UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una plataforma social iOS para compartir recetas de cocina

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Alberto Hernández Pellicer

Tutor: Pedro José Valderas Aranda

Curso 2018-2019

Resum

L'objectiu principal d'este projecte és el de crear una aplicació amb la intenció de crear una xarxa social amb la idea de poder compartir receptes pròpies i recrear plats realitzats per altres persones de la comunitat.

Paraules clau: Aplicació mòbil, Firebase, MVC, iOS, receptes de cuina

Resumen

El objetivo principal de este proyecto es el de crear una aplicación con la intención de crear una red social con la idea de poder compartir recetas propias y recrear platos realizados por otras personas de la comunidad.

Palabras clave: Aplicación móvil, Firebase, MVC, iOS, recetas de cocina

Abstract

The main objective of this project is to create an application with the intention of creating a social network with the idea of being able to share their own recipes and recreate recipes made by other people in the community.

Key words: Mobile application, Firebase, MVC, iOS, cooking recipes

Índice general

CAPÍTULO 1
Introducción

1.1 Motivación

Hoy en día existen múltiples personas con interés en la elaboración de recetas que no tienen las adecuadas herramientas para poder realizarlas. Por otra parte, existen chefs que ya saben cocinar y que les gustaría compartir sus platos con otras personas.

Ante estas necesidades nace la idea de Cookit, una aplicación pensada para todos los amantes de la cocina, un lugar donde los mas experimentados pueden compartir sus recetas y donde cualquier persona puede aprender a cocinar deliciosos manjares.

1.2 Objetivos

El objetivo principal de este proyecto es el de crear una aplicación iOS con la idea de poder compartir recetas propias y recrear platos realizados por otras personas de la comunidad.

Los objetivos que se pretenden cumplir con el proyecto son:

Generales

- Demostrar la capacidad del alumno para desarrollar un proyecto completo.
- Aplicar los conocimientos aprendidos en algunas asignaturas de la carrera como *Ingeniería del Software, Bases de datos, Algorítmica...*
- Aprender un nuevo lenguaje de manera autodidacta (Swift) partiendo de los lenguajes ya conocidos y aprendidos en la titulación.
- Aprender el uso y funcionamiento de bases de datos no relacionales con Firebase.

Específicos

- Proporcionar una búsqueda por categorías y distintos parámetros de filtrado para buscar recetas.
- Posibilidad de CRUD (crear, leer, actualizar y borrar) de las recetas.
- Guardar las recetas favoritas del usuario así como poder valolarlas y ver cuales son las más populares.
- Manejar información bidireccionalmente entre aplicación y base de datos en formato JSON.
- Adaptabilidad de los elementos de la interfaz a los distinos tipos de pantalla que tienen los dispositivos iOS.
- Seguir unas líneas de diseño modernas, en este caso las que ofrece Apple, que den como resultado una interfaz usable e intuitiva.

1.3 Estructura de la memoria

Esta memoria está dividida en múltiples capítulos, en los que se cubrirán distintos aspectos en relación al trabajo realizado.

El primer capitulo es esta presente introducción la cual ha sido dividida en tres apartados. El primero de ellos es una breve pero concisa motivación donde explicamos el porqué nace este proyecto y como afecta a la sociedad. En el segundo veremos cuáles son los objetivos generales y específicos que hemos abarcado. El último apartado es esta misma estructura de la memoria donde veremos cuales son los capítulos que la conforman así como una breve explicación de los mismos.

En el segundo capítulo estudiaremos y analizaremos las aplicaciones o sistemas existentes más utilizados, viendo cuáles son sus virtudes y defectos. Por otro lado, veremos unas breves conclusiones, las cuales se han obtenido en base a este estudio previo. Además, examinaremos qué ofrece nuestra aplicación para rivalizar ante éstas.

En el tercer capítulo veremos el contexto tecnológico, donde nos adentraremos en las tecnologías y herramientas utilizadas para llevar a cabo nuestro sistema.

Seguidamente veremos la metodología que se ha empleado a la hora del desarrollo, así como las ventajas e inconvenientes que tiene.

El quinto capítulo trata sobre el análisis de los requisitos. En él podremos ver los pasos que se han seguido para recabar toda la información y funcionalidades que incorporaría nuestra aplicación. Además, veremos los casos de uso así como una explicación de cada uno de ellos.

A continuación vendrá el capítulo del diseño. Comenzaremos viendo la sección del diagrama UML, el cual nos dará una visión global de nuestro sistema. Seguidamente, veremos y compararemos el resultado de los primeros mockups ante el diseño que ha tenido la aplicación final. En último lugar veremos una sección sobre como es el flujo de la vista y como podemos ir de unas pantallas a otras.

En el séptimo capítulo estudiaremos la arquitectura de nuestro sistema desde tres puntos de vista. El primero será externamente, donde veremos como nos comunicamos con los servicios externos y como es el intercambio de información. También veremos como está dividida nuestra aplicación, los distintos servicios internos que se han definido

y como interactúan unos con otros. Por concluir con este capitulo veremos el diseño de la base de datos que hemos creado.

El octavo capítulo hace referencia a la implementación. Lo primero que haremos es analizar la estructura de nuestro sistema de archivos, es decir, como hemos organizado las diferentes piezas que conforman la aplicación. Posteriormente, entraremos en detalles de código de cada una de estas partes, donde veremos algunos ejemplos de implementación que nos han parecido más interesantes.

Ya por concluir, veremos el noveno y último capítulo de esta memoria, donde os hablaré de las conclusiones que he extraído de este proyecto. Además os contaré mi experiencia personal en este desarrollo, así como las dificultades que he tenido y las futuras mejoras que se podrían realizar.

CAPÍTULO 2
Estado del arte

En este capítulo se van a presentar las aplicaciones existentes más utilizadas y semejantes a la realizada en este proyecto. Estas aplicaciones se han tomado como referencia para ver cuales son sus puntos fuertes, analizar sus flaquezas y realizar una comparativa de qué novedades aporta nuestra aplicación ante estas. Vamos a ver una por una cuales son estas aplicaciones las cuales han sido puestas en orden ascendente en base a la puntuación y aparición que tienen en el App Store.

2.1 Kitchen Stories

Esta es la aplicación por referencia en recetas de cocina en todo el App Store. Tiene una interfaz muy cuidada. Dispone de interesantes funcionalidades como poder filtrar por tipos de cocina (china, europea, italiana...). Además, también cuenta con un buscador de recetas, lo que permite al usuario buscar platos conociendo su nombre.

Dejando a un lado las bondades que ofrece nos hemos fijado que tiene algunas carencias, por ejemplo, un usuario no puede subir sus propias recetas, por lo que está limitado a recrear las recetas ya existentes. Por otro lado, los chefs no tienen un perfil público ni se les hacen mención en las propias recetas, lo que provoca que el usuario final no pueda ver más platos realizados por ese chef a pesar de que le pueda agradar su forma de cocinar.

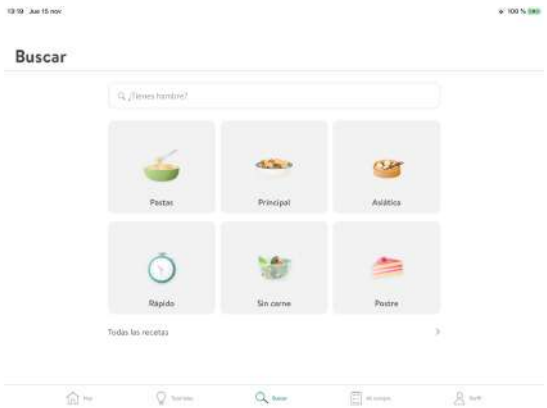

Figura 2.1: Menú para buscar recetas en base a su categoría

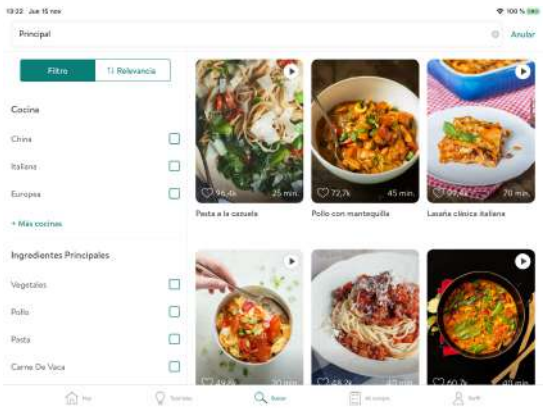

Figura 2.2: Menú de filtrado para los platos

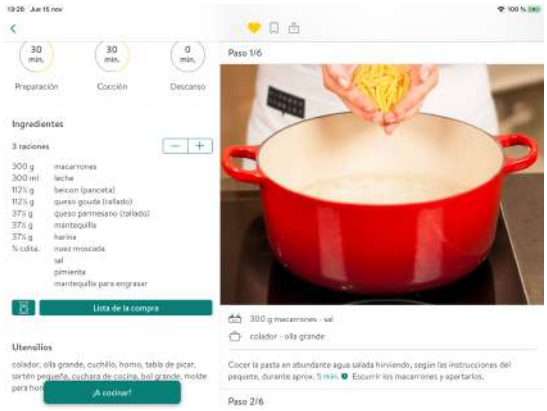

Figura 2.3: Elaboración de la receta

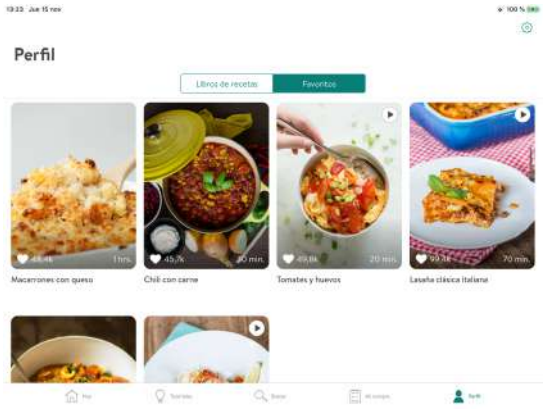

Figura 2.4: Menú de favoritos

2.2 Hatcook

Hatcook es la segunda aplicación que vamos a analizar. Como podemos comprobar el aspecto estético no está tan cuidado, cosa que no se puede pasar por alto cuando estamos compitiendo con otras aplicaciones que si lo tienen muy en cuenta. Respecto al rendimiento deja mucho que desear. La aplicación se cierra en muchos momentos por lo que poder utilizarla se hace realmente complicado.

Dejando a un lado el tema estético y de rendimiento podemos fijarnos que esta aplicación cuenta con un sistema donde los usuarios pueden subir sus propias recetas. Esta funcionalidad nos ha gustado gratamente debido a su sencillez, aunque tiene algunas carencias como no poder realizar fotografías desde la aplicación, por lo que obliga al usuario a tener que ir a la aplicación de cámara, realizar la fotografía, volver a la aplicación para importar dicha imagen desde la galería... En conclusión, la usabilidad en este aspecto es pésima.

Figura 2.5: Menú 1 para crear una nueva receta

Figura 2.6: Menú 2 para crear una nueva receta

Por concluir también echamos en falta un apartado para buscar los platos más populares o mejor valorados en la aplicación, funcionalidad que nos parece primordial en este tipo de aplicaciones.

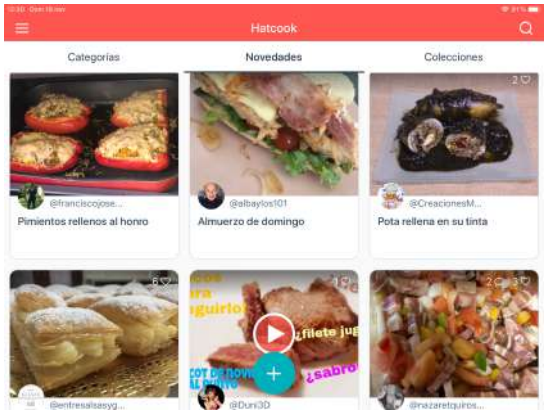

Figura 2.7: Menú de novedades

Figura 2.8: Menú de filtrado por ingredientes

2.3 Cookpad

Cookpad es la tercera aplicación que vamos a estudiar. Una novedad ante las otras aplicaciones ya analizadas es que permite a los usuarios subir sus propias recetas así como guardarlas en favoritas. Además, los usuarios disponen de un perfil público donde pueden añadir una breve información como un correo electrónico, un teléfono...

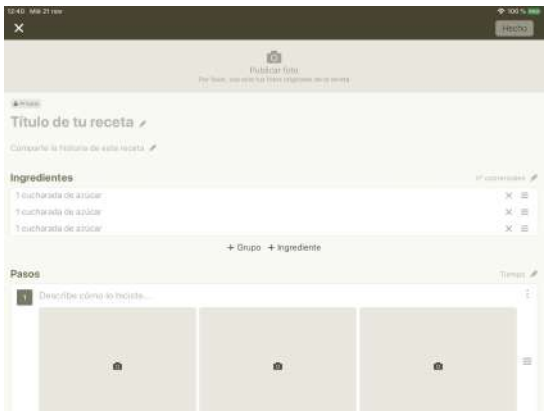

Figura 2.9: Menú para crear una nueva receta

Figura 2.10: Perfil público de un usuario

En contra, no hay ningún sistema de valoración de recetas por lo que no es posible llevar un listado de las mejores hasta la fecha. Asimismo, los platos no están categorizados por lo que resulta complicado encontrar un plato nuevo si no sabes cual es su nombre.

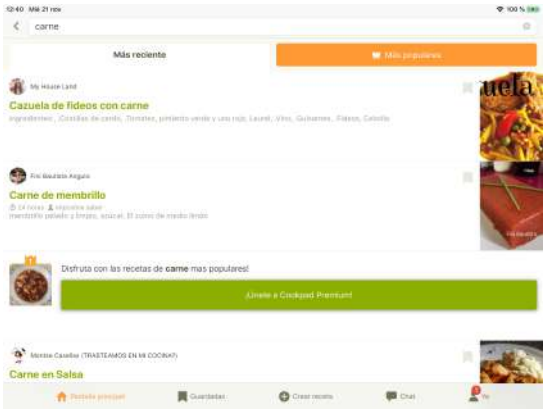

Figura 2.11: Sistema de filtrado

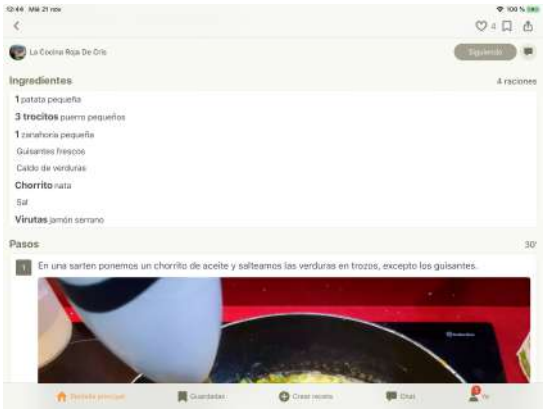

Figura 2.12: Elaboración de una receta

2.4 Petit Chef

En cuarto lugar tenemos Petit Chef. Un aspecto que nos ha gustado es que podemos puntuar y evaluar las recetas. Además, en el menú inicial disponemos de un listado de las últimas recetas que han sido creadas lo que facilita el descubrimiento diario de nuevos platos. También podemos marcar una receta como favorita para poder tenerla a mano más fácilmente.

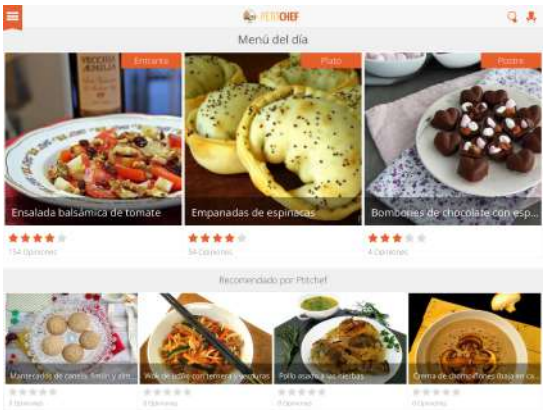

Figura 2.13: Menú del día

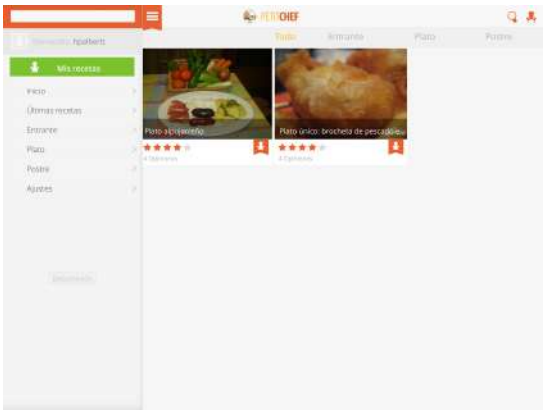

Figura 2.14: Recetas favoritas

Por el contrario, una carencia que tiene esta app es que los usuarios no pueden subir sus propias recetas, además de que el sistema de filtrado es muy básico, limitándose solo a buscar por los nombre de las recetas.

Figura 2.15: Elaboración 1 de una receta

Figura 2.16: Elaboración 2 de una receta

2.5 Conclusiones

Al analizar todas estas aplicaciones nos hemos dado cuenta de que la mayoría son muy parecidas en cuanto a funcionalidades básicas, aunque algunas de ellas si que se han atrevido a incorporar algunas características nuevas para intentar marcar la diferencia.

Después de analizar todas estas apps, hemos llegado a la conclusión de que la mejor forma de realizar la nuestra para que esté a la altura de las actuales es incorporar características interesantes que puedan demandar los usuarios. Es por ello que se han realizado unas entrevistas, las cuales se hablarán de ellas en el quinto capítulo, donde se les ha preguntado a diferentes personas por las funcionalidades que les gustaría que tuviera nuestra aplicación.

En base a este estudio previo, se han seleccionado las características más interesantes para comenzar desarrollar el mínimo producto viable de nuestra aplicación.

A continuación, se muestra una tabla de comparación con las características que disponen cada una de las aplicaciones ya estudiadas en frente de la desarrollada en este proyecto.

Características	Kitchen Stories	Hatcook	Cookpad	Petit Chef	Cookit
Interfaz adaptada	Si	Si	Si	Si	Si
Descubrimiento diario	No	Si	Si	Si	Si
Guardar platos fav.	Si	Si	Si	Si	Si
Subir nuevas recetas	No	Si	Si	No	Si
Puntuar recetas	No	No	Si	Si	Si
Ranking recetas	No	No	Si	Si	Si
Recetas por chefs	Si	No	No	Si	Si
Login usuarios	Si	Si	Si	Si	Si
Perfiles públicos	No	No	Si	No	Si

CAPÍTULO 3
Contexto tecnológico

En este capítulo vamos a conocer las tecnologías y herramientas sobre las cuales se ha desarrollado la aplicación. De esta manera, podremos conocer cada uno de los engranajes que han hecho posible este desarrollo.

3.1 Elección de la tecnología

Cuando nos planteamos lanzar una aplicación móvil llega un momento de vital importancia el cual puede decidir el éxito o fracaso de la misma. Se trata de la decisión de como va a estar construida técnicamente. De forma general podemos separar las aplicaciones en 3 categorías: aplicaciones nativas, híbridas y web.

