Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Aplicación multiplataforma para mejorar la convivencia en pisos compartidos

---

Autor:
Miguel Imaz Higuera

Tutores:
César Pablo Gutiérrez Martínez

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todos los miembros de la escuela, incluyendo profesores, secretarios y personal de limpieza, por brindarnos un entorno seguro y propicio para nuestro crecimiento como informáticos y como personas. En este lugar, hemos tenido la oportunidad de mejorar nuestras habilidades y conocimientos, y también hemos tenido el privilegio de conocer a personas que nos acompañarán a lo largo de toda nuestra vida.

También quiero agradecer a mi familia por su apoyo incondicional y por brindarme la fuerza y la motivación necesarias para superar los desafíos de esta carrera. Su respaldo ha sido fundamental en cada paso que he dado y me ha impulsado a seguir adelante en momentos difíciles.

Por último, quiero expresar mi gratitud a mis compañeros de carrera. Después de pasar tanto tiempo juntos, estudiando, trabajando y disfrutando, hemos forjado una conexión especial que trasciende lo académico. Han sido mi apoyo en los momentos buenos y también en los momentos difíciles, y juntos hemos creado recuerdos que perdurarán para siempre. Agradezco profundamente su compañía y valoro enormemente todo lo que hemos compartido. Gracias por los momentos buenos y muchas gracias por los momentos malos que al menos no hemos pasado solos.

Muchas gracias a todos por todo

Resumen

La finalidad de este proyecto es la creación de Homies, una aplicación multiplataforma fácil de utilizar que ayuda a los usuarios a mejorar su convivencia entre compañeros de vivienda. La app permite a los usuarios organizar las tareas del hogar, llevar un seguimiento de las compras y las deudas, y visualizar un historial de todas estas actividades. Además, se ha implementado una función para asignar puntos a los usuarios según su desempeño en las tareas del hogar, incentivando así una convivencia más armoniosa.

Se ha creado esta aplicación utilizando Flutter, un lenguaje multiplataforma. Se ha seguido una metodología ágil en todas las etapas del desarrollo de este software, con el fin de asegurar la calidad del producto final y garantizar el cumplimiento de todas las funcionalidades requeridas. Se ha puesto especial énfasis en la utilización de las mejores prácticas y herramientas de desarrollo, y se ha llevado a cabo una evaluación continua del software para detectar posibles mejoras y corregir errores.

Abstract

The purpose of this project is the creation of Homies, an easy-to-use multi-platform application that helps users improve their coexistence among roommates. The app allows users to organize household tasks, keep track of purchases and debts, and view a history of all these activities. Additionally, a feature has been implemented to assign points to users based on their performance in household tasks, thus incentivizing more harmonious coexistence.

This application has been created using Flutter, a cross-platform language. Agile methodology has been followed in all stages of the development of this software, in order to ensure the quality of the final product and guarantee compliance with all required functionalities. Special emphasis has been placed on the use of best development practices and tools, and continuous software evaluation has been carried out to detect potential improvements and correct errors.

Índice

Listado de figuras

Listado de tablas

CAPÍTULO 1

1. Introducción

1.1. Contexto

El presente proyecto se basa en el Trabajo de Fin de Grado de Ingeniería Informática de la Universidad de Valladolid. Al tratarse de un proyecto de software, se debe desarrollar cumpliendo una serie de fases que permitan ofrecer una aplicación de calidad. En primer lugar, se llevará a cabo la planificación y el análisis de viabilidad y riesgos. Posteriormente, se iniciará el análisis de los requisitos y, una vez definidos, se procederá a su diseño. Finalmente, se llevará a cabo la implementación y la prueba del correcto funcionamiento de la aplicación. Todo el proceso será documentado detalladamente en esta memoria.

En la actualidad, el uso de dispositivos móviles es cada vez más común, siendo un crecimiento imparable desde su invención. Según estadísticas, el 97,5% de los hogares cuentan con al menos un teléfono móvil. Estos dispositivos se utilizan cada vez más para tareas importantes y de ocio, siendo el 19,1% de las personas en España quienes utilizan su teléfono móvil por más de 4 horas al día.

La pandemia del Covid-19 y la guerra de Ucrania han generado un impacto significativo en los precios a nivel nacional e internacional. De acuerdo con el INE, los precios de las viviendas subieron un 1,7% durante el tercer trimestre de 2022. Por su parte, la OCU señala que, en el último año, los precios de alimentación han aumentado un 15,3%.

Ante esta situación, se ha creado la aplicación Homies para aprovechar la masificación del uso de dispositivos móviles y ayudar a los usuarios a afrontar las subidas de precios. La aplicación permitirá la visualización de los gastos de la casa y evitar cambios de vivienda tras disputas con los compañeros. Todo esto se realizará de manera fácil, sencilla y efectiva.

1.2. Motivación

La realización de las tareas domésticas es una actividad necesaria y constante en la vida diaria de cualquier hogar. Sin embargo, la falta de organización y planificación adecuada en el desempeño de estas actividades puede generar estrés, tensión y desavenencias entre los miembros del hogar. Por lo tanto, se hace cada vez más relevante la necesidad de una herramienta que permita una planificación eficiente y una ejecución efectiva de las tareas del hogar.

La idea de crear una aplicación para la organización de tareas del hogar surgió como trabajo de fin de grado para el autor, cuando se independizó y compartió una vivienda con otros compañeros. Fue entonces cuando se dio cuenta del desequilibrio de tareas entre compañeros de hogar, la falta de comunicación a la hora de hacer

la compra o el olvido de deudas entre compañeros. A pesar de la existencia de aplicaciones que realizan estas funcionalidades, como Tricount, que organiza deudas o Cozi Family Organizer, orientada a familias, se encontró que estas aplicaciones eran demasiado complejas y contenían funcionalidades que no eran necesarias para viviendas compartidas.

Por todo ello, el principal objetivo de desarrollar esta aplicación móvil es simplificar la vida de aquellos individuos que compartan una vivienda, ya sean compañeros de piso, familia o incluso personas que vivan solas. La aplicación se centrará en la organización de tareas del hogar y la comunicación entre los miembros del hogar para garantizar una planificación eficiente y una ejecución efectiva de las tareas, lo que resultará en una convivencia más armoniosa y menos estresante.

1.3. Objetivos

El objetivo de este proyecto es crear una aplicación multiplataforma, principalmente Android, que pueda ser utilizada por todas aquellas personas que lo deseen, tanto dentro del ámbito de la Universidad de Valladolid como fuera de él.

1.3.1. Objetivos de la aplicación

- Servir de organización de tareas para compañeros de vivienda
- Tener una lista de la compra compartida entre todos los compañeros
- Almacenar los pagos que ha realizado cada usuario y las deudas que se tienen entre ellos
- Realizar un seguimiento de las actividades que realiza cada usuario
- Reforzar el esfuerzo y compañerismo entre compañeros comparando el esfuerzo hecho por cada uno
- Conseguir una aplicación compatible con cualquier dispositivo.
- Diseño atractivo y sencillo de utilizar.

1.3.2. Objetivos de formación

- Desarrollar una aplicación multiplataforma.
- Elaboración de una planificación para el proyecto y su supervisión durante todo el proceso de desarrollo.
- Llevar a cabo todas las fases del ciclo de vida de una aplicación, que incluyen la recolección y análisis de requisitos, el diseño, la implementación y las pruebas.
- Redactar un informe que recoja toda la información pertinente de los aspectos mencionados previamente.

1.3.3. Estructura de la memoria

El documento se estructura de la siguiente manera:

Capítulo 2 Planificación: Se identifican los posibles riesgos, se establecen estrategias de mitigación, se registran las fechas de cada tarea y se examina el presupuesto correspondiente.

Capítulo 3 Análisis de requisitos: Se realiza un análisis exhaustivo de los requisitos del proyecto, con el objetivo de describir de forma detallada el sistema a desarrollar. Se incluyen diagramas de casos de uso, descripción de escenarios, requisitos funcionales y no funcionales, así como las restricciones del sistema.

Capítulo 4 Análisis: Se realiza el análisis del proyecto, creando el modelo de dominio y análisis. Estos proporcionan una representación estructurada de las entidades, relaciones y componentes clave del sistema, sentando las bases para el diseño e implementación del proyecto.

Capítulo 5 Diseño: Se aborda el diseño de la aplicación, incluyendo la arquitectura elegida, los diagramas de secuencia y los patrones arquitectónicos utilizados tanto en el código como en el diseño. Se busca definir una estructura sólida y eficiente para la aplicación, asegurando su correcto funcionamiento y mantenibilidad.

Capítulo 6 Implementación: En este capítulo se describe la implementación de la aplicación, incluyendo la tecnología utilizada, la organización del código y los aspectos relevantes. Se resumen los rasgos más importantes de la implementación.

Capítulo 7 Pruebas: Se explican las pruebas que se han realizado durante este proyecto, ya sean unitarias, de integración, casos de prueba o pruebas de usabilidad.

Capítulo 8 Resultados: En este apartado se explican los resultados que se han encontrado durante las pruebas.

Capítulo 9 Conclusiones: El capítulo de conclusiones ofrece un cierre a todo el proceso llevado a cabo en el proyecto, brindando un análisis reflexivo sobre los resultados obtenidos, las lecciones aprendidas y las recomendaciones para futuras mejoras.

CAPÍTULO 2

2. Planificación

Ya que se han establecido los objetivos de esta aplicación, se ha de desarrollar la planificación de las diferentes fases que conllevara el proyecto para poder conseguirlo organizadamente y en un tiempo determinado. En este capítulo se detalla todos los aspectos relacionados con la planificación del proyecto.

2.1. Método de trabajo del TFG

Al iniciar el proyecto, se llevó a cabo un análisis exhaustivo de diferentes metodologías disponibles, considerando enfoques tradicionales como el modelo en cascada. El modelo en cascada se caracteriza por una planificación y ejecución secuencial de las etapas del proyecto, donde cada fase se completa antes de pasar a la siguiente. Sin embargo, se determinó que este enfoque podría resultar inflexible y poco adaptativo a medida que se enfrentaran cambios y nuevas necesidades en el proyecto.

En contraste, se exploraron las metodologías ágiles que permite una mayor adaptabilidad y flexibilidad en el desarrollo del proyecto.

El marco de trabajo seleccionado es Scrum. Los sprints, el enfoque iterativo e incremental, los roles definidos y la colaboración activa de los stakeholders contribuyeron a la eficiencia y la entrega de incrementos funcionales del producto de manera continua. En este proyecto se utilizó Scrum For One en la que se personaliza el equipo de desarrollo a un único integrante, el estudiante que ademas es el Product Owner, el rol de Scrum Master seria hecho por el tutor. Esto coincide con la disponibilidad de los integrantes dado que tienen un horario laboral a parte de este proyecto.

