Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Ingeniería de Software

Casual: una aplicación Android para una red social centrada en crear y compartir planes

---

Alumno: Miguel Antonio Núñez Sáiz

Tutor: Yania Crespo González-Carvajal

A mis padres. Por apoyarme en cada paso del camino.

Agradecimientos

Me gustaría agradecer a Laura, por todos estos años junto a mi, por apoyarme en mis mejores y mis peores momentos y, sobretodo, por motivarme a ser mejor persona. No estaría aquí si no es por ti.

Agradecer también, como no podía ser de otro modo, a mis queridísimos compañeros de carrera, con los cuales he vivido tantas experiencias que ya me cuesta recordarlas todas.

Y por último, a mi tutora Yania. Porque su compromiso y dedicación a la enseñanza no sólo es un ejemplo, sino también una inspiración. Gracias por tu apoyo e interminable paciencia.

Muchas gracias a todos.

Resumen

El objetivo de este proyecto es materializar una red social que facilite la creación y descubrimiento de planes casuales, tanto con amigos como con otros usuarios. La aplicación actuará como un espacio centralizado para controlar el acceso a los planes y facilitar la comunicación de sus miembros a través de un chat, permitiendo además la gestión de una red de amistad para poder realizar planes privados.

Se ha desarrollado para ello una aplicación Android, utilizando el lenguaje de programación *Kotlin*, el *framework* de interfaz gráfica *Jetpack Compose*, y *Firebase* como *Backend-as-a-Service*. El sistema se ha estructurado siguiendo los principios de la *Clean Architecture*, empleando *Scrum* como marco de trabajo ágil.

Abstract

The goal of this project is to materialize a social network that facilitates the creation and discovery of casual plans, both with friends and with other users. The application will act as a centralized space to control access to the plans and facilitate the communication of its members through a chat, also allowing the management of a friendship network to make private plans.

An Android application has been developed for this purpose, using the programming language *Kotlin*, the graphical interface *Jetpack Compose*, and *Firebase* as *Backend-as-a-Service*. The system has been structured following the principles of the *Clean Architecture*, using *Scrum* as an agile framework.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

Internet se ha convertido en un elemento imprescindible que impregna todos los aspectos de nuestras vidas. Desde la educación hasta el entretenimiento o la comunicación, internet ha transformado la forma de interactuar con el mundo que nos rodea. Esto es cada vez más evidente si consideramos que el 67 % de la población mundial utiliza internet diariamente [8], un incremento del 45 % respecto del 2018.

El uso del teléfono móvil también ha influido significativamente en la forma de comunicarnos, donde el 78 % de la población mayor de diez años posee un dispositivo móvil [9]. Esta proliferación ha dado como resultado un ecosistema en el que las aplicaciones móviles se han convertido en el principal medio de interacción, en especial a través de las redes sociales.

Estas aplicaciones introdujeron en un comienzo un nuevo paradigma de conexión, marcando una nueva era en la comunicación interpersonal. Revolucionaron nuestra percepción de las relaciones y la forma de cultivarlas. Al salvar sin esfuerzo las distancias geográficas y superar las barreras de comunicación, estas aplicaciones han dado a las personas el poder de establecer conexiones con una facilidad y flexibilidad sin precedentes.

Sin embargo, el panorama actual presenta una nueva problemática. Las redes sociales, ahora controladas por grandes corporaciones [10], buscan maximizar ingresos priorizando la retención de sus usuarios. Se estima que de media, los usuarios dedican 2 horas y 23 minutos al día en redes sociales, un 35 % del tiempo que destinan a internet [11]. Esta tendencia se atribuye a la existencia de algoritmos [12] que seleccionan el contenido que se muestra a los usuarios, a menudo a expensas de la conexión genuina para la que se concibieron originalmente estas plataformas.

En este contexto surge la idea de **Casual**, como una herramienta que permite crear planes genuinos, para poder relacionarte con amigos y conocer nuevas personas, de una manera simple y directa.

1.2. Motivación

Existe una gran cantidad de plataformas sociales, cada una de las cuales nos permite conectar con diferentes círculos de amigos. Aunque la tecnología promete comodidad, la dispersión de estas plataformas y su tendencia cada vez más orientada a retener la atención del usuario complica a menudo el simple acto de hacer planes. Con conversaciones desperdigadas en varias aplicaciones, proponer o entender los planes de los amigos, u otras personas a conocer, se vuelve innecesariamente complejo.

Esta observación condujo a una reflexión más profunda: ¿Hay espacio para una plataforma que, aprovechando la conveniencia que ofrece el formato móvil, permita al mismo tiempo conexiones sociales espontáneas, sencillas y sin filtros? ¿Puede existir un entorno digital en el que la interacción no esté dirigida por algoritmos, sino por una conexión auténtica, inmediata y casual?

Todos estos factores subrayaron la necesidad de **Casual**, un espacio centralizado para sugerir y descubrir planes con amigos, conocidos, o futuros contactos, simplificando nuestra manera de socializar.

1.3. Estudio de alternativas

1.3.1. Eventbrite

Eventbrite [13] es una plataforma que proporciona a los usuarios herramientas para crear, promocionar y asistir a eventos de distintas escalas. Posee un conjunto de funciones dirigidas tanto a los organizadores como a los asistentes.

Ventajas

- Eventbrite puede gestionar una amplia gama de eventos, desde pequeñas reuniones hasta festivales a gran escala.
- Ofrece soluciones integradas para la venta de entradas y los pagos.
- Al ser una aplicación con una gran base de usuarios, los usuarios pueden descubrir numerosos eventos en los que podrían estar interesados.
- La plataforma puede integrarse con redes sociales para promover la asistencia.

Inconvenientes

- Para organizar reuniones sencillas e informales, Eventbrite puede resultar demasiado complejo.
- Se cobran cuotas, especialmente por los eventos de pago.

- La interfaz es un poco abrumadora debido a sus numerosas características y configuraciones.
- La aplicación móvil no ofrece una experiencia tan fluida como la versión de escritorio.
- La plataforma está más pensada para eventos públicos que para reuniones privadas e íntimas entre amigos.

1.3.2. Fever

Fever [14], de la misma manera que Evenbrite, es una plataforma de descubrimiento de eventos en donde existe una recomendación de planes personalizados basados en las preferencias del usuario.

Ventajas

- Permite encontrar eventos exclusivos, lo que facilita que los usuarios descubran experiencias que no encontrarían en otros lugares.
- La posibilidad de reservar billetes o hacer reservas directamente puede agilizar la experiencia del usuario

Inconvenientes

- La lista de eventos de Fever se centra principalmente en las grandes ciudades.
- La abundancia de contenidos visuales e información, genera una interfaz demasiado compleja.

1.4. Objetivos

El objetivo principal de este trabajo es

- Realizar el análisis, diseño e implementación en Android de una aplicación social, que permita a los usuarios crear y participar en planes con sus amigos, conocidos, y posibles nuevos contactos, centrándose en la simplicidad y la facilidad de uso.

Además, se buscan las siguientes metas:

- Participar en un proyecto de software integral, llevándolo desde su fase de idea inicial hasta su realización, fomentando una profunda comprensión del ciclo de vida de desarrollo.

- Entender y dominar los matices del desarrollo nativo de Android, enriqueciendo las habilidades técnicas del estudiante.
- Familiarizarse y perfeccionar el dominio de un proceso desarrollo *Agile* basado en Scrum, asegurando que el proyecto se gestiona de manera eficiente y adaptable.

1.5. Estructura de la memoria

Este documento se estructura de la siguiente forma:

Capítulo 1: Introducción. En este capítulo se presenta el proyecto, explicando el contexto y la motivación para llevarlo a cabo, así como los objetivos principales del mismo.

Capítulo 2: Planificación. Este capítulo aborda el marco de trabajo Scrum seleccionado y su integración en el proyecto. Se detalla el *Product Backlog inicial* y se desglosa la planificación.

Capítulo 3: Tecnologías utilizadas. Este capítulo detalla las diversas tecnologías y herramientas empleadas durante cada etapa del desarrollo del proyecto, proporcionando una comprensión profunda de su aplicación y relevancia.

Capítulo 4: Requisitos y Análisis. Este capítulo describe los resultados de las tareas de análisis realizadas a partir de los requisitos presentados previamente.

Capítulo 5: Diseño. En este capítulo se exponen las decisiones de diseño adoptadas, los patrones usados y el diseño de la interfaz de usuario. Se incluyen también los diagramas, modelos y bocetos que ilustran la arquitectura del sistema creado.

Capítulo 6: Implementación y pruebas. Este capítulo aborda los aspectos clave de la implementación del sistema, incluyendo la integración continua. Se detalla tanto las pruebas desarrolladas como los resultados obtenidos de cada una.

Capítulo 7: Seguimiento del proyecto. Este capítulo detalla la progresión del proyecto, describiendo los avances en cada Sprint, las incidencias y eventos ocurridos, y las dificultades enfrentadas. Se brinda un resumen detallado sobre el tiempo empleado en cada tarea y los pasos seguidos en el desarrollo.

Capítulo 8: Conclusiones. Este capítulo destaca las conclusiones finales alcanzadas y los objetivos logrados. Además, se presentan propuestas y posibles mejoras para ser consideradas en etapas futuras del proyecto.

Anexo B: Enlaces adicionales. Incluye enlaces de interés sobre el proyecto, como el repositorio de código.

Capítulo 2

Planificación

2.1. Scrum

Scrum [15] es un marco ágil y ligero utilizado para facilitar la resolución colaborativa de problemas y producir soluciones incrementales para proyectos complejos. Se basa en ciclos iterativos llamados *Sprints*, que fomentan el trabajo en equipo, la responsabilidad y la mejora continua. Es un proceso guiado por roles, eventos y artefactos, establecidos para maximizar el valor del producto final. Está basado en los principios del **Manifiesto Ágil** [16], que promueve cuatro valores fundamentales:

- Priorizar a las personas y a las interacciones frente a los procesos y las herramientas.
- Valorar el software funcional frente a la documentación exhaustiva.
- Centrarse en la colaboración con el cliente frente a la negociación de contratos.
- Adaptarse al cambio frente a seguir un plan fijo.

En esencia, Scrum promueve un enfoque empírico, haciendo hincapié en la observación y la experimentación, más que en la planificación rígida y detallada.

Los principios esenciales de Scrum: transparencia, inspección y adaptación; son vitales para la efectividad del marco. La transparencia se asegura de que cada fase del proceso y sus desenlaces sean visibles para todos los involucrados. Este nivel de visibilidad facilita la inspección, permitiendo un análisis riguroso del avance comparado con los objetivos definidos. Finalmente, basándose en la inspección, se promueve la adaptación, permitiendo realizar modificaciones y cambios adecuados en tiempo real.

2.1.1. Roles

En el equipo de Scrum se establecen distintos roles [17] con el objetivo de repartir las tareas y asegurar una gestión de trabajo eficaz durante todo el proceso. Estas posiciones son:

- ***Scrum Master***: tiene la tarea principal de verificar que el resto del equipo siga adecuadamente los fundamentos, métodos y directrices de Scrum. Coordina las reuniones, resuelve cualquier problema que pueda surgir en el equipo y sirve como nexo con el resto de la organización.
- ***Product Owner***: actúa como el portavoz de los *stakeholders* y es el intermediario con el cliente. Su misión es gestionar y actualizar el *Product Backlog*, ordenar las tareas en base a su importancia, y asegurar que el equipo entienda las tareas al detalle.
- **Equipo de desarrollo**: Su labor principal es producir un incremento del producto, de manera que pueda ser presentado al concluir cada *Sprint*.

Estos roles son fundamentales en Scrum y se estructuran de manera que se complementan mutuamente, buscando un equilibrio entre ofrecer soluciones funcionales para el cliente y mantener la eficiencia y bienestar del equipo.

2.1.2. Eventos

Dentro del contexto de Scrum, los eventos son espacios de tiempo delimitados diseñados para un propósito específico. Estos eventos crean coherencia y tienen por objeto reducir la necesidad de reuniones no programadas. Los principales eventos en Scrum son:

- ***Sprint***: Es la esencia de Scrum, un espacio de tiempo, fijo a lo largo del proyecto, de entre una y cuatro semanas durante el cual se elabora un incremento. Dentro del *Sprint* se llevan a cabo todas las actividades clave, incluyendo la planificación, *check-ins* diarios, revisiones y retrospectivas.
- ***Sprint Planning***: Al comienzo de cada *Sprint*, el equipo se reúne para delinear el trabajo que se va a realizar durante ese período. Juntos, seleccionan los elementos del *Product Backlog* y establecen un objetivo del *Sprint*.
- ***Sprint Review***: Al final de cada *Sprint*, el equipo muestra lo que ha completado al *Product Owner* y a las partes interesadas. Es una oportunidad para recoger opiniones y hacer los ajustes necesarios para el siguiente *Sprint*.
- ***Sprint Retrospective***: Tras la revisión del *Sprint* y antes de la siguiente planificación, el equipo se reúne para reflexionar sobre el *Sprint* anterior. Se señalan los éxitos y los retos, y se acuerdan las mejoras para el siguiente *Sprint*.
- ***Daily Scrum***: Se trata de una breve reunión diaria en la que el equipo de desarrollo informa sobre los logros del día anterior, su plan para el día en curso y los posibles impedimentos.

Estos eventos están interrelacionados, diseñados para fomentar la transparencia, la inspección y la adaptación a lo largo del desarrollo del producto. Facilitan una comunicación eficaz entre los miembros del equipo y garantizan que el producto progrese de acuerdo con las expectativas y necesidades del cliente.

2.1.3. Artefactos

Scrum emplea varios artefactos [17] que representan el trabajo o el valor, con el objetivo de lograr una mayor transparencia de la información esencial. De esta forma se asegura que cualquier persona que examine estos artefactos está trabajando con datos consistentes, fomentando una base uniforme para cualquier ajuste. Los artefactos generados son los siguientes:

- ***Product Backlog***: Una colección dinámica de tareas, ordenadas por prioridad, destinadas a guiar los esfuerzos del equipo. Esta lista prepara las tareas para los *Sprints* más próximos, siendo refinada de manera regular. Central a este artefacto es el *Product Goal*, que esboza el estado previsto del producto y constituye el objetivo principal del equipo. El principal responsable del *Product Backlog* es el *Product Owner*, aunque todo el equipo contribuye a su refinamiento.
- ***Sprint Backlog***: Consiste en una selección específica de historias de usuario del *Product Backlog*, que el equipo de desarrollo se compromete a completar en un *Sprint* determinado. El objetivo de este artefacto es el *Sprint Goal*, donde se establece una meta clara, permitiendo cierta flexibilidad en cuanto a cómo se alcanza. El equipo de desarrollo es el principal responsable de este artefacto y lo utiliza como referencia durante las *Daily Scrum*.
- **Incremento**: Artefacto generado como resultado de cada *Sprint*, proporcionando valor añadido al producto en desarrollo. Cada paso dado en forma de incremento se agrega con los pasos previos, encaminándose progresivamente hacia el *Product Goal*. Un aspecto esencial es que cualquier labor no se reconoce como parte del incremento a menos que satisfaga la *Definition of Done*, donde se especifica las normas de calidad que el producto debe alcanzar.

2.1.4. Adaptación de Scrum al proyecto

Se ha decidido llevar a cabo los *Sprints* en intervalos de dos semanas. Las reuniones de planificación, revisión y retrospectiva del *Sprint* se realizarán en la intersección entre el final de cada *Sprint* y el comienzo del siguiente. Este enfoque garantiza una utilización eficaz del tiempo disponible.

En lugar de realizar reuniones diarias, se ha decidido llevar a cabo reuniones semanales, o ***Weekly Scrum***. Esta decisión ha sido tomada para garantizar el mantenimiento de una comunicación regular, pudiendo abordar cualquier problema emergente o cambio sin sobrecargar las agendas innecesariamente.

Dada la estructura única del proyecto, que consta en sólo dos personas (tutora y estudiante autor del TFG), es necesario un enfoque a medida para la aplicación de Scrum. Para maximizar la eficiencia y la claridad de roles, la tutora asumirá el papel de *Scrum Master*, mientras que al estudiante se le adjudicará la doble responsabilidad de *Product Owner* y Equipo de Desarrollo.

Dado el doble papel del estudiante como Product Owner y Equipo de Desarrollo, recae en él la responsabilidad de gestionar y mantener todos los artefactos Scrum. Esto abarca la actualización y refinamiento del *Product Backlog*, la gestión del *Sprint Backlog*, y asegurar que cada incremento se adhiere a al *Definition of Done*. Este enfoque fomenta un mayor entendimiento de los requisitos del proyecto.

2.2. Product Backlog Inicial

Siguiendo lo mencionado en la Sección 2.1.4, el *Product Owner* (en este caso el alumno) es el encargado de crear el *Product Backlog*. En Scrum los requisitos se consolidan en formas de historias de usuario. En esta primera etapa, se utilizan historias épicas [18], que representan a grandes rasgos los requisitos fundamentales de la aplicación. En la Sección 4.1 las épicas serán descompuestas en historias más pequeñas una vez sean abordadas en cada *Sprint*, con ayuda de la tutora.

Tanto las épicas como las historias de usuario tienen un identificador único, y siguen el siguiente formato: "Como <*stakeholder*>quiero <*objetivo*>para <*razón que aporta valor*>", donde "*stakeholder*" hace referencia a una entidad con intereses en el proyecto. Para este proyecto se han identificado dos *stakeholders*:

- **Usuarios finales**. Son aquellas personas que utilizarán la aplicación, mayoritariamente jóvenes y familiarizadas con el uso de aplicaciones móviles y redes sociales. Estos usuarios tendrían un estilo de vida social, con interés en asistir a eventos. Requerirán de una conexión a internet.
- **Equipo Scrum**. Son aquellas personas encargadas de materializar la aplicación: el Scrum Master, el Product Owner y el equipo de desarrollo (ver Sección 2.1.4).

En la **Tabla 2.1** se muestran las épicas seleccionadas para el *Product Backlog* inicial.

2.3. Planificación Inicial

Debido a la necesidad de compaginar el desarrollo del proyecto con la realización de un Máster universitario, con las fluctuaciones de disponibilidad que eso conlleva, se ha establecido una carga de trabajo de entre 10 y 20 horas semanales. Se dedicarán por tanto entre 20 y 40 horas por *Sprint*, lo que resulta en una media de **30 horas**.

ID	Épica
EP1	Como nuevo usuario, quiero gestionar mi cuenta y configurar mi perfil, para poder personalizar mi experiencia e interactuar con otros usuarios.
EP2	Como usuario, quiero crear y gestionar planes con diferentes configuraciones de privacidad (Público, Privado o Secreto), para poder organizar eventos que se adapten a diferentes tamaños de audiencia y preferencias de privacidad.
EP3	Como usuario, quiero descubrir planes de otros usuarios, enviar solicitudes de ingreso y gestionar mi participación, para poder implicarme en eventos y conectar con otras personas.
EP4	Como usuario que participa en un plan, quiero interactuar con el resto de participantes a través de una función de chat, de modo que pueda mejorar mi experiencia en el evento comunicándome y colaborando antes y durante el mismo.
EP 5	Como usuario, quiero gestionar las solicitudes de amistad y ver los perfiles de mis amigos, de modo que pueda construir y mantener una red, facilitando la participación en planes privados y secretos.

Tabla 2.1: Product Backlog Inicial

Para designar las historias del *Product Backlog* a cada *Sprint*, se empleará un sistema de puntos de historia lineal, donde cada punto de historia corresponde a 5 horas de trabajo. La selección de historias y la asignación de puntos se llevarán a cabo durante la reunión de planificación del *Sprint*.

Teniendo en cuenta el alcance del proyecto, se ha determinado un plan inicial de 9 *Sprints*. Se dispone, sin embargo de dos *Sprints* adicionales para completar cualquier tarea que no se haya podido realizar dentro de los *Sprints* previstos. Cabe destacar que antes del inicio del primer *Sprint*, se realizó un ***Sprint 0***, dedicando 40 horas a la documentación, preparación del proyecto y formación en las tecnologías utilizadas.

