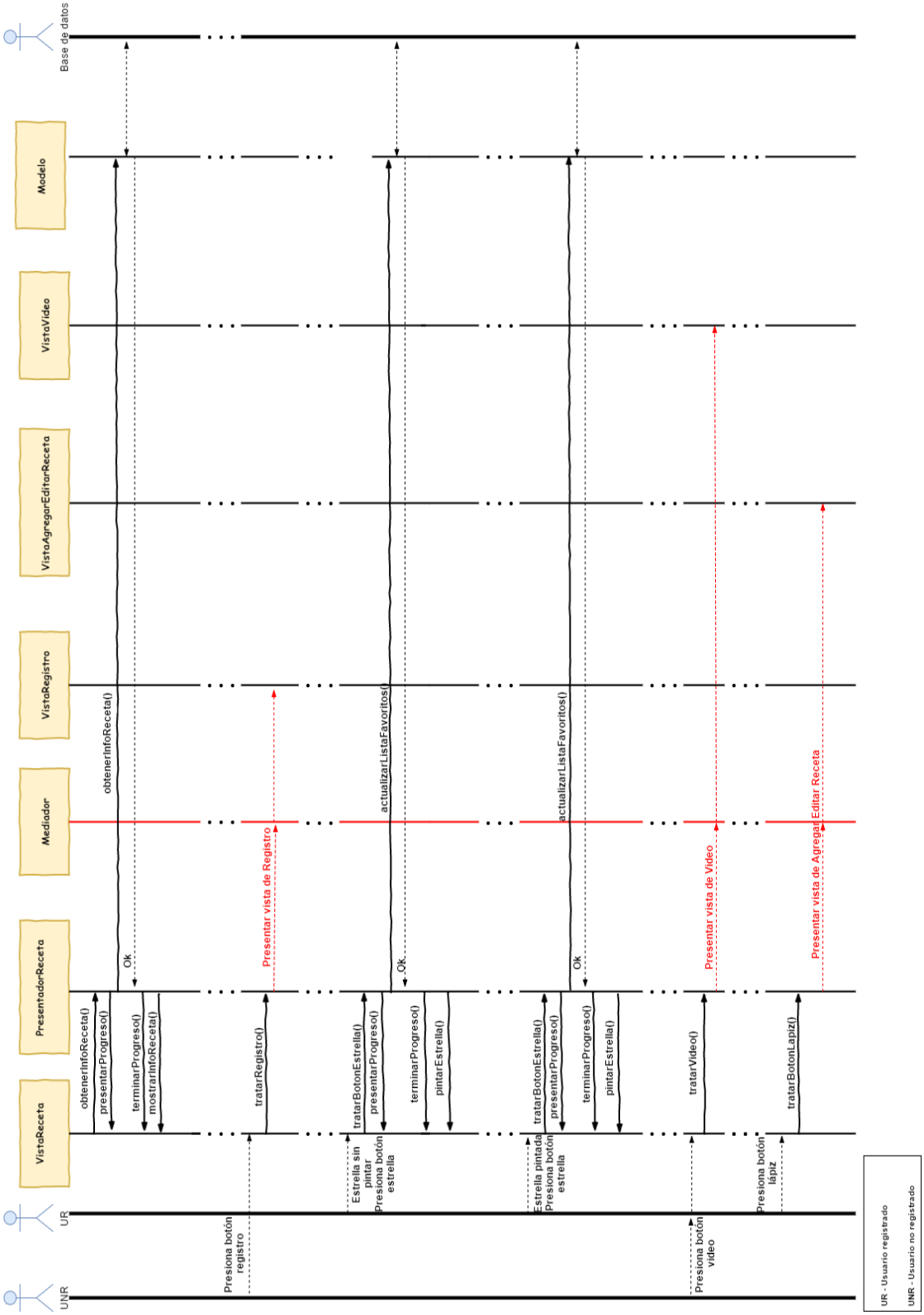


Figura 3.5: Diagrama de secuencia *Receta*

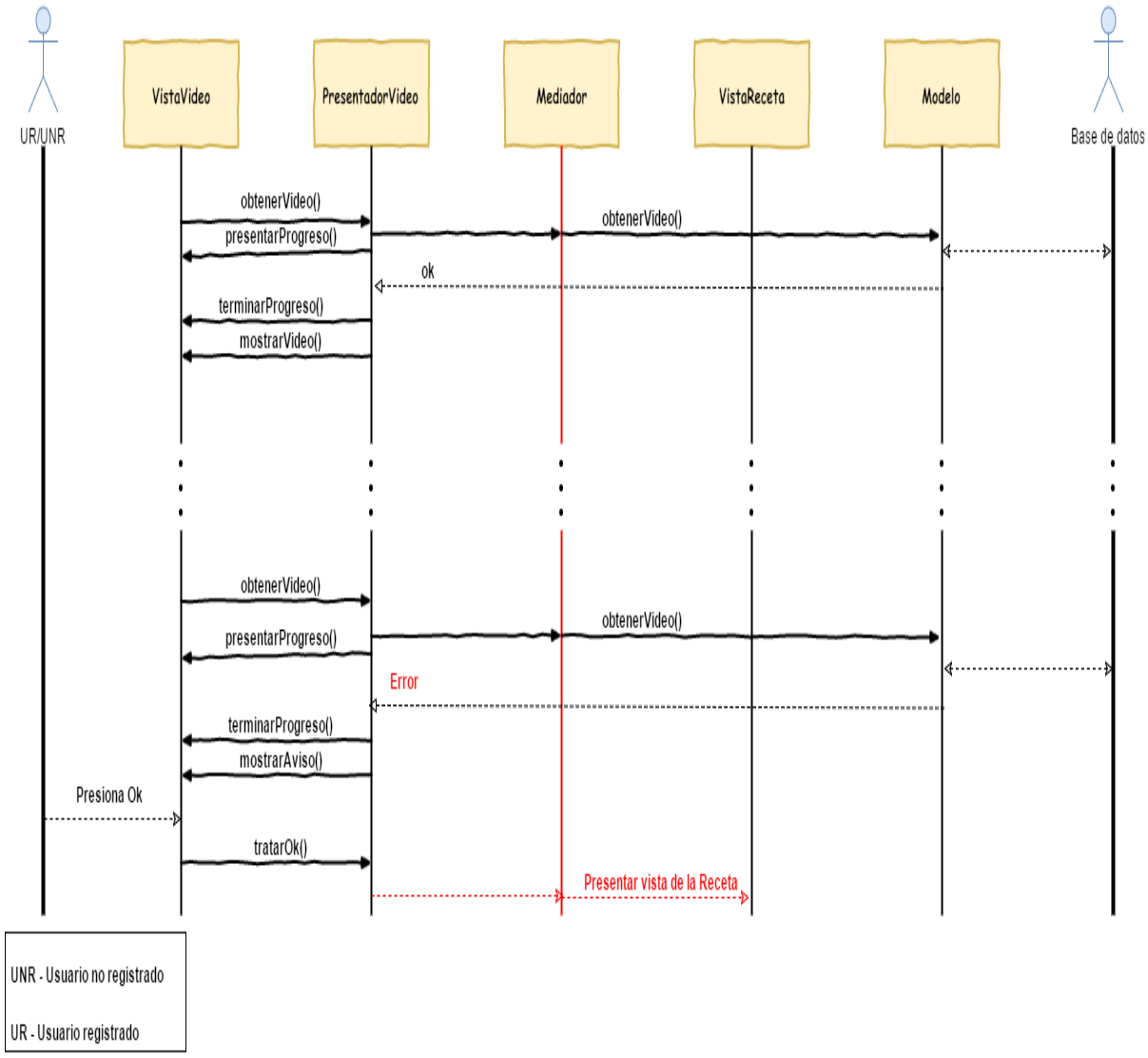


Figura 3.6: Diagrama de secuencia *Video*

3.2.7 **Diagrama de secuencia para Agregar Editar Receta**

En las figuras 3.8 y 3.9, se puede observar el flujo de mensajes intercambiados entre el modelo, la vista y el presentador. Es una vista a la que sólo pueden acceder los usuarios registrados, los cuales podrán acceder de dos maneras diferentes. En el caso de que el usuario quiera editar una receta creada por él, la vista mostrará la información guardada sobre la receta para que pueda modificar el contenido. En el caso de que el usuario quiera añadir una nueva receta, se mostrarán todos los campos vacíos.

El usuario puede guardar la receta para continuar con su edición en otro momento o puede subir la receta, momento en el que se añadirá a la categoría que pertenezca.

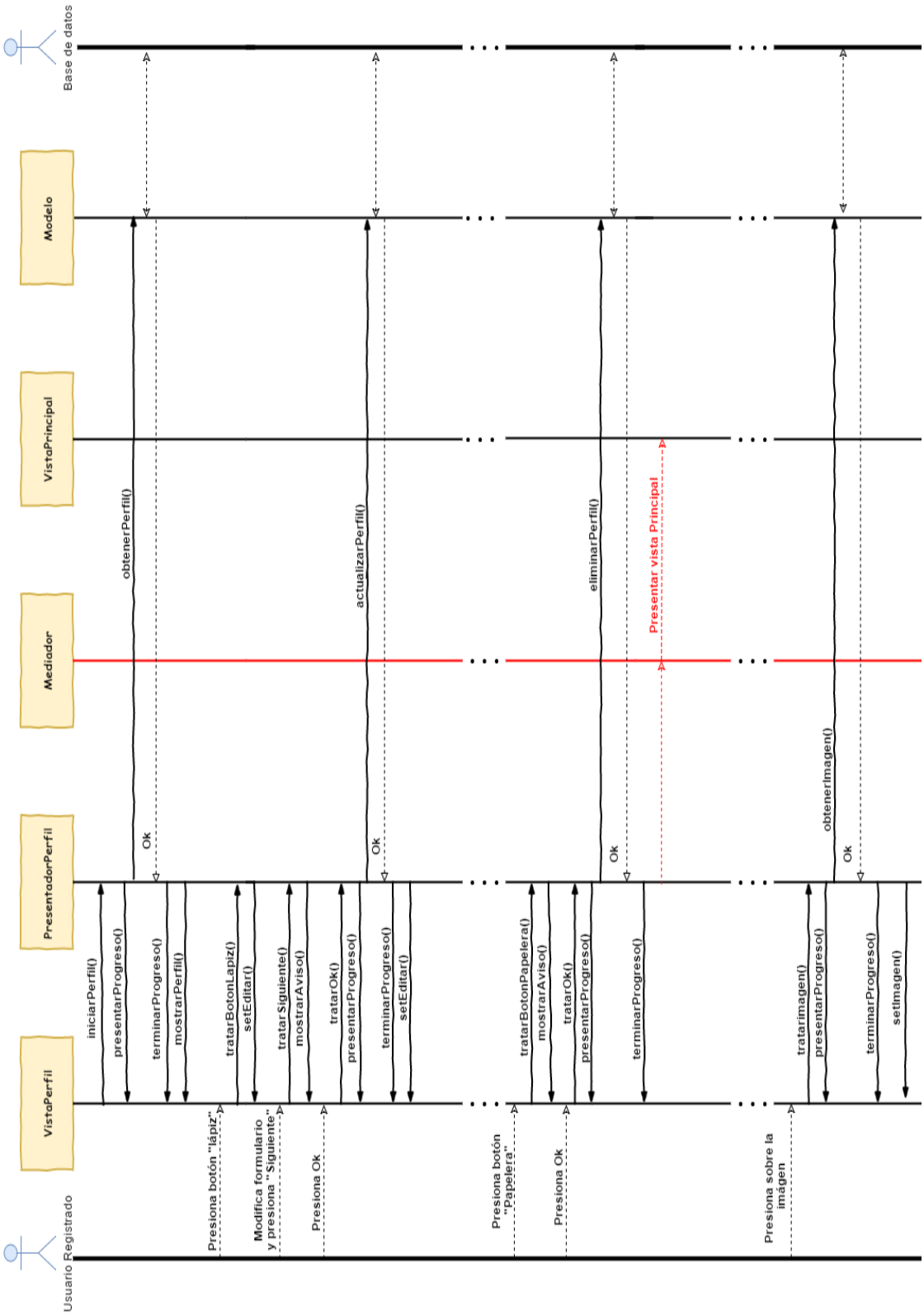


Figura 3.7: Diagrama de secuencia *Perfil*

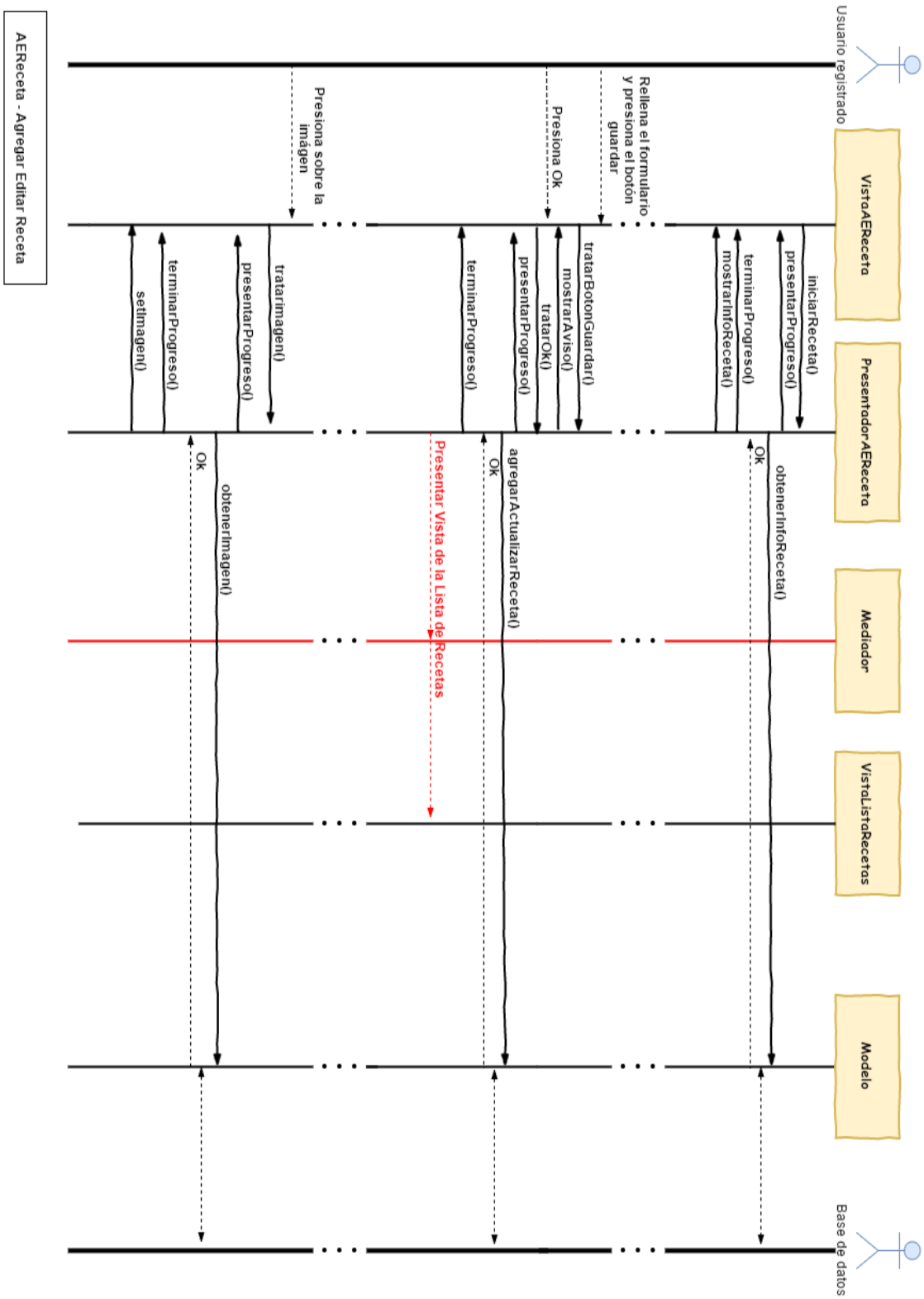


Figura 3.8: Diagrama de secuencia *Agregar Editar Receta 1*

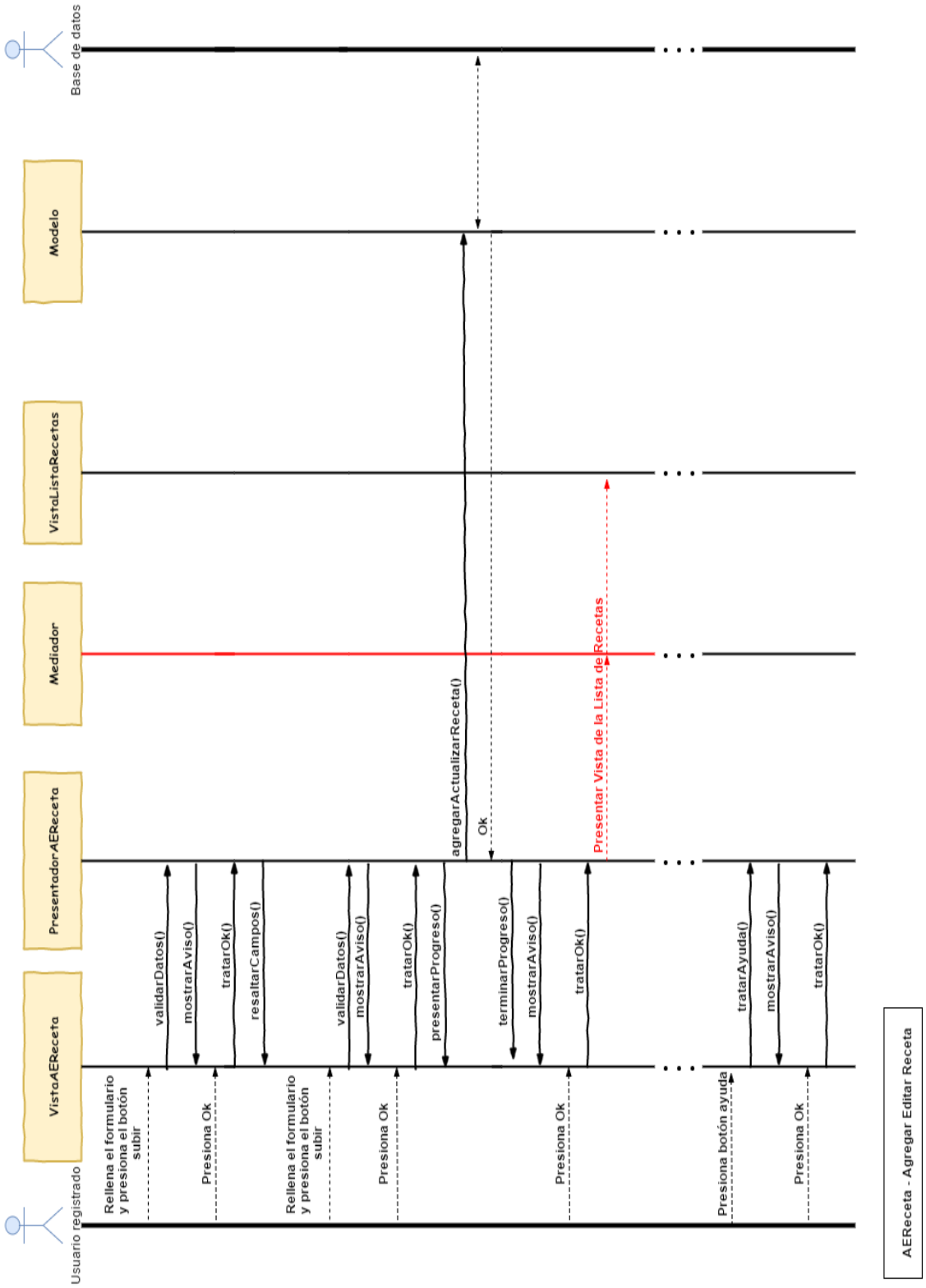


Figura 3.9: Diagrama de secuencia *Agregar Editar Receta 2*

3.2.8 Diagrama de secuencia para los Favoritos

En la figura 3.10 se puede observar el flujo de mensajes intercambiados entre el modelo, la vista y el presentador. Es una vista a la que sólo tienen acceso los usuarios registrados. En esta vista, el usuario puede visualizar las recetas que anteriormente ha marcado como favoritas. Podrá acceder a la información detallada de cualquiera de sus recetas favoritas. A su vez, puede eliminar recetas de su lista de favoritos.