2.2. Metodología ágil y Scrum

La metodología ágil se basa en un enfoque iterativo e incremental, donde el proyecto se divide en ciclos de trabajo cortos y enfocados llamados sprints. Cada sprint tiene una duración fija y se planifica para alcanzar objetivos específicos. Esto permite una entrega temprana y continua de incrementos funcionales del producto, brindando una mayor flexibilidad y adaptabilidad a medida que se obtiene una mayor comprensión del proyecto.

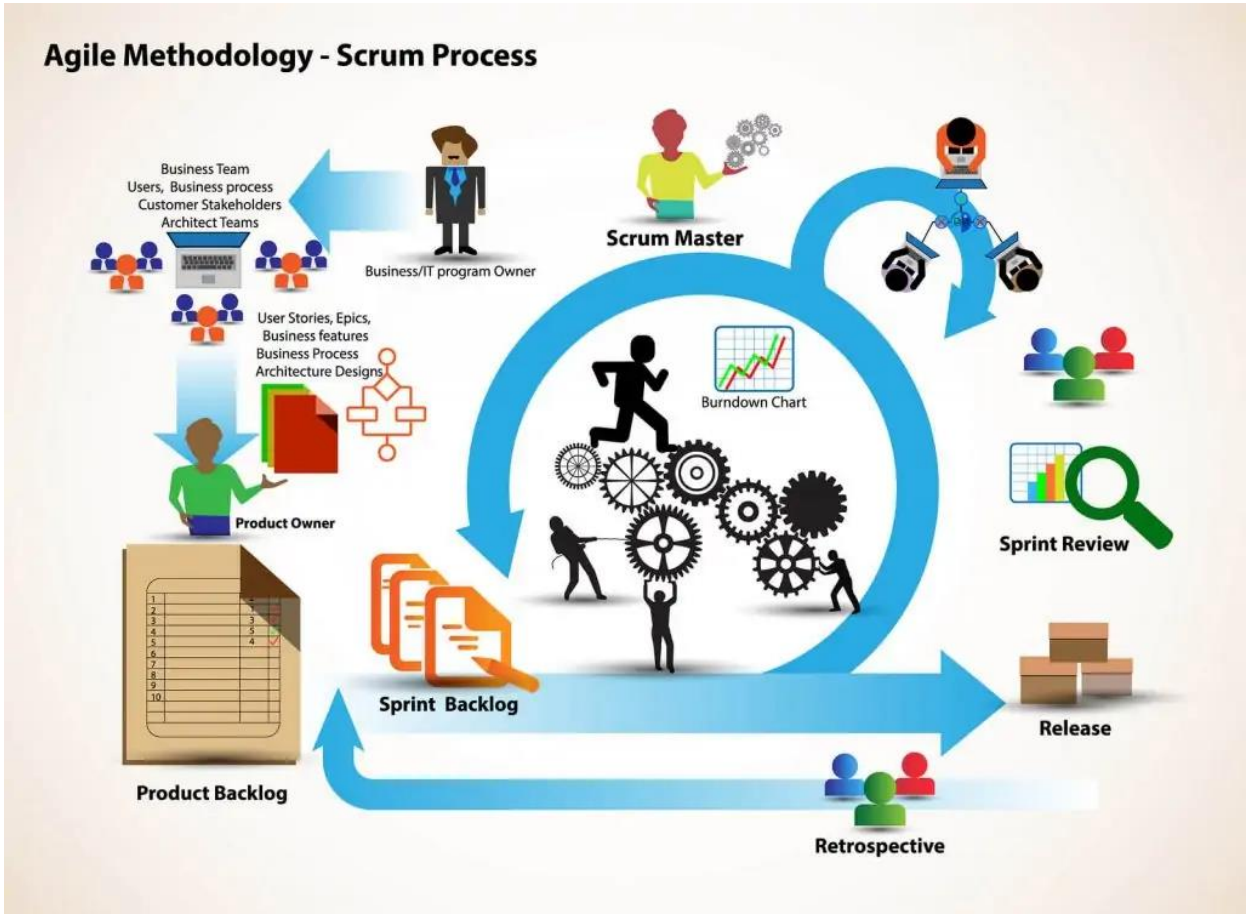


Figura 1. Diagrama de metodología ágil: Scrum

Scrum, como marco de trabajo ágil utilizado en este proyecto, se rige por una serie de principios fundamentales. Se organiza en torno a roles claramente definidos, incluyendo el Scrum Master, el Product Owner y el equipo de desarrollo.

- El Scrum Master actúa como facilitador y defensor de las prácticas ágiles dentro del equipo, asegurando que se sigan los principios de Scrum.
- El Product Owner es responsable de representar las necesidades y expectativas de los stakeholders, gestionando eficazmente el backlog del producto.
- El equipo de desarrollo es autónomo y multidisciplinario, responsable de ejecutar las tareas y entregar los incrementos del producto.

En el contexto de Scrum, los sprints son períodos de tiempo fijos y cortos en los que se trabaja para alcanzar objetivos específicos. Se inician con una reunión de planificación, donde se definen los objetivos y se seleccionan las tareas a abordar durante el sprint. Durante el sprint, el equipo de desarrollo se autoorganiza y trabaja de manera colaborativa, teniendo reuniones diarias conocidas como "daily scrum" para sincronizar actividades, revisar el progreso y abordar posibles obstáculos.

Además, Scrum pone un gran énfasis en la colaboración y la retroalimentación. Los stakeholders, tienen un rol activo en el proceso, proporcionando retroalimentación regular y contribuyendo a la toma de decisiones para asegurar la alineación del producto con los requisitos y expectativas.

Al adaptar esta metodología ágil, se tuvo en cuenta la importancia de gestionar los riesgos y garantizar el éxito del proyecto. Se realizaron actividades de planificación de riesgos para identificar posibles obstáculos y se desarrollaron estrategias de mitigación y contingencia. Además, se utilizó un Product Backlog, una lista priorizada de funcionalidades y requisitos del producto, que se actualizaba regularmente y se utilizaba para la planificación de los sprints y la asignación de tareas.

Para la implementación de la metodología ágil en este proyecto, se utilizó un tablero Kanban en la herramienta Jira. El tablero Kanban permite visualizar y gestionar el flujo de trabajo de manera efectiva, organizando las tareas en columnas como "Por hacer", "En progreso", "Comprobación" y "Completado". Esto proporciona una visión clara del estado de cada tarea y facilita la colaboración entre los miembros del equipo.

Además, en Jira se registró un historial detallado de todas las tareas realizadas, los cambios realizados y las entregas realizadas en cada sprint. Esto permite un seguimiento exhaustivo del progreso del proyecto, así como la identificación de áreas de mejora y la generación de informes útiles para evaluar el rendimiento del equipo y la calidad del producto entregado.

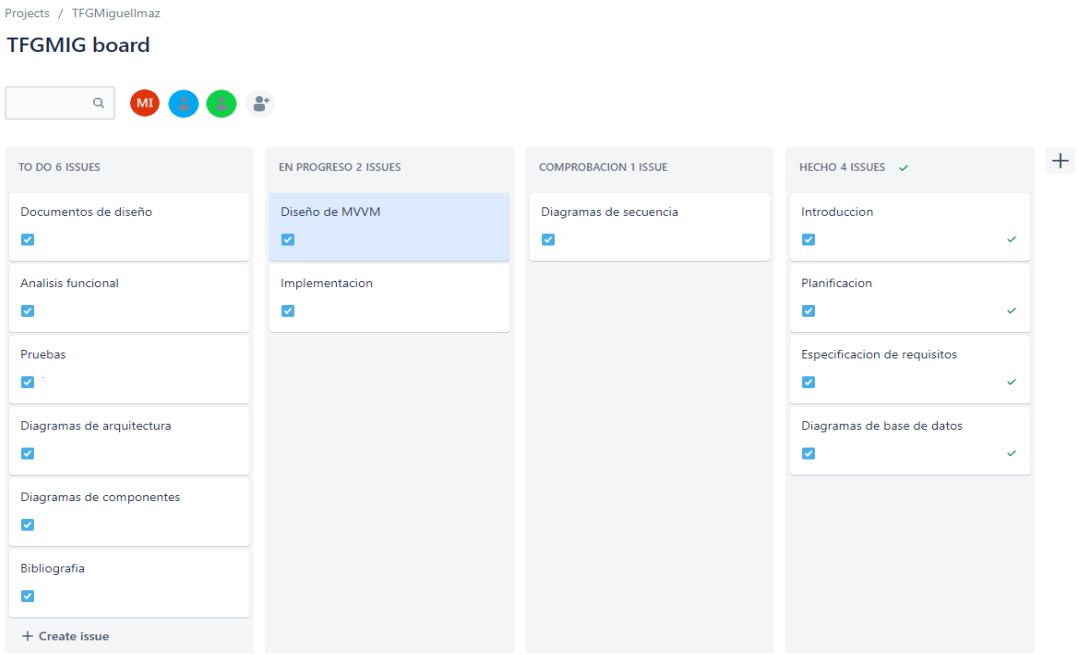


Figura 2. Tablero Kanban para la planificación del TFG.

Durante el desarrollo de este proyecto, se ha utilizado una herramienta para seguir una metodología ágil, permitiendo al autor autogestionar su tiempo de manera eficiente. Esta herramienta también ha brindado a los tutores una visión general del progreso del proyecto, verificando la correcta entrega de tareas y los archivos necesarios en cada etapa.

Se ha mencionado el uso de una metodología ágil, donde se han establecido sprints con una duración de 3 semanas. Aunque el período habitualmente es menor, se ha ajustado a las necesidades del autor, quien simultáneamente trabajaba durante el desarrollo del proyecto. Estos sprints se han definido en Jira, y para cada

uno se ha creado un sprint backlog que contiene las historias de usuario, representando los requisitos definidos y guardados inicialmente en el Product Backlog.

A lo largo del proyecto, se ha alternado el trabajo entre el desarrollo de la aplicación y la creación de la documentación. Esto se ha realizado para evitar la confrontación con tareas demasiado extensas en un solo sprint, permitiendo entregas incrementales. La última entrega, en julio de 2023, incluyó el diseño, la implementación y la documentación completos del proyecto.

En cuanto a la comunicación, se ha utilizado Teams, utilizando las cuentas oficiales de la Universidad, para facilitar la interacción entre el tutor (César Pablo Gutiérrez) y el alumno (Miguel Imaz Higuera). En algunos casos puntuales, se han utilizado correos electrónicos para coordinar reuniones en términos de tiempo y lugar.