Según los cálculos, en la planificación inicial se ha designado un total de 270 horas a lo largo de los 9 *Sprints*, sumadas a las 40 horas del *Sprint 0* , lo que equivale a un total de 310 horas. Esta planificación se ilustra en la **Figura 2.2**. Esto puede ampliarse a 340 o 370 horas, dependiendo de si se necesitan uno o dos *Sprints* de apoyo adicionales. Esto se ajusta a las indicaciones de la guía docente del Trabajo Fin de Grado, donde se estipula una carga de trabajo aproximada de 300 horas para el estudiante.

Nº Sprint / Evento	Fecha Inicio	Fecha Fin
Sprint 0	17/11/2023	01/12/2023
Sprint 1	01/12/2023	15/12/2023
Sprint 2	15/12/2023	29/12/2023
Sprint 3	12/01/2023	26/01/2024
Sprint 4	09/02/2024	23/02/2024
Sprint 5	23/02/2024	08/03/2024
Sprint 6	08/03/2024	22/03/2024
Sprint 7	22/03/2024	05/04/2024
Sprint 8	05/04/2024	19/04/2024
Sprint 9	19/04/2024	03/05/2023
Sprint Adicional 1	03/05/2024	17/05/2024
Sprint Adicional 2	17/05/2024	31/05/2024

Tabla 2.2: Planificación inicial de los Sprints

2.4. Plan de gestión de riesgos

Se entiende el **riesgo** [19] como un evento o situación que, de suceder, impacta al proyecto causando alteraciones en los resultados u objetivos establecidos.

En la fase de planificación, un buen plan de gestión de riesgos es un componente esencial para garantizar el éxito del proyecto, ya que sirve de guía estructurada para identificar, evaluar y mitigar dichos riesgos. Es crucial para mantener el control sobre las distintas facetas del proyecto y garantizar la realización de sus objetivos. Un plan bien diseñado permite a los equipos anticiparse y abordar estratégicamente los distintos obstáculos que se puedan presentar.

Para evaluar el nivel de los riesgos se tienen en cuenta dos factores principales: la probabilidad y el impacto. La probabilidad es la posibilidad de que se produzca el riesgo, y el impacto es la consecuencia potencial que podría afectar al proyecto en caso de producirse. La evaluación de la probabilidad frente al impacto permite priorizar los riesgos y centrar la atención y los recursos en aquellos más importantes. Los niveles de riesgo se clasifican en bajo, medio y alto:

- **Bajo**: Implica un impacto y una probabilidad mínimos, requiere pocos recursos para su mitigación y a menudo se considera aceptable, sin necesidad de medidas inmediatas.

Riesgo R01	
Título	Ausencia temporal del estudiante por enfermedad o baja
Descripción	Debido a causa de enfermedad o baja, el estudiante no puede realizar sus tareas asignadas durante un periodo determinado de tiempo
Probabilidad	Media
Impacto	Bajo
Plan de mitigación	Llevar un estilo de vida saludable. Planificar dos Sprints extra para permitir holgura en la fecha de entrega
Plan de contingencia	Para ausencias breves, redistribuir las tareas dentro del mismo Sprint. Para ausencias prolongadas, posponer las tareas para el siguiente Sprint. En el caso de que la ausencia se produzca en el Sprint 9, hacer uso de los Sprints extra destinados para ese propósito

Tabla 2.3: Riesgo 01: Ausencia temporal del estudiante por enfermedad o baja

- **Medio**: Posee un impacto y una probabilidad moderados, y requiere medidas de mitigación equilibradas y proactivas, así como un seguimiento cuidadoso.

- **Alto**: Conlleva un impacto significativo y una alta probabilidad, lo que requiere una planificación inmediata y una asignación sustancial de recursos para una gestión eficaz.

Desde la **Tabla 2.3** a la **Tabla 2.10** se exponen los riesgos analizados. Por cada uno de ellos se muestra título, descripción, probabilidad, impacto, plan de mitigación y plan de contingencia.

2.5. Plan de Presupuestos

Dada la naturaleza académica de este proyecto, se han planteado dos presupuestos diferentes. El **presupuesto simulado** estima los costes del proyecto si se llevase a cabo y tuviera como objetivo salir al mercado. El **presupuesto real** estima los costes reales de desarrollar el proyecto como Trabajo de Fin de Grado.

Riesgo R02	
Título	Ausencia temporal de la tutora por enfermedad o baja
Descripción	Debido a causa de enfermedad o baja, la tutora no puede desempeñar sus funciones como Scrum Maste
Probabilidad	Media
Impacto	Bajo
Plan de mitigación	Tener contacto de manera continua para poder detectar el riesgo a tiempo.
Plan de contingencia	Continuar el trabajo ya asignado, documentando cualquier tipo de duda, hasta que se pueda realizar la reunión por videoconferencia. Emplear otros medios de comunicación para enviar dudas, que pueden ser tratadas en la reunión

Tabla 2.4: Riesgo 02:Ausencia temporal de la tutora por enfermedad o baja

Riesgo R03	
Título	Fallos técnicos del ordenador del estudiante
Descripción	Debido a un fallo técnico o percance. El estudiante no puede utilizar su equipo.
Probabilidad	Media
Impacto	Bajo
Plan de mitigación	Mantener una copia de seguridad en la nube con relativa frecuencia. Evitar utilizar el ordenador en situaciones o lugares donde puede resultar perjudicado
Plan de contingencia	Utilizar otro dispositivo para recuperar los documentos relevantes de la nube. En el caso de pérdida de documentos, emplear los métodos de recuperación disponibles o contratar la ayuda de un profesional

Tabla 2.5: Riesgo 03: Fallos técnicos del ordenador del estudiante

Riesgo R04	
Título	Estimación inexacta del tiempo
Descripción	Planificación de una línea de tiempo excesivamente optimista, lo que resulta en una prolongación del tiempo requerido para realizar las tareas.
Probabilidad	Media
Impacto	Medio
Plan de mitigación	Tener en consideración las estimaciones de tiempo de tareas previas. Estimar las tareas con un margen extra para tener en cuenta imprevistos. Centrarse primero en las partes esenciales del proyecto y dejar los detalles para el final
Plan de contingencia	Búsqueda de la solución al problema concreto en las guías y documentación oficial. Emplear herramientas alternativas ya conocidas por el estudiante

Tabla 2.6: Riesgo 04: Estimación inexacta del tiempo

Riesgo R05	
Título	Desconocimiento de la tecnología a utilizar
Descripción	Algunas de las herramientas que se emplean en este proyecto no se conocen, o se tiene poca destreza en ellas
Probabilidad	Media
Impacto	Medio
Plan de mitigación	Documentarse previamente al inicio del desarrollo de las tecnologías que se van a utilizar.
Plan de contingencia	Búsqueda de la solución al problema concreto en las guías y documentación oficial. Emplear herramientas alternativas ya conocidas por el estudiante

Tabla 2.7: Riesgo 05: Desconocimiento de la tecnología a utilizar

Riesgo R06	
Título	Cambio en los requisitos del proyecto
Descripción	Los requisitos del proyecto cambian por parte del Product Owner (el estudiante), una vez ya se ha establecido el Product Backlog inicial
Probabilidad	Media
Impacto	Medio
Plan de mitigación	Tener claro desde el principio cuales son las funcionalidades que se desean para el proyecto.
Plan de contingencia	Analizar el nuevo requisito y determinar si puede ser incluido sin suponer un aumento significativo del tiempo de desarrollo del proyecto. Replanificar las tareas para acomodar el nuevo requisito, en caso de que sea viable.

Tabla 2.8: Riesgo 06: Cambio en los requisitos del proyecto

Riesgo R07	
Título	Disponibilidad de menor tiempo por asuntos académicos
Descripción	Debido a la realización de un Máster universitario por parte del estudiante, la planificación de el Sprint puede verse truncada por la entrega de trabajos o exámenes.
Probabilidad	Alta
Impacto	Medio
Plan de mitigación	Planificar el Sprint teniendo en cuenta los eventos académicos que tendrán lugar las dos semanas posteriores.
Plan de contingencia	Reasignar las tareas para el siguiente Sprint. Realizar un mayor número de tareas cuando se tengan periodos de menor carga académica. Hacer uso de los Sprints de refuerzo

Tabla 2.9: Riesgo 07: Disponibilidad de menor tiempo por asuntos académicos

Riesgo R08	
Título	Comunicación deficiente entre estudiante y tutora
Descripción	Escasa comunicación provocada por realización de pocas reuniones de seguimiento del proyecto
Probabilidad	Baja
Impacto	Medio
Plan de mitigación	Usar el marco de trabajo Scrum para delinear los procedimientos adecuados para reunirse
Plan de contingencia	Organizar reuniones fuera de la planificación ordinaria para lograr un mayor entendimiento entre tutora y alumno. Reorganizar las tareas que no hayan podido realizarse por la falta de comunicación.

Tabla 2.10: Riesgo 08: Comunicación deficiente entre estudiante y tutora

2.5.1. Presupuesto Simulado

Los salarios de los empleados representan el mayor coste del proyecto. De acuerdo con las necesidades del proyecto, se requieren los siguientes puestos:

- **Desarrollador Android**: Ambas fuentes consultadas [20][21] estiman que el salario medio para un desarrollador de Android en España es de 30.000 € anuales. Esto supone un coste de **15,38 €/hora**, suponiendo una jornada de 8 horas/día.
- ***Product Owner***: Las fuentes consultadas [22][23] realizan una estimación similar, por lo que se tomará la media entre ambas, dando un salario anual de 44.250 €. Suponiendo nuevamente jornadas de 8 horas/día, implica un coste de **22,69 €/hora**.
- ***Scrum Master***: Tras consultar varias fuentes [24][25], se estima que el salario medio para un *Scrum Master* en España es de 42.600 € anuales. Esto supone un coste de **20,17 €/hora**.

Es necesario tener en cuenta el coste adicional por empleado en relación la cuota de la Seguridad Social [26], que supone un incremento del **33,4 %** para el empleador. Esta cuota está añadida al precio unitario de la **Tabla** 2.11.

Otro coste relevante es el equipo que utiliza el desarrollador. Se ha seleccionado un **Lenovo ThinkBook 14** con un precio de 699 €. Se estima que tiene una vida útil de cuatro años, por lo que supone un coste de **14.5 €/mes**. Para probar la aplicación durante el desarrollo se le asignará un **Google Pixel 8 Pro**, con un valor de 1.099 € y una vida útil de 7 años. Esto supone un coste de **13,08 €/mes**.

Concepto	Precio Unitario	Cantidad	Total
Desarrollador Android	20,51 €	240 horas	4.922,40 €
Product Owner	30,27 €	35 horas	1.059,45 €
Scrum Master	26,91 €	35 horas	941,85 €
Lenovo ThinkBook 14	14.50 €	5 meses	72,50 €
Google Pixel 8 Pro	13,08 €	5 meses	65,40 €
Microsoft Empresa Básico	16,80 €	5 meses	84,00 €
Figma Organization	25,00 €	5 meses	125,00 €
Astah Professional	15,00 €	5 meses	77,00 €
Subtotal			7.347,60 €
+25 % de normalización			**9.184.5 €**

Tabla 2.11: Presupuesto simulado

Para realizar los *wireframes*, se necesitaría una licencia de **Figma** [27], cuyo precio es de **25 €/mes** utilizando el plan *Organization*. La licencia **Astah Professional** [28] para realizar los modelos y diagramas supone 180 € por un año de uso, es decir, **15 €/mes**. Por último, se requiere una licencia de **Microsoft 365 Empresa** Básica, con un precio de 5,60 €/mes por persona, lo que sería **16,8 €/mes**.

Una vez calculado el total, se le añadirá un incremento del 25 % para hacer frente a aquellos imprevistos que puedan surgir. En la **Tabla 2.11** se aprecia el desglose de los costes y el total, **10.311,00 €**.

2.5.2. Presupuesto real

En el contexto académico de este proyecto, no es necesario tener en cuenta los salarios, ya que todas las funciones son desempeñadas por el estudiante y el tutor.

El ordenador del estudiante, un **Asus ROG Strix Scar**, valorado en 2.100 €s, tiene una vida útil de 4 años, lo que supone un coste de**43,75 €/mes**. Para las pruebas de la aplicación, el estudiante utilizó un **Google Pixel 8** con una vida útil de 7 años y un precio de 799 €, resultando en un coste mensual de **9,51 €/mes**. Se utilizaron licencias académicas de **Microsoft 365**, **Astah Professional** y **Figma**, que permiten el uso gratuito del software.

En la **Tabla 2.12** se aprecia el coste de cada elemento, con un coste total de **266,5 €**.

Concepto	Precio Unitario	Cantidad	Total
Asus ROG Strix Scar	43,75 €	5 meses	218,75 €
Google Pixel 8	9,51 €	5 meses	47,55 €
Subtotal			**266,5 €**

Tabla 2.12: Presupuesto real

Capítulo 3

Tecnologías utilizadas

3.1. Android

Android [29] es un sistema operativo de código abierto basado en Linux, diseñado principalmente para dispositivos móviles con pantalla táctil. El entorno de desarrollo recomendado, en el que se ha llevado a cabo este proyecto, es **Android Studio**.

Para este proyecto se ha utilizado la última API estable disponible a fecha de comienzo del proyecto, la **API 34**. Esta API corresponde con la versión de **Android 14**. Pese a que el porcentaje de dispositivos que utilizan esta versión es reducido, se ha tomado esta decisión para poder aprovechar al máximo las últimas tecnologías que ofrece el SDK de Android.

3.1.1. Kotlin

Kotlin [30] es un lenguaje de programación estático de código abierto, permite tanto programación orientada a objetos como funcional. Es gestionado por JetBrains y Google. Un elemento fundamental de Kotlin es su interoperabilidad con Java, siendo posible utilizar todas sus librerías [31]. Se ha optado por desarrollar la aplicación en Kotlin ya que es el lenguaje recomendado por Google. El 67 % de los desarrolladores profesionales afirman un aumento de productividad al usar Kotlin. Las aplicaciones de Android que contienen código de Kotlin tienen un 20 % menos de probabilidades de fallar [32]. Cuenta además con varias ventajas como reducir la base de código estándar, interoperabilidad y simultaneidad estructurada. Se han utilizado dos elementos imprescindibles en Kotlin: corrutinas y *flows*.

Corrutinas

Las corrutinas [33] son un patrón de diseño de concurrencia en Kotlin, introducido en la versión 1.3, que simplifica la programación asíncrona en Android. Son especialmente eficaces

para gestionar tareas de larga duración que, de otro modo, podrían bloquear el hilo principal y provocar que las aplicaciones no respondan. Más de la mitad de los desarrolladores profesionales que utilizan corrutinas han constatado un aumento de la productividad.

Flows

Un **Flow** es un tipo de datos que emite valores secuencialmente de forma asíncrona, siendo estos valores emitidos del mismo tipo [34]. En la Figura 3.1 se indican las tres entidades clave:

- **Productor**: Genera los datos para el flujo. Al emplear corrutinas, los flows pueden producir datos de manera asíncrona. Para esta aplicación, la implementación de los repositorios en la capa de datos son los encargados de producir los datos (ver en el Capítulo 5 todas las cuestiones relacionadas con la arquitectura y diseño de esta aplicación).
- **Intermediario** (Opcional): Modifica cada valor del flujo. Suelen ser clases de los Casos de Uso de la capa de dominio. Las operaciones intermedias no ejecutan ningún código, sólo establecen una cadena de operaciones para su futura ejecución y regresan rápidamente. Esto se conoce como una propiedad de flujo *frío*.
- **Consumidor**: Utiliza los valores del flujo. Para esta aplicación se refiere a los ViewModels de la capa de presentación. Consumir los valores se considera una operación terminal, las operaciones terminales desencadenan la ejecución de todas las operaciones intermedias y siempre se realizan de forma suspendida, sin bloqueo real.

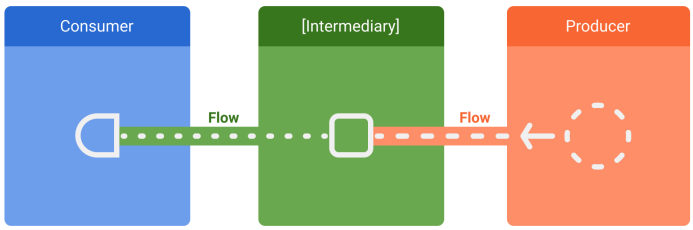

Figura 3.1: Entidades involucradas en los flujos de datos [1]

3.1.2. Jetpack Compose

Jetpack Compose [35] es un kit de herramientas moderno desarrollado por Google para crear interfaces de usuario nativas en aplicaciones de Android.

Es un marco declarativo, lo que contrasta con el enfoque imperativo del diseño con XML. Esto simplifica y acelera el desarrollo de interfaces gráficas, permitiendo a los desarrolladores describir interfaces con código basado en Kotlin. Utiliza un modelo de *widgets* sin estado, es decir, no expone funciones *setter* ni *getter*. En Compose, los *widgets* no se tratan como

objetos, sino que la interfaz de usuario se actualiza llamando a funciones componibles con distintos argumentos. Por tanto, son las funciones componibles las encargadas de modificar el estado actual en una interfaz de usuario cada vez que los datos de la lógica son actualizados. Esto se observa en la Figura 3.2.

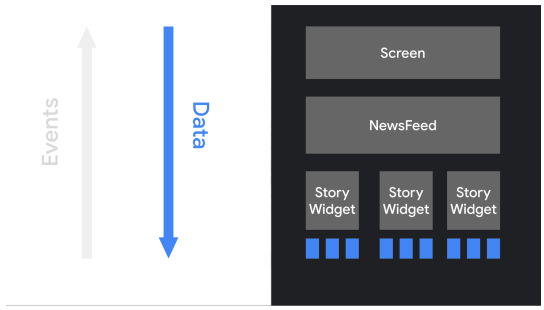

Figura 3.2: Flujo de datos en Jetpack Compose [2]

Las interacciones del usuario desencadenan eventos (por ejemplo, onClick) que notifican a la lógica de la aplicación, lo que provoca cambios en el estado de la aplicación. Este cambio provoca una recomposición, en la que las funciones componibles se vuelven a invocar con nuevos datos, lo que da lugar a que los elementos de la IU se redibujen para reflejar el estado actualizado. Esto se indica en la Figura 3.3.

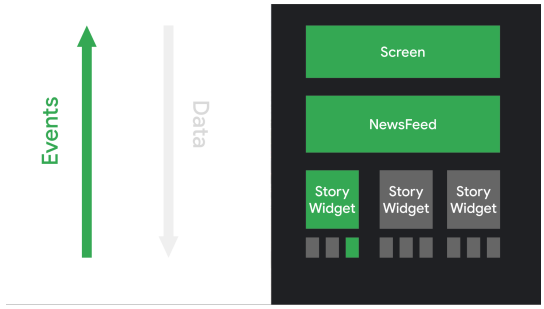

Figura 3.3: Flujo de eventos en Jetpack Compose [2]

Cuando Jetpack Compose ejecuta las funciones componibles por primera vez, durante la composición inicial, mantendrá un registro de los mismos para describir la interfaz gráfica. Cuando el estado de la aplicación cambia, Jetpack Compose programa una recomposición, en la que se vuelven a ejecutar los componibles que pueden haber cambiado en respuesta a los cambios de estado. Por último, actualiza la composición para reflejar los cambios. Este ciclo de vida se ejemplifica en la Figura 3.4.

3.1.3. Material Design 3

Material Design 3 [36] es la última iteración del sistema de diseño de Google para crear interfaces de usuario intuitivas y visualmente atractivas. Se basa en los principios establecidos

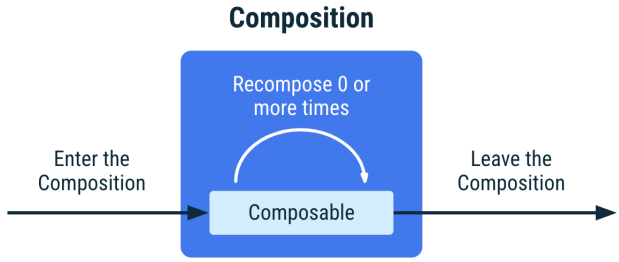

Figura 3.4: Ciclo de vida de una composición [3]

en previas iteraciones, centrándose en elementos de diseño más flexibles y adaptables y en una mayor interactividad. Introduce nuevas directrices y componentes que se adaptan a una amplia gama de tipos de dispositivos y tamaños de pantalla, garantizando una experiencia de usuario coherente y cohesionada en diferentes plataformas.