A continuación vamos a ver un gráfico a modo de esquema de las principales singularidades de cada una de las tecnologías de las que vamos a proceder a hablar posteriormente.

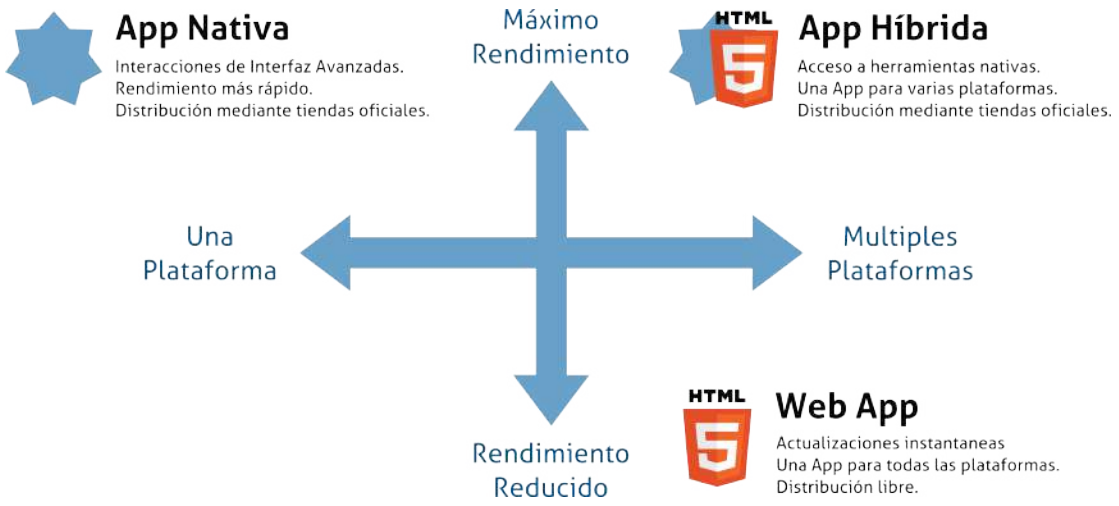

Figura 3.1: Tipos de tecnologías al construir una App [3]

3.1.1. Aplicaciones nativas

Una aplicación nativa [1] es una app desarrollada en el lenguaje específico para esa plataforma. Es decir, si queremos que nuestra app funcione tanto en iOS como en Android deberíamos de desarrollar dos aplicaciones, lo que conlleva que el código no sea reutilizable y el costo de desarrollo sea muy elevado.

Sin embargo, la principal fuente de poder de estas aplicaciones es su espléndido rendimiento. Esto se debe a que este tipo de aplicaciones se adapta al completo con las funcionalidades y características del dispositivo, obteniendo así, una mejor experiencia de uso y permitiendo a su vez aprovechar al máximo el poder de los procesadores gráficos que llevan en su interior.

3.1.2. Aplicaciones híbridas

Mientras que con las nativas estamos obligados a usar los lenguajes específicos de cada plataforma, con las híbridas podemos escribir el código solamente una vez, y ejecutarlo en todos los sistemas operativos en los que queramos lanzar nuestra app. Es decir, es altamente reutilizable.

Una aplicación híbrida [2] se distingue de la nativa por las tecnologías que emplean para construir la interfaz de usuario. El uso de esta tecnología nos deja realizar llamadas a librerías propias de cada sistema, no obstante construye mediante HTML la interfaz de usuario, como si fuera una página web y sobre una ventana que es un navegador web camuflado.

El principal problema de estas apps reside en la necesidad de consumir más recursos del dispositivo para poner la interfaz en una web, además de que no se adaptan ni al lenguaje de diseño del sistema ni a su usabilidad, por lo que generalmente son más lentas. Por otro lado, al tener la interfaz basada en un intérprete de HTML, CSS y Javascript, tiene más riesgo de contener fallos de seguridad.

En la imagen que se presenta a continuación se puede observar de forma más visual el desarrollo nativo en oposición al híbrido.

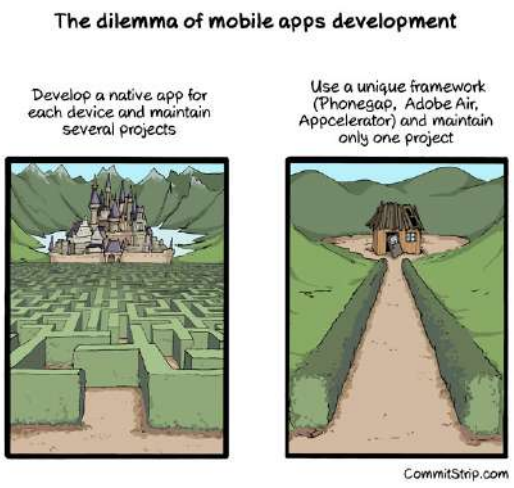

Figura 3.2: App nativa vs una híbrida [2]

3.1.3. Aplicaciones web

Una aplicación web para móvil consiste en el desarrollo de páginas web que son optimizadas para ser visualizadas y utilizadas en dichos dispositivos.

Esta es la opción más sencilla y económica de crear aplicaciones, puesto que al desarrollar una única aplicación se reducen al máximo los costes de desarrollo. Por el contrario, ofrece una peor experiencia de uso, ya que ignora las cualidades de los distintos dispositivos.

Para acceder a estas aplicaciones los usuarios usan los navegadores web que tienen en sus móviles, aunque en función del dispositivo pueden crear un enlace directo en sus pantallas principales que les permita acceder a ella.

3.1.4. Elección de nuestra tecnología

Como vemos para elegir la tecnología adecuada debemos de analizar las ventajas e inconvenientes de cada una y responder a algunas preguntas como: ¿cuánto coste estoy dispuesto a asumir?, ¿cuál es el público al que se dirigirá la app?, ¿necesito un diseño complejo y muy personalizado?

En nuestra aplicación debido a que queremos aprovechar el rendimiento al máximo para demostrar todo nuestro potencial a la vez que ofrecer una excelente experiencia de uso, hemos decidido optar por realizar una aplicación nativa.

3.2 Elección de la plataforma

Una vez decidido que queremos crear una aplicación nativa nos vuelven a surgir dos caminos a seguir, el primero de ellos es realizar una aplicación para Android y el otro es realizarlo para iOS.

3.2.1. Android

Android [4] es un sistema operativo diseñado por Google para teléfonos móviles. Lo que lo hace distinto es que está basado en Linux, un núcleo de sistema operativo libre, gratuito y multiplataforma. Inicialmente fue creado en el año 2003 por la compañía Android Inc. pero en 2005 la gigante Google lo compró y 2 años después presentó el sistema operativo.

Android es la plataforma más popular en el mercado[5], estimándose que está presente en el 90 % de los móviles, debido en parte a su simpleza, funciones y rendimiento lo que permite llevar la experiencia de usuario a un siguiente nivel.

Este crecimiento ha ido de la mano de un aumento exponencial en el número de apps disponibles en Google Play, teniendo a día de hoy más de 3 millones de apps en dicha tienda.

3.2.2. iOS

iOS [6] es un sistema operativo propiedad de Apple Inc. utilizado en dispositivos como smartphones, tablets... Está diseñado para equipos táctiles y se caracteriza por tener una interfaz intuitiva y muy sencilla de usar.

En cuanto a sus diferencias con Android, la principal es que su código es cerrado y únicamente está disponible para los equipos de dicha compañía. Esto origina que el consumidor no tenga una abundante posibilidad de personalización pero ofrece una de las experiencias más cómodas del mercado. Esto es debido a que iOS está diseñado para exprimir al máximo el hardware lo cual siempre se ha diferenciado considerablemente de los demás fabricantes.

Por otra parte, la única forma de instalar aplicaciones en este sistema es mediante la App Store, donde todas las aplicaciones son revisadas por la propia Apple realizando un proceso de validación muy estricto para que así sus usuarios tengan la mejor experiencia, seguridad y privacidad posible. Esto produce que el nivel de seguridad sea mucho más alto en estos dispositivos, sobre todo en lo referente a virus o malware.

3.2.3. Elección de nuestra plataforma

Como hemos visto tenemos dos grandes opciones a elegir, cada una con sus ventajas y desventajas. Sin embargo, lo que nos ha llevado a decantarnos por la plataforma de Apple es un estudio realizado por SensorTower[7] donde han publicado datos relativos a la categoría de aplicaciones que han conseguido recaudar un millón de dólares a lo largo del año 2018 y que nos ha ofrecido una visión del panorama global para saber cual sería la plataforma idónea para lanzar nuestra aplicación.

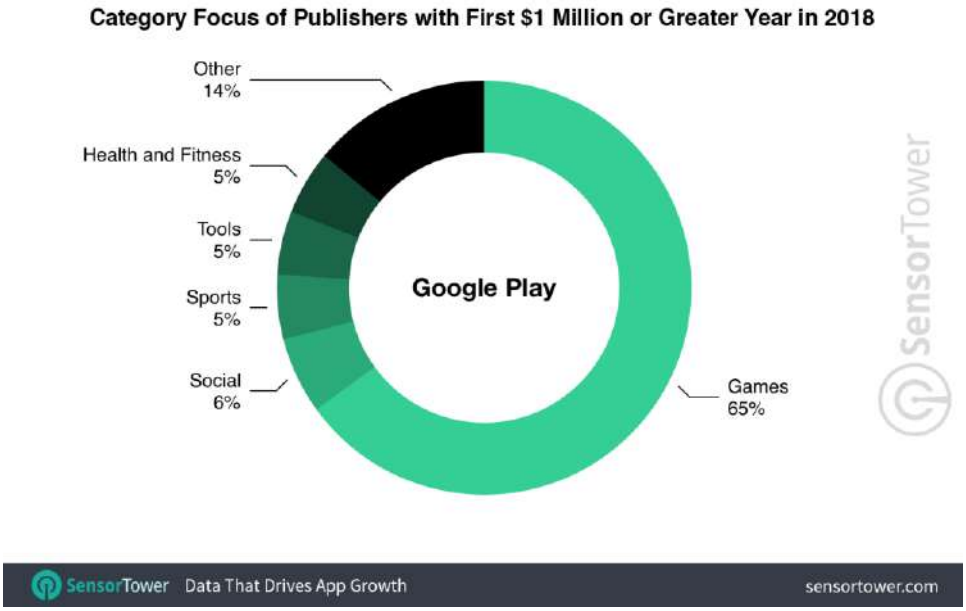

Figura 3.3: Categorías de la Google Play alcanzar el millón de dolares en 2018 [7]

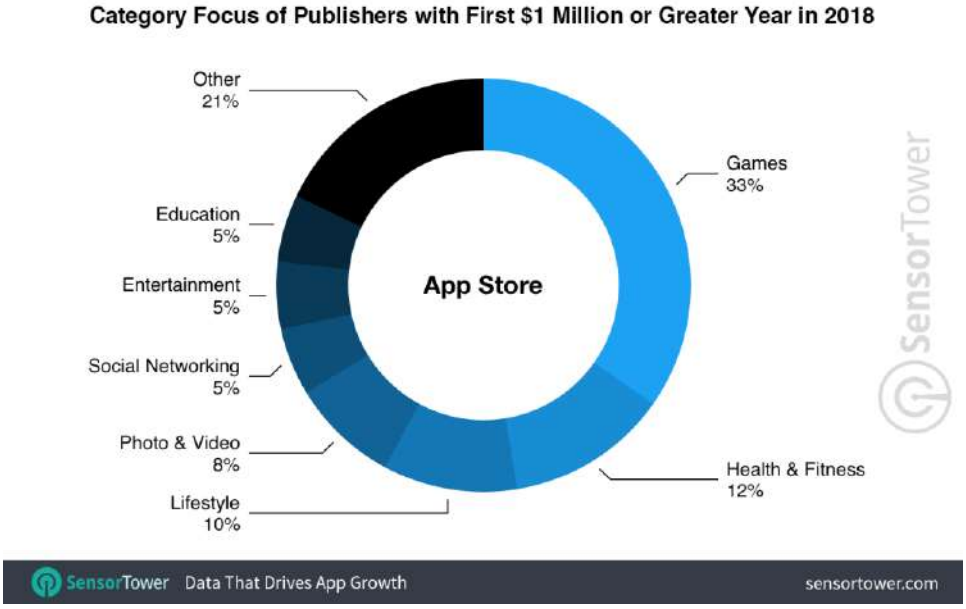

Figura 3.4: Categorías de la App Store en superar el millón de dolares en 2018 [7]

Como podemos comprobar nos encontramos ante una diferencia más que notable entre la App Store y la Google Play en lo que respecta a estas aplicaciones del millón de dólares. En la tienda de Android la gran mayoría son juegos pero en la App Store está mucho más diversificado en diversas categorías siendo un bloque muy importante el de salud.

He de aquí que hayamos preferido lanzar la aplicación para iOS en lugar de para Android.

3.3 Herramientas utilizadas

3.3.1. Xcode

Xcode[12] es un IDE (entorno de desarrollo integrado) de Apple que se ofrece de manera gratuita en sistemas MacOS. Incluye un conjunto de herramientas destinadas al desarrollo de software para los sistemas operativos de esta compañía.

Xcode trabaja conjuntamente con Interface Builder[13], el cual es una herramienta integrada en el propio IDE que nos permite diseñar interfaces de una manera sencilla y rápida. Esta herramienta nos permite diseñar interfaces de usuario completas sin la necesidad de escribir ningún código. Simplemente arrastrando y soltando ventanas, botones, campos de texto y otros objetos sobre el lienzo de diseño podemos crear una interfaz de usuario funcional.

Figura 3.5: Logo de CocoaPods

3.3.2. Swift

Swift[15] es un lenguaje de programación multiparadigma creado por Apple enfocado en el desarrollo de aplicaciones para iOS, Mac, Apple TV y Apple Watch. Es un lenguaje rápido y eficaz que proporciona información en tiempo real y se integra a la perfección con código escrito en Objective-C.

Figura 3.6: Logo de Swift

3.3.3. CocoaPods

Cocoapods[14] es un gestor de dependencias para los proyectos Swift y Objective-C. Este gestor nos permite agregar frameworks a nuestros proyectos y mantenerlos actualizados de una manera muy sencilla.

Figura 3.7: Logo de Cocoapods

3.3.4. Draw.io

Draw.io[16] es un software gratuito para la elaboración de diagramas de flujo, diagramas de proceso, organigramas, UML, ER y diagramas de red.

Figura 3.8: Logo de Draw.io

3.3.5. Git

Git[21] es un sistema de control de versiones distribuido, gratuito y de código abierto, diseñado para manejar todo tipo de proyectos, desde pequeños hasta muy grandes, con rapidez y eficiencia. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos.

Figura 3.9: Logo de Git

3.3.6. Firebase

Firebase[17] es una plataforma de backend como servicio, que dispone de una serie de herramientas para el desarrollo de aplicaciones.

Permite construir mejores apps delegando determinadas operaciones a Firebase, para así poder ahorrar tiempo, evitar errores y obtener un aceptable nivel de calidad. Uno de los aspectos que queremos destacar de esta plataforma es su asombrosa documentación la cual podemos consultar una vez accedemos a dicha plataforma.

Firebase tiene cuatro grandes pilares de negocio:

- Características para el desarrollo: permite la creación de mejores apps, minimizando el tiempo de optimización y desarrollo, mediante diferentes funciones, entre las que destacan el almacenamiento en la nube, procesos de autenticación de usuarios, servicio de bases de datos...

- Servicio de analítica: permite tener un control máximo del rendimiento de la app mediante métricas analíticas, todo desde un único panel y de forma gratuita.

- Recursos para el crecimiento: permite gestionar de manera fácil todos los usuarios de las aplicaciones, con el añadido de que se pueden captar nuevos usuarios, mediante invitaciones o notificaciones.

- Monetización: Firebase está integrada con una herramienta de monetización a través de la inclusión de publicidad en las aplicaciones llamada AdMob[18].

3.3.7. Autenticación de usuarios

Muchas de las aplicaciones que existen hoy en día se ven en la necesidad de identificar a los usuarios. La principal idea es que las apps sean capaces de reconocer al usuario, almacenar sus datos personales y ofrecer la misma experiencia personalizada desde cualquier dispositivo y sistema operativo.

Todo el proceso de autenticación de los usuarios en Firebase, bajo el servicio Firebase Authentication[19], se hace bajo estándares del mercado como OAuth 2.0 y OpenID Connect.

3.3.8. Base de datos en tiempo real

Firebase ofrece una plataforma denominada Firebase Realtime Database[20] que almacena y sincroniza la información alojada en una base de datos NoSQL en la nube. Esos datos que se almacenan en formato JSON, se sincronizan en tiempo real y continúan estando disponibles cuando la aplicación pierde la conexión a Internet.

La API está diseñada para permitir solo operaciones que se puedan ejecutar rápidamente. Eso nos permite crear una excelente experiencia de tiempo real que puede servir a millones de usuarios sin afectar la capacidad de respuesta.

Figura 3.10: Logo de Firebase

CAPÍTULO 4
Metodología

El desarrollo de software, es uno de los campos tecnológicos más competitivos y ha tenido una evolución constante en lo que se refiere a las metodologías con la finalidad de mejorar, optimizar procesos y ofrecer una mejor calidad.

Una metodología [22] en el ámbito del desarollo del software es una manera de estructurar, planificar y controlar el proceso de creación del mismo. Existe una amplia variedad de metodologías, cada una con sus propias ventajas e inconvenientes, por lo que es importante que se seleccione la idónea el proyecto que se va a realizar.

Dicho esto, mostramos a continuación un breve resumen de cuáles son algunas de las metodologías de desarrollo más utilizadas.

- **Modelo en cascada:** no se podrá avanzar a la siguiente false de desarrollo si la anterior no se encuentra totalmente terminada.
- **Prototipado:** en base a los requisitos y necesidades del cliente, se realiza un prototipo que permitirá a cualquier desarrollador continuar trabajando en él hasta finalizar el proyecto.
- **Incremental:** consiste en completar varias iteraciones del modelo en cascada dejando que así vaya evolucionando el producto, permitiendo que se agreguen nuevas especificaciones, funcionalidades u opciones que el usuario vaya requiriendo.
- **Espiral:** es el mismo proceso que se sigue en el modelo en cascada pero añadiéndole gestión de riesgos.

En el proyecto se ha utilizado la metodología incremental, por lo que a continuación vamos a proceder a analizarla en más profundidad.

4.1 Modelo incremental

Como hemos comentado anteriormente, el modelo incremental [8] repite el modelo de cascada una y otra vez, pero con pequeñas modificaciones o actualizaciones que se le puedan ir agregando. Es decir, aplica secuencias lineales de forma escalonada donde cada una de ellas produce un incremento del software.

Cuando se emplea este modelo, el primer incremento es un MVP (mínimo producto viable), sólo con los requisitos básicos. Los primeros incrementos son versiones incompletas del producto final, pero proporcionan al usuario la funcionalidad que precisa y también una plataforma para la evaluación.