3.2.9 Diagrama de secuencia para Mis Recetas

En la figura 3.11 se puede observar el flujo de mensajes intercambiados entre el modelo, la vista y el presentador. Esta vista solo es accesible para los usuarios retrasados. En esta vista el usuario puede observar sus recetas añadidas o las recetas que ha guardado para continuar con su edición. El usuario podrá realizar una búsqueda entre sus recetas subidas y guardadas.

3.2.10 Diagrama de secuencia del Menú

En la figura 3.12 se puede observar el flujo de mensajes intercambiados entre el modelo, la vista y el presentador. Los ítem del menú son diferentes dependiendo de si es un usuario no registrado o un usuario registrado. Todas las opciones tienen el mismo flujo que da comienzo en la vista que se seleccione el menú. Dicha vista, delega en su presentador correspondiente que se encargará de presentar la vista de la opción seleccionada.

Los usuarios no registrados y registrados comparten las opciones “Login”, “Recetas y los sub-ítem” y “Salir”.

El menú del usuario registrado, a parte de las opciones anteriores, contiene las opciones *Mis Recetas*, *Mi Perfil* y *Mis Favoritos*.

Asimismo, en la figura 3.13, se muestra el intercambio de mensajes entre la vista el presentador y el modelo, cuando se inicia la aplicación. La información suministrada en este intercambio, se utiliza, entre otros, para la construcción del menú que visualiza el usuario.

3.3 Diseño de las clases e interfaces del modelo

Partiendo de los diagramas de secuencia generados en el apartado 3.2, las interfaces del modelo están compuestas por una serie de métodos que se pasarán a describir en los subapartados siguientes.

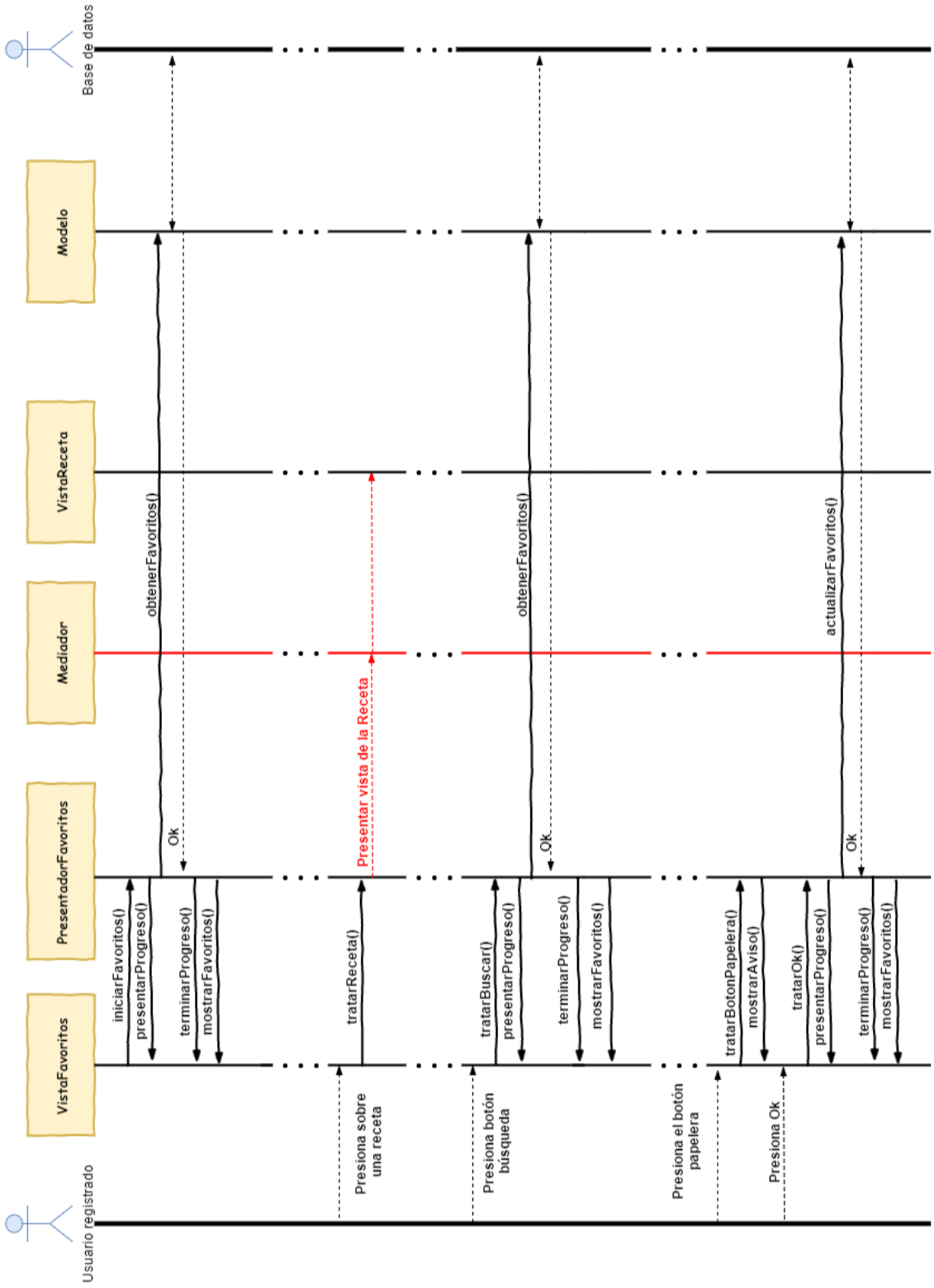


Figura 3.10: Diagrama de secuencia *Favoritos*

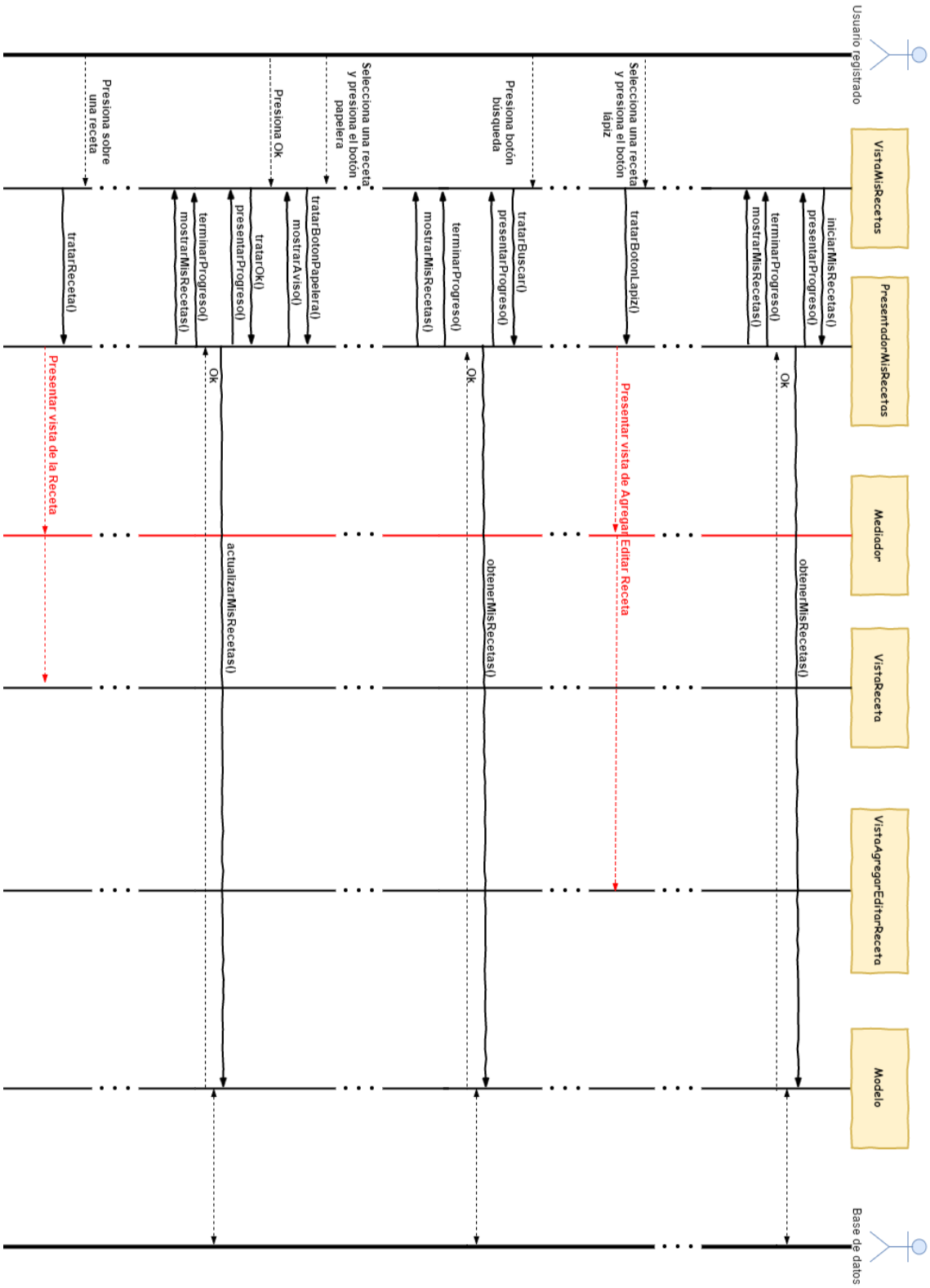


Figura 3.11: Diagrama de secuencia para *Mis Recetas*

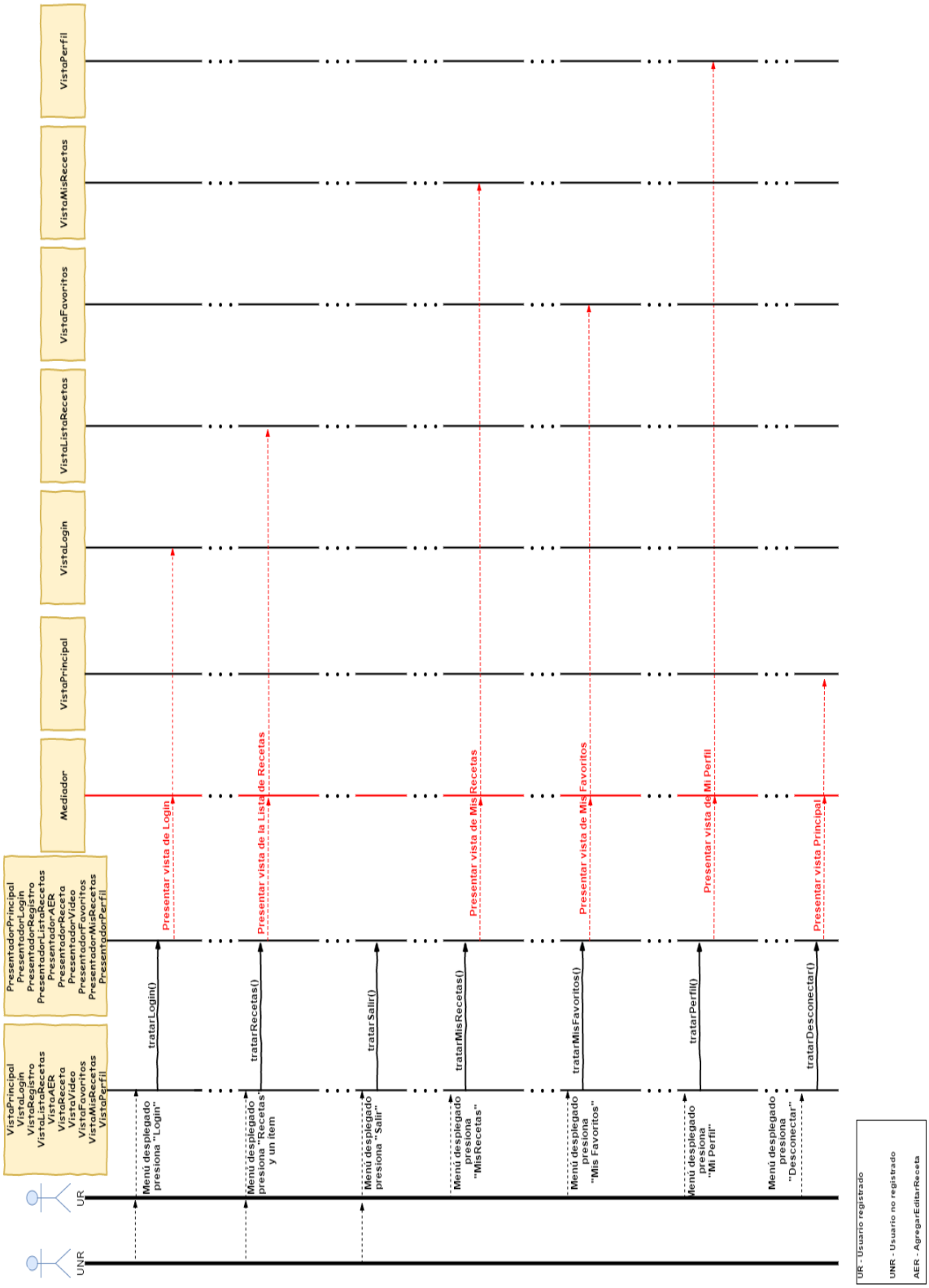


Figura 3.12: Diagrama de secuencia del *Menú*

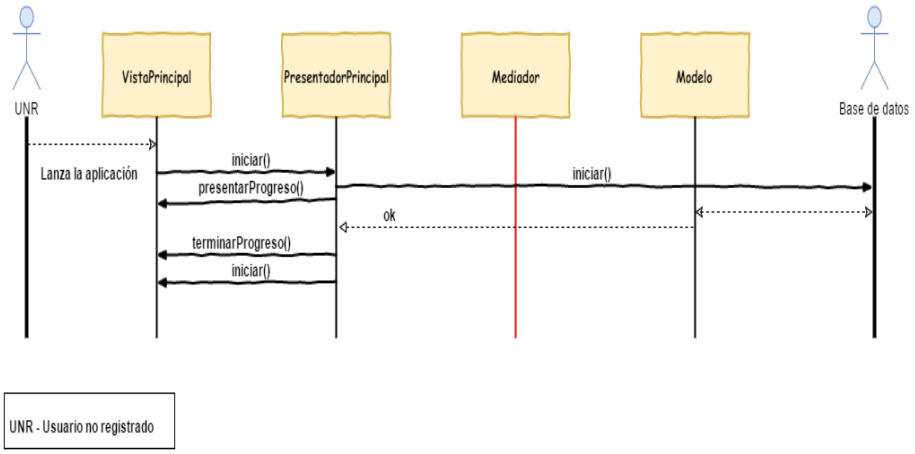


Figura 3.13: Diagrama de secuencia del inicio de la aplicación

3.3.1 Descripción del modelo de datos

Si bien los diagramas de secuencia dan una idea inicial de cómo debe ser el modelo de datos (en función de los mensajes enviados entre objetos), es necesario completar este modelo haciendo un estudio exhaustivo de la información utilizada en la aplicación (de cómo se estructurará ésta, de cómo se debe almacenar, de cómo se debe acceder, entre otras cosas). Debido a esto, en los siguientes apartados se analiza, en detalle, la base de datos y su diseño.

3.3.1.1. Diseño de la base de datos

Para la aplicación *AppRecetas* toda la información referente a la aplicación se almacenará en una base de datos en la nube. Entre los distintos sistemas que existen, se ha elegido *Firebase* [21], que es la nueva y mejorada plataforma de desarrollo móvil en la nube de Google, disponible para: Android, iOS y Web, entre otros. Esta plataforma ofrece servicios de autenticación, base de datos en tiempo Real y compartida, *Analytics*, *Hosting*, mensajeria y varios servicios más, que se muestran en su sitio web.

La *Firebase Realtime Database* es una base de datos alojada en la nube, en la que los datos se almacenan en formato JSON y se sincronizan en tiempo real con cada cliente conectado. Esta base de datos permite crear aplicaciones ricas y colaborativas, posibilitando un acceso seguro a la información, de forma directa, desde el código del cliente. Los datos persisten de manera local, e incluso cuando no hay conexión, los eventos en tiempo real se siguen activando, brindándole al usuario final una experiencia receptiva. A diferencia de las bases de datos SQL, no existen tablas ni registros. Cuando se agregan datos al árbol JSON, éstos se convierten en un nodo en la estructura JSON existente.

Teniendo en cuenta lo explicado anteriormente, a continuación se describirán los distintos nodos que tendrá la base de datos de la aplicación. Por motivos de claridad, a cada nodo se le denomina, *tabla*.

- **Tabla usuarios**. Encargada de contener y gestionar toda la información relacionada con el usuario. Se realizarán consultas de *login*, *registro* y actualización de los datos que el usuario edite. Sus atributos serán:

  - **idUsuario: String**. Identificador único de usuario.
  - **correo: String**. Dirección de correo del usuario, utilizada como *login* del usuario.
  - **password: String**. Contraseña de acceso.
  - **nombre: String**. Nombre o alias del usuario.
  - **imagen: String**. Ruta de la imagen del usuario.

  Es importante recalcar, que uno de los servicios que proporciona *Firebase*, es el sistema de autenticación, *Firebase Authentication* [22], que proporciona servicios de *backend*, SDK fáciles de usar y bibliotecas de IU ya hechas para autenticar usuarios en la aplicación a desarrollar. Este servicio admite autenticación con contraseñas y con proveedores de identidades federadas populares, como "Google", "Facebook" y "Twitter". Asimismo, es un servicio que se integra estrechamente con otros servicios de *Firebase* y aprovecha estándares industriales como *OAuth 2.0* y *OpenID Connect*.

  En el servicio de autenticación, un usuario de *Firebase* tiene un conjunto fijo de propiedades básicas que son: **uid** (identificador único de usuario), **correo** (utilizado como *login*), **password** (contraseña del usuario), **nombre** (del usuario) y **URI** (con la dirección de la foto del usuario). Todas estas propiedades se almacenan en el objeto *usuario* de *Firebase*, de forma independiente a los nodos que pueda crear el usuario en su proyecto. El desarrollador, en cualquier momento, puede actualizar estas propiedades, pero no puede agregar otras. Para agregar nueva información (propiedades) el desarrollador define nuevos nodos en la *Firebase Realtime Database*.

  Analizando la *tabla usuarios* propuesta y el *objeto usuario de Firebase* se puede apreciar que coinciden en sus campos, por tanto, esta tabla no será implementada como un nuevo nodo de *Firebase*.

- **Tabla categorias**. Encargada de contener y gestionar toda la información relacionada con las categorías y sus recetas. Se realizarán consultas destinadas a obtener dichas categorías. Sus atributos serán:

  - **idCategoria: String**. Identificador único de categoría.
  - **nombreCategoria: String** Nombre de la categoría.
  - **listaRecetas: Object**. Objeto que contendrá la lista de recetas de la categoría, en el siguiente formato:

    - **idReceta: String**. Identificador de la receta asociada a la categoría.
    - **nombreReceta: String**. Nombre de la receta asociada a la categoría.
    - **origen: String**. Denominación de origen de la receta.
    - **imagenReceta: String**. Ruta de la imagen principal, asociada a la receta.

- **Tabla recetas**. Encargada de almacenar toda la información referente a una determinada receta (en la base de datos interna y en la base de datos en la nube). Se realizarán consultas destinadas a obtener información sobre los productos, preparación, imágenes y vídeos relacionados. Sus atributos serán:

    - **idReceta: String**. Identificador de la receta.
    - **idCategoria: String** Identificador de la categoría a la que pertenece.
    - **idUsuario: String**. Identificador del usuario que ha incluido la receta en la base de datos.
    - **nombreReceta: String**. Nombre de la receta.
    - **origen: String**. Denominación de origen de la receta.
    - **ingredientes: Object**. Lista de ingredientes de de la receta.
    - **preparacion: String**. Preparación de la receta.
    - **imagenes: Object**. Lista de rutas de las imágenes de la receta.
    - **url: String**. URL del vídeo de explicación de la receta.