2.3. Riesgos

Como mencionamos anteriormente, en la metodología ágil se ha llevado a cabo un análisis de riesgos para garantizar el éxito del proyecto. Este análisis se ha realizado siguiendo los principios aprendidos en la asignatura de "Planificación y Gestión de Proyectos" del cuarto año de Ingeniería Informática. Se ha utilizado una tabla con los siguientes apartados para identificar y evaluar los riesgos:

- **Descripción**: Una síntesis concisa que expone el riesgo y sus implicaciones.
- **Probabilidad**: Posibilidad de que el riesgo se materialice durante el proyecto. Puede tener tres valores: Baja, Media o Alta.
- **Impacto**: Consecuencia que puede tener en el proyecto o en los objetivos establecidos el riesgo. Puede tener tres valores: bajo, medio y alto.
- **Acciones de mitigación**: Medidas y estrategias implementadas para reducir la probabilidad de ocurrencia o minimizar el impacto negativo de un riesgo.
- **Acciones de contingencia**: medidas y estrategias establecidas de antemano para hacer frente a la materialización de un riesgo y reducir sus consecuencias negativas.
- **Estado**: Modo de actuar dependiendo del impacto negativo de cada riesgo y su probabilidad de darse, estos son los diferentes estados: aceptar, mitigar, transferir, cerrado o abierto. En la siguiente figura se representa la acción que se tiene que tomar según las características del riesgo.

NIVEL DE RIESGO		IMPACTO		
		baja	medio	alta
PROBABILIDAD	bajo	bajo	bajo	medio
	medio	bajo	medio	medio
	alto	medio	medio	alta

bajo	No hace falta hacer nada
medio	Se aplica mitigacion dado que el riesgo es medio
alta	Se ha de preparar la contigencia y mitigar lo que se pueda

Figura 3. Tabla para calcular las acciones que se han de aplicar a un riesgo

Así podemos hacer un análisis de riesgos identificando estos y creando planes para mitigar el impacto y probabilidad de esos. En las siguientes tablas están representados los riesgos que se han encontrado.

R001	**Plazos**		
Responsables	Miguel Imaz Higuera, César Pablo Gutiérrez.		
Descripción	Debido a algún problema de salud u otros compromisos del autor o de los tutores, se retrasa las entregas de las tareas planificadas lo que supondría no entregar el proyecto en este año lectivo.		
Probabilidad	Baja	**Impacto**	Alto
Acciones de Mitigación	1. Comenzar cuanto antes el proyecto. 2. Establecer objetivos cada cierto tiempo para poder realizar un seguimiento del proyecto. 3. Utilizar metodología ágil para programar estos objetivos (entregables incrementales) para poder adaptarse al cambio		
Acciones de Contingencia	1. Reducir la funcionalidad entregable. 2. Retrasar la fecha de entrega.		
Estado	Mitigándose		
Proximidad	En cada sprint	**Alcance**	Hasta la entrega del TFG

Tabla 1. Riesgo R001 – Plazos

R002	**Desarrollo**		
Responsable	Miguel Imaz Higuera		
Descripción	La complejidad del proyecto puede ocasionar retrasos en el desarrollo de la aplicación. Ya sea por desafíos técnicos, requerimientos o la necesidad de investigaciones adicionales.		
Probabilidad	Media	**Impacto**	Alto
Acciones de Mitigación	1. Planificar que tecnologías se van a utilizar y estudiar exhaustivamente las tecnologías que el autor no ha utilizado anteriormente. 2. Añadir más tiempo del que parece necesario a las tareas que han de implementarse con nuevas tecnologías, sobretodo en los primeros sprints que el autor estará más confuso.		
Acciones de Contingencia	1. Reducir complejidad de las tareas con nuevas tecnologías. 2. No implementar alguna funcionalidad. 3. Retrasar la fecha de entrega.		
Estado	Mitigándose		
Proximidad	En cada sprint.	**Alcance**	Hasta la entrega del TFG

Tabla 2. Riesgo R002 – Desarrollo

R003	**Escalabilidad**		
Responsable	Miguel Imaz Higuera		
Descripción	Existe el riesgo de que la aplicación no pueda manejar eficientemente un aumento en la carga de usuarios o datos, lo que podría llevar a problemas de rendimiento y funcionamiento.		
Probabilidad	Baja	**Impacto**	Alto
Acciones de Mitigación	1. Hacer un análisis y diseño exhaustivo de la arquitectura de la aplicación. 2. Programar en los sprints necesarios una prueba que determinen el correcto funcionamiento de la aplicación.		
Acciones de Contingencia	1. Reajuste del diseño de la aplicación, lo que conllevaría un retraso del proyecto. 2. Cambio de requisitos funcionales		
Estado	Cerrado		
Proximidad	En cada sprint	**Alcance**	Entrega TFG

Tabla 3. Riesgo R003 - Escalabilidad

R004	**Falta de habilidades técnicas**		
Responsable	Miguel Imaz Higuera		
Descripción	El autor está usando un nuevo lenguaje de programación, con una usabilidad diferente al que está acostumbrado		
Probabilidad	Media	**Impacto**	Medio
Acciones de Mitigación	1. Llevar a cabo una investigación inicial sobre las herramientas requeridas, su funcionalidad y encontrar documentación de calidad. 2. Planificar que las tareas del Kanban de desarrollo van a llevar más tiempo, sobretodo al iniciar el desarrollo.		
Acciones de Contingencia	1. Modificación total de las tecnologías usadas. 2. Buscar nuevas fuentes de información y pedir ayuda a los tutores u otros miembros de la UVa. 3. Retrasar la entrega del proyecto.		
Estado	Mitigigándose		
Proximidad	En cada sprint.	**Alcance**	Hasta la entrega del TFG.

Tabla 4. Riesgo R004 – Falta de habilidades técnicas

R005	**Problemas con la calidad del código y posibles errores**		
Responsable	Miguel Imaz Higuera		
Descripción	El código desarrollado presente problemas de calidad, errores en las funcionalidades o en los requisitos funcionales.		
Probabilidad	Baja	**Impacto**	Bajo
Acciones de Mitigación	1. Planificar una tarea antes de empezar el desarrollo, de repaso de los principios y reglas del diseño de software. 2. Planificar tareas de pruebas para cada tarea de implementación.		
Acciones de Contingencia	1. Retrasar la entrega de algún iterable.		
Estado	Aceptado		
Proximidad	En cada sprint	**Alcance**	Hasta la entrega del TFG.

Tabla 5. Riesgo R005 – Problemas con la calidad del código y posibles errores.

R006	**Problemas con la iteración aplicación usuario**		
Responsable	Miguel Imaz Higuera		
Descripción	La interfaz de la aplicación no resulta fácil de usar o no se adapta a la variedad de dispositivos de los clientes		
Probabilidad	Media	**Impacto**	Medio
Acciones de Mitigación	1. Planificar reuniones de usabilidad donde el cliente tendrá oportunidad de probar una parte de la aplicación 2. Hacer pruebas en varios usuarios y con diferentes resoluciones de pantalla 3. Organizar una prueba donde varios usuarios fuera del proyecto aportan su opinión sobre la interfaz 4. Bocetar y enseñarle estos diseños al cliente para su aprobación.		
Acciones de Contingencia	1. Alterar la interfaz de la aplicación para acomodar las diferentes condiciones y opiniones de usuarios		
Estado	Mitigándose		
Proximidad	Al finalizar una pantalla	**Alcance**	Hasta la entrega del TFG.

Tabla 6. Riesgo R006 – Problemas con la iteración aplicación usuario.

R007	**Problemas Hardware**		
Responsable	Miguel Imaz Higuera		
Descripción	Este proyecto se está desarrollando en el ordenador personal del autor, si este se estropease.		
Probabilidad	Baja	**Impacto**	Alto
Acciones de Mitigación	1. Planificar pruebas periódicas del dispositivo. 2. Almacenar el trabajo realizado en la nube, tanto la memoria como el proyecto.		
Acciones de Contingencia	1. Solicitar a la UVa un nuevo equipo o comprar otro		
Estado	Mitigándose		
Proximidad	En cada sprint	**Alcance**	Hasta la entrega del TFG.

Tabla 7. Riesgo R007 – Problemas Hardware.

R008	**Cambios de requisitos**		
Responsable	Miguel Imaz Higuera		
Descripción	El cliente cambia uno o varios de los requisitos del proyecto.		
Probabilidad	Baja	**Impacto**	Alto
Acciones de Mitigación	1. Mantener a los clientes informados sobre el diseño de las funcionalidades. 2. Planificar reuniones periódicas con los clientes para enseñarles el progreso y que aporten su opinión.		
Acciones de Contingencia	1. Hacer un análisis de cambios y adaptar la planificación para adaptar estos cambios		
Estado	Mitigándose		
Proximidad	En cada sprint	**Alcance**	Hasta la entrega del TFG.

Tabla 8. Riesgo R008 – Cambios de requisitos.

2.4. Presupuesto

Dentro del marco de este proyecto, se ha llevado a cabo una planificación minuciosa del presupuesto, el cual representa la inversión global requerida para lograr la ejecución completa del proyecto en un plazo definido, con el propósito de alcanzar resultados específicos.

El presupuesto tiene dos secciones, lo gastado en realizar las tareas y lo que cuestan los recursos necesarios para realizar estas:

2.4.1. Horas presupuestadas y calendario

Tarea	Fecha de inicio	Fecha de finalización	Participantes	Horas estimadas	Horas realizadas	Coste
Matriculación curso 2023-2023	28/06/23	28/06/23	Miguel Imaz Higuera	0	0	0,00 €
Pensar en ideas de proyecto	01/02/23	01/02/23	Miguel Imaz Higuera	7	7	96,95 €
Kick of del proyecto	07/02/23	07/02/23	Miguel Imaz Higuera y tutores	2	2	27,70 €
Estudio del tema del proyecto	08/02/23	10/02/23	Miguel Imaz Higuera	14	12	166,20 €
Realización de la introducción	10/02/23	10/02/23	Miguel Imaz Higuera	5	3	41,55 €
Repaso de los principios de diseño	12/02/23	13/02/23	Miguel Imaz Higuera	10	18	249,30 €
Análisis y especificación de requisitos	17/02/23	19/02/23	Miguel Imaz Higuera	10	8	110,80 €
Planificación del proyecto	24/02/23	26/02/23	Miguel Imaz Higuera	8	10	138,50 €
Diseño de arquitectura	01/03/23	07/03/23	Miguel Imaz Higuera	25	24	332,40 €
Diseño de casos de uso	10/03/23	16/03/23	Miguel Imaz Higuera	35	40	554,00 €
Diagramas	17/03/23	30/03/23	Miguel Imaz Higuera	25	24	332,40 €
Implementación proyecto	05/04/23	14/05/23	Miguel Imaz Higuera	100	116	1.606,60 €
Pruebas	20/04/23	03/06/23	Miguel Imaz Higuera	20	32	443,20 €
Pruebas de usuario	06/04/23	08/06/23	Miguel Imaz Higuera	15	16	221,60 €
Diseño de interfaz	10/06/23	26/06/23	Miguel Imaz Higuera	30	14	193,90 €
Reuniones de seguimiento	30/06/23	30/06/23	Miguel Imaz Higuera y tutores	2	2	27,70 €
Elaboración de la presentación	28/06/23	28/06/23	Miguel Imaz Higuera	8	6	83,10 €
			Total:	**316**	**334**	**4.625,90 €**

Tabla 9. Horas presupuestadas y calendario

La tabla que se muestra a continuación contiene información detallada sobre el calendario y los presupuestos planificados para el desarrollo de este proyecto. Al analizarla, se puede apreciar una ligera secuencialidad en las etapas iniciales, donde las tareas deben ser ejecutadas de manera consecutiva. No obstante, a medida que avanzamos hacia la etapa final, esta secuencialidad se ve modificada debido a la implementación de una metodología ágil, lo cual implica la entrega incremental de cada una de las tareas. Es importante destacar que este enfoque permite una mayor flexibilidad y adaptabilidad a medida que se avanza en el desarrollo del proyecto, optimizando así los resultados obtenidos. También se puede ver cómo, a pesar, de que las horas para este proyecto deberían ser 300 se han tanto estimado como ejecutado más.