En la figura que os presentamos a continuación se puede observar de una forma más visual como es este modelo.

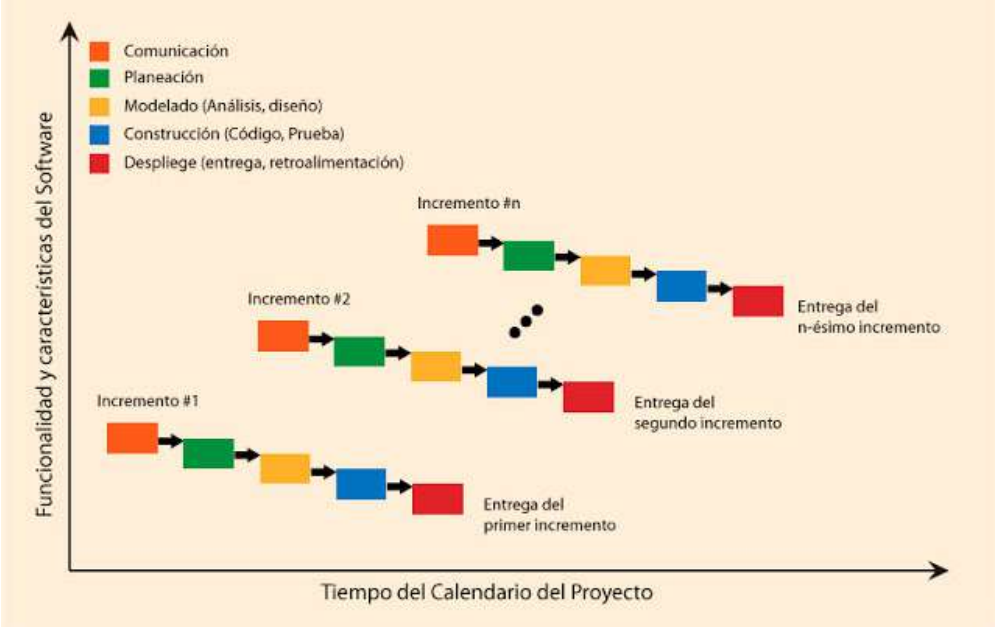

Figura 4.1: Modelo incremental de la ingeniería del Software [9]

4.1.1. Ventajas

- Se genera código operativo de forma rápida.
- Ofrece una retroalimentación en fases muy tempranas.
- Minimización del riesgo de falla en el proyecto porque los errores se van corrigiendo progresivamente.
- Permite separar la complejidad del proyecto, debido al desarrollo por iteraciones.

4.1.2. Inconvenientes

- Requiere de mucha planificación, tanto administrativa como técnica.
- Requiere de metas claras para conocer el estado del proyecto.

4.1.3. Sprints realizados

Para nuestro desarrollo dividimos el proyecto en un total de 6 sprints, cada uno con una duración de 1 semana. Vamos a proceder a comentar a continuación cada uno de ellos.

Antes de comenzar con los sprints de desarrollo se decidió realizar una fase previa donde se configuró todo el entorno, esto involucra tanto la creación del proyecto en Xcode como toda la configuración de Firebase necesaria, además de la instalación de librerías que íbamos a utilizar mediante la herramienta de Cocoapods. De esta manera, luego ya tendríamos todo montado y listo para comenzar con el desarrollo.

El primer sprint consistió en la creación de las ventanas con los mínimos aspectos visuales para así tener ya creada toda la navegación que puede realizar el usuario. Básicamente eran unos botones donde al pulsar sobre uno de ellos te llevaba a otra ventana. Además se aprovechó y comenzamos a construir el servicio de autentificación de Firebase.

El segundo sprint consistió en mejorar la interfaz gráfica con los más información sobre las recetas, añadiendo títulos, editables, listas, etc. También se comenzó el servicio de base de datos, de esta manera un usuario ya podía realizar las CRUD de las recetas.

En el tercer sprint nos centramos en realizar el servicio de almacenamiento, de forma que un usuario pudiera guardar una foto de perfil además de las fotos de las recetas. También se aprovechó y se incorporó una cámara que permitiera realizar estas fotografías. Por concluir se realizó algunos refactor sobre los otros servicios de Firebase.

El objetivo del cuarto sprint fue todo el tema de la creación de recetas, esto involucraba añadir nuevos botones, la dificultad, poder añadir pasos a las recetas... Todo esto venia de la mano con una mejora del servicio de base de datos para incorporas todos estos cambios.

El quinto sprint se dedicó a añadir las categorías de las recetas, de esta forma podríamos buscar dentro de categorías, añadirlas a las recetas, etc. En este punto comenzó toda la carga dinámica de la información de la cual hablaremos más profundamente en el capitulo de la implementación pero que abstractamente consiste en carga unos u otros datos dependiendo de la fuente de la que partimos.

En el sexto sprint nos dedicamos a mejorar el perfil de usuario, mostrando las recetas propias y que les gustaban a cada uno de ellos. Además hicimos públicos los perfiles de los usuarios, de esta forma, cuando una persona pulsara sobre la foto de otro usuario le llevara al perfil de este.

Ya por concluir, dedicamos otro sprint para realizar alguna deuda técnica generada a lo largo del desarrollo y además comprobar el correcto funcionamiento de todo el sistema.

CAPÍTULO 5
Análisis de requisitos

El análisis de requisitos [8][10] trata de capturar y describir detalladamente los requerimientos de funcionalidad y de calidad de servicio del producto que se va a desarrollar.

El requisito es una condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado.

Esta fase es una actividad que cada vez predomina más en la gerencia de proyectos, dado que se ha demostrado que una causa recurrente en su fracaso se origina de una inadecuada especificación de requisitos.

Una manera de describir las actividades que se realizan en la fase de análisis de requisitos según Raghavan[11] sería la siguiente:

- Determinación de requisitos: proceso mediante el cual los clientes o futuros usuarios del software descubren, revelan, articulan y comprenden los requisitos que desean.

- Análisis de requisitos: proceso de razonamiento sobre los requisitos obtenidos en la etapa anterior, detectando y resolviendo posibles inconsistencias o conflictos, coordinando los requisitos relacionados entre sí, etc.

- Especificación de requisitos: proceso de redacción o registro de los requisitos. Suele recurrirse a un lenguaje natural, lenguajes formales, modelos, gráficos, etc.

- Validación de los requisitos: confirmación, por parte del usuario o el cliente de que los requisitos especificados son válidos, consistentes, completos, etc.

5.1 Encuestas

Para conocer las características que deseaban tener los futuros usuarios de nuestra aplicación se ha llevado a cabo un estudio mediante entrevistas a un total de 10 personas. La entrevista comenzaba con un breve formulario el cual mostramos a continuación y luego un par de minutos uno a uno. Esto se ha realizado así para obtener una opinión más personal y que el usuario pudiera retransmitirnos mejor sus sentimientos y deseos.

El formulario que se realizó fue el siguiente:

1. ¿Eres hombre o mujer?

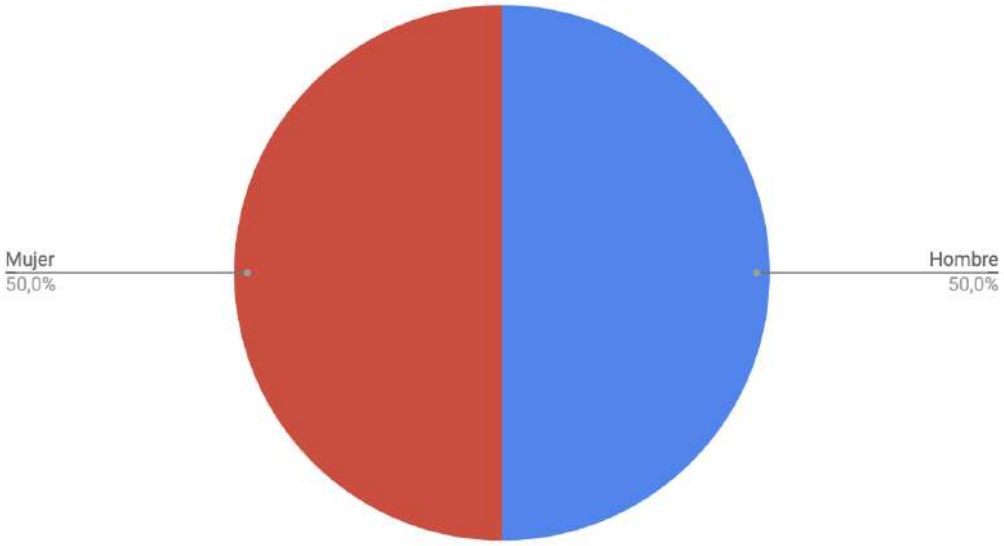

Figura 5.1: Resultados de la pregunta: ¿Eres hombre o mujer?

2. ¿Qué edad tienes?

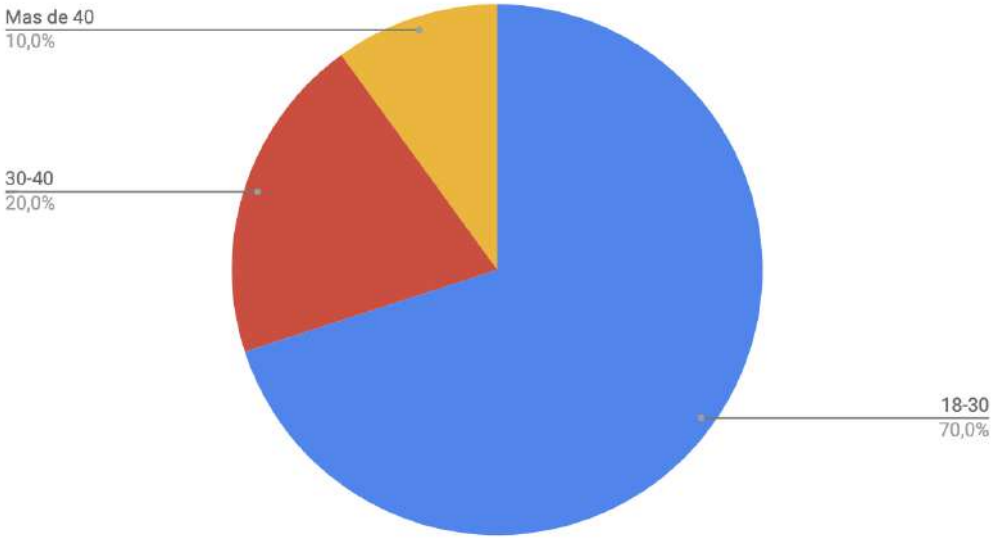

Figura 5.2: Resultados de la pregunta: ¿Qué edad tienes?

3. ¿Con qué frecuencia cocinas?

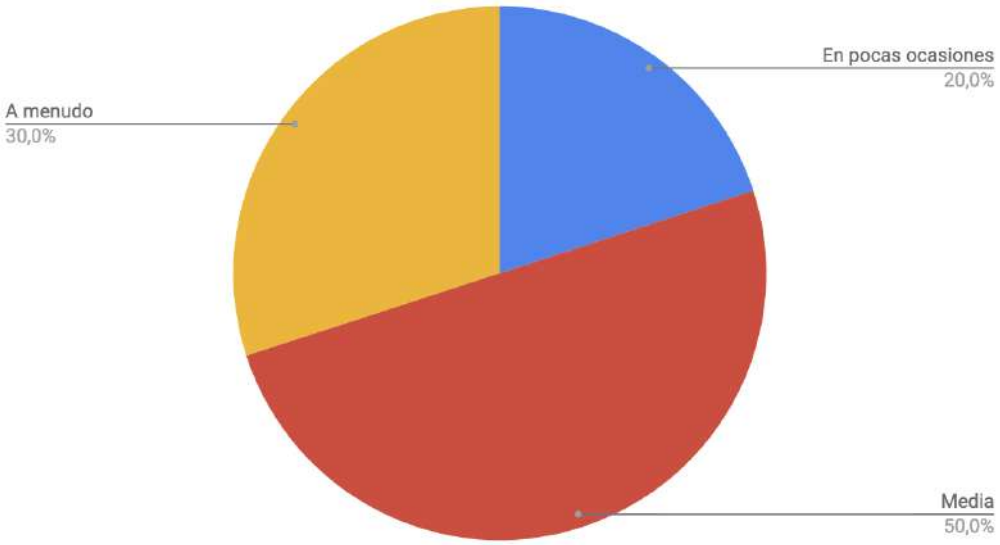

Figura 5.3: Resultados de la pregunta: ¿Con qué frecuencia cocinas?

4. ¿Alguna vez has usado alguna guía para cocinar, por ejemplo, libros de cocina, recetas que salen el la televisión, aplicaciones de cocina, etc.?

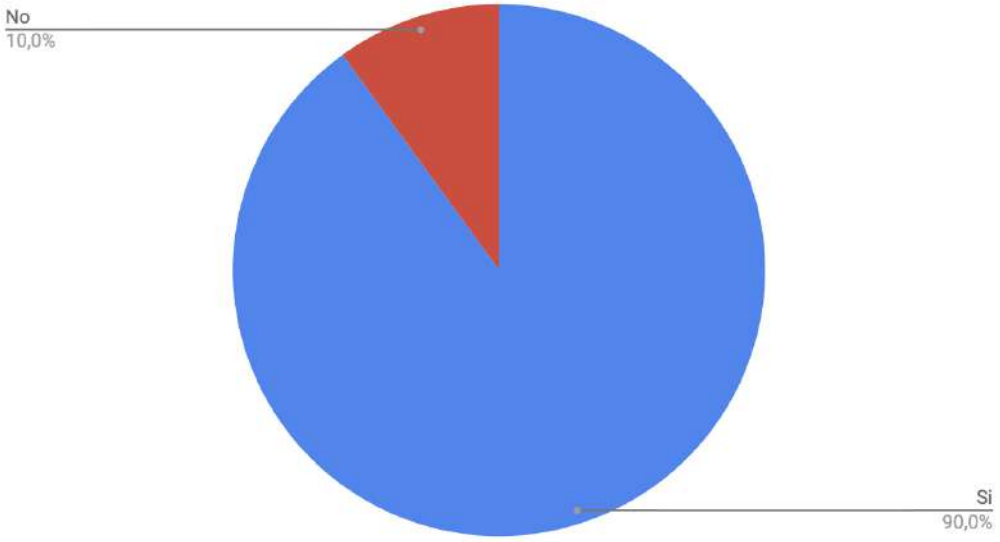

Figura 5.4: Resultados de la pregunta: ¿Alguna vez has usado alguna guia para cocinar?

5. ¿Consideras que sabes cocinar?

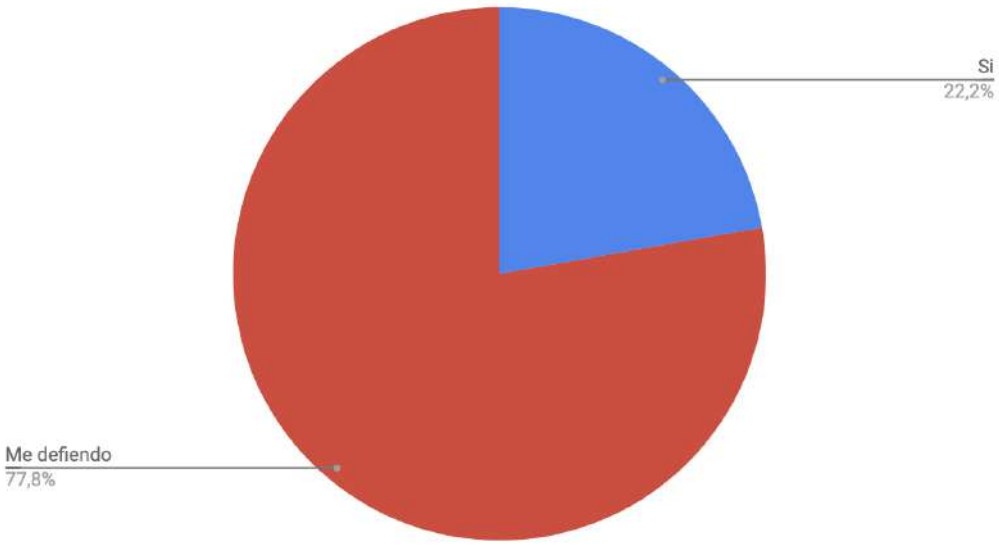

Figura 5.5: Resultados de la pregunta: ¿Qué edad tienes?

6. ¿Te gustaría aprender a cocinar nuevas recetas?

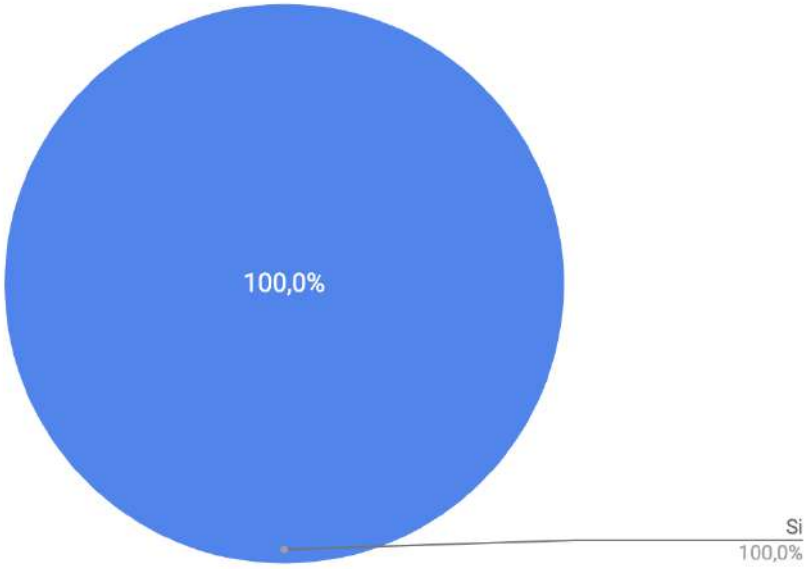

Figura 5.6: Resultados de la pregunta: ¿Te gustaría aprender a cocinar nuevas recetas?

5.2 Resultados obtenidos por el formulario

Tras haber realizado las anteriores preguntas podemos observar unos resultados bastante claros.

La media de edad de las personas suele estar entre los 18 y 30 años la cual es ideal para lanzar una aplicación donde los usuarios se vean con el tiempo y conocimiento suficiente sobre nuevas plataformas para poder desenvolverse con soltura en nuestro sistema.

La frecuencia en la que suelen cocinar también es idónea, como se refleja en la edad y en las respuestas libres que hemos realizado, muchas de las personas están comenzando a ser independientes, lo que conlleva realizar sus propias comidas.

Como punto fundamental, todas las personas están dispuestas a aprender nuevas recetas de cocina, lo que nos viene perfecto para lanzar nuestra aplicación.

5.3 Modelo de persona

Con toda la información extraída de la sección previa podemos crearnos un perfil de usuario. Esta es una técnica ampliamente aplicada en el entono del diseño centrado en el usuario (DCU) [31] que nos sirve para ser capaces de especificar los perfiles de los usuarios que pueden participar en nuestro sistema.