- **Tabla favoritos**. Encargada de almacenar el identificador de las recetas favoritas de un usuario. Sus atributos serán:

    - **idUsuario: String**. Identificador del usuario que ha incluido la receta en la base de datos.
    - **favoritos: Object**. Lista con los identificadores de las recetas favoritas del usuario.

3.3.1.2. Diseño de las nuevas clases del modelo

Una vez declaradas las tablas, se presentan las clases relacionadas con las mismas y los métodos que incluyen cada uno de ellas. Así, la tabla x tendrá relacionadas dos clases: una clase simple que representa a un objeto almacenado en la tabla (se llamará clase X o similar) y una clase adaptador, que manejará los campos de esa tabla (se llamada *BDAdaptadorX* o similar). Las nuevas clases son:

- **Clase Categoria**. Relacionada con la entidad *categorias*. Los atributos de esta clase son los campos de la tabla e incluye el constructor y los métodos *getters* y *setters* de sus atributos. Estos son:

- **Categoria(idCategoria: String, nombreCategoria: String, listaRecetas: Object)**. Constructor de la clase.
- **getIdCategoria(): String**. Método que devuelve el identificador de la categoría.
- **getnombreCategoria(): String**. Método que devuelve el nombre de la categoría.
- **getListaRecetas(): Object**. Método que devuelve la lista de recetas de la categoría (la información devuelta tiene un formato explicado posteriormente).
- **setIdCategoria(idCategoria: String): void**. Método que modifica el identificador de la categoría.
- **setnombreCategoria(nombreCategoria: String): void**. Método que modifica el nombre de la categoría.
- **setListaRecetas(listaRecetas: Object): void**. Método que modifica la lista de recetas de la categoría.

El atributo *listaRecetas* corresponde con una lista de recetas de la categoría, en la que se almacena la información en un determinado formato. Debido a esto, se necesita una clase específica para representar la información almacenada en esta lista. Esta nueva clase, con sus atributos y métodos, es:

- **Clase recetaBD**. Relacionada con el atributo *listaRecetas* de la tabla *categorias*. El constructor y los métodos *getter* y *setter* de la clase son los siguientes:
  - **RecetaBD(idReceta: String, nombreReceta: String, origen: String, imagenReceta: String)**. Constructor de la clase.
  - **getIdReceta(): String**. Método que devuelve el identificador de la receta.
  - **getNombreReceta(): String**. Método que devuelve el nombre de la receta.
  - **getOrigen(): String**. Método que devuelve el origen de la receta.
  - **getImagenReceta(): String**. Método que devuelve la ruta de la imagen de la receta.
  - **setIdReceta(idReceta: String): void**. Método que modifica el identificador de la receta.
  - **setNombreReceta(nombreReceta: String): void**. Método que modifica el nombre de la receta.
  - **setOrigen(origen: String): void**. Método que modifica el origen de la receta.
  - **setImagenReceta(imagenReceta: String): void**. Método que modifica la ruta de la imagen de la receta.

- **Clase Receta**. Relacionada con la entidad *recetas*. Los atributos de esta clase son los campos de la tabla e incluye el constructor y los métodos *getters* y *setters* de sus atributos.

  - **Receta(idReceta: String, idCategoria: String, idUsuario: String, nombreReceta: String, origen: String, ingredientes: Object, preparacion: String, imagenes: Object, url: String)**. Constructor de la clase.
  - **getIdReceta(): String**. Método que devuelve el identificador de la receta.
  - **getIdCategoria(): String**. Método que devuelve el identificador de la categoría a la que pertenece la receta.
  - **getIdUsuario(): String**. Método que devuelve el identificador del usuario que ha subido la receta a la nube.
  - **getNombreReceta(): String**. Método que devuelve el nombre de la receta.
  - **getOrigen(): String**. Método que devuelve el origen de la receta.
  - **getIngredientes(): Object**. Método que devuelve los ingredientes de la receta.
  - **getPreparacion(): String**. Método que devuelve la preparación de la receta.
  - **getImagenes(): Object**. Método que devuelve las rutas de las imágenes de la receta.
  - **getUrl(): String**. Método que devuelve la url del video de la receta.
  - **setIdReceta(idReceta: String): void**. Método que modifica el identificador de la receta.
  - **setIdCategoria(idCategoria: String): void**. Método que modifica el identificador de la categoría a la que pertenece la receta.
  - **setIdUsuario(idUsuario: String): void**. Método que modifica el identificador del usuario que ha subido la receta a la nube.
  - **setNombreReceta(nombreReceta: String): void**. Método que modifica el nombre de la receta.
  - **setOrigen(origen: String): void**. Método que modifica el origen de la receta.
  - **setIngredientes(ingredientes: Object): void**. Método que modifica los ingredientes de la receta.
  - **setPreparacion(preparacion: String): void**. Método que modifica la preparación de la receta.
  - **setImagenes(imagenes: Object): void**. Método que modifica las rutas de las imágenes de la receta.

  - **setUrl(url: String): void**. Método que modifica la url del vídeo de la receta.

- **Clase Favorito**. Relacionada con la entidad *favoritos*. Los atributos de esta clase son los campos de la tabla e incluye el constructor y los métodos *getters* y *setters* de sus atributos.

  - **Favorito(idUsuario: String, favoritos: Object)**. Constructor de la clase.
  - **getIdUsuario(): String**. Método que devuelve el identificador del usuario que ha subido la receta a la nube.
  - **getFavoritos(): Object**. Método que devuelve una lista con los identificadores de las recetas favoritas del usuario.
  - **setIdUsuario(idUsuario: String): void**. Método que modifica el identificador del usuario que ha subido la receta a la nube.
  - **setFavoritos(favoritos: Object): void**. Método que modifica la lista de recetas favoritas del usuario.

  Clase BDAdaptadorCategoria. Clase puente entre el modelo y el actor base de datos, concretamente con la tabla *categorias*, e incluye diferentes métodos para acceder a los datos de esta tabla. Los métodos son los siguientes:

  - **obtenerCategorias(): void**. Método que busca las categorías y notifica un objeto con las categorías encontradas.
  - **obtenerListaRecetas(idCategoria: String): void**. Método que busca las recetas que pertenecen a la categoría, cuyo identificador coincide con el parámetro *idCategoria*.

  Clase BDAdaptadorReceta. Clase puente entre el modelo y el actor base de datos, concretamente con la tabla *recetas*, e incluye los siguientes métodos:

  - **obtenerInfoReceta(info: Object): void**. Método que busca, en la base de datos, la información de una receta y notifica el resultado. A través del parámetro se indica el identificador de la receta a buscar y si ésta está en la base de datos interna o en la nube.
  - **obtenerRecetasDeUsuario(info: Object): void**. Método que devuelve la lista de recetas de un usuario y notifica el resultado. A través del parámetro se indica el identificador del usuario y si existen recetas de éste en la base de datos interna.
  - **actualizarReceta(info: Object): void**. Método que actualiza una receta y notifica el resultado. A través del parámetro se indica la información de la receta a actualizar y si ésta está en la base de datos interna o en la nube.

- **eliminarReceta(info: Object): void**. Método que elimina una receta y notifica el resultado. A través del parámetro se indica la receta a eliminar y si ésta está en la base de datos interna o en la nube.
- **guardarReceta(info: Object): void**. Método que almacena, en la base de datos, la información de la receta que se recibe por parámetros y notifica el resultado. A través del parámetro se indica si la información se almacena en la base de datos interna o en la nube.

Clase BDAdaptadorFavorito. Clase puente entre el modelo y el actor base de datos, concretamente con la tabla *favoritos*, e incluye los siguientes métodos:

- **obtenerFavoritos(idUsuario: String): void**. Método que notifica la lista de recetas favoritas del usuario cuyo identificador se recibe por parámetros.
- **agregarFavorito(idReceta: String, idUsuario: String): void**. Método que agrega una receta a la lista de favoritos de un usuario y notifica el resultado.
- **eliminarFavorito(idReceta: String, idUsuario: String): void**. Método que elimina una receta de la lista de favoritos de un usuario y notifica el resultado.

3.3.2 Clase Modelo

Clase que implementa la interfaz *IModelo*, que define todos los métodos necesarios, para que los distintos presentadores puedan interactuar con los datos de la base de datos. Los métodos definidos en la interfaz han sido descubiertos a partir de los diagramas de secuencia de la sección 3.2. La clase *Modelo* implementará estos métodos para darles la funcionalidad requerida y todos ellos realizarán algún proceso sobre la base de datos. Una vez terminado el proceso, el modelo notificará la finalización del mismo, de forma que aquellos observadores (presentadores) que estén esperando por los resultados, puedan obtenerlos. A continuación se presenta una descripción de estos métodos.

- **iniciar(): void**. Método que inicia la aplicación, obteniendo la información necesaria, entre otros, para completar el menú. El método notifica al finalizar la operación.
- **validarDatos(informacion: Object): void**. Método que valida los datos del formulario para *loguear* a un usuario. Estos datos y demás información importante se encuentran almacenados en el parámetro *informacion*. El método notifica al finalizar la operación.
- **recuperarPassword(email: String): void**. Método que notifica la contraseña del email introducido por parámetro. En caso de que el email no sea correcto, se indica en la notificación.

- **registrarUsuario(informacion: Object): void**. Método que agrega los datos del registro de un usuario. Estos datos y demás información importante se encuentran almacenados en el parámetro *informacion*. El método notifica al finalizar la operación.
- **obtenerListaRecetas(idCategoria: String): void**. Método que busca la lista de recetas pertenecientes a la categoría indicada por parámetros y notifica el resultado.
- **actualizaListaFavoritos(informacion: Object): void**. Método que actualiza la lista de favoritos del usuario con una nueva receta. Los datos se encuentran almacenados en el parámetro *informacion*. El método notifica al finalizar la operación.
- **obtenerfavoritos(idUsuario: String): void**. Método que busca las recetas favoritas del usuario, con identificador recibido por parámetro, y notifica el resultado.
- **actualizarfavoritos(idReceta: String): void**. Método que elimina la receta, con identificador recibido por parámetro, de la lista de favoritos del usuario y notifica el resultado.
- **obtenerVideo(idReceta: String): void**. Método que devuelve la url del vídeo de la receta y notifica el resultado.
- **obtenerInfoReceta(idReceta: String): void**. Método que busca la información de la recetas, con identificador recibido por parámetros, y notifica el resultado.
- **agregarActualizarReceta(informacion: Object): void**. Método que añade o actualiza la información de una receta. Los datos de la receta se encuentran almacenados en el parámetro *informacion*. El método notifica al finalizar la operación.
- **obtenerImagen(idReceta: String): void**. Método que devuelve la ruta de la imagen de la receta, cuyo identificador se recibe por parámetros. El método notifica al finalizar la operación.
- **obtenerPerfil(idUsuario: String): void**. Método que devuelve la información del usuario, cuyo identificador se indica por parámetros. El método notifica al finalizar la operación.
- **actualizarPerfil(idUsuario: String): void**. Método que actualiza la información del usuario, cuyo identificador se indica por parámetros. El método notifica al finalizar la operación.
- **eliminarPerfil(idUsuario: String): void**. Método que elimina la información del usuario, cuyo identificador se indica por parámetros. El método notifica al finalizar la operación.

- **obtenerMisRecetas(idUsuario: String): void**. Método que busca la receta subidas o guardada del usuario, cuyo identificador se recibe por parámetros. El método notifica al finalizar la operación.
- **actualizarMisRecetas(idReceta: String): void**. Método que elimina la receta subidas o guardada del usuario, cuyo identificador se recibe por parámetros. El método notifica al finalizar la operación.

3.4 Diseño de las clases e interfaces de la vista

Partiendo de los diagramas de secuencia generados en la sección 3.2, las interfaces de la vista están compuestas por una serie de métodos que se pasarán a describir a continuación.

3.4.1 Clase VistaPrincipal

Vista entrada de la aplicación. A través de esta vista, el usuario puede acceder al resto de pantallas mediante el menú. Implementa la interfaz *IVistaPrincipal* compuesta del siguiente método:

- **presentarProgreso(): void**. Muestra una barra de progreso para indicar al usuario que se está realizando una acción.
- **terminarProgreso(): void**. Elimina la barra de progreso para indicar al usuario que la acción ha terminado.
- **iniciar(informacion: Object): void**. Solicita, entre otros, la información a mostrar en el menú de la aplicación. A través del parámetro se recibe la información necesaria para el funcionamiento del método.

3.4.2 Clase VistaLogin

Vista que permite al usuario *loguearse* en la aplicación. Implementa la interfaz *IVistaLogin* compuesta de los siguientes métodos:

- **presentarProgreso(): void**. Muestra una barra de progreso para indicar al usuario que se está realizando una acción.
- **terminarProgreso(): void**. Elimina la barra de progreso para indicar al usuario que la acción ha terminado.
- **mostrarAviso(informacion: Object): void**. Muestra al usuario un aviso que precisa de su interacción. A través del parámetro se recibe la información necesaria para el funcionamiento del método.

3.4.3 Clase VistaRegistro

Vista que permite al usuario registrarse en la aplicación. Implementa la interfaz *IVistaRegistro* compuesta de los siguientes métodos:

- **presentarProgreso(): void**. Muestra una barra de progreso para indicar al usuario que se está realizando una acción.
- **terminarProgreso(): void**. Elimina la barra de progreso para indicar al usuario que la acción ha terminado.
- **mostrarAviso(informacion: Object): void**. Muestra al usuario un aviso que precisa de su interacción. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **limpiarFormulario(): void**. Limpia los campos del formulario, dejándolos en blanco.

3.4.4 Clase VistaListaRecetas

Vista que permite al usuario ver las recetas pertenecientes a una categoría. Implementa la interfaz *IVistaListaRecetas* compuesta de los siguientes métodos:

- **presentarProgreso(): void**. Muestra una barra de progreso para indicar al usuario que se está realizando una acción.
- **terminarProgreso(): void**. Elimina la barra de progreso para indicar al usuario que la acción ha terminado.
- **mostrarListaRecetas(informacion: Object): void**. Muestra la lista con todas las recetas pertenecientes a la categoría seleccionada por el usuario. La información que se necesita para recuperar la lista se recibe a través del parámetro.

3.4.5 Clase VistaReceta

Se trata de la vista que permite al usuario ver la información detallada de una receta seleccionada. Implementa la interfaz *IVistaReceta* compuesta por los siguientes métodos:

- **presentarProgreso(): void**. Muestra una barra de progreso para indicar al usuario que se está realizando una acción.

- **terminarProgreso(): void**. Elimina la barra de progreso para indicar al usuario que la acción ha terminado.
- **mostrarInfoReceta(informacion: Object): void**. Muestra toda la información detallada de la receta seleccionada previamente por el usuario. La información se recibe a través del parámetro.
- **pintarEstrella(estado: boolean): void**. Rellena o vacía la estrella indicando si la receta está entre las recetas favoritas del usuario o no. El parámetro *estado* indica si la receta es o no favorita.

3.4.6 Clase VistaVideo

Vista que muestra al usuario un vídeo, en caso de existir, sobre la elaboración de una receta. Implementa la interfaz *IVistaVideo* compuesta por los siguientes métodos:

- **presentarProgreso(): void**. Muestra una barra de progreso para indicar al usuario que se está realizando una acción.
- **terminarProgreso(): void**. Elimina la barra de progreso para indicar al usuario que la acción ha terminado.
- **mostrarAviso(informacion: Object): void**. Muestra al usuario un aviso que precisa de su interacción. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **mostrarVideo(informacion: Object): void**. Muestra el vídeo de la receta previamente seleccionada por el usuario. A través del parámetro se recibe la información necesaria para el funcionamiento del método.

3.4.7 Clase VistaAgregarEditarReceta

Vista que permite al usuario añadir una nueva receta o editar una receta que se ha añadido con anterioridad. Implementa la interfaz *IAgregarEditarReceta* con los siguientes métodos:

- **presentarProgreso(): void**. Muestra una barra de progreso para indicar al usuario que se está realizando una acción.
- **terminarProgreso(): void**. Elimina la barra de progreso para indicar al usuario que la acción ha terminado.

- **mostrarAviso(informacion: Object): void**. Muestra al usuario un aviso que precisa de su interacción. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **mostrarInfoReceta(informacion: Object): void**. Muestra toda la información detallada de la receta seleccionada previamente por el usuario. La información se recibe a través del parámetro.
- **resaltarCampos(informacion: Object): void**. Resalta los campos necesarios para continuar con el progreso (en el caso de que el formulario de adición o edición, no se haya completado correctamente). A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **setImagen(imagen: Object): void**. Permite subir las fotos sobre la receta. A través del parámetro se reciben las imágenes seleccionadas.

3.4.8 Clase VistaPerfil

Vista que permite al usuario visualizar su perfil y realizar algún cambio en la información. Implementa la interfaz *IVistaPerfil* con los siguientes métodos:

- **presentarProgreso(): void**. Muestra una barra de progreso para indicar al usuario que se está realizando una acción.
- **terminarProgreso(): void**. Elimina la barra de progreso para indicar al usuario que la acción ha terminado.
- **mostrarAviso(informacion: Object): void**. Muestra al usuario un aviso que precisa de su interacción. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **mostrarPerfil(infoCuenta: Object): void**. Muestra la información sobre la cuenta del usuario. La información a mostrar se recibe por parámetro.
- **setEditar(): void**. Habilita los campos del formulario para que el usuario pueda editarlos.

3.4.9 Clase VistaFavoritos

Vista que permite al usuario visualizar todas las recetas que ha marcado como favoritas. Implementa la interfaz *IVistaFavoritos* con los siguientes métodos:

- **presentarProgreso(): void**. Muestra una barra de progreso para indicar al usuario que se está realizando una acción.

- **terminarProgreso(): void**. Elimina la barra de progreso para indicar al usuario que la acción ha terminado.

- **mostrarAviso(informacion: Object): void**. Muestra al usuario un aviso que precisa de su interacción. A través del parámetro se recibe la información necesaria para el funcionamiento del método.

- **mostarFavoritos(favoritos: Object): void**. Muestra la lista con las recetas que el usuario ha marcado como favoritas. A través del parámetro se recibe la lista de recetas favoritas del usuario.

3.4.10 Clase VistaMisRecetas

Vista que permite al usuario visualizar todas las recetas que ha creado o las recetas que está creando. Implementa la interfaz *IVistaMisRecetas* con los siguientes métodos:

- **presentarProgreso(): void**. Muestra una barra de progreso para indicar al usuario que se está realizando una acción.

- **terminarProgreso(): void**. Elimina la barra de progreso para indicar al usuario que la acción ha terminado.

- **mostrarAviso(informacion: Object): void**. Muestra al usuario un aviso que precisa de su interacción. A través del parámetro se recibe la información necesaria para el funcionamiento del método.

- **mostarMisRecetas(misRecetas: Object): void**. Muestra la lista con las recetas que el usuario ha creado o está creando. A través del parámetro se recibe la lista de recetas del usuario.

3.5 Diseño de las clases e interfaces del presentador

Partiendo de los diagramas de secuencia generados en la sección 3.2, las interfaces del presentador están compuestas por una serie de métodos que se pasarán a describir a continuación.

3.5.1 Clase PresentadorPrincipal

Presentador de la vista *VistaPrincipal*, encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorPrincipal* con los siguientes métodos:

- **tratarLogin(): void**. Método que mostrará la vista *VistaLogin*.
- **tratarRecetas(informacion: Object): void**. Método que mostrará la vista *VistaListaRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarMisRecetas(informacion: Object): void**. Método que mostrará la vista *VistaMisRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarFavoritos(informacion: Object): void**. Método que mostrará la vista *VistaFavoritos*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarPerfil(informacion: Object): void**. Método que mostrará la vista *VistaPerfil*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarDesconectar(informacion: Object): void**. Método que desconecta al usuario de la aplicación. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarSalir(): void**. Método que provoca la salida de la aplicación.
- **iniciar(): void**. Método que solicita al modelo la información necesaria para iniciar la aplicación.

3.5.2 Clase PresentadorLogin

Presentador de la vista *VistaLogin*, encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorLogin* con los siguientes métodos:

- **tratarLogin(): void**. Método que mostrará la vista *VistaLogin*.
- **tratarRecetas(informacion: Object): void**. Método que mostrará la vista *VistaListaRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.