1.1.1. Estimación y monitorización de los recursos a utilizar

La totalidad de los recursos han sido suministrados por el alumno o por la Universidad de Valladolid, por lo que en la siguiente tabla se representa el presupuesto que se hubiese gastado sin estas ayudas.

Recurso	Fecha inicio	Fecha fin	Descripción	Precio
Ordenador	01/02/23	01/07/23	Ordenador portátil para desarrollar el proyecto y la memoria	1.200,00 €
Microsoft office estudiantes	01/02/23	01/07/23	Programa para desarrollar la memoria	79,00 €
Android studio	01/02/23	01/07/23	Ide para emular un dispositivo Android	0,00 €
Visual Studio Code	01/02/23	01/07/23	Ide para desarrollar el código	0,00 €
Balsamiq	15/02/23	20/06/23	Programa para crear bocetos	196,00 €
MySQL	01/02/23	01/07/23	Programa de base de datos. Gratis hasta determinado tamaño	0,00 €
Astah Profesional	16/02/23	30/06/23	Programa para hacer diseños UML	39,96 €
Internet	01/02/23	01/07/23	Precio de internet de 300Mb/s	170,00 €
Luz	01/02/23	01/07/23	Precio de la luz durante todo el proyecto	129,13 €
			Total:	**1.814,09 €**

Tabla 10. Estimación y monitorización de recursos

2.4.2. Presupuesto total del proyecto.

En consecuencia, el presupuesto total del proyecto asciende a 6.439,99. Esta cantidad se compone de la suma de los recursos necesarios para llevar a cabo el proyecto y de la compensación por las horas de trabajo del autor. El cálculo del trabajo por horas se ha realizado en base al promedio salarial de los ingenieros informáticos en España, que es de 27.000€ al año, lo que equivale a un promedio de 13,85€ por hora.

CAPÍTULO 3

3. Análisis de requisitos

En este capítulo se documentará el análisis de requisitos realizado en el contexto de este proyecto. Para llevar a cabo dicho análisis, se detallarán los requisitos funcionales y no funcionales que son indispensables para la entrega, así como aquellos que podrían considerarse como mejoras futuras. Además, se especificarán los diferentes actores que pueden interactuar con esta aplicación, y se describirán múltiples diagramas de casos de uso que abarcarán las diversas funcionalidades del proyecto.

3.1. Actores del sistema

En la aplicación de este proyecto, todos los usuarios que la utilicen serán considerados como un único tipo de actor. No se diferenciará entre diferentes roles o categorías de usuarios, ya que todos tendrán acceso y utilizarán la aplicación de manera similar.

- **Usuario**: El actor principal de la aplicación es el usuario, quien representa a todas las personas interesadas en utilizarla. Cualquier persona que haya iniciado sesión podrá acceder a todas las funcionalidades implementadas en el proyecto.

3.2. Especificación de requisitos

Dentro de esta sección se detallan los diferentes requisitos funcionales, no funcionales y de información que se encuentran presentes en la aplicación. Estos requisitos son fundamentales para comprender y definir el alcance y las características de la aplicación.

3.2.1. Requisitos funcionales

Los requisitos funcionales son las características y funcionalidades específicas que se esperan de una aplicación para cumplir con las necesidades de los usuarios. Representan las acciones y tareas que la aplicación debe ser capaz de realizar, brindando una guía clara sobre su comportamiento y los resultados que debe producir.

En la siguiente tabla se muestran los requisitos funcionales de la aplicación. Cada uno de estos requisitos ha sido asignado con una prioridad, lo que permite enfocarse primero en aquellos que son considerados más

importantes y que brindan un mayor significado a la aplicación. Esta priorización ayuda a asegurar que los elementos clave se implementen y se centren en la entrega de valor a los usuarios.

ID	Nombre	Descripción	Prioridad
RF01	Iniciar Sesión	El sistema deberá permitir al usuario registrado que inicie sesión en su cuenta.	Alta
RF02	Registrar Usuario	El sistema deberá permitir que el usuario se registre.	Alta
RF03	Cerrar Sesión	El sistema deberá permitir al usuario cerrar la sesión de su cuenta.	Alta
RF04	Mostrar casas	El sistema deberá permitir al usuario ver las casas en las que esta apuntado	Baja
RF05	Registras una casa	El sistema deberá permitir al usuario crear una casa e invitar a sus amigos	Alta
RF06	Mostrar casa	El sistema deberá mostrar al usuario una casa con un resumen sobre la diferencia de tareas hechas por usuario, la lista de deudas que hay en la casa y las cosas por hacer	Alta
RF07	Mostrar deudas	El sistema deberá calcular las deudas que hay entre usuarios mediante los pagos que se han hecho entre ellos y mostrárselo al usuario	Alta
RF08	Mostrar Tareas	El sistema deberá mostrar un listado de tareas por hacer.	Alta
RF09	Crear una tarea	El sistema deberá permitir al usuario crear una tarea	Alta
RF10	Marcar tarea como hecha	El sistema deberá permitir al usuario marcar una tarea como hecha	Alta
RF11	Mostrar historial de tareas	El sistema deberá permitir al usuario ver las tareas que ya han sido hechas y por quien	Media
RF12	Ver una tarea	El sistema deberá permitir al usuario ver la información de una tarea	Baja
RF13	Editar una tarea	El sistema deberá permitir al usuario editar la información de una tarea	Baja
RF14	Mostrar pagos	El sistema deberá mostrar un listado de pagos por hacer.	Alta
RF15	Crear un pago	El sistema deberá permitir al usuario crear un pago	Alta
RF16	Pagar una deuda	El sistema deberá permitir al usuario saldar una deuda	Alta
RF17	Mostrar historial de pagos	El sistema deberá permitir al usuario ver los pagos que ya han sido hechas y por quien	Media
RF18	Ver un pago	El sistema deberá permitir al usuario ver la información de un pago	Baja
RF19	Editar un pago	El sistema deberá permitir al usuario editar la información de un pago	Baja
RF20	Mostrar lista de compra	El sistema deberá permitir al usuario ver la lista de la compra de una casa	Alta
RF21	Mostrar compras hechas	El sistema deberá mostrar un listado de tareas que han sido hechas	Media
RF22	Marcar una compra como hecha	El sistema deberá permitir al usuario marcar una compra como hecha	Alta
RF23	Marcar toda la lista	El sistema deberá permitir al usuario marcar toda la lista	Media

	de compras como hecha	de la compra como hecha	
RF24	Crear una compra	El sistema deberá permitir al usuario crear una cosa que comprar	Alta
RF25	Ver una compra	El sistema deberá permitir al usuario ver la información de una compra	Baja
RF26	Editar una compra	El sistema deberá permitir al usuario editar la información de una compra	Baja

Tabla 11. Requisitos funcionales

3.2.2. Requisitos de información

La siguiente tabla almacena los requisitos de información de la aplicación. Estos requisitos definen la información necesaria que debe ser almacenada y gestionada en el sistema para respaldar las reglas de negocio. Su objetivo es garantizar que los datos relevantes estén disponibles y sean administrados de manera adecuada para el correcto funcionamiento de la aplicación.

ID	Nombre	Descripción	Prioridad
RF01	Usuario	El sistema deberá almacenar información sobre cada usuario: correo electrónico, nombre, apellido contraseña, casas en las que este añadido, pagos que ha hecho y que debe y tareas que ha realizado.	Alta
RF02	Casa	El sistema deberá almacenar información sobre la vivienda: el nombre, el país, la provincia, la ciudad, la dirección, el código postal, los usuarios que tiene, los pagos, tareas y compras que se han hecho en esta	Alta
RF03	Pago	El sistema deberá almacenar información sobre los pagos que se han hecho y la información de cada pago: el título, la descripción, que usuario lo pago, la fecha, la cantidad, la repetición, la casa a la que pertenece y los usuarios que deben a causa de este pago	Alta
RF04	Tarea	El sistema deberá almacenar información sobre las tareas que se han realizado y la información de cada tarea: la descripción, fecha de creación y de realización, estado, repetición, el usuario que la ha hecho y la casa a la que pertenece	Alta
RF05	Compra	El sistema deberá almacenar información sobre las compras que se han realizado y la información de cada compra: el nombre, estado, cantidad y la casa a la que pertenece	Medio

Tabla 12. Requisitos de información

3.2.3. Requisitos no funcionales

Los requisitos no funcionales son criterios que se utilizan para evaluar la calidad de un sistema o aplicación, pero no están directamente relacionados con sus funcionalidades específicas. Estos requisitos describen atributos y características del sistema que afectan su rendimiento, seguridad, usabilidad y otras cualidades importantes.

Existen diferentes tipos de requisitos no funcionales, algunos de los cuales son:

1. Requisitos de rendimiento: Establecen los criterios relacionados con la eficiencia y la velocidad de respuesta del sistema.
2. Requisitos de usabilidad: Definen la facilidad de uso y la experiencia del usuario al interactuar con el sistema, incluyendo aspectos como la interfaz de usuario intuitiva, la accesibilidad y la consistencia en el diseño.
3. Requisitos de seguridad: Especifican los controles y mecanismos necesarios para proteger la información y los recursos del sistema.
4. Requisitos de disponibilidad: Indican el tiempo de actividad necesario para el sistema, incluyendo la capacidad de recuperación ante fallas y la tolerancia a fallos.
5. Requisitos de escalabilidad: Establecen la capacidad del sistema para adaptarse y crecer en términos de usuarios, datos y carga de trabajo, asegurando que pueda manejar eficientemente aumentos en la demanda sin degradar su rendimiento.
6. Requisitos de mantenibilidad: Se refieren a la facilidad de mantener, modificar y mejorar el sistema a lo largo del tiempo.