Esta técnica toma su primera aparición en el libro de Alan Cooper *The Inmates are Running The Asylum Alan Cooper* donde necesitaba conocer los perfiles de sus usuarios para prevenir equivocaciones por parte de estos.

Algunas de las características de esta técnica son:

- Una persona es un ente imaginario con el objetivo de describir a un usuario particular.
- Se deben de definir de forma solida y detallada.
- Debe proporcionarnos una imagen del usuario final.

Por otra parte, algunos de los beneficios que nos aporta son:

- Todos los miembros del personal de desarrollo tienen en mente a la misma persona cuando piensan en el usuario final.
- La persona creada puede ser empleada en reuniones para debatir aspectos a su alrededor. Por ejemplo, sobre que haría esa persona, lo que usaría y lo que no, etc.
- La incorporación de nuevos integrantes al equipo es más fácil ya que interiorizan al usuario final de manera más clara.

Dada esta breve introducción, vamos a ver como es la persona que hemos definido en base de los resultados recabados por el formulario y las distintas preguntas abiertas. La persona quedaría del siguiente modo:

Nombre: Pablo García Ramírez

Biografía

- 25 años de edad
- Vive en un piso con 2 amigos
- No tiene hijos
- Acaba de comenzar su primer trabajo tras acabar la universidad

Salud

- No tiene problemas físicos o de movilidad
- Padece de miopía en su visión

Tecnología

- Cuenta con acceso a Internet, tanto en el móvil como en casa
- Tiene un iPhone 7 Plus el cual siempre lleva encima
- Utiliza las redes sociales con frecuencia

Objetivos

- Quiere aprender a realizar nuevos platos de una forma sencilla
- Quiere poder elegir el tipo de cocina a aprender dependiendo del momento
- Necesita poder filtrar las recetas en base a su dificultad, para así comenzar por las más sencillas y poco a poco ir avanzando en dificultad
- Quiere una interfaz usable incluso cuando no lleve gafas, que le permita ver fácilmente la información y así descansar los ojos

5.4 Requisitos de la aplicación

En esta sección se representa un listado con los requisitos que se han extraído en base a las encuestas y las preguntas libres formuladas en las entrevistas.

Usuario no autentificado:

- **Crear un nuevo usuario:** crea un usuario nuevo.
- **Login:** autentifica al usuario para poder acceder a la aplicación.

Usuario autentificado:

- **Publicar una nueva receta:** crea una nueva receta.
- **Editar una nueva receta:** editar una receta si el usuario es el creador.
- **Añadir paso a una receta:** añade paso de cocina a una receta que se va a publicar.
- **Borrar paso a una receta:** elimina paso de cocina a una receta que se va a publicar.
- **Borrar una receta:** elimina una receta que haya elaborado de la base de datos.
- **Marcar como favorita una receta:** marcará la receta como favorita para que pueda ser consultada más tarde.
- **Desmarcar como favorita una receta:** desmarcará la receta como favorita y desaparecerá de la lista de favoritos del usuario.
- **Consultar el listado de recetas:** permite consultar las recetas que todos los usuarios hayan publicado.
- **Consultar el listado de una categoría:** permite consultar las recetas que pertenecen a una categoría.
- **Poner filtros a las categorías:** permite consultar las recetas que cumplen una serie de filtros.
- **Consultar perfil de un usuario:** permite consultar los perfiles públicos de otros chefs.
- **Consultar detalle de una receta:** permite consultar toda la información detallada de una receta (pasos, ingredientes).
- **Consultar el listado de recetas favoritas:** permite consultar las recetas que el usuario haya marcado como favorita.
- **Ver perfil:** permite visualizar el perfil del usuario.
- **Editar información del perfil:** permite añadir y editar información del usuario en la base de datos.
- **Consultar recetas propias:** permite consultar todas las recetas que ha elaborado.
- **Logout:** sale de la cuenta de usuario.

5.5 Diagrama de caso de uso

Los diagramas de caso de uso nos permiten modelar una parte del comportamiento de un sistema, concretamente identificando los principales requisitos funcionales. Los casos de uso capturan los requisitos funcionales del sistema a desarrollar. Las entidades que participan en estos casos de uso se les llaman actores. Un caso de uso también podría definirse como la secuencia de interacciones que inicia un actor sobre el sistema.

Los casos de uso, aunque son simples, debemos incluir más expresividad, en concreto podemos incluir las siguientes relaciones entre casos de uso:

- **Inclusión:** un caso de uso A incluye a un caso de uso B, si una instancia de A puede realizar todos los eventos que aparecen descritos en B. Se representa mediante una linea discontinua con la etiqueta de include.
- **Extensión:** un caso de uso B extendiéndose a un caso de uso A, si en la descripción de A figura una condición cuyo cumplimiento origina la ejecución de todos los eventos que aparecen descritos en B. Su representación es mediante una linea discontinua con la etiqueta de extends.
- **Herencia:** es una especialización de casos de uso, esto significa que los casos de uso especializados son refinamientos del flujo de eventos del caso base. En la práctica se representa mediante una flecha sin etiqueta.

y las siguientes relaciones entre actores:

- **Herencia:** los actores descendientes puede jugar todos los roles que juega el actor antecesor. Su representación es la misma que en la herencia entre casos de uso.

En nuestra aplicación tenemos dos actores principales, el usuario no autentificado y el usuario autentificado. Vamos a mostrar en el siguiente diagrama de casos de uso las distintas funcionalidades que pueden realizar cada uno de ellos en nuestro sistema.

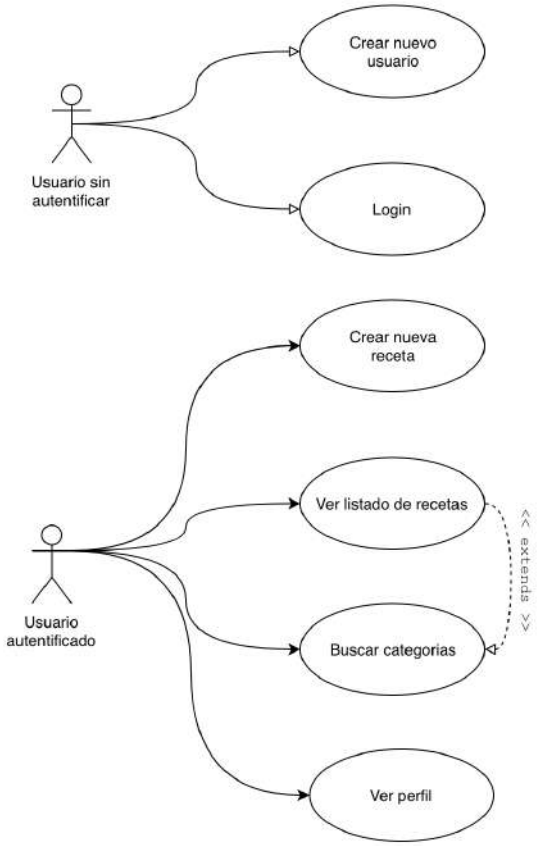

Figura 5.7: Caso de uso general

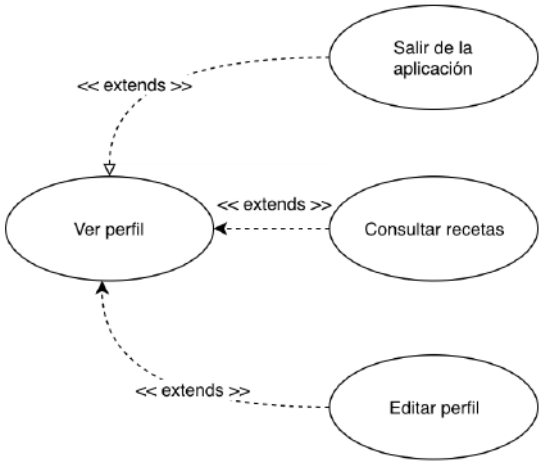

Figura 5.8: Caso de uso ver perfil

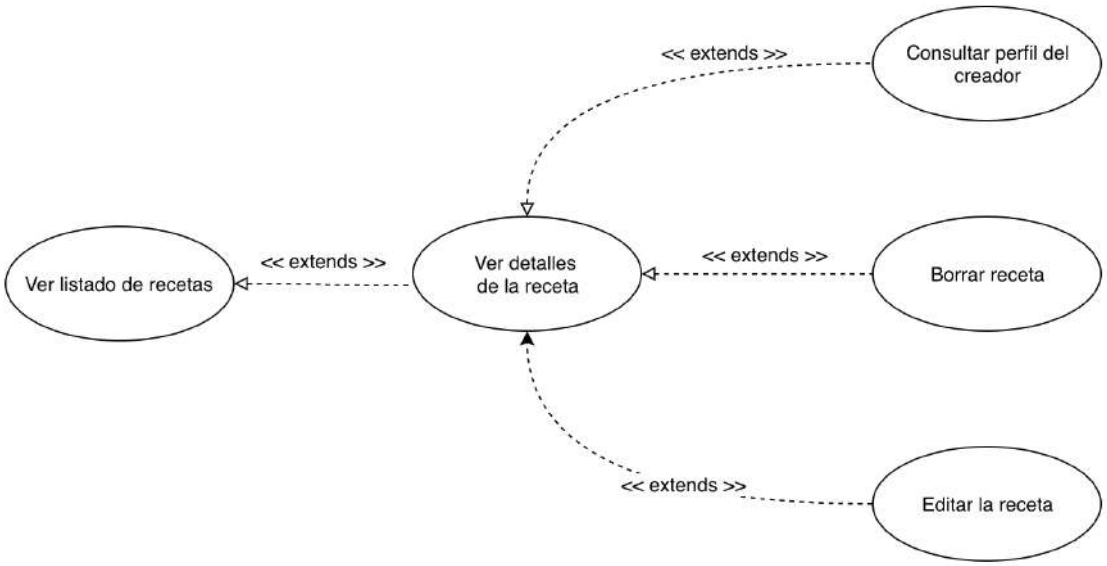

Figura 5.9: Caso de uso ver listado de recetas

Cuando no estamos autentificados en la aplicación una persona puede crearse un nuevo usuario en nuestro sistema mediante un correo electrónico y una contraseña o autentificarse para entrar a la app mediante esas credenciales.

Por otro lado, un usuario ya autentificado puede realizar 4 acciones principales:

- **Creación de nuevas recetas:** sirve para crear nuevas recetas, para ello un usuario debe introducir los siguientes datos: titulo, descripción, imagen de la receta (opcional), tiempo de elaboración, listado de ingredientes necesarios, categoría a la que pertenece, dificultad que tiene la receta y por último los pasos necesarios para llevar a desarrollar la receta.
- **Consultar el listado de recetas:** como su nombre indica es un listado de las recetas que tiene nuestra aplicación. Además el usuario podrá filtrar por nombres, dificultad y fecha de creación. Además, cuando un usuario seleccione una receta se le mostrará una vista detallada de dicha receta. Aquí dispondrá de más información de la misma, como pueden ser los pasos, ingredientes, categoría, etc. En adición a esto, también podrá marcar la receta como favorita y ver el perfil del creador de ese plato. En el caso de que sea el dueño de la receta podrá tanto eliminarla como editar la información de la misma siguiendo el mismo flujo de la creación, pero con la información ya cargada.
- **Búsqueda por categorías:** es un listado con las categorías que tienen recetas disponibles donde una vez se selecciona una de ellas se muestra el listado de recetas visto anteriormente pero filtradas por las que pertenecen a la categoría seleccionada.
- **Ver perfil:** aquí un usuario puede editar su nombre y fotografía pública, consultar su listado de recetas propias y recetas favoritas, y además salir de la aplicación.

CAPÍTULO 6
Diseño

6.1 Diagrama UML

La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. Un modelo es una representación simplificada de la realidad; el modelo UML [28] describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema.

A continuación se muestra el diagrama UML de nuestra aplicación.

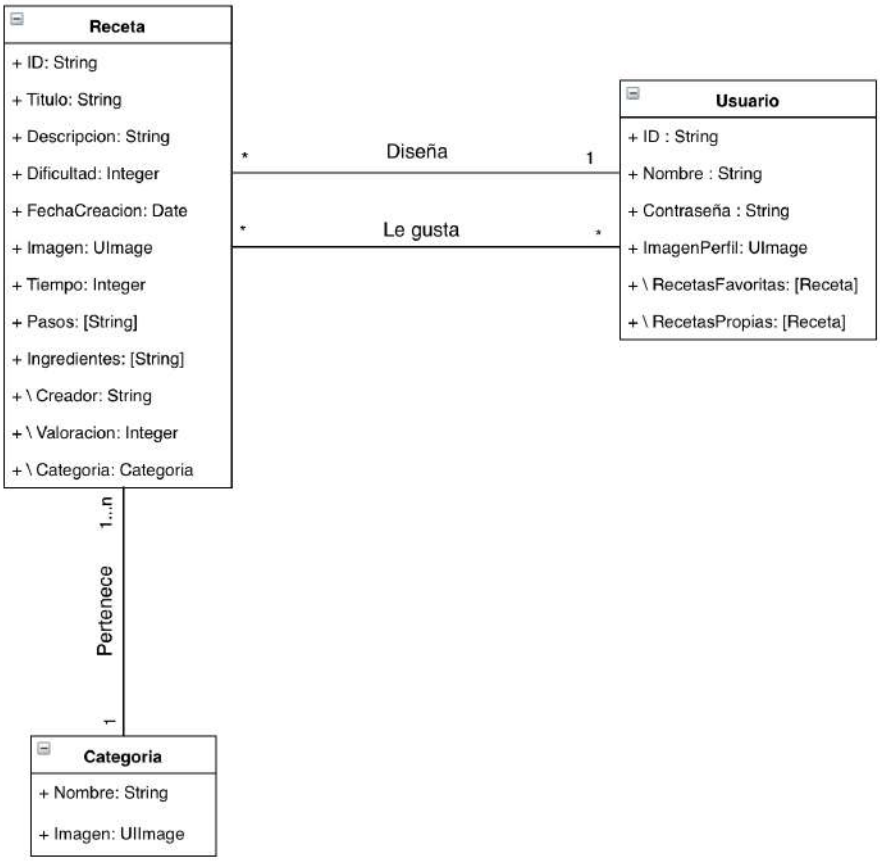

Figura 6.1: Diagrama UML de Cookit

6.2 Mockup

Esta sección consiste en un mockup rápido, el cual nos ha servido para tener una idea mental de como será la interfaz y que nos ha ayudado a situar las vistas de nuestra aplicación. A continuación vamos a ver algunos de los bocetos más representativos, dejaremos en el apéndice B el resto de ellos.

La siguiente figura ilustra la pantalla de perfil, como se puede comprobar se dividió en dos secciones. La primera de ellas es donde aparecería la información personal y una fotografía del usuario. A continuación tendríamos la lista de las recetas creadas por el usuario y la lista de recetas que le han gustado.

Figura 6.2: Mockup pantalla perfil

El siguiente mockup sería la pantalla de los detalles de una receta. En él podemos ver como hay un titulo de la receta, una descripción y una fotografía, además tendríamos un listado con los distintos pasos que conforman el plato.

Figura 6.3: Mockup pantalla detalle de receta

Por concluir vamos a ver el mockup del listado de recetas. Esta contaría como hemos dicho con una lista donde mostraremos diferente información de una receta, como puede ser una fotografía, su nombre, la dificultad y una breve descripción de esta.

Figura 6.4: Mockup pantalla inicial

6.3 Diseño de la interfaz

Basándonos en el mockup de la sección anterior hemos realizado la interfaz final de nuestra app con la herramienta de Interfaz Builder de Xcode. El diseño de nuestra app se ha realizado de forma que este se adapte a los distintos dispositivos que puedan existir, desde un iPhone 6 con la pantalla más pequeña, hasta un iPad de 12.9 pulgadas.

Ahora vamos a ver como serían algunas de las pantallas vista desde un iPhone. El resto de pantallas se podrán consultar en el manual de usuario, disponible en el apéndice A. Además, en el apéndice C se encuentra como serían estas pantallas vistas desde un iPad.

La siguiente figura ilustra la pantalla con el listado de las recetas disponibles. Como se puede observar cuenta con una imagen bastante grande de la misma para que llame al apetito del usuario y así entre en la misma. También se muestra información relevante con un estilo más *material design [30]*. Aquí podemos encontrar algunos elementos como son el título, la dificultad, el numero de personas a las que le gusta la receta...

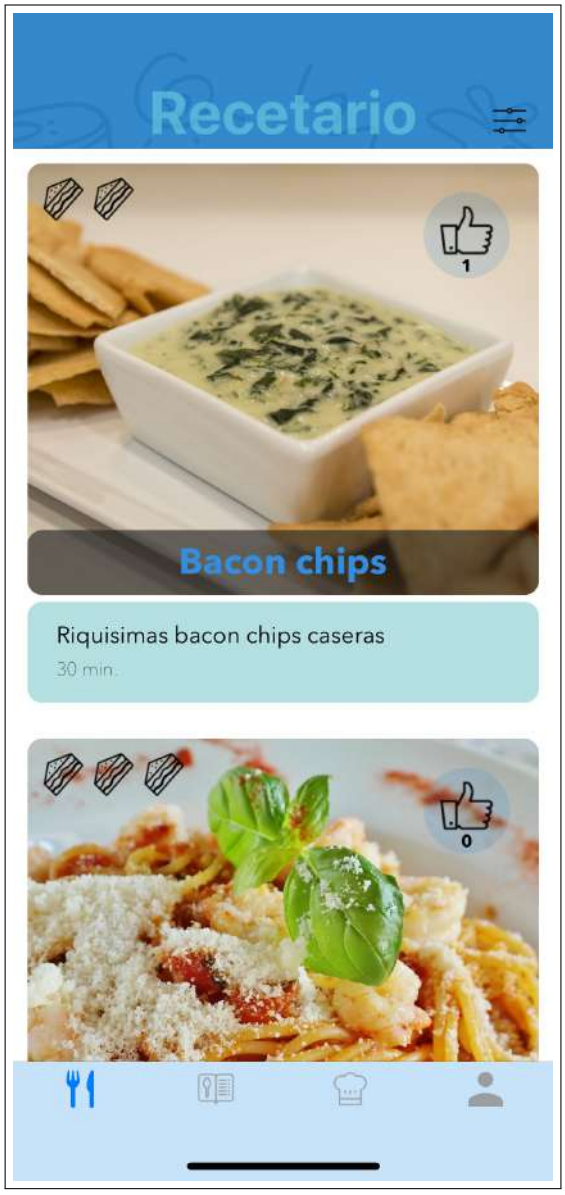

Figura 6.5: Pantalla de inicio en iPhone

Esta figura corresponde a la vista de una receta. Aquí podemos encontrar un listado ordenado de los pasos necesarios para elaborar la receta. Se ha intentado seguir un estilo minimalista y solamente mostrar la información imprescindible para esta se pueda realizar correctamente.

Figura 6.6: Vista de una receta en iPhone

La siguiente pantalla hace referencia al perfil de un usuario. Cuenta con dos listas horizontales donde están las recetas propias y favoritas del usuario que se está visitando el perfil. Como se puede observar, también se ha seguido un estilo minimalista y con matices de *material design* como pueden ser las imágenes redondeadas o sus bordes.