- **tratarMisRecetas(informacion: Object): void**. Método que mostrará la vista *VistaMisRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarFavoritos(informacion: Object): void**. Método que mostrará la vista *VistaFavoritos*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarPerfil(informacion: Object): void**. Método que mostrará la vista *VistaPerfil*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarDesconectar(informacion: Object): void**. Método que desconecta al usuario de la aplicación. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarSalir(): void**. Método que provoca la salida de la aplicación.
- **validarDatos(informacion: Object): void**. Método que pide al modelo que compruebe los datos introducidos por parámetros.
- **tratarOk(informacion: Object): void**. Método que actúa sobre una interacción del usuario (respondiendo a un *Ok*). A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarRegistro(): void**. Método que mostrará la vista *VistaRegistro*.
- **tratarRecuperarPassword(informacion: Object): void**. Método que solicita al modelo el *password* del usuario. A través del parámetro se recibe la información necesaria para el funcionamiento del método.

3.5.3 Clase PresentadorRegistro

Presentador de la vista *VistaRegistro*, encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorRegistro* con los siguientes métodos:

- **tratarLogin(): void**. Método que mostrará la vista *VistaLogin*.
- **tratarRecetas(informacion: Object): void**. Método que mostrará la vista *VistaListaRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarMisRecetas(informacion: Object): void**. Método que mostrará la vista *VistaMisRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.

- **tratarFavoritos(informacion: Object): void**. Método que mostrará la vista *VistaFavoritos*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarPerfil(informacion: Object): void**. Método que mostrará la vista *VistaPerfil*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarDesconectar(informacion: Object): void**. Método que desconecta al usuario de la aplicación. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarSalir(): void**. Método que provoca la salida de la aplicación.
- **validarDatos(informacion: Object): void**. Método que pide al modelo que compruebe los datos introducidos por parámetros.
- **tratarOk(informacion: Object): void**. Método que actúa sobre una interacción del usuario (respondiendo a un *Ok*). A través del parámetro se recibe la información necesaria para el funcionamiento del método.

3.5.4 Clase PresentadorListaRecetas

Presentador de la vista *VistaListaRecetas*, encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorListaRecetas* con los siguientes métodos:

- **tratarLogin(): void**. Método que mostrará la vista *VistaLogin*.
- **tratarRecetas(informacion: Object): void**. Método que mostrará la vista *VistaListaRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarMisRecetas(informacion: Object): void**. Método que mostrará la vista *VistaMisRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarFavoritos(informacion: Object): void**. Método que mostrará la vista *VistaFavoritos*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarPerfil(informacion: Object): void**. Método que mostrará la vista *VistaPerfil*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarDesconectar(informacion: Object): void**. Método que desconecta al usuario de la aplicación. A través del parámetro se recibe la información necesaria para el funcionamiento del método.

- **tratarSalir(): void**. Método que provoca la salida de la aplicación.
- **obtenerListaRecetas(idCategoria: String): void**. Método que pide al modelo que recupere la lista de recetas de una categoría indicada por parámetro.
- **tratarRegistro(): void**. Método que mostrará la vista *VistaRegistro*.
- **tratarBotonAgregar(informacion: Object): void**. Método que mostrará la vista *VistaAgregarEditarReceta*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarReceta(informacion: Object): void**. Método que mostrará la vista *VistaReceta*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarBuscar(filtro: String): void**. Método que pide al modelo el filtrado de las recetas. El filtro aplicado se recibe a través del parámetro *filtro*.

3.5.5 Clase PresentadorReceta

Presentador de la vista *VistaReceta*, encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorReceta* con los siguientes métodos:

- **tratarLogin(): void**. Método que mostrará la vista *VistaLogin*.
- **tratarRecetas(informacion: Object): void**. Método que mostrará la vista *VistaListaRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarMisRecetas(informacion: Object): void**. Método que mostrará la vista *VistaMisRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarFavoritos(informacion: Object): void**. Método que mostrará la vista *VistaFavoritos*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarPerfil(informacion: Object): void**. Método que mostrará la vista *VistaPerfil*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarDesconectar(informacion: Object): void**. Método que desconecta al usuario de la aplicación. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarSalir(): void**. Método que provoca la salida de la aplicación.

- **obtenerReceta(idReceta: String): void**. Método que pide al modelo que recupere la información de la receta, cuyo identificador se recibe por parámetro.
- **tratarRegistro(): void**. Método que mostrará la vista *VistaRegistro*.
- **tratarBotonEstrella(informacion: Object): void**. Método que actúa sobre una interacción del usuario (selección del botón estrella de una receta). A través de parámetros se recibe la información necesaria para la funcionalidad del método.
- **tratarBotonLapiz(informacion: Object): void**. Método que mostrará la vista *VistaAgregarEditarReceta*. A través de parámetros se recibe la información necesaria para la funcionalidad del método.
- **tratarVideo(informacion: Object): void**. Método que mostrará la vista *VistaVideo*. A través de parámetros se recibe la información necesaria para la funcionalidad del método.

3.5.6 Clase PresentadorVideo

Presentador de la vista *VistaVideo*, encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorVideo* con los siguientes métodos:

- **tratarLogin(): void**. Método que mostrará la vista *VistaLogin*.
- **tratarRecetas(informacion: Object): void**. Método que mostrará la vista *VistaListaRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarMisRecetas(informacion: Object): void**. Método que mostrará la vista *VistaMisRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarFavoritos(informacion: Object): void**. Método que mostrará la vista *VistaFavoritos*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarPerfil(informacion: Object): void**. Método que mostrará la vista *VistaPerfil*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarDesconectar(informacion: Object): void**. Método que desconecta al usuario de la aplicación. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarSalir(): void**. Método que provoca la salida de la aplicación.

- **obtenerVideo(idReceta: String): void**. Método solicita al modelo el vídeo de la receta, cuyo identificador se recibe por parámetro.
- **tratarOk(informacion: Object): void**. Método que actúa sobre una interacción del usuario (respondiendo a un *Ok*). A través del parámetro se recibe la información necesaria para el funcionamiento del método.

3.5.7 Clase PresentadorAgregarEditarReceta

Presentador de la vista *VistaAgregarEditarReceta*, encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorAgregarEditarReceta* con los siguientes métodos:

- **tratarLogin(): void**. Método que mostrará la vista *VistaLogin*.
- **tratarRecetas(informacion: Object): void**. Método que mostrará la vista *VistaListaRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarMisRecetas(informacion: Object): void**. Método que mostrará la vista *VistaMisRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarFavoritos(informacion: Object): void**. Método que mostrará la vista *VistaFavoritos*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarPerfil(informacion: Object): void**. Método que mostrará la vista *VistaPerfil*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarDesconectar(informacion: Object): void**. Método que desconecta al usuario de la aplicación. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarSalir(): void**. Método que provoca la salida de la aplicación.
- **iniciarReceta(idReceta: String): void**. Método que solicita al modelo la información de la receta, cuyo identificador se recibe por parámetro.
- **tratarBotonGuardar(info: Object): void**. Método que actúa sobre una interacción del usuario (respondiendo a la selección del botón *Guardar* de la vista *VistaAgregarEditarReceta*).
- **tratarImagen(): void**. Método que actúa sobre una interacción del usuario (respondiendo a la selección de la imagen en la vista *VistaAgregarEditarReceta*).

- **tratarOk(informacion: Object): void**. Método que actúa sobre una interacción del usuario (respondiendo a un *Ok*). A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **validarDatos(informacion: Object): void**. Método que pide al modelo que compruebe los datos introducidos por parámetros.
- **tratarAyuda(informacion: Object): void**. Método que mostrará un mensaje con un texto de ayuda. A través de parámetros se recibe la información necesaria para la funcionalidad del método.

3.5.8 Clase PresentadorPerfil

Presentador de la vista *VistaPerfil*, encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorPerfil* con los siguientes métodos:

- **tratarLogin(): void**. Método que mostrará la vista *VistaLogin*.
- **tratarRecetas(informacion: Object): void**. Método que mostrará la vista *VistaListaRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarMisRecetas(informacion: Object): void**. Método que mostrará la vista *VistaMisRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarFavoritos(informacion: Object): void**. Método que mostrará la vista *VistaFavoritos*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarPerfil(informacion: Object): void**. Método que mostrará la vista *VistaPerfil*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarDesconectar(informacion: Object): void**. Método que desconecta al usuario de la aplicación. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarSalir(): void**. Método que provoca la salida de la aplicación.
- **iniciarPerfil(informacion: Object): void**. Método que solicita al modelo la información del usuario. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarBotonLapiz(): void**. Método que habilita los campos del formulario presente en la *VistaPerfil* para poder editarlos.

- **tratarSiguiente(): void**. Método que actúa sobre una interacción del usuario (respondiendo a la selección del botón *Siguiente*).
- **tratarBotonPapelera(): void**. Método que actúa sobre una interacción del usuario (respondiendo a la selección del botón *Papelera*).
- **tratarOk(informacion: Object): void**. Método que actúa sobre una interacción del usuario (respondiendo a un *Ok*). A través del parámetro se recibe la información necesaria para el funcionamiento del método.

3.5.9 Clase PresentadorFavoritos

Presentador de la vista *VistaFavoritos*, encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorFavoritos* con los siguientes métodos:

- **tratarLogin(): void**. Método que mostrará la vista *VistaLogin*.
- **tratarRecetas(informacion: Object): void**. Método que mostrará la vista *VistaListaRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarMisRecetas(informacion: Object): void**. Método que mostrará la vista *VistaMisRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarFavoritos(informacion: Object): void**. Método que mostrará la vista *VistaFavoritos*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarPerfil(informacion: Object): void**. Método que mostrará la vista *VistaPerfil*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarDesconectar(informacion: Object): void**. Método que desconecta al usuario de la aplicación. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarSalir(): void**. Método que provoca la salida de la aplicación.
- **iniciarFavoritos(informacion: Object): void**. Método que solicita al modelo la lista de recetas favoritas del usuario. A través de parámetros se recibe la información necesaria para la funcionalidad del método.
- **tratarReceta(informacion: Object): void**. Método que mostrará la vista *VistaReceta*. A través de parámetros se recibe la información necesaria para la funcionalidad del método.

- **tratarBuscar(filtro: String): void**. Método que solicita al modelo el filtrado de las recetas. El filtro aplicado se recibe a través del parámetro *filtro*.
- **tratarBotonPapelera(informacion: Object): void**. Método que actúa sobre una interacción del usuario (respondiendo a la selección del botón *Papelera*).
- **tratarOk(informacion: Object): void**. Método que actúa sobre una interacción del usuario (respondiendo a un *Ok*). A través del parámetro se recibe la información necesaria para el funcionamiento del método.

3.5.10 Clase PresentadorMisRecetas

Presentador de la vista *VistaMisRecetas*, encargado de mostrar dicha vista, actualizarla y presentar la información pertinente en ésta. Esta clase implementa la interfaz *IPresentadorMisRecetas* con los siguientes métodos:

- **tratarLogin(): void**. Método que mostrará la vista *VistaLogin*.
- **tratarRecetas(informacion: Object): void**. Método que mostrará la vista *VistaListaRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarMisRecetas(informacion: Object): void**. Método que mostrará la vista *VistaMisRecetas*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarFavoritos(informacion: Object): void**. Método que mostrará la vista *VistaFavoritos*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarPerfil(informacion: Object): void**. Método que mostrará la vista *VistaPerfil*. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarDesconectar(informacion: Object): void**. Método que desconecta al usuario de la aplicación. A través del parámetro se recibe la información necesaria para el funcionamiento del método.
- **tratarSalir(): void**. Método que provoca la salida de la aplicación.
- **iniciarMisRecetas(): void**. Método que carga la lista de recetas del usuario.
- **tratarBotonLapiz(informacion: Object): void**. Método que mostrará la vista *VistaAgregarEditarReceta*. A través de parámetros se recibe la información necesaria para la funcionalidad del método.

- **tratarReceta(informacion: Object): void**. Método que mostrará la vista *VistaReceta*. A través de parámetros se recibe la información necesaria para la funcionalidad del método.
- **tratarBuscar(filtro: String): void**. Método que solicita al modelo el filtrado de las recetas. El filtro aplicado se recibe a través del parámetro *filtro*.
- **tratarBotonPapelera(informacion: Object): void**. Método que actúa sobre una interacción del usuario (respondiendo a la selección del botón *Papelera*).
- **tratarOk(informacion: Object): void**. Método que actúa sobre una interacción del usuario (respondiendo a un *Ok*). A través del parámetro se recibe la información necesaria para el funcionamiento del método.

Capítulo 4

MODELO DE IMPLEMENTACIÓN

En este capítulo se presenta cómo se adapta el modelo de diseño planteado en un capítulo anterior, al lenguaje de programación elegido. Inicialmente, se presenta algunas características de Android para, a continuación, indicar las modificaciones del diseño en función de estas características.

4.1 Adecuación a Android

Android es un conjunto de herramientas de software de código abierto para dispositivos móviles que fueron creadas por *Google* y la *Open Handset Alliance*. Android proporciona un *framework* de aplicaciones que permiten a los desarrolladores construir aplicaciones innovadoras escribiendo código usando el lenguaje Java, y controlando el dispositivo a través de bibliotecas Java.

Las aplicaciones Android se construyen como una combinación de componentes que pueden ser invocados individualmente. Desde un componente se puede iniciar otro, lo que se hace en Android mediante un *Intent*.

Android proporciona un marco de aplicación adaptable a diferentes dispositivos móviles que le permite proporcionar recursos únicos para diferentes configuraciones. Esto se consigue mediante carpetas específicas en el directorio *res* del proyecto creado (para iconos en las carpetas *drawable*, para los literales, dimensiones, estilos, en las carpetas *values* y para los layouts de las pantallas en las carpetas *layout*).

Las herramientas del SDK de Android compilan el código junto con todos los datos y recursos en un *paquete de Android* (*APK*). Un archivo *APK* contiene todo el contenido de una aplicación para Android y es el archivo que los dispositivos con Android utilizan para instalar la aplicación. Una vez instalado en un dispositivo, cada aplicación Android vive en su propio entorno limitado en seguridad. De esta forma, el sistema Android implementa el principio de *privilegios mínimos*. Cada aplicación sólo tiene acceso a los componentes que necesita para hacer su trabajo y nada más. Esto crea un ambiente muy seguro en el que una aplicación no puede tener acceso a partes del sistema para el cual no se le ha dado permiso (los permisos se solicitan en el archivo *manifiesto* propio de cada proyecto).

Los componentes de una aplicación son los componentes esenciales para su construcción. Cada componente es un punto diferente a través del cual el sistema puede entrar a su aplicación. No todos los componentes son los puntos de entrada reales para el usuario y en algunos casos, unos componentes dependen de otros, pero cada uno existe como una entidad propia y juega un rol específico que ayuda a definir el comportamiento general de la aplicación.

En Android hay cuatro tipos diferentes de componentes. Cada tipo tiene un propósito distinto y tiene un ciclo de vida diferente:

- **Componente *Activity*: actividad**. Un *activity* representa una única pantalla con una interfaz de usuario. Por ejemplo, una aplicación de correo electrónico puede tener una actividad que muestra una lista de los nuevos mensajes de correo electrónico, otra actividad para redactar un correo electrónico, y otra de las actividades para la lectura de mensajes de correo electrónico. Aunque las actividades trabajan juntas para formar una experiencia de usuario coherente en la aplicación de correo electrónico, cada una es independiente de las demás. Una actividad de este estilo se implementa como una subclase de la clase *Activity*.

- **Componente *Service*: servicio**. *Service* es un componente que se ejecuta en un segundo plano para realizar operaciones de larga ejecución o para realizar un trabajo para procesos remotos. Este componente no proporciona una interfaz de usuario. Un ejemplo podría ser a la hora escuchar la radio en segundo plano mientras el usuario está en una aplicación diferente, o cuando obtiene datos sobre la red sin bloquear la interacción del usuario con una actividad. Un componente *service* se implementa como una subclase de la clase *Service*.

- **Componente *Content provider*: proveedor de contenido**. Un *Content provider* gestiona un conjunto compartido de datos de aplicaciones. Puede almacenar los datos en el sistema de archivos, una base de datos, en la web, o cualquier otro lugar de almacenamiento permanente donde la aplicación pueda tener acceso. A través del *content provider*, otras aplicaciones pueden consultar o incluso modificar los datos. Un ejemplo sería cuando el sistema Android ofrece un *content provider* que gestiona la información de contacto del usuario. Un *proveedor de contenido* se implementa como una subclase de la clase *ContentProvider* y debe implementar un conjunto estándar de APIs que permitan a otras aplicaciones realizar transacciones.

- **Componente *Broadcast receivers*: receptores de radiodifusión**. Un *Broadcast receivers* es un componente que responde a transmitir anuncios de todo el sistema. Muchas de estas emisiones se originan en el propio sistema, por ejemplo, a la hora de anunciar que la pantalla se ha apagado, la batería está baja, o una imagen ha sido capturada. Las aplicaciones también pueden iniciar este tipo de emisiones, por ejemplo, cuando algunos datos se han descargado en el dispositivo, la aplicación puede avisar de que está disponible para que los pueda utilizar. Aunque los receptores de radiodifusión no muestran

una interfaz de usuario, es posible crear una notificación en la barra de estado para alertar al usuario cuando se produce un evento de difusión. Un *receptor de radiodifusión* se implementa como una subclase de *BroadcastReceiver* y cada emisión se entrega como un objeto *Intent*.

Un aspecto único del diseño del sistema Android es que cualquier aplicación puede iniciar el componente de otra aplicación. Por ejemplo, si se deseara capturar una fotografía con la cámara del dispositivo, es probable que exista otra aplicación que haga esto y que su aplicación pueda utilizarlo, en lugar de desarrollar una actividad para capturar una fotografía por sí mismo. No es necesario incorporar o incluso enlazar con el código de la aplicación de la cámara. En su lugar, sólo tiene que iniciar la actividad en la aplicación de la cámara que captura una foto. Cuando se haya completado la operación, la foto no regresará a su aplicación, la de la cámara, sino a la que lo llamó para realizar la acción, y para el usuario parece como si la cámara es en realidad una parte de su aplicación.

4.2 La clase AppMediador

Como se comentó en el *Modelo de diseño*, la aplicación utiliza la arquitectura *Modelo-Vista-Presentador* (MVP) adaptada al lenguaje elegido (Java). En esta adaptación, se incorpora una nueva clase, llamada *AppMediador* (introducida en el *Modelo de Diseño*) que va a funcionar como punto de enlace de las distintas partes de la arquitectura. El *AppMediador* representa a la clase *Application* (Aplicación) y se encarga de:

- Definir los presentadores y vistas y los métodos *accessor* de éstos.
- Definir la navegación en la aplicación, es decir, qué vistas (de la interfaz de usuario) se deben lanzar cuando se considere oportuno (normalmente, ante peticiones del usuario). Asimismo, definir qué servicios se lanzan cuando sea necesario.
- Definir las constantes correspondientes a los avisos de notificación, enviadas desde el modelo cuando se haya terminado de realizar una determinada acción.
- Definir las constantes correspondientes a los datos comunicados entre vistas, o recuperados desde el modelo, entre otros, ya que en Android los datos se comunican usando pares *clave, valor*.
- Definir los métodos de manejo de los componentes de Android, correspondientes a: el lanzamiento de actividades, la creación de servicios, los registros de receptores de notificación, los envíos de notificaciones *broadcast*, entre otros.

Es importante comentar que para que la aplicación desarrollada utilice este objeto *AppMediador*, en el archivo *AndroidManifest.xml* hay que indicar que el nombre de la aplicación corresponde con este archivo (indicando el paquete en el que se encuentra el mismo). Para ello, en este archivo hay que añadir, dentro de la etiqueta *application*, lo siguiente:

```
<application
        android:name="tfg.android.omm.AppMediador"
        ...>
</application>
```

El *AppMediador* contiene una variable de instancia (*private*) por cada una de las interfaces del presentador y la vista de la aplicación desarrollada, además de la variable *singleton* (*private* y *static*) de tipo *AppMediador*, para poder acceder a éste desde cualquier punto.

