UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Flatlife: gestiona tu vivienda compartida

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Saúl Blanco Villorejo

Tutor: David de Andres Martínez

Curso 2019-2020

Resumen

A día de hoy vivimos en una sociedad en la que más y más estudiantes llegan a la universidad, muchos de ellos compartiendo vivienda con más gente ya sea por temas económicos o porque no les gusta vivir solos. También jóvenes trabajadores que se mudan de su ciudad natal necesitan encontrar una vivienda en el lugar de destino. Sea como fuere, compartir vivienda es una situación común entre los jóvenes estudiantes y trabajadores cuya prioridad debe ser estudiar y trabajar. Es por ello que muchos de ellos se olvidan de las tareas que requiere una vivienda compartida, como limpiezas o compras compartidas. Cuando esto ocurre, el ambiente de compañerismo se pierde y las relaciones personales pueden empeorar. Este proyecto busca facilitar las labores de organización de aquellas tareas que pueden realizarse entre los compañeros de una vivienda compartida para que haya un ambiente agradable y que el rendimiento de los compañeros sea bueno.

Palabras clave: sociedad, vivienda compartida, compañerismo, React Native, aplicación móvil, Android, iOS

Abstract

Today we live in a society in which more and more students attend the university, many of them sharing housing with more people either for economic reasons or because they do not like to live alone. Also young workers who move from their hometown need to find housing at the destination. Be that as it may, sharing housing is a common situation among young students and workers whose priority must be to study and work. That is why many of them forget about the tasks that a shared home requires, such as cleaning or shared purchases. When this happens, the atmosphere of camaraderie is lost and personal relationships can worsen. This project seeks to facilitate the organization of those tasks that can be carried out among the companions of a shared house to keep a pleasant environment and the performance of the companions as high as possible.

Key words: society, shared housing, companionship, React Native, mobile application, Android, iOS

Índice general

Índice de figuras

Índice de tablas

Índice de fragmentos de código

CAPÍTULO 1
Introducción

Los seres humanos somo seres sociales, vivimos en compañía de otros seres humanos y trabajamos conjuntamente para conseguir cosas impresionantes. Además de trabajar y pasar el tiempo de ocio con otras personas, convivimos bajo un mismo techo donde a veces pasamos gran parte del tiempo y a veces no. Cuando esto ocurre se necesita una gran organización por parte de las personas que comparten techo para mantener una buena relación y un lugar en el que estar tranquilo y vivir. Una buena organización podría incluir mantener un calendario de limpieza, un registro de pagos y deudas y/o una lista con productos necesarios que se deben comprar. Por supuesto, una forma de llevar a cabo estas tareas de organización es coger bolígrafo y papel e ir apuntando todo según se necesite, pero si no hay una buena estructura o no se le dedica tiempo, al final acaba siendo contraproducente.

En España, desde el 2015 hasta el 2018 los precios de los alquileres ha sufrido un aumento considerable que provoca que los inquilinos no puedan costearse vivir solos y tengan que compartir vivienda [1, 2]. En 2017, un 33 % de las personas que alquilaron viviendas con otras eran estudiantes, formando una mayoría. Los motivos más abundantes para compartir vivienda son: "no puedo pagar un alquiler yo solo", "se adapta a lo que necesito" o "prefiero gastar dinero en otras cosas". Entre estudiantes, mantener una buena relación con los compañeros de piso es crucial para mantener un buen rendimiento. Además, es una etapa en la que muchos jóvenes dan el paso a empezar a vivir lejos de la vivienda familiar y que marca la vida de los mismos. Y no sólo ellos son los que comparten vivienda y necesitan mantener un buen clima, sino que también jóvenes empleados y profesionales que encuentran trabajo lejos de casa. Se habla tanto de jóvenes porque el grueso de personas que comparten vivienda se encuentra entre los 18 y los 35 años.

Actualmente vivimos una época en la cual la tecnología es un pilar clave tanto en la vida profesional como en la cotidiana. Una de las tecnologías más usadas actualmente son los teléfonos móviles, muchos de ellos llamados teléfonos inteligentes puesto que tienen funcionalidad de ordenador. Existe una vasta variedad de aplicaciones que incluyen desde una simple calculadora hasta aplicaciones que monitorizan tus pasos, pulsaciones, ejercicio realizado y proporcionan estadísticas de tu evolución. Tareas que antes se realizaban manualmente ahora todas se realizan mediante una aplicación. Muchas personas incluso se han vuelto adictas al teléfono móvil y han desarrollado un miedo a no disponer de él[1].

Dado el contexto, ¿Por qué no utilizar una aplicación de un teléfono móvil para la organización de una vivienda compartida?.

[1] https://www.sanitas.es/sanitas/seguros/es/particulares/biblioteca-de-salud/prevencion-salud/nomofobia.html

1.1 Motivación

Al comenzar el grado, tuve que dejar mi ciudad y buscar una nueva vivienda en la ciudad destino, Valencia. Allí, tuve que compartir vivienda con varios compañeros de piso, y aunque solíamos mantener un control sobre lo que hacíamos, notaba que no todo estaba apuntado debidamente. Desde el primer año siempre quería disponer de una aplicación que permitiera, de manera sencilla, tener una organización sobre lo que ocurría en el piso, ya fuera relacionado con los pagos, los productos comunes que debíamos comprar, las limpiezas, etc. Nunca utilicé ninguna y muchas cosas las apuntábamos en un papel. Mientras tanto, buscaba aplicaciones que pudiésemos utilizar. Las mejores siempre se especializaban en una tarea específica: pagos, limpiezas, listas de la compra, etc. Las que contenían más funcionalidad no satisfacían las necesidades buscadas. Algunas aplicaciones tenían funcionalidad básica de pago, otras, interfaces poco intuitivas o no funcionaban online, otras no incluían datos en tiempo real. Además, cuando hablaba con personas de mi edad que también compartían pisos, algunos no llevaban muy bien el tema de la organización y les hubiera gustado tener una aplicación para ese propósito.

Hasta ahora, en esta sección no se ha mencionado qué tipo de aplicación se buscaba, es decir, en qué dispositivos estaría disponible. La opción de una aplicación de escritorio se descartó desde un principio ya que se buscaba la facilidad de uso y que se pudiera acceder a ella cuando se quisiera (imagina tener que llevar un ordenador cuando vas a comprar porque tienes ahí apuntada la lista). Hoy día todo el mundo tiene un teléfono móvil, la mayoría con sistemas operativos iOS o Android, por lo que realizar una aplicación para esta plataforma sería un acierto. La posibilidad de implementar una aplicación web también estaba presente ya que ésta podría ser accedida tanto desde un ordenador como desde un móvil. No obstante, teniendo presente el auge de las aplicaciones híbridas para dispositivos móviles (mirar sección 4.1), usar un framework que permitiera realizar este tipo de aplicaciones también me ayudaría a tener nuevo conocimiento para un futuro.

Por estas razones comencé con este proyecto, pensando en todas aquellas personas que debían compartir piso, especialmente en los estudiantes universitarios, para que su convivencia fuera mucho más organizada, implicando una mejor relación entre compañeros.

1.2 Objetivos

La realización de este proyecto busca crear de una aplicación móvil que ayude a mejorar la organización entre los compañeros de una vivienda compartida. Esta aplicación no dependerá de ningún sistema externo, será autónoma.

1.3 Impacto esperado

Como ya se ha dicho, compartir vivienda es una tarea común entre los jóvenes y es importante que exista buena relación entre ellos. Para que una persona rinda al máximo debe de tener un estado de ánimo positivo, es decir, debe estar alegre y motivado. En el estado de ánimo de una persona influyen las relaciones personales y, como cuando convives con otras personas sueles coincidir con ellas en la vivienda, tener una buena relación con los compañeros de piso es muy importante para tener un estado de ánimo positivo. Así pues, Flatlife busca conseguir que estas relaciones entre compañeros de vivienda sean buenas.

1.4 Metodología

Para conseguir realizar el proyecto se ha seguido una estructura de trabajo cíclica que comprende tres fases: una fase de escritura o implementación, otra de corrección por parte del tutor del trabajo, David de Andrés, y una última fase de aplicación de las correcciones y mejoras. Este ciclo se ha repetido una o varias veces sobre cada bloque de trabajo.

Los bloques de trabajo de este proyecto han sido los siguientes: investigación de aplicaciones similares, especificación de requisitos, análisis, diseño, implementación y finalización de la memoria.

En la figura 1.1 se muestra el tiempo empleado en las diferentes actividades del proyecto mediante un diagrama de Gantt.

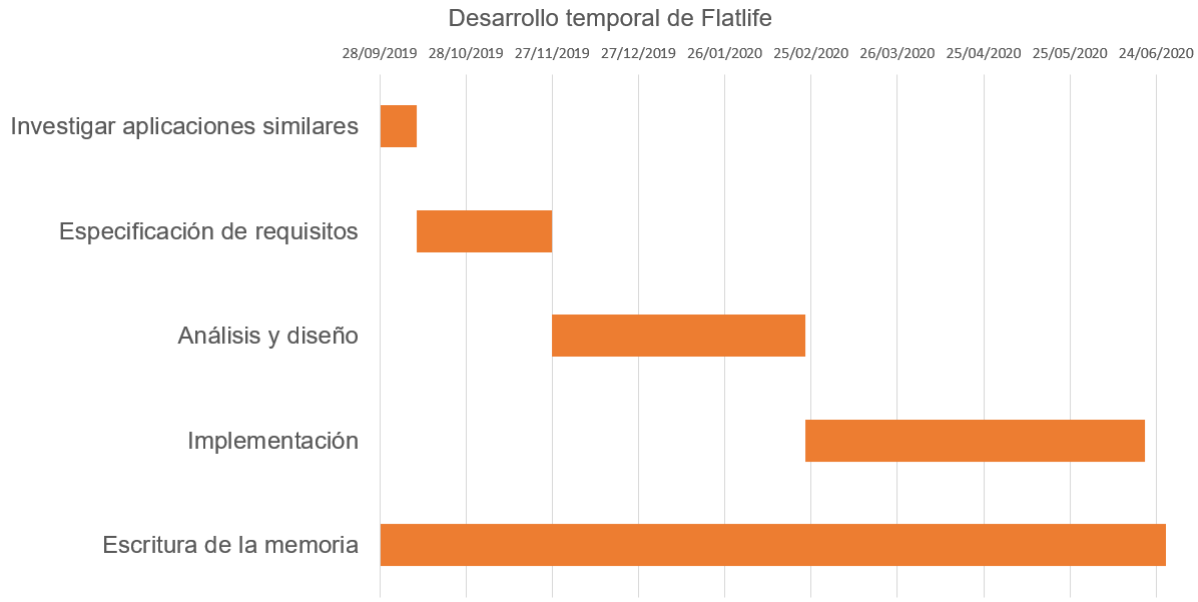


Figura 1.1: Diagrama de Gantt que muestra el trabajo realizado para completar el proyecto

1.5 Estructura de la memoria

Este documento está dividido en 9 capítulos, cuyos contenidos se describen en esta sección.

Un primer capítulo, *Introducción*, expone el por qué y el para qué de todo el trabajo que se ha realizado.

El segundo capítulo, *Análisis comparativo de aplicaciones actuales*, muestra una comparación entre Flatlife y otras aplicaciones actuales que pueden ser descargadas desde la tienda de Android o iOS. Aquí se hace una pequeña descripción del funcionamiento de cada aplicación y, finalmente, mediante una tabla comparativa de características, se muestra el valor añadido de Flatlife.

El tercer capítulo, *Especificación de requisitos software*, es una descripción formal de los requisitos de Flatlife. Incluye una lista extensa de casos de uso que describen la funcionalidad que tiene la aplicación.

El capítulo cuarto, *Análisis*, presenta cuáles eran las alternativas que se han tenido en cuenta antes de comenzar a desarrollar la aplicación, es decir, las tecnologías que podían utilizarse para conseguir los objetivos. Además, se explica cuál ha sido la elección y el motivo.

En el quinto capítulo, *Diseño de la solución*, se describe mediante texto e imágenes cuál es la estructura de los datos en la base de datos seleccionada para Flatlife, la arquitectura, la tecnología utilizada y un resumen que incluye las partes más importantes sobre el diseño de la interfaz gráfica.

El sexto capítulo, *Desarrollo de la solución*, presenta de manera resumida todas las partes que forman la implementación de la aplicación como la estructura de directorios, las traducciones, la navegación entre pantallas o los servicios, entre otras.

El capítulo séptimo, *Resultados*, es en el que se puede ver el resultado final de la aplicación mediante figuras y una descripción de lo que se puede hacer a través de una historia. Además, es en este capítulo dónde aparecen las métricas de calidad que se han utilizado.

El capítulo ocho, *Conclusiones y trabajos futuros*, expone las dificultades de la realización del trabajo, junto con las habilidades aprendidas y una explicación de cómo las asignaturas cursadas en el grado han sido útiles para desarrollar el trabajo. Además, comenta algunos flecos que pueden implementarse a continuación para hacer de Flatlife una aplicación más robusta y atractiva.

Finalmente, el capítulo noveno, *Glosario*, es un diccionario donde se describen algunos vocablos técnicos cuya definición puede no ser conocida.

CAPÍTULO 2
Análisis comparativo de aplicaciones actuales

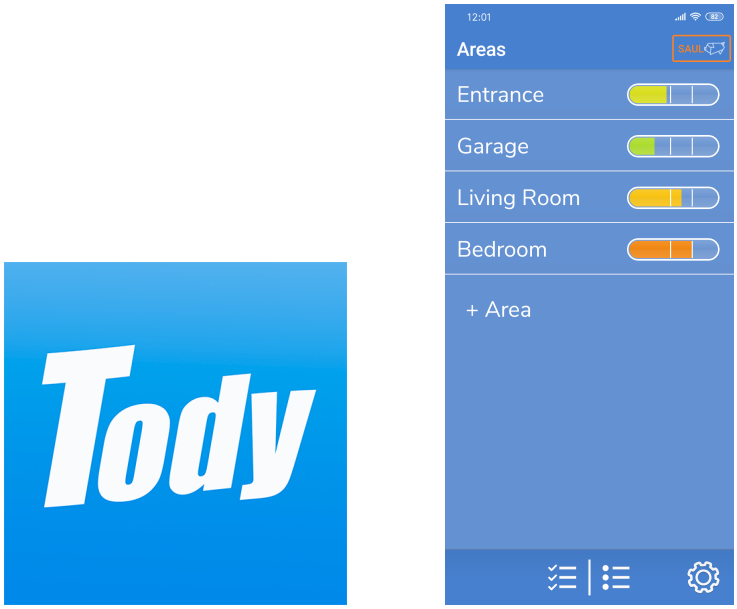
En la actualidad, si bien existen aplicaciones que ayudan a gestionar algunas de las tareas que se llevan a cabo cuando se comparte vivienda, pocas intentan abarcar la totalidad de las tareas de la convivencia. En este apartado vamos a nombrar algunas de las aplicaciones más conocidas y mejor valoradas. Primero de todo, para orientarnos sobre la acogida que tienen en el mercado, en la tabla 2.1 se listan las plataformas soportadas, el número de descargas, el número de valoraciones obtenidas y su valor medio, para las diferentes aplicaciones consideradas.

Tabla 2.1: Estadísticas de aplicaciones actualmente en el mercado (08/10/2019).

Aplicaciones	**Descargas**		**Valoraciones**		**Valoración media sobre 5**	
	Android	IOS	Android	IOS	Android	IOS
Tody	100K+	-	1437	9	4.7	4.6
Splitwise	5M+	-	94126	471	4.6	4.7
Our Groceries	1M+	n/d	44276	n/d	4.7	n/d
Flatastic	100K+	-	4259	2	4.3	3.0
RoomMate	10K+	-	353	1	4.2	5.0

A continuación se explican las características que soportan cada una de las aplicaciones.

2.1 Tody



(a) Logo

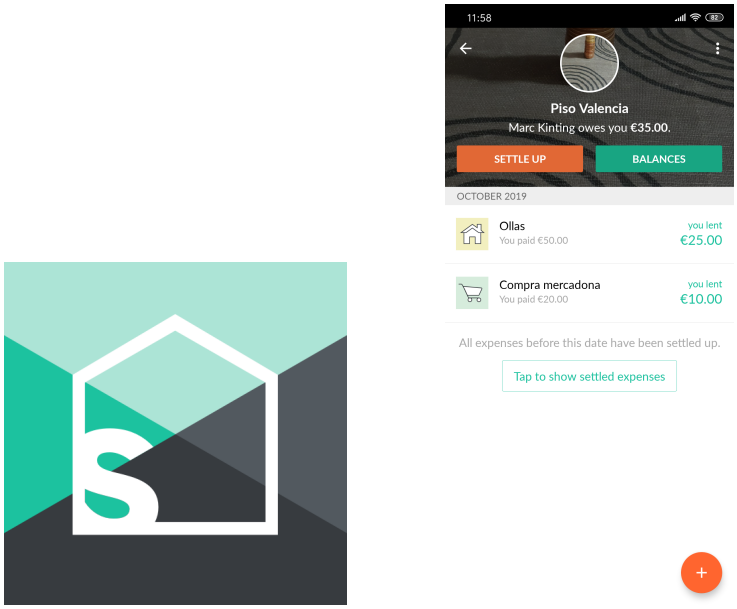
(b) Captura de pantalla

Figura 2.1: Tody

Está aplicación[1] ha sido diseñada para gestionar únicamente la limpieza en un piso. La primera vez que abres la aplicación, te va enseñando paso a paso todo lo que se puede hacer en la versión gratuita. Principalmente podemos crear áreas y dentro de esas áreas añadir tareas. Estas áreas son zonas de la casa que se van a querer limpiar, como por ejemplo la cocina, el baño o el comedor, mientras que las tareas son aquellas actividades de limpieza que se realizarán sobre la zona que se quiera, ya sea quitar el polvo, barrer el suelo, limpiar las paredes, etc. Cada tarea tiene asignados un nombre, una frecuencia, los meses durante los cuales está activa (por defecto, todo el año) y el estado que indica lo limpia que está el área. Cuando se crea la tarea, se crea una barra de estado que indica lo limpia que está el área con respecto a la tarea. Esta barra se irá rellenando (suciedad) en función de los días que pasan y la frecuencia que se le haya asignado para que cuando toque limpiar, la barra de estado esté llena y nos indique que se tiene que realizar dicha limpieza. Las tareas se pueden marcar como realizadas, se pueden adelantar y se pueden atrasar cuando se quiera. Además, la aplicación cuenta con una vista resumen en la que se pueden ver todas las zonas de la casa y su respectiva barra de estado (figura 2.1b), que es la media de las barras de estado de las tareas que se encuentran en ella. Finalmente es importante comentar que la versión gratuita trabaja localmente y los datos no se comparten entre diferentes dispositivos. No obstante, la versión de pago incluye la sincronización de datos entre diferentes dispositivos, además de alguna que otra funcionalidad más.

[1] http://www.todyapp.com/

2.2 Splitwise



(a) Logo

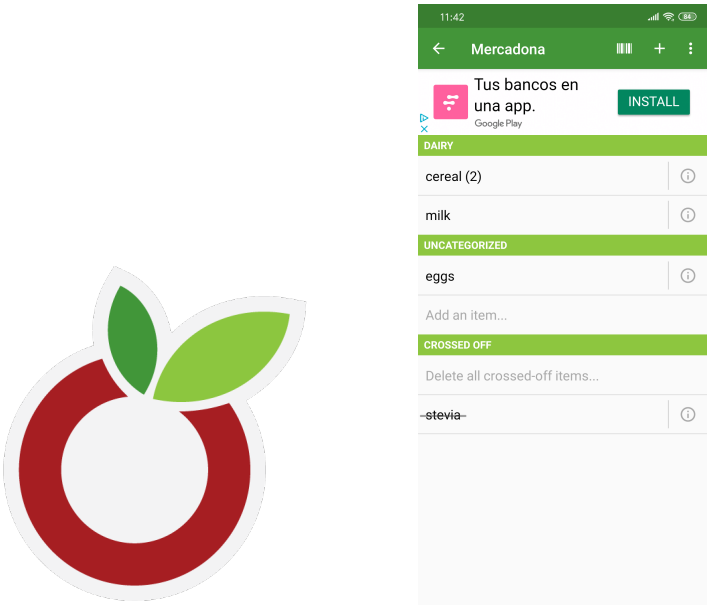
(b) Captura de pantalla

Figura 2.2: Splitwise

Otra gestión importante que se puede llevar a cabo entre compañeros de vivienda es la gestión de los pagos compartidos. Para esta tarea, Splitwise[2] es una aplicación muy completa y sencilla. Primero de todo, permite crear grupos de usuarios con los que compartir los gastos (figura 2.2b). Dentro de un grupo, se pueden registrar pagos, ya sean compartidos con compañeros de piso, con personas externas o no compartidos, pudiendo seleccionar uno a uno los usuarios con los que se compartirá dicho desembolso. Los pagos permiten adjuntar una imagen por si fuera necesario incluir un comprobante que indique qué se ha comprado y el precio o cualquier otra información, además de permitir añadir notas y comentarios. Asimismo, como cabía esperar, se pueden saldar las cuentas pendientes ya sea parcial o completamente. Más características destacadas son la pestaña de amigos, en la que se encuentran los usuarios con los que se comparte vivienda y los que se añaden como amigos a través de la aplicación, indicando si existe alguna deuda pendiente, y la pestaña de actividad, en la cual se pueden consultar las actividades en las que participas directa o indirectamente. Para terminar, se explicará la manera en la que esta aplicación se lucra, ya que su descarga y uso es gratuito y temporalmente ilimitado. La fuente de ingresos de la aplicación es su versión PRO, una versión de pago por suscripción que incluye características tales como el escaneo de recibos, la ausencia de anuncios, tablas y gráficos y conversión de monedas, entre otras.

[2] https://www.splitwise.com/

2.3 Our Groceries



(a) Logo

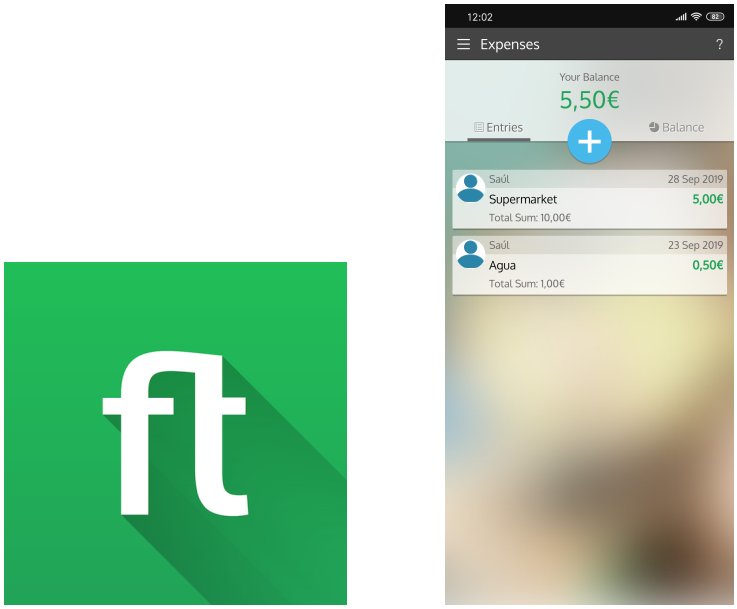
(b) Captura de pantalla

Figura 2.3: Our groceries

Como se puede deducir de su nombre, esta aplicación[3] se encarga de gestionar listas de la compra. Para realizar esta actividad, se podrán crear listas de la compra bajo el nombre que se quiera y añadir elementos a estas lista, además de marcar estos elementos como comprados, como no comprados e incluso eliminarlos (figura 2.3b). Tras añadir un elemento a una lista, se le puede asignar un código de barras, una categoría y la lista a la que pertenece, es decir, se puede cambiar un producto de una lista a otra fácilmente. Otra característica curiosa es la creación de recetas, que son listas de alimentos necesarios para preparar dicha receta, y estos elementos se pueden añadir a una de las listas de la compra que tengamos en ese momento. Compartir las listas entre diferentes dispositivos requiere la creación de una cuenta y que desde todos los dispositivos se acceda a la misma, pero se pueden compartir utilizando aplicaciones de mensajería instantánea cuyo soporte viene integrado en la propia aplicación. Para concluir, se hablará de nuevo sobre el plan de negocio. Como las aplicaciones anteriores, es gratuita, temporalmente ilimitada y con anuncios molestos, pero no tenemos a nuestro alcance todas las características que ofrece la aplicación, como añadir imágenes a los elementos, añadir elementos a las listas usando el código de barras, eliminar la publicidad, etc. que se pueden conseguir a través de un pago único.

[3] https://www.ourgroceries.com/overview

2.4 Flatastic



(a) Logo

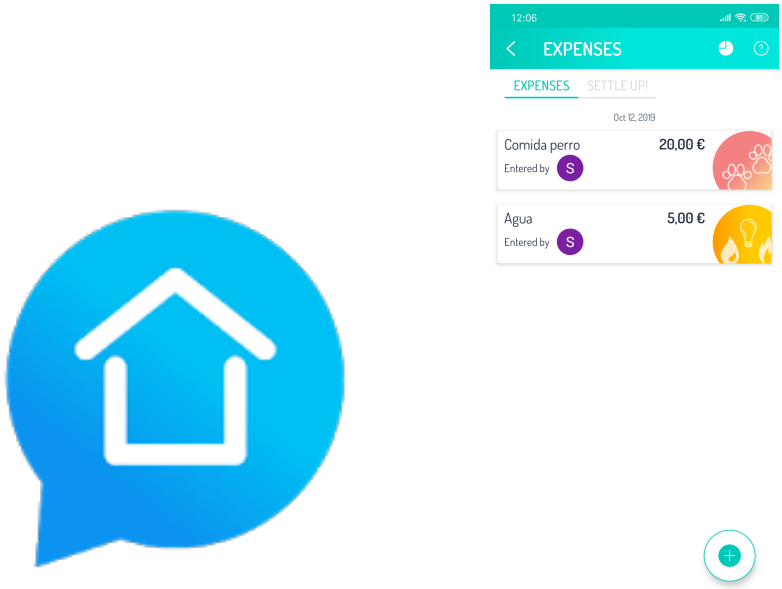
(b) Captura de pantalla

Figura 2.4: Flatastic

Esta aplicación[4], al contrario que las anteriores, es más completa y abarca más gestiones dentro de una vivienda compartida. Tiene un menú lateral que permite dirigirse a las diferentes pantallas. Una de estas pantallas es la encargada de mostrar los detalles de los pagos que se realizan dentro del grupo (figura 2.4b). Esta función se lleva a cabo mediante la adición y la eliminación de pagos que son, básicamente, listas de elementos que tienen asignados un precio y los usuarios que abonarán el pago. Así pues, cuando se quiere saldar una deuda hay que eliminar el pago completo y no tiene la capacidad de realizar un pago que no sea el total de la deuda. Otra gestión de la cual dispone es la limpieza. Se puede registrar una limpieza asignándole un título que debería incluir la zona a limpiar, la frecuencia de limpieza que va desde una limpieza esporádica hasta una vez al año, el esfuerzo de esa limpieza y los usuarios que se verán involucrados. El esfuerzo se mide en puntos y se utilizan para comprobar quién ha trabajado más siempre que se quiera. Aunque a un usuario no le toque limpiar una zona, puede hacerlo y marcarlo para conseguir más puntos. También se puede marcar que la persona a la que le toca realizar la limpieza lo ha hecho, sin que ella tenga que hacerlo. Por otro lado aparece la lista de la compra, única y algo limitada. Simplemente se pueden añadir elementos con el nombre que se quiera, marcarlos como comprados, como no comprados y eliminarlos. La edición de los mismos es una característica premium. Además de estas gestiones, Flatastic incluye un apartado en el que indicar los diferentes pagos repetitivos o suscripciones como pueden ser luz, agua, Internet, Netflix, etc. Finalmente, incluye el envío de mensajes comunes que todos los participantes del grupo podrán leer, además de una lista con todas las acciones que se realizan dentro del grupo en modo de resumen, ya sea una limpieza, una compra, etc. Para terminar de explicar la aplicación y como se ha comentado, la aplicación cuenta con una versión de pago que aumenta las funcionalidades disponibles.

[4] https://flatastic-app.com/

2.5 RoomMate



(a) Logo

(b) Captura de pantalla

Figura 2.5: RoomMate

Finalizamos esta lista con RoomMate[5], una aplicación polivalente como la anterior. De manera similar a Flatastic, RoomMate separa los pagos repetitivos o suscripciones y los pagos corrientes (figura 2.5b). Al contrario que Splitwise, un pago por suscripción en RoomMate genera un aviso cada vez que se espera pagar. Dentro de los pagos corrientes podemos registrar un pago sobre una categoría de las predefinidas o personalizada. Cada uno de los abonos contiene una lista de elementos que se han comprado junto con su precio. En este caso también se puede elegir con quién compartir el reembolso y se pueden añadir notas. Tampoco es posible editar un pago, así que tendremos que eliminarlo y volverlo a crear en caso de que queramos modificar algún dato. Como en las demás aplicaciones que gestionan los pagos, tiene una lista con las deudas pendientes dentro del grupo para que se puedan ver rápidamente. Los pagos repetitivos funcionan como los corrientes excepto que tienes que registrar más información como el proveedor del servicio o recurso, la persona que se va a hacer cargo del pago, la cantidad, el período durante el cual se va a tener el servicio contratado, la fecha en la que se espera pagar y la frecuencia. La frecuencia está limitada a mensual, bimensual, cuatrimestral o bienal. También se pueden añadir notas a estos pagos. La adición de imágenes se reserva a la versión de pago. En cuanto a la gestión de la limpieza, RoomMate permite añadir una zona que se quiera limpiar de las predeterminadas o personalizada, añadiendo la frecuencia, limitada a semanal, cada dos semanas, cada cuatro semanas o libre, y las personas que entran en la cola circular para que cada cierto tiempo sean avisados de que les toca limpiar dicha zona. Esta pantalla trae un calendario mensual, por lo que podremos ver los días que se ha limpiado en un golpe de vista. Otra característica que incorpora la aplicación es la gestión de la lista de la compra, a la cual podemos ir añadiendo elementos, marcarlos como comprados, como no comprados o eliminarlos. Una vez se tenga la lista completa, se puede crear un pago con todos los elementos indicando la categoría y el precio de cada uno.

[5] https://www.the-roommate.com/en/

Finalmente encontramos la gestión de tareas. Una tarea podrá ser añadida indicando un título, las personas asignadas a ella, la fecha en la que se debe realizar, notas y, en caso de haber, el precio de realizarla. Cada tarea creada puede editarse, eliminarse, marcar como completada y como no completada. En cuanto a la monetización, esta aplicación utiliza la misma estrategia que las anteriores, te sirve una aplicación gratuita y temporalmente ilimitada pero con ciertas características bloqueadas que se podrán adquirir pagando, esta vez una suscripción mensual.

2.6 Propuesta

Después de conocer el funcionamiento de varias aplicaciones en el mercado cuya funcionalidad se quiere ampliar o mejorar, se debe explicar el valor añadido que Flatlife aporta y así entender la importancia de este proyecto. Para ello, se ha creado la tabla 2.2 con las características que se han considerado más importantes. Las características escogidas son las siguientes:

- **Gestión de pagos**. La gestión de pagos se ha considerado indispensable en cualquier vivienda compartida ya que en la mayoría de casos va a haber compras que se realicen en común y se hace necesario tener al alcance de la mano un sistema capaz de calcular los balances económicos y saber en todo momento las deudas pendientes entre los diferentes compañeros. Esta funcionalidad tiene como objetivo evitar conflictos con las demás personas, además de evitar la preocupación de estar pendiente de que todo el mundo te pague, creando un clima de compañerismo estupendo.
- **Gestión de la limpieza**. Otra funcionalidad que se ha considerado indispensable para mantener una buena higiene en la vivienda y saber en todo momento quién ha realizado sus labores y quién no.
- **Gestión de eventos**. Crear un buen clima entre las personas que viven en una misma vivienda es primordial, y por eso se ha pensado que un apartado encargado de gestionar todas aquellas actividades que puedan realizarse estando todos juntos, ya sea ir a comer, a patinar o ver una película, es un aspecto importante a tener en cuenta.
- **Listas de la compra**. No es raro que un día vayamos a cocinar y no tengamos aceite, ya que se trata de un producto de uso compartido entre todos los habitantes de la vivienda. Para evitar esta situación, las listas de la compra proveen una o varias listas de productos que se necesitan en la vivienda y que todos los compañeros tengan acceso a ella o ellas.
- **Recursos compartidos**. Cuando existen recursos compartidos en la vivienda, es posible que varias personas necesiten utilizar uno de ellos a la vez. En estos casos, es de utilidad que los compañeros puedan reservar estos recursos por un tiempo determinado y que los demás sepan cuándo no van a poder utilizarlos.
- **Calendario**. Algunas de las características mencionadas trabajan con fechas, por ejemplo, la gestión de la limpieza. El calendario pretende ser una ayuda visual para poder ver de un golpe de vista los días en que se tiene previsto hacer algo o los días en que se han hecho cosas. En el caso de la limpieza, aparecerán los días en que se tiene previsto limpiar, cuando se ha limpiado y cuándo, por el contrario, no se ha limpiado cuando debería.