Entre los aspectos clave de Material Design 3 que se han incorporado al proyecto se incluyen esquemas de color actualizados con funciones de tematización dinámica, que permiten a la interfaz de usuario adaptar su aspecto en función de las preferencias del usuario.

3.1.4. Coil

Coil [37] es una librería de carga de imágenes para Android, diseñada para funcionar con Kotlin y Jetpack Compose. Simplifica las tareas de obtención, decodificación y visualización de imágenes de diversas fuentes, incluyendo la web y el almacenamiento local. Todas la imágenes mostradas en la aplicación de esté proyecto han sido cargadas con Coil.

3.1.5. Dagger-Hilt

Dagger-Hilt [38] es una librería de inyección de dependencias para Android. Hilt ofrece una manera estándar de inyectar dependencias, generando contenedores para cada clase de Android y gestionando su ciclo de vida. Al estar desarrollada sobre Dagger [39], se beneficia de su corrección en tiempo de compilación, eficiencia de ejecución y escalabilidad.

3.2. Firebase

Firebase [40] es una plataforma basada en la nube diseñada para el desarrollo de aplicaciones web y móviles, utilizando la infraestructura de **Google Cloud**. Funciona como *Backend-as-a-Service* y engloba una gran variedad de servicios *backend* en la nube, así como plataformas de desarrollo de aplicaciones.

3.2.1. Authentication

Firebase Authentication [41] es una función de la plataforma de desarrollo de Google, diseñada para gestionar la autenticación de usuarios tanto en aplicaciones web como móviles.

Para registrar un usuario en la aplicación, se obtiene primero sus credenciales, como su correo electrónico y contraseña o un *token OAuth* de un proveedor de identidad federado. Se pasan dichas credenciales al SDK de autenticación de Firebase, que devuelve una respuesta con información básica del perfil del usuario.

De entre la variedad de métodos de autenticación, en esta aplicación se ha empleado la identificación por correo electrónico y contraseña; y a través de Google como proveedor externo.

3.2.2. Cloud Storage

Cloud Storage [42] es un servicio de almacenamiento de objetos en la nube simple y potente, que garantiza la seguridad de las cargas y descargas de archivos desde la aplicación, incluso en condiciones de mala calidad de red.

Este servicio guarda los archivos en *buckets* de **Google Cloud Storage**, lo que ofrece flexibilidad para gestionar las transferencias desde el cliente móvil utilizando el SDK de Firebase. Está diseñado para escalar automáticamente, eliminando la necesidad de migrar a otros proveedores de almacenamiento a medida que crezca la base de usuarios de la aplicación.

Además, los SDK de Firebase para almacenamiento en la nube están integrados con Firebase Authentication, lo que proporciona una identificación segura de los usuarios.

3.2.3. Firestore

Cloud Firestore [43] es una base de datos NoSQL basada en documentos, escalable horizontalmente, que permite almacenar y sincronizar datos en la nube. Está diseñada para proporcionar una sincronización de datos en tiempo real sin fisuras entre los dispositivos de los usuarios, por lo que es ideal para aplicaciones sociales como la de este proyecto.

La plataforma accede directamente a la aplicación Android a través de una SDK nativa. Utiliza un modelo de datos NoSQL, en el que los datos se almacenan en documentos conformados por campos a los que se asignan valores. Estos documentos se organizan en colecciones, contenedores que ayudan tanto a estructurar los datos como a facilitar la construcción de consultas. Cloud Firestore admite además la creación de subcolecciones dentro de los documentos, permitiendo construir estructuras de datos jerárquicas que escalan de manera eficiente con el crecimiento de la base de datos.

Firestore ofrece además sólidas normas de seguridad para controlar el acceso a los datos tanto a nivel de documento como de campo [44]. Para este proyecto se han creado reglas de seguridad específicas para controlar dicho acceso.

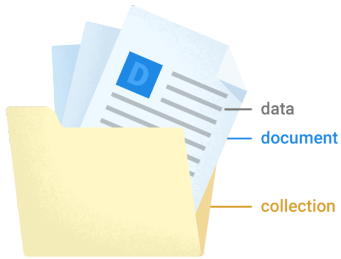

Figura 3.5: Modelo de datos de Firestore [4]

3.3. Gestión del proyecto

3.3.1. Git

Git [45] es un sistema de control de versiones distribuido, gratuito y de código abierto que permite rastrear los cambios realizados a cualquier conjunto de archivos. Se suele emplea para registrar y coordinar los cambios en el código fuente y la configuración de proyectos de software.

La unidad conceptual fundamental en Git es el *commit*, que representa instantáneas de las modificaciones realizadas en los archivos dentro de la carpeta del proyecto. Cada vez que se realiza un cambio en un archivo, Git crea una versión comprimida del mismo y la almacena en un *commit* individual.

Otro elemento importante es el concepto de rama [46]. Una rama es una línea de desarrollo separada del código base, que permite a los desarrolladores trabajar en nuevas funciones o corregir errores de forma aislada. Cada rama representa una serie independiente de *commits*, permitiendo desarrollar de manera paralela. Las ramas pueden crearse, fusionarse y eliminarse según sea necesario.

En Git, el proceso de desarrollo gira en torno a tres áreas principales: el directorio de trabajo, el área de preparación (o índice) y el directorio Git (repositorio). En la Figura 3.6 se indica el flujo de trabajo habitual con Git.

3.3.2. GitHub

GitHub [47] es una plataforma web y un servicio de *hosting* que facilita el desarrollo colaborativo de software mediante el sistema de control de versiones Git. Proporciona además control de acceso, seguimiento de errores, gestión de tareas e integración continua.

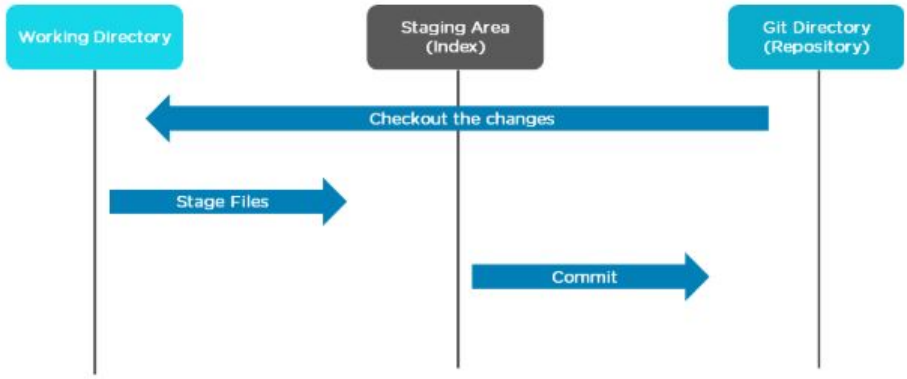

Figura 3.6: Flujo de trabajo con Git [5]

3.4. Diseño, análisis y documentación

3.4.1. Astah Professional

Astah Professional [28] es una herramienta avanzada de modelado de software, utilizado principalmente en las etapas de análisis y diseño. Es compatible con numerosos tipos de diagramas, como UML, Entidad-Relación y diagrama de flujo. Los diagramas resultantes de las tareas de análisis y diseño de la aplicación han sido realizado con esta herramienta.

3.4.2. Figma

Figma es una herramienta de diseño colaborativa basada en la nube que permite crear y compartir diseños para páginas web, aplicaciones móviles y otros productos digitales[27]. Los prototipos de la aplicación se han realizado utilizando esta plataforma.

3.4.3. Overleaf

Overleaf [48] es un editor LaTeX basado en la nube, diseñado para la redacción y publicación de documentos académicos y profesionales. Permite la colaboración en tiempo real, lo que hace que varios autores puedan editar documentos simultáneamente. Entre sus principales características se incluyen la comprobación automática de errores, el control de versiones y la integración con servicios de publicación académica. El presente documento ha sido creado usando esta herramienta.

3.5. Comunicación

3.5.1. Microsoft Teams

Microsoft Teams [49] es una plataforma de comunicación y colaboración desarrollada por Microsoft. Ofrece un conjunto de herramientas que facilitan la comunicación, la colaboración y la productividad. Proporciona funcionalidades como chat, videoconferencia, almacenamiento de archivos e integración con diversas aplicaciones de Microsoft. Microsoft Teams ha sido el principal medio de comunicación entre la tutora y el estudiante, manteniendo reuniones semanales en esta plataforma.

Capítulo 4

Requisitos y Análisis

4.1. Requisitos funcionales

Como se ha indicado en la Sección 2.2, en la etapa de análisis se han desglosado las historias épicas en historias de usuario. Al utilizar Scrum como metodología de desarrollo, estas historias actúan como requisitos funcionales del proyecto. Las historias de usuario se indican desde la Tabla 4.1 a la Tabla 4.5, cada tabla indica las historias correspondientes a cada épica.

Código	Historia de usuario
HU01	Como nuevo usuario, quiero registrarme en la aplicación para crear una cuenta.
HU02	Como nuevo usuario, quiero iniciar sesión en mi cuenta para acceder a la aplicación.
HU03	Como usuario, quiero cambiar mi contraseña para mantener mi cuenta segura.
HU04	Como usuario, quiero poder editar mi perfil para agregar información personal.
HU05	Como usuario, quiero cargar una foto de perfil para personalizar mi cuenta.
HU06	Como usuario, quiero poder cerrar sesión en mi cuenta cuando no la esté utilizando.

Tabla 4.1: Historias de usuario de la épica EP1

Código	Historia de usuario
HU07	Como usuario, quiero poder crear un nuevo plan, definir su información relevante, así como su privacidad (Público, Privado o Secreto).
HU08	Como anfitrión de un plan, quiero poder ver una lista de las solicitudes de unión a mis planes para ver quien está interesado en asistir.
HU09	Como anfitrión de un plan, quiero tener la capacidad de aceptar o rechazar cada solicitud de unión para gestionar la asistencia al mismo.
HU10	Como usuario, quiero poder ver una lista de mis planes creados para un rápido acceso y gestión.
HU11	Como anfitrión de un plan, quiero tener la opción de editar la información de un plan existente, para mantener a todos los participantes informados de cambios.
HU12	Como anfitrión de un plan, quiero eliminar el plan creado para evitar que se acumulen planes ya finalizados.
HU13	Como anfitrión de un plan, quiero poder eliminar participantes de mi plan para controlar la asistencia.

Tabla 4.2: Historias de usuario de la épica EP2

Código	Historia de usuario
HU14	Como usuario, quiero poder visualizar los planes públicos creados por el resto de usuarios para descubrir experiencias interesantes y participar en ellas.
HU15	Como usuario, quiero poder visualizar los planes privados creados por mis amigos mantenerme informado y unirme a ellos si lo deseo.
HU16	Como usuario, quiero ver los detalles de un plan para decidir si quiero unirme.
HU17	Como usuario, quiero poder enviar una solicitud de unión a un plan para esperar la aprobación del anfitrión.
HU18	Como usuario, quiero ver la lista de los planes a los que me he unido para un rápido acceso y gestión
HU19	Como participante de un plan, quiero poder abandonar un plan en el que ya no deseo participar.

Tabla 4.3: Historias de usuario de la épica EP3

Código	Historia de usuario
HU20	Como participante en un plan, quiero poder acceder a un chat dedicado al plan para comunicarme con otros participantes.
HU21	Como participante en un plan, quiero poder enviar mensajes en el chat del plan para discutir detalles y coordinar.
HU22	Como anfitrión, quiero tener la capacidad de moderar y administrar el chat del plan para garantizar un entorno adecuado para todos los participantes.

Tabla 4.4: Historias de usuario de la épica EP4

Código	Historia de usuario
HU22	Como usuario, quiero ver el perfil de mis amigos, incluyendo su información básica, para conocer más sobre ellos.
HU23	Como usuario, quiero ver la lista de mis amigos aceptados, para tener un registro de mis conexiones actuales.
HU24	Como usuario, quiero eliminar amigos de mi lista si ya no deseo mantener la conexión, para gestionar mi lista de amigos.
HU25	Como usuario, quiero poder ver la lista de solicitudes de amistad pendientes, para saber quiénes buscan conectar conmigo.
HU26	Como usuario, quiero tener la opción de aceptar o rechazar cada solicitud de amistad de otros usuarios, para controlar quiénes se agregan a mi lista de amigos.
HU27	Como usuario, quiero poder buscar otras personas por nombre de usuario, para ampliar mi red de amigos y encontrar a otros usuarios.
HU28	Como usuario, quiero poder enviar solicitudes de amistad a otros usuarios, para conectar con nuevas personas.

Tabla 4.5: Historias de usuario de la épica EP5

4.2. Requisitos no funcionales

Para la obtención de los requisitos no funcionales, representados en la Tabla 4.6, se han seguido las recomendaciones del modelo FURPS [50].

ID	Historia de usuario
RNF1	Como equipo de desarrollo quiero implementar la aplicación utilizando la versión 14 de Android para poder ofrecer las últimas funcionalidades de su SDK.
RNF2	Como usuario quiero que los datos que muestra la aplicación estén actualizados en tiempo real con una latencia de menos de 1 segundo, para poder crear y unirme a planes de manera fluida
RNF3	Como usuario quiero que la interfaz gráfica de la aplicación siga las directrices de Material Design 3, de manera que que un usuario promedio pueda realizar una tarea principal en menos de 1 minuto, para otorgar a la aplicación de buenos atributos de usabilidad.

Tabla 4.6: Requisitos no funcionales como historias de usuario

4.3. Modelo de proceso de negocio

En la Figura 4.1 se muestra el principal proceso de negocio que se realizará en la aplicación, unirse a un plan. En este proceso están involucrados dos actores: el anfitrión del plan y el usuario que quiere participar. El primer paso toma lugar cuando el anfitrión crea un plan acorde a sus especificaciones: hora, lugar, visibilidad, título, descripción e imagen. Tras haber completado esta etapa, el participante podrá ver la información del plan creado, siempre y cuando el plan sea público o el participante sea amigo del anfitrión. Si el participante desea formar parte del plan, envía una petición de unión al anfitrión. El anfitrión por su lado puede ver todas las solicitudes de unión a sus planes creados, pudiendo acceder al perfil del emisor de dicha petición. El anfitrión puede entonces aceptar o rechazar la petición, siendo notificado el participante en caso de aceptarla.

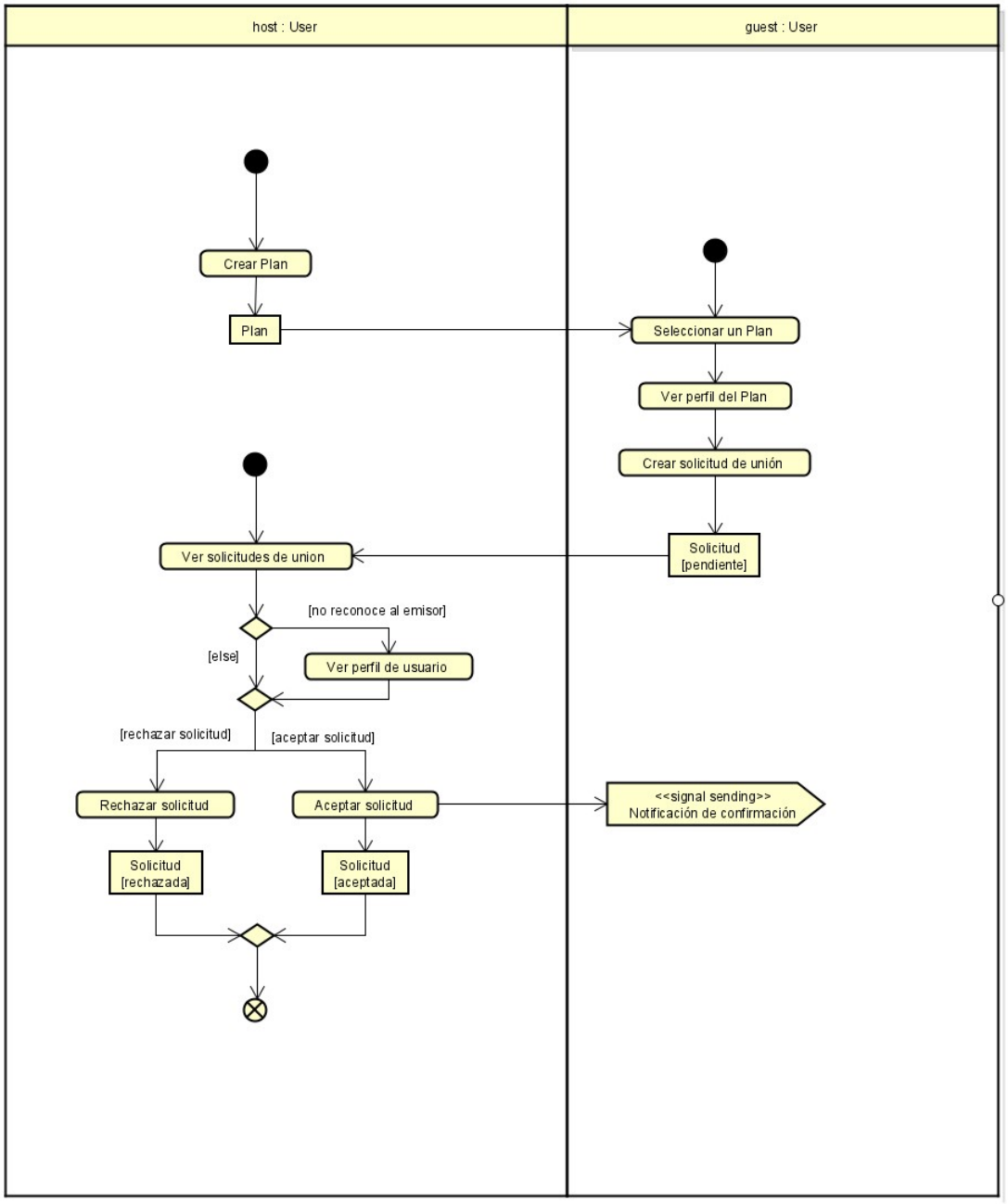

Figura 4.1: Modelo de proceso de negocio

4.4. Modelo de dominio inicial

Los requisitos de información (Tabla 4.7) se han ido refinando a lo largo de los sprints.

ID	Historia de usuario
RI01	Como equipo de desarrollo, quiero guardar los siguientes datos de un **usuario**: identificador, nombre de usuario, nombre, fecha de nacimiento, imagen de perfil, descripción del perfil, lista de amigos.
RI02	Como equipo de desarrollo, quiero guardar los siguientes datos de un **plan**: identificador, título, tipo (público, privado o secreto), fecha, hora, localización, descripción del plan, anfitrión, lista de participantes, chat.
RI02	Como equipo de desarrollo, quiero guardar los siguientes datos de una **mensaje de chat**: identificador, marca temporal, emisor, texto.
RI02	Como equipo de desarrollo, quiero guardar los siguientes datos de una **petición de amistad**: identificador, marca temporal, emisor, receptor, estado
RI02	Como equipo de desarrollo, quiero guardar los siguientes datos de una **petición de unión**: identificador, marca temporal, plan, emisor, receptor, estado

Tabla 4.7: Requisitos de información como historias de usuario

A partir de estos requisitos de información se ha ido obteniendo el modelo de dominio inicial, el cual está representado en el diagrama de clases que se muestra en la Figura 4.2. Este diagrama también es fruto de varias iteraciones.