Por otro lado, se definen unas constantes de petición y notificación (*public* y *static*) que serán las claves de los datos a comunicar (clave, valor) o los avisos de notificación. Las constantes de notificación se utilizarán en los objetos *BroadcastReceiver* y en los métodos *sendBroadcast* para notificar cuándo ha finalizado un evento en la aplicación (por ejemplo, cuando se notifica la inserción de datos en la base de datos). Estos serán utilizados por los presentadores y el modelo.

Además, se implementan los siguientes métodos:

- Los métodos *accessor* de las vistas definidas en el capítulo *Modelo de Diseño*: *getVistaXXX* y *setVistaXXX*, donde *XXX* corresponde con el nombre de cada una de las vistas. Estos métodos son utilizados para obtener o iniciar las variables de las actividades (*Activity* de Android).
- Los métodos de creación y eliminación de los distintos presentadores definidos en el *Modelo de Diseño*: *getPresentadorXXX* y *removePresentadorXXX*, donde *XXX* corresponde con el nombre de cada uno de los presentadores.
- Los métodos de lanzamiento de actividades, servicios y receptores broadcast:
  - **launchActivity**: lanza una actividad.
  - **launchActivityForResult**: lanza una actividad y espera un resultado de respuesta.
  - **launchService**: lanza un servicio.
  - **stopService**: para un servicio previamente lanzado.
  - **registerReceiver**: registra un receptor a la espera de una determinada notificación.
  - **unRegisterReceiver**: des-registra un receptor.

  - **sendBroadcast**: envía una notificación a todos los receptores (sólo aquel que espere la notificación enviada, la atenderá).

- Los métodos de navegación para cada una de las vistas: *getVistaParaXXX*, donde *XXX* corresponde con el nombre de cada una de las vistas definidas en el *Modelo de Requisitos*. Estos métodos devuelve un objeto de tipo *Class* que representa a la clase de la vista correspondiente.

- Y por último, el método que crea la aplicación: *onCreate*. Este método inicia todas las variables de los presentadores a *null* e indica que la clase *AppMediador* es un *singleton*.

4.3 Paquete modelo

En este apartado se explican los cambios y adaptaciones que se han tenido que realizar en las clases e interfaces del modelo con respecto a las definidas en el *Modelo de Diseño*. Los cambios realizados son los siguientes:

1. **Modificaciones de los métodos descubiertos:**

   - Se ha modificado el método ***actualizarPerfil(Object info)*** para que reciba por parámetros todos los campos que se actualizarán.
   - Se ha eliminado el método ***obtenerVideo(String idReceta)*** porque Android tiene una forma particular de realizar este proceso.
   - Se ha eliminado el método ***obtenerImagen(String idReceta)*** porque su cometido lo realizan otros métodos.

2. **Modificaciones no descubiertas en el *Modelo de diseño*:**

   - Se ha añadido una clase por cada una de las listas personalizadas que se han creado en la aplicación, para agilizar la recolección y manejo de los datos. Las listas son: lista de recetas, lista de mis recetas y lista de favoritos. Así, las nuevas clases son:
     - **InfoListaReceta**. Clase que contiene los métodos *accessor* de los datos de las recetas a mostrar en la lista de recetas.
     - **InfoListaMisRecetas**. Clase que contiene los métodos *accessor* de los datos de las recetas creadas por el usuario a mostrar en la lista de sus recetas particulares.
     - **InfoListaFavoritos**. Clase que contiene los métodos *accessor* de los datos de las recetas marcadas por el usuario como favoritas a mostrar en la lista de recetas favoritas.
     - **TipoFila**. Clase que contiene los métodos *accessor* de todos los datos de una receta (de la lista de recetas).

  - **TipoFilaMisRecetas**. Clase que contiene los métodos *accessor* de todos los datos de una receta creada por el usuario (de la lista de mis recetas).
  - **TipoFilaFavoritos** Clase que contiene los métodos *accessor* de todos los datos de una receta favorita del usuario (de la lista de recetas favoritas).
- Se ha añadido el método ***void obtenerRecetasFavoritas(Object info)***. Método que obtiene de la base de datos la información de las recetas que el usuario tiene marcadas como favoritas, cuyos identificadores recibe por parámetros.
- Se ha añadido el método ***void cerrar()***. Método que cierra la sesión de usuario y presenta la pantalla principal de la aplicación.

4.4 Paquete vista

En este apartado se explican los cambios y adaptaciones que se han tenido que realizar en las clases e interfaces de las distintas vistas. Inicialmente se presentarán los cambios realizados en todas las vistas por aspectos no descubiertos en el *Modelo de Diseño* y por el uso de Android. A continuación, se presentarán los cambios en cada vista particular, por los mismos motivos.

4.4.1 Modificaciones en todas las clases del paquete vista

1. **Modificaciones no descubiertas en el *Modelo de diseño*:**
   - Se ha añadido, en todas las clases del paquete, el método:
     - ***void iniciar(Object informacion)***. Método que inicia la vista con la información adecuada.
2. **Modificaciones debidas a las particularidades de Android:**
   - Se han añadido los métodos:
     - ***void onCreate(Bundle savedInstanceState)***. Método que inicia la actividad. Sólo se ejecuta al iniciarse la actividad.
     - ***void onDestroy()***. Método para finalizar y cerrar una actividad.
     - ***void onBackPressed()***. Método que se utiliza para realizar las operaciones oportunas cuando se quiere volver a la vista anterior.
     - ***boolean onNavigationItemSelected(MenuItem item)***. Método para navegar entre los ítem del menú.
     - Métodos para el manejo y la visualización del menú de navegación personalizado.

- Se han añadido las siguientes clases:
  - Para controlar el menú de navegación:
    - **ExpandListAdapter**. Clase para manejar el menú de navegación.
  - Clases adaptadores, encargadas de crear, con los datos correspondientes, las listas personalizadas, objetos *RecyclerView* , necesarias para la aplicación. Estos adaptadores son:
    - **AdaptadorListaRecetas**. Clase necesaria para crear la lista de recetas con el formato definido en el archivo xml: *content_lista_recetas*. Para este adaptador ha sido necesario diseñar el formato de cada fila de la lista, mediante el archivo xml, *row_recycler*.
    - **AdaptadorListaMisRecetas**. Clase necesaria para crear la lista de recetas creadas por el usuario con el formato definido en el archivo xml: *content_mis_recetas*. Para este adaptador ha sido necesario diseñar el formato de cada fila de la lista, mediante el archivo xml, *row_recycler_mis_recetas*.
    - **AdaptadorFavoritos**. Clase necesaria para crear la lista de recetas con el formato definido en el archivo xml: *content_favoritos*. Para este adaptador ha sido necesario diseñar el formato de cada fila de la lista, mediante el archivo xml, *row_recycler_favoritos*.

4.4.2 Modificaciones en la Clase VistaAgregarEditarReceta

Las modificaciones propias de esta clase en los métodos descubiertos en el *Modelo de diseño* o debidas a las particularidades de Android son:

1. **Modificaciones en los métodos descubiertos en el *Modelo de diseño*:**
   - Se ha eliminado el método ***setImagen(Object imagen)*** puesto que el método no es utilizado.
2. **Modificaciones debidas a Android:**
   - Se ha añadido el método ***void onClick(View v)***. Controla los eventos relacionados al apretar un botón.
   - Se ha añadido el método ***protected onActivityResult(int requestCode, int resultCode, Intent data)***. Espera por el resultado enviado por una vista que se ha lanzado desde esta vista.
   - Se ha añadido el método ***boolean onCreateOptionsMenu(Menu menu)***. Crea los botones en el *Action Bar*.
   - Se ha añadido el método ***boolean onOptionsItemSelected(MenuItem item)***. Maneja los botones en el *Action Bar*

4.4.3 Modificaciones en la Clase VistaFavoritos

Las modificaciones propias de esta clase en los métodos descubiertos en el *Modelo de diseño* o debidas a las particularidades de Android son:

1. **Modificaciones debidas a Android:**
   - Se ha añadido el método ***boolean onCreateOptionsMenu(Menu menu)***. Crea los botones en el *Action Bar*.
   - Se ha añadido el método ***void onClick(View v)***. Controla los eventos relacionados al apretar un botón.
   - Se ha añadido el método ***void onResume(View v)***. Invocado cuando la vista pasa a estar en *Foreground*.
   - Se ha añadido el método ***boolean onOptionsItemSelected(MenuItem item)***. Maneja los botones en el *Action Bar*

4.4.4 Modificaciones en la Clase VistaLogin

Las modificaciones propias de esta clase en los métodos descubiertos en el *Modelo de diseño* o debidas a las particularidades de Android son:

1. **Modificaciones debidas a Android:**
   - Se ha añadido el método ***void onClick(View v)***.Controla los eventos relacionados al apretar un botón.

4.4.5 Modificaciones en la Clase VistaPerfil

Las modificaciones propias de esta clase en los métodos descubiertos en el *Modelo de diseño* o debidas a las particularidades de Android son:

1. **Modificaciones debidas a Android:**
   - Se ha añadido el método ***boolean onCreateOptionsMenu(Menu menu)***. Crea los botones en el *Action Bar*.
   - Se ha añadido el método ***void onClick(View v)***. Controla los eventos relacionados al apretar un botón.
   - Se ha añadido el método ***boolean onOptionsItemSelected(MenuItem item)***. Maneja los botones en el *Action Bar*
   - Se ha añadido el método ***protected onActivityResult(int requestCode, int resultCode, Intent data)***. Espera por el resultado enviado por una vista que se ha lanzado desde esta vista.

4.4.6 Modificaciones en la Clase VistaReceta

Las modificaciones propias de esta clase en los métodos descubiertos en el *Modelo de diseño* o debidas a las particularidades de Android son:

1. **Modificaciones en los métodos descubiertos en el *Modelo de diseño*:**
   - Se ha eliminado el método ***pintarEstrella()*** debido a que su función, en Android, se realiza automáticamente al apretar un botón.
2. **Métodos no descubiertos en el *Modelo de diseño*:**
   - Se ha añadido el método ***void tratarVideo()***. Método que lanza la *vistaVideo*.
3. **Modificaciones debidas a Android:**
   - Se ha añadido el método ***boolean onCreateOptionsMenu(Menu menu)***. Crea los botones en el *Action Bar*.
   - Se ha añadido el método ***void onClick(View v)***. Controla los eventos relacionados al apretar un botón.
   - Se ha añadido el método ***boolean onOptionsItemSelected(MenuItem item)***. Maneja los botones en el *Action Bar*

4.4.7 Modificaciones en la Clase VistaRegistro

Las modificaciones propias de esta clase en los métodos descubiertos en el *Modelo de diseño* o debidas a las particularidades de Android son:

1. **Modificaciones debidas a Android:**
   - Se ha añadido el método ***void onClick(View v)***. Controla los eventos relacionados al apretar un botón.
   - Se ha añadido el método ***protected onActivityResult(int requestCode, int resultCode, Intent data)***. Espera por el resultado enviado por una vista que se ha lanzado desde esta vista.

4.4.8 Modificaciones en la Clase VistaVideo

Las modificaciones propias de esta clase en los métodos descubiertos en el *Modelo de diseño* son:

- Se ha añadido la clase ***MyWebClient*** que extiende de ***WebViewClient***, para el manejo y visualización de los videos desde cualquier plataforma de internet.

4.5 Paquete presentador

En este apartado se explican los cambios y adaptaciones que se han tenido que realizar en las clases e interfaces de los presentadores. Inicialmente se presentarán los cambios realizados en todas los presentadores por aspectos no descubiertos en el *Modelo de Diseño* y por el uso de Android. A continuación, se presentarán los cambios en cada presentador particular, por los mismos motivos.

4.5.1 Modificaciones en todas las clases del paquete presentador

1. **Modificaciones no descubiertas en el *Modelo de diseño*:**
   - En todos los presentadores se ha incluido el siguiente método:
     - ***void iniciar()***. Inicia el menú de navegación personalizado y lanza la vista correspondiente a cada opción de menú.
2. **Modificaciones debidas a Android:**
   - En todos los presentadores se ha añadido la variable de instancia (*private*) *receptor* de tipo **BroadcastReceiver** para detectar los avisos (*notificaciones por parte del modelo*). Además, en éstos, se redefine el siguiente método:
     - ***void onReceive(Context context, Intent intent)***. Método que se encarga de recibir las notificaciones de los diferentes avisos entre los distintos presentadores y el modelo y de realizar las acciones oportunas cuando este aviso se recibe.

4.5.2 Modificaciones en la Clase PresentadorFavoritos

La modificación propia de esta clase en los métodos descubiertos en el *Modelo de diseño* es:

- Se ha eliminado el método ***void tratarFavoritos(Object informacion)*** debido a que la vista no se llama a si misma.
- Se ha añadido el método ***void tratarEliminados(Object informacion)***. Método que manda al modelo la lista de recetas marcadas por el usuario para eliminar de su lista de recetas favoritas.
- Se ha eliminado el método ***tratarBuscar(String filtro)*** debido a que no es utilizado.

4.5.3 Modificaciones en la Clase PresentadorListaRecetas

La modificación propia de esta clase en los métodos descubiertos en el *Modelo de diseño* es:

- Se ha eliminado el método ***void tratarRecetas(Object informacion)*** debido a que la vista no se llama a si misma.
- Se ha eliminado el método ***tratarBuscar(String filtro)*** debido a que no es utilizado.

4.5.4 Modificaciones en la Clase PresentadorLogin

La modificación propia de esta clase en los métodos descubiertos en el *Modelo de diseño* es:

- Se ha eliminado el método ***void tratarLogin(Object informacion)*** debido a que la vista no se llama a si misma.

4.5.5 Modificaciones en la Clase PresentadorMisRecetas

La modificación propia de esta clase en los métodos descubiertos en el *Modelo de diseño* es:

- Se ha eliminado el método ***void tratarMisRecetas(Object informacion)*** debido a que la vista no se llama a si misma.
- Se ha eliminado el método ***void tratarBotonLapiz(Object informacion)*** debido a que no es utilizado.
- Se ha eliminado el método ***void tratarBotonPapelera(Object informacion)*** debido a que no es utilizado.
- Se ha eliminado el método ***void tratarBuscar(String filtro)*** debido a que no es utilizado.

4.5.6 Modificaciones en la Clase PresentadorPerfil

La modificación propia de esta clase en los métodos descubiertos en el *Modelo de diseño* es:

- Se ha eliminado el método ***void tratarPerfil(Object informacion)*** debido a que la vista no se llama a si misma.

4.5.7 Modificaciones en la Clase PresentadorVideo

La modificación propia de esta clase en los métodos descubiertos en el *Modelo de diseño* es:

- Se ha eliminado el método ***obtenerVideo(String idReceta)*** debido a que nunca es utilizado y su función se realiza de forma particular en Android.

Capítulo **5**

MODELO DE PRUEBAS

5.1 Test de usabilidad de la interfaz de usuario

El propósito de este test de usabilidad es descubrir en qué grado la interfaz de la aplicación *AppRecetas* cumple con su propósito y satisface los objetivos en cuanto a efectividad, eficiencia y satisfacción. El test es una observación de cómo interactúan los usuarios representativos con la interfaz. La realización de este test de usabilidad está motivado por la necesidad de detectar debilidades en el producto y su nivel de utilidad, es decir, revelar si el usuario encuentra provecho, beneficio y muestra interés. El test se realiza para personas habituadas al uso de aplicaciones móviles.

5.1.1 Explicación del producto

La Aplicación *AppRecetas* facilita al usuario la consulta de recetas de la gastronomía canaria. También incluye una de las características más importante de las redes sociales, al permitir al usuario el compartir y publicar la receta que conozca. Esto ayuda al usuario a conocer, preparar y conservar los platos típicos de cocina canaria. Sus características principales son:

- Diseño claro y simple.
- Clasificación natural de las recetas (enyesques, sopas y potajes, postres, etc.).
- Indicación de la localización origen de la receta.
- Explicación detallada sobre la elaboración de la receta.
- Visualización de la preparación de la receta.
- Añadir recetas a una lista de *Favoritos*.
- Creación de nuevas recetas.
- Conservación de acervo cultural.

5.1.2 Objetivos del producto

La aplicación *AppRecetas* tiene como objetivo dar a conocer y contribuir a la conservación de las recetas de la gastronomía canaria. La aplicación pretende ayudar al usuario en los siguientes aspectos:

- Tener un recetario disponible en cualquier momento y lugar.
- Repasar los ingredientes del plato para ver si le falta alguno.
- Visualizar la preparación de las recetas.
- Tener al alcance las diversas formas de preparar un mismo plato, puesto que en cada isla es probable que se prepare con distintas peculiaridades.

Las mejoras que aporta son:

- El usuario obtiene de forma rápida la lista de ingredientes y los pasos a seguir para elaborar una receta canaria.
- Contribuye a la conservación de las recetas canarias dando la posibilidad a los usuarios más jóvenes y que han tenido que emigrar, a preparar los platos con los que se criaron.

5.1.3 Usuarios y participantes

El producto está destinado a usuarios primarios. Se tendrán en cuenta los usuarios terciarios por su capacidad de influencia en la alimentación, ya que la preparación de la comida depende en muchas ocasiones de la opinión y los gustos de varias personas.

El tipo de usuario que realizará el test de usabilidad no debe ser experto, pero sí estar familiarizado con el uso de aplicaciones móviles, y habituado a preparar su comida, para ser capaces de sacar el máximo provecho al test.

5.1.3.1. Usuarios finales del producto

El perfil del usuario final a quien va dirigido el producto es una persona de entre 18 y 90 años, de cualquier género, que realiza de forma habitual tareas del hogar relacionadas con la preparación del menú familiar o bien contribuye a la misma (pareja, amigo, etc.). Adicionalmente, el usuario tiene intereses en llevar una alimentación sana y equilibrada.

5.1.3.2. Criterios de selección del grupo de test

Es muy importante que los participantes sean lo más representativos posible ya que, normalmente, no hay posibilidad de trabajar con grandes muestras para hacer las pruebas. Por esta razón se seguirán los siguientes criterios para seleccionar los usuarios que realizarán los tests:

- Español con edad del cliente objetivo (18-90 años)
- Debe disponer de un terminal móvil (teléfono o tableta).
- Sabe como usar un terminal móvil (teléfono o tableta).
- Alguna vez se ha interesado o descargado aplicaciones móviles.
- Habituado a preparar el menú del hogar.
- Tiene inquietud por elaborar platos canarios y obtener una dieta equilibrada.

Para realizar las pruebas, se elegirán usuarios de entornos familiares que nos permitan observar y detectar necesidades reales dentro de los procesos que se desarrollan de forma natural en el hogar.

5.1.4 Diseño del proceso de test

En los siguientes apartados se explica cómo se realizará el proceso de test, herramientas necesarias, tareas del usuario, cómo se obtendrán las evidencias, etc.

5.1.4.1. Logística

La evaluación se llevará a cabo en un ambiente libre de interferencias profesionales y personales durante el periodo del test, el cual tendrá una duración de 15 minutos. La ubicación ideal para esta tarea es el domicilio del usuario, pero no siempre será posible, por lo que se optarán por lugares donde el usuario se sienta cómodo.

Los tests se realizarán los días 11 y 12 de Marzo de 2017. El análisis de los datos se llevará a cabo la semana siguiente a la recopilación de los mismos, y se obtendrán los resultados el viernes 17 de Marzo de 2017.

Antes de iniciar la evaluación, se le dará al usuario una breve descripción sobre el propósito de la aplicación y se le explicará las tareas (no las acciones) que debe realizar de forma sencilla y clara. Una vez finalizada la evaluación, se entregará al usuario un cuestionario de usabilidad para obtener las evidencias. En dicho cuestionario se recopilarán datos de usabilidad, utilidad y estética de la aplicación.

5.1.4.2. Instrumentación

El material que necesitará el usuario para realizar el test será un ordenador o tableta que disponga de navegador y conexión a Internet. Desde el navegador se lanzará la herramienta de prototipado Balsamiq Mockups [23], con la que el usuario realizará el test en modo presentación siguiendo las pautas indicadas.

En caso de no disponer del material hardware o de conexión a Internet, debido al lugar donde se realizan las pruebas o a cualquier otra circunstancia, el evaluador dispondrá de un portátil o tableta con el PDF exportado de Balsamiq para que el usuario realice el test.