- **Multiplataforma**. Para llegar a una cantidad de usuarios mayor, se necesita que la aplicación esté disponible para diferentes plataformas o sistemas operativos, como Android e IOS.
- **Online**. Esta característica hace referencia a la capacidad de almacenar datos en la red y que puedan ser accedidos desde distintos dispositivos para que todos los integrantes de la vivienda puedan gestionar su hogar desde sus propios dispositivos móviles.
- **Puede trabajar offline**. Si algún usuario se encuentra sin conexión a Internet, se le permite trabajar con normalidad en la aplicación, es decir, tener acceso a los datos que mantenía la última vez que el usuario se conectó y que todas las acciones que realice mientras no tiene conexión se compartan con los demás compañeros cuando la conexión retorne.
- **Soporte multilenguaje**. Con la misma finalidad que la implementación multiplataforma de la aplicación, la incorporación de varios idiomas es importante para que el nicho de mercado sea más grande.
- **Publicidad**. La publicidad en las aplicaciones es molesta para todo el mundo, pero las aplicaciones necesitan amortizar el esfuerzo y recursos empleados en la creación de la misma. No obstante, existen otros métodos para lucrarse que no involucran esta práctica.
- **Versión de pago**. Además de la publicidad, las aplicaciones pueden conseguir beneficio económico extendiendo su funcionalidad a cambio de dinero.
- **Suscripción**. Una aplicación que tenga una versión de pago por suscripción significa que los usuarios podrán tener acceso a dicha versión mediante el pagamento de una cuota regularmente.

Tabla 2.2: Comparación de características entre aplicaciones del mercado y Flatlife

Característica \Aplicación	Flatlife	Tody	Splitwise	Our groceries	Flatastic	Roommate
Gestión de pagos	✓		✓		✓	✓
Gestión de la limpieza	✓	✓			✓	✓
Gestión de eventos	✓					
Listas de la compra	✓			✓	✓	✓
Recursos compartidos	✓					
Calendario	✓					✓
Multiplataforma	✓	✓	✓		✓	✓
Online	✓		✓	✓	✓	✓
Puede trabajar offline	✓	✓	✓	✓		✓
Soporte multilenguaje	✓		✓		✓	
Publicidad				✓		✓
Versión de pago		✓	✓	✓	✓	✓
Suscripción		✓	✓		✓	✓

Como se puede ver en la tabla 2.2, no muchas de las aplicaciones que se pueden utilizar para gestionar las actividades de una vivienda compartida abarcan la totalidad de estas, por lo que las personas que conviven juntas necesitarían varias aplicaciones para tener un control total de la vivienda. Además, las aplicaciones que abarcan más funcionalidad, a día de hoy son menos utilizadas y la valoración de los usuarios hacia las mismas es menor.

Lo que se desea realizar en este proyecto es una aplicación móvil multiplataforma, que pueda ser utilizada tanto en Android como en IOS, fácil de utilizar, atractiva y que incluya todas las funciones descritas anteriormente, haciendo uso de las ideas que aportan las aplicaciones analizadas.

CAPÍTULO 3
Especificación de requisitos software

En este apartado se procede a anunciar los requisitos software del producto a desarrollar y, para ello, se va a tomar como referencia el estándar de ingeniería de requisitos ISO/IEC/IEE 29148:2018 [3], sin hacer redundante la información mostrada en el capítulo 2.

3.1 Propósito

La especificación de requisitos software (ERS) es un documento que describe de manera simple y concisa cómo va a ser el futuro producto resultante del proyecto. En este caso, como no existe un cliente específico que compra el producto, la función verificadora de las características que aborda la especificación ante las peticiones de los clientes para incluir nueva funcionalidad no existe, por lo que la ERS será de utilidad para los analistas y diseñadores que tendrán una idea de lo que se quiere conseguir y para los desarrolladores, sirviendo de guía en el proceso de implementación porque contiene todo aquello el que sistema debe cumplir.

3.2 Ámbito del sistema

Como ya se ha mencionado anteriormente, Flatlife será una aplicación móvil que pretende ayudar a crear un buen clima de compañerismo en una vivienda compartida a través de una buena organización de las actividades que se pueden llevar a cabo en la misma, mencionadas en el capítulo 2.

Hacer que los usuarios de la aplicación tengan una estancia agradable con sus compañeros y el control sobre todas las actividades que se realizan en común es el objetivo del producto. Dicho objetivo lleva a la meta, que es, tratar de impactar en la sociedad mejorando el estado de ánimo y por ende la calidad de vida de los usuarios.

3.3 Características de los usuarios

Flatlife tiene como usuarios objetivo la población joven entre los 18 y 35 años, estudiantes y trabajadores que quieren o necesitan compartir vivienda. Al haber potencialmente tanta diversidad de usuarios, su nivel educativo, experiencia o conocimiento no

se puede determinar, pero cabe esperar que tengan un manejo ágil de sus dispositivos móviles. A pesar de esto, el sector mencionado no es el único que podría beneficiarse de su funcionalidad. Cualquier familia que disponga de teléfonos móviles con acceso a Internet podrá utilizar la aplicación para mejorar su organización en casa.

3.4 Restricciones

Las restricciones que debe cumplir la aplicación son las siguientes:

- Uso de Firebase[1] como base de datos (Cloud Firestore) y gestor de recursos (Cloud Storage).
- La comunicación entre Flatlife y Firebase debe ser segura.
- Soporte de registro y autenticación con Google[2].
- El diseño debe seguir la guía definida por Material Design[3] para que la interfaz sea fácil de usar por cualquier persona.

3.5 Asunciones y dependencias

Para el correcto funcionamiento de la aplicación se asume que:

- Los dispositivos Android utilizan una versión del sistema operativo superior a la 4.1 (Jelly Bean).
- Los dispositivos IOS utilizan una versión del sistema operativo superior a la 10.
- Firebase está disponible de manera ininterrumpida.

3.6 Requisitos funcionales

Para la especificación de requisitos funcionales se va a hacer uso del lenguaje de modelado UML (*Unified Modeling Language*) [4] ya que se trata de un lenguaje muy conocido, utilizado, sencillo y útil. Específicamente se van a utilizar los diagramas de casos de uso (CU), destinados a mostrar la interacción entre el sistema a desarrollar y los usuarios que lo utilizarán u otros sistemas externos.

Primero de todo, se van a exponer los actores que interactúan con el sistema para después referenciarlos en los casos de uso cuando aparezcan.

3.6.1. Actores

Los actores del sistema son aquellas entidades no pertenecientes a este pero que interactúan con él. Los actores especifican los roles que toman las entidades a la hora de la interacción. A continuación se describen todos los actores.

[1] https://firebase.google.com/
[2] https://developers.google.com/identity
[3] https://material.io/design/

Tabla 3.1: Actor - Usuario No Autenticado

Actor	ACT.1 Usuario No Autenticado
Descripción	Usuario cuya identidad no ha sido todavía resuelta.

Tabla 3.2: Actor - Usuario Autenticado

Actor	ACT.2 Usuario Autenticado
Descripción	Usuario registrado en el sistema y cuya identidad ha sido resuelta después de introducir sus datos.

3.6.2. Casos de uso

A continuación se muestran los casos de uso que se han especificado para Flatlife divididos en secciones según la funcionalidad que abordan.

3.6.2.1. Autenticación y cuenta

Estos casos de uso son los que están relacionados con los datos de un usuario (figuras 3.1, 3.2) y no con los datos de un grupo. Incluyen el registro (tabla 3.3) y autenticación (tabla 3.5), además de la modificación de los datos de la cuenta de un usuario (tablas 3.6, 3.7, 3.8, 3.9).

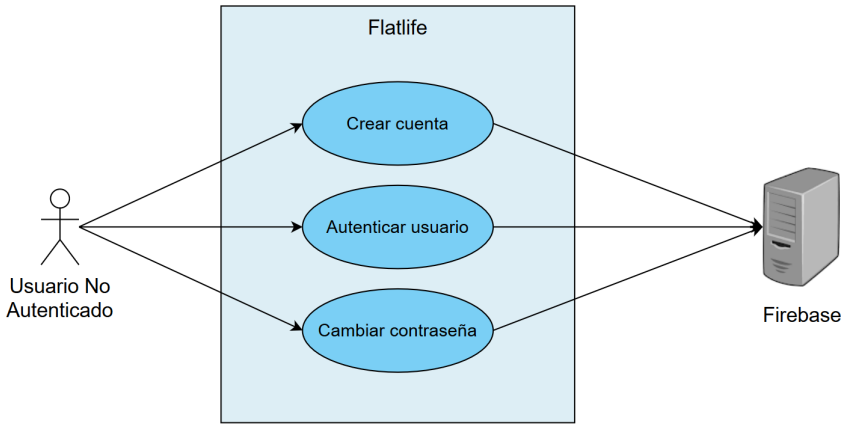


Figura 3.1: Diagrama de casos de uso - autenticación

Tabla 3.3: Caso de uso - crear cuenta

Caso de uso	CU.1 Crear cuenta
Actor	ACT. 1
Descripción	Un Usuario No Autenticado crea una cuenta de Flatlife
Flujo básico	1. **Introducir datos de la cuenta**. El caso de uso se inicia cuando un usuario solicita al sistema la creación de la cuenta. El sistema solicita al usuario un correo electrónico y contraseña (dos veces). El usuario rellena los campos con la información requerida e indica al sistema crear la cuenta. 2. **Validar datos de la cuenta**. El sistema verifica que todos lo campos tienen información y que son correctos: no existe una cuenta asociada a dicho correo, las contraseñas coinciden y tienen más de 6 caracteres y mínimo un número y el correo electrónico sigue el formato de una dirección de correo electrónico. 3. **Verificar correo**. El sistema envía un correo electrónico a la cuenta introducida por el usuario con un enlace para verificar el correo e indica al usuario que debe verificar la cuenta para poder autenticarse. 4. **Crear cuenta**. El sistema crea la cuenta sin verificar con los datos introducidos por el usuario. El caso de uso finaliza.
Flujo alternativo	1. **Error de verificación de datos**. En el paso 2 del flujo básico el sistema determina que algún dato no cumple los requisitos necesarios. El sistema muestra un mensaje al usuario por cada dato incorrecto indicando por qué lo es. El caso de uso vuelve al paso 1 del flujo básico con los datos introducidos por el usuario.
Pre-condiciones	-
Post-condiciones	El sistema contiene la nueva cuenta recién creada

Tabla 3.4: Caso de uso - cambiar contraseña

Caso de uso	CU.2 Cambiar contraseña
Actor	ACT. 1
Descripción	Un Usuario No Autenticado cambia la contraseña con la que se autentica en el sistema
Flujo básico	1. **Solicitar cambio de contraseña**. El caso de uso se inicia cuando un usuario solicita al sistema un cambio de contraseña. El sistema solicita al usuario que introduzca su dirección de correo electrónico. El usuario provee al sistema con el dato requerido. 2. **Validar correo**. El sistema verifica que el correo introducido está bien formateado y que está asociado a una cuenta del sistema. Posteriormente envía un mensaje a esa dirección con un enlace para que el usuario acceda e introduzca su nueva contraseña dos veces. 3. **Validar contraseña**. El usuario accede al enlace y escribe su nueva contraseña. El sistema verifica que las dos contraseñas coinciden y que cumplen los requisitos de seguridad: mínimo 6 caracteres y 1 número. 4. **Actualizar contraseña**. El sistema modifica la contraseña de la cuenta asignándole aquella introducida por el usuario. El caso de uso finaliza.
Flujo alternativo	1. **Error verificación de correo**. En el paso 2 del flujo básico el sistema determina que la dirección de correo introducida por el usuario no sigue el formato adecuado o que no existe ninguna cuenta asociada a dicha dirección. El sistema informa al usuario de la situación. El caso de uso vuelve al paso 1 del flujo básico con los datos introducidos por el usuario. 2. **Error de verificación de contraseña**. En el paso 3 del flujo básico el sistema determina que las contraseñas no coinciden o que no cumplen los requisitos de seguridad. El sistema muestra un mensaje al usuario indicando la situación. El caso de uso vuelva al paso 3 del flujo básico con los datos introducidos por el usuario.
Pre-condiciones	-
Post-condiciones	La contraseña que autentica al usuario que inicia el caso de uso se actualiza

Tabla 3.5: Caso de uso - autenticar usuario

Caso de uso	CU.3 Autenticar usuario
Actor	ACT. 1
Descripción	Un Usuario No Autenticado se autentica con sus credenciales en el sistema
Flujo básico	1. **Introducir datos de autenticación**. El caso de uso se inicia cuando un usuario solicita al sistema la autenticación. El sistema solicita al usuario su correo electrónico y contraseña. El usuario rellena los campos con la información requerida y solicita al sistema que lo autentique. 2. **Autenticación**. El sistema verifica que los datos introducidos por el usuario corresponden a una cuenta y autentica al usuario. El caso de uso finaliza.
Flujo alternativo	1. **Error de autenticación**. En el paso 2 del flujo básico el sistema determina que los datos introducidos por el usuario no corresponden con ninguna cuenta registrada en el sistema. El sistema indica al usuario que alguno de los datos que ha introducido es incorrecto. El caso de uso retorna al paso 1 del flujo básico con los datos introducidos por el usuario.
Pre-condiciones	-
Post-condiciones	El usuario que realiza el caso de uso pasa a ser un Usuario Autenticado

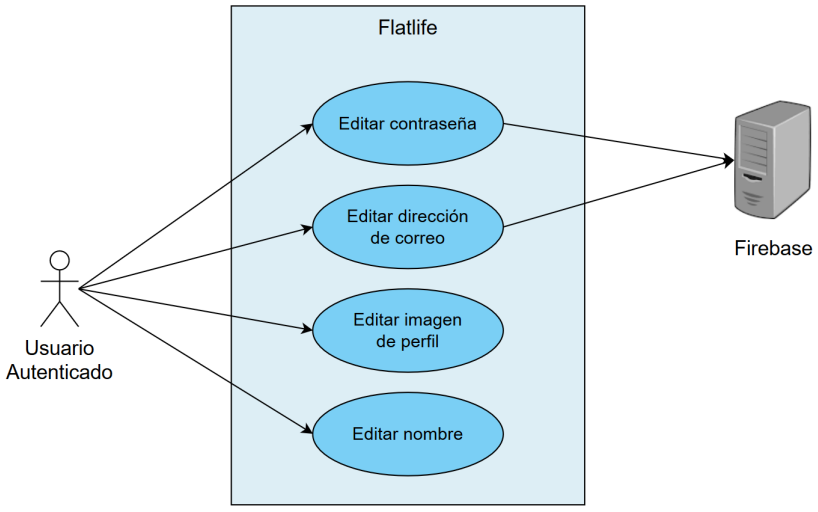


Figura 3.2: Diagrama de casos de uso - cuenta

Tabla 3.6: Caso de uso - editar contraseña

Caso de uso	CU.4 Editar contraseña
Actor	ACT. 2
Descripción	Un Usuario Autenticado cambia la contraseña con la que se autentica en el sistema
Flujo básico	1. **Solicitar cambio de contraseña**. El caso de uso se inicia cuando un usuario solicita al sistema un cambio de contraseña. El sistema le solicita al usuario que introduzca dos veces su contraseña. El usuario introduce dos veces su nueva contraseña. 2. **Validar contraseña**. El sistema verifica que las dos contraseñas coinciden y que cumplen los requisitos de seguridad: mínimo 6 caracteres y 1 número. 3. **Actualizar contraseña**. El sistema modifica la contraseña de la cuenta asignándole aquella introducida por el usuario. El caso de uso finaliza.
Flujo alternativo	1. **Error de verificación de contraseña**. En el paso 2 del flujo básico el sistema determina que las contraseñas no coinciden o que no cumplen los requisitos de seguridad. El sistema muestra un mensaje al usuario indicando la situación. El caso de uso vuelva al paso 1 del flujo básico con los datos introducidos por el usuario.
Pre-condiciones	-
Post-condiciones	La contraseña que autentica al usuario que inicia el caso de uso se actualiza

Tabla 3.7: Caso de uso - editar dirección de correo

Caso de uso	CU.5 Editar dirección de correo
Actor	ACT. 2
Descripción	Un Usuario Autenticado cambia la dirección de correo asociada a su cuenta.
Flujo básico	1. **Solicitar cambio de dirección de correo**. El caso de uso se inicia cuando un usuario solicita al sistema un cambio de dirección de correo. El sistema le solicita al usuario que introduzca la nueva dirección. El usuario introduce el dato solicitado por el sistema. 2. **Validar correo**. El sistema verifica que la dirección pasada por el usuario está bien formateada y envía un mensaje al correo electrónico inicial indicando el cambio de dirección de correo y un enlace para reasignar el correo inicial a la cuenta. 3. **Actualizar contraseña**. El sistema modifica la dirección de correo de la cuenta asignándole aquella introducida por el usuario. El caso de uso finaliza.
Flujo alternativo	1. **Error verificación de correo**. En el paso 2 del flujo básico el sistema determina que la dirección de correo introducida por el usuario no sigue el formato adecuado. El sistema informa al usuario de la situación. El caso de uso vuelve al paso 1 del flujo básico con los datos introducidos por el usuario.
Pre-condiciones	-
Post-condiciones	La dirección de correo asociada a la cuenta del usuario que inicia el caso de uso se actualiza

Tabla 3.8: Caso de uso - editar imagen de perfil

Caso de uso	CU.6 Editar imagen de perfil
Actor	ACT. 2
Descripción	Un Usuario Autenticado cambia la imagen de perfil asociada a su cuenta.
Flujo básico	1. **Solicitar cambio de imagen de perfil**. El caso de uso se inicia cuando un usuario solicita al sistema un cambio de imagen de perfil. El sistema le solicita al usuario la nueva imagen. El usuario selecciona la nueva imagen. 2. **Actualizar imagen de perfil**. El sistema modifica la imagen de perfil de la cuenta asignándole aquella seleccionada por el usuario. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	-
Post-condiciones	La imagen de perfil asociada a la cuenta del usuario que inicia el caso de uso se actualiza

Tabla 3.9: Caso de uso - editar nombre

Caso de uso	CU.7 Editar nombre
Actor	ACT. 2
Descripción	Un Usuario Autenticado cambia el nombre asociado a su cuenta.
Flujo básico	1. **Solicitar cambio de nombre**. El caso de uso se inicia cuando un usuario solicita al sistema un cambio de nombre. El sistema solicita al usuario el nuevo nombre. El usuario introduce el nombre. 2. **Actualizar nombre**. El sistema modifica el nombre de la cuenta asignándole aquel seleccionado por el usuario. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	-
Post-condiciones	El nombre asociado a la cuenta del usuario que inicia el caso de uso se actualiza

3.6.2.2. Grupos

Los casos de uso de esta sección son los relacionados con los grupos de Flatlife, es decir, con sus datos y con los usuarios con los que está relacionado (figura 3.3). Los datos de un grupo no incluyen las entidades relacionadas con él como los pagos o las reservas. Así pues, funcionalidad como la creación de un grupo (tabla 3.10), la gestión de usuarios (tablas 3.11 y 3.12) y la modificación de los datos del mismo (tablas 3.13 y 3.14) está explicada en esta sección.

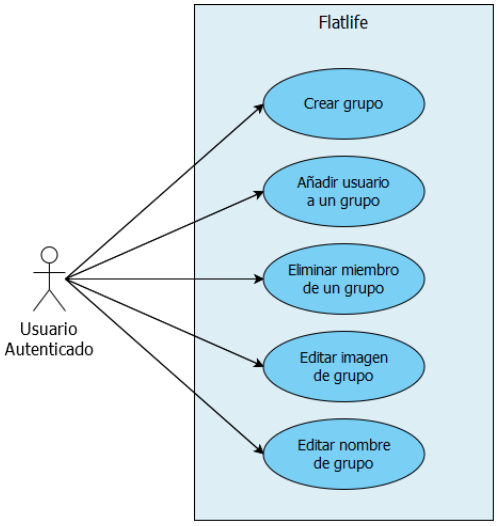


Figura 3.3: Diagrama de casos de uso - grupos

Tabla 3.10: Caso de uso -crear grupo

Caso de uso	CU.8 Crear grupo
Actor	ACT. 2
Descripción	Un Usuario Autenticado crea un grupo
Flujo básico	1. **Introducir información del grupo**. El caso de uso se inicia cuando un usuario solicita al sistema la creación de un nuevo grupo. El sistema solicita al usuario los datos asociados al grupo a crear: nombre e imagen (opcional). El usuario provee los datos e indica al sistema crear el grupo. 2. **Comprobar información**. El sistema verifica que el campo de nombre contiene información. 3. **Crear grupo**. El sistema crea el grupo. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	-
Post-condiciones	Un nuevo grupo se registra en el sistema

Tabla 3.11: Caso de uso - añadir usuario a un grupo

Caso de uso	CU.9 Añadir usuario a un grupo
Actor	ACT. 2
Descripción	Un Usuario Autenticado miembro de un grupo añade a un usuario al grupo al que pertenece
Flujo básico	1. **Enviar solicitud de unión**. El caso de uso se inicia cuando un usuario perteneciente a un grupo solicita al sistema la unión de otro usuario al mismo grupo. El sistema solicita al usuario la dirección de correo electrónico del destinatario. El sistema crea una petición de unión y se la hace llegar al usuario receptor. 2. **Confirmar solicitud**. El receptor de la solicitud acepta la solicitud. El sistema une el nuevo miembro al grupo. El caso de uso finaliza.
Flujo alternativo	1. **Usuario no encontrado**. En el paso 1 del flujo básico, el sistema determina que la dirección de correo introducida por el usuario no corresponde con la de un usuario registrado y muestra un mensaje al usuario para indicarle la situación. El caso de uso retorna al paso 1 del flujo básico con los valores introducidos por el usuario.
Pre-condiciones	-
Post-condiciones	El grupo contiene al nuevo miembro

Tabla 3.12: Caso de uso - eliminar un miembro de un grupo

Caso de uso	CU.10 Eliminar un miembro de un grupo
Actor	ACT. 2
Descripción	Un Usuario Autenticado miembro de un grupo elimina a un miembro del mismo grupo
Flujo básico	1. **Eliminar miembro de un grupo**. El caso de uso se inicia cuando un usuario perteneciente a un grupo solicita al sistema la eliminación de un usuario del grupo (él mismo incluido). El sistema solicita al usuario la confirmación de la acción. El usuario confirma la eliminación. El sistema elimina del grupo al usuario indicado. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	-
Post-condiciones	El grupo deja de contener al usuario indicado

Tabla 3.13: Caso de uso - editar imagen de grupo

Caso de uso	CU.11 Editar imagen de grupo
Actor	ACT. 2
Descripción	Un Usuario Autenticado miembro de un grupo cambia la imagen del grupo al que pertenece
Flujo básico	1. **Cambiar imagen de grupo**. El caso de uso se inicia cuando un usuario perteneciente a un grupo solicita al sistema la modificación de la imagen del grupo. El sistema solicita al usuario que seleccione la imagen. El usuario selecciona la imagen. El sistema actualiza la imagen del grupo, asignando la seleccionada por el usuario. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	-
Post-condiciones	El grupo tiene asociada la foto seleccionada por el usuario

Tabla 3.14: Caso de uso - editar nombre de grupo

Caso de uso	CU.12 Editar nombre de grupo
Actor	ACT. 2
Descripción	Un Usuario Autenticado miembro de un grupo cambia el nombre del grupo al que pertenece
Flujo básico	1. **Cambiar nombre de grupo**. El caso de uso se inicia cuando un usuario perteneciente a un grupo solicita al sistema la modificación del nombre del grupo. El sistema solicita al usuario el nuevo nombre. El usuario proporciona al sistema un nombre. El sistema actualiza el nombre del grupo por el proporcionado por el usuario. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	-
Post-condiciones	El nombre del grupo pasa a ser el indicado por el usuario

3.6.2.3. Gestión de pagos

La gestión de pagos incluye varias funcionalidades de la aplicación: deudas, pagos, recordatorios de pagos y solicitudes sobre pagos o deudas (figuras 3.4 y 3.5). Podría entenderse como aquellas actividades relacionadas con la economía dentro de un grupo.

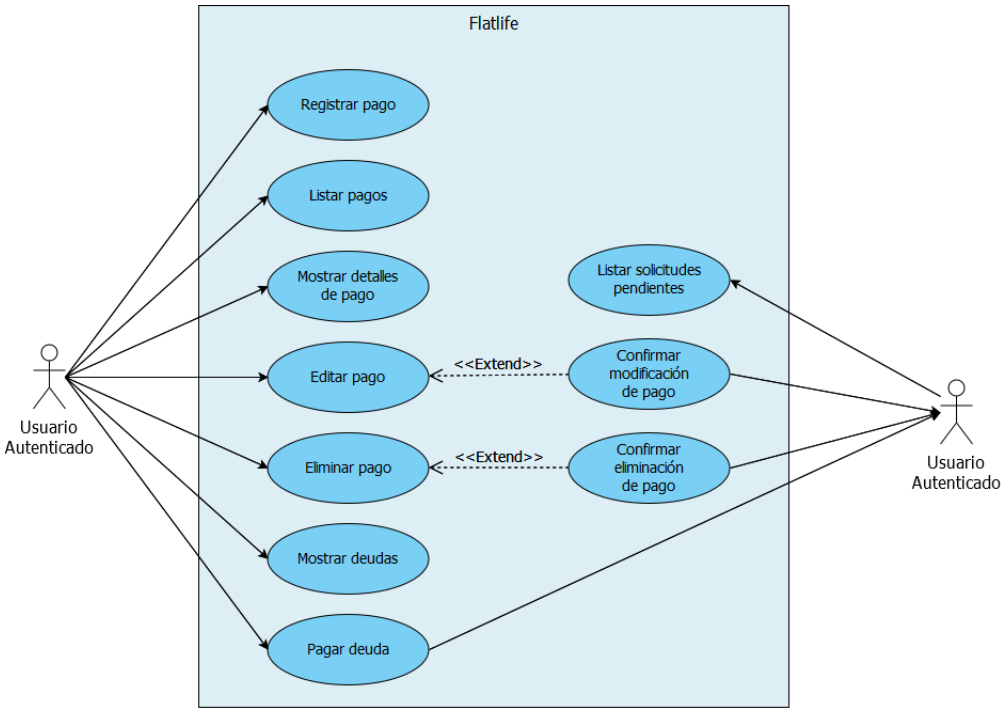


Figura 3.4: Diagrama de casos de uso - pagos

Tabla 3.15: Caso de uso - registrar pago

Caso de uso	CU.13 Registrar pago
Actor	ACT. 2
Descripción	Un Usuario Autenticado registra un pago dentro del grupo
Flujo básico	1. **Introducir información de pago**. El caso de uso se inicia cuando un usuario solicita al sistema el registro de un pago. El sistema solicita al usuario la descripción, el importe y los usuarios involucrados en el pago que se quiere crear, los cuales pertenecen al grupo. El usuario rellena los campos con la información requerida e indica al sistema crear el pago. 2. **Validar datos de pago**. El sistema verifica que todos lo campos tienen información y que son correctos: el formato de los datos es adecuado. 3. **Crear pago**. El sistema crea y registra un pago dentro del grupo. 4. **Actualizar deuda**. El sistema calcula cuáles son las deudas entre todos los usuarios del grupo después de crear el pago y las actualiza. El caso de uso finaliza.
Flujo alternativo	1. **Error en los datos**. En el paso 2 del flujo básico, el sistema determina que existe falta de información o que algún dato es incorrecto. El sistema indica al usuario el problema. El caso de uso retorna al paso 1 del flujo básico con los valores introducidos por el usuario. 2. **Añadir imagen a pago**. En el paso 1 del flujo básico, el usuario selecciona la opción de añadir imagen al pago. El sistema le da la opción de escoger una imagen de la galería. El usuario selecciona la imagen. El sistema introduce la imagen en el pago. El caso de uso retorna al paso en que fue llamado.
Pre-condiciones	-
Post-condiciones	El grupo contiene el pago recién creado

Tabla 3.16: Caso de uso - listar pagos

Caso de uso	CU.14 Listar pagos
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve los pagos que se han registrado dentro del grupo
Flujo básico	1. **Mostrar lista de pagos**. El caso de uso se inicia cuando un usuario solicita al sistema información sobre los pagos registrados dentro del grupo. El sistema le muestra al usuario una lista con información resumida de todos los pagos registrados: descripción, creador y el dinero que debe o le deben. El caso de uso finaliza.
Flujo alternativo	1. **Excepción: No se encuentran pagos registrados**. En el paso 1 del flujo básico el resultado de la búsqueda de pagos está vacío. El sistema muestra al usuario un mensaje indicando que no existen pagos registrados en el grupo.
Pre-condiciones	-
Post-condiciones	-

Tabla 3.17: Caso de uso - mostrar detalles de un pago

Caso de uso	CU.15 Mostrar detalles de un pago
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve toda la información asociada a un pago determinado que haya elegido
Flujo básico	1. **Mostrar detalles de un pago**. El caso de uso comienza cuando un usuario solicita al sistema información sobre un pago registrado en el grupo. El sistema le muestra toda la información disponible del pago seleccionado: imagen, descripción, creador, fecha de creación, cantidad y usuarios involucrados, indicando para cada uno lo que debe o le deben por el pago realizado. El caso de uno finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo tiene registrado al menos un pago
Post-condiciones	-

Tabla 3.18: Caso de uso - listar solicitudes pendientes

Caso de uso	CU.16 Listar solicitudes pendientes
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve las solicitudes pendientes recibidas de otros usuarios
Flujo básico	1. **Mostrar lista de solicitudes pendientes**. El caso de uso se inicia cuando un usuario solicita al sistema información sobre las solicitudes que tiene pendientes de contestar. El sistema le muestra al usuario una lista con información resumida de todas las solicitudes: tipo de petición (modificar pago, eliminar pago o deuda pagada), el usuario que creó la petición y la fecha de creación. El caso de uso finaliza.
Flujo alternativo	1. **Excepción: No se encuentran peticiones pendientes**. En el paso 1 del flujo básico el resultado de la búsqueda de solicitudes está vacío. El sistema muestra al usuario un mensaje indicando que no tiene solicitudes pendientes.
Pre-condiciones	-
Post-condiciones	-

Tabla 3.19: Caso de uso - mostrar detalles de una solicitud

Caso de uso	CU.17 Mostrar detalles de una solicitud
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve toda la información asociada a una solicitud determinada que haya elegido
Flujo básico	1. **Mostrar detalles de una solicitud**. El caso de uso comienza cuando un usuario solicita al sistema información sobre una solicitud pendiente. El sistema le muestra toda la información disponible de la solicitud seleccionada. ▪ **Solicitud de modificación de pago**. Muestra los detalles del pago actuales y los propuestos por el usuario creador de la solicitud. ▪ **Solicitud de eliminación de pago**. Muestra los detalles del pago. ▪ **Solicitud de deuda pagada**. Muestra el nuevo balance de deudas. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El usuario que realiza el caso de uso tiene al menos una solicitud pendiente
Post-condiciones	-

Tabla 3.20: Caso de uso - editar pago

Caso de uso	CU.18 Editar pago
Actor	ACT. 2
Descripción	Un Usuario Autenticado edita un pago dentro del grupo
Flujo básico	1. **Introducir información de pago**. El caso de uso se inicia cuando un usuario solicita al sistema la modificación de un pago. El sistema solicita la nueva información que debe contener el pago y, simultáneamente, le muestra al usuario la información actual: descripción, cuota, imagen y usuarios involucrados, los cuales pertenecen al grupo. El usuario introduce la nueva información e indica al sistema actualizar el pago. 2. **Validar datos de pago**. El sistema verifica que todos lo campos tienen información y que son correctos: el formato de los datos es adecuado, además de determinar que el usuario que modifica el pago es el creador de éste. 3. **Actualizar pago**. El sistema actualiza los datos del pago y todas las deudas entre los usuarios. El caso de uso finaliza.
Flujo alternativo	1. **Error en los datos**. En el paso 2 del flujo básico, el sistema determina que existe falta de información o que algún dato es incorrecto. El sistema indica al usuario el problema. El caso de uso retorna al paso 1 del flujo básico con la nueva información introducida por el usuario.
Pre-condiciones	El grupo tiene al menos un pago registrado
Post-condiciones	La información del pago seleccionado se actualiza
Puntos de extensión	1. **Confirmar modificación de pago**. En el paso 2 del flujo básico, el sistema determina que el usuario no es el creador del pago y le solicita al mismo permiso para la creación de una petición destinada al creador del pago para que confirme la validez de la nueva información. El usuario confirma la creación de la petición.