Se han identificado los siguientes requisitos no funcionales:

ID	Nombre	Descripción	Prioridad
RNF01	Dispositivo Android	El sistema deberá estar disponible para cualquier dispositivo Android de versión 8 o posterior	Alta
RNF02	Página Web	El sistema deberá estar disponible para cualquier explorador de internet.	Media
RNF03	Idioma	El sistema deberá estar disponible en el idioma español.	Alta
RNF04	Estándar de formato	El sistema deberá utilizar el formato UTF-8.	Alta
RNF05	Interfaz	El sistema deberá disponer de una interfaz que se adapte a los diferentes dispositivos.	Alta
RNF06	Tiempo de respuesta	El sistema debe garantizar que el tiempo de respuesta a las interacciones con el usuario no supere los 3 segundos.	Alta
RNF07	Privacidad	El sistema deberá garantizar la privacidad de todos los datos almacenados en él.	Alta
RNF08	Autentificación	El sistema tendrá una manera de autentificarse que sea segura mediante el uso de un correo y una contraseña.	Alta

RNF09	Concurrencia	El sistema deberá poder manejar varios usuarios a la vez.	Alta
RNF10	Disponibilidad	El sistema garantiza su disponibilidad todos los días del año.	Baja
RNF11	Documentación	El sistema deberá estar correctamente documentado para el futuro mantenimiento de este.	Baja

Tabla 13. Requisitos no funcionales

3.2.4. Casos de uso

Un caso de uso es una descripción de una interacción específica entre los usuarios y el sistema, que muestra cómo se utiliza el sistema para lograr un objetivo determinado.

En la siguiente tabla se detallan todos los casos de uso presentes en el sistema. Estos casos de uso representan las acciones que los clientes pueden llevar a cabo dentro de la aplicación.

ID	Nombre	Prioridad
CU-01	Identificarse	Alta
CU-02	Registrarse	Alta
CU-03	Cerrar Sesión	Baja
CU-04	Editar información usuario	Media
CU-05	Ver lista de casas	Alta
CU-06	Crear una casa	Alta
CU-07	Desasociarse de una casa	Alta
CU-08	Ver información de una casa	Media
CU-09	Editar una casa	Media
CU-10	Ver resumen de casa	Alta
CU-11	Ver lista de pagos	Alta
CU-12	Ver lista de deudas	Alta
CU-13	Saldar una deuda	Alta
CU-14	Crear un pago	Alta
CU-15	Ver información de pago	Media
CU-16	Editar pago	Media
CU-17	Ver lista de tareas	Alta
CU-18	Ver historial de tareas	Alta
CU-19	Ver información de una tarea	Media
CU-20	Editar una tarea	Media
CU-21	Marcar una tarea como hecha	Alta
CU-22	Crear una tarea	Alta
CU-23	Ver lista de la compra	Alta
CU-24	Ver historial de compra	Alta
CU-25	Marcar compra como comprado	Alta
CU-26	Comprar toda las compras	Media
CU-27	Añadir un objeto a la lista de la compra	Alta
CU-28	Editar compra	Media

Tabla 14. Casos de uso

Diagrama de casos de uso

Estos diagramas de casos de uso UML permiten una visualización clara y organizada de las interacciones entre los actores y el sistema, brindando una visión general de las funcionalidades y las relaciones entre los diferentes casos de uso. Esto facilita la comprensión y la comunicación de los requerimientos del sistema de una manera más visual y accesible. Se han separado las figuras como se ha visto necesario para que el diagrama no sea engorroso.