5.1.4.3. Tareas a realizar

El usuario debe realizar sin ayuda cada una de las tareas que se presentan a continuación. Para ello, debe seguir, en el orden indicados, las acciones asociadas a cada una.

Tarea nº 1: Acceder a la lista de las recetas catalogadas como postres.
Acciones:

1. Acceder a la primera pantalla de la aplicación.
2. Suponiendo que no se ha introducido el usuario y la contraseña, acceder a las recetas de los *Postres*.
3. Regresar a la pantalla de inicio.

Tarea nº 2: Seleccionar la receta de "Rancho Canario".
Acciones:

1. Acceder a la primera pantalla de la aplicación.
2. Suponiendo que no se ha introducido el usuario y la contraseña, acceder a las recetas de las *Sopas y potajes*.
3. Acceder a la receta *"Rancho Canario"*.

Tarea nº 3: Acceder al vídeo sobre la receta.
Acciones:

1. Acceder a la primera pantalla de la aplicación.

2. Suponiendo que no se ha introducido el usuario y la contraseña, acceder a las recetas de las *Sopas y potajes*.
3. Acceder a la receta *"Rancho Canario"*.
4. Visualizar el vídeo.

Tarea nº 4: Registrarse en la aplicación.
Acciones:

1. Acceder a la primera pantalla de la aplicación.
2. Suponiendo que no se ha introducido el usuario y la contraseña, acceder a las recetas de los *Condumios*.
3. Seleccionar el botón "nuevo usuario".
4. Seleccionar el botón "Aceptar".

Tarea nº 5: Ver lista de favoritos.
Acciones:

1. Acceder a la primera pantalla de la aplicación.
2. Acceder a la pantalla de *login*.
3. *Loguearse* en la aplicación.
4. Acceder a la lista de favoritos.

Tarea nº 6: Eliminar una receta de la lista de favoritos.
Acciones:

1. Acceder a la primera pantalla de la aplicación.
2. Acceder a la pantalla de *login*.
3. *Loguearse* en la aplicación.
4. Acceder a la lista de favoritos.
5. Eliminar la receta *Quesadillas*.

Tarea nº 7: Añadir una receta a la lista de favoritos.
Acciones:

1. Acceder a la primera pantalla de la aplicación.
2. Acceder a la pantalla de *login*.
3. *Loguearse* en la aplicación.
4. Acceder a la receta *Rancho Canario*.
5. Añadir la receta a la lista de favoritos.

Tarea nº 8: Crear nueva receta.
Acciones:

1. Acceder a la primera pantalla de la aplicación.
2. Acceder a la pantalla de *login*.
3. *Loguearse* en la aplicación.
4. Acceder a añadir *Nueva Receta*.

Tarea nº 9: Editar Perfil.
Acciones:

1. Acceder a la primera pantalla de la aplicación.
2. Acceder a la pantalla de *login*.
3. *Loguearse* en la aplicación.
4. Acceder a *Mi Perfil*.
5. Apretar el botón *editar*.
6. Editar perfil.

Tarea nº 10: Eliminar cuenta.
Acciones:

1. Acceder a la primera pantalla de la aplicación.
2. Acceder a la pantalla de *login*.
3. *Loguearse* en la aplicación.
4. Acceder a *Mi Perfil*.
5. Apretar el botón *Eliminar Cuenta*.

5.1.4.4. Indicadores de éxito

Las tareas mencionadas anteriormente se habrán realizado correctamente cuando:

- **Tarea 1:** Aparezca la pantalla de inicio.
- **Tarea 2:** Aparezca la pantalla explicando detalladamente dicha receta.
- **Tarea 3:** Aparezca la pantalla de reproducción del vídeo.
- **Tarea 4:** Aparezca la pantalla con la lista de recetas "Enyesques".
- **Tarea 5:** Muestre en la pantalla la lista de las recetas marcadas como favoritas.
- **Tarea 6:** Desaparezca de la pantalla de favoritos, la receta eliminada.
- **Tarea 7:** Aparezca en la pantalla de favoritos, la receta añadida.
- **Tarea 8:** Aparezca la pantalla de creación de recetas nuevas.
- **Tarea 9:** Aparezca la pantalla de perfil con datos actualizados.
- **Tarea 10:** Aparezca la pantalla principal de la aplicación.

5.1.4.5. Respuesta de los usuarios a los indicadores

En la tabla 5.1 se muestra un resumen de las evidencias obtenidas, donde todos los usuarios han completado el 100 % de las tareas.

5.1.5 Conclusiones finales

Tras realizar el test a los usuarios y analizar los resultados obtenidos, se han detectado carencias en el diseño. La navegación a través del menú debe ser menos repetitiva y confusa. Mayor simplificación en el diseño reforzará los beneficios que ofrece la aplicación. La modificación en el diseño del prototipo contempla lo siguiente:

- Modificar el diseño del menú para que resulte más atractivo a la vista y con un mejor funcionamiento evitando la redundancia.
- Rediseñar los nombres y las pantallas de las categorías cambiando sus nombres.
- Rediseñar la vista donde se detalla la receta para añadir una galería de fotos, mostrando diversos momentos sobre la preparación del plato.

Parámetro observado	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
Nº de usuarios que no realizaron la tarea a tiempo	0	0	0	0	1	0	0	0	0	0
Nº de acciones ejecutadas incorrectamente o no realiadas	0	0	0	0	0	0	1	0	0	0
Nº de veces que los usuarios pidieron ayuda	0	0	0	0	0	0	0	0	0	1
Errores enncontrados	0	0	0	0	0	0	0	0	0	1
Nº de aspectos positivos expresados por el usuario	0	1	0	0	0	0	0	0	0	0
Nº de aspectos negativos expresados por el usuario	1	2	1	1	2	2	2	4	0	1

Tabla 5.1: Respuesta de los usuarios

- Eliminar el botón "Nueva receta" en la pantalla visualización del video para un usuario registrado.
- Eliminar el botón "Registrarse" en la pantalla visualización del video para un usuario no registrado.
- Rediseñar la vista *"Mis Favoritos"* para señalizar a qué categoría pertenece cada receta de dicha lista.
- Rediseñar la vista *"Nueva Receta"* para añadir la posibilidad de subir más de una foto, guardar la receta para poder continuar con su redacción en otro momento y añadir un botón de ayuda que explique la forma correcta para añadir más de una foto.
- Rediseñar la vista *"Nueva Receta"* para añadir un icono (botón) que permita compartir recetas y otro que permita guardar recetas no completas (hasta completarlas).
- Rediseñar la vista de información detallada de una receta para eliminar el botón + y sustituirlo por el botón *lápiz* (sólo debe aparecer para el usuario que añadió la receta a la aplicación).
- Añadir un botón para unir directamente a *"Mis favoritos"* una receta desde la vista de su explicación en detalle.
- Añadir un botón en el menú para desconectarse, pero continuar en la aplicación.

- Añadir una pantalla donde aparezcan todas las recetas creadas por el usuario y que permita editar o eliminar sus recetas. Esta vista debe presentar las recetas compartidas y las guardadas sin compartir separadas de forma clara.
- Añadir, en la vista de *login*, un botón para permitir al usuario recuperar su contraseña.

5.2 Test de verificación del modelo de datos

En este apartado se realizan algunos test para verificar el grado de funcionamiento de la aplicación Android desarrollada en este Trabajo Fin de Grado. A pesar de la realización de estos test, hay que señalar que nunca se va a poder demostrar que el software está completamente libre de defectos (*bugs*), pero sí ayudan a resolver los problemas de implementación más evidentes.

Es importante destacar que, antes de publicar cualquier aplicación, es fundamental realizar una batería de test completa. Llevar a cabo estas pruebas conlleva una técnica dinámica de verificación, ya que se requiere un prototipo ejecutable del sistema. Realmente, no se precisa que todo el sistema esté operativo, solamente se necesita que aquellas partes a probar, estén funcionando. En el caso de la arquitectura MVP, descrita en el capítulo *Modelo de diseño*, y que es la utilizada en el desarrollo de la aplicación *AppRecetas*, al estar los distintos componentes de la arquitectura desacoplados, el código de cualquiera de las partes se puede verificar de forma independientes.

En el caso que nos ocupa, las pruebas a realizar verificarán alguna de las funcionalidades del **modelo** de la aplicación, ya que ésta es la parte más crítica de la arquitectura MVP. De todos los posibles tests que se pueden realizar sobre el modelo (que verificarán algunos métodos significativos de las clases del modelo), se han elegido una serie de ellos , por motivos de tiempo de realización de este trabajo. Para verificar el código de las clases del modelo, se necesitan crear clases de tests, formadas por métodos de tests que prueben los distintos casos que se vayan a encontrar a la hora de invocar el método del modelo.

A continuación se describen los tests que se han realizado a seis métodos del modelo: *validarDatos*, *recupearPassword*, *registrarUsuario*, *agregarActualizarReceta* (todos de la clase *Modelo*), *obtenerListaRecetas* y el método *obtenerCategorias* (ambos de la clase *BDAdaptadorCategoria*).

5.2.1 Tests del método validarDatos

En este test, se comprueba el inicio de sesión de un usuario probando el método *validarDatos* de la clase *Modelo*. Para ello, como parámetros iniciales se dan un nombre de usuario (*email*) y un *password*, que serán variables predefinidas en cada método de test. Así, se han creado los siguientes métodos de test:

- **testComprobarLoginCorrecto(): void**. Método en el que se define un *email* y una contraseña que coinciden con los de un usuario registrado en la aplicación y, por tanto, almacenados en la nube. En este test se llama al método *validarDatos* del *Modelo*, pasándole por parámetros el *email* y la contraseña anteriores, los cuáles son válidos. Tras esto, se accede a la base de datos para verificar los valores introducidos. Una vez realizado este proceso, se responde con un resultado, que debe ser comprobado en el método de test. En la figura 5.1 se demuestra el resultado final del test, donde la barra de color verde indica que se ha ejecutado satisfactoriamente.

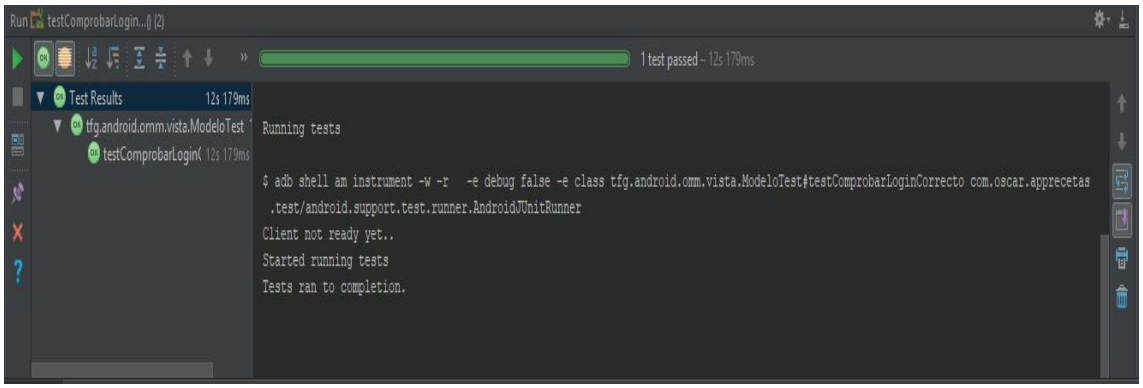


Figura 5.1: Resultado del test *testCommprobarLoginCorrecto*

- **testComprobarLoginIncorrecto(): void**. Método en el que se define un *email* inválido, es decir, que no se ajusta al formato de *emails* o que no es correcto, y una contraseña. Al igual que en el test anterior, se llama al método *validarDatos* de la clase *Modelo*, al que se le pasa por parámetros el *email* y la contraseña anteriores. Tras comprobar los valores introducidos en la base de datos, se responde con un resultado que indica que el mail es incorrecto (que es lo esperado). En la figura 5.2 se demuestra el resultado final del test, donde la barra de color verde indica que se ha ejecutado satisfactoriamente.

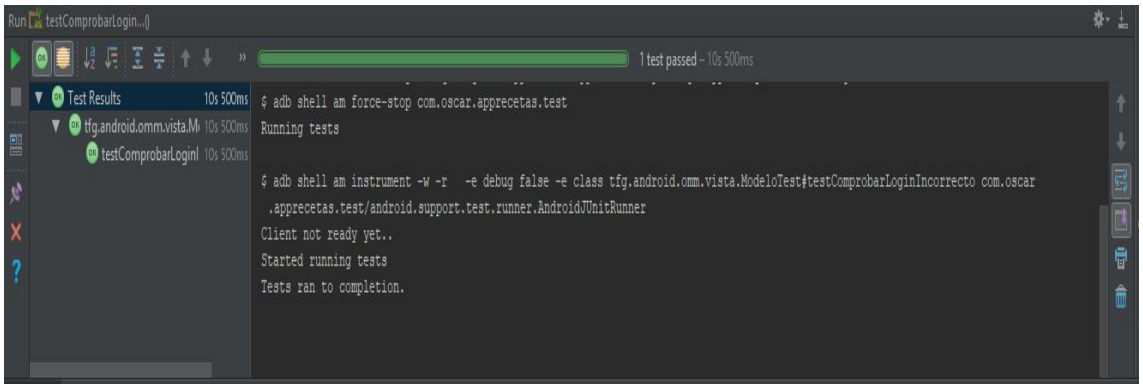


Figura 5.2: Resultado del test *testComprobarLoginIncorrecto*

- **testComprobarLoginEmailCorrectoPassIncorrecto(): void**. Método en el que se define un *email* correspondiente al de un usuario de la aplicación, y una contraseña inválida, es decir, que no coincide con la del usuario pasado por parámetros. Al igual que en el test anterior, se llama al método *validarDatos* de la clase *Modelo*, al que se le pasa por parámetros el *email* y la contraseña. Tras esto, se accede a la nube y se comprueba los datos, esperando a que se reciba el resultado que indica que la contraseña es incorrecta. En la figura 5.3 se demuestra el resultado final del test, donde la barra de color verde indica que se ha ejecutado satisfactoriamente.

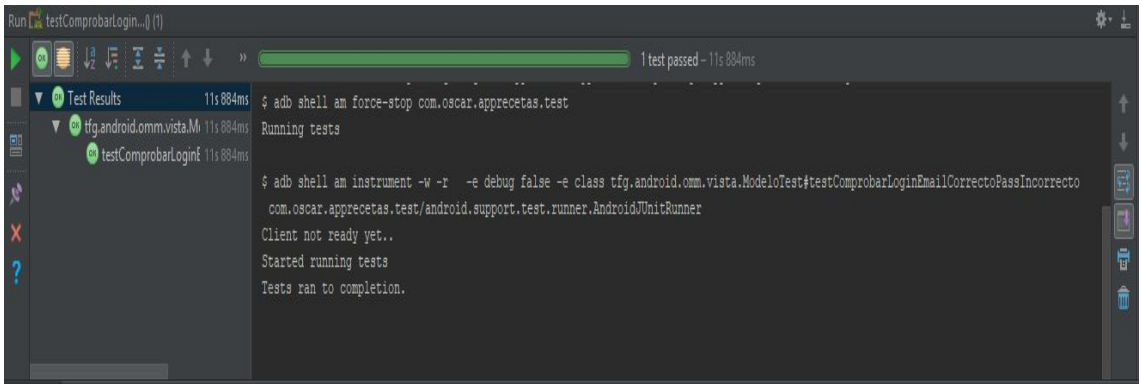


Figura 5.3: Resultado del test *testComprobarLoginEmailCorrectoPassIncorrecto*

5.2.2 Tests del método recuperarPassword

En este test, se comprueba el envío de un email de recuperación de la contraseña de un usuario probando el método *recuperarPassword* de la clase *Modelo*. Para ello, como parámetros iniciales, se dan un nombre de usuario (*email*) que será una variable predefinidas en cada método de test. Para esta verificación, se han creado los siguientes métodos de test:

- **testRecuperarPasswordEmailCorrecto(): void**. Método en el que se define un *email* correspondiente al de un usuario de la aplicación. En este test se llama al método *recuerarPassword* de la clase *Modelo*, al que se le pasa por parámetros el *email*. Tras esto, se accede a la nube y se comprueba la veracidad del mismo, esperando que se reciba el resultado. Como resultado se recibirá un correo electrónico de recuperación de contraseña en la cuenta de correo del usuario. En la figura 5.4 se demuestra el resultado final del test, donde la barra de color verde indica que se ha ejecutado satisfactoriamente la petición. Asimismo, en la figura 5.5 se muestra el correo recibido por parte del usuario cuando solicita una recuperación de password.

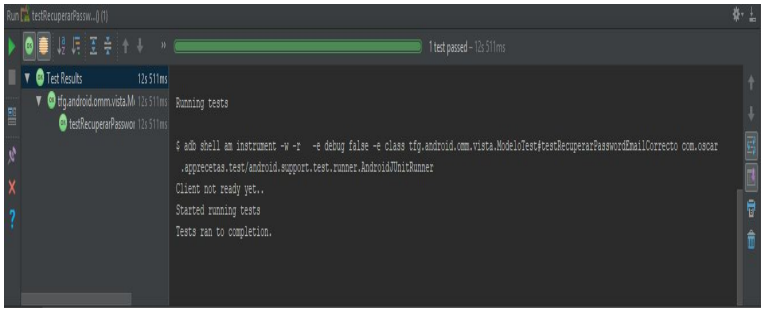


Figura 5.4: Resultado del test testRecuperarPasswordEmailCorrecto

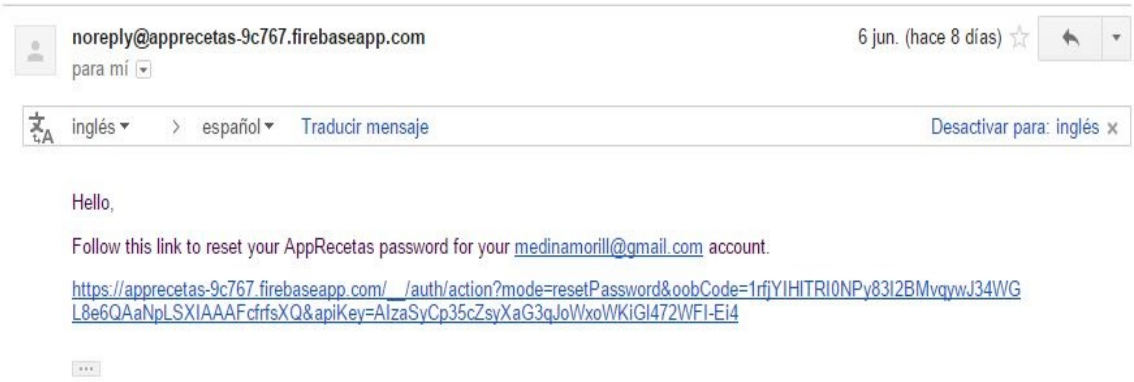


Figura 5.5: Correo recibido desde *Firebase* cuando se solicita la recuperación de la contraseña

- **testRecuperarPasswordEmailIncorrecto(): Void**. Método en el que se define un *email* inválido es decir, que no corresponde al de un usuario de la aplicación. En este test se llama al método *recuerarPassword* de la clase *Modelo*, al que se le pasa por parámetros el *email*. Tras esto, se accede a la nube y se comprueba la veracidad del mismo, esperando que se reciba el resultado. En este caso, en el resultado se indica que el *email* no es correcto (hecho comprobado en el test). En la figura 5.6 se demuestra el resultado final del test, donde la barra de color verde indica que se ha ejecutado satisfactoriamente.