Figura 6.7: Pantalla de perfil de un usuario en iPhone

6.4 Flujo de la vista

En esta sección vamos a ver un diagrama que nos muestra el flujo de datos de la aplicación, es decir, la forma en la que el usuario puede ir navegando entre las distintas pantallas. En dicho diagrama se puede observar como se ha reutilizado las diferentes pantallas que conforman nuestra aplicación, de esta forma estamos cargando los datos de cada pantalla de una forma dinámica dependiendo del punto en el que se encuentra el usuario. Esta forma de trabajo se verá más detalladamente en la octava sección del octavo capítulo el cual hace referencia a dicha implementación.

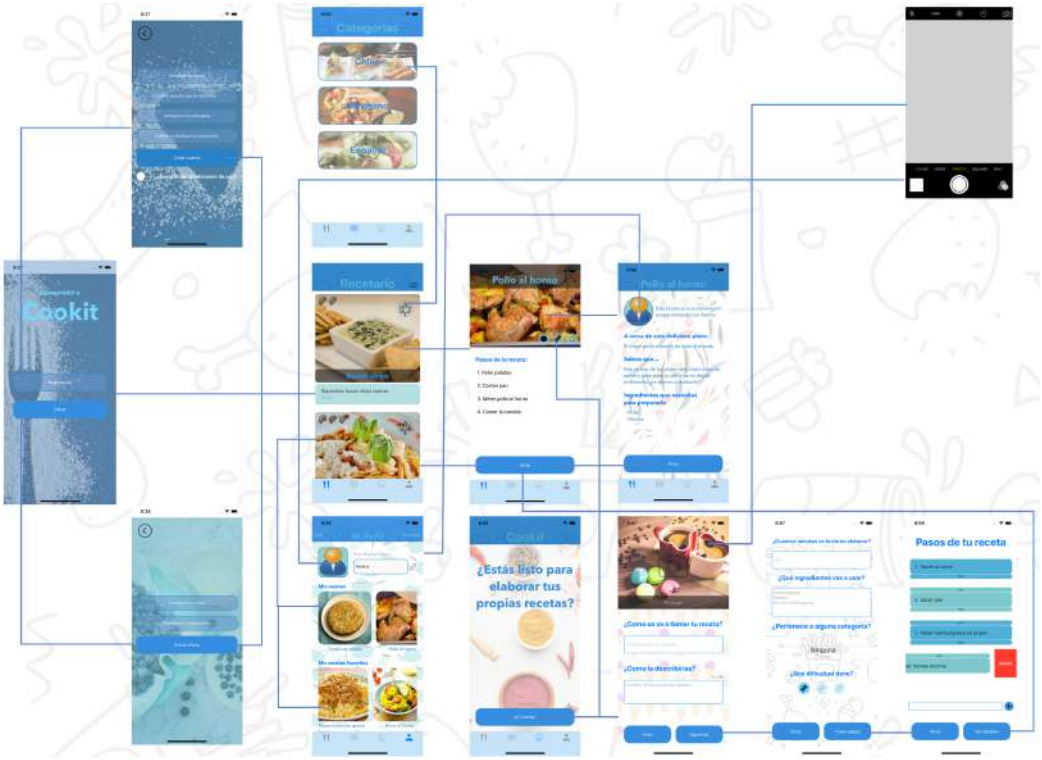

Figura 6.8: Flujo de navegación de Cookit

CAPÍTULO 7
Arquitectura

En este capitulo vamos a ver como está compuesta la arquitectura de todo nuestro sistema. Lo hemos dividido en tres secciones. En el primero hablaremos de la arquitectura externa, donde veremos como son las comunicaciones entre el dispositivo del usuario y el servidor de Firebase. Posteriormente, continuaremos analizando la arquitectura interna de nuestra aplicación donde veremos mas detalladamente como esta compuesta y organizada, pero sin entrar en detalles de implementación. Por último, veremos la estructura y la organización de la información en la base de datos.

Gracias a que comenzamos realizando encuestas y analizando las necesidades de los usuarios, hemos podido diseñar la arquitectura de nuestro sistema de la manera más optima y extensible a futuras funcionalidades.

A continuación, vamos a ver estas tres secciones.

7.1 Arquitectura externa

El objetivo de esta sección es la de tener una visión general de como esta formada nuestra aplicación. Como se puede observar a continuación básicamente seguimos una arquitectura de cliente-servidor [27].

Por un lado, tenemos la aplicación iOS que se instala en el dispositivo del usuario y tiene las siguientes características:

- Administra la interfaz de usuario.
- Interactúa con el usuario mediante una interfaz gráfica.
- Procesa la lógica de la aplicación y realiza validaciones locales.
- Se encarga de realizar las solicitudes al servidor y de recibir su respuesta.
- Formatea los resultados obtenidos para presentárselos al usuario.

Por otro lado tenemos Firebase, que nos hace de servidor en nuestra infraestructura. Entre sus diversos papeles, destacamos las siguientes tres:

- Tras recibir una solicitud, la procesa y devuelve una respuesta al cliente.
- Puede aceptar un gran número de clientes simultáneos.
- Procesa la lógica de la aplicación y realizar validaciones a nivel de bases de datos.

Además, como se puede visualizar en la figura que presentamos a continuación, dentro de Firebase estamos consumiendo tres servicios: el de autentificación, el de base de datos y el de almacenamiento. Este servidor es mantenido completamente por Google, lo que nos encapsula toda la complejidad del sistema y para nosotros es transparente, simplemente lo consumimos.

Por ultimo, la comunicación entre nuestra app y Firebase se realiza mediante peticiones https, estas peticiones también están encapsuladas en una API mediante funciones que Firebase ofrece para consumir sus servicios, de forma que la complejidad para nosotros desaparece y simplemente tenemos que realizar las llamadas a dichas funciones. Como respuesta obtenemos datos JavaScript Object Notation (JSON) [23], el cual es un formato ampliamente utilizado para transmitir datos en aplicaciones.

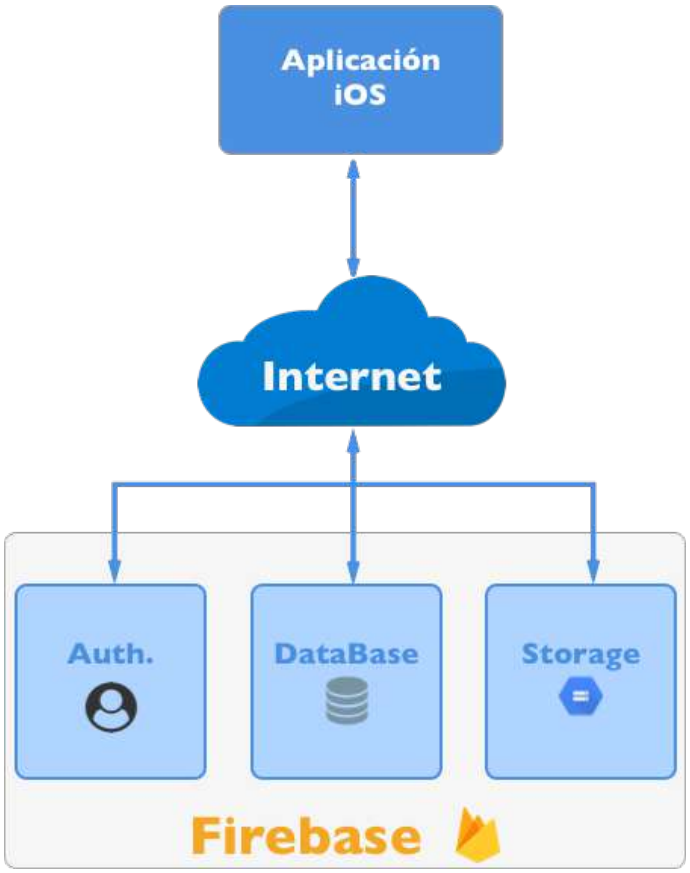

Figura 7.1: Arquitectura externa de Cookit

7.2 Arquitectura interna

En esta sección vamos a analizar la arquitectura interna de nuestra aplicación, es decir, como está formada la aplicación que el usuario se instala en su dispositivo móvil.

Como se puede observar en el diagrama de abajo, hemos seguido la clásica arquitectura de Modelo-Vista-Controlador (MVC) [24]. En términos generales, esta arquitectura separa la lógica de la aplicación de la vista, de forma que, cuando realizamos un cambio en alguna parte, no afecte a la otra.

Veamos los distintos componentes de esta arquitectura y una breve definición de cada una de ellos:

- **Modelo:** es la representación de la información.
- **Controlador:** se encarga de recibir las ordenes del usuario para pedirle al modelo los datos oportunos y devolvérselos a la vista.
- **Vista:** son la representación visual de los datos.

Aparte de la arquitectura MVC, hemos añadido una capa por encima que la hemos denominado servicios. Aquí podemos encontrar una envoltura mediante el patrón de diseño fachada [25] de los distintos servicios que estamos consumiendo de Firebase. De esta forma, en nuestra aplicación llamamos a estos servicios y ya son ellos los encargados de realizar las peticiones oportunas. Esto se ha realizado así para abstraer toda comunicación, de manera que, si el día de mañana queremos dejar de usar Firebase y desarrollar nuestro propio backend, solamente tengamos que cambiar estos tres ficheros, y el resto de la aplicación funcionaria igualmente.

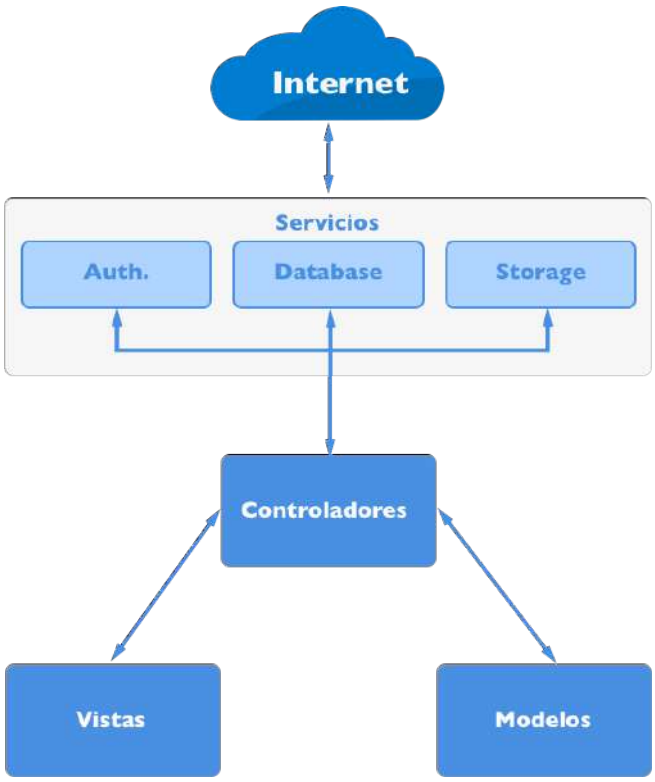

Figura 7.2: Arquitectura interna de Cookit

7.3 Arquitectura de la base de datos

En esta sección vamos a ver un diagrama de como esta estructurada nuestra base de datos en Firebase. Esta base de datos es no relacional [26], donde el acceso a los datos se realiza mediante clave-valor. Esto nos permite una mayor flexibilidad, escalabilidad y un alto rendimiento. En este diagrama se puede observar el uso de patrones de acceso que nos permiten un mayor rendimiento a la hora de añadir y modificar ciertos valores.

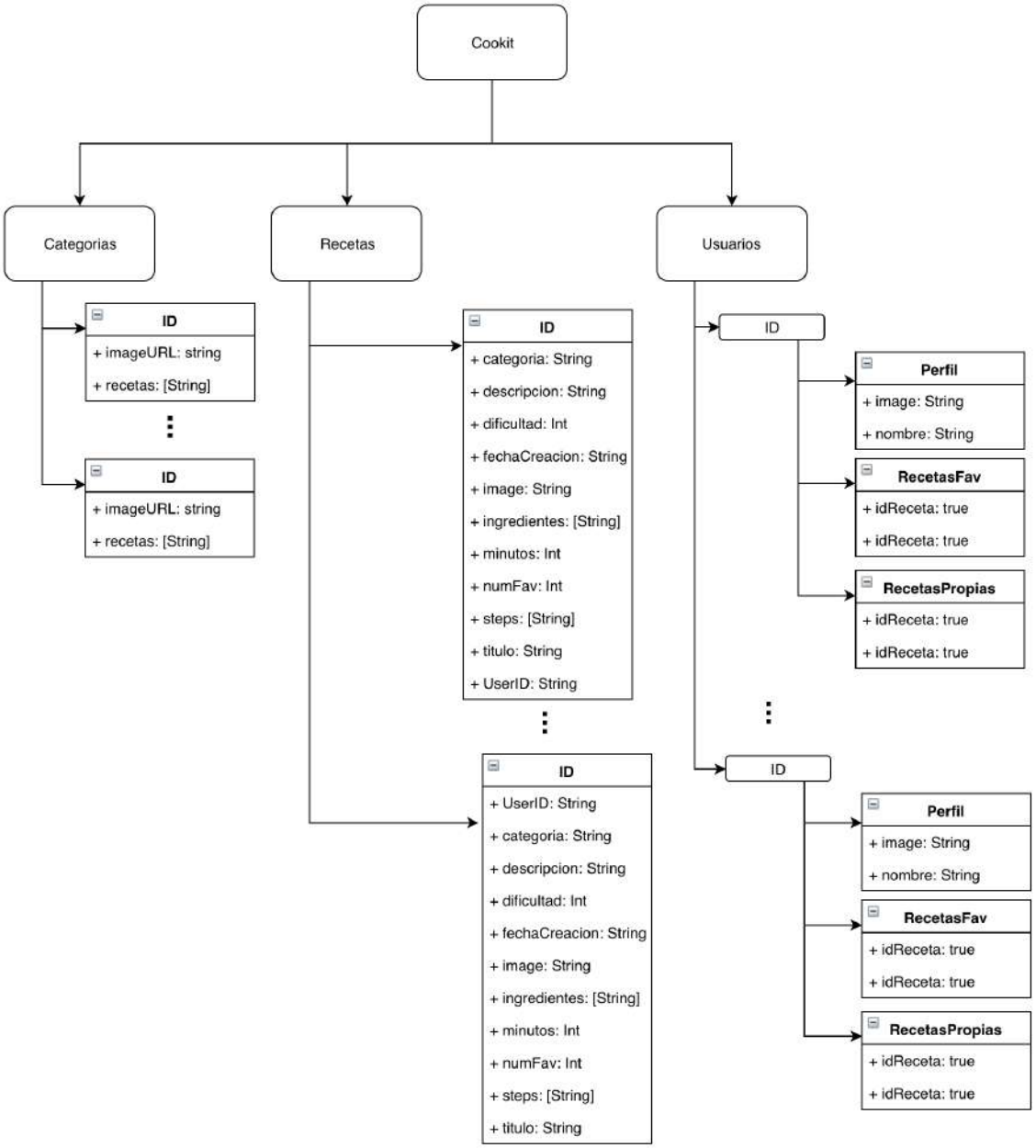

Figura 7.3: Arquitectura de la base de datos de Cookit

CAPÍTULO 8
Implementación

La figura que se muestra a continuación refleja una visión global de las distintas carpetas que agrupan la arquitectura de nuestro sistema.

Figura 8.1: División de las carpetas

A continuación, vamos a desglosarlas, estudiarlas y ver el rol que juegan cada una de estas piezas. Además, cuando nos adentremos en cada una de ellas veremos algunos ejemplos de implementación.

8.1 Extensiones

Las extensiones nos permiten añadir funcionalidades propias a un tipo ya existente. Dicho con otras palabras, nos permiten ampliar aún más la versatilidad que nos ofrece un lenguaje.

En nuestro caso, cuando nos encontrábamos desarrollando nuestra app nos dimos cuenta de que habían ciertos métodos que estábamos constantemente utilizando, por lo que decidimos extender ciertas clases para añadir estas funciones.

Figura 8.2: Ficheros de extensión

8.1.1. Extensión a la clase String

En la siguiente imagen se puede observar como se ha añadido la funcionalidad de poner la primera la primera letra de una palabra en mayúscula en la clase String.

```
1 extension String {
2     func capitalizingFirstLetter() → String {
3         return prefix(1).capitalized + dropFirst()
4     }
5
6     mutating func capitalizeFirstLetter() {
7         self = self.capitalizingFirstLetter()
8     }
9 }
```

Figura 8.3: Implementación de la extensión de los ViewControllers

8.2 Helpers

Esta carpeta tiene ficheros de ayuda que nos facilitarán futuras modificaciones en el código. Nosotros solamente hemos creado un único fichero, el cual es el encargado de devolvernos las imágenes por defecto, tanto la de un usuario como la de una receta. El motivo por el cual se ha realizado así es que si en un futuro queríamos cambiar alguna de estas imágenes, solamente tendríamos que venir a este fichero, cambiar las imágenes por las nuevas y listo. De no hacerlo así tendríamos que ir fichero por fichero cambiando estas imágenes manualmente, lo que no sería para nada extensible.

```
1 import UIKit
2
3 func getDefaultRecetaImage() → UIImage {
4     return 
5 }
6
7 func getDefaultUserImage() → UIImage {
8     return 
9 }
```

Figura 8.4: Implementación del helper de las imágenes

8.3 Ficheros de soporte

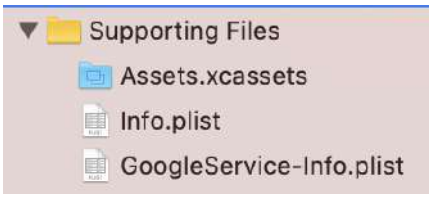

Figura 8.5: Ficheros de soporte

8.3.1. Assets

Carpeta que contiene todas las imágenes del sistema. Como se puede observar en la siguiente figura, cada imagen esta formada por tres imágenes de la misma pero en distintas resoluciones. Esto se ha hecho así para ofrecer la mayor calidad posible al usuario final.

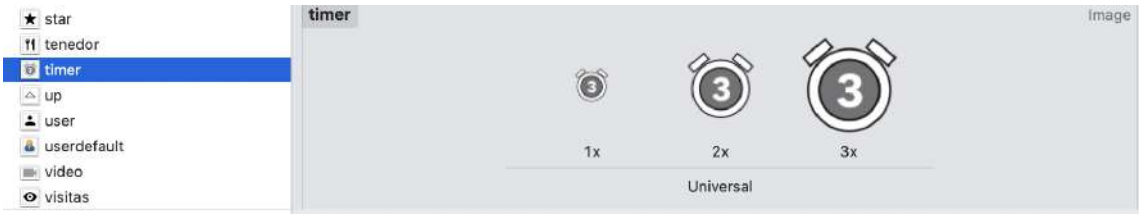

Figura 8.6: Carpeta de Assets

8.3.2. Info.plist

Contiene toda la información de los permisos que hacen falta en la aplicación. Entre ellos se encuentran los del acceso a la cámara, al micrófono, etc.