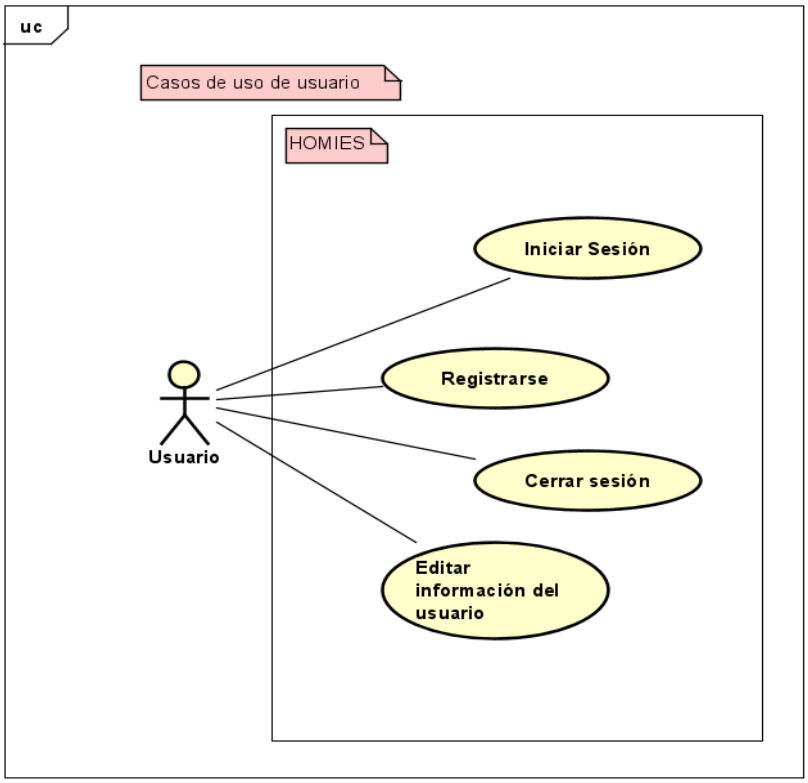


Figura 4. Diagrama Casos de Uso Usuario

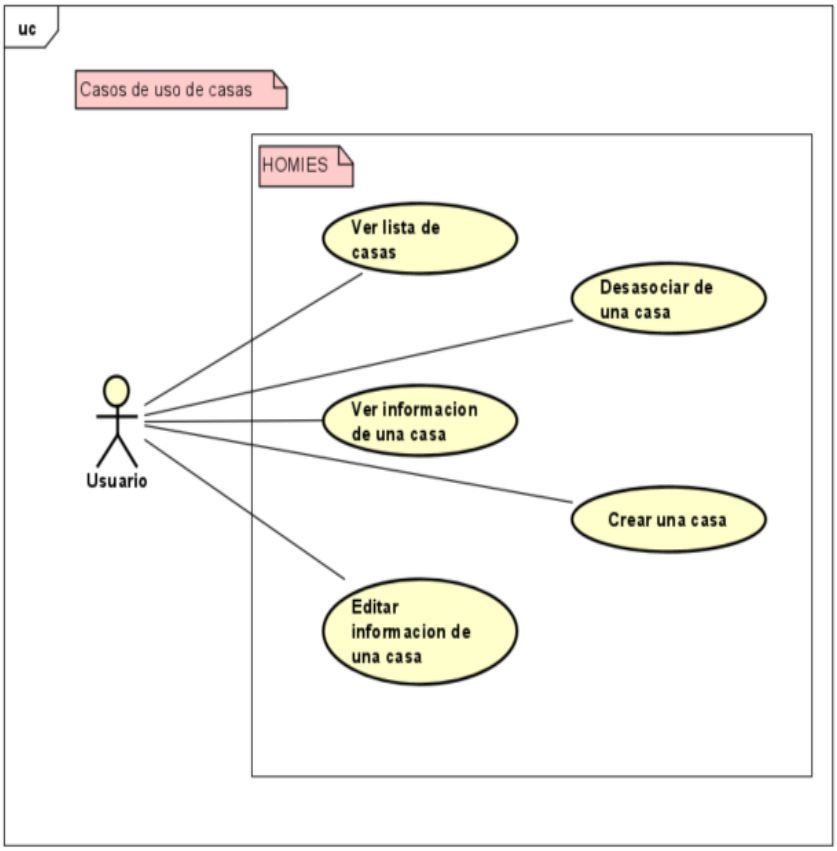


Figura 5. Diagrama Casos de Uso Casas

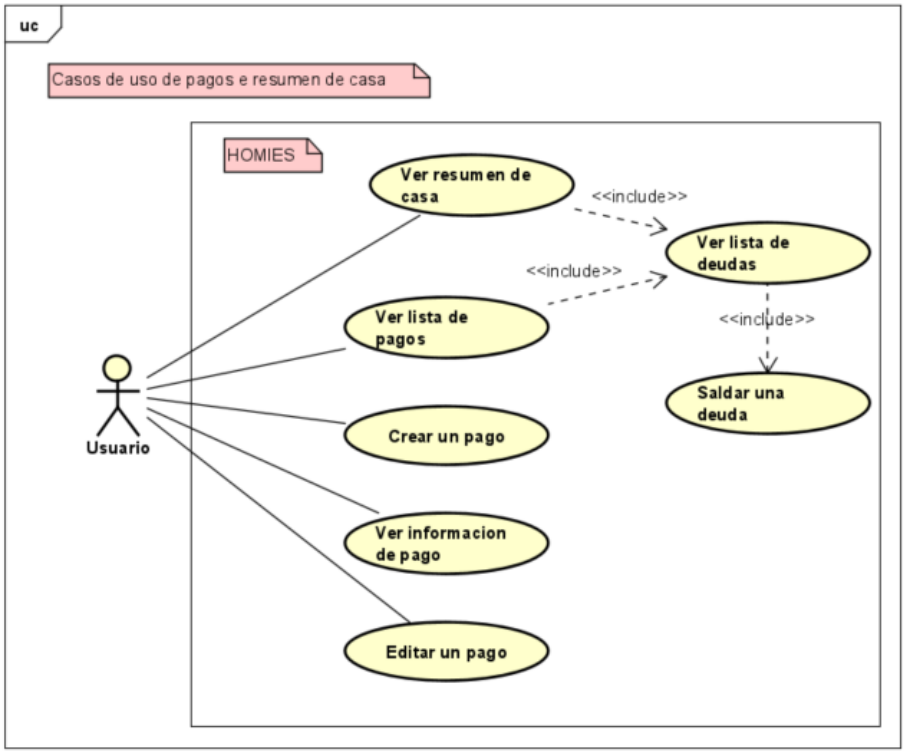


Figura 6. Diagrama Casos de Uso Tareas

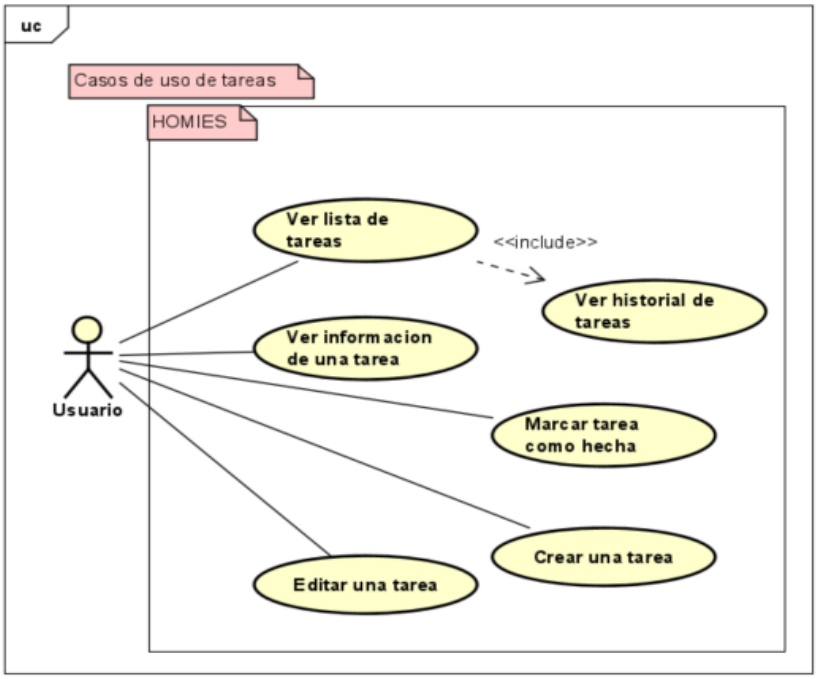


Figura 7. Diagrama Casos de Uso Tareas

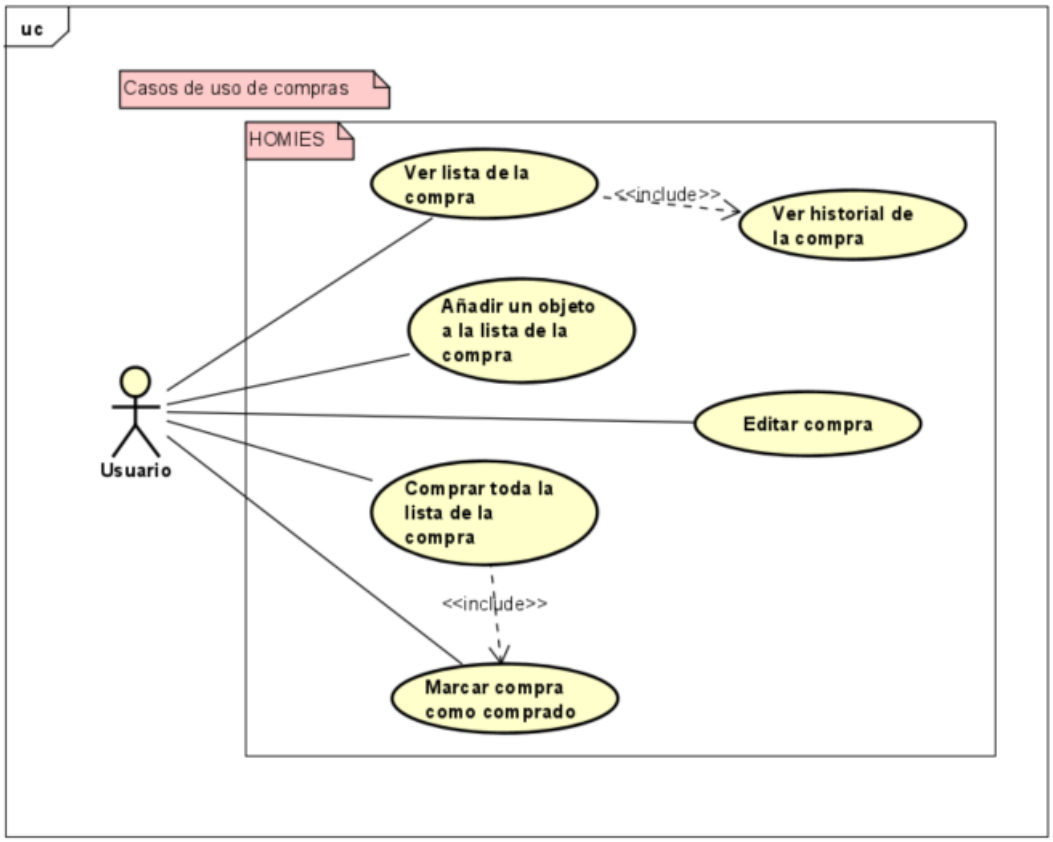


Figura 8. Diagrama Casos de Uso Compras

Descripción de los casos de uso

A continuación, se proporciona una descripción concisa de los casos de uso más destacados, dado que muchos son iguales que otros solo que con diferentes objetos. Se presentan resúmenes breves de cada caso, así como la secuencia base que representa el flujo normal del escenario, la secuencia de excepción que se produce cuando el flujo se altera, y las condiciones previas y posteriores de cada caso.

CU-01	**Identificarse**
Descripción	El actor tiene la opción de iniciar sesión en la aplicación ingresando su dirección de correo electrónico y contraseña, siempre y cuando haya completado el proceso de registro anteriormente.
Actor	Usuario.
Precondición	El actor tiene que haberse registrado previamente.
Postcondición	El actor entra en la aplicación con sus datos personales.
Flujo normal	1. El actor Usuario inicia el proceso de inicio de sesión en la aplicación. 2. El sistema solicita al usuario que ingrese la información necesaria para iniciar sesión. 3. El actor Usuario proporciona los datos solicitados en el formulario de inicio de sesión. 4. El sistema verifica la autenticidad de la combinación de correo electrónico y contraseña proporcionada por el usuario, comprobando si corresponden a un usuario registrado en la aplicación. 5. Una vez verificada la información, el sistema muestra las pantallas de navegación y funcionalidades disponibles al actor "Usuario".
Flujo alternativo	2a) Si el usuario decide cancelar el proceso de inicio de sesión y selecciona la opción de "Registrarse", el caso de uso actual no se ejecuta y se activa el caso de uso "Registrarse". 3a) Si no existe un usuario registrado con las credenciales proporcionadas, el sistema muestra un mensaje de error indicando que las credenciales son inválidas y el caso de uso actual no tiene efecto. 3b) Si el correo electrónico o la contraseña introducidos no son correctos, el sistema muestra un mensaje de error específico indicando qué dato es incorrecto y regresa al paso 2, permitiendo al usuario corregir los datos y volver a intentarlo.

Tabla 15. Descripción del caso de uso CU-01 Identificarse

CU-02	Registrarse
Descripción	Un usuario tiene la opción de registrar una cuenta en la aplicación proporcionando su dirección de correo electrónico y una contraseña. Esta cuenta permitirá al usuario acceder a todas las funcionalidades disponibles en la aplicación.
Actor	Usuario.
Precondición	Ninguna
Postcondición	El actor entra en la aplicación con sus datos personales.
Flujo normal	1. El actor Usuario inicia el proceso de inicio de sesión en la aplicación. 2. El sistema solicita al usuario que ingrese la información necesaria para iniciar sesión. 3. El actor Usuario proporciona los datos solicitados en el formulario de inicio de sesión. 4. El sistema verifica la autenticidad de la combinación de correo electrónico y contraseña proporcionada por el usuario, comprobando si corresponden a un usuario registrado en la aplicación. 5. Una vez verificada la información, el sistema muestra las funcionalidades disponibles al Usuario.
Flujo alternativo	3a) Si el actor Usuario decide cancelar el proceso de registro, el caso de uso se interrumpe sin ningún efecto. 4a) Si algún campo está vacío, el sistema muestra un mensaje de error y vuelve al paso 2. 4b) Si el email introducido por el usuario no cumple con el formato válido, el sistema muestra un mensaje de error y vuelve al paso 2. 4c) Si el email introducido ya está siendo usado por otro usuario, el sistema muestra un mensaje de error y vuelve al paso 2. 5a) Si ocurre un error durante el proceso de almacenamiento de los datos, el sistema muestra un mensaje de error y el caso de uso se interrumpe sin ningún efecto.

Tabla 16. Descripción del caso de uso CU-02 Registrarse

CU-03	Cerrar sesión
Descripción	El Usuario puede cerrar la sesión en la que se encuentra
Actor	Usuario.
Precondición	Se ha ejecutado el caso de uso “Iniciar Sesión” o “Registrarse”
Postcondición	El actor sale de la aplicación.
Flujo normal	1. El Usuario hace click en el botón de cerrar sesión 2. El sistema cierra la sesión del Usuario y muestra la pantalla de login.
Flujo alternativo	

Tabla 17. Descripción del caso de uso CU-03 Cerrar Sesión

CU-04	Editar información del usuario
Descripción	Un usuario puede modificar la información que el sistema tiene guardado sobre el
Actor	Usuario.
Precondición	Se ha ejecutado el caso de uso “Iniciar Sesión” o “Registrarse”
Postcondición	Se ha actualizado la información del Usuario
Flujo normal	1. El actor Usuario comienza el proceso de edición 2. El sistema muestra la información guardada actualmente y deja que el Usuario modifique toda la información 3. El actor Usuario cambia la información que quiere. 4. El sistema verifica que todos los campos se hayan introducido correctamente 5. Una vez verificada la información, el sistema guarda los cambios hechos muestra la información del Usuario
Flujo alternativo	3a) Si el actor Usuario decide cancelar el proceso de registro, el caso de uso se interrumpe sin ningún efecto. 4a) Si algún campo está vacío, el sistema muestra un mensaje de error y vuelve al paso 2. 5a) Si ocurre un error durante el proceso de almacenamiento de los datos, el sistema muestra un mensaje de error y el caso de uso se interrumpe sin ningún efecto.

Tabla 18. Descripción del caso de uso CU-04 Editar información del usuario

CU-05	Ver lista de casas
Descripción	El Usuario puede ver la lista de las casas en las que está invitado
Actor	Usuario.
Precondición	Se ha ejecutado el caso de uso “Iniciar Sesión” o “Registrarse”
Postcondición	
Flujo normal	1. El autor Usuario hace click en el botón de casa 2. El sistema muestra una lista de las casas a las que el Usuario está invitado, con los nombres de las casas y el número de usuarios que tiene cada una
Flujo alternativo	

Tabla 19. Descripción del caso de uso CU-05 Ver lista de casas

CU-06	Crear una casa
Descripción	Un usuario puede añadir una casa
Actor	Usuario.
Precondición	Se ha ejecutado el caso de uso “Iniciar Sesión” o “Registrarse”
Postcondición	Se ha guardado la nueva casa en el sistema correctamente
Flujo normal	1. El actor Usuario selecciona el botón de crear una casa 2. El sistema muestra un formulario vacío con los campos que se ha de rellenar para crear la nueva casa 3. El actor Usuario introduce la información requerida y pulsa guardar 4. El sistema verifica que todos los campos se hayan introducido correctamente 5. Una vez verificada la información, el sistema guarda la información de la nueva casa y muestra la lista de casas del usuario con esta añadida.
Flujo alternativo	3a) Si el actor Usuario decide cancelar el proceso de registro, el caso de uso se interrumpe sin ningún efecto. 4a) Si algún campo está vacío, el sistema muestra un mensaje de error y vuelve al paso 3. 4b) Si el campo de invitados no cumple el formato correcto, el sistema muestra un mensaje de error y vuelve al paso 3. 5a) Si ocurre un error durante el proceso de almacenamiento de los datos, el sistema muestra un mensaje de error y el caso de uso se interrumpe sin ningún efecto.