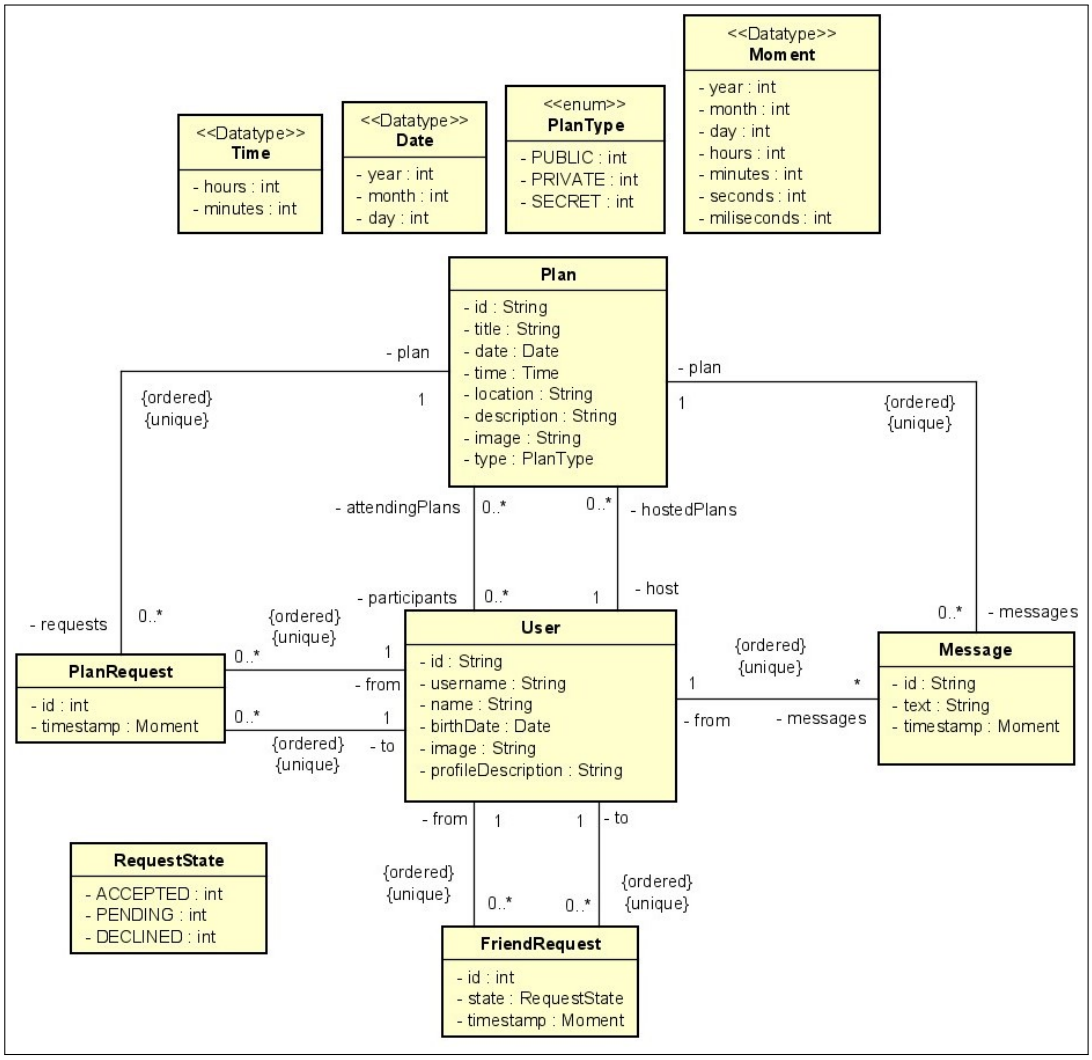

Figura 4.2: Modelo de dominio inicial

Capítulo 5

Diseño

5.1. Clean Architecture

Se denomina **Clean Architecture** a la guía que encapsula los principios de diseño establecidos por **Robert C. Martin** para producir sistemas de software [51]. Siguiendo estos principios, se obtienen sistemas donde las reglas de negocio son independientes de *frameworks*, bases de datos y agente externos; así como fácilmente comprobables. Este proyecto se ha desarrollado siguiendo estos principios.

Clean Architecture involucra muchos patrones y principios de diseño, dentro de los cuales se ha querido destacar los dos más relevantes: la separación de intereses y la Regla de la Dependencia.

5.1.1. Separación de intereses

La separación de intereses [52] es un principio fundamental en el diseño de software (no exclusivo de *Clean Architecture*) que busca dividir un sistema en diversos componentes, lógicamente independientes, responsables cada uno de un aspecto específico de la funcionalidad del sistema. Esto se consigue dividiendo al software en capas. En el caso de este proyecto, se han utilizado las capas de **presentación**, **dominio** y **datos**, donde la capa de dominio es el centro del sistema.

5.1.2. Regla de la Dependencia

La *Regla de la Dependencia* [53] establece que las dependencias del código deben dirigirse únicamente hacia las capas internas. En otras palabras, las capas internas no pueden tener conocimiento alguno sobre las capas externas (Figura 5.1). Siguiendo los principios de la

Clean Architecture, el flujo de datos atraviesa todas las capas, desde la entrada de datos por parte del usuario, pasando por la lógica de negocio, hasta llegar a la persistencia de datos o su envío a un servicio externo, y viceversa con el resultado de la operación, hasta su presentación al usuario. Este flujo se indica en la Figura 5.2.

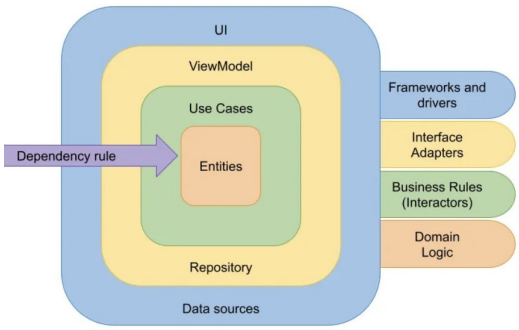

Figura 5.1: Regla de la Dependencia [6]

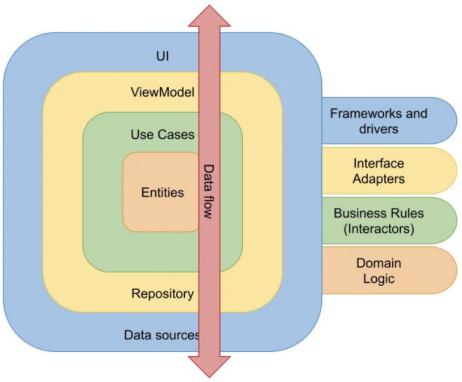

Figura 5.2: Flujo de datos en *Clean Architecture* [6]

Para resolver la aparente contradicción en la que la capa de dominio necesite a la capa de datos, se emplea el **Principio de Inversión de Dependencias** [54]. Este principio dicta que a) los módulos de alto y bajo nivel deben depender de abstracciones y b) estas abstracciones no deben depender de los detalles de implementación. La manera utilizada para implementar este principio se ejemplifica en la Figura 5.3. En la Sección 5.5 se indican los diagramas en los que se muestra el contexto en el que se ha utilizado este principio en la aplicación.

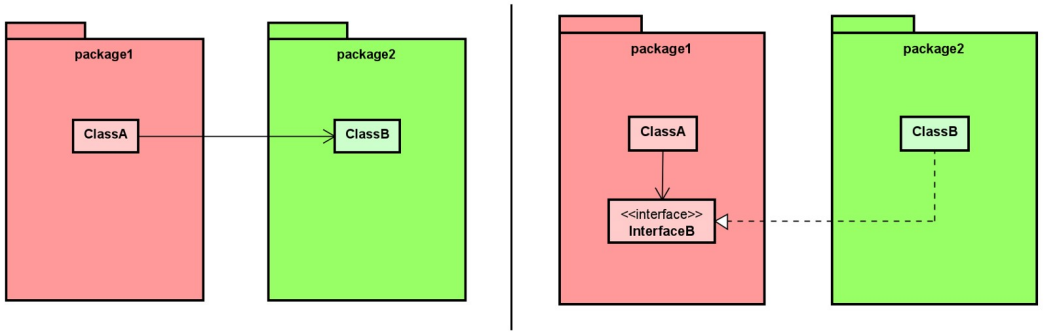

Figura 5.3: Implementación del Principio de Inversión de Dependencias

5.2. MVVM

El patrón MVVM (Modelo - Vista - Modelo de Vista) [7] permite separar la lógica de presentación y negocio de la interfaz de usuario (IU) de la aplicación. Esto hace que la aplicación sea más fácil de probar, mantener y expandir. Existen tres componentes clave en el patrón MVVM: el modelo, la vista, y el modelo de vista. La Figura 5.4 se observa la relación entre estos componentes. La vista conoce al modelo de vista, y el modelo de vista conoce al modelo, pero el modelo no es consciente del modelo de vista, y el modelo de vista no lo es de la vista. De esta manera, el modelo de vista desacopla la vista del modelo.

- **Vista**: Es la responsable de definir la estructura y apariencia de lo que el usuario ve en la pantalla. En este proyecto, la vista esta conformada por las funciones componibles de **Jetpack Compose** (Sección 3.1.2). Pese a que pueden contener lógica de la IU que implemente comportamiento visual, nunca deben contener lógica de negocio.
- **Modelo de Vista**: Conocido como **ViewModel**, implementa las propiedades y comportamientos asociadas a la vista, a la cual notifica de cualquier cambio en el estado a través de notificaciones de evento. El modelo de vista también es responsable de coordinar las interacciones requeridas entre la vista y el modelo. Para esta aplicación, se tiene un modelo de vista por cada pantalla.
- **Modelo** Las clases del modelo son clases que encapsulan los datos de la aplicación, incluyendo el modelo de datos y la lógica de negocio. El modelo en este caso hace referencia a las clases dentro de la capa de dominio.

Al ser Casual una aplicación reactiva, es decir, una aplicación donde se actualizan los cambios en los datos en tiempo real, la unión de datos se realiza entre la vista el modelo de vista se realiza utilizando **StateFlow** [55]. StateFlow es una API que permite a los Flows (Sección 3.1.1) emitir actualizaciones de estado a múltiples consumidores, en este caso las vistas de Jetpack Compose.

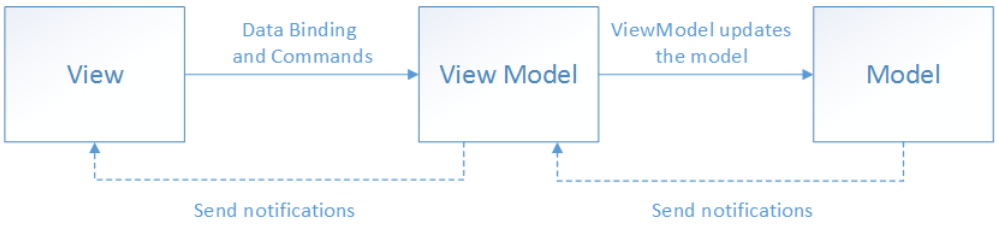

Figura 5.4: Relación entre los componentes del patrón MVVM [7]

5.3. Patrones de diseño

Los patrones de diseño son soluciones típicas a problemas comunes en el diseño de software [56]. No son porciones específicas de código, sino una abstracción que resuelve un problema conocido, por lo que es necesario personalizar la implementación en función de las necesidades del proyecto. Los patrones, en resumen, son descripciones de alto nivel.

5.3.1. Observador

El patrón **Observador** es un patrón de comportamiento que establece un mecanismo de subscripción para informar a varios objetos sobre cualquier cambio o evento que ocurra en el objeto que están monitorizando. Este mecanismo requiere que el notificador almacene una lista de referencias a los objetos subscriptores, así como varios métodos públicos para añadir y eliminar subscriptores de dicha lista [57]. En la **Figura 5.5** se indica la estructura de este patrón.

Este patrón se ha implantado en Casual en aquellos componentes que requieren **actualizaciones en tiempo real**, como los repositorios de la capa de datos que escuchan cambios en la base de datos y las vistas que reaccionan a estos cambios una vez se han propagado hasta su ViewModel.

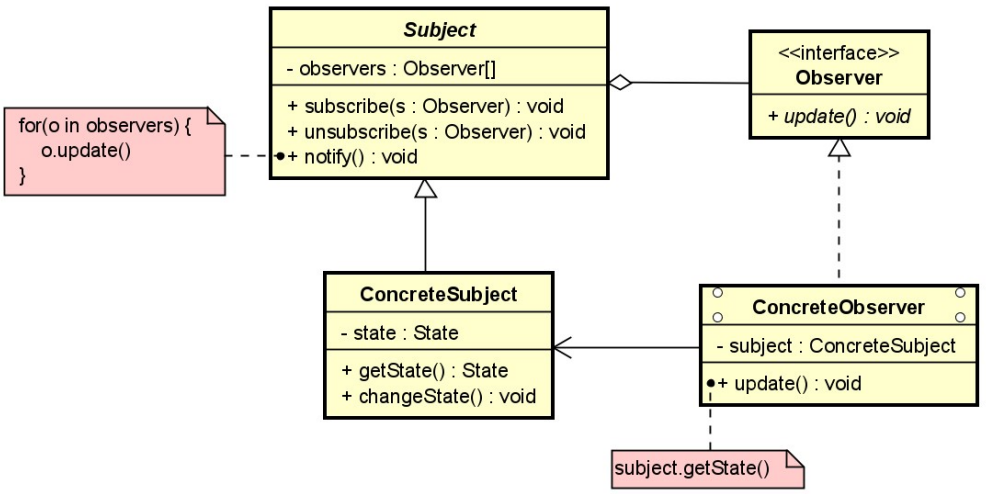

Figura 5.5: Estructura del patrón Observador

5.3.2. Singleton

El patrón **Singleton** es un patrón de diseño creacional que asegura que sólo existe una única instancia de una clase, y provee un punto de acceso global a dicha instancia [58]. En la Figura 5.6 se indica su estructura. Todas las implementaciones de este patrón tienen dos pasos comunes:

- Convertir el constructor de la clase en un constructor privado, para evitar que otros objetos instancien la clase.

- Crear un método de creación estático que actúa como constructor. La primera vez que se llama al método, este llama internamente al constructor privado y almacena la instancia del objeto en un atributo estático. Todas las llamadas posteriores al método de creación devuelven la instancia ya creada.

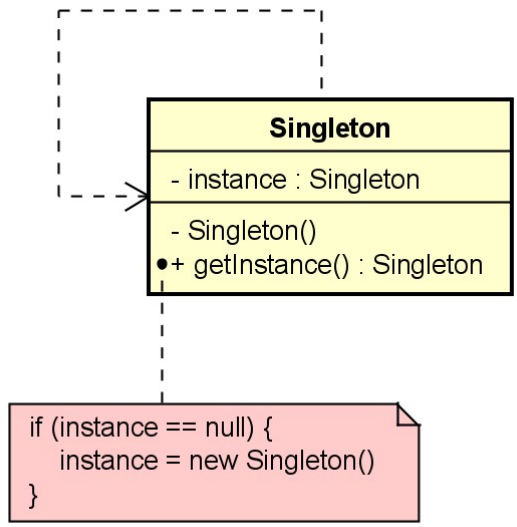

Figura 5.6: Estructura del patrón Singleton

En este proyecto, se ha empleado el patrón Singleton para tener una sola instancia de las clases que representan el punto de entrada a los servicios de Firebase: *FirebaseAuth, FirebaseFirestore* y *FirebaseStorage*.

5.4. Interfaz de usuario

5.4.1. Esquema de colores

Se ha decidido emplear la siguiente paleta de colores para dar uniformidad a la aplicación, y otorgale un aspecto atractivo y profesional. En la Figura 5.7 se muestra de manera más visual.

Gamboge #EE9B00

Raisin Black #1F263

Prussian Blue #213045

Indigo Dye #283A53

Powder Blue #91A9CA

Fire Engine Red #CF1818

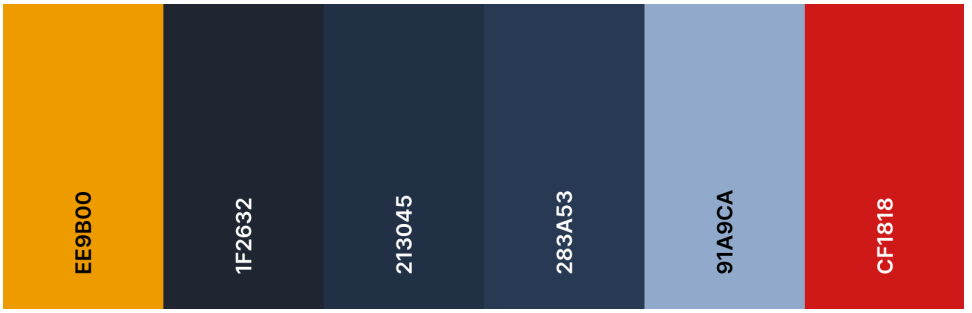

Figura 5.7: Paleta de colores elegida para Casual

5.4.2. Prototipos

En esta sección se incluyen los prototipos para las distintas pantallas de la aplicación. Estos prototipos se muestran en la Figura 5.8 a la Figura 5.17. Como se puede observar se ha mantenido una coherencia estética, tanto en el color como en las formas y la disposición de los elementos.

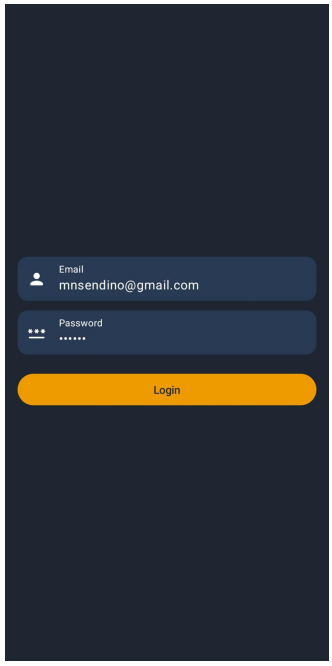

Figura 5.8: Pantalla de LogIn

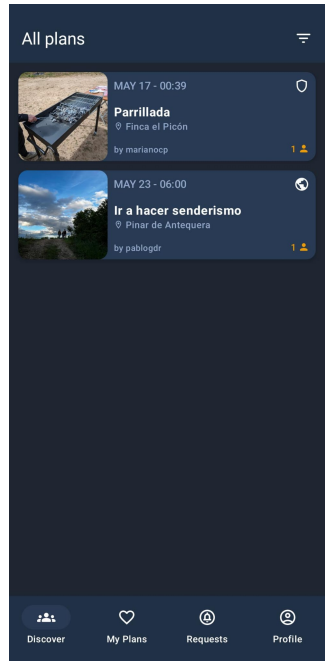

Figura 5.9: Pantalla de Discover

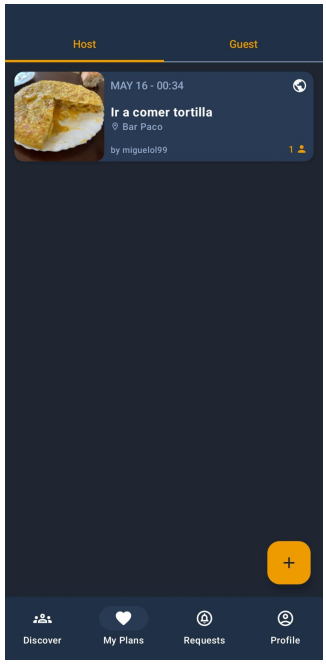

Figura 5.10: Pantalla de MyPlans

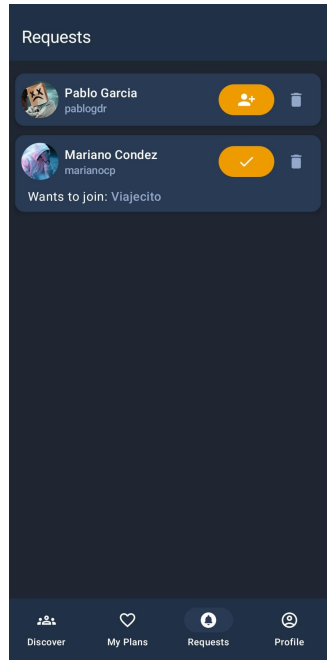

Figura 5.11: Pantalla de Requests

Figura 5.12: Pantalla de MyProfile

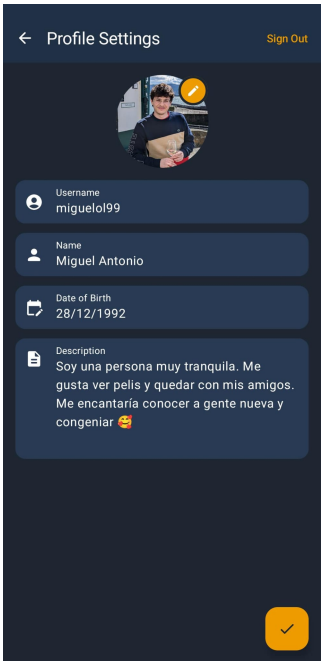

Figura 5.13: Pantalla de Edit Profile

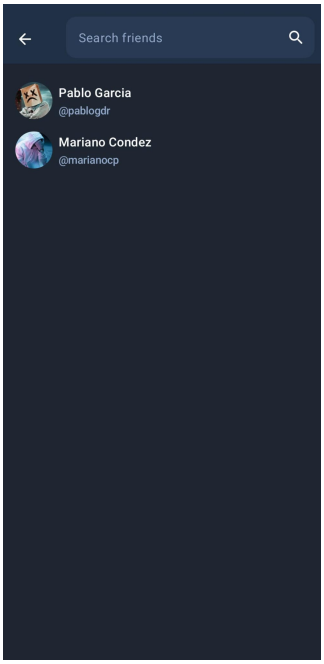

Figura 5.14: Pantalla de MyFriends

Figura 5.15: Pantalla de Other User Profile

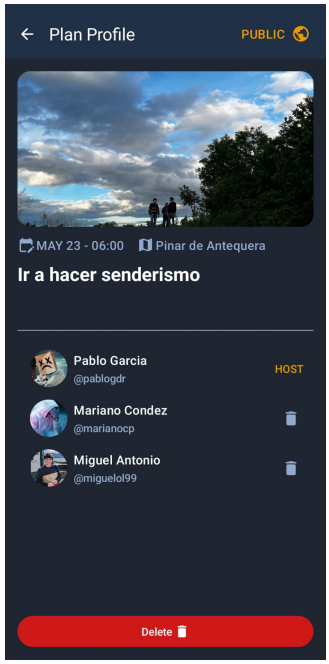

Figura 5.16: Plan Profile

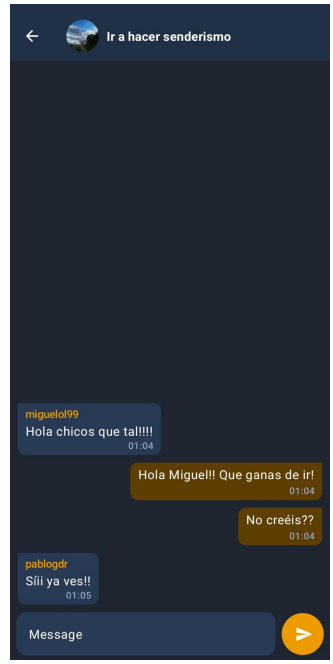

Figura 5.17: Pantalla de Plan Chat

5.5. Descomposición del sistema

Como se ha visto en las secciones anteriores, la aplicación está compuesta por tres capas diferenciadas siguiendo los principios de la *Clean Arquitecture*. El código está estructurado en los siguientes paquetes:

- ***pesentation***: Este paquete representa la capa de presentación de la aplicación. Aquí se encuentran los elementos que permiten mostrar y controlar la interfaz gráfica. Este módulo es nativo y depende del SDK de Android.
  - ***navigation***: En este paquete se encuentran las clases encargadas de la navegación entre las distintas pantallas de la aplicación. El elemento principal es el *CasualNavGraph*, que representa el nodo raíz en la jerarquía de pantallas y está escrito utilizando funciones de Jetpack Compose.
  - ***screens***: Dentro de este paquete están contenidas las distintas pantallas de la aplicación. Al estar utilizando el patrón arquitectónico MVVM (Sección 5.2), cada pantalla está conformada por funciones de Jetpack Compose y de un ViewModel que controla su comportamiento y actúa como puente entre la IU y la capa de dominio.
  - ***theme***: Dentro de este paquete se encuentra la configuración del tema de la aplicación, así como los colores y la tipografía utilizados.
- ***domain***: Este es el paquete central de la aplicación, encapsula toda la lógica de negocio y representa la capa de dominio de la arquitectura. En el patrón MVVM, este paquete representa el Modelo. Este módulo debe estar aislado del resto, y no depender de ninguna librería externa, por lo que es un módulo exclusivamente de Kotlin.
  - ***usecase***: Este paquete contiene reglas de negocio especificas de la aplicación. Encapsula e implementa todos los casos de uso del sistema. Para mejorar la organización, se han agrupado los casos de uso que comparten una funcionalidad común en subpaquetes.
  - ***model***: Este paquete contiene las clases del modelo de dominio de la aplicación.
  - ***respositories***: En este paquete se encuentran los repositorios abstractos. Estos permiten la inversión de dependencia con la capa de datos. Las clases de este paquete definen las funciones que deben ser implementadas por la capa de datos para poder llevar a cabo las reglas de negocio.
- ***data***: Representa la capa de datos. Contiene las clases encargadas de controlar la persistencia de los datos en la base de datos.
  - ***repositories***: Contiene las clases que implementan los repositorios definidos en la capa de dominio.
- ***di***: Este paquete contiene las clases encargadas de la inyección de dependencias. Son clases que utilizan el *framework* Dagger-Hilt.

La **Figura 5.18** muestra un diagrama *Decomposition&Uses style* representando la arquitectura general de la aplicación. Los diagramas para cada capa específica se representan en las **Figuras 5.19, 5.20 y 5.21**.

Es importante señalar que en el diagrama del paquete *presentation* (**Figura 5.19**) no se ha detallado el contenido de los subpaquetes de *screens* para evitar complicar el diagrama. En su lugar, se ha incluido un diagrama detallado del paquete *createPlan* (**Figura 5.22**), que sirve como ejemplo de la estructura seguida por las demás pantallas.

De la misma manera no se ha incluido el contenido de los subpaquetes de *usecase* en el diagrama de *domain* (**Figura 5.20**), sino que se ha indicado en la **Figura 5.23** las clases del paquete *plans*, sirviendo de ejemplo de como se estructuran las clases del resto de casos de uso.

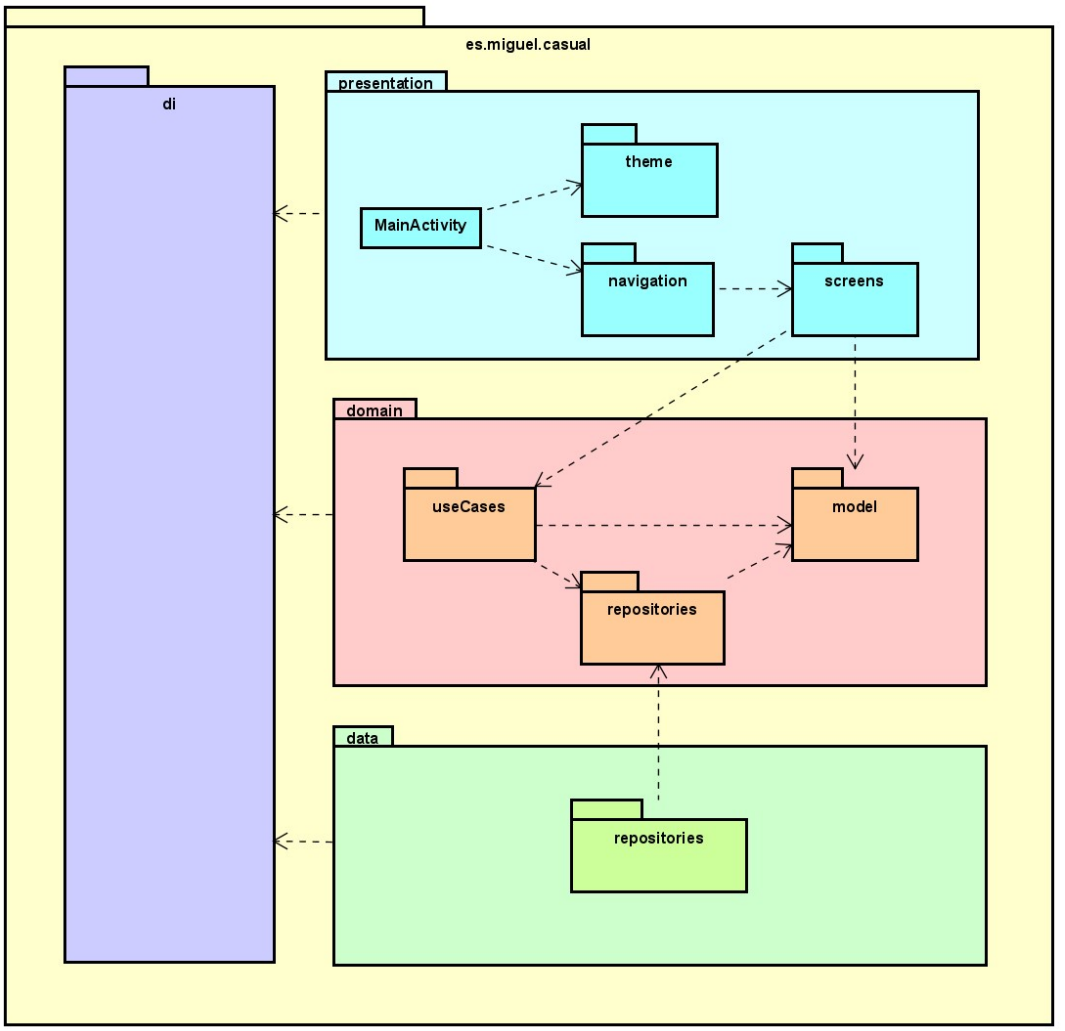

Figura 5.18: Arquitectura general de la aplicación

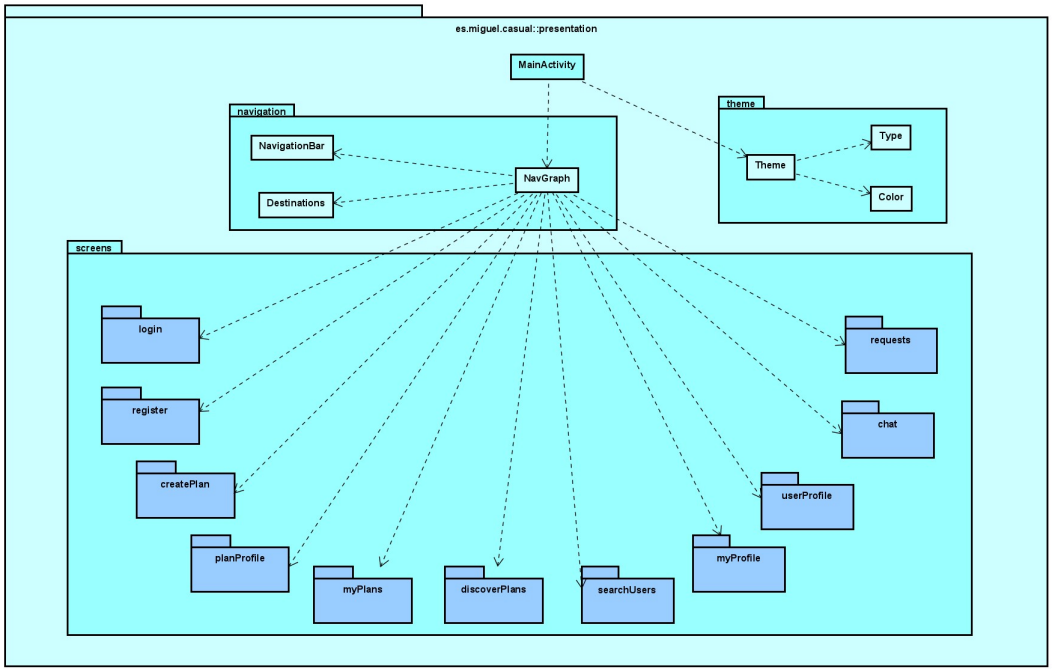

Figura 5.19: *Decomposition&Uses style* del paquete **presentation**

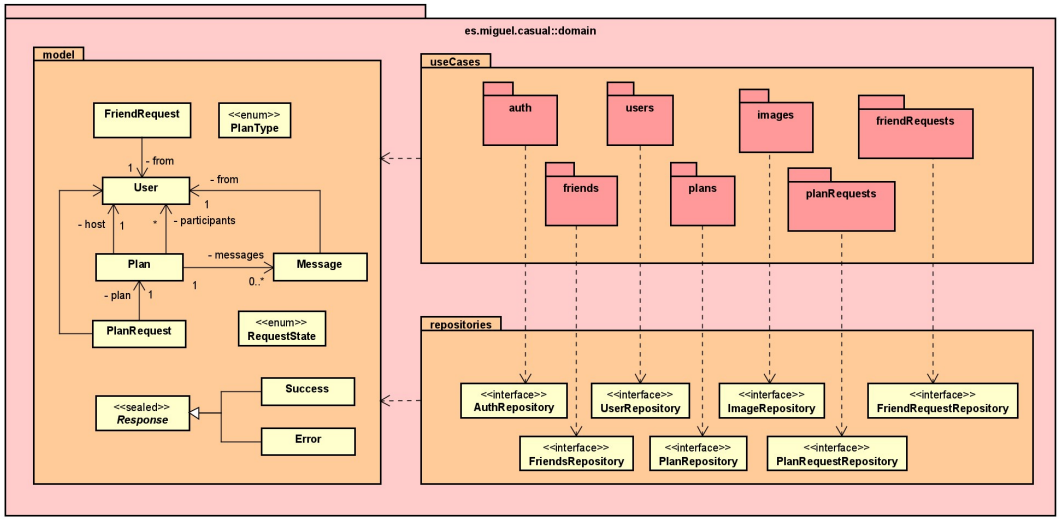

Figura 5.20: *Decomposition&Uses style* del paquete **domain**

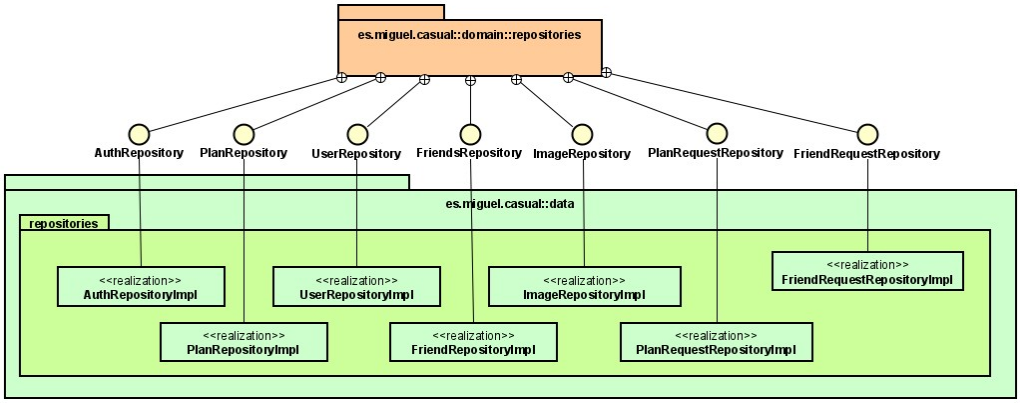

Figura 5.21: *Decomposition&Uses style* del paquete **data**

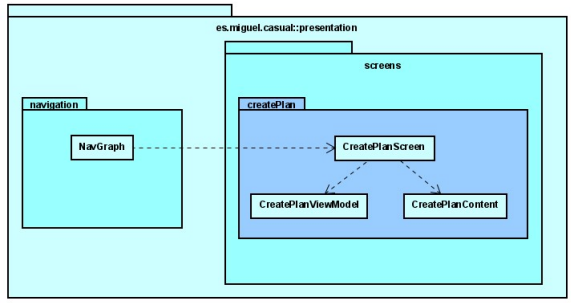

Figura 5.22: Detalle de *Decomposition&Uses style* del paquete **createPlan**

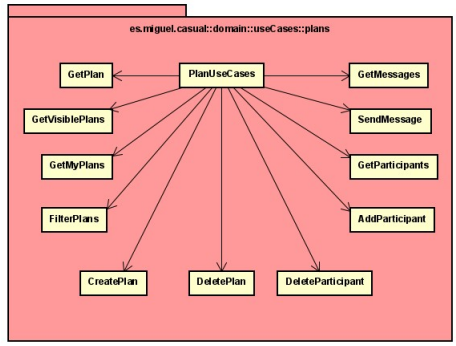

Figura 5.23: Detalle de *Decomposition&Uses style* del paquete **plans**

5.6. Comunicación entre objetos

Para entender cómo se comunican los objetos del sistema, se realizarán los diagramas de secuencia principales de las historias de usuario, relacionadas con enviar y recibir solicitudes de unión a un plan.

5.6.1. HU17: Enviar una petición de unión

La descripción de la historia es la siguiente: **Como usuario, quiero poder enviar una solicitud de unión a un plan para esperar la aprobación del anfitrión.** El diagrama de secuencia principal para esta historia se indica en la **Figura 5.24**. Este diagrama comienza cuando el usuario presiona el botón de unirse a un plan, y finaliza cuando se muestra por pantalla el mensaje de confirmación o error. Se tienen dos consideraciones, la primera es que el usuario se encuentra en la pantalla del perfil del plan, y la segunda es que el usuario intenta unirse a un plan público o a un plan privado creado por un amigo, de lo contrario no tendría acceso al perfil.

Al presionar el botón, la función *PlanProfileScreen* (de Jetpack Compose) ejecuta la llamada a *onEvent* del *PlanProfileViewModel*, que le fue pasada como parámetro. Este, a su vez, notifica a la capa de dominio, en concreto a la clase *PlanRequestsUseCases*. Esta clase encapsula todos los casos de uso relacionados con las peticiones de unión a planes (siguiendo una estructura similar a lo mostrado en la Figura 5.23). En Kotlin, la sobrecarga del operador *invoke* permite llamar a una clase como si fuera una función. Realmente, cuando el ViewModel llama a *createRequest()*, está ejecutando la función *invoke* de *CreatePlanRequest*. Se ha decido modelarlo así en el diagrama para resaltar la estructura del sistema.

En *CreatePlanRequest* se llama a otros casos de uso para crear el objeto *PlanRequest*, que se envía al repositorio de la capa de datos. Hay que tener en cuenta que *PlanRequestRepository* es una interfaz, siguiendo con la Regla de la Dependencia (Sección 5.1.2), por lo que la llamada a su método *createPlanRequest()* es polimórfica. Su implementación se indica en la **Figura 5.25**.

Una se ha completado la escritura en la base de datos, se devuelve un objeto *Response*, que puede ser *Success* en caso éxito o *Error* en caso de que se haya lanzado alguna excepción. Esta respuesta se devuelve al ViewModel, que actualiza el estado de la vista.