5.2.3 Test del método registrarUsuario

En este test se comprueba el correcto funcionamiento del proceso de creación y registro de un usuario en la aplicación, a través del test al método *registrarUsuario* de la clase *Modelo*. En este test se requieren como parámetros iniciales los datos del nuevo usuario que se desea registrar. Todos estos datos serán variables predefinidas en el método de test (requisitos iniciales). En este caso, y al igual que con muchos de los tests que se pueden encontrar en la documentación adjunta al Trabajo Fin de Grado, la realización de los tests, concretamente los que implican escrituras en la

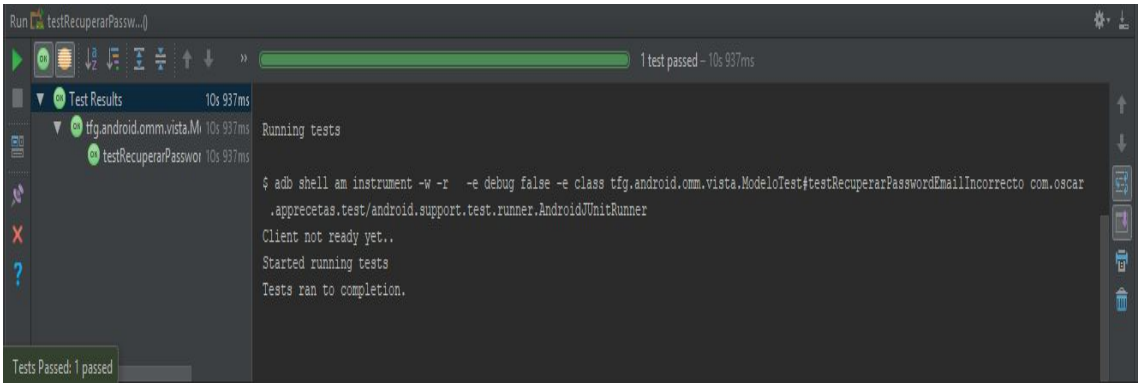


Figura 5.6: Resultado del test *testRecuperarPasswordEmailIncorrecto*

nube, se realizaron mediante el apoyo del *Dashboard* de Firebase, en el que se puede apreciar en tiempo real las creaciones, eliminaciones y modificaciones de los datos almacenados en dicho servicio, de forma gráfica. Gracias a ello, además de comprobar el valor devuelto a la hora de invocar al método a probar, se puede apreciar de forma visual el efecto de los tests sobre la base de datos. Para esta prueba, se realizó un método de test, que se comenta a continuación:

- **testCrearUsuario(): void**. Método en el que se define los datos necesarios para la creación de un usuario en la nube. En este test se llama al método *registrarUsuario* de la clase *Modelo*, pasándole por parámetros los datos del nuevo usuario. Tras la llamada, se accede a la base de datos y se comprueba la correcta creación del usuario, esperando a que se reciba el resultado. En la figura 5.7 se demuestra el resultado final del test, donde la barra de color verde indica que se ha ejecutado satisfactoriamente. Si bien con esto podría ser suficiente, como se comentó anteriormente, ante la posibilidad de utilizar el *dashboard* de Firebase como herramienta de verificación de tests, este test se apoyó en el uso de la misma, por lo que, tras ejecutar el método de test en el entorno de desarrollo, si se accede al servicio a través de la web, se podría comprobar el correcto resultado del test ejecutado. En la figura 5.8 se puede el antes (a) y el después (b) de la creación de un usuario.

5.2.4 Test del método obtenerCategorias

En este test se comprueba el correcto funcionamiento del proceso de obtención de datos desde la nube a través del test al método *obtenerCategorias* de la clase *BDAdaptadorCategoria*, que pertenece al modelo de datos. Para esta prueba se realizó el método de test que se presenta a continuación:

- **testObtenerCategorias(): void**. En este test se llama al método *obtenerCaegorias* de la clase *BDAdaptadorCategoria*. Tras llamar al método, se accede

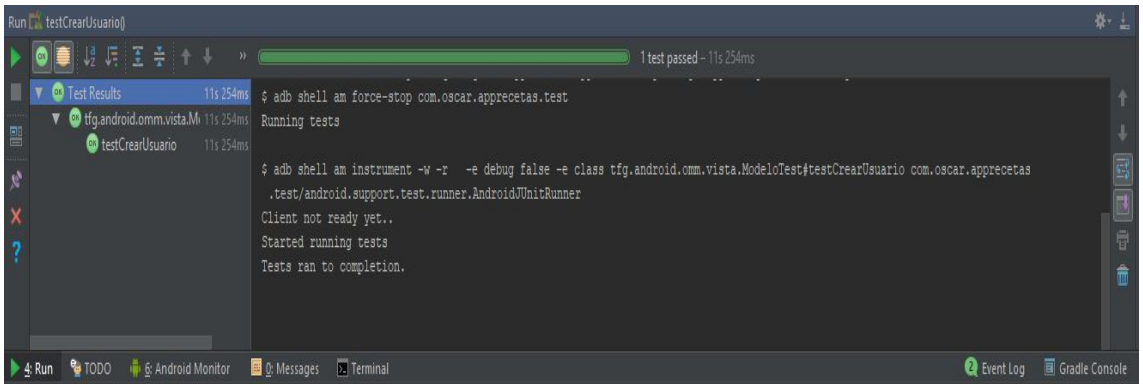
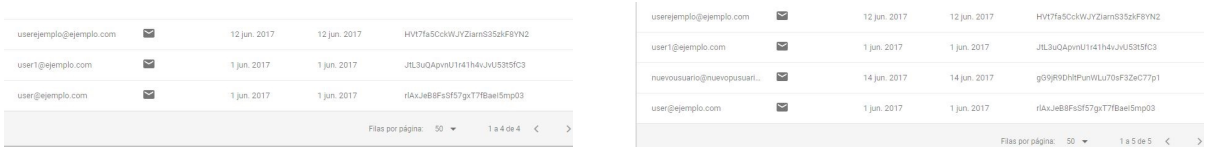


Figura 5.7: Resultado del test *testCrearUsuario*



(a) (b)

Figura 5.8: Resultado de la creación de un usuario en Firebase. a) Antes y b) Después

a la base de datos para recuperar la lista de categorías almacenadas, esperando a que se reciba el resultado. En este caso, el resultado será un array de objetos de tipo *Categoria* con las categorías almacenadas y esperadas. En la figura 5.9 se demuestra el resultado final del test, donde la barra de color verde indica que se ha ejecutado satisfactoriamente.

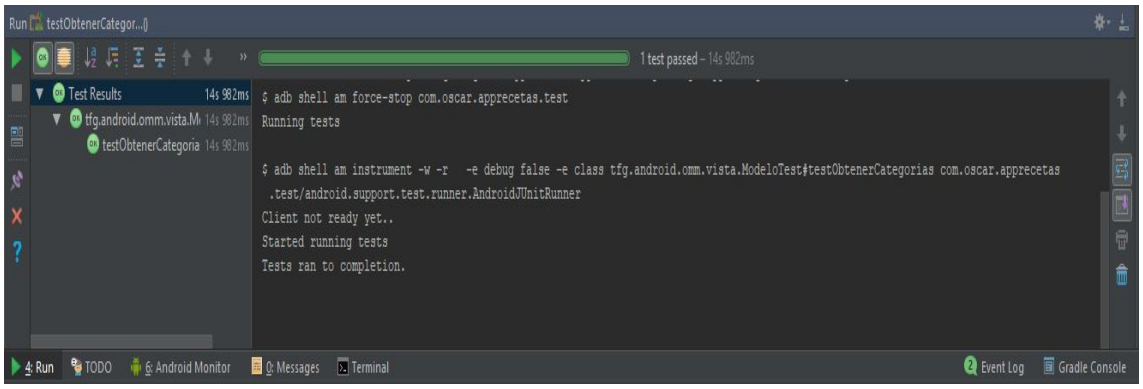


Figura 5.9: Resultado del test testObtenerCatgorias

5.2.5 **Test del método obtenerListaRecetas**

En este test se comprueba el correcto funcionamiento del proceso de obtención de una lista con las recetas de una determinada categoría, desde la nube a través del

test al método *obtenerListaRecetas* de la clase *BDAdaptadorCategoria*. Se requiere como parámetro inicial el identificador único de la categoría, que será una variable predefinida para el método de test (requisito inicial). Para esta prueba se realizó el método de test que se presenta a continuación:

- **testObtenerListaRecetas(): void**. Método en el que se define el identificador único de la categoría, de la que se desea obtener sus recetas. En este test se llama al método *obtenerListaRecetas* de la clase *BDAdaptadorCategoria*, pasándole por parámetro el identificador anterior. Tras esto, se accede a la base de datos para comprobar que existe una categoría asociada al identificador pasado, esperando a que se reciba el resultado. En este caso, el resultado será un array de objetos de tipo *RecetaBD* con la información de todas las recetas esperadas de la categoría. En caso contrario, el test fallará. En la figura 5.10 se demuestra el resultado final del test, donde la barra de color verde indica que se ha ejecutado satisfactoriamente.

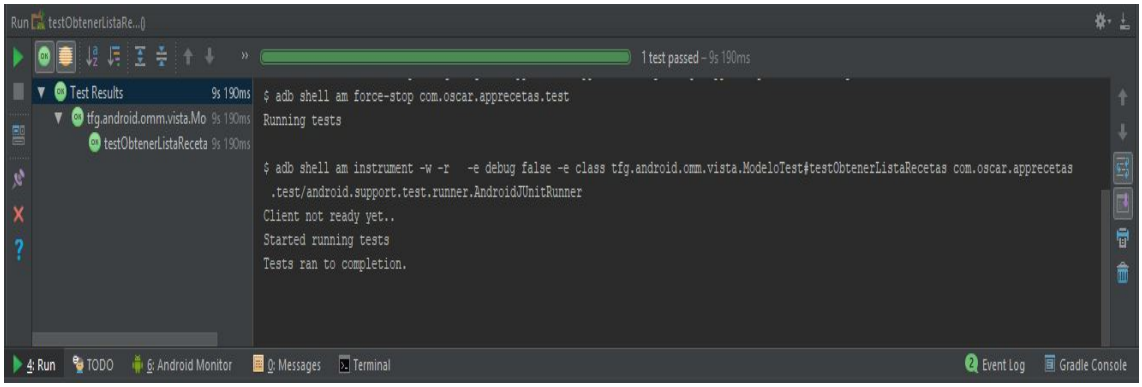


Figura 5.10: Resultado del test *testObtenerListaRecetas*

5.2.6 Test del método agregarActualizarReceta

En este test se comprueba el correcto funcionamiento del proceso de creación de una nueva receta para añadirla a la nube a través del test al método *agregarActualizarReceta* de la clase *Modelo*. Se requiere como parámetros iniciales los datos necesarios para la creación de la receta, que serán variables predefinidas para el método de test (requisito inicial). Para esta prueba se realizó el método de test que se presenta a continuación:

- **testAgregarReceta(): void**. Método en el que se definen los parámetros necesarios para la creación de una nueva receta. En este test se llama al método *AgregarActualizarReceta* de la clase *Modelo*, pasándole por parámetros la información anterior. Tras esto, se accede a la base de datos para crear un nuevo

registro. En este caso, el resultado será la creación del nuevo registro en la base de datos, en caso contrario, el test fallará. En la figura 5.11 se demuestra el resultado final del test, donde la barra de color verde indica que se ha ejecutado satisfactoriamente, mientras que en la figura 5.12 se muestra el resultado usando el *dashboard* de Firebase, tal y como se hizo anteriormente, donde se puede ver el antes (a) y el después (b) de la creación de la receta.

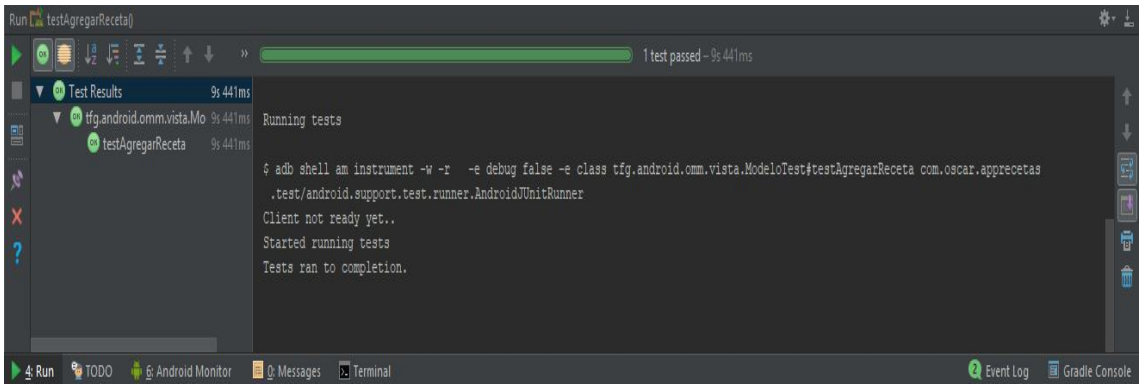


Figura 5.11: Resultado del test testAgregarReceta

Figura 5.12: Resultado de la creación de una receta en Firebase. a) Antes y b) Después

Capítulo **6**

CONCLUSIONES Y MEJORAS

6.1 Conclusiones

El Trabajo de Fin de Grado presentado en esta memoria fue desarrollado con el objetivo final de crear una aplicación en *Android* que permitiera a la población joven (y no tan joven) acceder a un almacenamiento de datos, en la que éstos, a través de sus dispositivos móviles, puedan consultar recetas de la cocina canaria que anteriormente han realizado nuestros mayores.

Para lograr este objetivo, inicialmente se planteó que los usuarios fueran capaces de realizar una serie de tareas como: registrarse y acceder a su propia cuenta en la aplicación, visualizar todas las recetas compartidas por los usuarios, añadir una nueva receta a la apliación, administrar sus recetas, añadir recetas a su lista de favoritos, entre otras funcionalidades.

En la tabla 6.1 se puede analizar, de forma más detallada, el grado de consecución de las diferentes funcionalidades planteadas en el comienzo del Trabajo de Fin de Grado presentado. En esta tabla se indican los objetivos planteados así como los resultados obtenidos y, en caso de que no se llegara a realizar alguno de ellos, se especifica el motivo por el cual no se desarrolló.

Tal y como se puede apreciar en la tabla 6.1, la mayoría de los objetivos planteados inicialmente se han logrado con éxito. Sin embargo, algunos de estos objetivos no han podido ser finalizado por motivos de tiempo, por lo que se han eliminado de la aplicación final en esta primera versión (las tareas relacionadas con estos objetivos no constituyen una parte fundamental de la aplicación).

Por lo tanto, se puede concluir que la aplicación *AppRecetas*, objeto del presente Trabajo de Fin de Grado, ha sido desarrollada con éxito, convirtiéndose así en una aplicación real que permitirá la consulta (y ampliación) de las recetas de la cocina canaria que anteriormente han realizado nuestros mayores, gracias a las funcionalidades logradas presentadas en la tabla 6.1, entre otras.

En cualquier caso, la consecución de los objetivos anteriormente planteados no ha sido una tarea sencilla. Desde el planteamiento del supuesto inicial hasta la obtención y ejecución de las soluciones, han surgido multitud de problemas de diversa dificultad

Objetivo	**Resultado**	**Motivo**
Uso de servicio de "Backend" en la nube para la gestión de cuentas, almacenamiento y procesado de datos	Logrado	-
Visualización de la lista de recetas catalogadas	Logrado	-
Visualizar una receta en concreto	Logrado	-
Capacidad para editar los datos personales de un usuario, cambiar imagen de perfil y darse de baja de la aplicación	Logrado	-
Capacidad para visualizar un video sobre la receta desde cualquier plataforma	Logrado	-
Capacidad de subir una nueva receta a la nube	Logrado	-
Capacidad para subir archivos a la nube desde la galería o toma de fotos	Logrado	-
Capacidad para añadir una receta a su lista de favoritos facilitando el acceso a la receta	Logrado	-
Posibilidad de editar archivos subidos por el usuario	Logrado	-
Posibilidad de subir mas de una foto por receta	No realizado	Se decidió eliminar de la app final por cuestiones de tiempo
Capacidad para filtrar las recetas de la lista	No Logrado	No se ha conseguido realizar por cuestiones de tiempo

Tabla 6.1: Grado de cumplimiento de los objetivos planteados en el Trabajo de Fin de Grado

donde encontrar la solución de alguno de ellos para alguien con limitada experiencia en el desarrollo de aplicaciones móviles, resultó ser complicado.

En esta línea hay que indicar que, en un primer momento, se planteó la utilización de *Firebase* como servicio en la nube para el almacenamiento de archivos y gestión de usuarios,servicio que casi al término del desarrollo anunció el cambio de políticas gratuitas y de pago de la plataforma. Con este nuevo cambio y sin actualizar la suscripción actual, sólo es posible realizar muy pocas consultas al *Storage* al día, complicando el correcto funcionamiento de la aplicación, puesto que sin el acceso a las fotos de las recetas, éstas no se muestran y por consiguiente, la aplicación no sería funcional.

Relacionado con este punto, una de las principales fuentes de esfuerzo fue el manejo de la API del servicio ofrecido por *Firebase*, pues implica estudiar su funcionamiento y el tratamiento de la misma respecto a los datos que se van a alojar en

dicha plataforma. Cabe destacar que, al ser servicios en la nube implica trabajar con servicios a nivel de red, por consiguiente, se debe tener en cuenta que es probable la aparición de posibles fallos de conexión, periodos de obtención de datos prolongados e incidencias externas al propio funcionamiento de la aplicación o del usuario.

Por otro lado, y si bien el funcionamiento a nivel de red podía estar controlado, el funcionamiento mismo de la aplicación en diferentes dispositivos generaba efectos indeseados o que diferían del planteado inicialmente. En ese sentido, en el dispositivo *Xiaomi Redmi Note 3* utilizado para las pruebas, se ocasionaban errores a la hora de mostrar los datos en la pantalla debido al doble marco que utiliza. Con un estudio y profundización mayor se pudo solucionar el problema mediante la prueba con diferentes tipos de dispositivos, de los cuales, en el momento del desarrollo del presente Trabajo de Fin de Grado, no se disponían.

Ante todos estos casos siempre se identificó el problema y, en la mayoría de ellos, se controló y solucionó, garantizando el correcto funcionamiento de la aplicación. Debido a la grana cantidad de fuentes de problemas tanto a nivel externo como interno (cómo la app responderá ante una gran cantidad de usuarios, variabilidad en el entorno de red, actualizaciones en el funcionamiento del servicio, variedad de dispositivos, entre otros), es posible que la aplicación no sea completamente funcional en todos los casos de uso existentes. En esta línea, se pretende que las posibles incidencias o errores detectados en el futuro puedan ser resueltas, garantizándose una actualización de la aplicación para todos sus usuarios, de modo que siempre esté operativa y se alcance su funcionamiento óptimo (generando nuevas versiones de la aplicación).

Finalmente, comentar que la aplicación está diseñada de tal forma que, en caso de querer expandirla a otra comunidad podría realizarse con una re-estructuración del diseño e incluyendo una capa "superior", en la que el usuario pudiera seleccionar entre diferentes Comunidades Autónomas y, una vez seleccionado, se accediera a aplicación actual, con los datos e información actualizadas a la comunidad seleccionada.

6.2 Mejoras

Si bien la aplicación cumple con la práctica totalidad de los objetivos planteados inicialmente, es cierto que se pueden realizar mejoras, con el objetivo de aumentar sus prestaciones. Algunas de las posible mejoras a realizar se proponen a continuación:

- Incluir la posibilidad de "loguearse" y registrarse en la aplicación mediante las cuentas de *Facebook*, *Google* o *Twitter*.
- Dotar a la aplicación de un filtro de recetas funcional.

- Añadir una categoría para las recetas aptas para celíacos.
- Permitir a los usuarios establecer algún tipo de comunicación y seguimiento con el resto de usuarios de la aplicación, incluyendo la posibilidad de observar la imagen de perfil del resto de usuarios.
- Introducir el envío de notificaciones periódicas sobre la agregación de nuevas recetas a la base de datos, sobre todo si la receta es subida por un usuario al que sigue.
- Añadir la posibilidad de subir más de una imagen por receta.

En cuanto al diseño también se pueden realizar mejoras en el aspecto visual incluso en la optimización del código. Por supuesto, éstas y otras mejoras se pueden llevar a cabo, con la ayuda de expertos en el diseño gráfico, con tiempo y experiencia en el diseño y elaboración de aplicaciones.

Como idea final, se podría utilizar la aplicación como un apoyo para la promoción turística de una comunidad autónoma específica o incluso de un país entero potenciando el turismo gastronómico.

BIBLIOGRAFÍA

[1] *Encuesta de jóvenes de Canarias 2012: diagnóstico de la situación de la juventud de Canarias*. Consejería de Presidencia, Justicia e Igualdad, Gobierno de Canarias.
Disponible en: http://www.juventudcanaria.com/juventudcanaria/programas/publicaciones/.
Consultada diciembre 2017.