Tabla 20. Descripción del caso de uso CU-06 Crear una casa

CU-07	Desasociarse de una casa
Descripción	Un usuario puede eliminar una casa de su conjunto
Actor	Usuario.
Precondición	Se ha ejecutado el caso de uso “Iniciar Sesión” o “Registrarse” y la lista de casas no está vacía.
Postcondición	Se ha quitado la casa de la lista del Usuario
Flujo normal	1. El actor Usuario desliza una casa en la lista de casas 2. El sistema borra la casa de la lista de casas del usuario y actualiza la pantalla
Flujo alternativo	2a) Si ocurre un error durante el proceso de almacenamiento de los datos, el sistema muestra un mensaje de error y el caso de uso se interrumpe sin ningún efecto.

Tabla 21. Descripción del caso de uso CU-07 Desasociarse de una casa

CU-10	Ver resumen de una casa
Descripción	Un usuario puede ver un resumen de las tareas y compras
Actor	Usuario.
Precondición	Se ha ejecutado el caso de uso “Iniciar Sesión” o “Registrarse”
Postcondición	
Flujo normal	1. El actor Usuario selecciona una casa en la lista de casas 2. El sistema muestra una gráfica con las tareas que ha hecho cada usuario, la cantidad de tareas por hacer, la longitud de la lista de la compra y
Flujo alternativo	

Tabla 22. Descripción del caso de uso CU-10 Ver resumen de una casa

CU-11	Ver lista de pagos
Descripción	Un usuario puede ver la lista de pagos de una casa
Actor	Usuario.
Precondición	Se ha ejecutado el caso de uso “Iniciar Sesión” o “Registrarse” y ha seleccionado una casa
Postcondición	
Flujo normal	1. El actor pulsa en la barra de navegación el botón de los pagos 2. El sistema muestra una lista con los pagos que se han hecho en la casa y la lista de deudas que existen entre los usuarios
Flujo alternativo	2a) Si no hay deudas entre usuarios no se muestra nada

Tabla 23. Descripción del caso de uso CU-11 Ver lista de pagos

CU-13	Saldar una deuda
Descripción	Un usuario puede saldar una deuda con un usuario.
Actor	Usuario.
Precondición	Se ha ejecutado el caso de uso “Iniciar Sesión” o “Registrarse” y se ha ejecutado el caso de uso “Ver lista de pagos”
Postcondición	Se guarda el pago para saldar la deuda en el sistema y se ve actualizado en la lista de deudas
Flujo normal	1. El actor Usuario pulsa el botón saldar que hay al lado de cada deuda 2. El sistema comprueba que el Usuario es el que debe dinero en esta deuda 3. El sistema guarda un pago para saldar la deuda
Flujo alternativo	2a) Si el Usuario no es el que debe dinero se muestra un mensaje de error y se cancela el caso de uso

Tabla 24. Descripción del caso de uso CU-13 Saldar una deuda

CU-18	Ver historial de tareas
Descripción	Un usuario puede ver las tareas que ya han sido realizadas
Actor	Usuario.
Precondición	Se ha ejecutado el caso de uso “Iniciar Sesión” o “Registrarse” y se ha ejecutado el caso de uso “Ver lista de tareas”
Postcondición	
Flujo normal	1. Se ha ejecutado [Ver lista de tareas] 2. El actor Usuario pulsa el botón de historial en la lista de tareas 3. El sistema muestra la lista de tareas ya realizadas en la casa
Flujo alternativo	

Tabla 25. Descripción del caso de uso CU-18 Ver historial de tareas

CU-21	Marcar una tarea como hecha
Descripción	Un usuario puede marcar que ha realizado una tarea
Actor	Usuario.
Precondición	Se ha ejecutado el caso de uso “Iniciar Sesión” o “Registrarse”, se ha ejecutado el caso de uso “Ver lista de tareas” y existen tareas en la lista
Postcondición	La tarea marcada desaparecerá de la lista de tareas y estará en el historial
Flujo normal	1. El actor Usuario desliza la tarea que ha realizado 2. El sistema borra la tarea de la lista de tareas y añade esta como tarea hecha por el Usuario
Flujo alternativo	2a) Si ocurre un error durante el proceso de almacenamiento de los datos, el sistema muestra un mensaje de error y el caso de uso se interrumpe sin ningún efecto.

Tabla 26. Descripción del caso de uso CU-21 Marcar tarea como hecha

CU-26	Comprar toda la lista de la compra
Descripción	Un usuario puede marcar que ha realizado una tarea
Actor	Usuario.
Precondición	Se ha ejecutado el caso de uso “Iniciar Sesión” o “Registrarse” y “Ver lista de tareas” y existen compras en la lista
Postcondición	Todas las compras de la lista desaparecerán y se podrán ver en el historial
Flujo normal	1. El actor Usuario pulsa el botón de comprar todo 2. El sistema marca todas las tareas como hechas y lo guarda
Flujo alternativo	2a) Si ocurre un error durante el proceso de almacenamiento de los datos, el sistema muestra un mensaje de error y el caso de uso se interrumpe sin ningún efecto.

Tabla 27. Descripción del caso de uso CU-26 Comprar toda la lista de la compra

CAPÍTULO 4

4. Análisis

Una vez establecidos los requisitos del proyecto, en esta sección se presentará en detalle el análisis realizado para la realización del mismo. Se proporcionará tanto el modelo de dominio como el modelo de análisis, con el objetivo de brindar una visión más clara del ecosistema de este proyecto. Estos modelos permitirán comprender la estructura, las interacciones y las funcionalidades clave del sistema, facilitando así la planificación y el diseño adecuados para su implementación exitosa.

4.1. Modelo de dominio

El modelo de dominio es una herramienta poderosa para alinear el entendimiento entre los desarrolladores y los usuarios y garantizar que el software se ajuste adecuadamente a las necesidades del dominio. En la siguiente figura podemos apreciar el modelo de dominio de Homies la aplicación de este proyecto.

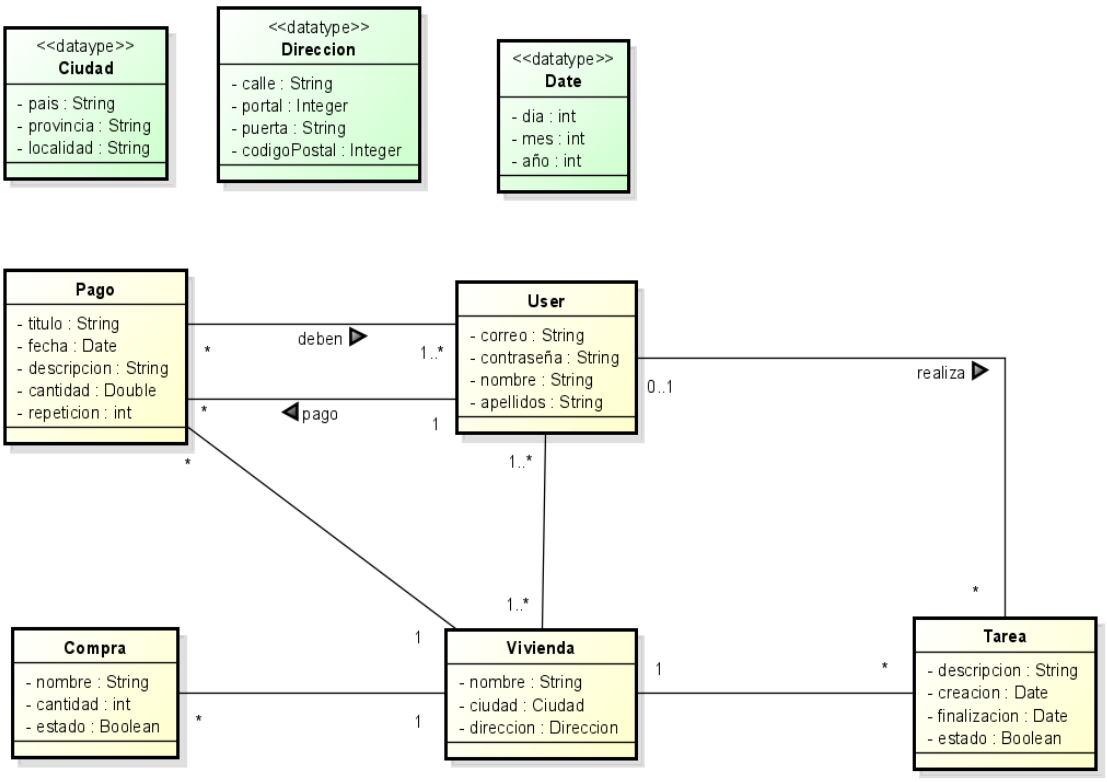


Figura 9. Modelo de dominio de la aplicación

4.2. Modelo de análisis

Los modelos de análisis desempeñan un papel fundamental en el proceso de desarrollo de software. Estos modelos representan de manera detallada las funcionalidades, procesos y comportamientos del sistema en base a los requisitos establecidos. Utilizando diversas técnicas y herramientas, se crean diagramas y artefactos visuales que permiten visualizar y comprender cómo interactúan los diferentes componentes del sistema. Esto facilita la identificación de flujos de información, validaciones necesarias y posibles mejoras en el diseño. Los modelos de análisis son una herramienta clave para asegurar que el software cumpla con los objetivos planteados y satisfaga las necesidades de los usuarios de manera efectiva.

4.2.1. Diagrama de clases de análisis

Una vez que se ha comprendido el modelo de dominio del proyecto, se asignan a las clases del proyecto una serie de operaciones para llevar a cabo la funcionalidad descrita en los casos de uso y servir como una primera aproximación al diseño. Con el objetivo de evitar la complejidad, se han incluido solo algunas de las operaciones que son relevantes para las clases, ya que solo se implementarán los casos de uso más importantes o significativos en el análisis. En la siguiente figura se muestra el modelo de análisis.