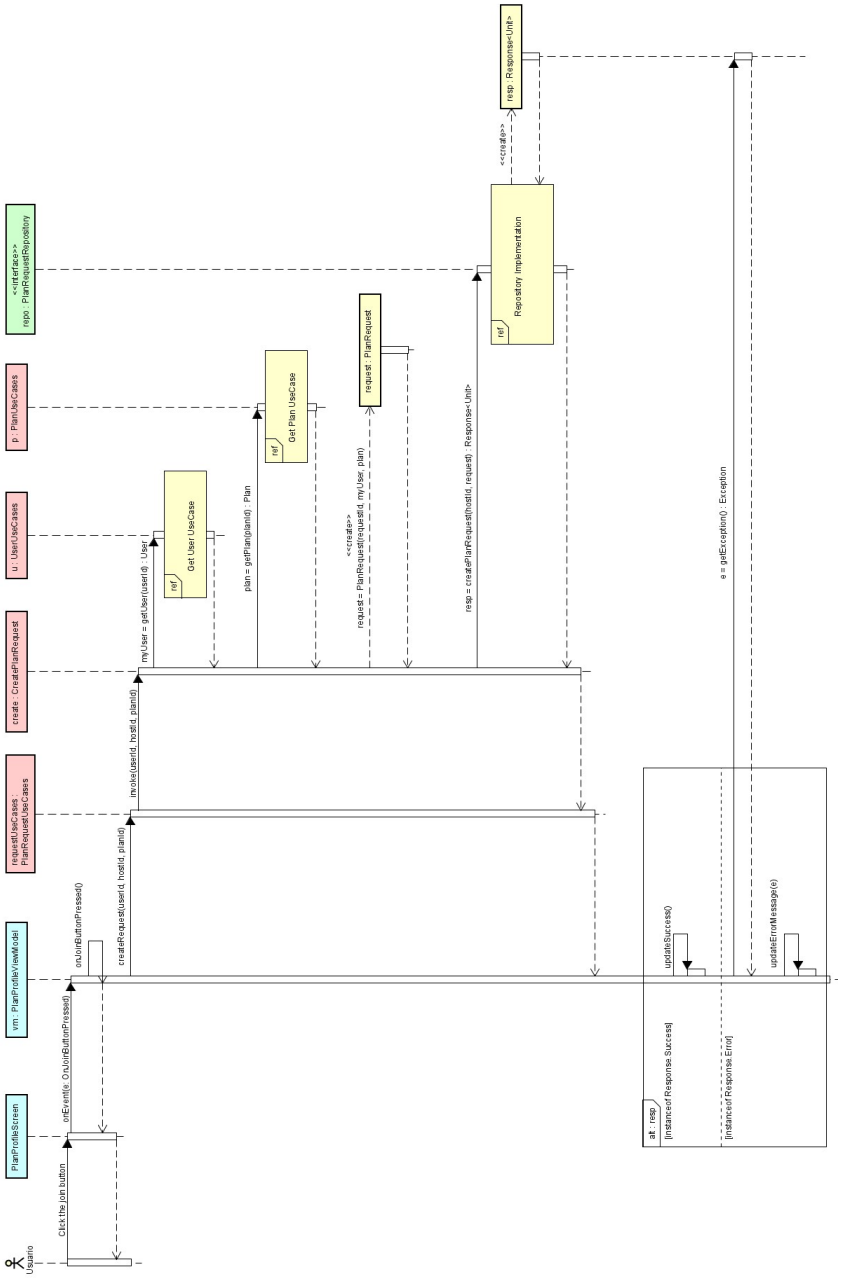

Figura 5.24: Diagrama de secuencia principal de la **HU16**

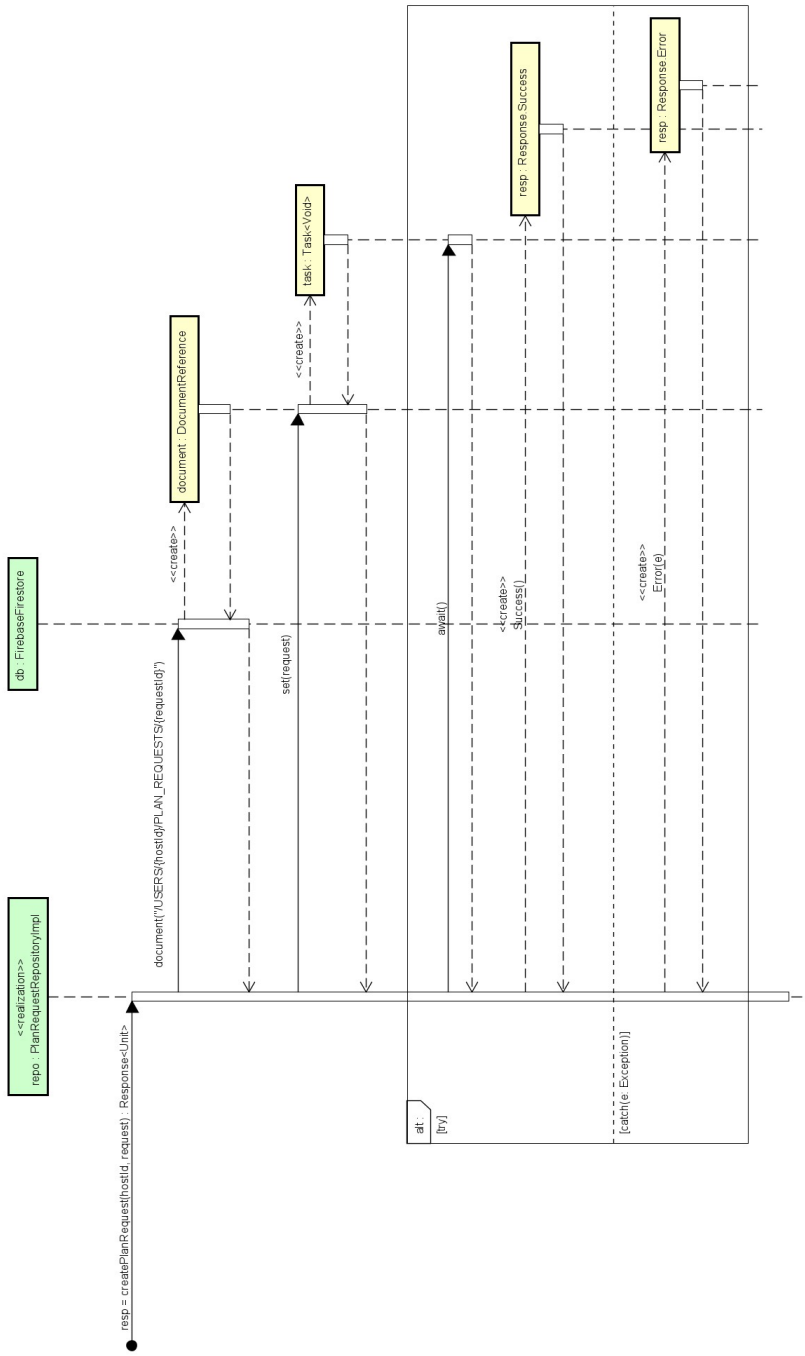

Figura 5.25: Implementación de la capa de datos de la **HU16**

5.6.2. HU08: Ver lista de solicitudes pendientes

La descripción de la historia es la siguiente: **Como anfitrión de un plan, quiero poder ver una lista de las solicitudes de unión a mis planes para ver quien está interesado en asistir.**

El diagrama comienza justo después de que el usuario haya presionado el botón de navegación a la pantalla de *requests* (cuando se está inicializando el ViewModel), y termina cuando se crea el StateFlow, el cual será observado por las funciones de Jetpack Compose para construir la interfaz gráfica. Esta decisión se ha tomado para poder mostrar como funciona el mecanismo de los **Flows** y el **StateFlow**, que permiten actualizaciones en tiempo real de manera reactiva. El diagrama de secuencia que representa este flujo se indica en la **Figura 5.26**.

Al inicializarse el *RequestViewModel*, se obtiene primero del *AuthUseCasess* el identificador del usuario actual. Luego se llama al método *invoke()* de *GetPlanRequests* de la misma manera que la sección anterior.

Posteriormente se notifica a *PlanRequestRepository*, cuya implementación se indica en la **Figura 5.27**. Es aquí donde se obtiene una una referencia a la colección que almacena las peticiones que se quieren obtener, y se crea una consulta para ordenarlas dichas peticiones. A través del método *snapshots()*, se crea por primera vez un Flow.

Como se explica en la Sección 3.1.1, aplicar operaciones intermedias (como *map()* o *catch()*) no realiza ninguna acción, sólo establece una cadena de operaciones que serán ejecutadas cuando se consuma el flujo. Una vez creado el flujo, se utiliza *map()* para transformar el formato de los datos de Firestore al formato del modelo del dominio. Finalmente se devuelve el flujo al ViewModel, donde se realizan dos pasos fundamentales. Primero se transforma los valores del flujo al formato que requiere la interfaz gráfica y, después, convierte el flujo en un StateFlow. Como se ha mencionado anteriormente, StateFlow es un flujo **observable** que emite actualizaciones a sus consumidores, en este caso las funciones de Jetpack Compose.

Una vez se ha creado el StateFlow, este no comienza a emitir valores hasta que la vista empiece a consumir el flujo, utilizando el método *collectWithLifeCycle()*. Gracias a este mecanismo, la interfaz se actualiza automáticamente cada vez que se modifiquen los datos.

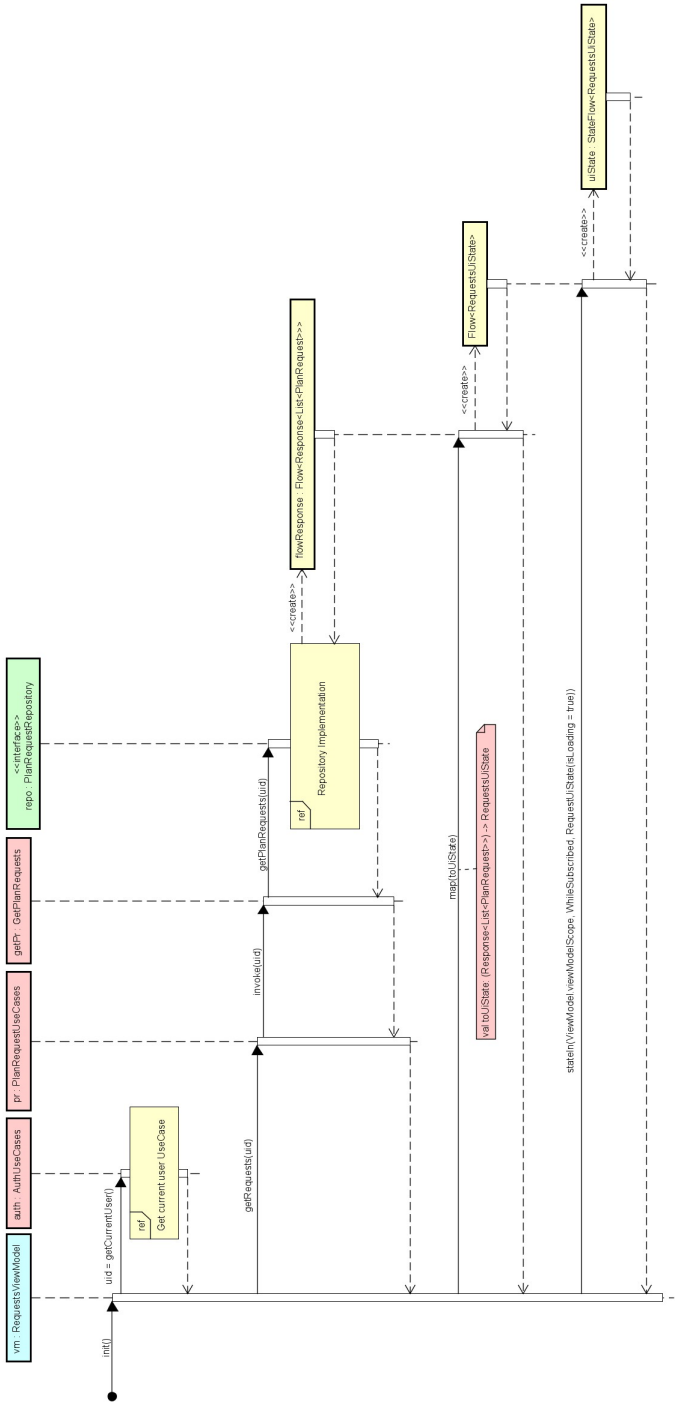

Figura 5.26: Flujo de datos de la **HU13**

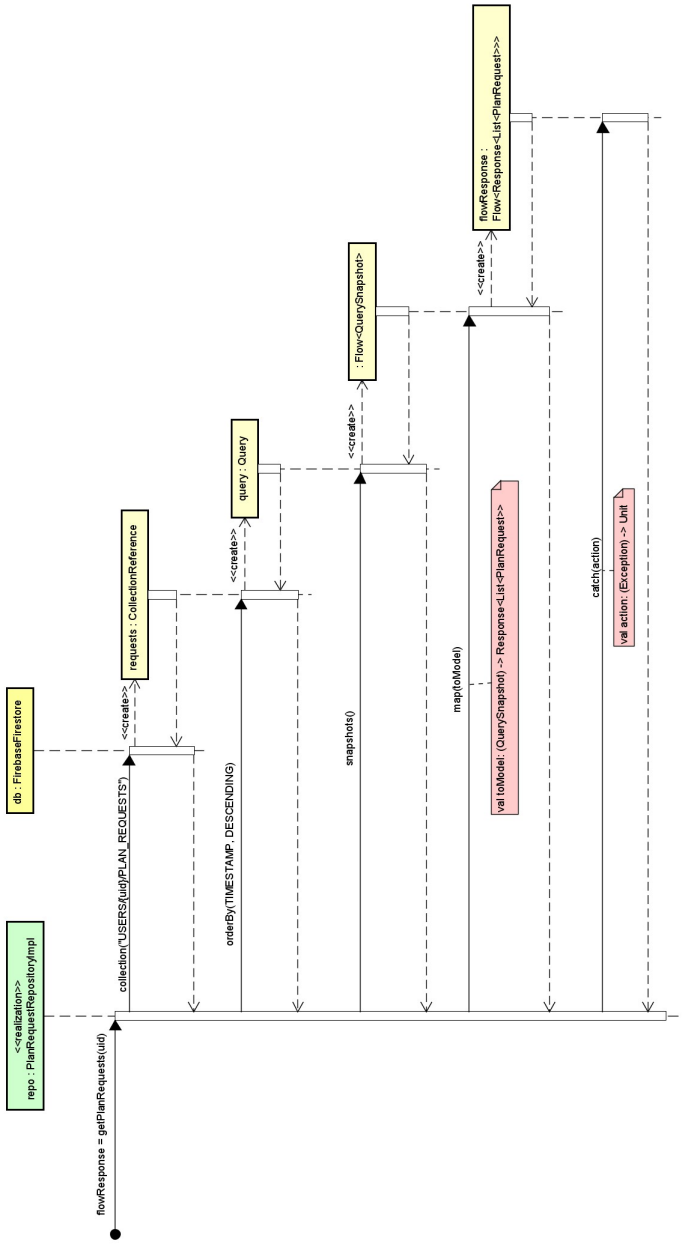

Figura 5.27: Implementación de la capa de datos de **HU13**

5.7. Despliegue

Al utilizar ***Firebase*** como *Backend-as-a-service*, el despliegue se simplifica notablemente, ya que la aplicación solo tiene una única dependencia. En la **Figura 5.28** se observa el diagrama de despliegue.

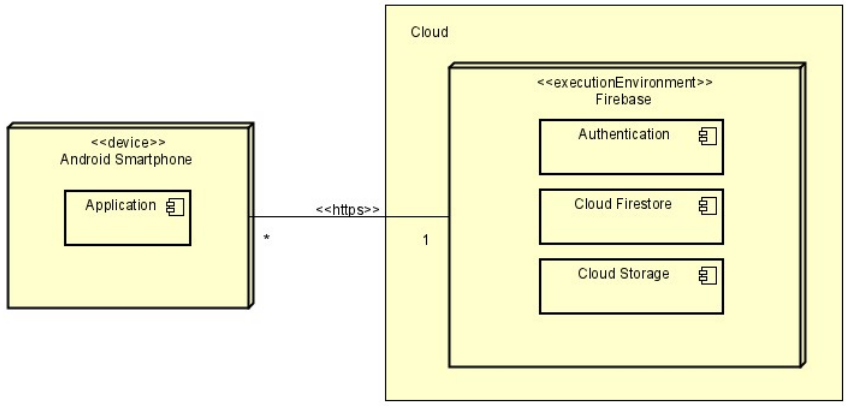

Figura 5.28: Diagrama de despliegue de la aplicación

Capítulo 6

Implementación y pruebas

6.1. Problemas encontrados durante la implementación

Durante la implementación del proyecto, se encontraron varios problemas que dificultaron el desarrollo y, en algunos casos, modificaron el alcance del proyecto. Los siguientes son los más relevantes:

- **Falta de experiencia con Firestore.** Particularmente, me resultó complicado adaptarme al hecho de que los datos en Firestore no están normalizados, priorizando la velocidad de las lecturas sobre las escrituras. Esto implica almacenar datos de manera redundante para evitar operaciones *join* costosas. Otro gran desafíos fue la falta de soporte de Firebase para búsquedas de texto completo, lo que limitó las funcionalidades de filtrar que se tenían pensadas en un primer momento.

- **Mala optimización de la IU.** Debido a una mala comprensión de Jetpack Compose, en especial del concepto de la *recomposición*, la interfaz de usuario iba en ocasiones demasiado lenta. La *recomposición* hace referencia a la reconstrucción selectiva de componentes de la interfaz de usuario en respuesta a cambios en el estado de la aplicación. Antes de darme cuenta del error que estaba cometiendo, la interfaz realizaba cálculos costosos e innecesarios cada ciclo de recomposición.

- **Comprensión limitada de los Flows.** Este conceptos aparentemente sencillo resultó mas complejo una vez tuve que implementar lógica de negocio mas elaborada. Me costó comprender como los datos se propagaban desde la creación hasta su consumo, y como las operaciones intermedias como *map()* no se ejecutan hasta que se consume el flujo. Esto produjo que tuviera que refactorizar varias pantallas para poder explotar todos los beneficios que ofrecen los Flows.

6.2. Casos de prueba

En este capitulo se indican las distintas pruebas realizadas a lo largo del desarrollo de la aplicación. Dado que se han centrado los esfuerzos en la experiencia del usuario final y del cliente, se ha decidido realizar **pruebas de aceptación de historias de usuario**. Este tipo de pruebas se utilizan para asegurar que la implementación de cada historia es la correcta y se ajusta a la especificación [59]. Hay que destacar que las pruebas se han realizado de forma manual, utilizando dispositivos físicos Android.

El conjunto de casos de prueba se muestra de Tabla 6.1 a la Tabla 6.21. Para mantener este documento conciso, no se indican los casos de prueba en lo referente a las solicitudes de amistad, las cuales son muy similares a las referentes a las peticiones de unión (de la Tabla 6.15 a la Tabla 6.20).

En los casos de prueba **CP09, CP10 y CP17**, se indica que el escenario es la pantalla *Profile*. En realidad esos casos de prueba pueden comenzarse desde cualquiera de las pantallas principales de la barra de navegación (*Discover*, *My Plans*, *Requests* y *Profile*), pero por simplicidad se muestra solo uno de esos casos.

CP01	**Iniciar sesión**
Descripción	El usuario inicia sesión con credenciales válidas.
Precondición	El usuario está registrado.
Escenario	Pantalla de inicio de sesión.
Pasos	1. Introducir *mnsendino@gmail.com* en el campo *email*. 2. Introducir *123456* en el campo contraseña. 3. Presionar el botón *Log in*.
Res. Esperado	El usuario ha accedido a la aplicación y ha sido redirigido a la pantalla *Discover*.
Comprobación	El usuario se encuentra en la pantalla *Discover*.

Tabla 6.1: CP01 - Iniciar de sesión

CP02	**Cerrar sesión**
Descripción	El usuario cierra sesión.
Precondición	El usuario está registrado.
Escenario	Pantalla de configuración de perfil.
Pasos	1. Presionar el botón *Sign Out*.
Res. Esperado	El usuario ha cerrado la sesión actual y ha sido redirigido a la pantalla de inicio de sesión.
Comprobación	El usuario se encuentra en la pantalla de inicio de sesión.

Tabla 6.2: CP02 - Cerrar Sesión

CP03	**Editar mi perfil de usuario**
Descripción	El usuario modifica la información personal de su perfil.
Precondición	El usuario está registrado.
Escenario	Pantalla *Profile*.
Pasos	1. Presionar el icono del engranaje de la parte superior derecha de la pantalla. 2. Introducir *miguelol99* en el campo *Username*. 3. Introducir *Miguel Antonio* en el campo *Name*. 4. Presionar el el campo *Date of Birth*. 5. Seleccionar en el calendario la fecha *04/05/1999*. 6. Introducir *Esta es la descripción de mi perfil* en el campo *Description*. 7. Presionar el botón de acción de la parte inferior de la pantalla.
Res. Esperado	Se actualiza la información personal y se redirige a la pantalla *Profile*.
Comprobación	El usuario se encuentra en la pantalla *Profile* y la información mostrada está actualizada correctamente.

Tabla 6.3: CP03 - Editar mi perfil de Usuario

CP04	**Actualizar imagen de perfil**.
Descripción	El usuario actualiza su imagen de perfil.
Precondición	El usuario está registrado.
Escenario	Pantalla *Profile*.
Pasos	1. Presionar el icono del engranaje de la parte superior derecha de la pantalla. 2. Presionar la imagen de perfil de usuario. 3. Seleccionar una foto almacenada en el dispositivo de las mostradas en la ventana emergente. 4. Presionar el botón de acción de la parte inferior de la pantalla.
Res. Esperado	Se actualiza la imagen de perfil y se redirige al usuario a la pantalla *Profile*
Comprobación	El usuario se encuentra en la pantalla *Profile* y la imagen de perfil se ha actualizado correctamente

Tabla 6.4: CP04 - Actualizar imagen de perfil

CP05	**Ver lista de amigos**
Descripción	El usuario visualiza una lista con todos sus amigos
Precondición	El usuario está registrado y tiene al menos un amigo.
Escenario	Pantalla *Profile*
Pasos	1. Presionar el botón con el contador de amigos y el icono de una persona.
Res. Esperado	Se redirige al usuario a la pantalla *Friends* donde se muestran una lista con sus amigos.
Comprobación	Se muestra el nombre, nombre de usuario e imagen de todos los amigos del usuario registrado.