[2] *Sitio Web de statista*.
Disponible en: https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/.
Consultada diciembre 2017.

[3] *Gartner Says Five of top ten Worldwide mobile phone vendors increased Sales in second quarter of 2016*.
Disponible en: http://www.gartner.com/newsroom/id/3415117.
Consultada diciembre 2017.

[4] *Sitio Web de Android*.
Disponible en: http://www.android.com/.
Consultada diciembre 2017.

[5] *Sitio Web de iOS*.
Disponible en: http://www.apple.com/es/ios/.
Consultada diciembre 2017.

[6] *Sitio Web de Java*.
Disponible en: https://www.java.com/es/.
Consultada diciembre 2017.

[7] *Sitio Web de Objetive-C*.
Disponible en: https://developer.apple.com/reference/objectivec.
Consultada diciembre 2017.

[8] *Sitio Web de Swift*.
Disponible en: https://developer.apple.com/swift/.
Consultada diciembre 2017.

[9] *Sitio Web de Google Play*.
Disponible en: https://play.google.com/.
Consultada diciembre 2017.

[10] *Sitio Web de App Store.*
Disponible en: http://www.apple.com/es/.
Consultada diciembre 2017.

[11] *Sitio Web de la Aplicación "Recetas de cocina gratis".*
Disponible en: https://play.google.com/store/apps/details?id=com.recetasgratisnet.recetasdecocina.
Consultada diciembre 2017.

[12] *Sitio Web de la Aplicación "Recetas de todo gratis".*
Disponible en: https://play.google.com/store/apps/details?id=com.cookware.worldcusinerecipes.
Consultada diciembre 2017.

[13] *Sitio Web de la Aplicación "Cookpad recetas".*
Disponible en: https://itunes.apple.com/es/app/cookpad-recetas-la-cocina/id585332633?mt=8.
Consultada diciembre 2017.

[14] *Agencia estatal; Boletín oficial del Estado. Ley orgánica 15/ 1999, de 13 de Diciembre, de Protección de Datos de Carácter Personal*
. Disponible en: http://www.boe.es/buscar/act.php?id=BOE-A-1999-23750.
Consultada diciembre 2017.

[15] Nielsen J., *10 Usability Heuristics for User Interface Design.*
Disponible en: http://www.nngroup.com/articles/ten-usability-heuristics/.
Consultada diciembre 2017.

[16] *Modelo Objectory.*
Disponible en: http://metodologiasoo.wikispaces.com/Objectory,+por+Ivar+Jacobson
Consultada diciembre 2017.

[17] Alfredo Weitzenfield,
Ingeniería de software orientada a objeto con UML, Java e Internet.
Editorial Thomson, 2005.

[18] Megali Tin, *An Introduction to Model View Presenter on Android.* Marzo 2016.
Disponible en: https://code.tutsplus.com/tutorials/an-introduction-to-model-view-presenter-on-android–cms-26162.
Consultada diciembre 2017.

[19] *Sitio Web sobre los diagramas.*
Disponible en: https://ingsotfwarekarlacevallos.wordpress.com/2015/07/07/uml-diagrama-de-secuencia/.
Consultada diciembre 2017.

[20] *Sitio Web de Draw.io.*
Disponible en: https://www.draw.io/.
Consultada diciembre 2017.

[21] *Sitio Web de Firebase.*
Disponible en: https://firebase.google.com/?hl=es.
Consultada diciembre 2017.

[22] *Sitio Web de Firebase Authentication.*
Disponible en: https://firebase.google.com/docs/auth/users?hl=es-419.
Consultada diciembre 2017.

[23] *Sitio Web de Balsamiq Mockups.*
Disponible en: https://balsamiq.com/products/mockups/.
Consultada diciembre 2017.

Parte II

Pliego de condiciones

PLIEGO DE CONDICIONES

Introducción

A continuación se especificarán los requisitos técnicos que permitirán la ejecución de la aplicación en terminales móviles Android. Para ello, se establecerán las diferentes condiciones referentes al hardware y al software, así como la instalación de la aplicación. Por último, se expondrá la licencia de uso del software.

Requisitos

El hardware requerido para el correcto funcionamiento de la aplicación consiste en un terminal *Android* (*smartphone* o tableta) con la versión 5.0 o superior de su sistema operativo, conexión a Internet y una pantalla superior a 4.5 pulgadas. En este sentido, la aplicación ha sido probada en los siguientes dispositivos:

- Teléfono móvil Xiaomi Redmi Note 3.
- Tableta Samsung Galaxy Tab A.

Para los cuales se garantiza (a través de las pruebas) su correcto funcionamiento.

Los distintos fabricantes de terminales móviles *Android* especifican, a través de las hojas de características, la versión del sistema operativo soportado. Asimismo, el terminal también lo indica, típicamente accediendo a: *Ajustes* → *Acerca del teléfono* → *Info de software* (o similar).

Para el funcionamiento de la parte del servidor de la aplicación, se ha utilizado *Firebase*. Dentro de esta plataforma fueron utilizados tres servicios concretos: *Firebase Authentication* para la gestión de las cuentas de usuario, *Firebase Database* para el almacenamiento de la información en bases de datos, y *Firebase Storage* para el alojamiento de imágenes. Estos servicios corresponden con una parte fundamental para el manejo de los datos de interés de la aplicación, sin los cuales se perdería la práctica totalidad de su funcionamiento. No obstante, la aplicación está desarrollada de tal forma que, en caso de necesitar en un futuro un cambio de servidor o servicio de *backend*, esto se pueda realizar con la mínima modificación del código de la aplicación.

Condiciones de instalación

Para la instalación de la aplicación desarrollada, el archivo a instalar se encuentra en la entrada *Código* del menú del disco compacto adjunto a esta memoria. Así, se procede a copiar el archivo *AppRecetas.apk* en la memoria externa (*sdcard* o memoria SD) en una ubicación cualquiera del terminal (se recomienda la carpeta *Aplicaciones* o similar). La copia puede llevarse a cabo mediante USB, *Bluetooth* o *Wi-Fi*. Otra forma más sencilla de recuperar el archivo para su instalación consiste en enviarla por email o accediendo a un enlace en la nube (donde esté almacenado el archivo *apk*). Asimismo, es necesario tener activada la opción de poder instalar aplicaciones que no se hayan descargado desde el *Google Play*. Para ello, hay que activar la opción que hay en *Ajustes* → *Pantalla bloque y seguridad* → *Fuentes desconocidas* (o similar).

Una vez copiado el archivo *apk* en la memoria externa del terminal móvil (primera opción), se procede a la instalación. Para ello es necesario que el usuario haya descargado e instalado previamente, a través del *Google Play* de *Android*, cualquier explorador de archivos, tal como *ES Explorador de Archivos* de *File Manager*, entre otros. Así, se ejecuta el explorador de archivos en el terminal móvil y se entra en la carpeta donde fue transferido el archivo *AppRecetas.apk*. Al seleccionar el archivo ha de salir un menú contextual en el que se sugieren varias opciones. Entre ellas, *Instalar*. Si se elige *Instalar*, aparecerá una ventana que explica al usuario los permisos que concede a la aplicación. Si el usuario está de acuerdo, se elige *Aceptar*, con lo que la instalación queda realizada. En el caso de haber utilizado la segunda opción, una vez descargado el archivo por email o mediante un enlace, se selecciona *Instalar*, apareciendo la ventana indicada anteriormente.

Una vez instalada la aplicación, se ha de acceder al listado de aplicaciones del terminal, donde la aplicación objeto de este trabajo vendrá representada por un icono característico (figura 6.1). Mediante la selección del icono será ejecutada la aplicación.

Figura 6.1: Icono de la aplicación *AppRecetas*

Licencia del programa

Normas generales

De conformidad con lo dispuesto en la normativa reguladora en materia de Propiedad Intelectual, el TFT se considera una obra en colaboración entre el estudiante y la tutora. Para la explotación industrial del TFT será de aplicación lo establecido en los Estatutos de la Universidad de Las Palmas de Gran Canaria.

Asimismo, el uso de esta aplicación ha de ser bajo la expresa autorización del autor o de la tutora del trabajo o de la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria.

Derechos de autor

La aplicación y la documentación están protegidos por la ley de Propiedad Intelectual aplicable, así como por las disposiciones de los tratados internacionales. En consecuencia, el usuario podrá usar copia de esta aplicación, así como del código fuente de programación y de la documentación, siempre bajo la autorización del autor o de la tutora del trabajo o de la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria.

Garantía

Se garantiza el correcto funcionamiento de la aplicación en el momento de la instalación de acuerdo con las especificaciones vistas con anterioridad. La instalación de la aplicación no ocasionará la aparición de defectos en los dispositivos que cumplan con las especificaciones técnicas. En este sentido, la aplicación se encuentra, en el momento de la redacción, en su versión inicial (1.0). Esto significa que a medida que se vayan descubriendo posibles incidencias o errores, estos podrán ser detectados y solucionados, llevándose a cabo actualizaciones de la aplicación para garantizar su correcto y óptimo funcionamiento.

Con la única excepción de lo expresamente expuesto en el párrafo anterior, la aplicación ha sido desarrollada sin garantías de ninguna clase. El autor no asegura, garantiza o realiza ninguna declaración respecto al uso o los resultados derivados de la utilización del programa o de la documentación. Tampoco se garantiza que la operación del programa sea interrumpida o sin errores. En ningún caso serán el autor, tutora o la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria responsables de los perjuicios directos, indirectos, incidentales, ejemplarios o consiguientes gastos, lucro cesante, pérdida de capital, interrupción de negocios, pérdida de información comercial o de negocio o

cualquier otra pérdida que resulte del uso o de la incapacidad de usar la aplicación o la documentación. El usuario conoce y acepta que los derechos de licencia reflejan esta asignación de riesgo como el resto de cláusulas y restricciones.

Otras consideraciones

La fiabilidad de operación de la aplicación puede estar afectada por factores adversos, a los que se denominan "fallas del sistema". En estos se incluyen errores en el funcionamiento del hardware del dispositivo, sistema operativo o entorno del mismo, compiladores o software de desarrollo usado para realizar la aplicación, software externo para el funcionamiento de la misma, errores de instalación, problemas de compatibilidad del software y hardware, fallos o funcionamiento incorrectos de equipos de control, fallas por uso, errores por parte del usuario de la aplicación o problemas en el acceso de Internet. En el supuesto de que cualquier disposición de esta licencia sea declarada total o parcialmente inválida, la cláusula afectada será modificada convenientemente de manera que sea ejecutable y una vez modificada, plenamente eficaz, permaneciendo el resto de este contrato en plena vigencia. Esta licencia se regirá por las leyes de España. El usuario o licenciatario acepta la jurisdicción exclusiva de los tribunales de este país en relación con cualquier disputa que pudiera derivarse de la presente licencia.

Parte III

Presupuesto

PRESUPUESTO

Introducción

El COITT/AEGITT (Colegio Oficial de Ingenieros Técnicos de Telecomunicación y La Asociación Española de Graduados e Ingenieros Técnicos de Telecomunicación) informa, en su página web sobre orientación al libre ejercicio[1], que en base a la nota remitida a todos los colegios profesionales por el Ministerio de Economía y Hacienda y siguiendo directivas europeas, se deben eliminar los baremos orientativos de honorarios. Esto significa que los honorarios se establecen como un libre acuerdo entre el profesional y el cliente. A continuación se detalla la forma en la que el autor de este Trabajo Fin de Grado establece la baremación del proyecto presentado.

Así, el presupuesto presentado se divide en las siguientes partes:

- Tarifa de honorarios por tiempo empleado.
- Amortización de los equipos empleados.
  - Amortización del material hardware.
  - Amortización del material software.
- Coste de acceso a Internet.
- Redacción de la documentación.
- Derechos de visado.
- Gastos de tramitación y envío.

En las siguientes secciones se analizará, en detalle, cada una de estas partes.

[1] http://www.coitt.es/index.php?page=libre_docs&xcod=4&mcod=48

Tarifa de honorarios por tiempo empleado

Este concepto contabiliza los gastos correspondientes a la mano de obra. Para realizar este cálculo, se propone la siguiente fórmula:

$$H = (14,48 \times Hn) + (20,27 \times He)$$

Siendo:

- **H**: honorarios.
- **Hn**: honorarios en jornada laboral normal.
- **He**: honorarios fuera de la jornada laboral normal.

En el trabajo presentado en esta memoria, se ha empleado un periodo de aproximadamente **7** meses (ya que no se pudo entregar el trabajo final en la Convocatoria Extraordinaria), trabajando alrededor de **5** horas diarias, en horario laboral normal (300 horas totales).

Distribución de la temporización del trabajo

El trabajo se divide en cuatro etapas bien diferenciadas, que se pasan a definir:

Definición

Esta etapa está dedicada a la definición de los requisitos de la aplicación, es decir, ¿qué problema se quiere resolver? Para realizar este proceso hay que implementar un prototipo de la aplicación y probar con usuarios reales para detectar las debilidades y fortalezas del diseño planteado.

Diseño

En esta etapa se estudian cada una de los módulos de la aplicación. A medida que se desarrolle esta parte debe distinguirse la actividad que desempeña cada módulo y la interrelación de los mismos (mensajes que se envían). Asimismo, se especifica cómo será el modelo de datos y dónde se debe almacenar la información.

Codificación

En esta etapa se realiza el desarrollo de la aplicación, es decir, se traduce lo establecido en el diseño a código, concretizando desde la generalidad a la particularidad, en función del lenguaje elegido (en este caso, Java para Android).

Verificación

En esta etapa, el desarrollador verifica que el código realizado cumple los requisitos establecidos en el primer punto. En este Trabajo Fin de Grado se verifica, exclusivamente, parte del modelo de datos, que es la parte más crítica de la aplicación. Dentro de la parte de verificación se ha incluido la etapa de *Control y Seguimiento* del Trabajo Fin de Grado, que corresponde con la última tarea del anteproyecto presentado.

Redacción

En esta etapa se lleva a cabo la redacción de la documentación y su posterior revisión. En el caso de este Trabajo Fin de Grado, esta etapa se extiende a lo largo del tiempo dedicado al trabajo, correspondiendo a los entregables definidos en su día en el diagrama de Gantt presentado en el anteproyecto.

Cálculo de tarifa de honorarios por tiempo empleado

En la tabla 6.1 se detalla el tiempo empleado para cada una de las etapas anteriormente descritas y se realiza el cálculo de los honorarios.

Etapa	**Horas laborales**	**Honorarios**
Definición	52	752,96 €
Diseño	57	825,36 €
Codificación	88	1274,24 €
Verificación	30	434,40 €
Redacción	73	1057,04 €
Total		**4344 €**

Tabla 6.1: Honorarios por tiempo empleado

Por lo que sumadas cada una de las etapas, el cálculo de honorarios por tiempo empleado asciende a CUATRO MIL TRESCIENTOS CUARENTA Y CUATRO EUROS.

Amortización de los equipos empleados

Dentro de este concepto se considera tanto la amortización del hardware como del software empleado en la realización del trabajo presentado. De este modo, se estipula el coste de amortización para un período de 3 años, utilizando un sistema de amortización lineal o constante. En este sistema, se supone que el inmovilizado material se deprecia de forma constante a lo largo de su vida útil. La cuota de amortización anual se calcula haciendo uso de la siguiente fórmula:

$$C = \frac{Vad - V}{N}$$

Donde:

- **C**: cuota de amortización anual.
- **Vad**: valor de la adquisición.
- **V**: valor residual.
- **N**: número de años de vida útil de la adquisición.

Siendo el valor residual el valor teórico que se supone tendrá el elemento en cuestión después de su vida útil, teniendo en cuenta los índices de depreciación actual. En el caso del hardware y del software son 3 años (al 33 % de depreciación máximo por año).

Amortización del material hardware

Debido a que el trabajo se ha elaborado en un periodo inferior a 3 años, que es el periodo en que se calcula la amortización de material hardware, se realizará una amortización equiparable al período de duración del mismo (300 horas en 7 meses). Según esto, se obtienen los gastos expuestos en la tabla 6.2.

Por lo tanto, el coste total de hardware asciende a la cantidad de CUATROCIENTOS NOVENTA Y SEIS EUROS CON SETENTA Y CINCO CÉNTIMOS.

Amortización del material software

El coste total del software utilizado es la suma de las herramientas software empleadas para la realización del trabajo (tabla 6.3). De esta manera se describe cada uno de las herramientas utilizadas y el coste supuesto para su utilización usando la fórmula de amortización anteriormente citada.

Por tanto, el coste total de software asciende a CUARENTA Y CUATRO EUROS CON CINCUENTA Y SIETE CÉNTIMOS.

Descripción	Unid.	Valor de adquisición	Tiempo de uso	Coste anual	Total
Ordenador portátil Toshiba Satellite S50, Intel Core i7 @ 2.4 GHz, 8 Gb de RAM	1	1100,00 €	7 meses	641,66 €	374,30 €
Teléfono móvil Xiaomi Redmi Note 3	1	160,00 €	7 meses	93,33 €	54,40 €
Tableta Samsung Tablet A	1	200,00 €	7 meses	116,60 €	68,05 €
Total (€)					**496,75**

Tabla 6.2: Precios y costes de amortización del hardware

Descripción	Valor de adquisición	Tiempo de uso	Coste anual	Total
Android Studio	0,00 €	7 meses	0,00 €	0,00 €
SDK y ADT Android	0,00 €	7 meses	0,00 €	0,00 €
Balsamiq Mockups (para desarrollo de muestras de la aplicación en formato pdf)	62 €	7 meses	36.16 €	21,09 €
Microsoft Office 365 Personal	69,00 €	7 meses	40,25 €	23,48 €
Total (€)				**44,57**

Tabla 6.3: Precios y costes de amortización del software

Coste de acceso a Internet

Para la conexión a Internet se dispone de una solución de fibra óptica a 300Mbps cuyo coste mensual es de 56,68 euros. Debido a que se ha usado dicha conexión durante cada una de las etapas de creación del mismo (7 meses), el coste de acceso a Internet se traduce a un total de TRESCIENTOS NOVENTA Y SEIS EUROS CON SETENTA Y SEIS CÉNTIMOS.

Redacción de la documentación

Al coste de redacción obtenido hasta el momento se le deben añadir otros gastos, quedando el importe final de redacción del trabajo como se describe en la tabla 6.4.

Por lo tanto, la redacción del trabajo asciende a un total de MIL OCHENTA EUROS CON OCHENTA Y CUATRO CÉNTIMOS.

Concepto	**Coste**
Redacción del trabajo	1.057,04 €
Coste de impresión con tinta color	15,80 €
Encuadernación	5,00 €
CDs de 700MB	3,00 €
Total	**1.080,84** €

Tabla 6.4: Coste final de redacción del trabajo

Presupuesto antes de impuestos

Sumando todos los conceptos calculados hasta el momento, se obtiene el presupuesto, sin incluir los impuestos, que se muestra en la tabla 6.5.

Concepto	**Coste**
Tarifa de honorarios por tiempo empleado	4.344 €
Amortización del material hardware	496,75 €
Amortización del material software	44,57 €
Coste de acceso a Internet	396,76 €
Gastos adicionales de redacción	23,80 €
Total	**5.305,88** €

Tabla 6.5: Presupuesto total sin impuestos

El presupuesto calculado, antes de incluir los impuestos, asciende a CINCO MIL TRESCIENTOS CINCO EUROS CON OCHENTA Y OCHO CÉNTIMOS.

Presupuesto incluyendo impuestos

Al presupuesto calculado anteriormente hay que incluirle un 7 % de IGIC obteniendo el coste del presupuesto final (tabla 6.6).

Por tanto, el presupuesto total, incluyendo impuestos, asciende a la cantidad de CINCO MIL SEISCIENTOS SETENTA Y SIETE EUROS CON VEINTINUEVE CÉNTIMOS.

Concepto	**Coste**
Total (sin IGIC)	5.305,88 €
IGIC (7 %)	371,41 €
Total	**5.677,29** €

Tabla 6.6: Presupuesto total

Las Palmas de Gran Canaria, a 4 de diciembre de 2017.

Fdo: Oscar Medina Morillo