Tabla 3.21: Caso de uso - confirmar modificación de un pago

Caso de uso	CU.18.1 Confirmar modificación de un pago «extend»
Actor	ACT. 2
Descripción	Un Usuario Autenticado creador de un pago confirma la petición de modificación de ese pago realizada por otro usuario
Flujo básico	1. **Confirmar petición**. El caso de uso se inicia cuando un usuario confirma una solicitud de modificación de un pago realizada por otro usuario. 2. **Actualizar pago**. El sistema actualiza los datos del pago. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El usuario realizador del caso de uso tiene una petición pendiente de modificación de un pago creado por él
Post-condiciones	El pago incluido en la solicitud contiene la nueva información

Tabla 3.22: Caso de uso - eliminar pago

Caso de uso	CU.19 Eliminar pago
Actor	ACT. 2
Descripción	Un Usuario Autenticado elimina un pago dentro del grupo
Flujo básico	1. **Solicitar eliminación de pago**. El caso de uso se inicia cuando un usuario solicita al sistema la eliminación de un pago registrado. 2. **Verificar eliminación de pago**. El sistema verifica que el usuario que realiza la petición es el creador de dicho pago y solicita la confirmación de la acción solicitada al usuario. 3. **Eliminar pago**. El usuario confirma la eliminación del pago seleccionado. El sistema elimina el pago y actualiza todas las deudas entre los usuarios. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo tiene registrado al menos un pago
Post-condiciones	El grupo deja de tener el pago eliminado
Puntos de extensión	1. **Confirmar eliminación de pago**. En el paso 2 del flujo básico, el sistema determina que el usuario no es el creador del pago y le solicita al mismo permiso para la creación de una petición destinada al creador del pago para que confirme la eliminación. El usuario confirma la creación de la petición.

Tabla 3.23: Caso de uso - confirmar eliminación de un pago

Caso de uso	CU.19.1 Confirmar eliminación de un pago «extend»
Actor	ACT. 2
Descripción	Un Usuario Autenticado creador de un pago confirma la petición de eliminación de ese pago realizada por otro usuario
Flujo básico	1. **Confirmar petición**. El caso de uso se inicia cuando un usuario confirma una solicitud de eliminación de un pago realizada por otro usuario. 2. **Eliminar pago**. El sistema elimina el pago del grupo. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El usuario realizador del caso de uso tiene una petición pendiente de eliminación de un pago creado por él
Post-condiciones	El pago incluido en la solicitud no pertenece al grupo

Tabla 3.24: Caso de uso - mostrar deudas

Caso de uso	CU.20 Mostrar deudas
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve las deudas pendientes con los demás usuarios del grupo
Flujo básico	1. **Mostrar deudas**. El caso de uso se inicia cuando un usuario solicita al sistema información sobre las deudas pendientes con los usuarios del grupo. El sistema le muestra al usuario, por cada usuario dentro del grupo, el dinero que debe o que le deben. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	-
Post-condiciones	-

Tabla 3.25: Caso de uso - pagar deuda

Caso de uso	CU.21 Pagar deuda
Actor	ACT. 2
Descripción	Un Usuario Autenticado paga parcial o totalmente una deuda pendiente con otro usuario
Flujo básico	1. **Introducir cantidad a saldar**. El caso de uso se inicia cuando un usuario indica al sistema su voluntad de reducir o acabar con una deuda pendiente. El sistema le pide al usuario la cantidad que desea retribuir. El usuario indica la cantidad y solicita al sistema saldar la deuda. 2. **Verificar de la cantidad introducida por el usuario**. El sistema comprueba que la cantidad introducida por el usuario no es un valor negativo ni mayor que la deuda. 3. **Crear solicitud de pago**. El sistema crea una solicitud que llegará al usuario destino informando sobre la voluntad del deudor de reducir o eliminar la deuda pendiente. 4. **Confirmar nuevo estado de la deuda**. El usuario receptor de la solicitud confirma la petición de saldar la deuda. El sistema modifica la deuda para reducir la cantidad, llegando a 0 en caso de que desaparezca.
Flujo alternativo	1. **Cantidad de retribución negativa**. En el paso 2 del flujo básico el sistema determina que la cantidad introducida por el usuario es negativa. El sistema cambia automáticamente el valor a uno positivo. El caso de uso retorna al paso en el que fue llamado. 2. **Cantidad de retribución mayor que la deuda**. En el paso 2 del flujo básico el sistema determina que la cantidad introducida por el usuario supera la deuda a saldar. El sistema cambia automáticamente el valor al valor de la deuda. El caso de uso retorna al paso en el que fue llamado. 3. **Rechazo del nuevo estado de la deuda**. En el paso 4 del flujo básico, el receptor rechaza la solicitud de retribución. El sistema elimina la solicitud de confirmación del nuevo estado de la deuda y avisa al usuario pagador el rechazo de su solicitud. El caso de uso finaliza.
Pre-condiciones	Existe una deuda entre dos usuarios
Post-condiciones	Se reduce o elimina una deuda pendiente entre dos usuarios

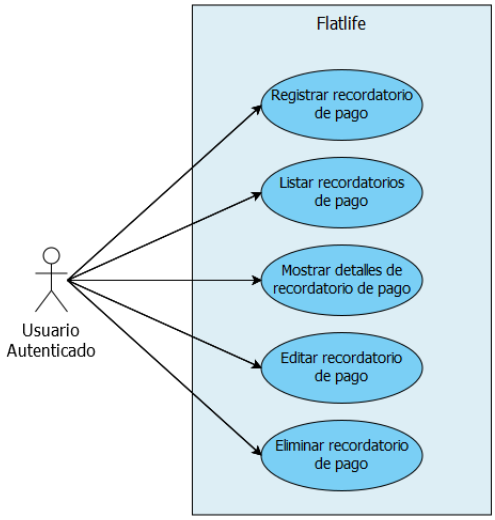


Figura 3.5: Diagrama de casos de uso - recordatorios de pago

Tabla 3.26: Caso de uso - registrar recordatorio de pago

Caso de uso	CU.21 Registrar recordatorio de pago
Actor	ACT. 2
Descripción	Un Usuario Autenticado registra un recordatorio de pago dentro del grupo
Flujo básico	1. **Introducir información de recordatorio**. El caso de uso es iniciado con la petición de un usuario para registrar un recordatorio de pago. El sistema solicita al usuario toda la información necesaria para realizar la acción: descripción, cuota (si ésta varía, será una aproximación), usuarios involucrados pertenecientes al grupo, frecuencia de pago y fecha del próximo pago que será la fecha del primer pago. El usuario proporciona la información al sistema y solicita el registro. 2. **Validar datos de pago**. El sistema verifica que todos los campos tienen información y que son correctos: el formato de los datos es adecuado. 3. **Crear pago**. El sistema crea y registra un recordatorio de pago dentro del grupo. El caso de uso finaliza.
Flujo alternativo	1. **Error en datos**. En el paso 2 del flujo básico, el sistema determina que existe falta de información o que algún dato es incorrecto. El sistema indica al usuario la situación detallando el motivo. El caso de uso retorna al paso 1 del flujo básico con la nueva información introducida por el usuario.
Pre-condiciones	-
Post-condiciones	El grupo contiene el recordatorio de pago recién creado

Tabla 3.27: Caso de uso - listar recordatorios de pago

Caso de uso	CU.22 Listar recordatorios de pago
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve los recordatorios de pago que se han registrado dentro del grupo
Flujo básico	1. **Mostrar lista de recordatorios de pago**. El caso de uso se inicia cuando un usuario solicita al sistema información sobre los recordatorios de pago registrados dentro del grupo. El sistema le muestra al usuario una lista con información resumida de todos los recordatorios de pago registrados: descripción, cuota, días hasta el siguiente pago requerido. El caso de uso finaliza.
Flujo alternativo	1. **Excepción: No se encuentran recordatorios de pago registrados**. En el paso 1 del flujo básico el resultado de la búsqueda de pagos está vacío. El sistema muestra al usuario un mensaje indicando que no existen recordatorios de pago registrados en el grupo.
Pre-condiciones	-
Post-condiciones	-

Tabla 3.28: Caso de uso - mostrar detalles de un recordatorio de pago

Caso de uso	CU.23 Mostrar detalles de un recordatorio de pago
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve toda la información asociada a un recordatorio de pago determinado que haya elegido
Flujo básico	1. **Mostrar detalles de un pago**. El caso de uso comienza cuando un usuario solicita al sistema información sobre un recordatorio de pago registrado en el grupo. El sistema le muestra toda la información disponible del pago seleccionado: descripción, cantidad, usuarios involucrados, frecuencia de pago, día en que se tiene previsto pagar, días hasta el siguiente pago.
Flujo alternativo	-
Pre-condiciones	El grupo tiene registrado al menos un recordatorio de pago
Post-condiciones	-

Tabla 3.29: Caso de uso - editar recordatorio de pago

Caso de uso	CU.24 Editar recordatorio de pago
Actor	ACT. 2
Descripción	Un Usuario Autenticado edita un recordatorio de pago dentro del grupo
Flujo básico	1. **Introducir información de recordatorio**. El caso de uso se inicia cuando un usuario solicita al sistema la modificación de un recordatorio de pago. El sistema solicita la nueva información que debe contener y, simultáneamente, le muestra al usuario la información actual: descripción, cuota, usuarios involucrados pertenecientes al grupo, frecuencia de pago y la fecha del próximo pago. El usuario introduce la nueva información e indica al sistema actualizar el recordatorio. 2. **Validar datos de recordatorio**. El sistema verifica que todos lo campos tienen información y que son correctos: el formato de los datos es adecuado. 3. **Actualizar recordatorio de pago**. El sistema actualiza los datos del recordatorio. El caso de uso finaliza.
Flujo alternativo	1. **Error en datos**. En el paso 2 del flujo básico, el sistema determina que existe falta de información o que algún dato es incorrecto. El sistema indica al usuario el problema. El caso de uso retorna al paso 1 del flujo básico con los nuevos valores introducidos por el usuario.
Pre-condiciones	El grupo tiene un recordatorio de pago registrado
Post-condiciones	La información del recordatorio de pago seleccionado se actualiza

Tabla 3.30: Caso de uso - eliminar recordatorio de pago

Caso de uso	CU.25 Eliminar recordatorio de pago
Actor	ACT. 2
Descripción	Un Usuario Autenticado elimina un recordatorio de pago dentro del grupo
Flujo básico	1. **Solicitar eliminación de recordatorio de pago**. El caso de uso se inicia cuando un usuario solicita al sistema la eliminación de un recordatorio de pago registrado. El sistema solicita la confirmación de la acción solicitada al usuarios. 2. **Eliminar recordatorio de pago**. El usuario confirma la eliminación del pago registrado. El sistema elimina el pago del grupo. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo tiene registrado al menos un recordatorio de pago
Post-condiciones	El grupo deja de tener el recordatorio de pago recién eliminado

3.6.2.4. Gestión de la limpieza

La gestión de la limpieza se realiza mediante zonas de limpieza (figura 3.6) y planes de limpieza (figura 3.7) que se programarán sobre las zonas.

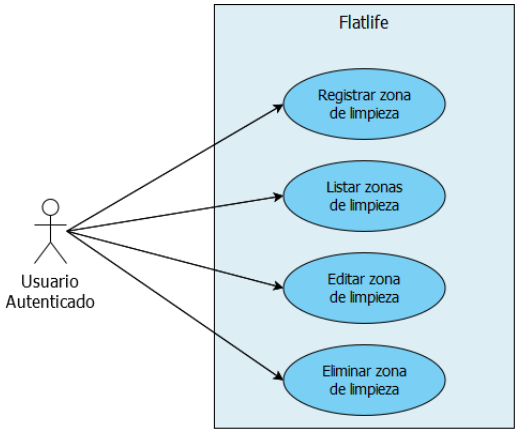


Figura 3.6: Diagrama de casos de uso - zonas de limpieza

Tabla 3.31: Caso de uso - registrar zona de limpieza

Caso de uso	CU.26 Registrar zona de limpieza
Actor	ACT. 2
Descripción	Un Usuario Autenticado registra una zona de limpieza
Flujo básico	1. **Introducir información de la zona**. El caso de uso se inicia cuando un usuario solicita al sistema el registro de una zona de limpieza. El sistema solicita al usuario los datos asociados a la zona creada: nombre. El usuario provee los datos requeridos e indica al sistema crear la zona. 2. **Crear zona**. El sistema crea una zona de limpieza y la registra en el grupo. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	-
Post-condiciones	El grupo contiene la zona de limpieza recién creada

Tabla 3.32: Caso de uso - listar zonas de limpieza

Caso de uso	CU.27 Listar zonas de limpieza
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve las zonas de limpieza que se han registrado dentro del grupo
Flujo básico	1. **Mostrar lista de zonas de limpieza**. El caso de uso se inicia cuando un usuario solicita al sistema información sobre las zonas de limpieza registradas dentro del grupo. El sistema le muestra al usuario una lista indicando el nombre de cada zona de limpieza. El caso de uso finaliza.
Flujo alternativo	1. **Excepción: No se encuentran zonas de limpieza registradas**. En el paso 1 del flujo básico el resultado de la búsqueda es vacío. El sistema muestra al usuario un mensaje indicando que no existen zonas de limpieza registradas en el grupo.
Pre-condiciones	-
Post-condiciones	-

Tabla 3.33: Caso de uso - editar zona de limpieza

Caso de uso	CU.28 Editar zona de limpieza
Actor	ACT. 2
Descripción	Un Usuario Autenticado edita una zona de limpieza dentro del grupo
Flujo básico	1. **Introducir información de la zona**. El caso de uso se inicia cuando un usuario solicita al sistema la modificación de una zona de limpieza. El sistema solicita la nueva información que debe contener y, simultáneamente, le muestra al usuario la información actual: nombre. El usuario introduce la nueva información e indica al sistema actualizar la zona de limpieza. 2. **Actualizar zona de limpieza**. El sistema actualiza los datos de la zona de limpieza. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo tiene una zona de limpieza registrada
Post-condiciones	La información de la zona de limpieza seleccionada se actualiza

Tabla 3.34: Caso de uso - eliminar zona de limpieza

Caso de uso	CU.29 Eliminar zona de limpieza
Actor	ACT. 2
Descripción	Un Usuario Autenticado elimina una zona de limpieza
Flujo básico	1. **Solicitar eliminación de zona de limpieza**. El caso de uso se inicia cuando un usuario solicita al sistema la eliminación de una zona de limpieza. 2. **Eliminar zona**. El sistema comprueba que no hay ningún plan de limpieza que contenga la zona que se quiere eliminar. El sistema elimina la zona de limpieza seleccionada. El caso de uso finaliza.
Flujo alternativo	1. **Existe una plan de limpieza que contiene la zona a eliminar**. En el paso 2 del flujo básico el sistema determina que la zona que el usuario desea eliminar pertenece a algún plan de limpieza. El sistema muestra un mensaje al usuario por cada dato incorrecto indicando la imposibilidad de borrar esa zona. El caso de uso finaliza.
Pre-condiciones	El grupo dispone de al menos una zona de limpieza registrada
Post-condiciones	La zona de limpieza seleccionada desaparece del grupo

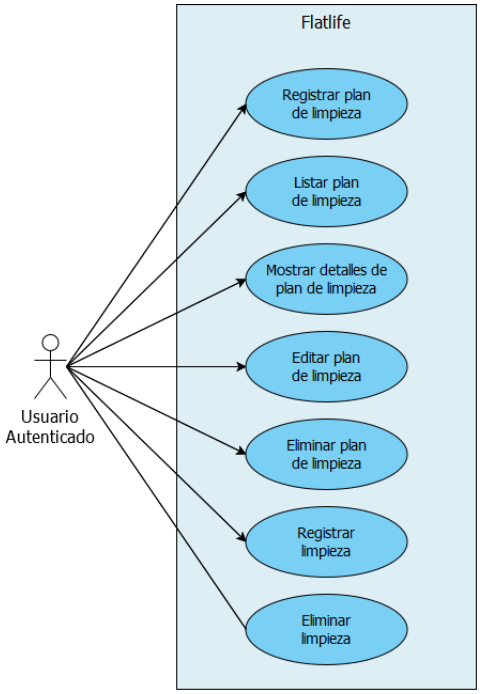


Figura 3.7: Diagrama de casos de uso - planes de limpieza

Tabla 3.35: Caso de uso - registrar plan de limpieza

Caso de uso	CU.30 Registrar plan de limpieza
Actor	ACT. 2
Descripción	Un Usuario Autenticado registra un plan de limpieza
Flujo básico	1. **Introducir información del plan**. El caso de uso se inicia cuando un usuario solicita al sistema el registro de un plan de limpieza. El sistema solicita al usuario los datos asociados al plan a crear: nombre, lista ordenada de usuarios involucrados (limpieza rotacional), zonas pertenecientes a la limpieza, frecuencia y fecha de la próxima limpieza. El usuario provee los datos requeridos e indica al sistema crear el plan de limpieza. 2. **Crear plan**. El sistema crea un plan de limpieza y lo registra en el grupo. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	-
Post-condiciones	El grupo contiene el plan de limpieza recién creado

Tabla 3.36: Caso de uso - listar planes de limpieza

Caso de uso	CU.31 Listar planes de limpieza
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve los planes de limpieza registrados dentro del grupo
Flujo básico	1. **Mostrar lista de planes**. El caso de uso se inicia cuando un usuario solicita al sistema información sobre los planes de limpieza del grupo. El sistema le muestra al usuario una lista con información resumida de todos los planes: nombre, nombre del usuario que debe limpiar a continuación y días restantes hasta la próxima limpieza prevista. El caso de uso finaliza.
Flujo alternativo	1. **Excepción: No se encuentran planes registrados**. En el paso 1 del flujo básico el resultado de la búsqueda está vacío. El sistema muestra al usuario un mensaje indicando que no existen planes de limpieza registrados en el grupo.
Pre-condiciones	-
Post-condiciones	-

Tabla 3.37: Caso de uso - mostrar detalles de un plan de limpieza

Caso de uso	CU.32 Mostrar detalles de un plan de limpieza
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve toda la información asociada a un plan de limpieza determinado que haya elegido
Flujo básico	1. **Mostrar detalles de un plan**. El caso de uso comienza cuando un usuario solicita al sistema información sobre un plan de limpieza del grupo. El sistema le muestra toda la información disponible del plan seleccionado: nombre, zonas, usuarios involucrados, frecuencia, fecha de la próxima limpieza y las limpiezas registradas en dicho plan.
Flujo alternativo	-
Pre-condiciones	El grupo tiene registrado al menos un plan de limpieza
Post-condiciones	-

Tabla 3.38: Caso de uso - editar plan de limpieza

Caso de uso	CU.33 Editar plan de limpieza
Actor	ACT. 2
Descripción	Un Usuario Autenticado edita un plan de limpieza
Flujo básico	1. **Introducir información del plan**. El caso de uso se inicia cuando un usuario solicita al sistema la modificación de un plan de limpieza. El sistema solicita al usuario la nueva información del plan mientras muestra la actual: nombre, usuarios involucrados, zonas, frecuencia y fecha de la próxima limpieza. El usuario provee los nuevos datos e indica al sistema actualizar el plan de limpieza. 2. **Actualizar plan**. El sistema actualiza la información del plan de limpieza seleccionado. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo tiene registrado al menos un plan de limpieza
Post-condiciones	El plan de limpieza seleccionado contiene la nueva información proporcionada por el usuario

Tabla 3.39: Caso de uso - eliminar plan de limpieza

Caso de uso	CU.34 Eliminar plan de limpieza
Actor	ACT. 2
Descripción	Un Usuario Autenticado elimina un plan de limpieza
Flujo básico	1. **Solicitar eliminación de un plan**. El caso de uso se inicia cuando un usuario solicita al sistema la eliminación de un plan de limpieza. El sistema solicita al usuario la confirmación de la acción que se va a llevar a cabo. El usuario confirma la eliminación. 2. **Eliminar plan**. El sistema elimina el plan de limpieza seleccionado. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo dispone de al menos un plan de limpieza registrado
Post-condiciones	El plan de limpieza seleccionado desaparece del grupo

Tabla 3.40: Caso de uso - registrar limpieza

Caso de uso	CU.35 Registrar limpieza
Actor	ACT. 2
Descripción	Un Usuario Autenticado registra una limpieza dentro de un plan de limpieza
Flujo básico	1. **Solicitar registrar limpieza**. El caso de uso se inicia cuando un usuario solicita al sistema que registre una limpieza dentro de un plan de limpieza. El sistema comprueba que el usuario que va a registrar la limpieza es a quien le toca limpiar las zonas especificadas en ese momento. 2. **Registrar limpieza**. El sistema crea una limpieza y le asigna el usuario que la ha registrado y la fecha actual. El caso de uso finaliza.
Flujo alternativo	1. **Usuario no tiene la limpieza asignada**. En el paso 1 del flujo básico, el sistema determina que el usuario que realiza la petición de registro no es quien tenía que limpiar en ese momento. El sistema indica al usuario la situación y solicita la confirmación de completar la ronda de limpieza a su nombre. El usuario confirma la solicitud. El caso de uso finaliza.
Pre-condiciones	El grupo tiene registrado al menos un plan de limpieza
Post-condiciones	El plan de limpieza seleccionado contiene la limpieza recién creada

3.6.2.5. Gestión de eventos

La gestión de los eventos está comprendida en cinco casos de uso, mostrados en la figura 3.8. Corresponden a las operaciones *CRUD* sobre los eventos (crear, leer, modificar y eliminar).

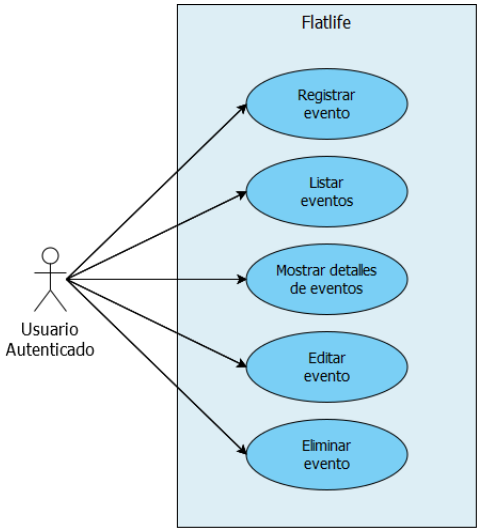


Figura 3.8: Diagrama de casos de uso - eventos

Tabla 3.41: Caso de uso - registrar evento

Caso de uso	CU.36 Registrar evento
Actor	ACT. 2
Descripción	Un Usuario Autenticado registra un evento dentro del grupo
Flujo básico	1. **Introducir información del evento**. El caso de uso se inicia cuando un usuario solicita al sistema el registro de un evento. El sistema solicita al usuario la información del evento: nombre, usuarios asociados, inicio y fin (fechas y horas). El usuario provee los datos e indica al sistema registrar el evento. 2. **Comprobar información**. El sistema verifica que todos los campos tienen información y que son correctos: la fecha de fin tiene que ser posterior a la de inicio. 3. **Registrar evento**. El sistema crea y publica el evento en el grupo. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	-
Post-condiciones	El grupo contiene el evento recién creado

Tabla 3.42: Caso de uso - listar eventos

Caso de uso	CU.37 Listar eventos
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve los eventos registrados dentro del grupo
Flujo básico	1. **Mostrar lista de eventos**. El caso de uso se inicia cuando un usuario solicita al sistema información sobre los eventos registrados en el grupo. El sistema le muestra al usuario una lista con información resumida de todos los eventos: nombre, número de usuarios y días restantes hasta su inicio. El caso de uso finaliza.
Flujo alternativo	1. **Excepción: No se encuentran eventos registrados**. En el paso 1 del flujo básico el resultado de la búsqueda está vacío. El sistema muestra al usuario un mensaje indicando que no existen eventos registrados en el grupo.
Pre-condiciones	-
Post-condiciones	-

Tabla 3.43: Caso de uso - mostrar detalles de un evento

Caso de uso	CU.38 Mostrar detalles de un evento
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve toda la información asociada a un evento determinado que haya elegido
Flujo básico	1. **Mostrar detalles de un evento**. El caso de uso comienza cuando un usuario solicita al sistema información sobre un evento del grupo. El sistema le muestra toda la información disponible del evento seleccionado: nombre, usuarios asociados, inicio y fin.
Flujo alternativo	-
Pre-condiciones	El grupo tiene registrado al menos un evento
Post-condiciones	-

Tabla 3.44: Caso de uso - editar evento

Caso de uso	CU.39 Editar evento
Actor	ACT. 2
Descripción	Un Usuario Autenticado modifica un evento
Flujo básico	1. **Introducir información del evento**. El caso de uso se inicia cuando un usuario solicita al sistema la modificación de un evento. El sistema solicita al usuario la nueva información del evento mientras muestra la actual: nombre, usuarios asociados, inicio y fin. El usuario provee los nuevos datos e indica al sistema actualizar la información del evento. 2. **Actualizar evento**. El sistema actualiza la información del evento seleccionado. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo tiene registrado al menos un evento y el iniciador del caso de uso es el creador del evento seleccionado
Post-condiciones	El evento seleccionado contiene la nueva información proporcionada por el usuario

Tabla 3.45: Caso de uso - eliminar evento

Caso de uso	CU.40 Eliminar evento
Actor	ACT. 2
Descripción	Un Usuario Autenticado elimina un evento de un grupo
Flujo básico	1. **Solicitar eliminación de evento**. El caso de uso se inicia cuando un usuario solicita al sistema la eliminación de un evento. El sistema solicita la confirmación de la acción al usuario. 2. **Eliminar evento**. El usuario confirma la eliminación del evento. El sistema elimina el evento del grupo.
Flujo alternativo	-
Pre-condiciones	El grupo tiene registrado al menos un evento y el iniciador del caso de uso es el creador del evento seleccionado
Post-condiciones	El evento seleccionado desaparece del grupo

3.6.2.6. Gestión de recursos compartidos

La gestión de recursos compartidos, de manera parecida a las limpiezas, se realiza mediante recursos compartidos (figura 3.9) y las reservas de estos recursos (figura 3.10).

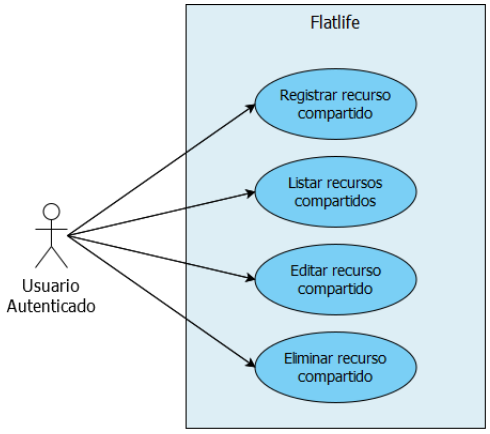


Figura 3.9: Diagrama de casos de uso - recursos compartidos

Tabla 3.46: Caso de uso - registrar recurso compartido

Caso de uso	CU.41 Registrar recurso compartido
Actor	ACT. 2
Descripción	Un Usuario Autenticado registra un recurso compartido
Flujo básico	1. **Introducir información del recurso compartido**. El caso de uso se inicia cuando un usuario solicita al sistema el registro de un recurso compartido. El sistema solicita al usuario los datos asociados al recurso a crear: nombre. El usuario provee los datos requeridos e indica al sistema crear el recurso. 2. **Crear recurso compartido**. El sistema crea un recurso compartido dentro del grupo. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	-
Post-condiciones	El grupo contiene el recurso compartido recién creado

Tabla 3.47: Caso de uso - listar recursos compartidos

Caso de uso	CU.42 Listar recursos compartidos
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve los recursos compartidos que se han registrado dentro del grupo
Flujo básico	1. **Mostrar lista de recursos compartidos**. El caso de uso se inicia cuando un usuario solicita al sistema una lista de todos los recursos compartidos registrados en el grupo. El sistema le muestra al usuario dicha lista, mostrando el nombre de cada recurso. El caso de uso finaliza.
Flujo alternativo	1. **Excepción: No se encuentran recursos compartidos**. En el paso 1 del flujo básico el resultado de la búsqueda de recursos es vacío. El sistema muestra al usuario un mensaje indicando que no existen recursos registrados en el grupo.
Pre-condiciones	-
Post-condiciones	-

Tabla 3.48: Caso de uso - editar recurso compartido

Caso de uso	CU.43 Editar recurso compartido
Actor	ACT. 2
Descripción	Un Usuario Autenticado edita un recurso compartido dentro del grupo
Flujo básico	1. **Introducir información del recurso**. El caso de uso se inicia cuando un usuario solicita al sistema la modificación de un recurso compartido. El sistema solicita la nueva información que debe contener y, simultáneamente, le muestra al usuario la información actual: nombre. El usuario introduce la nueva información e indica al sistema actualizar el recurso compartido. 2. **Actualizar recurso**. El sistema actualiza los datos del recurso compartido. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo tiene un recurso compartido registrado
Post-condiciones	La información del recurso seleccionado se actualiza

Tabla 3.49: Caso de uso - eliminar recurso compartido

Caso de uso	CU.44 Eliminar recurso compartido
Actor	ACT. 2
Descripción	Un Usuario Autenticado elimina un recurso compartido de un grupo
Flujo básico	1. **Solicitar eliminación de recurso compartido**. El caso de uso se inicia cuando un usuario solicita al sistema la eliminación de un recurso compartido. 2. **Eliminar recurso**. El sistema comprueba que no hay ninguna reserva realizada sobre el recurso que se va a eliminar. El sistema elimina el recurso compartido del grupo. El caso de uso finaliza.
Flujo alternativo	1. **Existe reserva sobre el recurso a eliminar**. En el paso 2 del flujo básico, el sistema determina que existe al menos una reserva sobre el recurso a eliminar. El sistema informa al usuario sobre la situación y la imposibilidad de llevar a cabo la eliminación. El caso de uso finaliza.
Pre-condiciones	El grupo tiene registrado al menos un recurso compartido
Post-condiciones	El recurso seleccionado desaparece del grupo si no estaba reservado

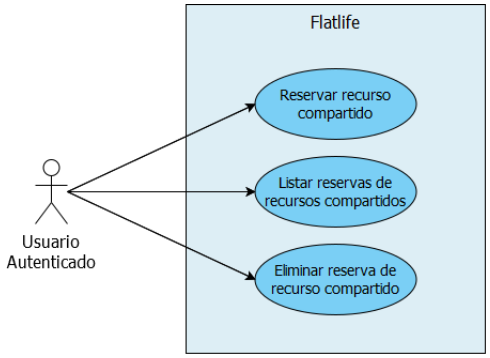


Figura 3.10: Diagrama de casos de uso - reservas de recursos compartidos

Tabla 3.50: Caso de uso - reservar recurso compartido

Caso de uso	CU.45 Reservar recurso compartido
Actor	ACT. 2
Descripción	Un Usuario Autenticado reserva un recurso compartido en una fecha y hora determinada
Flujo básico	1. **Introducir información de la reserva**. El caso de uso se inicia cuando un usuario solicita al sistema la reserva de un recurso compartido. El sistema solicita al usuario la información de la reserva: recurso, fecha de inicio y fecha de fin. El usuario provee los datos e indica al sistema reservar el recurso. 2. **Comprobar reserva**. El sistema comprueba que no existe una reserva sobre el mismo recurso que se solape con la reserva a crear. 3. **Registrar reserva**. El sistema crea y publica la reserva en el grupo. El caso de uso finaliza.
Flujo alternativo	1. **Solape de reservas**. En el paso 2 del flujo básico, el sistema determina que existe una reserva sobre el mismo recurso y que se solapa en el tiempo con la nueva reserva a crear. El sistema informa al usuario la situación y la imposibilidad de reservar el recurso. El caso de uso retorna al paso 1 del flujo básico con la información introducida por el usuario.
Pre-condiciones	-
Post-condiciones	El recurso compartido seleccionado por el usuario queda registrado como reservado

Tabla 3.51: Caso de uso - listar reservas de recursos compartidos

Caso de uso	CU.46 Listar reservas de recursos compartidos
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve las reservas de recursos compartidos que se han registrado dentro del grupo para un recurso
Flujo básico	1. **Mostrar lista de reservas de recursos compartidos**. El caso de uso se inicia cuando un usuario solicita al sistema una lista de todas las reservas de un recurso compartido registradas en el grupo. El sistema le muestra al usuario dicha lista, mostrando para cada reserva el nombre del recurso, el nombre del usuario que la ha realizado, la fecha de inicio y la fecha de fin. El caso de uso finaliza.
Flujo alternativo	1. **Excepción: No se encuentran reservas del recurso compartido**. En el paso 1 del flujo básico el resultado de la búsqueda de reservas está vacío. El sistema muestra al usuario un mensaje indicando que no existen reservas registradas sobre el recurso seleccionado en el grupo. 2. **Filtrado y ordenación**. En el paso 1 del flujo básico, el sistema permite al usuario filtrar las reservas por el recurso reservado. También permite ordenar la lista según la fecha de creación, fecha de comienzo o fecha de fin. El usuario le indica al sistema el filtrado y la ordenación que quiere que se realicen. El caso de uso retorna al paso 1 del flujo básico con la nueva información de filtrado y ordenación.
Pre-condiciones	-
Post-condiciones	-

Tabla 3.52: Caso de uso - eliminar reserva de recurso compartido

Caso de uso	CU.47 Eliminar reserva de recurso compartido
Actor	ACT. 2
Descripción	Un Usuario Autenticado elimina una reserva de un recurso compartido que haya realizado
Flujo básico	1. **Solicitar eliminación de la reserva**. El caso de uso se inicia cuando un usuario solicita al sistema la eliminación de una reserva de un recurso compartido. El sistema solicita al usuario la confirmación de la acción que se va a llevar a cabo. El usuario confirma la eliminación. 2. **Eliminar zona**. El sistema elimina la reserva seleccionada. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El usuario que inicia el caso de uso es el creador de la reserva a eliminar
Post-condiciones	La reserva seleccionada desaparece

3.6.2.7. Lista de productos

La gestión de las listas de productos o listas de la compra puede dividirse en la gestión de las listas en sí (figura 3.11) más la gestión de los productos que incluye (figura 3.12), siendo lo más parecido a cómo una persona la utilizaría haciendo uso de papel y bolígrafo.