Information Property List	Dictionary	(18 items)
Localization native development re...	String	$(DEVELOPMENT_LANGUAGE)
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
Privacy - Camera Usage Description	String	¿Nos dejas acceder a la cámara para hacer fotos?
Privacy - Microphone Usage Desc...	String	¿Nos dejas acceder al micrófono?
Privacy - Photo Library Usage Des...	String	¿Nos dejas guardar fotos en tu librería?
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main

Figura 8.7: Carpeta de Assets

8.3.3. GoogleService-Info.plist

Fichero que contiene la configuración con los servicios de Google. Por motivos de seguridad no se va a proporcionar una imagen de este, pero es muy similar al Info.plist visto anteriormente.

8.4 La camara

Debido a que realizar la implementación de una cámara personalizada es un proceso muy laborioso se ha decidido utilizar una librería de Apple denominada AVCam [29] que ya nos ofrece toda esta implementación, de modo que, lo que hemos realizado, es adaptar esta librería a nuestras necesidades.

Figura 8.8: Estructura de la cámara

A continuación vamos a ver como hemos adaptado parte de esta librería para nuestro uso actual, así como futuras extensiones.

```
72      //MARK: - AAPL Camera View Methods
73
74      var segueID: String = "ShowPerfil"
75      var receta: Receta?
76      var img: Data?
77
78      func videoRecordingComplete(videoURL: URL) {
79          performSegue(withIdentifier: segueID, sender: ["videoURL":
                videoURL])
80      }
81
82      func videoRecordingFailed(errorMessage: String?) {
83          print(errorMessage as Any)
84      }
85
86      func imageTaken(data: Data) {
87          DispatchQueue.main.async {
88              self.performSegue(withIdentifier: self.segueID, sender:
                    ["imageData": data])
89          }
90      }
91
92      func imageFailed(errorMessage: String?) {
93          print(errorMessage as Any)
94      }
```

Figura 8.9: Implementación de la cámara

8.5 Modelos de datos

Como ya vimos en el diagrama de clases, nuestra aplicación contiene tres modelos de datos, el de las recetas, el de las categorías y por último el del usuario.

Figura 8.10: Carpeta de modelos

A continuación vamos a ver como se ha realizado la implementación del modelo de usuario. Como se puede observar en la siguiente figura, el modelo consta de tres partes. Por un lado tenemos las variables privadas, luego todas las funciones de lectura y modificación de estas variables, y ya por último los inicializadores del modelo.

```
1  import UIKit
2
3  struct User {
4
5      private var _uid         : String
6      private var _nombre      : String
7      private var _imageURL    : String?
8      private var _imagePerfil : UIImage?
9
10     var uid: String {
11         set {
12             _uid = newValue
13         }
14         get {
15             return _uid
16         }
17     }
18     var nombre: String {
19         set {
20             _nombre = newValue
21         }
22         get {
23             return _nombre
24         }
25     }
26     var imageURL: String? {
27         set {
28             _imageURL = newValue
29         }
30         get {
31             return _imageURL
32         }
33     }
```

```
34      var imagePerfil: UIImage? {
35          set {
36              _imagePerfil = newValue
37          }
38          get {
39              return _imagePerfil
40          }
41      }
42
43      init(uid : String, nombre : String, imagePerfil : UIImage?, imageURL : String?) {
44          self._uid         = uid
45          self._nombre      = nombre
46          self._imageURL    = imageURL
47          self._imagePerfil = imagePerfil
48      }
49
50      init(uid: String, nombre: String) {
51          self.init(uid: uid, nombre: nombre, imagePerfil: nil, imageURL: nil)
52      }
53  }
```

Figura 8.11: Implementación del modelo de datos del usuario

8.6 Capa de servicios

Como ya vimos en el capitulo de la arquitectura, nuestra aplicación consta de una serie de servicios los cuales nos hacen de intermediarios entre los controladores y los distintos servicios de Google que estamos consumiendo.

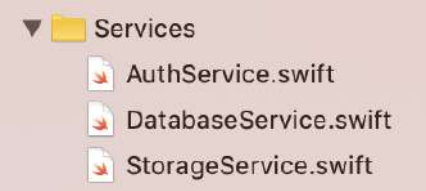

Figura 8.12: Estructura de los servicios

Ahora, es el momento de adentrarnos en la implementación de estos servicios.

8.6.1. Servicio de autentificación

Este servicio consta de varias funcionalidades, entre los que destacan:

creación de nuevos usuarios:

```
32      func createUser(email: String, password: String, userName: String, onComplete:
            Completion?) {
33          Auth.auth().createUser(withEmail: email, password: password) { (user, error) in
34              if let error = (error as NSError?) {
35                  self.handleFirebaseError(error: error, onComplete: onComplete)
36              } else {
37                  DatabaseService.shared.saveUser(uid: user!.uid, userName: userName)
38                  onComplete?(nil, user!)
39              }
40          }
41      }
```

Figura 8.13: Implementación del registro de nuevos usuarios

manejo de posibles errores que puedan suceder en la comunicación de nuestra app y Firebase:

```
52      func handleFirebaseError(error: NSError, onComplete: Completion?) {
53          if let errorCode = AuthErrorCode(rawValue: error.code) {
54              switch errorCode {
55              case .invalidEmail:
56                  onComplete?("Email incorrecto", nil)
57                  break
58              case .wrongPassword, .invalidCredential:
59                  onComplete?("Contraseña incorrecta", nil)
60                  break
61              case .userDisabled:
62                  onComplete?("Este usuario no tiene permisos para entrar", nil)
63                  break
64              case .emailAlreadyInUse:
65                  onComplete?("No se ha podido crear la cuenta. Este email ya está
                        registrado", nil)
66                  break
67              case .networkError:
68                  onComplete?("Se ha producido un error en la red. Intentelo más tarde", nil)
69                  break
70              case .weakPassword:
71                  onComplete?("Contraseña muy débil. Debe tener 6 o más caracteres", nil)
72                  break
73              default:
74                  onComplete?("Ha habido un problema. Prueba de nuevo en unos minutos", nil)
75              }
76          }
77      }
```

Figura 8.14: Implementación del manejo de errores de autentificación

8.6.2. Servicio de almacenamiento

Este servicio es el que se encarga de acceder al almacenamiento de Firebase, lugar donde guardamos las distintas imágenes de las recetas y fotos de perfil de nuestros usuarios.

Nos gustaría remarcar las siguientes funcionalidades de este servicio:

- Implementación de una **cache local**, de forma que, si una imagen ya se había descargado, no se vuelva a descargar de Firebase.
- **Compresión de las imágenes** que se envían a Firebase. De esta forma se ha conseguido limitar el almacenamiento a una media de 250KB, por lo que se ha producido un notable ahorro de capacidad de almacenamiento a la vez que de consumo de datos en el dispositivo del cliente.

```
43
44      func getImgFromUrl(imageURL: String, completion: @escaping (Error?, UIImage?) → Void) {
45          if let imagenCacheada = self.cacheImagenes[imageURL] {
46              completion(nil, imagenCacheada)
47          } else {
48              self.getDataFrom(imageURL: imageURL) { (error, data) in
49                  if error ≠ nil {
50                      completion(error, nil)
51                  } else {
52                      let img = UIImage(data: data!)
53                      self.cacheImagenes[imageURL] = img
54                      completion(nil, img)
55                  }
56              }
57          }
58      }
59
60      func borrarImage(imageURL: String, completion: @escaping (String?) → Void) {
61          let ref = self.imageStoreRef.child(imageURL)
62          ref.delete { error in
63              if error ≠ nil {
64                  completion(error?.localizedDescription)
65              } else {
66                  completion(nil)
67              }
68          }
69      }
```

x

Figura 8.15: Parte de la implementación del servicio de storage

8.6.3. Servicio de base de datos

Este es el último servicio que tenemos en el sistema. Es el encargado de toda comunicación (recepción, envió y actualización de la información) con la base de datos en tiempo real de Firebase que estamos consumiendo.

Esta clase comienza definiendo una serie de referencias a la base de datos de Firebase, de esta manera luego podemos acceder a estas referencias utilizando dichas variables.

```
15 class DatabaseService {
16
17     //ID de el usuario que está ejecutando la aplicacion
18     var userID = Auth.auth().currentUser!.uid
19
20     func reloadUserID() {
21         userID = Auth.auth().currentUser!.uid
22     }
23
24     static let _shared = DatabaseService()
25
26     static var shared: DatabaseService {
27         return _shared
28     }
29
30     //MARK: - Refecencias a la BD
31
32     ///Referencia a la base de datos principal
33     var mainRef: DatabaseReference! {
34         return Database.database().reference()
35     }
36     ///Referencia a las recetas
37     var recetasRef: DatabaseReference {
38         return mainRef.child(FIR_CHILD_RECERAS)
39     }
```

Figura 8.16: Implementación de las referencias a la base de datos

Posteriormente nos hemos definido una API que es la que utilizan los controladores para realizar las peticiones a Firebase.

```
401     func updateFavReceta(recetaID: String, newNumFav: Int) {
402         self.recetasRef.child(recetaID).updateChildValues(["numFav" : newNumFav])
403     }
544     func addRecetaToCategoria(recetaID: String, categoriaID: String) {
545         let atributos = [ recetaID : true] as [String : Any]
546         self.categoriasRef.child(categoriaID).child("recetas").updateChildValues(atributos)
547
548     }
549
550     func removeCategoriaFromReceta(recetaID: String, categoriaVieja: String) {
551
                self.categoriasRef.child(categoriaVieja).child("recetas").child(recetaID)
                .removeValue()
552         self.recetasRef.child(recetaID).child("categoria").removeValue()
553
554     }
```

Figura 8.17: Parte de la implementación de la API que se ofrece a los controladores

Por último tenemos definidos una serie de métodos que se utilizan en esta propia clase.

```
187     private func getListRecetasFrom(ref: DatabaseReference, completion: @escaping (Bool,
            [Receta]?) -> Void) {
188         ref.observeSingleEvent(of: .value) { (DataSnapshot) in
189             var misRecetas = [Receta]()
190             if let recetasBuscar = DataSnapshot.value as? Dictionary <String, AnyObject> {
191                 self.recetasRef.observeSingleEvent(of: .value, with: { (DataSnapshotAll) in
192                     if let recetas = DataSnapshotAll.value as? Dictionary <String,
                            AnyObject> {
193                         var numRecetasEsperadas = recetasBuscar.count - 1
194                         var isEmpty = true
195                         for recetaBuscar in recetasBuscar.keys {
196                             if let recetaJSON = recetas[recetaBuscar] {
197                                 if isEmpty {
198                                     isEmpty = false
199                                 }
200                                 self.getRecetaClassFrom(recetaID: recetaBuscar, recetaJSON:
                                        recetaJSON, completion: { (error, receta) in
201                                     if error == nil {
202                                         misRecetas.append(receta!)
203                                     } else {
204                                         numRecetasEsperadas -= 1
205                                     }
206                                     if numRecetasEsperadas == misRecetas.count {
207                                         completion(true, misRecetas)
208                                     }
209                                 })
210                             } else {
211                                 if recetaBuscar != "vacio" {
212                                     numRecetasEsperadas -= 1
213                                     if numRecetasEsperadas == misRecetas.count {
214                                         completion(true, misRecetas)
215                                     }
216                                 }
217 
218                             }
219                         }
220                         if isEmpty {
221                             completion(true, misRecetas)
222                         }
223                     }
224                 })
225             }
226         }
227     }
```

Figura 8.18: Implementación de métodos privados que se usan dentro de la clase

8.7 Interfaz Gráfica

Esta carpeta contiene algunas de las piezas de la interfaz gráfica, por ejemplo, la información que se pinta en las celdas de las tablas o el diseño customizado de algunos elementos de la pantalla como pueden ser botones, textfields, etc.

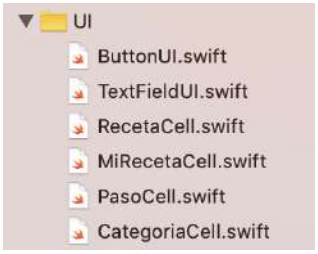

Figura 8.19: División de las carpetas

Vamos a mostrar algunos ejemplos de implementación de estos ficheros:

```
9  import UIKit
10
11 class PasoCell: UITableViewCell {
12
13     @IBOutlet weak var textoLabel: UILabel!
14
15     @IBOutlet weak var view: UIView!
16     @IBOutlet weak var numeroPasoLabel: UILabel!
17     @IBOutlet weak var botonAbajo: BotonUI!
18     @IBOutlet weak var botonArriba: BotonUI!
19     func updateCell(paso: String, posicion: Int, showArriba: Bool, showAbajo: Bool){
20         if posicion % 2 == 0 {
21             self.view.backgroundColor =
22             self.botonArriba.backgroundColor =
23             self.botonAbajo.backgroundColor =
24
25         } else {
26             self.view.backgroundColor =
27             self.botonArriba.backgroundColor =
28             self.botonAbajo.backgroundColor =
29         }
30         view.layer.cornerRadius = 10
31         self.numeroPasoLabel.text = "\(posicion+1)."
32         self.textoLabel.text = paso
33         botonArriba.posicion = posicion
34         botonAbajo.posicion = posicion
35         if showArriba {
36             botonArriba.isHidden = false
37         } else {
38             botonArriba.isHidden = true
39         }
40
41         if showAbajo {
42             botonAbajo.isHidden = false
43         } else {
44             botonAbajo.isHidden = true
45         }
46     }
47
48 }
```

Figura 8.20: Implementación de la celda de pasos de la receta

8.8 Controladores

Esta carpeta contiene todos los controladores de la aplicación que como ya hemos visto son los intermediarios y encargados de gestionar el flujo de la información entre los modelos y las vistas.

La siguiente imagen muestra los distintos controladores que tenemos en nuestro sistema.

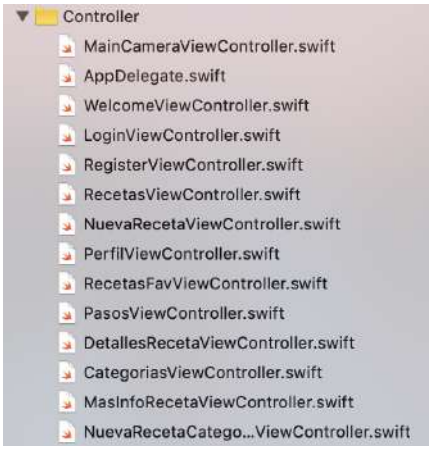

Figura 8.21: Estructura de los controladores

A continuación, vamos a ver algunos métodos y características interesantes de algunos de nuestros controladores.

Firebase nos ofrece el patrón de diseño observador de una manera muy sencilla. Tan solo tenemos que indicar en nuestro controlador qué referencia en la base de datos queremos estar a la escucha, de forma que, cuando cambie algún dato de esta referencia, se ejecute la funcionalidad que tenemos en nuestro código.

Nosotros lo hemos utilizado bastante para saber cuando actualizar datos en la interfaz gráfica, por ejemplo, cuando se ha publicado una nueva receta, poder actualizar el listado y así mostrarla al usuario.

Como se puede observar, los observadores son creados en los métodos del ciclo de vida, en concreto en el viewDidLoad, que se ejecuta la primera vez que se abre esta pantalla. De esta manera, solo se crea un único observador.

```
33      override func viewDidLoad() {
34          DatabaseService._shared.recetasRef.observe(.childChanged) { (data) in
35              self.hanCambiadoRecetas = true
36          }
```

Figura 8.22: Implementación un observador

Destacamos que nuestra respuesta a este evento es solamente poner una variable booleana a true, indicando que se han actualizado las recetas. De forma que, solamente cuando vuelva a aparecer esta pantalla, cargamos si hacen falta, las nuevas recetas.

```
192     override func viewDidAppear(_ animated: Bool) {
193         if self.hanCambiadoRecetas {
194             self.hanCambiadoRecetas = false
195             self.loadRecetas()
196         }
197     }
```

Figura 8.23: Implementación un observador

Otro aspecto muy interesante de nuestros controladores es la capacidad de cargar unos u otros datos dependiendo de la pantalla de la que procedamos. Esto nos permite reutilizar vistas y que la información sea dinámica.

Veamos un sencillo ejemplo de uso de este mecanismo.

La siguiente figura corresponde a la implementación del controlador de las categorías. Su función es la de mostrar un listado con las distintas categorías que existen y que contienen recetas.

```
12  class CategoriasViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {
13
14
15      @IBOutlet weak var categoriasTableView: UITableView!
16      private var categorias = [Categoria]()
17
18      override func viewDidLoad() {
19          super.viewDidLoad()
20
21          self.loadCategorias()
22          self.categoriasTableView.refreshControl = UIRefreshControl()
23          self.categoriasTableView.refreshControl?.attributedTitle = NSAttributedString(string:
                "Tira para recargar las categorias")
24          self.categoriasTableView.refreshControl?.addTarget(self, action:
                #selector(CategoriasViewController.loadCategorias), for: .valueChanged)
25      }
26
27      func numberOfSections(in tableView: UITableView) -> Int {
28          tableView.estimatedRowHeight = 50
29          tableView.rowHeight = UITableView.automaticDimension
30          tableView.separatorStyle = .none
31
32          return 1
33
34      }
35
36      func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
37          return self.categorias.count
38      }
39
40      func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
            UITableViewCell {
41          let categoria = self.categorias[indexPath.row]
42          let cellID = "categoriaCell"
43                  let cell = tableView.dequeueReusableCell(withIdentifier: cellID, for:
                        indexPath) as! CategoriaCell
44
45          cell.selectionStyle = .none
46          cell.updateCategoriaUI(categoria: categoria)
47          return cell
48      }
```

```
50      @objc func loadCategorias() {
51          ProgressHUD.show()
52          DatabaseService.shared.getListCategorias { (sucess, listCategorias) in
53              if sucess {
54                  let categorias: [Categoria] = listCategorias!.filter({ (c) -> Bool in
55                      c.recetasID.count > 1
56                  })
57                  self.categorias = categorias
58                  self.categoriasTableView.reloadData()
59                  self.categoriasTableView.refreshControl?.endRefreshing()
60              }
61              ProgressHUD.dismiss()
62
63          }
64      }
65
66      @IBAction func elegidaCategoria(_ sender: BotonUI) {
67          if let tituloCategoria = sender.currentTitle {
68
69              let categoriaElegida = self.categorias.filter { (categoria) -> Bool in
70                  categoria.nombre.capitalizingFirstLetter() ==
                        tituloCategoria.capitalizingFirstLetter()
71              }
72
73              if categoriaElegida.count > 0 {
74                  self.performSegue(withIdentifier: "showListadoRecetas", sender:
                        categoriaElegida[0])
75              }
76
77          }
78      }
79
80
81      override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
82          if segue.identifier == "showListadoRecetas" {
83              if let categoria = sender as? Categoria {
84                  let destinationViewController = segue.destination as! RecetasViewController
85                  destinationViewController.categoria = categoria
86              }
87          }
88      }
89
90  }
```

Figura 8.24: Implementación del controlador de categorías

Si ponemos el foco en la función prepare (lineas 81-88) esta es la encargada de cargar la siguiente pantalla, más concretamente la pantalla del listado de recetas, enviándole una serie de datos. Estos datos que le estamos enviando es una categoría, en concreto, la categoría que ha seleccionado el usuario (véase la función elegidaCategoria, lineas 52-64), que, como ya vimos en su modelo, una categoría contiene un titulo, una imagen y un listado con los ID de las recetas que pertenecen a esa categoría.