Tabla 6.5: CP05 - Ver lista de amigos.

CP06	**Buscar usuarios**
Descripción	El usuario registrado busca a cualquier usuario de la aplicación por su nombre de usuario.
Precondición	El usuario está registrado y existen usuarios cuyo nombre de usuario comienza con el término de busqueda.
Escenario	Pantalla *Profile*
Pasos	1. Presionar el icono de la persona con una lupa situado en la parte superior de la pantalla. 2. Introduce *m* en el campo *Search Users*.
Res. Esperado	Se muestra una lista con usuarios que encajan con el término de búsqueda.
Comprobación	Todos los usuarios mostrados comienzan por la subcadena introducida en el campo *Search Users*

Tabla 6.6: CP06 - Buscar usuarios

CP07	**Crear plan**
Descripción	El usuario crea un nuevo plan.
Precondición	El usuario está registrado.
Escenario	Pantalla de creación de plan.
Pasos	1. Presionar el *placeholder* de la imagen en la parte superior de la pantalla. Seleccionar una foto almacenada en el dispositivo de las mostradas en la ventana emergente. 2. Presionar el campo *Type*, seleccionar la opción *Public* de la ventana emergente y presionar el botón *Confirm*. 3. Presionar el campo *Date* y seleccionar *31/05/2024* del calendario emergente. 4. Presionar el campo *Time* y seleccionar *10:00* de la ventana emergente. 5. Introducir *Comida con amigos* en el campo *Title*. 6. Introducir *Bar Paco* en el campo *Location*. 7. Introducir *Esta es la descripción del plan* en el campo *Description*. 8. Presionar el botón de acción de la parte inferior de la pantalla.
Res. Esperado	Se crea el nuevo plan y se redirige al usuario a la pantalla de *My Plans*.
Comprobación	El plan creado aparece en la pantalla de *My Plans* mostrando la información introducida.

Tabla 6.7: CP07 - Crear plan

CP08	Eliminar plan
Descripción	El usuario elimina un plan.
Precondición	El usuario está registrado y es el anfitrión del plan.
Escenario	Pantalla de detalles del plan.
Pasos	1. Presionar el botón *Delete* de la parte inferior de la pantalla.
Res. Esperado	Se elimina el plan y se redirige al usuario a la pantalla de *My Plans*.
Comprobación	El plan ya no aparece en la pantalla *My Plans* del usuario.

Tabla 6.8: CP08 - Eliminar plan

CP09	Ver todos los planes
Descripción	El usuario visualiza la lista de los planes públicos creados por el resto de usuarios y de los planes privados creados por sus amigos.
Precondición	El usuario está registrado y hay al menos un plan creado por otro usuario en la aplicación.
Escenario	Pantalla *Profile*.
Pasos	1. Presionar el botón *Discover* de la barra de navegación situada en la parte inferior de la pantalla.
Res. Esperado	Se redirige al usuario a la pantalla *Discover* y se muestran la lista de planes públicos creados por otros usuarios y de los planes privados creados por sus amigos.
Comprobación	Ningún plan tiene como participante o anfitrión al usuario. Todos los planes privados tienen como anfitrión a un amigo del usuario.

Tabla 6.9: CP09 - Ver todos los planes

CP10	**Ver los planes creados por el usuario**
Descripción	El usuario visualiza la lista de sus planes creado.
Precondición	El usuario está registrado y ha creado al menos un plan.
Escenario	Pantalla *Profile*
Pasos	1. Presionar el botón *Discover* de la barra de navegación situada en la parte inferior de la pantalla.
Res. Esperado	Se redirige al usuario a la pantalla *My Plans* y se muestran sus planes creados.
Comprobación	Los planes mostrados en la pantallas *My Plans* tienen al usuario como anfitrión.

Tabla 6.10: CP10 - Ver los planes creados por el usuario

CP11	**Ver los planes a los que se ha unido el usuario**
Descripción	El usuario visualiza la lista de planes en los que es participante.
Precondición	El usuario está registrado y se ha unido al menos a un plan.
Escenario	Pantalla *My Plans*
Pasos	1. Presionar el botón *Guest* de la barra de navegación secundaria situada en la parte superior de la pantalla.
Res. Esperado	Se muestran los planes a los que el usuario se ha unido.
Comprobación	Todos los planes mostrados tienen al usuario como participante y ninguno lo tiene como anfitrión.

Tabla 6.11: CP11 - Ver los planes a los que se ha unido el usuario

CP12	Filtrar planes privados
Descripción	El usuario filtra la lista de planes privados creados por sus amigos.
Precondición	El usuario está registrado y sus amigos han creado al menos un plan privado.
Escenario	Pantalla *Discover*
Pasos	1. Presionar el icono de filtrar situado en la parte superior de la pantalla. 2. Seleccionar la opción *Private* del menú desplegable.
Res. Esperado	Se muestran la lista de planes privados creados por otros usuarios.
Comprobación	Los planes mostrados son todos privados, y tienen como anfitrión a un amigo del usuario.

Tabla 6.12: CP12 - Filtrar planes privados

CP13	Ver perfil de un plan sin ser miembro
Descripción	El usuario visualiza los detalles del perfil de un plan del que no es miembro.
Precondición	El usuario está registrado y se ha creado al menos un plan
Escenario	Pantalla *Discover*
Pasos	1. Presionar sobre el plan del que se desea visualizar su perfil.
Res. Esperado	Se redirige al usuario a la pantalla del perfil del plan, y se muestra la información del plan.
Comprobación	La información mostrada en el perfil del plan corresponde con el plan seleccionado por el usuario. En la parte inferior se muestra un botón para solicitar unirse al plan.

Tabla 6.13: CP13 - Ver detalles de un plan sin ser miembro

CP14	**Ver perfil de un plan siendo miembro**
Descripción	El usuario visualiza los detalles del perfil de un plan en el que participa.
Precondición	El usuario está registrado y se participa en al menos un plan
Escenario	Pantalla *My Plans*
Pasos	1. Presionar sobre el plan del que se desea visualizar su perfil. 2. Tras ser redireccionado al chat del plan, presionar en el título del plan en la parte superior de la pantalla.
Res. Esperado	Se redirige al usuario a la pantalla del perfil del plan, y se muestra la información del plan.
Comprobación	La información mostrada en el perfil del plan corresponde con el plan seleccionado por el usuario. En la parte inferior se muestra un botón para salir o eliminar el plan, en caso de que el usuario sea un invitado o el anfitrión, respectivamente.

Tabla 6.14: CP14 - Ver detalles de un plan siendo miembro

CP15	**Enviar solicitud de unión a un plan**
Descripción	El usuario envía una solicitud de unión al anfitrión del plan.
Precondición	El usuario está registrado, no es miembro del plan y no tiene ninguna solicitud pendiente para ese plan.
Escenario	Pantalla del perfil del plan.
Pasos	1. Presionar el botón *Join* situado en la parte inferior de la pantalla.
Res. Esperado	Se crea la petición y se actualiza el texto el botón *Join* a *Pending*.
Comprobación	El botón *Join* muestra el texto *Pending*

Tabla 6.15: CP15 - Enviar una solicitud de unión a un plan

CP16	Cancelar envío de solicitud de unión a un plan
Descripción	El usuario cancela una solicitud de unión previamente enviada y en estado pendiente.
Precondición	El usuario está registrado, ha enviado previamente una solicitud de unión y está en estado pendiente.
Escenario	Pantalla del perfil del plan.
Pasos	1. Presionar el botón *Pending* situado en la parte inferior de la pantalla.
Res. Esperado	Se elimina la solicitud pendiente y se actualiza el texto el botón *Pending* a *Join*.
Comprobación	El botón *Pending* muestra el texto *Join*

Tabla 6.16: CP16 - Cancelar envío una solicitud de unión a un plan pendiente

CP17	Ver solicitudes de unión pendientes
Descripción	El usuario visualiza las solicitudes de unión pendientes para un plan que ha creado.
Precondición	El usuario está registrado, ha creado un plan y tiene solicitudes pendientes.
Escenario	Pantalla *Profile*.
Pasos	1. Presionar el botón *Requests* de la barra de navegación situada en la parte inferior de la pantalla.
Res. Esperado	Se redirige al usuario a la pantalla *Requests* y se muestran las solicitudes de unión para sus planes creados.
Comprobación	En la pantalla *Requests* se muestran las peticiones de unión a planes creados por el usuario, indicando el usuario del que proviene cada petición y el plan al que hace referencia.

Tabla 6.17: CP17 - Ver solicitudes de unión pendientes

CP18	**Aceptar solicitud de unión a un plan**
Descripción	El usuario acepta una solicitud de unión a un plan que ha creado.
Precondición	El usuario está registrado, ha creado un plan y tiene solicitudes pendientes.
Escenario	Pantalla de *Requests*
Pasos	1. Presionar el botón de aceptar (con el icono de un *check-mark*) de la solicitud que desea aceptar.
Res. Esperado	La petición desaparece de *Requests* y se añade al usuario que envió la petición al plan.
Comprobación	La petición desaparece de *Requests*. En la pantalla del perfil del plan al que hace referencia la petición, se muestra al usuario que envió la petición como uno de sus participantes.

Tabla 6.18: CP18 - Aceptar solicitud de unión a un plan

CP19	**Rechazar solicitud de unión a un plan**
Descripción	El usuario cancela una solicitud de unión a un plan que ha creado.
Precondición	El usuario está registrado, ha creado un plan y tiene solicitudes pendientes.
Escenario	Pantalla de *Requests*
Pasos	1. Presionar el botón de rechazar (con el icono de una papelera) de la solicitud que desea rechazar.
Res. Esperado	La petición desaparece de *Requests*.
Comprobación	La petición desaparece de *Requests*.

Tabla 6.19: CP19 - Rechazar solicitud de unión a un plan.

CP20	Ver perfil de usuario de un solicitante de unión.
Descripción	El usuario visualiza el perfil del usuario que ha mandado una petición de unión a uno de sus planes.
Precondición	El usuario está registrado, ha creado un plan y tiene solicitudes pendientes.
Escenario	Pantalla de *Requests*
Pasos	1. Presionar el nombre de usuario de una petición de unión.
Res. Esperado	El usuario es redirigido al perfil del solicitante de unión.
Comprobación	El usuario ha sido redirigido al perfil del solicitante de unión.

Tabla 6.20: CP20 - Ver perfil de usuario de un solicitante de unión.

CP21	Enviar mensaje por el chat de un plan.
Descripción	El usuario accede al chat de un plan exclusivo para sus miembros, y envía un mensaje
Precondición	El usuario está registrado y es miembro del plan.
Escenario	Pantalla de *My Plans*
Pasos	1. Presionar sobre el plan del que se desea acceder al chat. 2. Tras ser redirigido a la pantalla del chat del plan, introduce *Esto es un mensaje* en el campo *Message*.
Res. Esperado	Se muestra una burbuja con el texto del mensaje introducido en el chat.
Comprobación	La burbuja mostrada contiene el mismo texto que el introducido en el campo *Message* y muestra la hora exacta a la que fue enviado el mensaje.

Tabla 6.21: CP21 - Enviar un mensaje por el chat de un plan.

6.3. Evaluación de los casos de prueba

La Tabla 6.22 es un resumen de los resultados intermedios obtenidos tras llevar a cabo los casos de prueba, así como medidas llevadas a cabo para solucionar los errores.

Una vez solucionados todos los errores, aplicando las medidas indicadas en dicha tabla, se vuelven a ejecutar todos los casos de prueba documentados. El resultado final es de éxito en todos los casos de prueba.

Código	Resultado	Información Adicional
CP01	Éxito	-
CP02	Éxito	-
CP03	Éxito	-
CP04	Fallo	La imagen no se actualizaba en el perfil de usuario. Se detecto un error en la creación del identificador para indexar la imagen en Firebase.
CP05	Éxito	-
CP06	Éxito	-
CP07	Éxito	-
CP08	Fallo	Si el anfitrión eliminaba el plan mientras había un usuario en la pantalla de chat, saltaba una excepción. Esto se solucionó redirigiendo al usuario a otra pantalla cuando se eliminase el plan.
CP09	Fallo	En un comienzo el usuario podía ver sus propios planes en la pantalla *Discover*. Para evitar esto, filtraron los planes de los que el usuario era miembro.
CP10	Éxito	-
CP11	Éxito	-
CP12	Éxito	-
CP13	Éxito	-
CP14	Éxito	-
CP15	Éxito	-
CP16	Éxito	-
CP17	Éxito	-
CP18	Éxito	-
CP19	Éxito	-
CP20	Éxito	-
CP21	Éxito	-

Tabla 6.22: Evaluación de los casos de prueba

6.4. Licencia

Para evitar la reproducción, distribución o modificación de cualquier parte del proyecto sin la autorización expresa del titular, se ha decidido que el proyecto esté protegido bajo la ley de derechos de autor, o **Copyright**, que se aplica automáticamente tras la creación del proyecto.

Capítulo 7

Seguimiento del proyecto

7.1. Seguimiento de los Sprints

En esta sección, se presenta un análisis detallado del progreso del proyecto, desglosado por sprints. Cada sprint comprende un conjunto de tareas, centradas principalmente en las historias de usuario, junto con otras actividades esenciales.

Para mayor claridad, se estructurarán los datos del sprint en un formato tabular, incluyendo las siguientes columnas:

- **Historia de usuario.** Indican el código de la historia de usuario relacionado con las tareas.
- **Tareas.** Enumera las actividades específicas para cada historia de usuario.
- **Tiempo estimado.** Tiempo asignado a cada historia de usuario en puntos de historia durante la planificación.
- **Tiempo empleado.** Tiempo real empleado en desarrollar la historia de usuario.
- **Estado.** Muestra el estado actual de la historia de usuario al final del sprint, categorizada como *No iniciada*, *En curso* o *Completada*.

Las tareas no relacionadas con la historia de usuario, como la documentación o la refactorización, también se registran en la tabla. Este enfoque sistemático garantiza un seguimiento exhaustivo y transparente del progreso del proyecto, sprint a sprint.

Cada historia de usuario implica cinco tareas distintas, que corresponden a varias etapas de desarrollo:

- **Análisis (AN)**: se comprueba si es necesario modificar el modelo de dominio inicial y dividir las historias de usuario en historias de usuario épicas.
- **Diseño (DS)**: actualizar los diagramas existentes con nuevos elementos del desarrollo de la historia de usuario.
- **Implementación (IM)**: codificar la funcionalidad descrita en la historia de usuario.
- **Pruebas (PR)**: Crear pruebas unitarias y de la interfaz de usuario para las nuevas funciones

7.1.1. Sprint 0: 17/11/2023 - 01/12/2023

El sprint inicial, fue distinto de los sprints posteriores tanto por su naturaleza como por su duración, con un total de **30 horas**. Este sprint se dedicó a sentar las bases del proyecto, centrándose en los aspectos fundacionales y los preparativos iniciales. Se llevaron a cabo las siguientes tareas:

- ***Product Backlog* inicial**: El proyecto comenzó con la creación del *Product Backlog* inicial. Para ello se esbozaron los objetivos principales de la aplicación, se estructuraron como épicas y se desglosaron en historias de usuario. Este paso fue crucial para establecer una hoja de ruta clara para el proyecto.
- **Plan de riesgos**: Se elaboró un plan de riesgos exhaustivo en el que se identificaban los posibles problemas y se esbozaban estrategias para mitigarlos. Este plan es fundamental para abordar de forma pro-activa los problemas que puedan afectar al progreso del proyecto.
- **Planificación presupuestaria:** Se formuló el plan presupuestario, que abarcaba todos los costes previstos y los recursos necesarios para el proyecto. Esta tarea fue esencial para mantener el control financiero y garantizar la disponibilidad de recursos durante todo el ciclo de vida del proyecto.
- **Formación en Kotlin**: Dado que Kotlin es el lenguaje de programación utilizado para el proyecto, se invirtió un tiempo considerable en la formación y el perfeccionamiento en Kotlin. Esto fue fundamental para garantizar prácticas de codificación eficientes y eficaces en la fase de desarrollo del proyecto.
- **Formación en *Jetpack Compose***: La formación en *Jetpack Compose* se llevó a cabo para aprovechar sus capacidades en la construcción de interfaces de usuario modernas para la aplicación. Esto concuerda con el objetivo de crear una interfaz intuitiva y fácil de usar.
- **Investigación de la *Clean Architecture*:** Se llevó a cabo una amplia investigación sobre *Clean Architecture* para garantizar que la arquitectura de la aplicación sea sólida, mantenible y escalable. Este enfoque arquitectónico es vital para el éxito a largo plazo y la adaptabilidad de la aplicación.

- **Formación en *Firebase*:** Se realizó una investigación de este servicio de *Google* para lograr entender todas las funcionalidades, los puntos fuertes y las limitaciones que ofrece este servicio. Este tiempo se empleó en investigar acerca de las bases de datos no relacionales basadas en documentos, como lo es *Google Firestore*

Las formaciones en Kotlin, *Jetpack Compose* y *Firebase* fueron las actividades que mas tiempo llevaron, ya que el estudiante no las conocía de antemano. Requiere tiempo, además, entender y aprovechar las sinergias que el uso en conjunto de estas tecnologías ofrece. Este sprint sentó las bases para el desarrollo del proyecto, estableciendo una base sólida sobre la que se construyeron las fases posteriores.

7.1.2. Sprint 1: 01/12/2023 - 15/12/2023

En este sprint, se ha dado inicio a la fase de desarrollo. En la Tabla 7.1, se puede observar que el tiempo empleado en las tareas ha superado las estimaciones iniciales. Esto se debe a que, al representar el primer contacto práctico del estudiante con estas tecnologías, se encontraron desafíos mayores de los previstos. No obstante, se han completado con éxito todas las historias de usuario propuestas.

Durante este sprint, se ha realizado la creación del proyecto en *Android Studio* y se ha estructurado la distribución de paquetes en las diferentes capas, tal como se detalla en la Sección 5. La implementación de un flujo de datos completo ha contribuido significativamente a un mejor entendimiento de la *Clean Arquitecture*.

7.1.3. Sprint 2: 15/12/2023 - 29/12/2023

En la **Tabla 7.2** se muestran las tareas realizadas en este Sprint. La historia **HU27** en un principio iba a implementar una búsqueda de texto más compleja, pero debido a las limitaciones de Firebase, se decidió implementar una búsqueda por nombre de usuario simple.

7.1.4. Sprint 3: 12/01/2023 - 26/01/2024

En la **Tabla 7.3** se muestra las tareas realizadas. Para este sprint se había destinado un número de horas inferior al normal, debido a responsabilidades del alumno ajenas al proyecto. Sin embargo, la historia **HU26** resultó mucho mas compleja de lo inicialmente estimado. Pese a que el uso de Firestore como base de datos ha supuesto una mayor carga de trabajo, tras la realización de este sprint se ha logrado un mayor entendimiento que facilitara su uso en sprints siguientes.

ID	Tareas	Estimado	Empleado	Estado
HU02	▪ AN ▪ DS ▪ IMP ▪ PRB	5 horas	10 horas	Finalizado
HU04	▪ AN ▪ DS ▪ IMP ▪ PRB	15 horas	20 horas	Finalizado
HU05	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	10 horas	Finalizado
Total		**30 horas**	**40 horas**	**Completado**

Tabla 7.1: Tareas realizadas en el Sprint 1

ID	Tareas	Estimado	Empleado	Estado
HU27	▪ AN ▪ DS ▪ IMP ▪ PRB	15 horas	10 horas	Finalizado
HU22	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	5 horas	Finalizado
HU28	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	10 horas	Finalizado
Total		**30 horas**	**25 horas**	**Completado**

Tabla 7.2: Tareas realizadas en el Sprint 2

ID	Tareas	Estimado	Empleado	Estado
HU25	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	10 horas	Finalizado
HU26	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	20 horas	Finalizado
Total		**20 horas**	**30 horas**	**Completado**

Tabla 7.3: Tareas realizadas en el Sprint 3

7.1.5. Sprint 4: 09/02/2024 - 23/02/2024

En la **Tabla 7.4** se muestra las tareas realizadas en este sprint. Todas las tareas se completaron alrededor del tiempo estimado para ellas. Las similitudes de **HU24** con **HU26** del sprint anterior facilitaron su implementación. La historia **HU14** presento una dificultad inesperada, al tener filtrar los planes públicos creados por el usuario de los planes públicos de otros usuarios. Esto, nuevamente, es debido a las limitaciones en las consultas a la base de datos que se pueden realizar, dado que Firestore es una base de datos NoSQL basada en documentos.