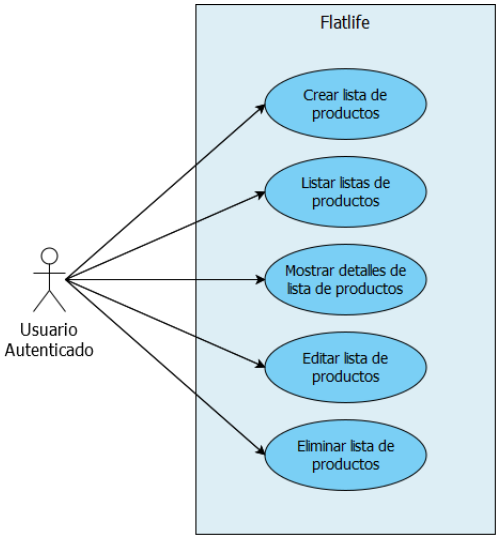


Figura 3.11: Diagrama de casos de uso - listas de productos

Tabla 3.53: Caso de uso - crear lista de productos

Caso de uso	CU.48 Crear lista de productos
Actor	ACT. 2
Descripción	Un Usuario Autenticado crea una lista de productos
Flujo básico	1. **Introducir información de la lista**. El caso de uso se inicia cuando un usuario solicita al sistema la creación de una lista de productos. El sistema solicita al usuario los datos necesarios para crear la lista: nombre. El usuario indica al sistema el nombre de la lista a crear. 2. **Crear lista**. El sistema crea una lista de productos con el nombre proporcionado, sin productos y únicamente con un usuario incluido, el creador. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	-
Post-condiciones	El grupo contiene la lista recién creada

Tabla 3.54: Caso de uso - listar listas de productos

Caso de uso	CU.49 Listar listas de productos
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve las listas de productos compartidas con él dentro del grupo
Flujo básico	1. **Mostrar lista de listas de productos**. El caso de uso se inicia cuando un usuario solicita al sistema una lista de todas las listas de productos compartidas con él. El sistema le muestra al usuario dicha lista indicando, para cada elemento, el nombre, la cantidad de productos por comprar de la lista y el número de usuarios con quienes se comparte la lista. El caso de uso finaliza.
Flujo alternativo	1. **Excepción: No se encuentran listas de productos**. En el paso 1 del flujo básico el resultado de la búsqueda está vacío. El sistema muestra al usuario un mensaje indicando que no existe ninguna lista de productos compartida con él en el grupo.
Pre-condiciones	-
Post-condiciones	-

Tabla 3.55: Caso de uso - mostrar detalles de una lista de productos

Caso de uso	CU.50 Mostrar detalles de una lista de productos
Actor	ACT. 2
Descripción	Un Usuario Autenticado ve toda la información asociada a una lista de productos determinada que haya elegido
Flujo básico	1. **Mostrar detalles de una lista de productos**. El caso de uso comienza cuando un usuario solicita al sistema información sobre una lista de productos. El sistema le muestra toda la información disponible de la lista seleccionada: nombre, usuarios con quien se comparte la lista y productos de la lista, indicando de cada producto el nombre y la cantidad.
Flujo alternativo	-
Pre-condiciones	El grupo tiene registrado al menos una lista de productos
Post-condiciones	-

Tabla 3.56: Caso de uso - editar lista de productos

Caso de uso	CU.51 Editar lista de productos
Actor	ACT. 2
Descripción	Un Usuario Autenticado edita la información de una lista de productos
Flujo básico	1. **Introducir información de la lista**. El caso de uso se inicia cuando un usuario solicita al sistema la modificación de una lista de productos. El sistema le solicita la nueva información mostrando al usuario la información actual de la lista que se desea modificar: nombre y usuarios con quienes compartir la lista, los cuales pertenecen al grupo. 2. **Actualizar lista**. El sistema actualiza los datos de la lista. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo dispone de al menos una lista de productos.
Post-condiciones	La información de la lista seleccionada se modifica

Tabla 3.57: Caso de uso - eliminar lista de productos

Caso de uso	CU.52 Eliminar lista de productos
Actor	ACT. 2
Descripción	Un Usuario Autenticado elimina una lista de productos
Flujo básico	1. **Solicitar eliminación de lista**. El caso de uso se inicia cuando un usuario solicita al sistema la eliminación de una lista de productos. El sistema solicita al usuario la confirmación de la acción que se va a llevar a cabo. El usuario confirma la eliminación. 2. **Eliminar lista**. El sistema elimina la lista de productos del grupo. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo dispone de al menos una lista de productos
Post-condiciones	El grupo deja de contener la lista recién eliminada

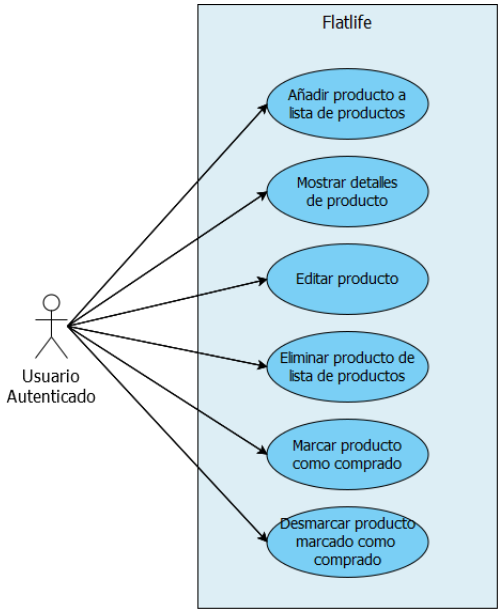


Figura 3.12: Diagrama de casos de uso - productos de la compra

Tabla 3.58: Caso de uso - añadir producto a lista de productos

Caso de uso	CU.53 Añadir producto a una lista de productos
Actor	ACT. 2
Descripción	Un Usuario Autenticado añade un producto a una lista de productos
Flujo básico	1. **Introducir información del producto**. El caso de uso se inicia cuando un usuario solicita al sistema la adición de un producto a una lista. El sistema solicita al usuario la información necesaria para crear el producto: nombre y cantidad. El usuario rellena los campos con la información requerida e indica al sistema la adición del producto. 2. **Añadir producto a la lista**. El sistema crea el producto y lo introduce en la lista. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo dispone de al menos una lista de productos
Post-condiciones	La lista seleccionada contiene el producto recién creado

Tabla 3.59: Caso de uso - editar producto de lista de productos

Caso de uso	CU.54 Editar producto de una lista de productos
Actor	ACT. 2
Descripción	Un Usuario Autenticado edita un producto de una lista de productos
Flujo básico	1. **Introducir información del producto**. El caso de uso se inicia cuando un usuario solicita al sistema la modificación de un producto de una lista. El sistema muestra al usuario la información actual de dicho producto y solicita al usuario la nueva información del producto: nombre y cantidad. El usuario indica la nueva información e indica al sistema la actualización del producto. 2. **Actualizar producto**. El sistema actualiza la información del producto. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo dispone de una lista de productos con al menos un producto
Post-condiciones	El producto seleccionado tendrá la nueva información suministrada por el usuario

Tabla 3.60: Caso de uso - eliminar producto de lista de productos

Caso de uso	CU.55 Eliminar producto de una lista de productos
Actor	ACT. 2
Descripción	Un Usuario Autenticado elimina un producto de una lista de productos
Flujo básico	1. **Eliminar el producto**. El caso de uso se inicia cuando un usuario solicita al sistema la eliminación de un producto de una lista. El sistema elimina el producto de la lista. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo dispone de una lista de productos con al menos un producto
Post-condiciones	La lista seleccionada deja de contener el producto eliminado

Tabla 3.61: Caso de uso - marcar como comprado un producto de una lista de productos

Caso de uso	CU.56 Marcar como comprado un producto de una lista de productos
Actor	ACT. 2
Descripción	Un Usuario Autenticado marca un producto como comprado
Flujo básico	1. **Marcar producto como comprado**. El caso de uso se inicia cuando un usuario solicita al sistema marcar un producto de una lista como comprado. El sistema lo marca como comprado. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo dispone de una lista de productos con al menos un producto
Post-condiciones	El producto seleccionado está marcado como comprado

Tabla 3.62: Caso de uso - desmarcar producto marcado como comprado

Caso de uso	CU.57 Desmarcar producto marcado como comprado
Actor	ACT. 2
Descripción	Un Usuario Autenticado desmarca un producto marcado como comprado
Flujo básico	1. **Desmarcar producto marcado como comprado**. El caso de uso se inicia cuando un usuario solicita al sistema desmarcar un producto de una lista previamente marcado como comprado. El sistema lo desmarca. El caso de uso finaliza.
Flujo alternativo	-
Pre-condiciones	El grupo dispone de una lista de productos con al menos un producto marcado como comprado
Post-condiciones	El producto seleccionado deja de estar marcado como comprado

3.7 Atributos del sistema

3.7.1. Fiabilidad

La aplicación confía en Firebase para el almacenaje de datos e imágenes. En cuanto al modo sin conexión, se almacenarán los datos en una base de datos local al dispositivo hasta que se puedan enviar al servidor sin que haya pérdida de información.

3.7.2. Disponibilidad

Al utilizar Firebase como servidor, la aplicación estará disponible para su uso de manera ininterrumpida ya que todas las características que hacen uso de la conexión a Internet pasan por él. Además, se podrá hacer uso de casi toda la funcionalidad de Flatlife sin tener conexión a Internet. Para ello, se guardan las acciones que va realizando el usuario localmente y se espera hasta que pueda establecerse una comunicación entre el servidor

y el dispositivo móvil. Una vez esta comunicación es posible, se realizan las peticiones pertinentes al servidor para que se actualicen los datos.

3.7.3. Seguridad

Para mantener el sistema lo más seguro posible se utilizarán siempre que sea posible las versiones más recientes de todos los frameworks o librerías utilizadas, que serán las que tengan menos fallos, incluidos fallos de seguridad.

Por otra parte, Flatlife tendrá un sistema de autenticación a través de Firebase para que los usuarios únicamente puedan acceder a la información del grupo en el que se encuentran siempre que ésta sea visible. Como cabe esperar, se aplicará una encriptación unidireccional a la contraseña de los usuarios cuando se guarde en la base de datos para que ninguna persona pueda descubrirla.

Finalmente, todo intercambio de información entre la aplicación y el servidor será cifrado para evitar posibles ataques man-in-the-middle (MitM)[4].

3.7.4. Mantenibilidad

Una característica deseable en todo código es que sea fácil de mantener, es decir, simple de comprender y rápido de modificar. Además, se buscará la reutilización de código evitando la redundancia del mismo. Para ello se seguirán las convenciones del lenguaje de programación que se utilice, además de utilizar la arquitectura de aplicaciones que mejor se adapte a la solución propuesta.

[4] https://es.wikipedia.org/wiki/Ataque_de_intermediario

CAPÍTULO 4
Análisis

Después de tener claro cuál será el funcionamiento del sistema, se va a realizar un análisis del proyecto indicando las diferentes opciones que existen para desarrollar Flatlife, cuál de todas ellas se va a utilizar y los riesgos que pueden darse junto con su plan de acción.

4.1 Identificación y análisis de soluciones posibles

Como ya se ha explicado, Flatlife va a ser una aplicación móvil multiplataforma, disponible tanto en IOS como en Android. Con esto en mente, las aproximaciones actuales más conocidas son las que se muestran a continuación.

La primera idea que se puede plantear es la implementación de dos aplicaciones distintas con la misma funcionalidad, cada una para su plataforma destino y, por lo tanto, haciendo uso del lenguaje de programación soportado por cada uno de los kits de desarrollo (SDK - Software Development Kit). Así pues, tendríamos una aplicación para Android escrita en Java o Kotlin y otra aplicación para IOS escrita en Objective-C o Swift. Este enfoque tiene sus ventajas, pero también sus inconvenientes. Una de las ventajas es que, al trabajar con código nativo de cada plataforma, se consiguen aplicaciones que se ejecutan muy rápido, además de proporcionar una flexibilidad mayor y tener disponibles todas las características de la plataforma incluyendo las más recientes. La rapidez de ejecución en comparación con otros enfoques se debe al uso de la compilación Ahead-of-time (AOT) a código máquina. Hasta la versión 5.0 de Android, éste utilizaba la máquina virtual Dalvik Virtual Machine (DVM) para ejecutar código a través de una compilación Just-in-Time (JIT). No obstante, con la llegada del entorno de ejecución Android Runtime (ART), la plataforma ha adoptado el enfoque AOT [5]. Gracias a éste, el código de alto nivel se compila a código máquina cuando la aplicación se instala para evitar tener que traducir las instrucciones de bytecode a instrucciones máquina cada vez que se ejecuta la misma. Por otra parte, la compilación de una aplicación en IOS se hace directamente a código máquina a través de su compilador basado en LLVM llamado de la misma manera (LLVM). En contraposición a las ventajas mencionadas, adoptar este enfoque significa mantener simultáneamente varios proyectos, cada uno escrito en un lenguaje de programación distinto sin ninguna conexión. Además del mantenimiento, también dificulta la implementación ya que será necesario conocer dos lenguajes o aumentar los recursos humanos con tal de tener a expertos de varias áreas.

El segundo y tercer enfoque aparecen con la intención de facilitar la creación de aplicaciones multiplataforma haciendo uso de una única tecnología. En este apartado se encuentran las aplicaciones híbridas [6], llamadas así porque mezclan funcionalidad nativa de la plataforma destino junto con la que aportan otras tecnologías. Tal y como explica

Matt Kremer del equipo de Ionic[1] en una publicación [7], las aplicaciones híbridas pueden dividirse en dos: aplicaciones híbridas web y aplicaciones híbridas nativas.

Las aplicaciones híbridas web, como es el caso de las creadas con el framework Ionic, pueden ejecutarse en diversas plataformas haciendo uso de la tecnología web. Así pues, se pueden crear aplicaciones de escritorio, aplicaciones iOS y Android o aplicaciones web progresivas (PWA – Progressive Web App) utilizando HTML, CSS y JavaScript. Para ello, la aplicación se ejecuta en un contenedor de la plataforma destino capaz de leer y ejecutar la tecnología web. También es posible acceder a funcionalidades nativas a través de APIs.

Por otro lado, las aplicaciones híbridas nativas pueden ser creadas con frameworks como React Native[2] o NativeScript[3]. Estas aplicaciones utilizan código nativo para que la experiencia del usuario sea más fluida. En el caso de los dos framworks mencionados anteriormente, la aplicación se crea utilizando JavaScript por lo que es necesario un intérprete de este lenguaje para ejecutar el código, que, en tiempo de ejecución, se comunicará con el código nativo para renderizar los diferentes componentes.

Finalmente se puede encontrar un nuevo enfoque aportado por Google, Flutter[4] [8]. Flutter es un kit de herramientas que incluye su propio SDK. Este SDK trae su propio motor de renderizado 2D, así como el manejo de entradas y eventos. Es por eso que Flutter no utiliza los componentes gráficos de los SDK propios de Android o iOS. Esto no quiere decir que no utilice las SDK de estas plataformas ya que éstas serán necesarias en el proceso de compilación. En este caso se empleará el lenguaje Dart y, a diferencia de React Native, Flutter compila el código de alto nivel a código nativo ya sea para arquitecturas ARM o para arquitecturas x86 (no existen a penas dispositivos con esta arquitectura), por lo que no necesita un intérprete y mejora tanto la velocidad de ejecución como el tamaño de la aplicación.

4.2 Solución propuesta

En este proyecto se quiere crear una aplicación multiplataforma con un framework de trabajo que facilite el mantenimiento y la programación de la misma, sin tener que mantener código tanto para Android como para iOS. Además, se busca que parte del código o todo ello se ejecute de manera rápida y eficiente de forma nativa. También se busca que el framework tenga suficiente soporte y sea capaz de cumplir los requisitos de la aplicación. Para ello, se han barajado las dos opciones más respaldadas que son React Native, creado y mantenido por Facebook y la comunidad y Flutter, creado y mantenido por Google y la comunidad. Como ya se ha comentado, la diferencia más notable entre ambas en cuanto a arquitectura es que React Native hace uso de un intérprete de JavaScript para ejecutar el código y comunicarse con el código nativo, mientras que Flutter compila directamente su código a código máquina y su motor se encarga de manejar todos los eventos. No obstante, también se debe tener en cuenta que la primera versión estable de Flutter fue lanzada en diciembre de 2018 mientras que la de React Native fue en junio de 2015. Por este motivo y por el uso de un lenguaje que desconozco por parte de Flutter, he decidido optar por React Native, un framework con cierto tiempo en el mercado, más maduro y con un gran número de contribuyentes.

[1] https://ionicframework.com/
[2] https://facebook.github.io/react-native/
[3] https://www.nativescript.org/
[4] https://flutter.dev/

4.3 Análisis de riesgos

Como en todo proyecto, siempre existen riesgos asociados que pueden ser de carácter económico, tecnológico y/o legal, entre otros. Los riesgos son condiciones cuya aparición es incierta y que, si aparece, tiene un efecto en alguno de los objetivos del proyecto, ya sea económico, temporal, etc. En ocasiones se asocia el término riesgo a una amenaza, pero no hay que confundirlo. Los riesgos pueden causar un impacto tanto positivo como negativo en los objetivos. En caso de que este impacto sea positivo, se le llama oportunidad, y en caso contrario amenaza.

Así pues, se procede a enunciar los riesgos con más impacto considerados para este proyecto, indicando para cada uno, una breve descripción, tipo, categoría, probabilidad, impacto y acciones a realizar en caso de que se materialice. Tanto la probabilidad como el impacto se evalúan cualitativamente haciendo uso de la siguiente escala: muy bajo, bajo, medio, alto, muy alto.

Tabla 4.1: Riesgo - el framework y las librerías de trabajo son insuficiente

Descripción	El framework de trabajo, React Native, junto con la funcionalidad de las librerías de terceros no son suficientes para implementar Flatlife al completo y se hace necesario trabajar con código nativo de cada una de las plataformas
Tipo	Amenaza
Categoría	Tecnológico
Probabilidad	Baja
Impacto	Alto
Acciones	Realizar la aplicación únicamente para Android debido a la disponibilidad de los recursos necesarios para trabajar con un entorno que permite su programación, generación de la aplicación y emulación

Tabla 4.2: Riesgo - disponibilidad de librerías de terceros

Descripción	Las librerías de terceros nuevas o existentes ayudan a reducir el tiempo de desarrollo de Flatlife.
Tipo	Oportunidad
Categoría	Tecnológico
Probabilidad	Media
Impacto	Medio
Acciones	Puesto que se trata de una aplicación gratuita, se publicará en la tienda de IOS o Android para recibir información sobre cómo usan la aplicación los usuarios y de sus quejas o solicitudes de mejora. Con esta información se retocará la aplicación hasta tener el producto final

Tabla 4.3: Riesgo - el diseño no contempla todo el trabajo a realizar

Descripción	El diseño de la aplicación no tiene en cuenta todas las necesidades del producto y se necesita más infraestructura con su necesaria programación
Tipo	Amenaza
Categoría	Tecnológico
Probabilidad	Baja
Impacto	Alto
Acciones	En caso de que la funcionalidad que requiera más infraestructura tenga una prioridad baja, no se implementará. Si se trata de una funcionalidad prioritaria, se dejarán de implementar algunas funcionalidades menos prioritarias. Si eso no es posible se reconsiderará la especificación de requisitos.

CAPÍTULO 5
Diseño de la solución

Ahora que ya se ha decidido la tecnología a utilizar para realizar Flatlife, se va a diseñar el sistema específicamente para la tecnología a emplear. Se abordará la elección de los datos que se van a usar y se realizará un mock-up en el que se podrán ver las diferentes pantallas de las que consta Flatlife y de la interacción que habrá entre ellas y con el usuario.

5.1 Diseño de los datos

En este apartado de diseño de datos se van a realizar dos actividades. La primera de ellas es un diagrama de clases con el que se dejará claro cuáles serán los datos que se van a manejar. Posteriormente se ajustarán esos datos a la estructura que sigue Cloud Firestore de colecciones y documentos. Esto servirá para determinar cómo interactúa la aplicación con el servidor de bases de datos.

5.1.1. Diagrama de clases

El diagrama mostrado a continuación muestra la información que contendrá cada entidad de la base de datos como si fueran clases además de mostrar las conexiones entre ellas. No se van a mostrar métodos ya que dentro de Flatlife se va a trabajar sin clases, simplemente con objetos planos de JavaScript.

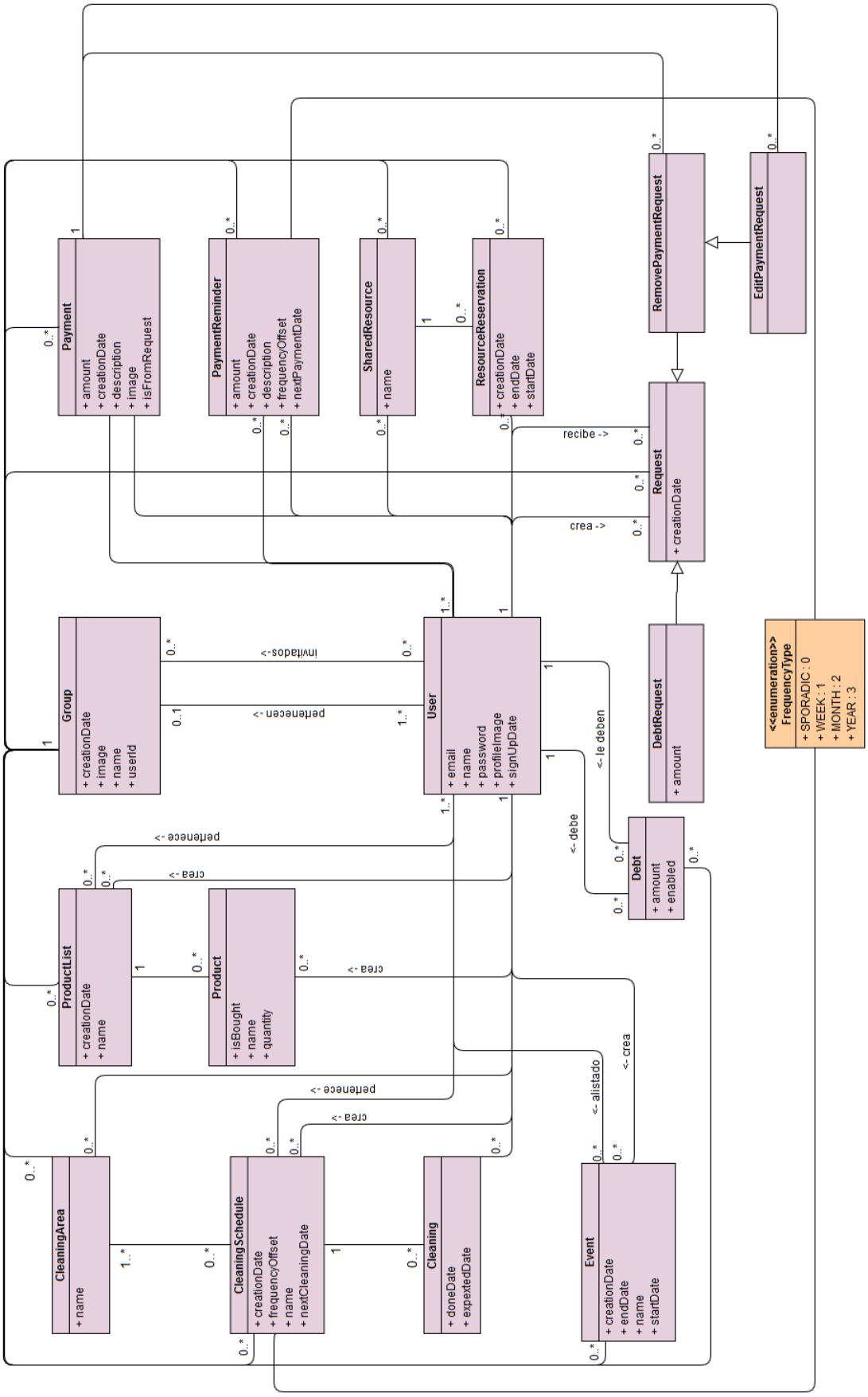


Figura 5.1: Diagrama de clases - Flatlife

5.1.2. Estructura de la base de datos

Al utilizar el servicio Cloud Firestore de Firebase como base de datos, la estructura de la misma estará adaptada a dicho servicio.

Una base de datos de Cloud Firestore es no relacional, por lo que no se utilizará SQL para realizar consultas ni tampoco se utilizan los términos "tablas" y "filas". En cambio, esta base de datos estará formada por documentos, los cuales siempre estarán almacenados en colecciones que no son más que contenedores de documentos. Un documento es un conjunto de pares clave-valor cuya clave es una cadena de caracteres y el valor puede ser de muchos tipos entre los cuales se encuentran la cadena de caracteres, un número, una marca de tiempo, una lista u otras colecciones (subcolecciones). Por otra parte, Cloud Firestore no define un esquema para cada documento que se inserta, por lo que cada uno puede contener una cantidad de campos distinta y pueden ser de tipos distintos.

En cuanto a la estructura que seguirá la base de datos de Flatlife, se muestra en la figura 5.2 el conjunto de colecciones raíz de la aplicación, es decir, aquellas que no son subcolecciones.

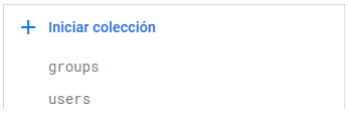


Figura 5.2: Base de datos - colecciones

Para aprovechar la estructura jerárquica que proporciona Cloud Firestore se ha considerado insertar todas las colecciones de documentos que pertenecen a un grupo dentro del documento que representa al grupo, como se ve en las figuras 5.4a y 5.4b. Así solo existirán dos colecciones raíz en la base de datos: la de los grupos y la de los usuarios. En la figura 5.3 se muestra la representación de un usuario dentro de un grupo.

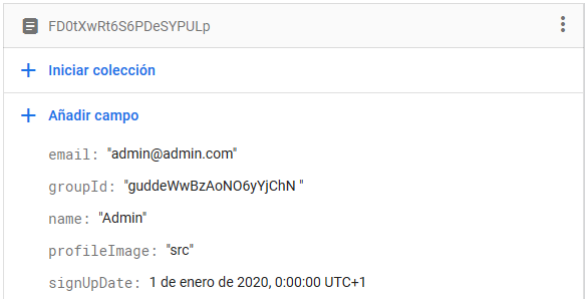
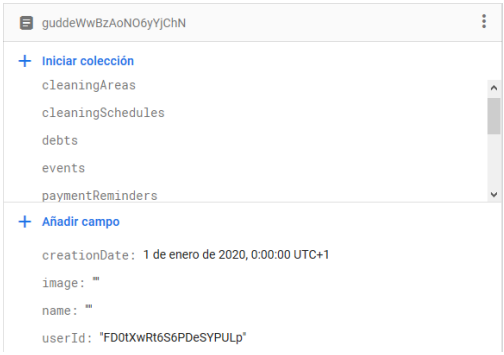
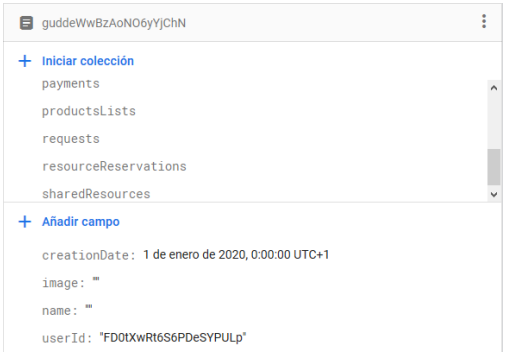


Figura 5.3: Base de datos - usuario. Representa un elemento de la colección *users* de la figura 5.2

En esta representación puede notarse la ausencia de una contraseña para el inicio de sesión del usuario, pero no es necesaria ya que el almacenaje de esta se realizará mediante una API de Firebase llamada Firebase Auth, encargada de la autenticación. Finalmente se muestra cómo se estructuran todos los datos dentro de un documento que representa a un grupo.

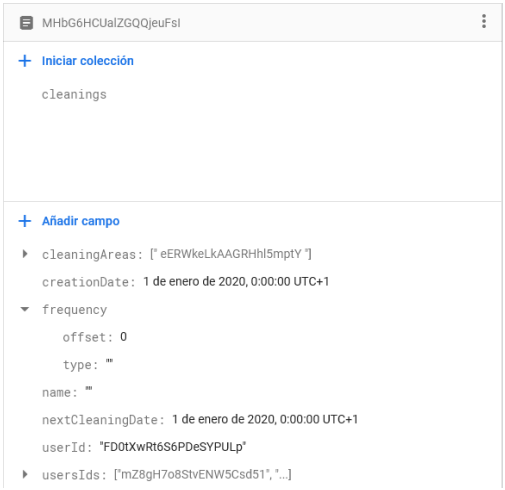


(a) Grupo (1). Representa un elemento de la colección *groups* de la figura 5.2



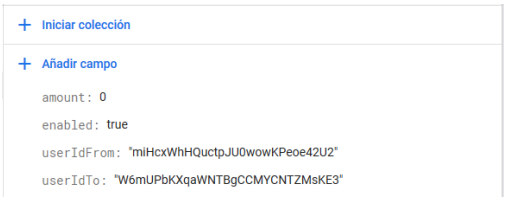
(b) Grupo (2). Representa un elemento de la colección *groups* de la figura 5.2

(c) Area de limpieza. Representa un elemento de la colección *cleaningAreas* de la figura 5.4a



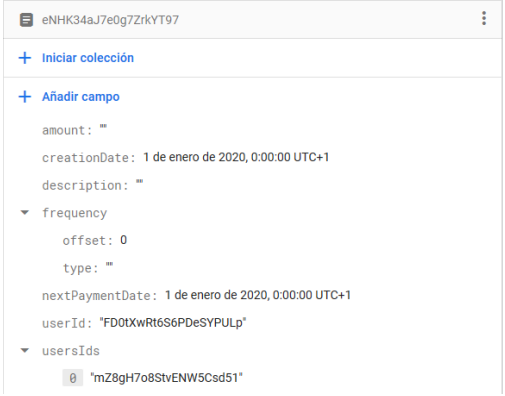
(d) Plan de limpieza. Representa un elemento de la colección *cleaignSchedules* de la figura 5.4a

(e) Limpieza. Representa un elemento de la colección *cleanings* de la figura 5.4d

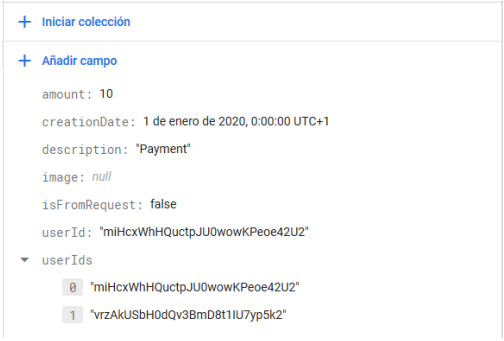


(f) Deuda. Representa un elemento de la colección *debts* de la figura 5.4a

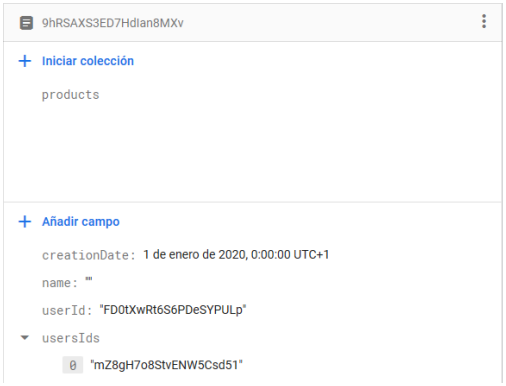
(g) Evento. Representa un elemento de la colección *events* de la figura 5.4a



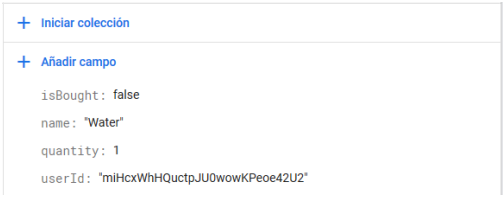
(h) Recordatorio de pago. Representa un elemento de la colección *paymentReminders* de la figura 5.4a



(i) Pago. Representa un elemento de la colección *payments* de la figura 5.4b

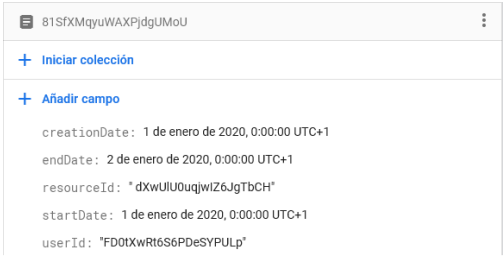


(j) Lista de la compra. Representa un elemento de la colección *productsLists* de la figura 5.4b

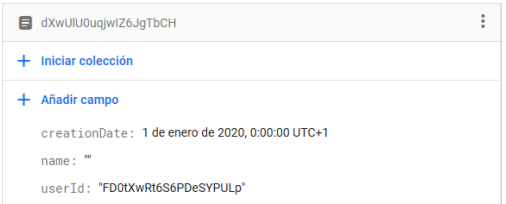


(k) Producto. Representa un elemento de la colección *products* de la figura 5.4j

(l) Petición. Representa un elemento de la colección *requests* de la figura 5.4b



(m) Reserva de recurso. Representa un elemento de la colección *resourceReservations* de la figura 5.4b



(n) Recurso compartido. Representa un elemento de la colección *sharedResources* de la figura 5.4b

Figura 5.4: Base de datos - grupo

Comparando con el diagrama de clases del apartado 5.1.1 se ven algunos cambios. Primeramente, desaparecen todas las especializaciones de la clase *Request* y se juntan todas en la colección *requests* (figura 5.4l). Esto se debe a la posibilidad que nos provee este enfoque para obtener de manera ordenada una cantidad determinada de documentos de la colección *requests* sin realizar más de una petición, por ejemplo, obtener los diez primeros documentos de la colección ordenados por fecha de creación. No obstante, para evitar almacenar más datos de los necesarios se va a guardar cada documento con la cantidad de campos necesarios en función de su tipo, tal y como aparece en el diagrama de clases (figura 5.1). De igual manera que dentro de un grupo se han metido todas las colecciones relacionadas a ese grupo, los productos de una lista de la compra se guardan en una colección dentro de la misma lista y las limpiezas se guardarán en una colección dentro de un plan de limpieza. Otro detalle a tener en cuenta es la manera en que se ha representado la frecuencia, tanto en un recordatorio de pago como en un plan de limpie-

za. Estas frecuencias ahora son mapas con dos datos, el tipo de frecuencia, que representa el enumerado que se ve en el diagrama de clases y el *offset*. También se puede observar una ausencia de referencias y, en cambio, el uso de identificadores para indicar la relación entre varios documentos. La razón es que la única utilidad de utilizar referencias es facilitar el código para el acceso a los documentos a los que apuntan las referencias, y va a ser útil tener los identificadores para más adelante trabajar con React Native y reducir el número de consultas a la base de datos.