Ahora vamos a pasar a ver como, en el controlador de las recetas, cargamos unas u otras.

En dicho controlador, tenemos un método llamado loadRecetas que lo único que hace es chequear si tiene una variable denominada categoría, que justamente, es la que le ha pasado el controlador anterior. De modo que, si esta no es nula, ejecutamos el método que carga las recetas de una categoría y sino llamamos al método que carga todas las recetas publicadas.

```
201     @objc fileprivate func loadRecetas() {
202         self.tituloLabel.text = self.titulo.capitalizingFirstLetter()
203         if categoria == nil { //Queremos mostrar todas las recetas
204             self.atrasButton.isHidden = true
205             self.loadAllRecetas()
206         } else { //Solo queremos ver las recetas de esa categoria
207             self.atrasButton.isHidden = false
208             self.tituloLabel.text = self.categoria?.nombre.capitalizingFirstLetter()
209             self.loadRecetasFromCategoria(categoria: categoria!)
210         }
211     }
```

Figura 8.25: Implementación de la carga de recetas

Veamos como serían estos dos métodos.

```
275     func loadRecetasFromCategoria(categoria: Categoria) {
276         ProgressHUD.show()
277         DatabaseService.shared.getRecetasFromCategoria(categoria: categoria) { (sucess,
                listRecetas) in
278             if sucess {
279                 self.reloadRecetas(listRecetas: listRecetas!)
280             }
281             ProgressHUD.dismiss()
282         }
283     }
```

Figura 8.26: Implementación de la carga de recetas dada una categoría

```
264     func loadAllRecetas() {
265         ProgressHUD.show()
266         DatabaseService.shared.getAllRecetas { (sucess, listRecetas) in
267             if sucess {
268                 self.reloadRecetas(listRecetas: listRecetas!)
269             }
270             ProgressHUD.dismiss()
271         }
272
273     }
```

Figura 8.27: Implementación de la carga de recetas publicadas

Como se ha podido observar, estos métodos lo único que hacen es ejecutar un método u otro del servicio de base de datos. Y, si todo sale bien, llaman al método reloadRecetas, que es el encargado de actualizar la vista con las nuevas recetas.

```
285     func reloadRecetas(listRecetas: [Receta]) {
286         self.recetas = getRecetasByFechaCreacion(recetas: listRecetas)
287         if let text = self.filterTextField.text {
288             self.filterByName(name: text)
289         }
290         self.aplicarFiltrosSeleccionados()
291         self.recetasTableView.refreshControl?.endRefreshing()
292     }
```

Figura 8.28: Implementación de la carga de recetas en la vista

Esta técnica se ha reutilizado a lo largo de la aplicación, de forma que, nos ha permitido con un solo desarrollo aprovechar al máximo el trabajo realizado.

Ya por finalizar con los controladores, nos gustaría mostrar también una técnica que hemos empleado para que, cuando escribamos, por ejemplo, cuando estamos poniendo el titulo de una receta, la parte de la vista se eleve y de esta forma el teclado no oculte los elementos de la interfaz.

```
262 extension NuevaRecetaViewController: UITextViewDelegate {
263     func textViewDidBeginEditing(_ textView: UITextView) {
264 
265         self.boxHeight.constant = 40 + 200
266         self.view.layoutIfNeeded()
267 
268         if textView.text == placeholder {
269             textView.text = ""
270             textView.textColor = textViewColor
271         }
272 
273         textView.becomeFirstResponder()
274     }
275 
276     func textViewDidEndEditing(_ textView: UITextView) {
277         self.boxHeight.constant = 40
278 
279         self.view.layoutIfNeeded()
280         if textView.text.isEmpty {
281             textView.text = placeholder
282             textView.textColor = placeholderColor
283         }
284     }
285 }
```

Figura 8.29: Implementación la modificación de las restricciones en tiempo real

CAPÍTULO 9
Conclusiones

Este es el último capitulo de memoria. Vamos a comentar de forma subjetiva los objetivos alcanzados, las posibles extensiones y mejoras que se pueden llevar a cabo tomando este trabajo como base, así como una breve sección personal donde comentaré mi opinión personal en el desarrollo de este proyecto.

9.1 Evaluación de la metodología

En una fase temprana, cuando no sabíamos qué funcionalidades podría tener la aplicación nos apoyamos en la realización de entrevistas a distintas personas para ver cuales eran las características que más deseaban tener en una aplicación de este estilo. Esto nos proporcionó una idea y una visión de lo que teníamos que realizar.

Posteriormente cuando ya estaba todo estudiado y sabíamos exactamente lo que incorporaríamos en nuestra app, se decidió emplear una metodología incremental. Una frase que me gusta mucho y que se escucha en el mundo de las startups es *El pez grande ya no se come al pequeño, sino que el rápido se come al lento* la cual ha sido nuestra filosofía al desarrollar esta aplicación, donde, como resultado, hemos dispuesto de un MVP (mínimo producto viable) a las 2 semanas de desarrollo, pudiéndolo lanzar al mercado desde un primer momento. Esto nos ha permitido ir probando la aplicación, ver que tal funcionaba todo y poco a poco ir añadiendo nuevas funcionalidades a nuestro sistema.

Como resultado, hemos realizado una aplicación que promete lo esperado por el cliente la cual es capaz de evolucionar para ir incorporando futuras novedades conforme se reclamen.

9.2 ¿Qué se ha alcanzado?

Se ha conseguido desarrollar una aplicación completamente nueva en un lenguaje también nuevo para el alumno, la cual es, en mi opinión, escalable y reutilizable. Además, se ha logrado diseñar una base de datos mediante tecnologías desconocidas como pueda ser una base de datos no relacional, así como un almacenamiento y un sistema de autentificación mediante los servicios de Firebase, conectándolos con éxito a nuestro sistema.

También se ha tenido mucha consideración en todo momento con el rendimiento que pueda tener nuestro sistema, intentando reducir los costes al máximo posible.

Con todo esto, y viendo los objetivos principales que vimos en el capítulo de la introducción, puedo afirmar que se han cumplido todos los propósitos que nos planteamos.

9.3 Posibles mejoras y ampliaciones

Cuando comencé a pensar en mi trabajo fin de grado la idea inicial era la de realizar una aplicación que permitiera a los usuarios obtener dietas personalizadas, todo en función de distintos parámetros como las kilocalorías que querían consumir, si estaban realizando algún tipo de entrenamiento físico...

Debido a que este proyecto era muy ambicioso decidí dividirlo en tres proyectos. El primero de ellos es este mismo trabajo, una aplicación que permita compartir recetas de cocina. De este modo se tendría una base inicial, con usuarios y un conjunto de recetas de partida. El segundo se centraría en crear un backend específico y no depender de Firebase, de modo que se pueda crear una una base de datos propia con distintos ingredientes, poniendo sus valores energéticos y otros atributos de estos para así incorporarlos a esta aplicación, de manera que permita a los usuarios poder filtrar por ingredientes, buscar por Kcal, alérgenos... Por último vendría la última pieza, la cual sería un modelo matemático, diseñado con técnicas de aprendizaje automático, que dado un tipo de entrenamiento o de ejercicios que estemos realizando, nos ofrezca una dieta personalizada utilizando las recetas y la información de los ingredientes que la conforman.

Como ya hemos dicho, este es un proyecto muy ambicioso, y nos gustaría poner a disposición de otros alumnos esta idea y trabajo para que puedan continuar con las siguientes fases.

9.4 Dificultad del proyecto

No os voy a engañar, este trabajo me ha costado mucho sudor llevarlo a cabo. No por la complejidad del mismo sino por la forma de trabajar que he seguido. Al inicio, cuando comencé viendo que era Swift y escribiendo mis primeras líneas de código en sus playground, no me supuso mucho trabajo, al fin y al cabo todos los lenguajes tienen los mismos mecanismos, solo cambia la sintaxis y poco más, pero luego, cuando me he adentrado en todo el tema de los closures, las promesas y llamadas asíncronas ahí ha empezado lo bueno.

También me he peleado duramente adaptando el diseño de la interfaz a todas las pantallas, donde he intentado reutilizar al máximo posible todas las funcionalidades que iba desarrollando y así ha quedado demostrado en la reutilización de las vistas.

En general me he encontrado con muchos problemas, desde los de arquitectura y diseño de la base de datos, hasta los niveles más bajo de implementación, pasando largas horas puliendo todos los detalles para conseguir una aplicación exitosa.

9.5 Opinión personal del trabajo

A pesar de que nunca había tocado nada de desarrollo en dispositivos móviles siempre había existido una parte de mí como desarrollador que quería experimentarlo. Nunca me había encargado de realizar un desarrollo completo solo, empezando por la arquitectura, pasando por el diseño de la base de datos, la implementación y terminando en las imágenes que iba a utilizar. Este proyecto lo he querido plantear como un reto personal, donde poder demostrarme a mí mismo de lo que era capaz de realizar con todo lo que había aprendido.

A pesar de las dificultades que he mencionado en la sección anterior considero que las he superado todas, logrando con ello tener una aplicación de la cual me siento verdaderamente orgulloso.

Es por ello que opino que este proyecto me ha ayudado a mejorar y a enriquecerme no solo a nivel profesional con todas las distintas tecnologías que he ido aprendiendo, sino también en lo personal, ya que me ha requerido una dedicación y un gran esfuerzo en mi día a día.

Apéndice A. Manual de usuario

En esta sección vamos a proceder a describir las distintas funcionalidades y como se deben de utilizar por parte del usuario que se descargue nuestra aplicación.

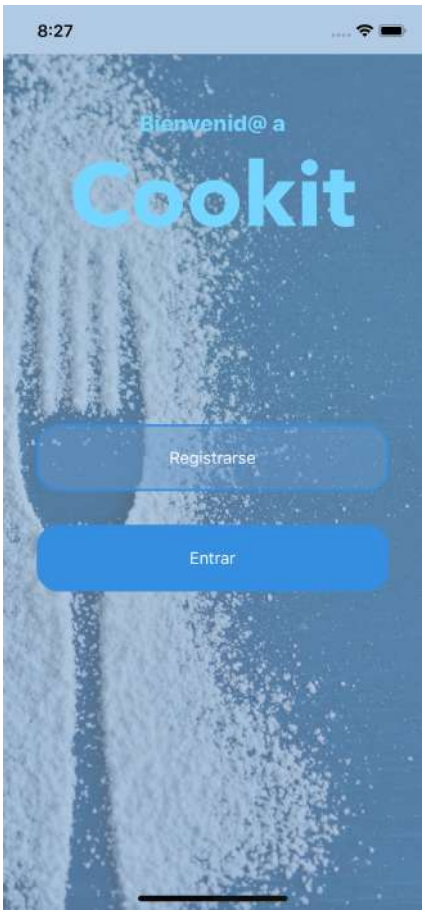

Figura 9.1: Pantalla inicial en iPhone

Esta es la pantalla de inicio de nuestra aplicación. Aquí el usuario puede elegir entre crearse una nueva cuenta o entrar en nuestro sistema.

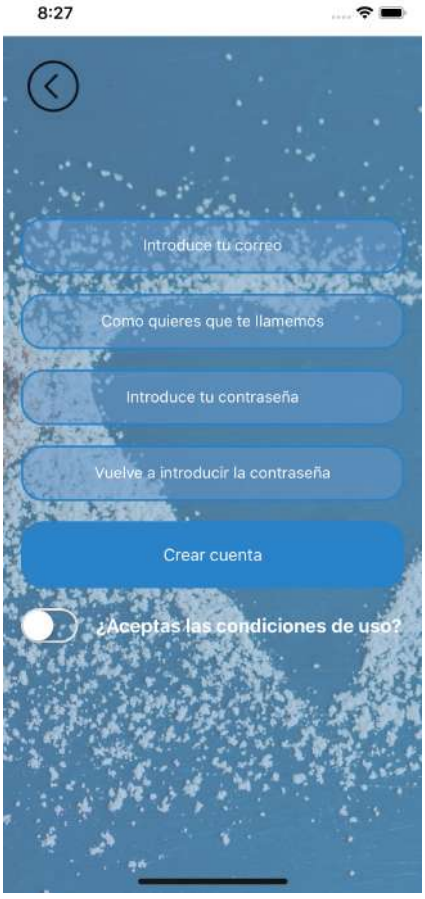

Figura 9.2: Pantalla de registro en iPhone

Esta pantalla corresponde al registro de usuarios. Ofrece una serie de campos que el usuario debe rellenar para poder registrarse satisfactoriamente en nuestro sistema. Entre ellos se encuentra un correo electrónico (el cual no debe de estar registrado previamente), una contraseña (que debe tener un mínimo de 8 dígitos para ser segura) y una sección para que el usuario introduzca su nombre público.

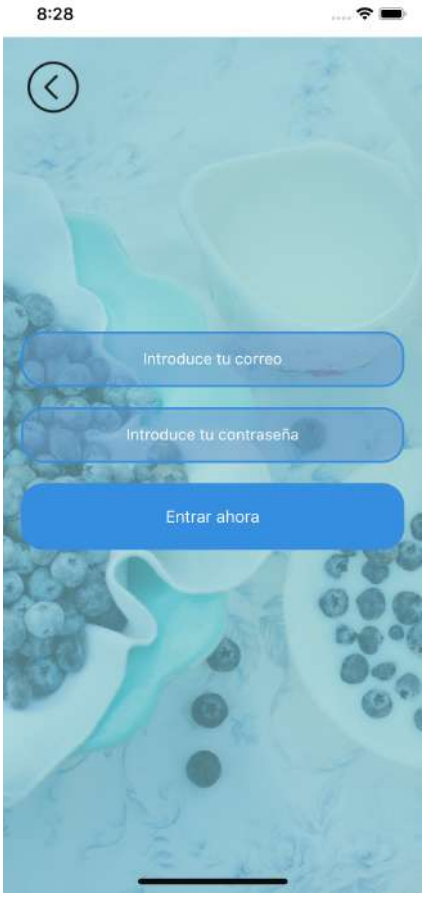

Figura 9.3: Pantalla de login de usuarios en iPhone

En esta vista un usuario puede poner sus credenciales para autentificarse en el sistema. En el caso de que esta autentificación se produzca de manera satisfactoria, accedería al listado de recetas, en otro caso se quedaría en esta misma pantalla.

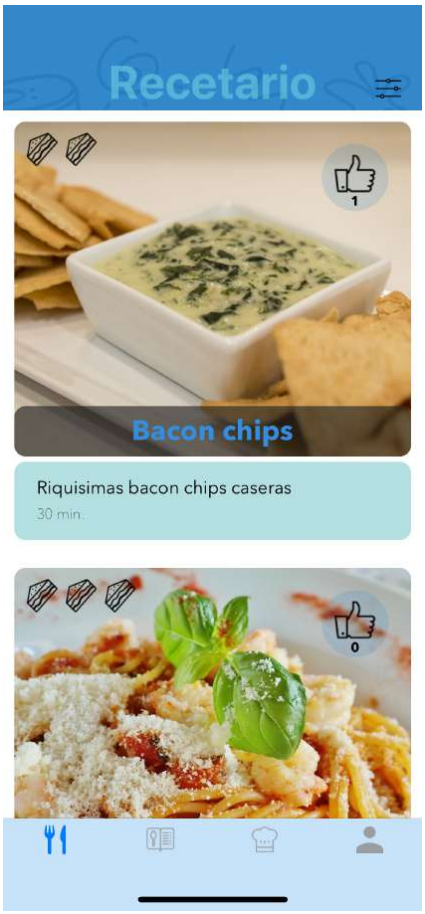

Figura 9.4: Pantalla de inicio en iPhone

Aquí es donde se muestran todas las recetas que tenemos en nuestra app. Cada una de ellas se visualizan mediante un título, una descripción, el tiempo de elaboración expresado en minutos y la imagen de la misma.

En la parte superior izquierda cuenta con unos sándwiches que indican el nivel de dificultad de la receta, siendo uno el más fácil y tres el más complicado. Si nos fijamos también en la parte superior derecha veremos un pulgar con un número abajo, esto indica la cantidad de personas a las que le gusta esta receta.

Figura 9.5: Pantalla de inicio con vista de filtros en iPhone

Por otro parte, si el usuario pulsa sobre el menú de hamburguesa disponible arriba y a la derecha se mostrará un menú adicional que corresponde al menú de filtrado. Aquí podrá ordenar las recetas por número de favorito, fecha de publicación, alfabéticamente o directamente buscando por el nombre de la receta.

Se podrá ir deslizando hacia abajo la pantalla para visualizar nuevas recetas. En el caso de que pulse sobre una de ellas, se abriría la vista detallada.

Figura 9.6: Vista de una receta en iPhone

Esta pantalla contiene toda la información disponible para realizar la receta. Como podemos comprobar contiene una imagen de la receta así como los pasos en el orden correcto para poder elaborarla correctamente. En la parte inferior izquierda de la imagen tenemos tres botones. Los dos primeros de ellos corresponden al borrado y a la edición del plato, los cuales solo estarán visibles en el caso de que el usuario que esté viendo la receta sea el creador de la misma. El último botón sirve para indicar que le gusta la receta. Cuando se pulsa sobre este botón el corazón se rellena, indicando así que esa receta se tiene como favorito. En el caso de que a un usuario ya no le guste la receta podrá volver a pulsar sobre este botón para así indicarlo.

Para ver más información interesante sobre la receta un usuario puede pulsar sobre la imagen de la misma.

Figura 9.7: Vista de más información de la receta en iPhone

Contiene información interesante sobre la receta como puede ser una descripción, los ingredientes que hacen falta, así como quien es su creador. En el caso de que se pulse sobre la imagen se abrirá el perfil de dicho dueño.

Utilizando esta vista, el usuario podrá filtrar por categorías los distintos platos. Una vez pulse sobre una de ellas se mostrará la pantalla del listado de recetas pero filtradas con la categoría seleccionada.

Figura 9.8: Pantalla de búsqueda por categorías en iPhone

Esta es la vista de perfil de los usuarios. Arriba a la izquierda contiene un botón, que sirve para cerrar la sesión actual. También se puede visualizar la imagen y el nombre público que tenga el usuario, en el caso de que quiera cambiarlos simplemente tendrá que pulsar sobre estos y poner la nueva información.

En esta pantalla también contamos con dos listas las cuales corresponden a las recetas creadas y favoritas del usuario que estamos viendo el perfil. En el caso de que una de ellas estuviera vacía se ocultaría para así no mostrar información irrelevante.

Figura 9.9: Pantalla de perfil de un usuario en iPhone

Figura 9.10: Pantalla inicial de creación de una receta en iPhone

Esta es la pantalla inicial de la creación de nuevas recetas. Simplemente contiene un botón en su inferior, el cual si es pulsado se mostraría la siguiente digura.