7.1.6. Sprint 5: 23/02/2024 - 08/03/2024

En este sprint se han realizado todas las historias empleando menos tiempo que el inicialmente estimado. Estas tareas se muestran en la **Tabla 7.5** El proceso de crear la interfaz gráfica con Jetpack Compose se ha agilizado, debido a un mayor entendimiento del framework por parte del alumno, y a poder reutilizar componentes con estado interno ya creados.

7.1.7. Sprint 6: 08/03/2024 - 22/03/2024

Debido a que en el anterior sprint se logró un mejor entendimiento del uso de Jetpack Compose, se decidió reducir la carga de trabajo de las historias de usuario para centrarse en

ID	Tareas	Estimado	Empleado	Estado
HU24	▪ AN ▪ DS ▪ IMP ▪ PRB	5 horas	5 horas	Finalizado
HU14	▪ AN ▪ DS ▪ IMP ▪ PRB	15 horas	18 horas	Finalizado
HU16	▪ AN ▪ DS ▪ IMP ▪ PRB	5 horas	5 horas	Finalizado
Total		**25 horas**	**28 horas**	**Completado**

Tabla 7.4: Tareas realizadas en el Sprint 4

ID	Tareas	Estimado	Empleado	Estado
HU07	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	10 horas	Finalizado
HU10	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	8 horas	Finalizado
HU17	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	5 horas	Finalizado
Total		**30 horas**	**23 horas**	**Completado**

Tabla 7.5: Tareas realizadas en el Sprint 5

ID	Tareas	Estimado	Empleado	Estado
HU08	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	7 horas	Finalizado
Refactorización de varias pantallas de la IU		20 horas	20 horas	Finalizado
Total		**30 horas**	**27 horas**	**Complet.**

Tabla 7.6: Tareas realizadas en el Sprint 6

refactorizar varias pantallas de la aplicación, obteniendo una mejor optimización y claridad del código. Se eligió la historia **HU08** (**Tabla 7.6**) dadas sus similitudes con la historia**HU26**, realizada en un sprint anterior, lo que aseguraba tener los conocimientos necesarios para completarla a tiempo. Esto se hizo para poder destinar la mayor parte del tiempo en la refactorización, mostrado en la **Tabla ??**. En total en este sprint se dedicaron **27 horas**.

7.1.8. Sprint 7: 22/03/2024 - 05/04/2024

En la **Tabla 7.7** se muestra las historias de usuario realizadas en este sprint. No se pudo completar la funcionalidad de la historia **HU18**, ya que se dedicó más tiempo del estimado a realizar **HU13** y el alumno tenía otras responsabilidades académicas. En la reunión de final de sprint, se decidió postponerla para la siguiente iteración.

7.1.9. Sprint 8: 05/04/2024 - 19/04/2024

En la tabla **Tabla 7.8** se muestra las tareas realizadas. En este sprint, se completó la historia **HU18** que se dejó sin finalizar en la iteración anterior. Se dedicó la mayor parte del tiempo a la realización de la historia **HU12**, relacionada con la eliminación de un plan. Al ser Casual una aplicación reactiva, la interfaz gráfica reacciona en tiempo real a la modificación de los datos. Esta característica planteó un escenario que no se había considerado, el de redirigir a los usuarios que estuvieran en la pantalla del plan a otra pantalla de la aplicación.

ID	Tareas	Estimado	Empleado	Estado
HU09	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	10 horas	Finalizado
HU13	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	15 horas	Finalizado
HU18	▪ AN ▪ DS ▪ IMP	10 horas	5 horas	En progreso.
Total		**30 horas**	**30 horas**	**Parcialmente completo**

Tabla 7.7: Tareas realizadas en el Sprint 7

ID	Tareas	Estimado	Empleado	Estado
HU18	▪ IMP ▪ PRB	5 horas	5 horas	Finalizado
HU12	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	15 horas	Finalizado
HU19	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	7 horas	En progreso.
Total		**25 horas**	**27 horas**	**Finalizado**

Tabla 7.8: Tareas realizadas en el Sprint 8

ID	Tareas	Estimado	Empleado	Estado
HU06	▪ AN ▪ DS ▪ IMP ▪ PRB	5 horas	2 horas	Finalizado
HU15	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	10 horas	Finalizado
Documentación de la memoria		20 horas	20 horas	Finalizado
Total		**35 horas**	**35 horas**	**Compl.**

Tabla 7.9: Tareas realizadas en el Sprint 9

7.1.10. Sprint 9: 19/04/2024 - 03/05/2023

Las tareas realizadas en este sprint se muestran en la **Tabla 7.9**. Viendo las tareas restantes en el Product Backlog, se decidió utilizar un sprint extra y repartir las historias para poder dedicar la mayor parte del tiempo a la documentación de la memoria.

7.1.11. Sprint Adicional 1: 03/05/2024 - 17/05/2024

Como se indica en el **Capítulo 2.5.2**, se ha utilizado un sprint adicional para poder completar las historias de usuario relevantes y documentar apropiadamente el proyecto. En la **Tabla 7.10** se muestran las tareas realizadas en este sprint. Como en el sprint anterior, se ha destinado la mayoría del tiempo a realizar la documentación y modificación de partes de la misma según las indicaciones de la tutora.

7.2. Resumen de la ejecución del proyecto

Debido a las limitaciones de tiempo y problemas en el desarrollo, no se han podido implementar las siguientes historias de usuario:

ID	Tareas	Estimado	Empleado	Estado
HU20	▪ AN ▪ DS ▪ IMP ▪ PRB	5 horas	5 horas	Finalizado
HU21	▪ AN ▪ DS ▪ IMP ▪ PRB	10 horas	10 horas	Finalizado
Documentación de la memoria		20 horas	25 horas	Finalizado
Total		**35 horas**	**40 horas**	**Compl.**

Tabla 7.10: Tareas realizadas en el Sprint Adicional 1

- **HU01** Como nuevo usuario, quiero registrarme en la aplicación para crear una cuenta.
- **HU03** Como usuario, quiero cambiar mi contraseña para mantener mi cuenta segura.
- **HU11** Como anfitrión de un plan, quiero tener la opción de editar la información de un plan existente, para mantener a todos los participantes informados de cambios.
- **HU22** Como anfitrión, quiero tener la capacidad de moderar y administrar el chat del plan para garantizar un entorno adecuado para todos los participantes.

7.2.1. Tiempo empleado

Acorde a lo mencionado en la Sección 2.3, el Trabajo de Fin de Grado tiene asignado 12 ECTS en el Grado de Ingeniería Informática, lo equivalente a 300 horas de trabajo del alumno. La planificación inicial estimaba una duración de 310 horas, ampliables a través de los dos sprints adicionales a 340 y 370 horas, respectivamente.

Utilizando las horas empleadas en cada sprint, indicadas en la sección anterior, y teniendo en cuenta el tiempo invertido en el sprint 0, el desarrollo de este proyecto suma un total de **340 horas**.

Concepto	Precio Unitario	Cantidad	Total
Desarrollador Android	20,51 €	260 horas	5.332,60 €
Product Owner	30,27 €	40 horas	1.210,80 €
Scrum Master	26,91 €	40 horas	1.076,40 €
Lenovo ThinkBook 14	14.50 €	6 meses	87,00 €
Google Pixel 8 Pro	13,08 €	6 meses	78,48 €
Microsoft Empresa Básico	16,80 €	6 meses	100,80 €
Figma Organization	25,00 €	6 meses	150,00 €
Astah Professional	15,00 €	6 meses	90,00 €
Subtotal			8.126,08 €
+25 % de normalización			**10.157,60 €**

Tabla 7.11: Presupuesto simulado final

7.2.2. Costes finales

Ajustando el tiempo de desarrollo de el presupuesto simulado inicial mostrado en la Sección 2.5 a la duración final del proyecto, obtenemos que el presupuesto final es de **10.157,60 €**, desglosado en la **Tabla 7.11**.

De la misma manera, para el presupuesto real, el coste ajustado sería de **319,56 €**, indicado en la **Tabla 7.12**.

Concepto	Precio Unitario	Cantidad	Total
Asus ROG Strix Scar	43,75 €	6 meses	262,50 €
Google Pixel 8	9,51 €	6 meses	57,06 €
Subtotal			**319,56 €**

Tabla 7.12: Presupuesto real final

Capítulo 8

Conclusiones

A lo largo de este proyecto, y a pesar de los momentos de dificultad y desánimo, se ha conseguido desarrollar una aplicación social que pretende ayudar a mucha gente a vivir experiencias únicas, y a compartir esas experiencias con el resto de personas.

Casual ha logrado materializarse en una aplicación funcional, pasando por todas las etapas del proceso de desarrollo siguiendo en enfoque ágil. El uso de Scrum no solo ha facilitado una organización efectiva del trabajo, sino que también ha permitido una adaptación rápida a los cambio e imprevistos. Todo esto ha permitido cumplir todos los objetivos planteados inicialmente (Sección 1.4).

Quizá más importante, es la posibilidad que me ha otorgado de poner en práctica todos los conocimientos y habilidades adquiridas en Ingeniería del Software a lo largo del Grado. Esta experiencia ha sido, sin duda, fundamental para consolidar mi formación.

8.1. Líneas de trabajo futuras

Mas allá del ámbito de este proyecto, se pretende continuar con el desarrollo de Casual para completarlo y pulirlo. Algunas de estas mejoras implican:

- **Implementar las historias de usuario restantes.** Como se ha mencionado en la Sección 7.2, no se ha podido implementar todas las historias del Product Backlog debido a las limitaciones de tiempo.
- **Utilizar todas las funcionalidades que ofrece el ecosistema Firebase.** En concreto Firebase *Messaging* y Firebase *Functions*. *Messaging* te permite enviar notificaciones *push* a los usuarios, por lo que consideramos que es un añadido imprescindible para la aplicación final. Con Firebase *Functions*, se pueden desplegar funciones en el

backend a modo de servidores temporales, que se despliegan automáticamente a demanda. Esto ayudaría a darle mayor seguridad al sistema y a reducir las operaciones costosas realizadas en el cliente.

Recientemente, Firebase ha incluido una base de datos PostgresSQL en su repertorio de servicios. Esto ayudaría sin duda a suplir los defectos en las consultas que tiene Firestore, y podría ser una añadido idóneo para obtener lo mejor de ambos mundos.

- **Desplegar la aplicación en la Google Play Store.** El objetivo de la aplicación es que pueda llegar a ser utilizada por personas reales, por lo que es necesario que esté incluida en el *marketplace* de Android.

Bibliografía

[1] Android Developers, "Kotlin flows on android," 2024, last accessed 26 April 2024. [Online]. Available: https://developer.android.com/kotlin/flow

[2] Android Developer, "Thinking in compose," 2024, last accessed 26 April 2024. [Online]. Available: https://developer.android.com/jetpack/compose/mental-model

[3] Android Developers, "Lifecycle of composables," 2024, last accessed 26 April 2024. [Online]. Available: https://developer.android.com/develop/ui/compose/lifecycle

[4] Cloud Firestore Data Model, 2024, last accessed 26 April 2024. [Online]. Available: https://firebase.google.com/docs/firestore/data-model

[5] SimpleLearn, "What is git: Features, commands, branch and workflow in git," 2023, last accessed 26 April 2024. [Online]. Available: https://www.simplilearn.com/tutorials/git-tutorial/what-is-git#git_workflow

[6] Armando Picón, "Demystifying clean architecture," 2022, last accessed 28 April 2024. [Online]. Available: https://devpicon.medium.com/demystifying-clean-architecture-1cf744a3692e

[7] Microsoft, "Model-view-viewmodel (mvvm)," last accessed 28 april 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm

[8] Unión Internacional de Telecomunicaciones (ITU), "Statistics," 2023, last accessed 12 April 2024. [Online]. Available: https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx

[9] Naciones Unidas, "Más del 75 % de la población mundial tiene un teléfono celular y más del 65 % usa el internet," 2023, last accessed 12 April 2024. [Online]. Available: https://news.un.org/es/story/2023/12/1526712

[10] Investopia (Dotdash Meredith), "Social networking companies and what they own," 2023, last accessed 17 April 2024. [Online]. Available: https://www.investopedia.com/social-networking-companies-6830677

[11] Data Reportal, "The time we spend on social media," 2024, last accessed 17 April 2024. [Online]. Available: https://datareportal.com/reports/digital-2024-deep-dive-the-time-we-spend-on-social-media#:~:text=Research%20from%20GWI%20reveals%20t

hat,per%20day%20using%20social%20platforms.&text=On%20average%2C%20that%20means%20that,attributed%20to%20social%20media%20platforms.7

[12] Sprout Social, Inc., "Everything you need to know about social media algorithms," 2023, last accessed 17 April 2024. [Online]. Available: https://sproutsocial.com/insights/social-media-algorithms/

[13] Eventbrite, last accessed 17 April 2024. [Online]. Available: https://www.eventbrite.es/

[14] Fever, last accessed 17 April 2024. [Online]. Available: https://feverup.com/es/madrid

[15] Atlassian, "¿qué es scrum?" 2024, last accessed 18 April 2024. [Online]. Available: https://www.atlassian.com/es/agile/scrum

[16] K. Beck, M. Beedle, A. van Bennekum y 14 autores más, "Manifesto for agile software development," 2001, last accessed 18 April 2024. [Online]. Available: https://agilemanifesto.org/

[17] Scrum Guides, "The 2020 scrum guide," 2020, last accessed 18 April 2024. [Online]. Available: https://scrumguides.org/scrum-guide.html

[18] Scrum.org, "What are epics and features?" 2022, last accessed 18 April 2024. [Online]. Available: https://www.scrum.org/resources/blog/what-are-epics-and-features?utm_source=google&utm_medium=adwords&utm_id=psmii&adgroup={groupid}&gad_source=1&gclid=CjwKCAjwz42xBhB9EiwA48pT77pSiQglze91ER6LL1TW121R17jhxHbaFvQTSmqqG-DfDGzd8tE0SBoC2EgQAvD_BwE/

[19] M. C. Bob Hughes, *Software Project Management, 5th edition.* McGraw-Hill Higher Education, 2005.

[20] Glassdoor, "Sueldos para el puesto de desarrollador android en españa," 2024, last accessed 18 April 2024. [Online]. Available: https://www.glassdoor.es/Sueldos/desarrollador-android-sueldos-SRCH_KO0,21.htm#:~:text=En%20Espa%C3%B1a%2C%20el%20sueldo%20medio,que%20trabajan%20de%20Desarrollador%20android.

[21] Talent.com, "Salario medio para desarrollador android en españa, 2024," 2024, last accessed 18 April 2024. [Online]. Available: https://es.talent.com/salary?job=desarrollador+android

[22] Glassdoor, "Sueldos para el puesto de product owner en españa," 2024, last accessed 18 April 2024. [Online]. Available: https://www.glassdoor.es/Sueldos/product-owner-sueldo-SRCH_KO0,13.htm

[23] Talent.com, "Salario medio para product owner en españa, 2024," 2024, last accessed 18 April 2024. [Online]. Available: https://es.talent.com/salary?job=product+owner

[24] Glassdoor, "Sueldos para scrum master en españa," 2024, last accessed 18 April 2024. [Online]. Available: https://www.glassdoor.com.ar/Sueldos/espa%C3%B1a-scrum-master-sueldo-SRCH_IL.0,6_IN219_KO7,19.htm#:~:text=El%20sueldo%20promedio%20de%20un,EUR%2013.522%20y%20EUR%2014.647.

[25] Agile Experience, "¿cuánto gana un scrum master en españa?" 2024, last accessed 18 April 2024. [Online]. Available: https://agileexperience.es/2021/06/30/cuanto-gana-un-scrum-master-en-espana/

[26] Seguridad Social, "Bases y tipos de cotización," 2024, last accessed 18 April 2024. [Online]. Available: https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537#36538

[27] Figma, last accessed 18 April 2024. [Online]. Available: https://www.figma.com/

[28] Astah Professional, last accessed 18 April 2024. [Online]. Available: https://astah.net/products/astah-professional/

[29] Android, 2024, last accessed 26 April 2024. [Online]. Available: https://www.android.com/

[30] Kotlin, 2024, last accessed 26 April 2024. [Online]. Available: https://kotlinlang.org/

[31] Android Developers, "Kotlin overview," 2024, last accessed 26 April 2024. [Online]. Available: https://developer.android.com/kotlin/overview

[32] Android Developer, "Android's kotlin-first approach," 2024, last accessed 26 April 2024. [Online]. Available: https://developer.android.com/kotlin/first

[33] Android Developers, "Kotlin coroutines on android," 2024, last accessed 26 April 2024. [Online]. Available: https://developer.android.com/kotlin/coroutines

[34] Kotlin, "Asynchronous flow," 2024, last accessed 26 April 2024. [Online]. Available: https://kotlinlang.org/docs/flow.htm

[35] Android Developers, 2024, last accessed 26 April 2024. [Online]. Available: https://developer.android.com/jetpack/compose

[36] Material Design 3, 2024, last accessed 26 April 2024. [Online]. Available: https://m3.material.io/

[37] Coil, 2024, last accessed 26 April 2024. [Online]. Available: https://coil-kt.github.io/coil/

[38] Hilt, 2024, last accessed 26 April 2024. [Online]. Available: https://developer.android.com/training/dependency-injection/hilt-android

[39] Dagger, 2024, last accessed 26 April 2024. [Online]. Available: https://dagger.dev/

[40] Firebase, 2024, last accessed 26 April 2024. [Online]. Available: https://firebase.google.com/

[41] Firebase Authentication, 2024, last accessed 26 April 2024. [Online]. Available: https://firebase.google.com/docs/auth

[42] Firebase Cloud Storage, 2024, last accessed 26 April 2024. [Online]. Available: https://firebase.google.com/docs/storage

[43] Cloud Firestore, 2024, last accessed 26 April 2024. [Online]. Available: https://firebase.google.com/docs/firestore

[44] Secure data in Cloud Firestore, 2024, last accessed 26 April 2024. [Online]. Available: https://firebase.google.com/docs/firestore/security/get-started

[45] Git, 2024, last accessed 18 April 2024. [Online]. Available: https://git-scm.com/

[46] Scott Chacon, "3.1 git branching - branches in a nutshell," 2024, last accessed 18 April 2024. [Online]. Available: https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell/

[47] GitHub, 2024, last accessed 26 April 2024. [Online]. Available: https://github.com/

[48] Overleaf, 2024, last accessed 26 April 2024. [Online]. Available: https://es.overleaf.com/

[49] Microsoft Teams, 2024, last accessed 26 April 2024. [Online]. Available: https://www.microsoft.com/es-es/microsoft-teams/group-chat-software

[50] Wikipedia, "Furps," 2024, last accessed 28 April 2024. [Online]. Available: https://en.wikipedia.org/wiki/FURPS

[51] Robert C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design, 1st Edition*, 2017.

[52] Wikipedia, "Separation of concerns," 2024, last accessed 28 April 2024. [Online]. Available: https://en.wikipedia.org/wiki/Separation_of_concerns

[53] Robert C. Martin, "The clean architecture," last accessed 28 april 2024. [Online]. Available: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

[54] Wikipedia, "Dependency inversion principle," last accessed 28 april 2024. [Online]. Available: https://en.wikipedia.org/wiki/Dependency_inversion_principle

[55] Android, "Stateflow and sharedflow," last accessed 28 april 2024. [Online]. Available: https://developer.android.com/kotlin/flow/stateflow-and-sharedflow

[56] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software, 1st Edition.* Addison-Wesley, 1994.

[57] A. Shvets, "Observer," last accessed 28 april 2024. [Online]. Available: https://refactoring.guru/design-patterns/observer

[58] ——, "Singleton," last accessed 28 april 2024. [Online]. Available: https://refactoring.guru/design-patterns/singleton

[59] Tamushi, "Pruebas de aceptación de software, ¿cuándo y por qué son necesarias?" 2022, last accessed 28 april 2024. [Online]. Available: https://www.testingit.com.mx/blog/pruebas-aceptacion-software

Apéndice A

Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio del código: `https://gitlab.inf.uva.es/mignune/CasualApp`. El acceso al repositorio está restringido a los miembros del tribunal durante el periodo de evaluación.