5.2 Arquitectura del sistema

Como se va a utilizar React Native para el desarrollo de la aplicación, la arquitectura de la misma se debe adaptar a los patrones utilizados por esta tecnología. Por lo tanto, se ha decidido optar por un diseño de tres capas, parecido a MVC[1] [9]. La aplicación consta de un estado global donde se guardan todos los datos traídos de la base de datos y que puede ser accedido desde cualquier pantalla. El usuario interactúa con las pantallas que le presentan la información. Las pantallas gestionan los eventos producidos por el usuario haciendo modificaciones en su estado local o realizando peticiones de cambio del estado global a la capa de estado. Cuando la capa de estado recibe una petición por parte de alguna pantalla, modifica el estado según manda la petición, pidiendo información, en caso de que sea necesario, a la base de datos a través de la API que muestra la capa de datos. Las funciones públicas de la capa de datos acceden al servidor de base de datos (Firebase Firestore en este caso) enviando peticiones de obtención, modificación o eliminación de los documentos. El funcionamiento de esta arquitectura puede verse en la figura 5.5.

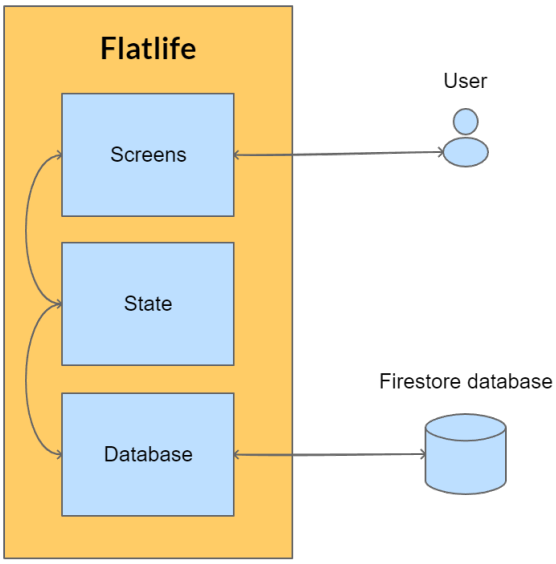


Figura 5.5: Arquitectura de Flatlife

[1] https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

5.3 Tecnología utilizada

Ya se ha ido explicando parte de la tecnología que va a utilizarse en Flatlife. Como framework base de desarrollo se usará React Native, sin hacer uso del SDK de Expo[2] que, aunque aporta funcionalidad al framework, tiene varias limitaciones, como no tener la posibilidad de añadir librerías de terceros o añadir mucho tamaño de aplicación aún cuando mucha funcionalidad no se utilice. Así pues, el lenguaje de desarrollo a utilizar será JavaScript. Como servidor de base de datos se ha optado por Firebase Firestore ya que tiene una API muy sencilla y la interfaz gráfica te permite observar fácilmente los cambios que se están realizando en la base de datos en tiempo real. Además, tiene una librería para React Native implementada en código nativo y los datos los provee en objetos de JavaScript lo que facilita su manejo. No obstante tiene algunas limitaciones a la hora de realizar peticiones, como no admitir el operador lógico OR ni la igualdad negada (!=). Tampoco se pueden utilizar filtros de rango en diferentes campos (edad >0 AND peso <50). Por último, se debe comentar que el IDE que se utilizará es Visual Studio Code[3] con la extensión React Native Tools[4], y que se va a utilizar Git[5] [10] con repositorio en GitHub[6] para llevar un control de las versiones del código. Tener control sobre las versiones del código funciona como un botón de "deshacer" con mucho poder, pudiendo volver a la versión del código guardado que se quiera. Además, permite conocer los cambios entre versiones, el número de líneas de código que se han añadido, qué ficheros son los que más se modifican, etc [11].

5.4 Diseño de la interfaz gráfica

Un buen diseño de la interfaz gráfica facilita el trabajo de los desarrolladores que no tendrán que pensar en la manera de presentar los datos mientras piensan en cómo gestionarlos y en cómo crear la interfaz que los presenta. Para Flatlife, se ha hecho uso de la herramienta Invision Studio[7] para conseguir un diseño organizado. Este programa es gratuito y su comunidad y trabajadores dejan a disposición de todas las personas varios recursos gráficos para crear las interfaces. Uno de estos recursos es un conjunto de componentes gráficos que siguen la guía de diseño de *Material Design* que facilitará la tarea. El diseño de Flatlife - todos los bocetos de las pantallas junto con cierta interacción entre ellas - está disponible en el siguiente enlace: https://projects.invisionapp.com/prototype/ck6se93ie000xna0156oy8to9/play.

Una vez se accede al enlace, se presenta la pantalla de autenticación, desde la cual podemos acceder a las pantallas de registro (pulsando el botón *SIGN UP*), cambio de contraseña (pulsando el texto *I don't remember my password*) y pantalla principal (pulsando el botón *SIGN IN*). Todas las interacciones del diseño se realizan mediante clics con el botón izquierdo del ratón. Cada clic es un cambio de pantalla, de manera que por cada interacción que existe, hay una pantalla asociada a ella. Esto quiere decir que una pantalla de la aplicación comprenderá una o varias pantallas del diseño. Por ejemplo, si al clicar un botón se abre un menú, ese menú forma parte de la misma pantalla de la aplicación (ya que es un elemento que se superpone a ella) mientras que se trata de otra pantalla en el diseño. Cuando se realiza un clic a una zona de la pantalla que no tiene interacción implementada, aparecerán rectángulos azules sobre aquellas zonas que sí reaccionan a

[2] https://expo.io/
[3] https://code.visualstudio.com/
[4] https://marketplace.visualstudio.com/items?itemName=msjsdiag.vscode-react-native
[5] https://git-scm.com/
[6] https://github.com/
[7] https://www.invisionapp.com/studio

los eventos de clicado. De esta forma podemos pasar por todas las pantallas del diseño viendo el flujo de navegación de la aplicación.

La diferencia en el número de pantallas entre el diseño y la implementación es de 78. En el caso del diseño, éstas suman 105 mientras que en la implementación se han realizado 27.

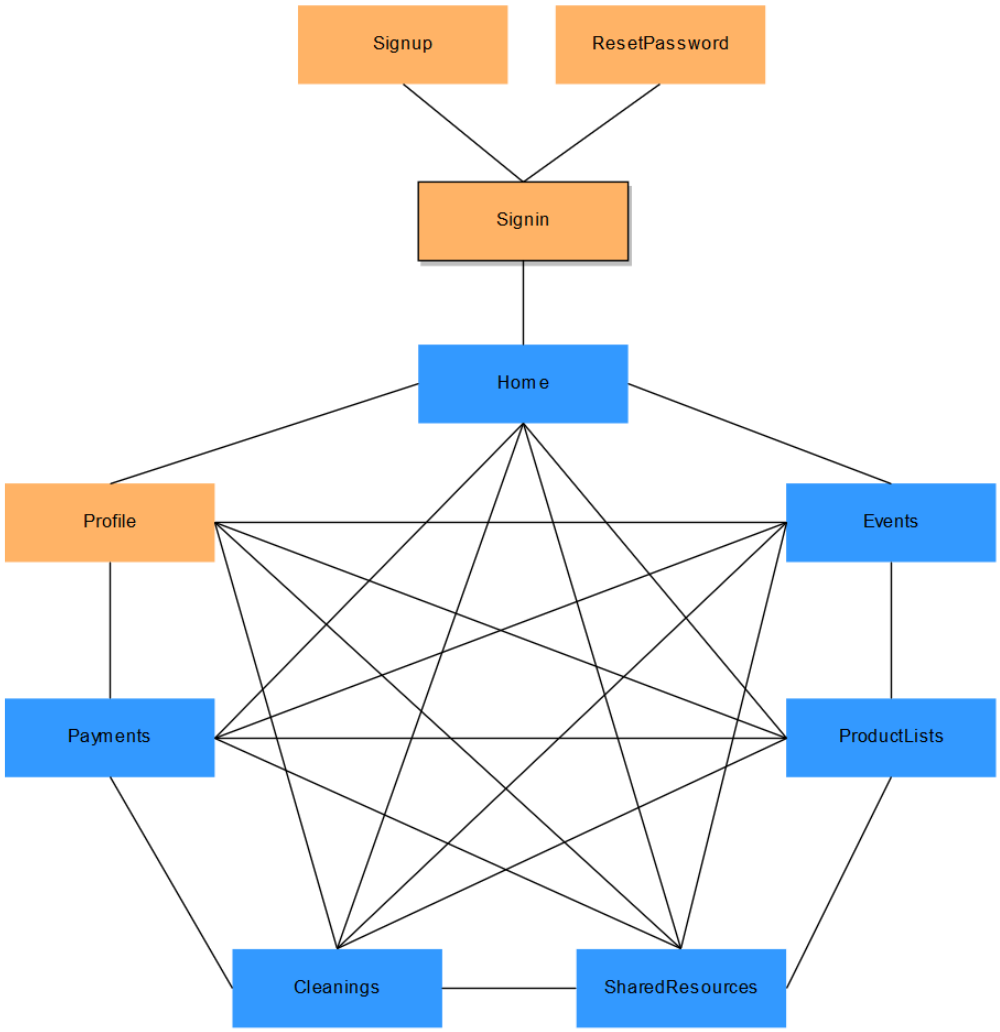
Para tener una idea del flujo de navegación de la aplicación entre todas sus pantallas se puede mirar la figura 5.6 y la descripción de las pantallas que se realizará enseguida.

Para entender la figura 5.6 se necesita saber cómo se ha representado el flujo de navegación. Este flujo es un grafo en el que los nodos pueden ser pantallas (naranjas) o grupos de pantallas (azul) y las aristas representan una transición bidireccional entre ellos. Todos los grupos de pantallas se desglosan en otras figuras. Cuando una arista llega a un grupo de pantallas significa que se puede acceder a cualquiera de ellas y viceversa. También encontraremos cuadrados blancos en los cuales se unen varias aristas cuyo significado es que, desde él, se puede llegar a cualquier nodo conectado. Estos cuadrados hacen que el diagrama sea más sencillo de entender ya que evita que se crucen las aristas. Finalmente, los nodos con borde negro y sombra son los puntos de entrada de cada conjunto de pantallas que aparece en cada diagrama.

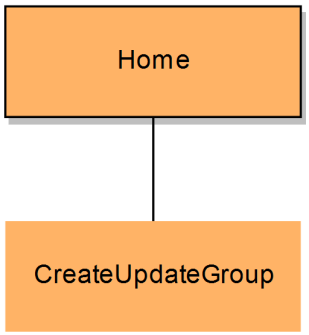
La descripción de todas las pantallas de Fatlife se nombran a continuación (no los grupos de pantallas aunque tengan el mismo nombre).

- **Signin**: permite la autenticación del usuario en Flatlife.
- **Singup**: permite la creación de una cuenta de usuario en Flatlife.
- **ResetPassword**: permite el cambio de contraseña a un usuario no autenticado.
- **Profile**: contiene la información del usuario autenticado y permite modificarla.
- **Home**: muestra el nombre, imagen y usuarios de un grupo. Además, si el usuario no pertenece a un grupo, le da la posibilidad de crear uno y le muestra las peticiones de unión a un grupo pendientes. Desde aquí se pueden echar a usuarios e invitarles a que se unan a tu grupo.
- **CreateUpdateGroup**: recoge la información de un grupo y lo crea o actualiza.
- **PaymentsHome**: muestra las deudas pendientes con el resto de usuario del grupo y lista los pagos que se han realizado.
- **PaymentDetails**: muestra toda la información relacionada con un pago en concreto. Desde aquí se puede eliminar el pago seleccionado.
- **CreateUpdatePayment**: recoge la información de un pago y lo crea o actualiza.
- **PaymentReminders**: lista los recordatorios de pago del grupo.
- **PaymentReminderDetails**: muestra toda la información relacionada con un recordatorio de pago en concreto. Desde aquí se puede eliminar el recordatorio de pago seleccionado.
- **CreateUpdatePaymentReminder**: recoge la información de un recordatorio de pago y lo crea o actualiza en función de un parámetro.
- **PaymentRequests**: lista las peticiones de modificación y eliminación de un pago así como las peticiones de pago de una deuda.

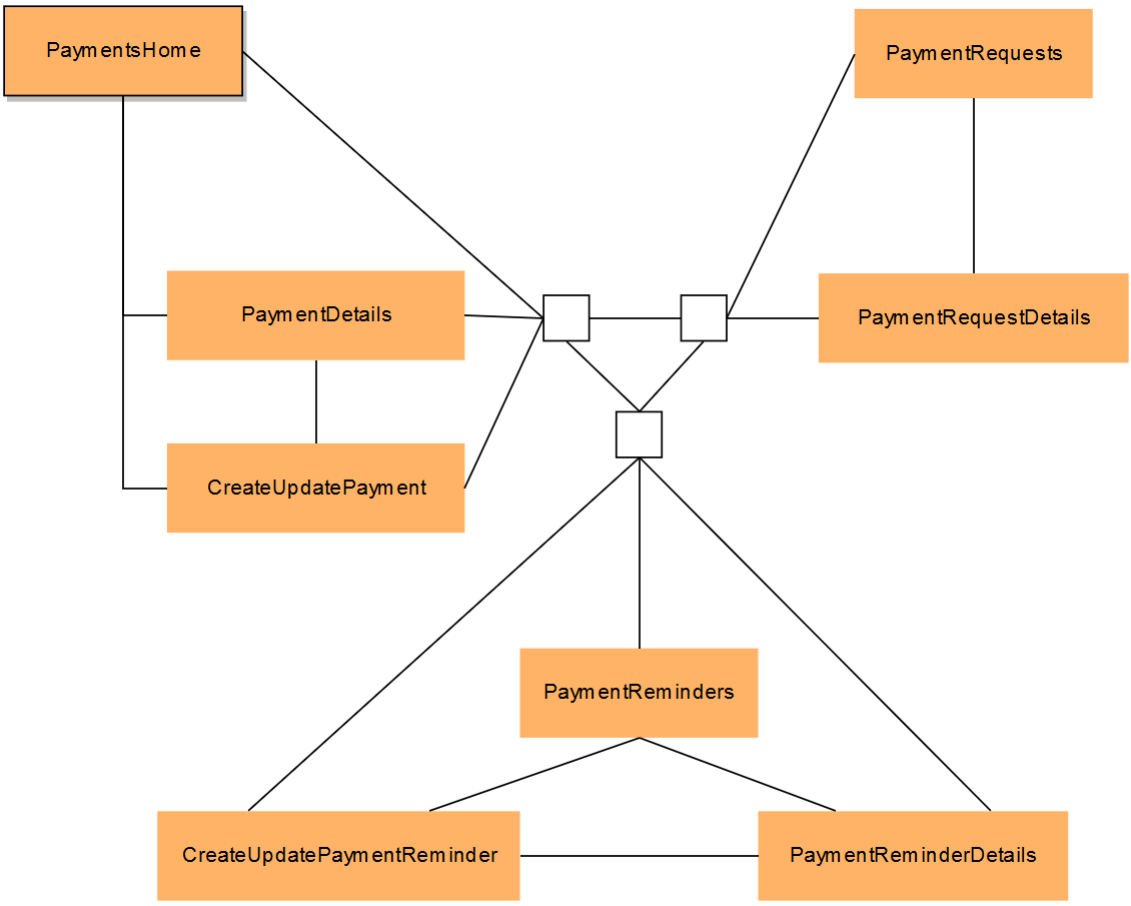
- **PaymentRequestDetails**: muestra toda la información relacionada con una petición en concreto. Desde aquí se puede eliminar la petición seleccionada.
- **CleaningSchedules**: lista los planes de limpieza de un grupo.
- **CleaningSchedulesDetails**: muestra toda la información relacionada con un plan de limpieza en concreto. Desde aquí se puede eliminar el plan de limpieza seleccionado.
- **CreateUpdateCleaningSchedule**: recoge la información de un plan de limpieza y lo crea o actualiza.
- **CleaningAreas**: lista las diferentes áreas que se han creado en el grupo. Desde aquí se crean, modifican y eliminan las mismas.
- **Reservations**: lista las reservas de recursos compartidos dentro del grupo. Permite realizar un filtrado en función del recurso reservado y una ordenación ascendente o descendente por fecha de creación, inicio o fin.
- **ReservationDetails**: muestra toda la información relacionada con una reserva en concreto. Desde aquí se puede eliminar la reserva seleccionada.
- **CreateReservation**: recoge la información de una reserva y la crea.
- **Resources**: lista los diferentes recursos compartidos que se han creado en el grupo. Desde aquí se crean, modifican y eliminan los mismos.
- **ProductLists**: lista las listas de productos del grupo.
- **ProductListDetails**: muestra toda la información de una lista de productos y permite su modificación. También se crean, modifican y eliminan los productos que pertenecen a esta lista desde esta pantalla.
- **Events**: lista los eventos del grupo.
- **EventDetails**: muestra toda la información de un evento en concreto. Desde aquí se puede eliminar el evento seleccionado.
- **CreateUpdateEvent**: recoge la información de un evento y lo crea o actualiza.



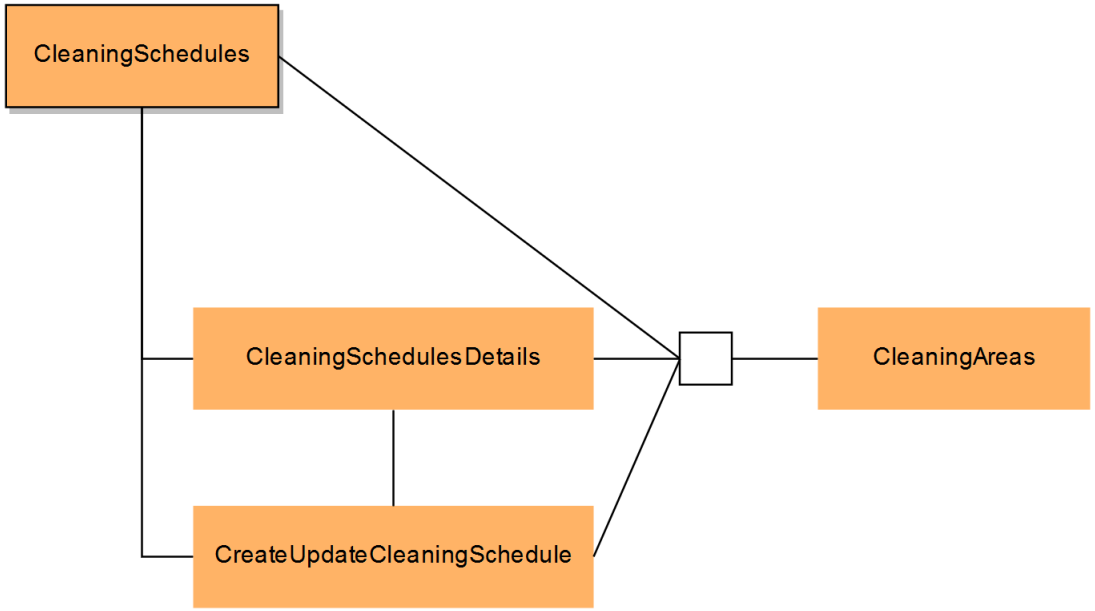
(a) Flujo de navegación sin desglosar



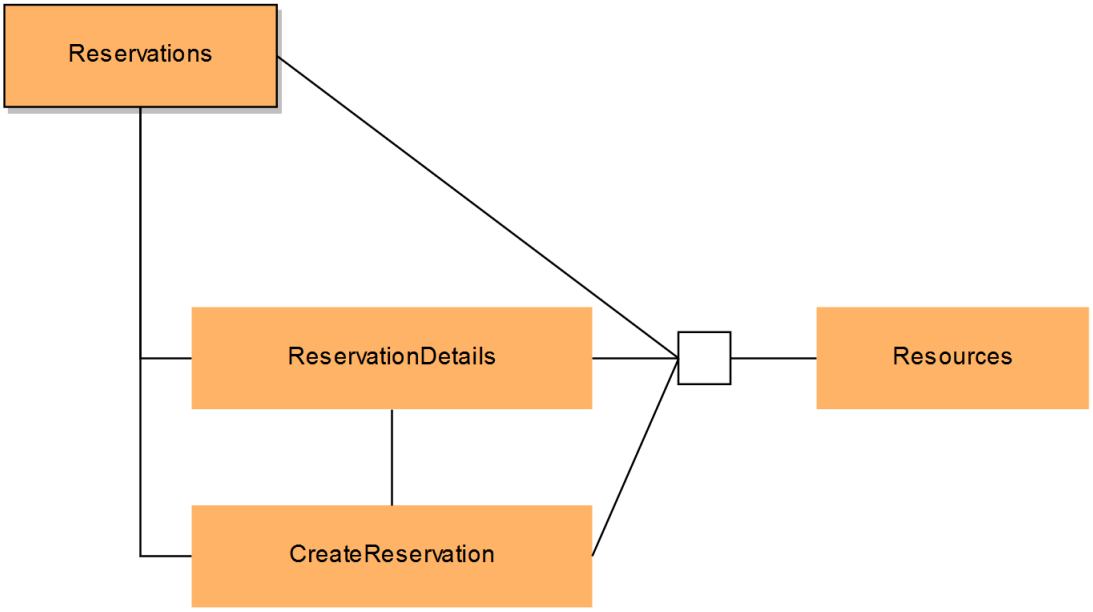
(b) Flujo de navegación del conjunto de pantallas "Home"



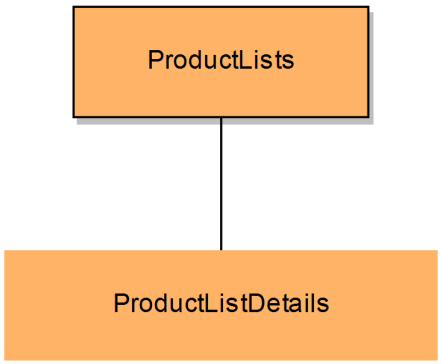
(c) Flujo de navegación del conjunto de pantallas "Payments"



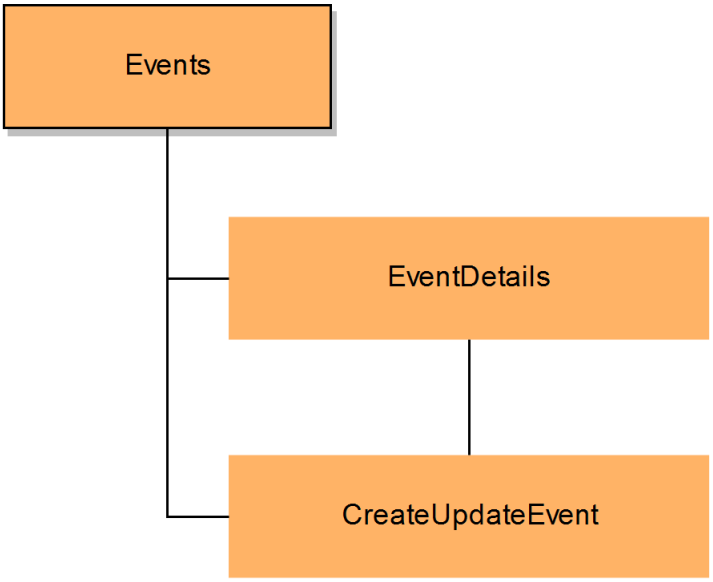
(d) Flujo de navegación del conjunto de pantallas "Cleanings"



(e) Flujo de navegación del conjunto de pantallas "SharedResources"



(f) Flujo de navegación del conjunto de pantallas "ProductLists"



(g) Flujo de navegación del conjunto de pantallas "Events"

Figura 5.6: Flujo de navegación de Flatlife representado mediante grafos no dirigidos.

A continuación se mostrará una descripción del diseño de los componentes gráficos que se utilizan en los bocetos iniciales de Flatlife. No obstante, estos elementos gráficos como colores, formas, iconos, fuentes, distribución de los elementos y algunas interacciones, pueden cambiar ligeramente a lo largo de la implementación hasta el producto final.

Varios puntos clave para dar identidad a Flatlife son los colores que utiliza y su logo. El color principal es el naranja de la figura 5.7a mientras que el color secundario es el azul de la figura 5.7b. El naranja se ha elegido por su relación con la alegría y la energía, buscando que sus usuarios asocien la aplicación con sus compañeros de piso de manera agradable. A su vez, siguiendo la guía de *Material Design*, se ha escogido como color secundario el azul, ya que tiene buen contraste con el naranja.

(a) Color primario: #FF6400

(b) Color secundario: #0099ff

Figura 5.7: Color primario y secundario de Flatlife

Para el logo, figura 5.8, se ha utilizado una casa con una puerta en forma de corazón metida dentro de una hoja de papel. La casa se asemeja a una cara alegre, donde las ventanas son los ojos y el corazón la boca, buscando evocar un sentimiento de amabilidad, para que haya buen clima de compañerismo en el piso. Luego el logo completo puede entenderse como: organización de mi piso, contrato con mis compañeros, la historia de mi piso.

Figura 5.8: Logo de Flatlife

Ahora que ya se han descrito los colores de la aplicación y su logo, se van a explicar algunos de los patrones que se han seguido.

Primero de todo, se va a hablar sobre la navegación, que es la forma en que un usuario accede a las diferentes pantallas de la aplicación. Flatlife contará con un menú lateral que se abrirá presionando un botón situado en el panel superior de la aplicación, llamado a partir de ahora barra de navegación superior. Este botón son tres líneas horizontales una debajo de otra. La barra de navegación superior será como la mostrada en la figura 5.9.

Figura 5.9: Barra de navegación superior de Flatlife

El menú lateral, accesible desde cualquier pantalla, contendrá un elemento por cada funcionalidad de la aplicación a la que se pueda acceder: pagos, lista de la compra, recursos compartidos, eventos, limpiezas y el perfil de usuario. Presionando uno de esos elementos la aplicación cambiará de pantalla. Este menú lateral será como el mostrado en la figura 5.10.

Existen apartados, como por ejemplo el de pagos, que necesitan otra navegación adicional porque trabajan con varias pantallas. El apartado de pagos tiene tres subapartados: pagos, recordatorios de pago y peticiones. La navegación entre subapartados en Flatlife se hará mediante un panel de navegación en la parte inferior de la pantalla, y con color igual al color secundario, que se llamará barra de navegación inferior. Lo mismo pasa con los recursos compartidos y con las limpiezas. La barra de navegación inferior tiene un botón por cada pantalla a la que se puede acceder, con un icono y un texto que indica al usuario en una palabra qué se va a encontrar al pulsar dicho botón. El aspecto de una barra de navegación inferior queda ilustrado en la figura 5.11

Casi para acabar con la navegación, en muchas pantallas aparecerá un botón circular del color secundario de Fatlife, con un símbolo "+" que servirá para crear datos, ya sean pagos, recordatorios de pagos, eventos, etc. Este botón es conocido como botón flotante de acción (*FAB (Floating Action Button)* en inglés) Este botón siempre se encontrará en la parte inferior derecha de la pantalla. Con él, se quiere que los usuarios asocien la acción de crear datos con el botón, para que sea más intuitivo el uso de Flatlife. Se puede ver el aspecto de este botón en la figura 5.12. La pulsación de dicho botón en ocasiones ejecuta un cambio de pantalla.

Finalmente, volvemos a la barra de navegación superior para hablar sobre los elementos que podrá contener. En el centro de la barra se mostrará un texto que indica el propósito de la pantalla. A la izquierda siempre habrá un icono, ya sea para abrir el menú lateral o para volver a la pantalla anterior. En la parte derecha podrán existir dos iconos

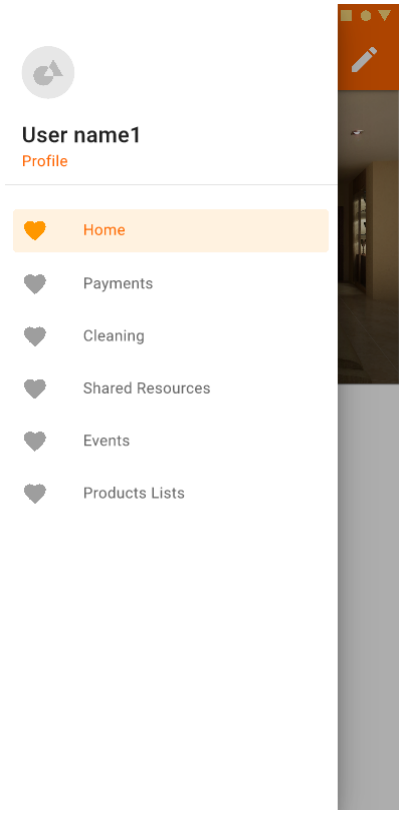


Figura 5.10: Menú de navegación lateral empleado en Flatlife

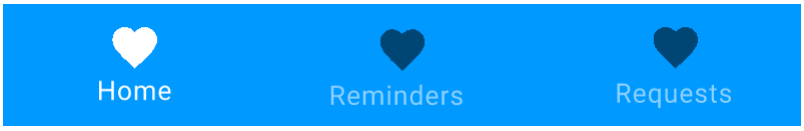


Figura 5.11: Barra de navegación inferior empleado en el apartado de pagos de Flatlife

Figura 5.12: Botón flotante de acción de Flatlife

más, uno de ellos utilizado para editar los datos que se están mostrando en la pantalla y otro para eliminarlos. Para que se entienda, se muestra en la figura 5.13 un ejemplo donde aparecen el botón izquierdo que no se ha mostrado (volver a pantalla anterior) y los dos botones de la derecha.