Figura 9.11: Pantalla 1 de creación de una receta en iPhone

Aquí el usuario sería capaz de darle el nombre a la receta así como añadir una descripción. Además, si se pulsa sobre la imagen podrá realizar una fotografía de esta receta que quiere elaborar. En su inferior contiene dos botones, el de retroceder y el de continuar con la elaboración. En el caso de que pulse sobre el segundo se mostraría la siguiente pantalla que vamos a ver.

Figura 9.12: Pantalla 2 de creación de una receta en iPhone

En esta vista podrá introducir información como los minutos de elaboración, los ingredientes que se necesitan, la pertenencia o no de la receta a una de las categorías y la dificultad que tiene dicho plato. Igual que en la pantalla anterior si pulsa segundo botón de abajo pasaríamos a la creación de pasos.

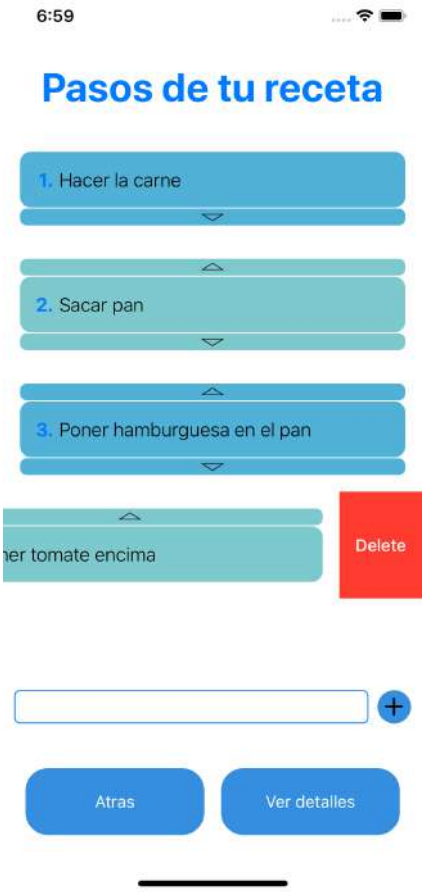

Figura 9.13: Pantalla 3 de creación de una receta en iPhone

Aquí podrá ir añadiendo pasos, cambiar el orden o si se ha equivocado en uno de ellos borrarlo.

En el caso de que se pulsara sobre el segundo botón inferior nuevamente, se abriría la vista detallada de la receta con toda la información. Esta pantalla en su inferior contaría con un botón que le permite al usuario publicar la receta en la aplicación.

Apéndice B. Mockups

Figura 9.14: Mockup pantalla inicial

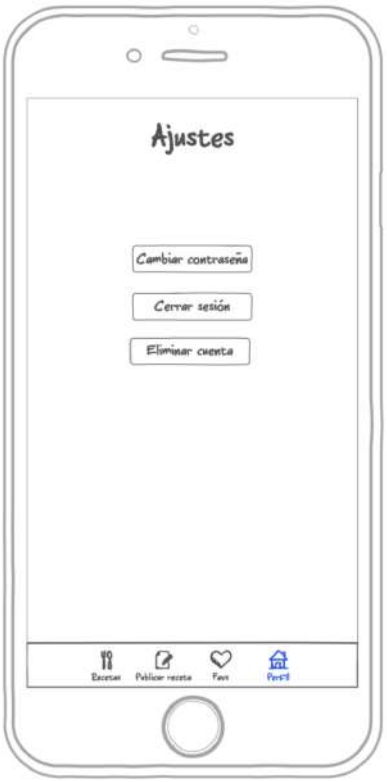

Figura 9.15: Mockup pantalla ajustes

Figura 9.16: Mockup pantalla nueva receta 1

Figura 9.17: Mockup pantalla nueva receta 2

Figura 9.18: Mockup pantalla nueva receta 3

Figura 9.19: Mockup pantalla recetas favoritas

Apéndice C. Interfaz vista desde un iPad

Figura 9.20: Pantalla inicial en iPad

Figura 9.21: Pantalla de registro en iPad

Figura 9.22: Pantalla de login de usuarios en iPad

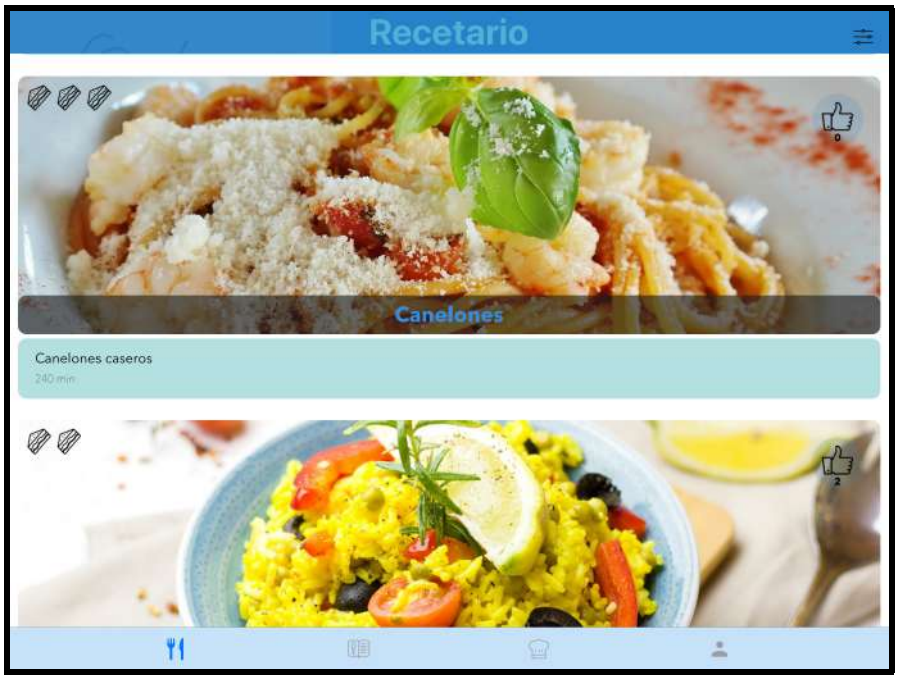

Figura 9.23: Pantalla de inicio en iPad

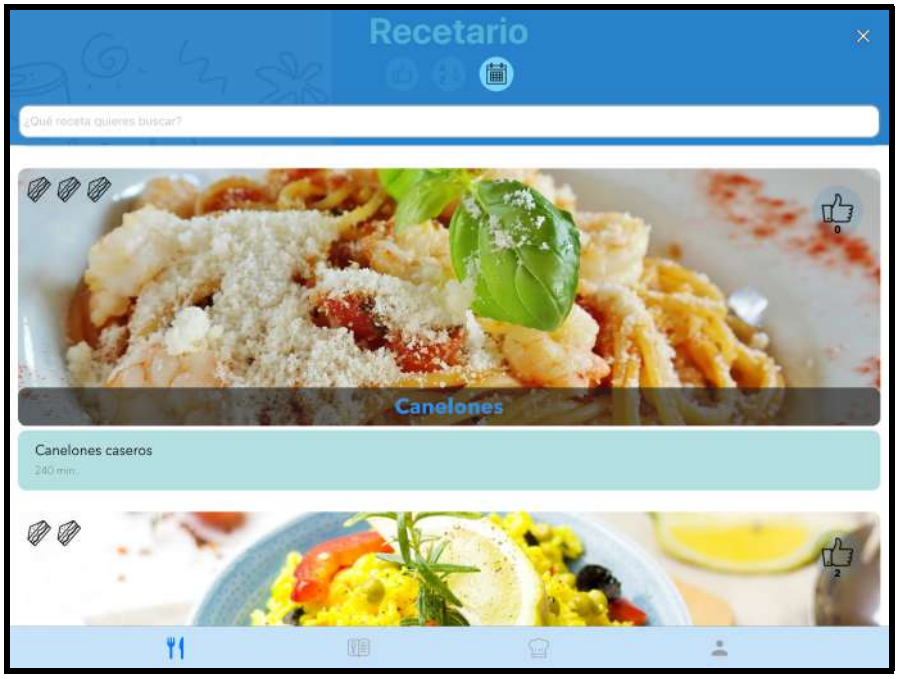

Figura 9.24: Pantalla de inicio con vista de filtros en iPad

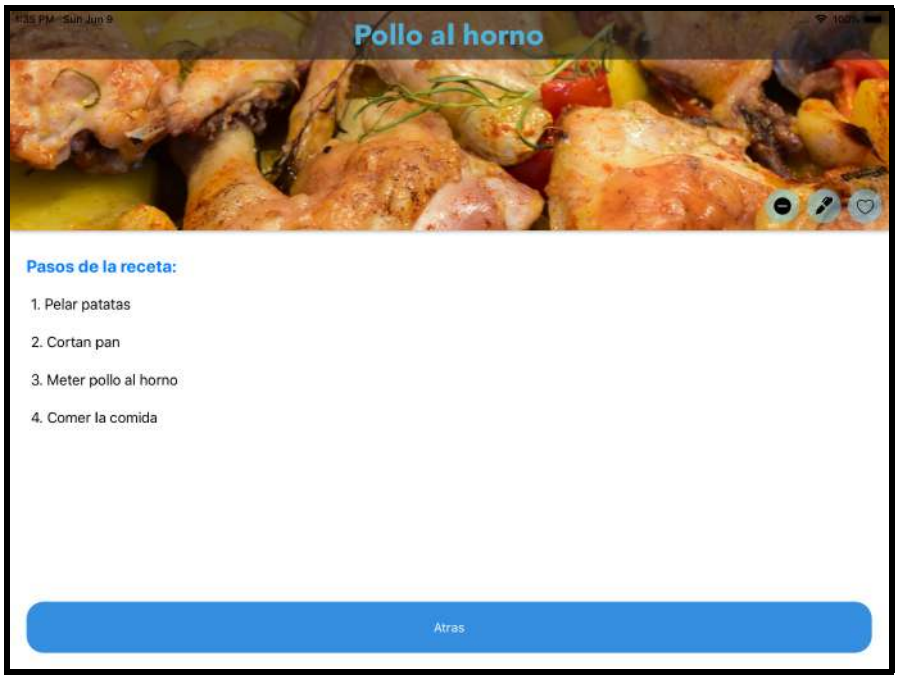

Figura 9.25: Vista de una receta en iPad

Figura 9.26: Vista de más información de la receta en iPad

Figura 9.27: Pantalla de búsqueda por categorías en iPad

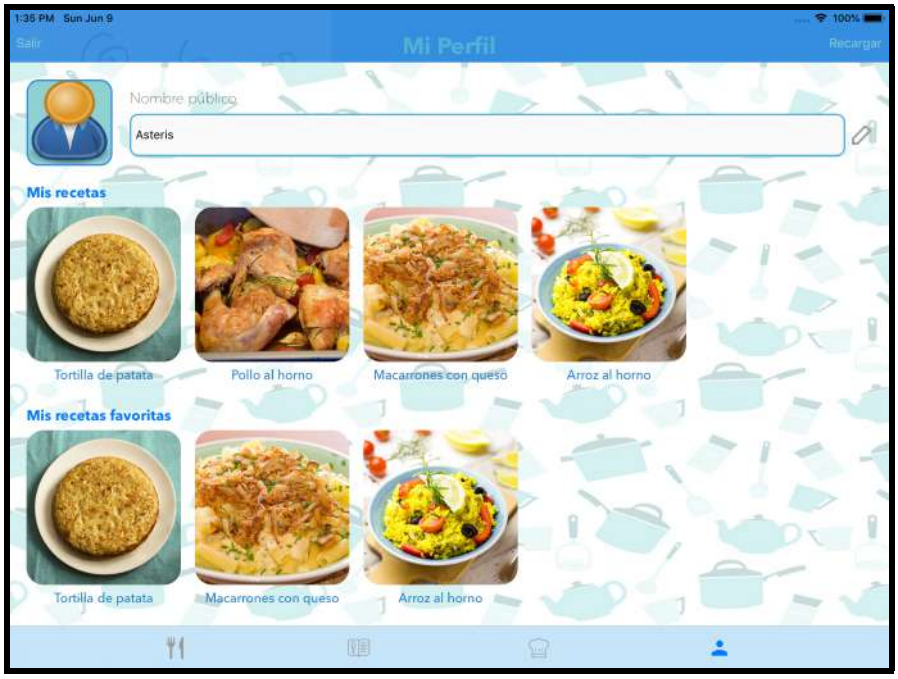

Figura 9.28: Pantalla de perfil de un usuario en iPad

Figura 9.29: Pantalla inicial de creación de una receta en iPad

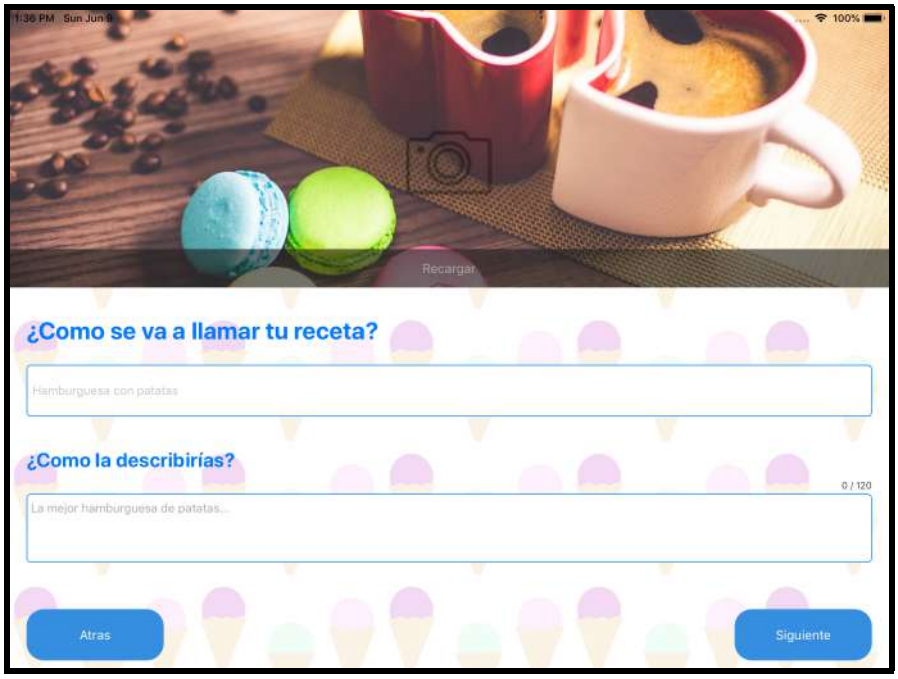

Figura 9.30: Pantalla 1 de creación de una receta en iPad

Figura 9.31: Pantalla 2 de creación de una receta en iPad

Figura 9.32: Pantalla 3 de creación de una receta en iPad

Bibliografía

[1] Sesma, M. *VERSUS: Apps híbridas VS Apps Nativas.* 10 de julio de 2017.
Disponible en
https://www.paradigmadigital.com/dev/versus-apps-hibridas-vs-apps-nativas/

[2] Fernández, J. C. *Así es Electron: el último intento del desarrollo híbrido en una carrera que ya han ganado las apps nativas.* 23 de Enero de 2019.
Disponible en https://www.applesfera.com/analisis/asi-electron-ultimo-intento-desarrollo-hibrido-carrera-que-han-ganado-apps-nativas

[3] Nubuser *Desarrollo de apps nativas vs apps híbrida.s* 14 de Noviembre de 2018.
Disponible en https://nubeser.com/desarrollo-apps-nativas-vs-apps-hibridas/

[4] Ulmeher, J. M. *Análisis De Android, El Sistema Operativo Para Móviles De Google.*
Disponible en https://www.ibertronica.es/blog/tutoriales/android-sistema-operativo/

[5] Nieto J. G. *Así es como Android se ha comido el mercado en diez años*
Disponible en https://www.xatakamovil.com/sistemas-operativos/asi-como-android-se-ha-comido-mercado-diez-anos

[6] Apple, iOS.
Disponible en https://developer.apple.com/ios/

[7] SensorTower. *Million dollar publishers.* 20 de diciembre de 2018
Disponible en https://sensortower.com/blog/million-dollar-publishers-2018

[8] ROGER, S. Pressman. *Ingeniería de Software: Un enfoque práctico.* McGraw Hill, 2002.

[9] Marlady Ortiz. *Modelo incremental*
Disponible en http://isw-udistrital.blogspot.com/2012/09/ingenieria-de-software-i.html

[10] José M. Drake. *Análisis de requisitos y especificacion de una aplicación.* Santander 2008
Disponible en https://www.ctr.unican.es/asignaturas/Ingenieria_Software_4_F/Doc/M3_08_Especificacion-2011.pdf

[11] RAGHAVAN, Sridhar; ZELESNIK, Gregory; FORD, Gary. *Lecture notes on requirements elicitation.* CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1994.

[12] Apple, Xcode IDE.
Disponible en https://developer.apple.com/xcode/ide/

[13] Apple, Xcode Interface Builder.
Disponible en https://developer.apple.com/xcode/interface-builder/

[14] Cocoapods.
Disponible en https://cocoapods.org/

[15] Apple, Swift.
Disponible en https://developer.apple.com/swift/

[16] Draw.io
Disponible en https://about.draw.io/

[17] Google, Firebase.
Disponible en https://firebase.google.com/

[18] Google, Firebase AdMob.
Disponible en https://firebase.google.com/docs/admob/

[19] Google, Firebase Authentication.
Disponible en https://firebase.google.com/docs/auth/?hl=es-419

[20] Google, Firebase Realtime Database.
Disponible en https://firebase.google.com/docs/database/?hl=es-419

[21] Git.
Disponible en https://git-scm.com/

[22] Gomez, K. *Top 5 Metodologías de Desarrollo de Software* 27 de julio de 2017.
Disponible en
https://www.megapractical.com/blog-de-arquitectura-soa-y-desarrollo-de-software/metodologias-de-desarrollo-de-software

[23] Mozilla, JavaScript Object Notation.
Disponible en https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON

[24] IBM Knowledge Center, Patrón de diseño de modelo-vista-controlador.
Disponible en https://www.ibm.com/support/knowledgecenter/es/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/csdmvcdespat.htm

[25] Wikipedia, Patrón de diseño fachada.
Disponible en https://es.wikipedia.org/wiki/Facade_(patr%C3%B3n_de_dise%C3%B1o)

[26] Amazon, ¿Qué es NoSQL?.
Disponible en https://aws.amazon.com/es/nosql/

[27] Wikipedia, Arquitectura cliente-servidor.
Disponible en https://es.wikipedia.org/wiki/Cliente-servidor

[28] UPV, Enrique Hernández Orallo, El Lenguaje Unificado de Modelado (UML).
Disponible en http://www.disca.upv.es/enheror/pdf/ActaUML.PDF

[29] Apple, Librería AVCam.
Disponible en https://developer.apple.com/documentation/avfoundation/cameras_and_media_capture/avcam_building_a_camera_app

[30] Material design.
Disponible en https://material.io/design/

[31] Diseño centrado en el usuario, DCU.
Disponible en http://mpiua.invid.udl.cat/perfil-de-usuario-tecnica-personas/