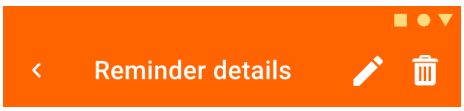


Figura 5.13: Barra de navegación superior completa usada en Flatlife

Pasamos ahora a comentar otros patrones que se repiten en las pantallas y hacen de Flatlife una aplicación con cohesión. Una de las cosas que se repite una y otra vez es el título de la pantalla, que indica, con texto y un fondo naranja curvado en la parte superior, qué información se está mostrando en pantalla. El texto puede ser el nombre de

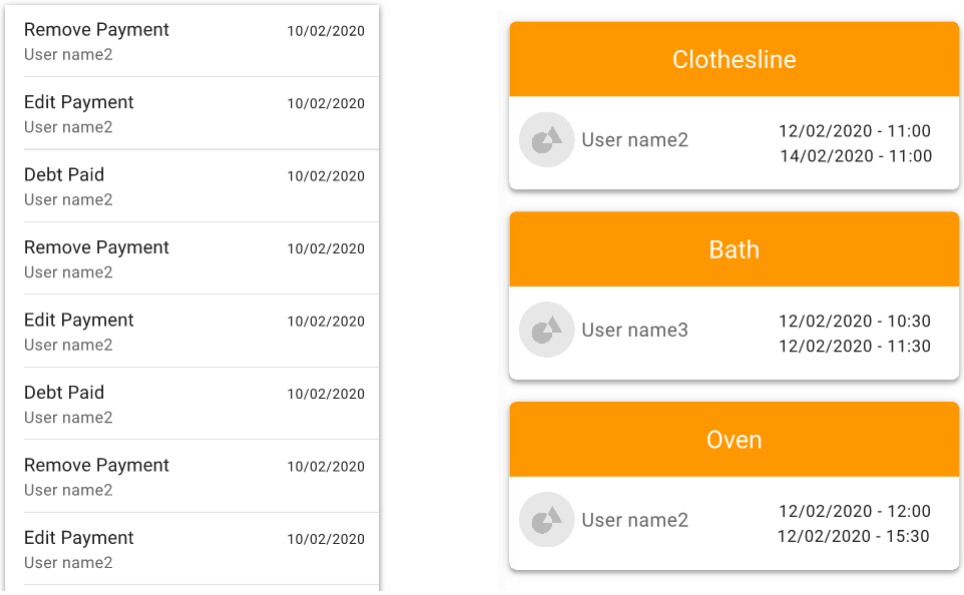
una lista de la compra, el nombre de una reserva de un recurso compartido, el nombre de un evento, etc. Algunos ejemplos de títulos se muestran en la figura 5.14.

(a) Título de la pantalla que muestra información de un recordatorio de pago.

(b) Título de la pantalla que muestra información de una reserva de un recurso compartido.

Figura 5.14: Ejemplos de títulos usados en diferentes pantallas de Flatlife.

Para las listas de elementos que no son muy destacados se utiliza una simple sombra alrededor de los elementos y se separan mediante una barra horizontal. Cuando se quieren realzar los elementos de una lista, se utiliza una sombra para cada uno de ellos, y contienen un título parecido al título de las pantallas (letra blanca sobre fondo naranja). Se ilustra esta explicación en la figura 5.15.



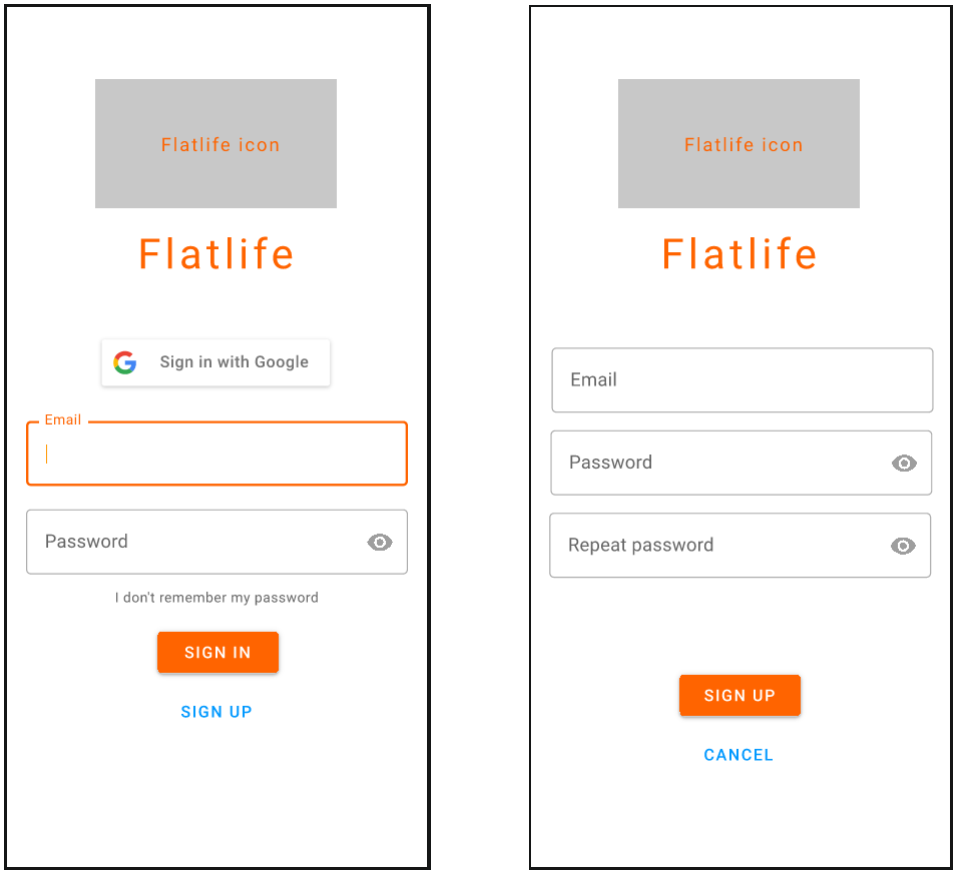
(a) Lista de peticiones. Esto es un ejemplo de lista de elementos no destacados.

(b) Lista de reservas de recursos compartidos. Esto es un ejemplo de lista de elementos destacados.

Figura 5.15: Estilos empleados en las listas que se ven en Flatlife.

Otro apartado importante es la pantalla inicial de la aplicación. Como Flatlife sólo puede ser utilizada teniendo una cuenta, su creación y posterior autenticación deben de ser pasos sencillos. Para conseguirlo, se han introducido dos botones bien visibles que permiten uno la autenticación y otro la creación de cuenta. Además, existe un botón para poder iniciar sesión con la cuenta de Google, por lo que no es necesario crear y recordar una nueva contraseña. Las pantallas de autenticación y creación de cuenta se pueden ver en la figura 5.16. Además, en estas pantallas vemos otro patrón utilizado: los botones principales, los que realizan la acción principal de la pantalla tienen letra blanca sobre un fondo del color primario y una pequeña sombra, mientras que los botones secunda-

rios, o los que queremos que sean menos visibles o se utilicen menos, se destacan menos dejándolos sin fondo y la letra del color secundario.



(a) Pantalla de autenticación.

(b) Pantalla de creación de cuenta.

Figura 5.16: Pantallas de autenticación y creación de cuenta de Flatlife.

CAPÍTULO 6
Desarrollo de la solución

Antes de comenzar a explicar la creación de un proyecto y la estructura de directorios que se va a utilizar, es necesaria una explicación algunos conceptos sobre React Native.

En React Native, todo aquello que aparezca en pantalla es un componente. Además, un componente puede contener otros componentes dentro de él. Esto nos permite dividir un componente en subcomponentes más sencillos y manejables, ayudando a la reutilización de código. Un ejemplo de componente puede ser un texto con el formato que decidas. También puede ser un elemento de una lista, un título, un selector de fechas o incluso una pantalla.

Por otro lado, para permitir la transición entre pantallas se va a hacer uso de la librería de navegación de React *react-navigation*[1]. Esta librería define diferentes Navegadores según la interacción que deseamos, como por ejemplo, aquel que nos permite tener un menú lateral en la aplicación, u otro que nos provee de una barra de navegación inferior.

Ahora sí, para comenzar la aplicación en React Native, una vez tenemos todas las dependencias instaladas utilizaremos el siguiente comando:

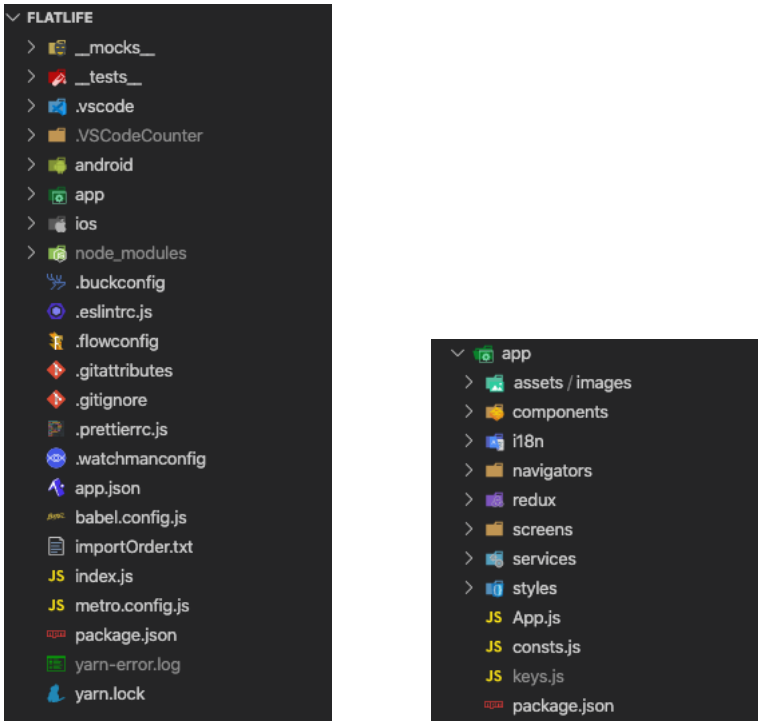
```
$ react-native init Flatlife
```

Este comando nos creará la estructura por defecto de un proyecto. Creamos directorios hasta que conseguimos la estructura deseada.

6.1 Árbol de directorios

Flatlife utiliza el árbol de directorios que aparece en la figura 6.1.

[1] https://reactnavigation.org/



(a) Estructura de directorios raíz

(b) Estructura del directorio "app"

Figura 6.1: Estructura de directorios

En la figura 6.1a se pueden observar varios directorios que vienen por defecto cuando iniciamos el proyecto. Tanto el directorio *android* como *ios* contienen el código nativo de la aplicación en la plataforma correspondiente. *node_modules* contiene todos los paquetes que son dependencias de la aplicación, como todo el framework React Native, la internacionalización, etc. Estos paquetes pueden contener tanto código nativo como código en JavaScript.

El directorio más importante y con el que se va a trabajar mayoritariamente es *app*. Su estructura la podemos ver en la figura 6.1b. El contenido de cada directorio es el siguiente:

- **assets**: recursos estáticos que importa la aplicación como fuentes o imágenes.
- **components**: componentes gráficos que se utilizan en las diferentes pantallas. Cada componente es un directorio con dos ficheros: *index.js* con la estructura y la lógica y *styles.js* con los estilos.
- **i18n**: formada por tantos ficheros como traducciones tenga Flatlife más un fichero que importa esas traducciones y se encarga de utilizar la más adecuada dentro de la aplicación.
- **navigators**: mantiene los diferentes navegadores de *react-navigation* con sus configuraciones.
- **redux**: mantiene la lógica y los datos del estado global de la aplicación. Este directorio se divide en tantos directorios como elementos hay en el menú lateral de la aplicación.
- **screens**: contiene las pantallas que se le presentan al usuario.
- **services**: formada por algunos servicios como la capa de base de datos, la autenticación, la validación de los datos que introduce el usuario, etc.

- **styles**: mantiene los estilos compartidos entre los diferentes componentes y pantallas.
- **App.js**: es el contenedor de toda la aplicación, dentro del cual se encuentran todas las pantallas.
- **consts.js**: mantiene las constantes necesarias como los mensajes de error producidos por Flatlife.
- **keys.js**: tiene todas las claves de servicios externos necesarias como la de Firebase.

6.2 Dependencias

Una vez tenemos clara la estructura de directorios se deben añadir las dependencias de paquetes al proyecto. Para ello podemos utilizar el gestor de paquetes de Node, *npm*, u otro alternativo como *yarn* de la siguiente manera.

```
$ npm install nombre-paquete@version --save
```

```
$ yarn add nombre-paquete@version
```

En la versión de React Native que se utiliza en este proyecto (0.61.5) la conexión entre código JavaScript y código nativo se hace automáticamente por lo que no es necesario realizar ningún paso extra para este propósito. En cambio, es necesario descargar e instalar las dependencias del proyecto para iOS que se acaban de añadir a través del gestor de paquetes CocoaPods[2]. Para ello, desde un terminal situado en el directorio *ios* de la aplicación se ejecuta el siguiente comando.

```
$ pod install
```

Un ejemplo de la instalación de un paquete necesario para Flatlife es el servicio de Firebase. Para añadir la dependencia al proyecto, desde un terminal con directorio de trabajo el directorio raíz, los pasos son los siguientes.

```
$ yarn add @react-native-firebase/app (o npm install @react-native-firebase/app
    --save)
$ cd ios
$ pod install
```

Cabe destacar las dependencias más importantes de este proyecto ya que van a influir en gran medida en su desarrollo.

Como se ha comentado en la sección 5.2, Flatlife dispone de un estado global en el que se mantiene la única fuente de información verdadera compartida por toda la aplicación. Para manejar este estado global se utilizará la librería Redux[3], sobre la cual existen librerías envoltorio para ser utilizada con React Native. Otra de las funcionalidades importantes es la traducción a varios idiomas para que llegue a un mayor público. La librería de internacionalización utilizada es *i18next*[4]. Esta librería también tiene su envoltorio para ser utilizada en una aplicación de React Native (*react-i18next*). Así pues, ya tendríamos todos los textos traducidos a sus respectivos idiomas. Junto con esta librería se utilizará otra llamada *moment*[5] para formatear las fechas en función del idioma de la aplicación.

[2] https://cocoapods.org/
[3] https://redux.js.org/
[4] https://www.i18next.com/
[5] https://momentjs.com/

Finalmente, para facilitar el desarrollo siguiendo la guía de diseño *Material Design* se utilizará la librería *react-native-paper*[6], que nos proporciona un conjunto muy amplio de componentes para cubrir las necesidades de Flatlife. Además, es fácilmente configurable utilizando un tema, objeto que indica a esta librería el estilo de los componentes que nos provee, como colores, tamaño de fuente, curvatura, etc.

6.3 Componentes y pantallas

Como ya se ha comentado, todo lo que aparece en pantalla en una aplicación de React Native es un componente, incluso una pantalla entera. Para implementar un componente podemos utilizar la sintaxis para clases que proporciona JavaScript a partir del estándar EcmaScript 2015[7]. Para ello, todo componente debe extender la clase React.Component. Además, la clase debe contener como mínimo un método llamado *render* que devuelva el propio componente.

Para definir la estructura, React Native utiliza JSX[8], una extensión de JavaScript que aporta una sintaxis parecida a un lenguaje de etiquetas con la opción de introducir código JavaScript y evaluar expresiones dentro de la misma estructura. Por otro lado, un componente tiene estilos que indican cómo tienen que mostrarse en pantalla. Algunos ejemplos son el tamaño de un texto, la posición en pantalla o la separación con el resto de componentes. Estos estilos pueden ser descritos mediante objetos planos, que son conjuntos de pares claves-valor donde la clave es la propiedad a modificar y el valor indica cómo va a afectar esa propiedad en la visualización de un componente. Las propiedades de los estilos en React Native son muy parecidos a los que podemos ver en un fichero CSS (Cascade Style Sheet), pero en vez de separar las distintas palabras que forman el nombre de una propiedad con un guión lo hace poniendo en mayúscula la primera letra de cada palabra exceptuando la primera palabra. Así pues, una propiedad *propiedad-uno* en CSS pasará a ser *propiedadUno* en React Native.

Muchos de los componentes que se van a crear necesitan acceder a las traducciones, al tema o al estado global de la aplicación. Para ello, cada librería aporta un envoltorio que provee de un contexto a todos los componentes llamado *Provider* o proveedor. Estos envoltorios se aplican en la función principal de la aplicación, la que devuelve la aplicación entera tal y como muestra el fragmento de código 6.1.

```
1  import * as React from 'react';
2  import {AppRegistry} from 'react-native';
3  import {name as appName} from './app.json';
4  import App from './app/App';
5  import {Provider as ReduxProvider} from 'react-redux';
6  import store from './app/redux/index';
7  import {Provider as PaperProvider} from 'react-native-paper';
8  import theme from './app/styles/theme';
9  import {I18nextProvider} from 'react-i18next';
10 import i18n from './app/i18n/index';
11
12 function Main() {
13     return (
14         <ReduxProvider store={store}>
15             <PaperProvider theme={theme}>
16                 <I18nextProvider i18n={i18n}>
17                     <App />
18                 </I18nextProvider>
```

[6] https://callstack.github.io/react-native-paper/index.html
[7] http://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf
[8] https://es.reactjs.org/docs/introducing-jsx.html

```
19                </PaperProvider>
20            </ReduxProvider>
21        );
22    }
23
24    AppRegistry.registerComponent(appName, () => Main);
```

Fragmento de código 6.1: Contenido del fichero ./index.js

Una vez hecho esto, todos los componentes que estén embebidos en el componente que representa toda la aplicación *App* podrán acceder a los servicios correspondientes a través de su contexto haciendo uso de las funciones que provee cada servicio. Un componente que tiene acceso tanto al estado, como a las traducciones como al tema se exportará según se puede ver en el fragmento de código 6.2.

```
1    export default connect(
2        mapStateToProps,
3        mapDispatchToProps,
4    )(withTranslation()(withTheme(Component)));
```

Fragmento de código 6.2: Plantilla de exportación de un componente que recibe como propiedades el estado, las traducciones y el tema

donde *connect()* le da acceso al estado global, *withTranslation()* le da acceso a las traducciones y *withTheme()* le da acceso al tema. Las dos variables *mapStateToProps* y *mapDispatchToProps* son las que indican a la función *connect()* a qué datos y funciones del estado, respectivamente, quiere acceder el componente.

Otro aspecto a comentar es que cuando se empezó la aplicación, las pantallas de creación y modificación de pagos y recordatorios de pago estaban implementadas en cuatro ficheros distintos, uno para cada pantalla. Al ver que las pantallas de modificación y creación de un documento eran visualmente iguales y que sólo cambiaba la lógica que se aplicaba, se decidió juntarlas, resultando en una reducción de código importante, además de ayudar a mejorar el principio de no repetir código. La reducción de código fue del 35 % y 40 % como se indica en la figura 6.2.

6.4 Traducción

Para tener traducciones de todos los textos a varios idiomas lo que se hace es asignar a cada texto a traducir un nombre. Después, en ficheros distintos, se tiene la traducción de ese texto en un idioma determinado, cada fichero en su idioma. Finalmente, se elige el idioma de la aplicación, y en función de esto, se cogerá la traducción de un texto en un idioma u otro. Esta es la tarea del paquete *react-i18next*. Además, este paquete nos permite formatear un texto si se le pasa una función de formateo en la inicialización. Gracias a ello podemos formatear fechas o divisas.

Flatlife está en el idioma del dispositivo que la ejecuta, por lo que en la inicialización de *react-i18next* se comprueba cuál de los idiomas en los que está traducida la aplicación se ajusta mejor al idioma del dispositivo y ese será con el que se ejecute.

Para que se entienda mejor vamos a ver una porción de código de la aplicación. Primero de todo se muestra parte de las traducciones tanto en español como en inglés en los fragmentos de código 6.3 y 6.4.

```
1  const spanish = {
2      appName: 'Flatlife',
3      shared: {
4          currency: '{{amount, $}}',
```

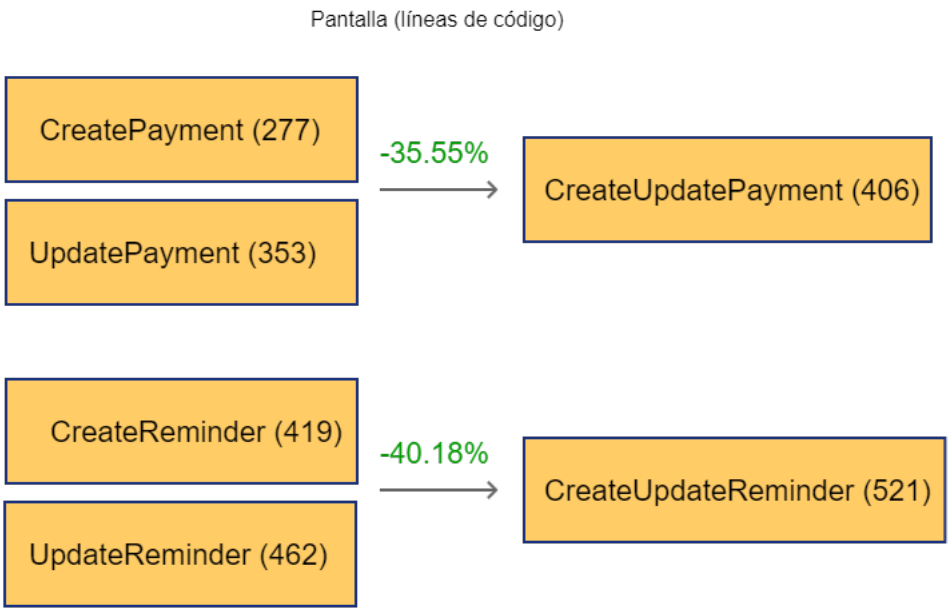


Figura 6.2: Reducción de código al juntar dos pantallas distintas en una sola.

```
        date: '{{date, date}}',
        apply: 'Aplicar',
        applyFilters: 'Aplicar filtros',
        name: 'Nombre',
        email: 'Correo',
        password: 'Contraseña',
        confirmPassword: 'Confirmar contraseña',
        ...
    },
    ...
};

export default spanish;
```

Fragmento de código 6.3: Contenido del fichero ./app/i18n/spanish.js

```
const english = {
    appName: 'Flatlife',
    shared: {
        currency: '{{amount, $}}',
        date: '{{date, date}}',
        apply: 'Apply',
        applyFilters: 'Apply filters',
        name: 'Name',
        email: 'Email',
        password: 'Password',
        confirmPassword: 'Confirm password',
        ...
    },
    ...
};

export default english;
```

Fragmento de código 6.4: Contenido del fichero ./app/i18n/english.js

Como puede observarse, se le ha asignado el nombre *appName* al texto "Flatlife" y el nombre *shared.apply* a los textos "Aplicar" y "Apply". De este modo, cuando en la aplicación queremos que aparezca el texto "Aplicar" en el idioma del dispositivo, llamaremos a la función de traducción *t* proporcionada por el proveedor pasando como argumento el nombre del texto que queremos.

```
<Button
    color={theme.colors.accent}
    style={sharedStyles.button}
    onPress={this.onApply}>
    {t('shared.apply')}
</Button>
```

Fragmento de código 6.5: Estructura de un botón cuyo texto es la traducción de "shared.apply"

El fragmento de código 6.5 forma parte de un componente. En él estamos indicando que se va a visualizar un botón (que es un componente) con el color secundario de la aplicación (*accent*), con un estilo determinado, y que llamará a la función *onApply* del componente *this* (es una función que pertenece al componente que contiene al botón) siempre que el botón sea presionado. Además, se puede ver cómo todas las expresiones en JavaScript están envueltas por corchetes, siguiendo la sintaxis de JSX. La última línea que falta por explicar, *t('shared.apply')*, está llamando a la función de traducción *t* pasando el nombre del texto que se mostrará en el botón. Es esta función la encargada de devolver el texto "Aplicar" o "Apply" según el idioma elegido.

6.5 Navegación

La navegación es la habilidad que tiene la aplicación de cambiar entre diferentes pantallas según la interacción que tenga el usuario. El paquete que usa Flatlife para implementar la navegación es *react-navigation*. En *react-navigation* la navegación se implementa mediante componentes llamados navegadores. A estos navegadores se les pasa las pantallas que van a mostrar y el nombre con el que se identifican para poder cambiar entre pantallas por código. Existen varios navegadores ya hechos por los creadores del paquete, pero se pueden configurar e incluso crear algunos personalizados.

Los navegadores que se utilizan en Flatlife son todos definidos por *react-navigation* adaptados a las necesidades de la aplicación.

- **StackNavigator**. El navegador con pila permite hacer transiciones entre pantallas donde cada nueva pantalla se añade a la cima de la pila. Este navegador permite una navegación a través de una barra superior de navegación.
- **DrawerNavigator**. Este navegador nos permite abrir y cerrar un menú lateral personalizado.
- **MaterialBottomTabNavigator**. Permite tener una barra de navegación inferior. Este navegador depende de un componente de la librería *react-native-paper* de la cual ya depende Flatlife. Esto nos ayuda en la meta de conseguir un diseño que siga la guía de *Material Designs*.

Cada uno de estos navegadores se añade como dependencia individualmente. Es posible también anidar los navegadores si es necesario. La estructura de navegadores que se utiliza Flatlife se muestra en la figura 6.3. En esta figura no se representan todos los navegadores de la aplicación, pero sí su estructura de anidación. Primero de todo existe un *DrawerNavigator* que contiene a todos los demás navegadores. Los hijos de éste pueden ser *StackNavigator*, en caso de que las pantallas de este navegador no necesiten una barra de navegación inferior o *TabNavigator* en caso de que sí que la necesiten. Además, dentro de un *TabNavigator* pueden haber varios *StackNavigators* con su propia navegación.

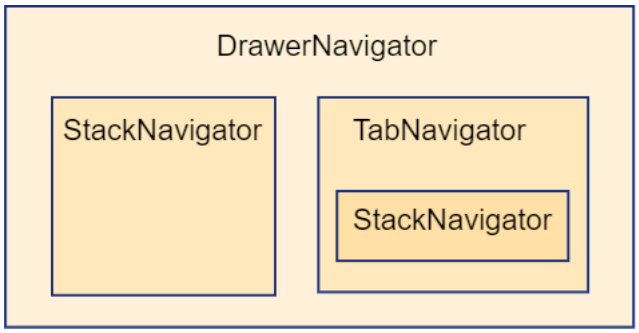


Figura 6.3: Estructura de navegadores utilizada en Flatlife

6.6 Estado

El estado de la aplicación es la parte más costosa y extensa. Nos ayudaremos de Redux para conseguir el objetivo de mantener un estado global sólo de lectura en el que los cambios se realizan únicamente mediante funciones pura llamadas *reducers*. Cuando se quiere modificar el estado, un componente emite una acción (objeto que contiene el tipo de acción y los datos necesarios para ser manejada) que será procesada por el *reducer* asignado a dicho tipo de acción. Estos *reducer*, a partir de ahora llamados reductores en este documento, pueden ser asíncronos y realizar peticiones a APIs externas de manera no bloqueante. De hecho, en Flatlife son los encargados de realizar las peticiones a la base de datos para adquirir toda la información con la que trabaja.

Cuando sólo se llevaba parte de la funcionalidad de Redux relacionada con los usuarios y con los grupos, se decidió hacer uso de RTK[9] (Redux Toolkit), una herramienta que ayuda a eliminar código repetitivo usado para declarar las acciones y para determinar qué reductor se encarga de gestionar la petición (un swith muy grande) entre otras cosas. La reducción de código relacionado con el estado fue de más del 50 % como se ilustra en la figura 6.4.

Como vemos, al principio se separaron las acciones y los reductores en ficheros distintos. En las acciones se definían los tipos de las acciones como constantes y se implementaban los creadores de acciones, que son funciones que devuelven acciones, con su tipo y los datos de los que precisan para que el reductor pueda manejarla. Después, en el fichero de los reductores se implementaba una función que, dado el estado y una acción, ejecutaba el reductor encargado de procesar la acción. Esto se hacía mediante un switch con muchos casos. Al final, con la introducción de RTK, los dos ficheros mencionados pasaron a ser uno sólo, en el que se definían las acciones y los reductores evitando la repetición de código.

[9] https://redux-starter-kit.js.org/introduction/quick-start

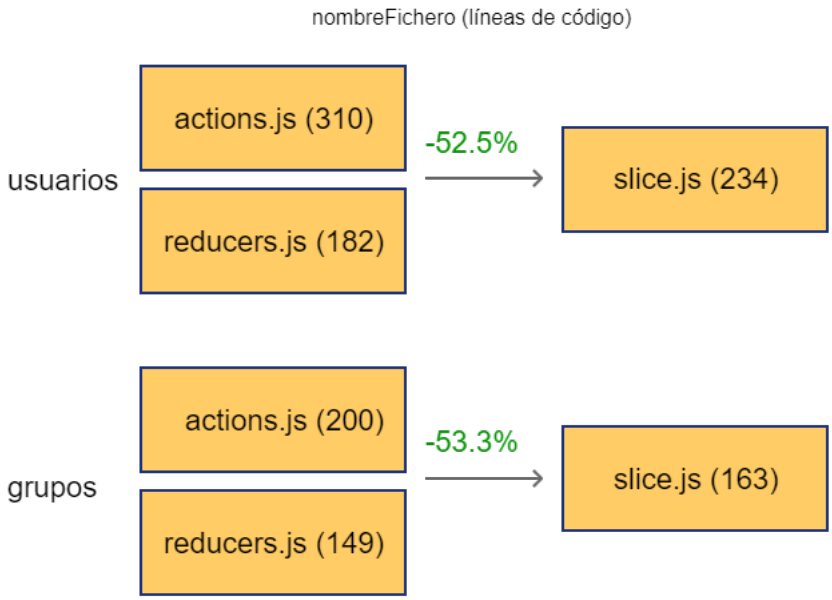


Figura 6.4: Reducción de código al utilizar Redux Toolkit

6.7 Servicios

Los servicios en Flatlife son ficheros JavaScript que contienen funciones necesarias para implementar las funcionalidades de la aplicación.

En concreto, los servicios disponibles en la aplicación son:

- **Base de datos**. El servicio de base de datos proporciona una API para la creación de peticiones a la base de datos de Firebase Firestore. La API permite las operaciones de creación, obtención y eliminación. Además, permite la suscripción a determinadas colecciones de datos para que cualquier cambio que se produzca en la base de datos sea notificado a la aplicación que se haya suscrito, es decir, para tener datos en tiempo real.
- **Autenticación**. El servicio de autenticación contiene las funciones necesarias para el registro de un usuario, el envío de un mensaje de confirmación al correo electrónico, el cambio de contraseña mediante un mensaje de correo electrónico, la actualización de la dirección de correo y contraseña de un usuario, la autenticación mediante dirección de correo y contraseña y autenticación con la cuenta de Google.
- **Fechas**. Este servicio contiene funciones útiles que trabajan con la librería ya mencionada *moment.js* para manejar fechas. Tiene funciones para formatear, para calcular la diferencia en milisegundos o días entre dos fechas o para añadir cierta cantidad (2 días por ejemplo) de tiempo a una fecha determinada.
- **Sistema de ficheros**. Servicio encargado del manejo de ficheros locales de la aplicación, actualmente las imágenes, ya sean de los usuarios, de los grupos o de las compras. Este servicio permite crear directorios, cambiar de nombre ficheros, moverlos entre directorios, etc.

- **Validación de datos**. Haciendo uso del paquete *validatejs*, este servicio comprueba que los textos introducidos por un usuario siguen el formato correcto en direcciones de correo, contraseñas, divisas o que no haya campos en blanco. El formato de los textos se definen mediante expresiones regulares. Algunos de estos formatos ya están predefinidos en el paquete.
- **Almacenamiento**. El servicio de almacenamiento se comunica con Firebase Storage para almacenar las imágenes con las que trabaja Flatlife. Esta API permite guardar y descargar imágenes de la nube.
- **Misceláneo**. Este servicio no está especializado en ninguna funcionalidad sino que aporta funciones compartidas entre distintos ficheros. Algunas funciones incluidas sirven para la de creación de identificadores para los objetos que se almacenan en la base de datos, para reemplazar las comas por puntos en divisas para después convertirlas en números o la conversión de euros a céntimos y viceversa, importante para guardar las deudas si no queremos problemas de precisión con los números.

6.8 Datos en tiempo real

Una de las funcionalidades que proporciona la API Firebase Firestore es la capacidad de obtener los cambios que se realizan en los datos en cuanto éstos se dan, por lo que la aplicación tiene acceso a los datos en tiempo real. Esta funcionalidad ha sido empleada sobre toda la aplicación para dotar de dinamismo la interacción del usuario con la misma.

Para que la aplicación no esté recibiendo los cambios de todos los datos que maneja, solamente se va suscribiendo a aquellos que va necesitando. Así pues, cuando se inicia la aplicación, sólo se suscribe a la información del grupo y a los usuarios que se encuentran en él. Cuando accedes por ejemplo a la pantalla de los recursos compartidos, Flatlife se suscribe tanto a los recursos compartidos como a las reservas de éstos. La suscripción a los datos cuando se necesitan ayuda a ahorrar ancho de banda en el dispositivo que utilice la aplicación.

6.9 Capacidades sin conexión

Otro de los requisitos importantes que se quería conseguir en Flatlife era la capacidad de trabajar sin conexión a Internet. Para conseguir esto, la API de Firebase Firestore también tiene solución. Cuando se realiza una suscripción sin conexión, Firestore busca en su cache cúales son los documentos que se devolvieron por última vez y los facilita. La primera vez se ejecuta la aplicación sí que se necesita conexión a Internet para recoger por primera vez todos los documentos. Por otro lado, cuando se efectúan peticiones de creación, modificación o eliminación, Firebase guarda en una cola todas las peticiones para ejecutarlas cuando se reestablece la conexión. Para que el usuario no tenga que esperar a que la petición se ejecute para ver los cambios que ha realizado, se modificarán los datos del estado conforme a la petición.

Otro aspecto necesario para poder trabajar sin conexión es la generación de identificadores (IDs) para los nuevos documentos a crear. Aunque la API de Firestore permite la generación automática de IDs, cuando se trabaja sin conexión, se necesitan dichos códigos antes de que se ejecute la petición en la base de datos. Para ello, se ha creado una función generadora de códigos de identificación, códigos de 20 caracteres que pueden contener letras mayúsculas, letras minúsculas y números, tal y como hace Firestore.

Como la generación de IDs es aleatoria, es posible que cuando se vaya a crear un documento en la base de datos, el identificador generado coincida con el de otro documento dentro de la misma colección, en cuyo caso se sobreescribirán los datos y no se creará un nuevo documento. No obstante, la probabilidad de que un ID se repita es muy baja, teniendo en cuenta que los códigos son de 20 caracteres y en cada posición puede ir uno entre 63 posibles valores:

$$P(XX) = \frac{1}{63^{20}} x \frac{1}{63^{20}} \approx 10^{-72} \tag{6.1}$$

donde X es el suceso de que se genere un determinado ID. Si además sumamos que los dos documentos deben ser de la misma colección, la probabilidad disminuye aún más, por lo que se puede considerar que esto no ocurrirá nunca.

Para los interesados en la implementación de la función generadora de IDs, se muestra en el fragmento de código 6.6.

```
    const generateId = () => {
        const CHARS =
        'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';

        let autoId = '';

        for (let i = 0; i < 20; i++) {
            autoId += CHARS.charAt(Math.floor(Math.random() * CHARS.length));
        }

    return autoId;
};
```

Fragmento de código 6.6: Función generadora de IDs para los documentos que se van a guardar en Firebase Firestore

CAPÍTULO 7
Resultados

En este capítulo de Resultados se muestra cuál es el aspecto final de la aplicación mediante una historia ficticia de varias personas que van a convivir juntas y van a utilizar Flatlife. Además, en la sección 7.1 se exponen cuáles son las métricas utilizadas para comprobar la calidad del código.

María, Isabel y Antonio son tres jóvenes estudiantes que han coincidido en una vivienda compartida sin conocerse anteriormente. Llevan un mes en su nuevo hogar temporal y han realizado varias tareas comunes. Han comprado muchas cosas y han limpiado algunas zonas de la casa. En un momento dado, María se da cuenta de que debe recibir dinero de sus compañeros pero no recuerda cuánto puesto que no ha guardado los tiques de la compra y nunca ha apuntado el valor de las compras que realizaba. Además, ya no saben a quién le toca limpiar cada zona de la casa ni cuándo se han realizado limpiezas. Es por ello que los tres han decidido utilizar Flatlife. La primera vez que abren la aplicación, María y Antonio se crean una cuenta mediante sus direcciones de correo y contraseñas (figura 7.1a).

Una vez se han creado la cuenta, utilizan las credenciales para autenticarse (figura 7.1b) En cambio, Isabel ha utilizado su cuenta de Google para el mismo fin.

(a) Pantalla *Signup*.

(b) Pantalla *Signin*. Puedes autenticarte mediante una dirección de correo electrónico y una contraseña o mediante tu cuenta de Google, si presionas el icono de Google.

Figura 7.1: Pantallas de registro y autenticación.

Una vez autenticados, dejan que Isabel cree un nuevo grupo (figura 7.2).

Una vez ya se encuentra dentro del grupo (figura 7.3a), Isabel invita a María y a Antonio (figura 7.3b). Ya están listos para comenzar a disfrutar de una organización dentro del piso, salvo porque aún no han configurado su perfil. Entonces, los tres acceden a sus respectivos perfiles a cambiarse la foto de perfil (figura 7.3c). Ahora ya sí que sí, los tres comienzan un viaje en busca de la mejor organización y compañerismo (figura 7.3d).

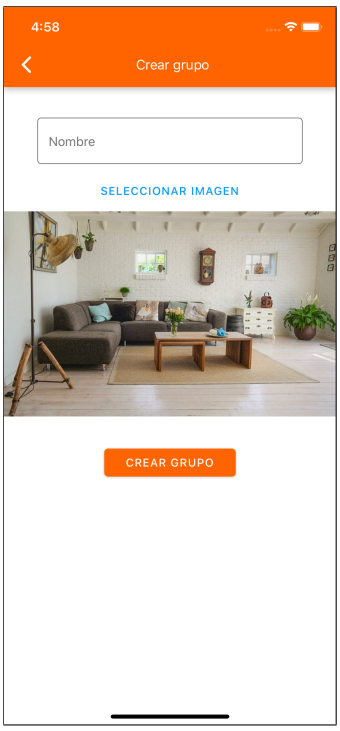
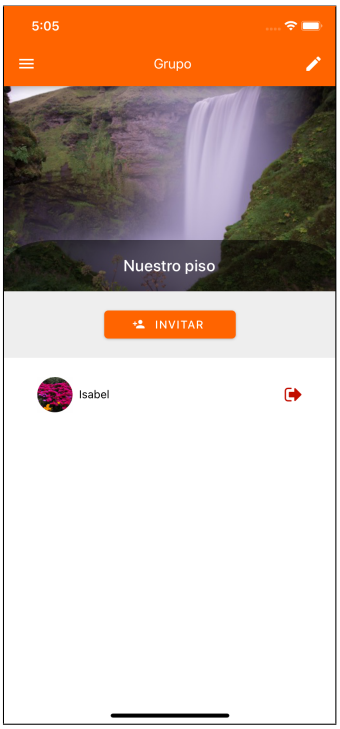
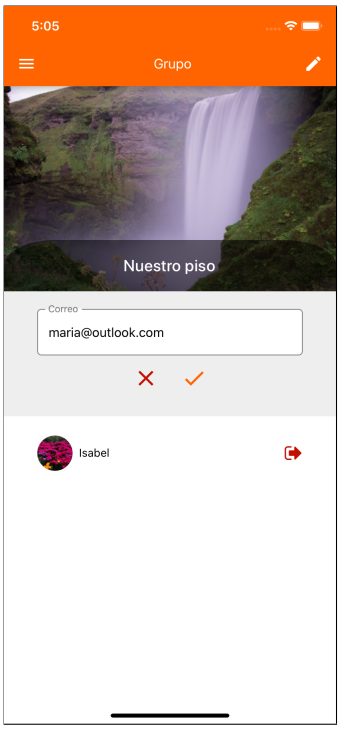


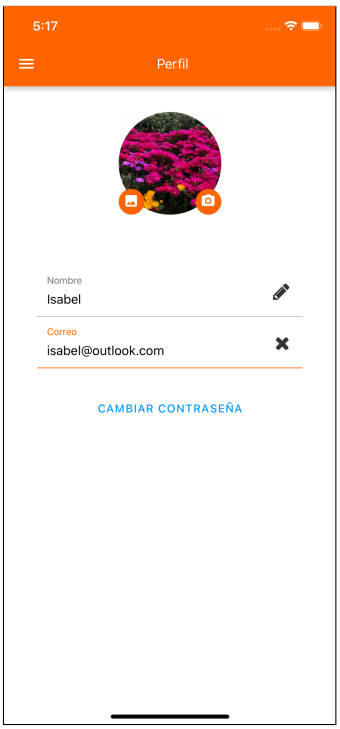
Figura 7.2: Pantalla *CreateUpdateGroup*.



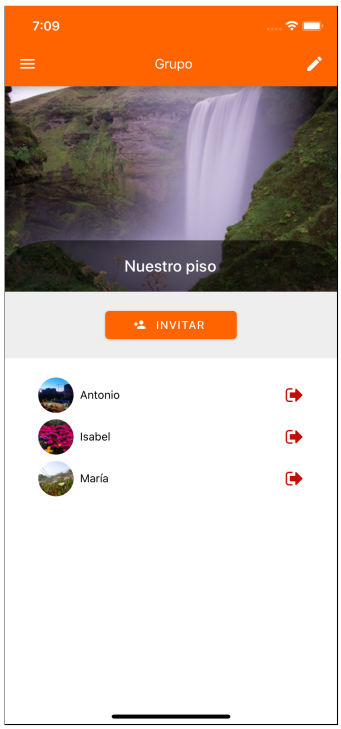
(a) Pantalla *Home*.



(b) Pantalla *Home* después de haber presionado el botón "Invitar".



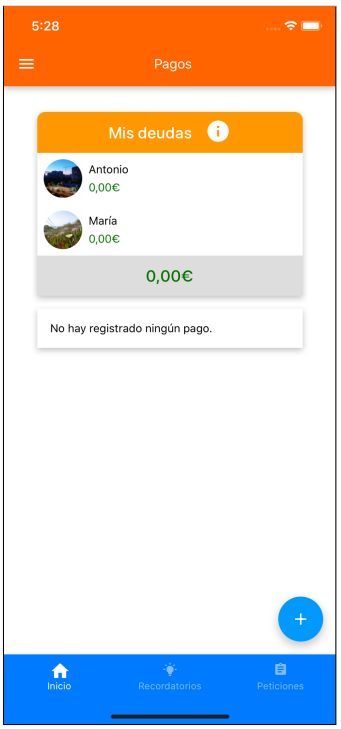
(c) Pantalla *Profile*.



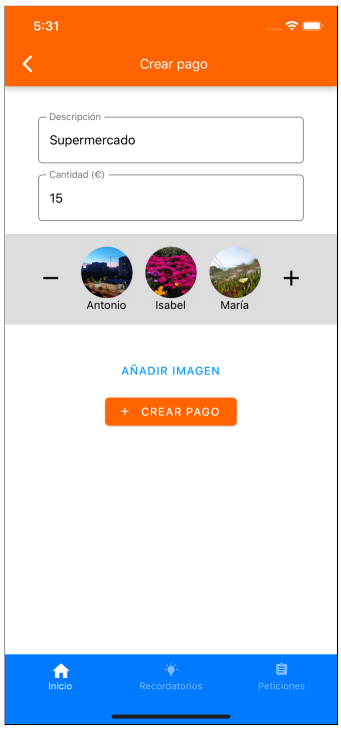
(d) Pantalla *Home* con todos los integrantes del grupo.

Figura 7.3: Pantallas de grupo y de perfil.

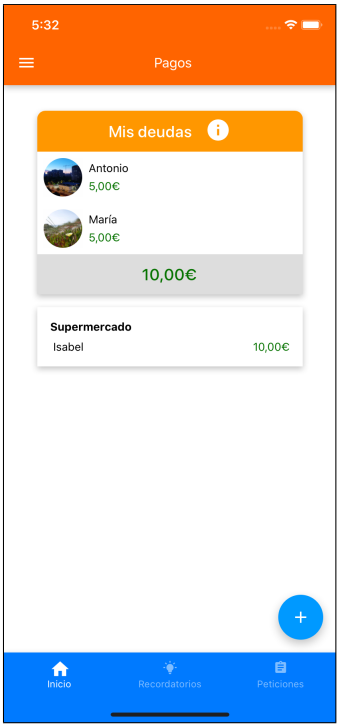
Por el momento no existe ninguna deuda entre los tres compañeros (figura 7.4a), ya que ninguno de ellos ha realizado un pago compartido. No obstante, el próximo día que Isabel va a comprar al supermercado registra en el grupo el pago de 15 euros (figura 7.4b), creando una deuda con Antonio y María, es decir, le deben 5 euros cada uno (figura 7.4c), ya que las compras comunes son equitativas. Nada más registrar el pago, Isabel se da cuenta de que ha escrito mal la cantidad de euros, ha incluido en el precio un kilo de naranjas que son únicamente para ella. Entonces, rápidamente va a los detalles del pago (figura 7.4d) y modifica el precio (figura 7.4e).



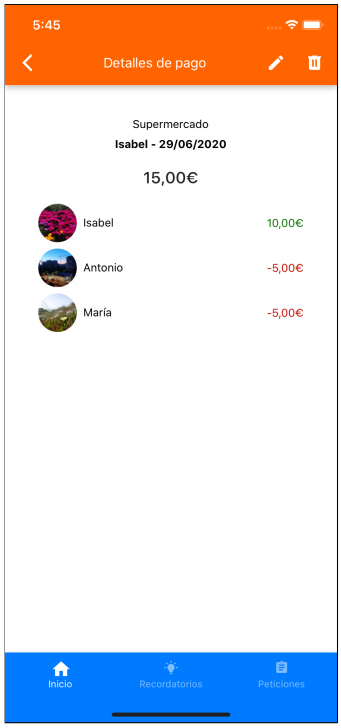
(a) Pantalla *PaymentsHome*.



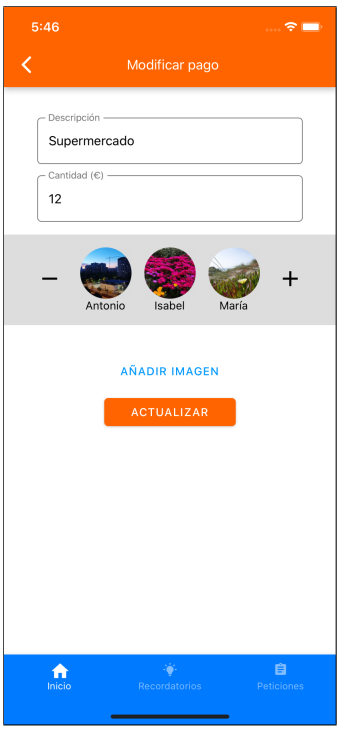
(b) Pantalla *CreateUpdatePayment*.



(c) Pantalla *PaymentsHome*.



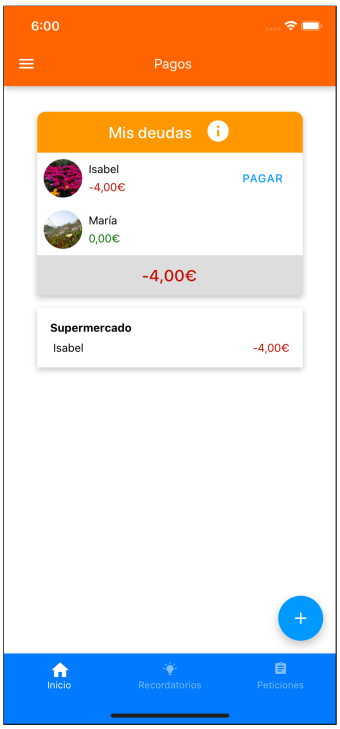
(d) Pantalla *PaymentDetails*.



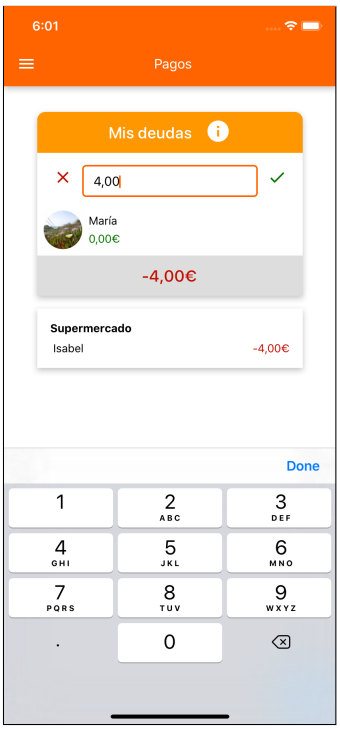
(e) Pantalla *CreateUpdatePayment*.

Figura 7.4: Pantallas de inicio, detalles, creación y modificación de pagos.

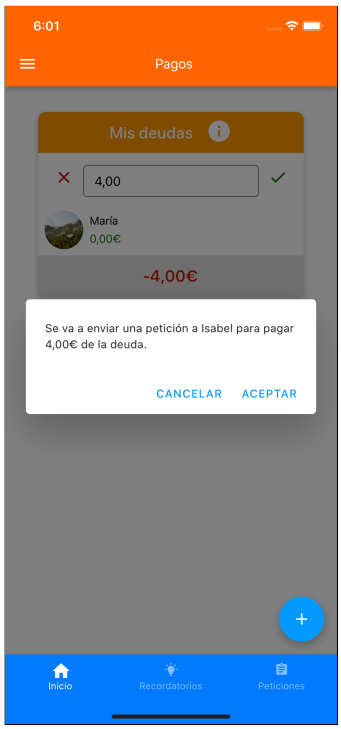
Antonio enseguida se da cuenta de que le debe dinero a Isabel (figura 7.5a) y mira los detalles del pago. Como sabe que esa compra sí que la hizo Isabel, enseguida se los devuelve en la vida real y realiza una petición en la aplicación para que la deuda pase a ser 0 (figuras 7.5b y 7.5c). Además, Antonio registra un pago de un gel y un champú que había hecho por la mañana.



(a) Pantalla *PaymentsHome*.



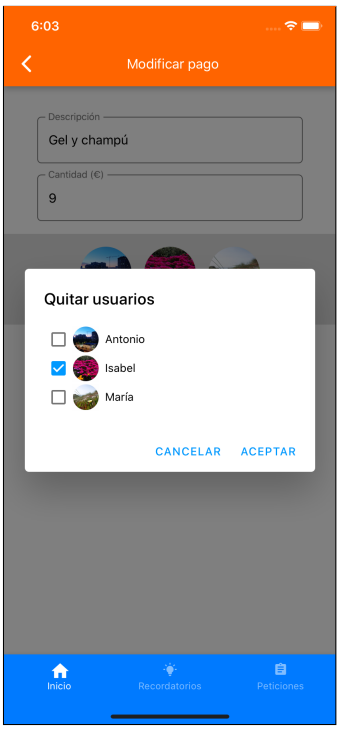
(b) Pantalla *PaymentsHome* tras presionar el botón "Pagar"



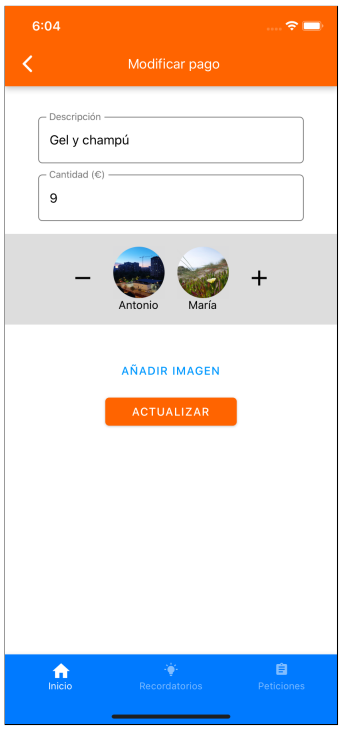
(c) Pantalla *PaymentsHome* tras presionar el *tick* verde que se encuentra a la derecha de la cantidad a pagar

Figura 7.5: Pantalla de inicio de pagos. Las tres figuras muestran los pasos que se siguen para realizar una petición de pago de deuda.

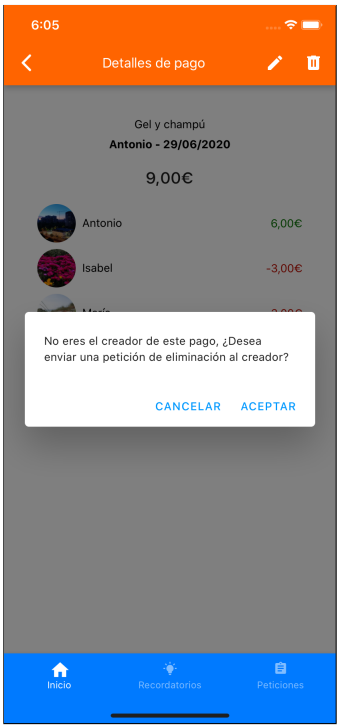
Cuando Isabel ve el pago que acaba de registrar Antonio, se da cuenta de que ella ahora va a empezar a utilizar un champú especial para su tipo de pelo así que le envía una petición a Antonio para que modifique el pago y que ella ya no aparezca en él (figuras 7.6a y 7.6b). Una vez ha realizado la petición, se acuerda de que su compañera María iba a compartir champú con ella, por lo que tampoco querrá pagar el champú que ha comprado Antonio y que no van a utilizar. Entonces, Isabel vuelve a los detalles del pago y hace una petición a Antonio para que elimine el pago (figura 7.6c). Además, como Antonio ya le ha pagado lo que le debía en la vida real, Isabel confirma la petición de pago de deuda que le había hecho Antonio. Como Antonio le había dado 4 euros, la deuda de Antonio e Isabel disminuirá en 4 euros. Como Antonio todavía no ha eliminado el pago del gel y champú, aparece que Isabel le debe 3 euros a Antonio (figuras 7.6e y 7.6d).



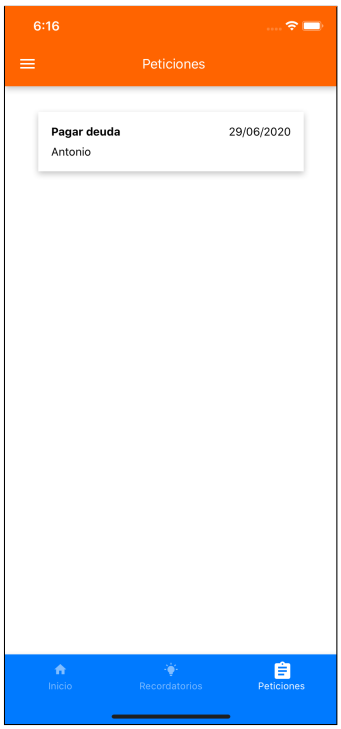
(a) Pantalla *CreateUpdatePayment* tras presionar el icono "-"



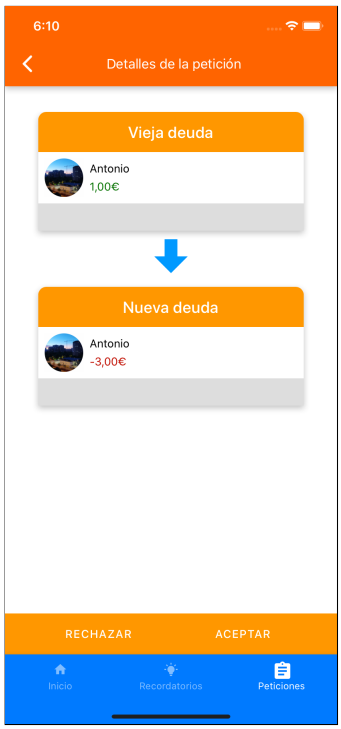
(b) Pantalla *CreateUpdatePayment*.



(c) Pantalla *PaymentDetails* tras presionar el botón "Actualizar" cuando el usuario no es el creador del pago.



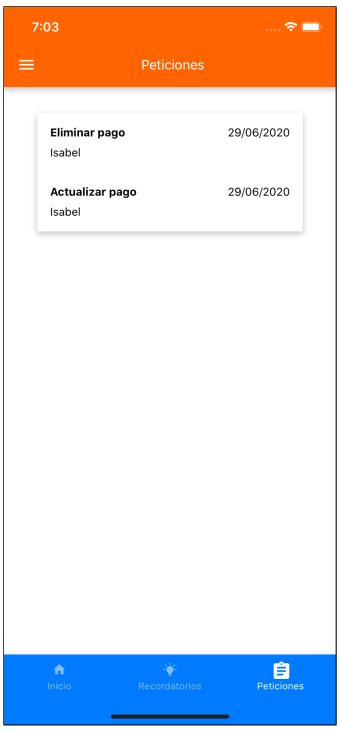
(d) Pantalla *PaymentRequests*.



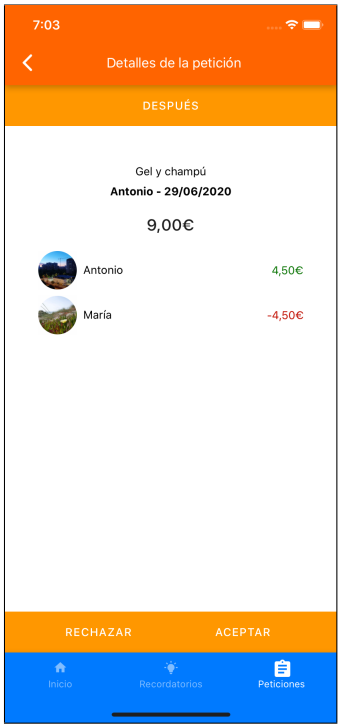
(e) Pantalla *PaymentRequestDetails* de una petición de pago de deuda.

Figura 7.6: Pantallas de modificar pago, peticiones y detalles de una petición.

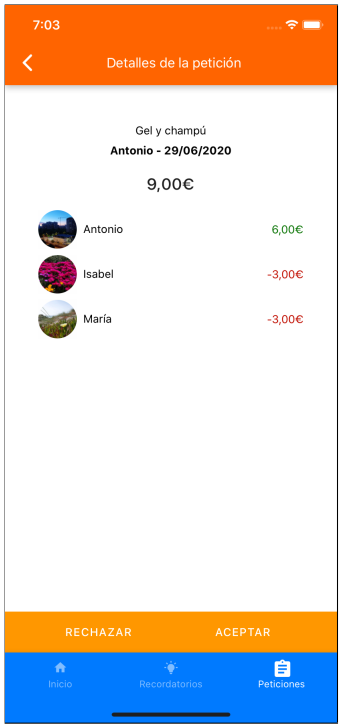
Para acabar con las deudas, Antonio se va al apartado de peticiones y ve que tiene 2 (figura 7.7a): una de modificar un pago (figura 7.7b) y otra de eliminarlo (figura 7.7c). Como Antonio ya ha hablado con sus dos compañeras, sabe que tiene que eliminar el pago, entonces acepta la petición y se acaba la deuda con Isabel y María.



(a) Pantalla *PaymentRequests*.



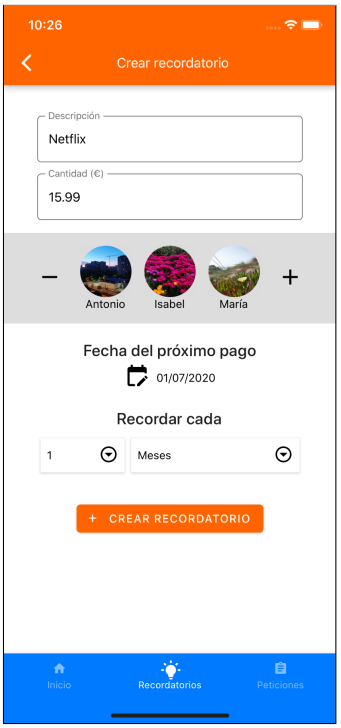
(b) Pantalla *PaymentRequestDetails* de una petición de actualización de un pago.



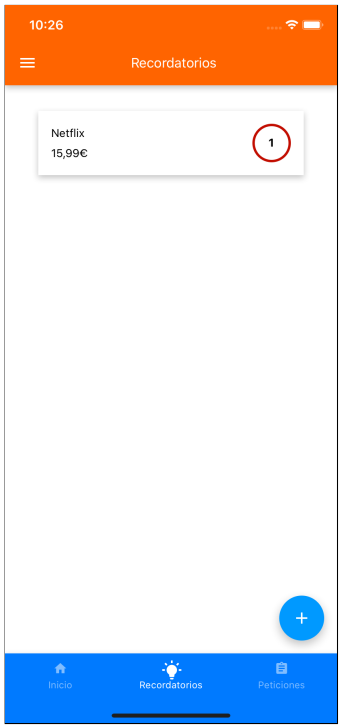
(c) Pantalla *PaymentRequestDetails* de una petición de eliminación de un pago.

Figura 7.7: Pantallas de peticiones y de detalles de una petición.

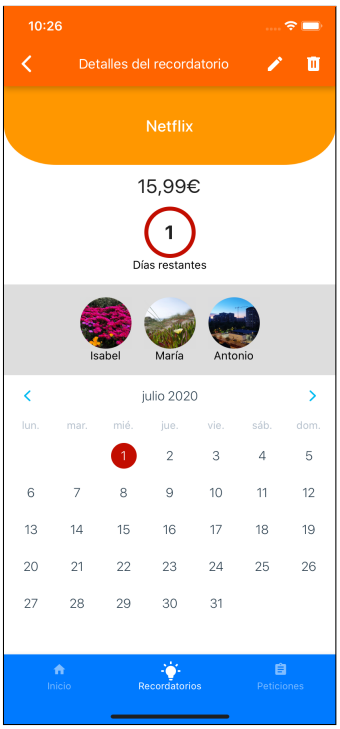
Pasados unos días, los 3 jóvenes deciden suscribirse al servicio que proporciona Netflix, por lo que deben pagar mensualmente una cantidad de dinero. Ese pago se realiza directamente desde la cuenta bancaria de Isabel, pero tanto María como Antonio tienen que pagar a Isabel su parte. Para que no se les olvide, registran el pago mensual en Fatlife (figuras 7.8a y 7.8b). Así pues, ahora siempre sabrán dentro de cuánto se paga una mensualidad del servicio. Además, cuando María y Antonio hayan pagado su parte a Isabel, podrán actualizar la fecha del próximo pago indicando que ya se ha pagado el día que tocaba (figuras 7.8c y 7.8d).



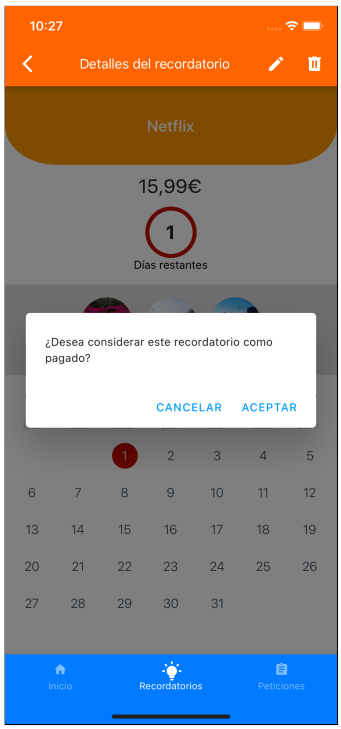
(a) Pantalla *CreateUpdatePaymentReminder*.



(b) Pantalla *PaymentReminders*.



(c) Pantalla *PaymentReminderDetails*.

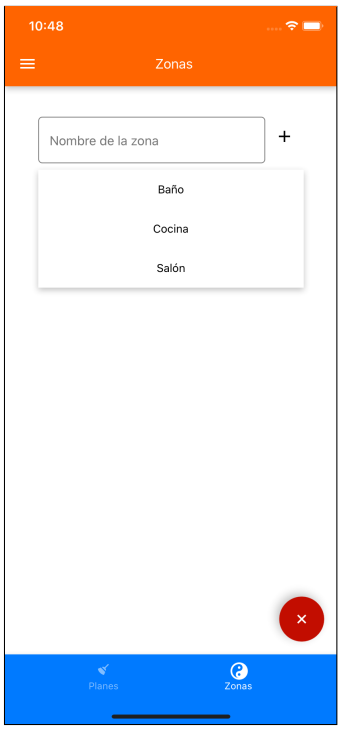


(d) Pantalla *PaymentReminderDetails* tras presionar el día marcado en rojo (día en que se tiene previsto pagar).

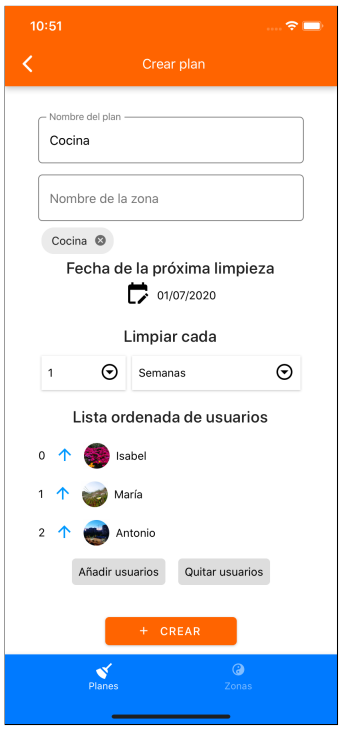
Figura 7.8: Pantallas de creación, lista y detalles de los recordatorios de pago.

Lo mismo hacen para el alquiler del piso y la mensualidad del servicio de Internet.

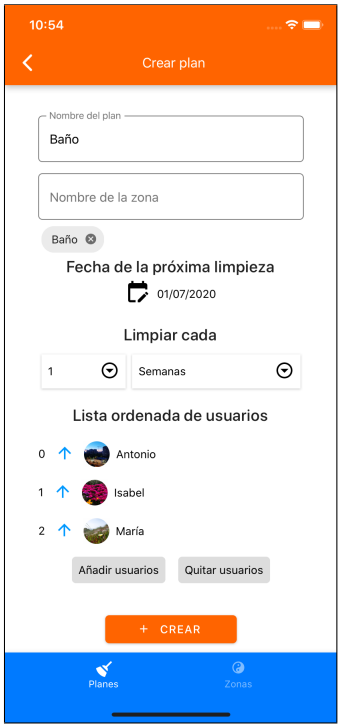
Cuando se vuelven a juntar todos en el piso con tiempo, se reúnen en la mesa del salón para organizar las limpiezas. Saben que existen tres zonas comunes que tienen que limpiar entre los tres, cada semana una persona distinta, el salón, la cocina y el baño. Una vez ya saben eso, crean las tres zonas a limpiar en Flatlife (figura 7.9a). A continuación crean los tres planes de limpieza, cada uno con la zona que hay que limpiar, teniendo en cuenta el orden en que lo van a hacer (figuras 7.9b, 7.9c, 7.9d).



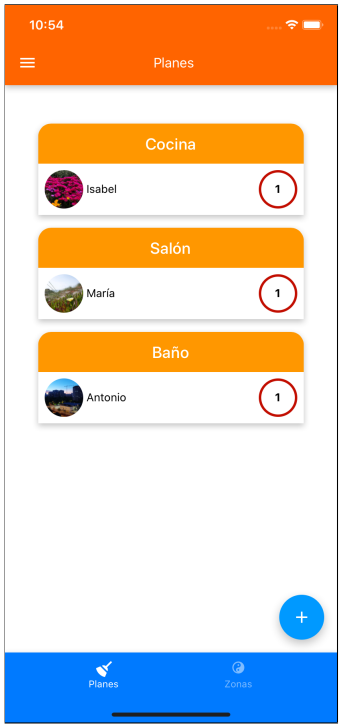
(a) Pantalla *CleaningAreas*.



(b) Pantalla *CreateUpdateCleaningSchedule*.



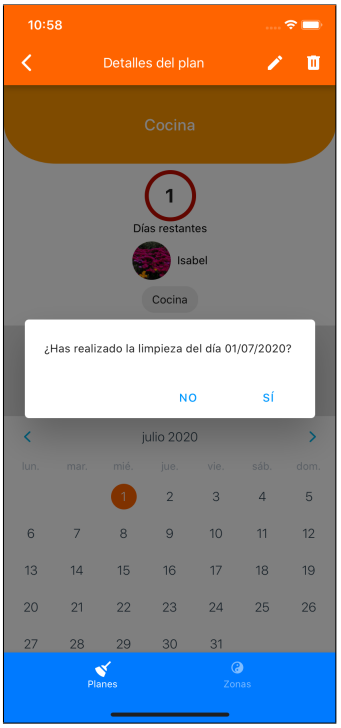
(c) Pantalla *CreateUpdateCleaningSchedule*.



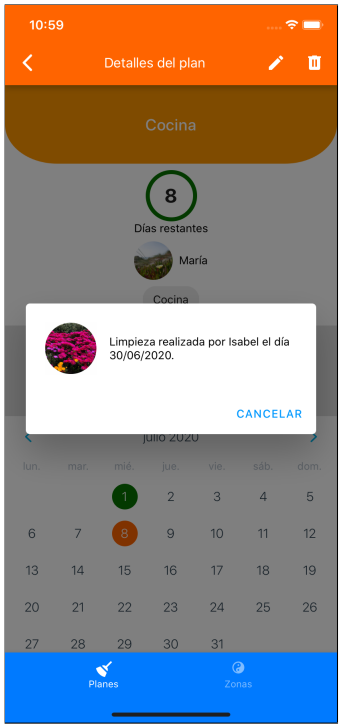
(d) Pantalla *CleaningSchedules*.

Figura 7.9: Pantallas de lista de áreas de limpieza, lista y creación de planes de limpieza.

Una vez realizada la limpieza de la cocina, Isabel vuelve a Flatlife para indicar que esa limpieza ha sido realizada por ella (figura 7.10a). A partir de este momento, queda registrada una limpieza por parte de Isabel en el calendario y no se le olvidará cuándo ha limpiado (figura 7.10b). También ocurrirá lo mismo cuando no se haya limpiado, la persona a la que le toque marcará el día previsto como no limpiado y se quedará registrado.



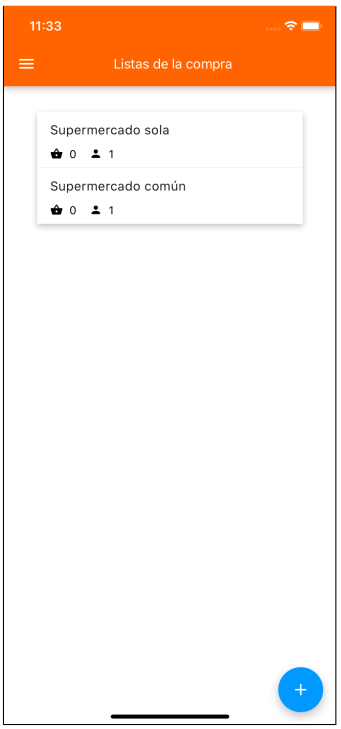
(a) Pantalla *CleaningScheduleDetails* tras presionar el día marcado en naranja (día previsto de limpieza) por la persona a la que le toca limpiar.



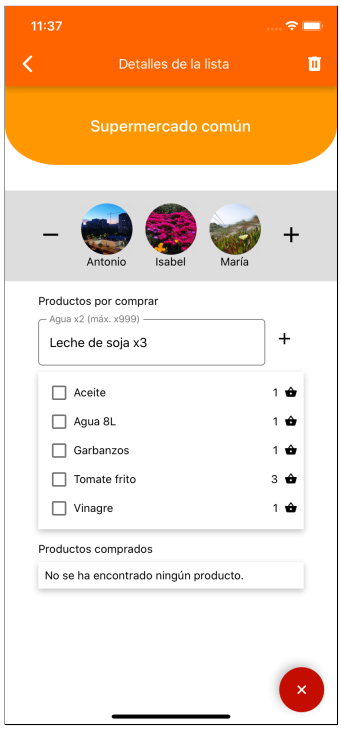
(b) Pantalla *CleaningScheduleDetails* tras presionar el día marcado en verde (indica que se ha realizado la limpieza prevista).

Figura 7.10: Pantalla de detalles de un plan de limpieza cuando se marca una limpieza prevista como realizada.

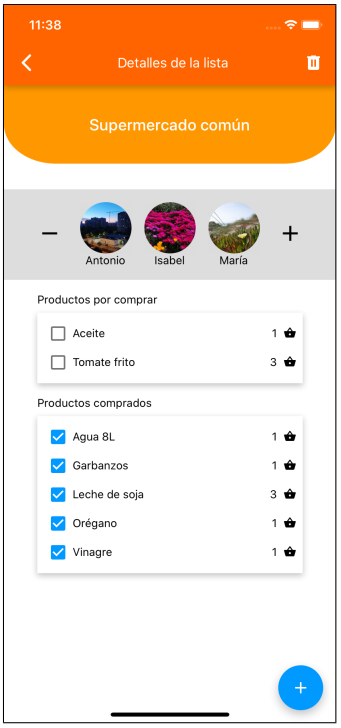
Después de limpiar, Isabel se acuerda de que tiene que ir a comprar al supermercado, entonces aprovecha que Flatlife dispone de una lista de la compra para apuntar lo que necesita. Crea dos listas, una para sus cosas y otra para las cosas comunes (figura 7.11a). Dentro de las listas añade aquellos productos que necesita (figura 7.11b) y avisa a Antonio y a María de que en una hora va a ir a comprar, por si se acuerdan de algún producto común que falte por comprar para que lo apunten. Entonces, Antonio ingresa en la aplicación y ve la lista compartida con él (no ve la que tiene Isabel a solas). Va a ver los detalles de la lista y apunta orégano que falta. Cuando Isabel va a comprar, pasa por las listas de la compra que tiene y mientras mete los productos en el carro, los va tachando en la aplicación (figura 7.11c) hasta que ya no queda ninguno. Mientras tanto, Antonio y María están viendo desde casa los productos que está comprando Isabel de la lista común y siguen pensando si todavía falta algo. María se da cuenta de que es mejor comprar dos garrafas de agua de ocho litros en vez de una, entonces lo modifica antes de que Isabel acabe de comprar (7.11d).



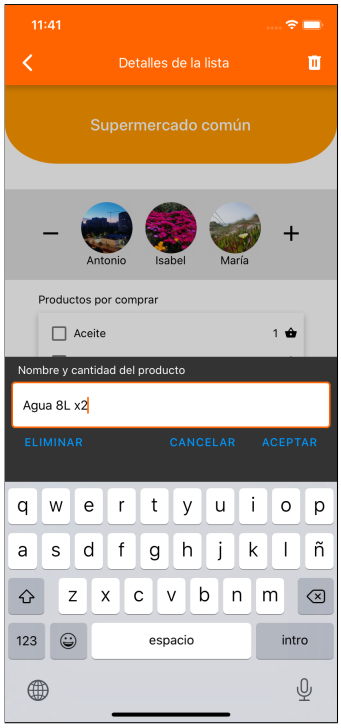
(a) Pantalla *ProductLists*.



(b) Pantalla *ProductListDetails*.



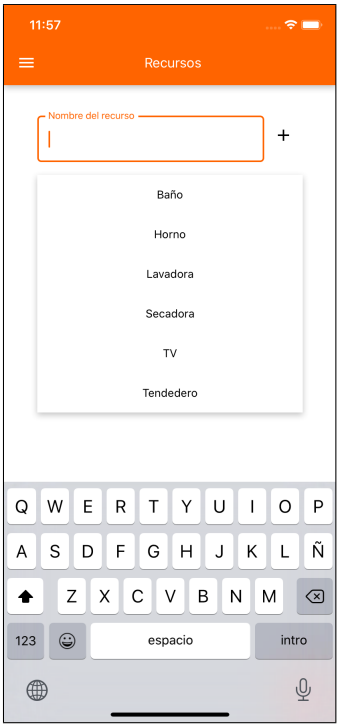
(c) Pantalla *ProductListDetails* tras marcar algunos productos como comprados.



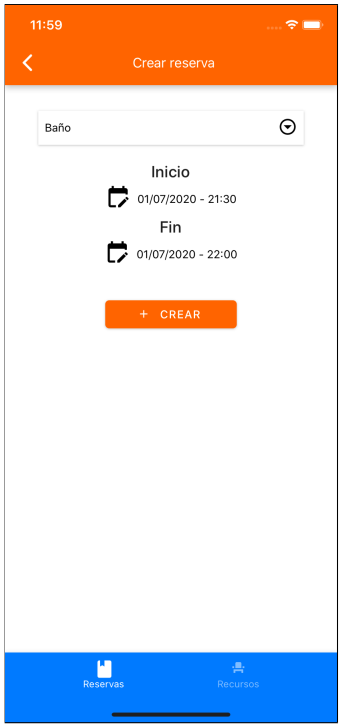
(d) Pantalla *ProductListDetails* tras presionar el nombre de un producto.

Figura 7.11: Pantallas de lista de listas de productos y de detalles de una lista particular.

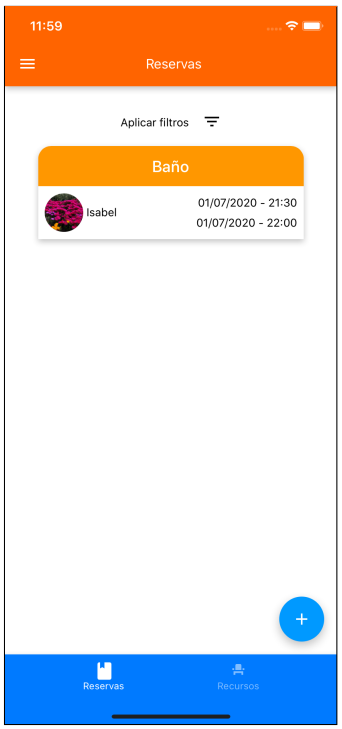
Ese mismo día, Isabel se va a hacer ejercicio a la calle y, sabiendo que volverá sobre las 21:30 a casa, ingresa en Flatlife para reservar el baño media hora para ducharse, ya que no quiere estar esperando mucho tiempo al llegar a casa. Como todavía no han registrado ningún recurso compartido, Isabel crea varios para tenerlos ya disponibles para reservar (figura 7.12a). Acto seguido, crea una reserva (figuras 7.12b y 7.12c). Por supuesto esto no asegura que vaya a tener disponible el baño ya que es posible que alguno de sus compañeros lo necesite con urgencia y lo utilice. No obstante, siempre que se vayan a utilizar recursos durante un largo tiempo, los compañeros deben mirar si están o no reservados. En caso de estarlo, siempre pueden hablar con la persona que lo ha reservado para ver si le deja ocupar parte del tiempo de su reserva. Una vez Isabel llega a casa y se ducha, accede a las reservas y elimina la que tenía del baño (figura 7.12d). Al cabo de unos días, cuando la cantidad de reservas crece (figura 7.12e), buscar si algún recurso está reservado se hace más complicado, por ello Flatlife tiene una funcionalidad de filtrado y ordenación que permite, por ejemplo, mostrar únicamente las reservas de la lavadora en orden ascendiente de fecha de inicio de la reserva. Es así como lo hace Antonio para buscar si alguien va a poner la lavadora al día siguiente (figura 7.12f).



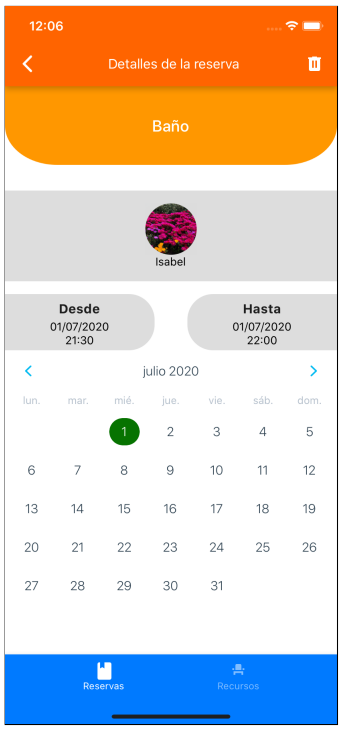
(a) *Pantalla Resources.*



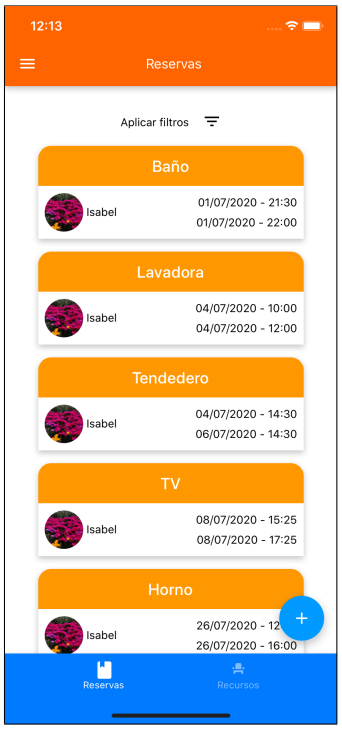
(b) Pantalla *CreateReservation.*



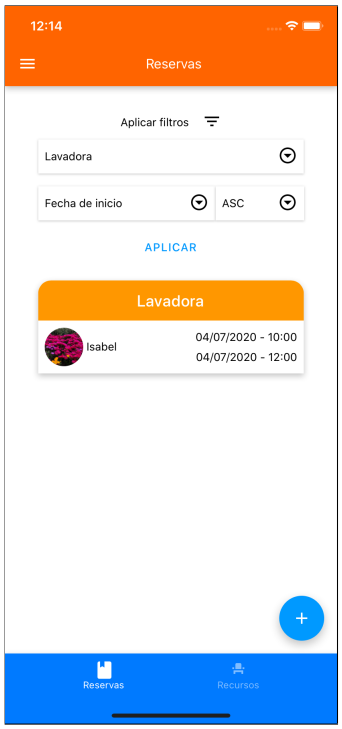
(c) Pantalla *Reservations*.



(d) Pantalla *ReservationDetails*.



(e) Pantalla *Reservations*.

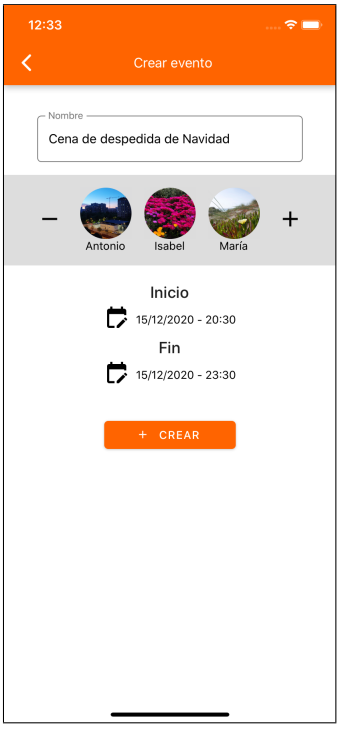


(f) Pantalla *Reservations*.

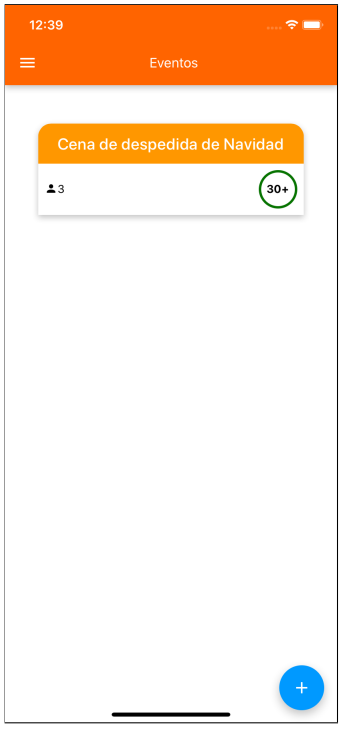
Figura 7.12: Pantallas lista de recursos compartidos y de lista, creación y detalles de reservas.

Se acerca Navidad y los tres compañeros se van de vacaciones con sus familias. Pero antes de eso han decidido hacer una cena los tres fuera de casa para despedirse. Para que

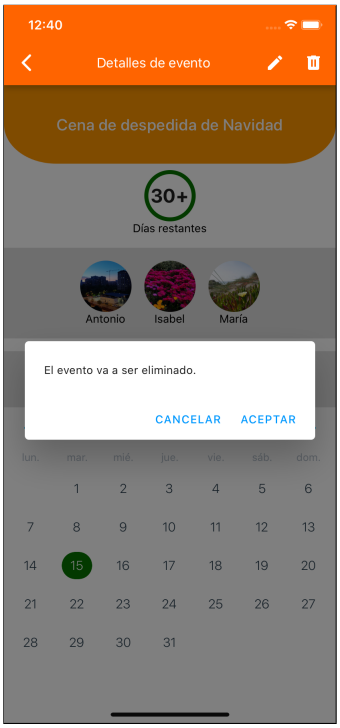
no se les olvide, María lo apunta en Flatlife (figura 7.13a y 7.13b). Además, la aplicación les dirá los días que faltan para que el evento ocurra. Una vez han cenado, María, la creadora del evento, lo elimina (figura 7.13c).



(a) Pantalla *CreateUpdateEvent*.



(b) Pantalla *Events*.



(c) Pantalla *EventDetails*.

Figura 7.13: Pantallas lista, creación, y detalles de eventos.

Con su nueva organización, María, Antonio e Isabel están muy contentos y su relación es excelente, ¡Qué suerte haber utilizado Flatlife!

7.1 Métricas de calidad

En esta sección se realiza un análisis de las métricas de calidad del proyecto que nos proporciona SonarQube[1], que realiza un análisis estático sobre el código del proyecto.

Primero de todo, para tener un visión general de la embergadura de Flatlife y del trabajo realizado se va a mostrar una tabla con las líneas de código escritas recogidas a través del complemento de Visual Studio Code llamado *VS Code Counter*. La información recaudada se muestra en la tabla 7.1.

Tabla 7.1: Líneas de código de los directorios principales de Flatlife junto con las líneas totales del directorio *app*. Las líneas de código que se muestran no incluyen comentarios ni líneas en blanco.

Directorio	Código
components	2717
i18n	456
navigators	578
redux	2881
screens	5615
services	1230
TOTAL	13670

En cuanto a los resultados obtenidos mediante SonnarQube, éstos pueden observarse en la figura 7.14. Para entender mejor qué significa cada métrica, se resumen a continuación.

- ***Bugs***. Problemas que representan algo que está mal en el código y pueden conllevar a que el código deje de funcionar correctamente en cualquier momento. Algunos bugs detectados por SonarQube son: usar NaN en una comparación o usar el operador "delete" sobre algo que no sea una propiedad de un objeto.
- ***Vulnerability***. Problema de seguridad que representa una puerta trasera para los atacantes. Algunas vulnerabilidades detectadas son el uso de instrucciones "alert" o la instrucción "debugger".
- ***Security Hotspot***. Piezas de código sensibles desde el punto de vista de la seguridad que tienes que revisar manualmente. Es posible que el código detectado no tenga ningún tipo de vulnerabilidad.
- ***Code Smells***. Problema de mantenibilidad en el código. Un *code smell* provoca que los que mantienen el código tarden más tiempo en realizar cambios. Algunos detectados por SonnarQube son: el uso de los comparadores "==" y "!=", tener una clausula "default" de un "trycatch" que no se encuentra al final o que la complejidad ciclomática de una función sea muy grande, entre otros.

Ahora bien, como vemos en la figura 7.14, el código de Flatlife revisado por SonnarQube está desprovisto de, vulnerabilidades, fallos de seguridad y *code smells*, por lo que

[1] https://www.sonarqube.org/

la calificación que se obtiene de fiabilidad, seguridad y mantenimiento es máxima (A). No obstante, al no haber realizado testing, el porcentaje de líneas cubiertas por el mismo es del 0 %. En cuanto a la duplicación de código, un 2.24 % se debe a las traducciones, que no pueden eliminarse, mientras que el restante 2.96 % se encuentra en las pantallas.

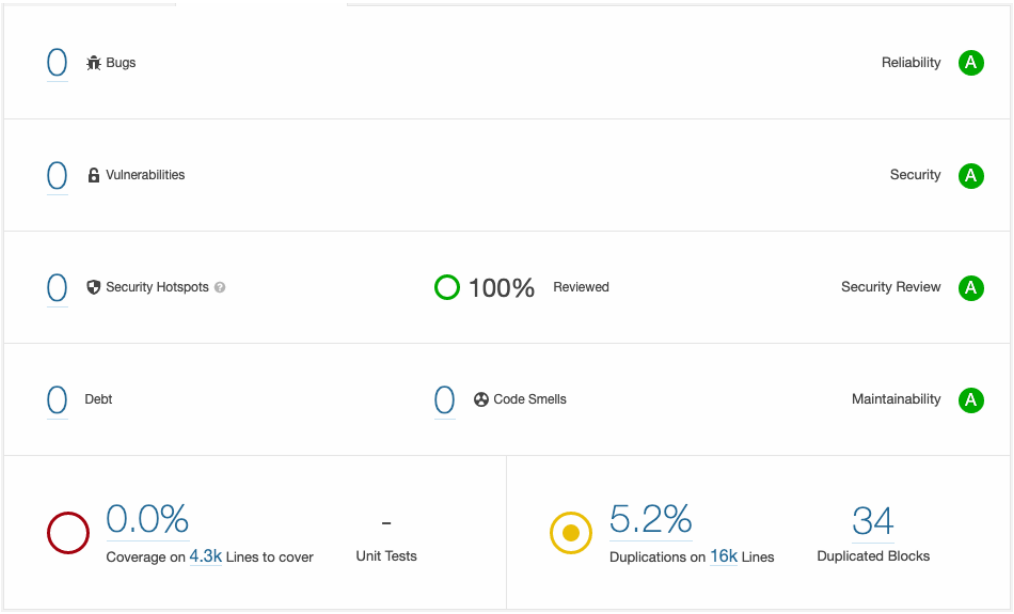


Figura 7.14: Métricas de calidad de Flatlife obtenidas mediante el analizador de código estático SonnarQube.

La última métrica que se va a mostrar, también obtenida mediante SonnarQube, es la complejidad ciclomática y la complejidad cognitiva [12]. La complejidad ciclomática indica el número de caminos que existen en el código, es decir, las veces que el flujo de instrucciones puede dar un salto. Como comentan en la documentación de SonnarQube, para el lenguaje JavaScript "La complejidad ciclomática aumenta en uno cada: función, instrucción if, operador &&, operador ||, expresión ternaria, bucle, clausula *case* dentro de un *switch* y expresiones *throw* y *catch*". En cambio, la complejidad cognitiva es un indicativo de cuán difícil es entender el código realizado.

En la figura 7.15 se observan las complejidades recabadas por SonnarQube. Esta complejidad ciclomática alta indica que han sido necesarias gran cantidad de divisiones en el flujo de control, lo que significa que el código no ha sido sencillo de implementar. No obstante, un nivel de complejidad cognitiva más bajo indica que es más fácil entender el código de lo que realmente es, implicando que la labor de mantenimiento será más sencilla.

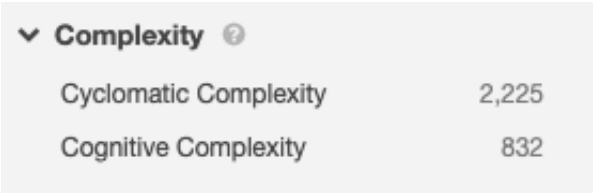


Figura 7.15: Métricas de calidad de Flatlife obtenidas mediante el analizador de código estático SonnarQube.

CAPÍTULO 8
Conclusiones y trabajos futuros

De la realización de este proyecto, la implementación de una aplicación que ayuda a organizar la convivencia de una vivienda compartida ha sido producida, respetando y cumpliendo todos los casos de uso especificados en la sección 3.6.2.

Personalmente, Flatlife no ha sido fácil de implementar puesto que los conocimientos de partida eran pequeños y de herramientas similares, no de React Native. Además, no conocía muchas de las características con las que nos provee el lenguaje de programación usado, JavaScript. Por otra parte, la escritura de este documento tampoco ha sido fácil puesto que mi experiencia no era suficiente tanto a nivel gramatical como estructural y ha requerido tiempo aprender y mejorar en la tarea de escritura. No obstante, gracias al esfuerzo de mi tutor David de Andrés, se ha conseguido una mejor estructura de la memoria y que sea más legible.

Así pues, he aprendido toda una tecnología puntera como React Native, un framework de código libre soportado por más de dos mil personas en GitHub, junto con cierto conocimiento en el desarrollo de aplicaciones móviles y web. También he mejorado la habilidad de buscar información en Internet y de escribir oraciones coherentes y más fáciles de entender.

8.1 Relación con los estudios cursados

Aunque en los estudios del grado no he visto nada relacionado con React Native ni con ninguna tecnología similar, varias asignaturas han ayudado en menor o mayor medida a que el proyecto haya salido adelante. Algunas de ellas se nombran a continuación.

Por supuesto, asignaturas del primer curso como "Introducción a la Informática y la Programación" y "Programación" me dieron los pilares más básicos de conocimiento, usando como apoyo un libro recomendado escrito por los mismo profesores de la universidad llamado *Empezar a programar usando Java* [13]. Por otro lado, "Bases de datos y sistemas de información" me ha enseñado cómo se manejan grandes cantidades de datos y, aunque hayamos trabajado principalmente con bases de datos relacionales, diferentes a la utilizada en Flatlife, saber qué es una base de datos, cómo organizar la información dentro de una y cómo se hacen peticiones ha ayudado a la hora de diseñar la estructura de los datos de Flatlife y a trabajar con ellos.

Varias asignaturas que también han servido de ayuda para conocer las partes de un proyecto y cómo se especifican formalmente sus componentes han sido "Ingeniería del software" y "Gestión de proyectos".

Finalmente, la asignatura optativa que me llevó a pensar en la posibilidad de realizar este trabajo y que hizo que mostrara interés en las aplicaciones móviles gracias a su profesor fue "Desarrollo de aplicaciones para dispositivos móviles',' impartida por mi tutor David de Andrés.

8.2 Trabajos futuros

Algunos trabajos que podrían añadirse a Flatlife para que fuera más robusto y atractivo son los siguientes:

- Implementar código de testing.
- Implementar un sistema de notificaciones de tipo push.
- Incorporar el servicio de Google Analytics.
- Traducir la aplicación a otros idiomas.
- Mantener un historial sobre todo lo que pasa en grupo.

Bibliografía

[1] Fotocasa, "Perfil de las personas que comparten vivienda." https://research.fotocasa.es/wp-content/uploads/2017/11/Presentacion-Perfil-han-buscado-piso-compartido.pdf. Acceso: 29-06-2020.

[2] Fotocasa, "Experiencia en el alquiler 2018-2019." https://research.fotocasa.es/wp-content/uploads/2020/01/Informe-Experiencia-en-alquiler-en-2018-2019.pdf. Acceso: 29/06/2020.

[3] ISO/IEC/IEEE 29148:2018, "Systems and software engineering — life cycle processes — requirements engineering," tech. rep., International Organization for Standarization, 2018.

[4] J. Arlow and I. Neustadt, *UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition)*. Addison-Wesley Professional, 2005.

[5] M. Wei, "ART vs Dalvik - Introducing the New Android x86 Runtime." https://software.intel.com/content/www/us/en/develop/blogs/art-vs-dalvik-introducing-the-new-android-x86-runtime.html. Acceso: 30/06/2020.

[6] M. Sesma and A. Batanero, "Apps híbridas VS Apps Nativas." https://www.paradigmadigital.com/dev/versus-apps-hibridas-vs-apps-nativas/, 2018. Acceso: 30/06/2020.

[7] M. Kremer, "Ionic vs. Everyone: Comparising Cross-Platform Frameworks." https://ionicframework.com/resources/articles/ionic-vs-react-native-a-comparison-guide. Acceso: 20-01-2020.

[8] A. Ravichandran, "React Native or Flutter - What Should I Pick To Build My Mobile App." https://medium.com/@adhithiravi/react-native-vs-flutter-what-are-the-differences-b6dc892f0d34. Acceso: 30/06/2020.

[9] G. Karwchan, "Redux Vs. MVC, Why and How?." https://blog.gisspan.com/2017/02/Redux-Vs-MVC,-Why-and-How.html. Acceso: 30/06/2020.

[10] S. Chacon and B. Straub, *Pro Git (Second Edition)*. Apress, 2014.

[11] D. Thomas and A. Hunt, *The Pragmatic Programmer (20th anniversary edition)*. Addison-Wesley, 2020.

[12] G. A. Campbell, "Cognitive complexity," tech. rep., SonnarSource, 2018.

[13] N. Prieto *et al.*, *Empezar a programar usando Java*. Byprint Percom, 2013.

CAPÍTULO 9
Glosario

- **Software Development Kit (SDK)**: Conjunto de herramientas software que permiten la creación de aplicaciones para un sistema concreto, como por ejemplo los SDK de Android, Windows o iPhone.
- **CSS (Cascade Style Sheet)**: lenguaje que se utiliza para describir la presentación de documentos HTML o basados en XML.
- **Compilación Ahead-of-time (AOT)**: Compilación de un programa a código máquina que se lleva a cabo antes de su ejecución.
- **Dalvik Virtual Machine (DVM)**: Máquina virtual Android optimizada para dispositivos móviles. La máquina virtual fue utilizada por las versiones de Android inferiores a Android 5.0 Lollipop.
- **Compilación Just-in-time (JIT)**: Compilación de un programa a código máquina que se lleva a cabo en tiempo de ejecución.
- **Android Runtime (ART)**: Entorno de ejecución utilizado por Android a partir de la versión Android 5.0 Lollipop, reemplazando la antigua máquina virtual Dalvik. ART introduce la compilación AOT en android, compilando ficheros en formato "dex"(bytecode para la Dalvik) para generar un ejecutable en el dispositivo en el que se instala. Posteriormente, el entorno de ejecución ejecuta el código máquina y se encarga de comunicarse con el kernel Linux del dispositivo.
- **Compilador LLVM**: Compilador basado en el proyecto LLVM, que es, un conjunto de librerias de código abierto que facilitan la creación de compiladores, además de tener implementaciones de librerias conocidas como la librería estándar de C++.
- **Progressive Web App (PWA)**: Aplicación móviles creadas usando tecnología web capaces de realizar funciones nativas del dispositivo como el soporte offline, la sincronización en segundo plano, las notificaciones de tipo push.
- **HyperText Markup Language (HTML)**: Lenguaje de marcado que define mediante etiquetas la estructura de una página web, indicando ordenadamente los diferentes elementos que aparecen en pantalla: texto, imágenes, listas, hipervínculos, etc.
- **Cascade Style Sheet (CSS)**: Lenguaje que indica cómo los elementos de un lenguaje de marcado deben de ser presentados visualmente. En HTML puede indicar aspectos como el tamaño de un texto, la separación entre elementos, colores, posiciones y muchas cosas más.

- **JavaScript (JS)**: Lenguaje de programación interpretado que en aplicaciones web se utiliza para programar el comportamiento de una página. También puede ser utilizado para programar servidores.
- **Dart**: Lenguaje de programación creado por Google para la creación de aplicaciones multiplataforma.
- **Ataque man-in-the-middle (MitM)**: Ataque en el cual el atacante consigue interceptar una conexión entre dos dispositivos haciendo que todos los datos pasen a través de él.
- **Función pura**: función que dadas unas entradas, devuelve siempre la misma salida. Una función pura no tiene efectos secundarios como la modificación de datos que no sean generados en dicha función.