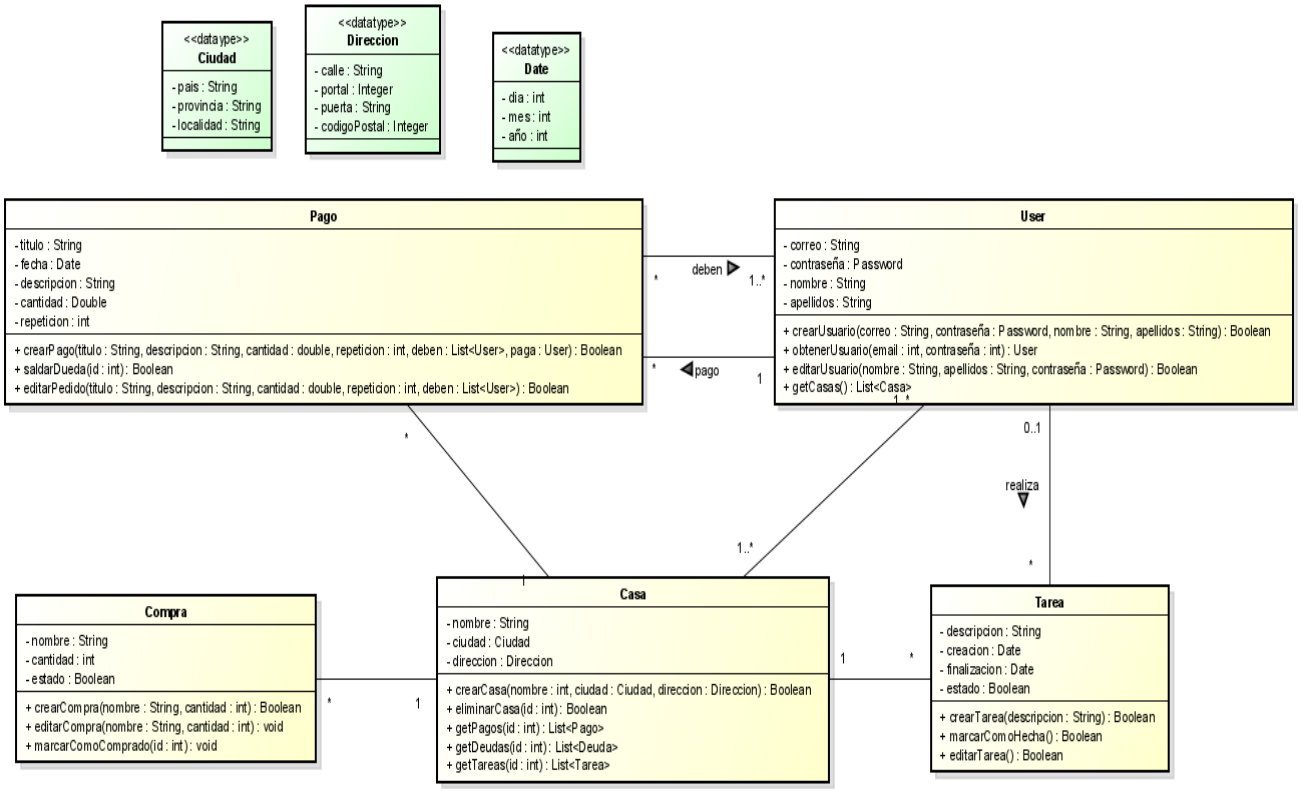


Figura 10. Diagrama del modelo de análisis

CAPÍTULO 5

5. Diseño

Este apartado tiene como objetivo proporcionar una guía para el programador sobre cómo implementar los requisitos mencionados en los capítulos anteriores. En este sentido, se abordan aspectos cruciales como la estructura lógica del sistema, los componentes que lo componen y la arquitectura utilizada para su despliegue, entre otros aspectos relevantes. El objetivo es ofrecer un modelo claro y coherente que sirva como referencia para el desarrollo del sistema y garantice su correcta implementación.

5.1. Arquitectura lógica del sistema

En esta sección se describe el patrón arquitectónico que se ha utilizado para establecer la estructura lógica del sistema. El patrón arquitectónico proporciona un enfoque y una estructura organizativa para el diseño del sistema, definiendo cómo se organizan y relacionan los componentes y módulos del sistema.

Estoy utilizando una arquitectura de microservicios en el backend de mi aplicación, lo que me permite dividir la lógica de negocio en componentes independientes y escalables. En la capa de interfaz de Flutter, sigo el patrón MVVM, donde el Modelo representa los datos y la lógica de negocio, la Vista se encarga de la presentación visual y el ViewModel actúa como un intermediario entre el Modelo y la Vista, facilitando la comunicación y actualización de los datos.

5.2. Patrón microservicios

El patrón de microservicios es una arquitectura de software que se basa en el desarrollo y despliegue de una aplicación como un conjunto de servicios pequeños e independientes, cada uno ejecutándose en su propio proceso y comunicándose a través de protocolos ligeros como HTTP/REST o mensajería.

En lugar de construir una aplicación monolítica, los microservicios dividen la funcionalidad en servicios más pequeños y especializados, lo que permite un desarrollo, implementación y escalamiento más flexibles y ágiles. Cada microservicio se enfoca en una tarea específica dentro del dominio de la aplicación y puede ser desarrollado, desplegado y escalado de manera independiente.

Las características clave del patrón de microservicios incluyen:

- Descentralización: Cada microservicio es responsable de su propia lógica de negocio y base de datos, lo que permite una mayor autonomía y desacoplamiento.
- Comunicación a través de API: Los microservicios se comunican entre sí mediante interfaces bien definidas, generalmente a través de API RESTful o eventos. Esto promueve la interoperabilidad y facilita la integración de servicios.
- Escalabilidad individual: Cada microservicio puede ser escalado de manera independiente según su demanda, lo que permite un uso eficiente de los recursos y una mejor capacidad de respuesta ante picos de carga.
- Reemplazo y actualización independientes: Los microservicios pueden ser reemplazados o actualizados sin afectar al resto de la aplicación, lo que facilita la evolución y la introducción de nuevas funcionalidades.
- Enfoque en el dominio del negocio: Cada microservicio se enfoca en una parte específica del dominio de la aplicación, lo que mejora la comprensión y mantenibilidad del código.
- Despliegue y entrega continua: La arquitectura de microservicios favorece la implementación y entrega continua, lo que permite una mayor agilidad en el desarrollo y despliegue de nuevas funcionalidades.

5.3. Patrón MVVM

El patrón MVVC (Modelo-Vista-Vista-Modelo, también conocido como Modelo-Vista-Vista-Modelo) es un patrón de diseño arquitectónico utilizado en el desarrollo de aplicaciones de software. Su objetivo principal es separar la lógica de presentación de los datos y las reglas de negocio, permitiendo una mayor modularidad y mantenibilidad del código.

El patrón MVVC consta de las siguientes partes:

- **Modelo** (Model): Representa los datos y la lógica de negocio de la aplicación. El modelo es responsable de obtener, actualizar y almacenar los datos, así como de aplicar las reglas y operaciones necesarias.
- **Vista** (View): Es la interfaz de usuario o la capa de presentación de la aplicación. La vista es responsable de mostrar los datos al usuario y de capturar las interacciones del usuario, como clics de botones o entrada de datos. La vista se mantiene lo más desacoplada posible de la lógica de negocio.
- **Vista-Modelo** (View-Model): Actúa como un intermediario entre la vista y el modelo. El vista-modelo es responsable de exponer los datos y funcionalidades del modelo a la vista, y también de manejar las interacciones del usuario en la vista. Sirve como un puente para comunicar la vista con el modelo, evitando que la vista acceda directamente al modelo.

En la siguiente figura se representa el flujo de las aplicaciones móviles que adoptan este tipo de arquitecturas. Cada vista dentro de la aplicación cuenta con un modelo de vista, el cual se encarga de comunicar los cambios que suceden en el modelo de datos. La modelo de vista se comunica con el microservicio del backend, el cual se encarga de obtener los datos de la base de datos. Esta interacción entre el modelo de vista y el microservicio del backend permite la obtención y actualización de los datos necesarios en la vista, para el funcionamiento de la aplicación móvil.

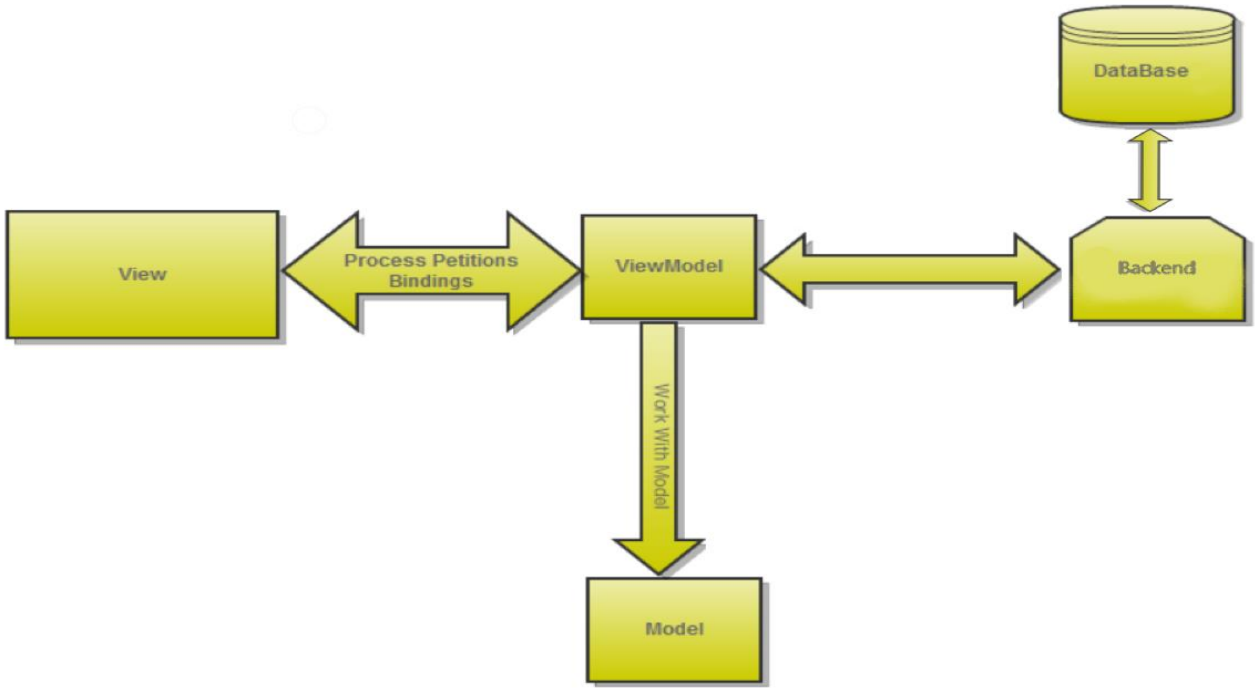
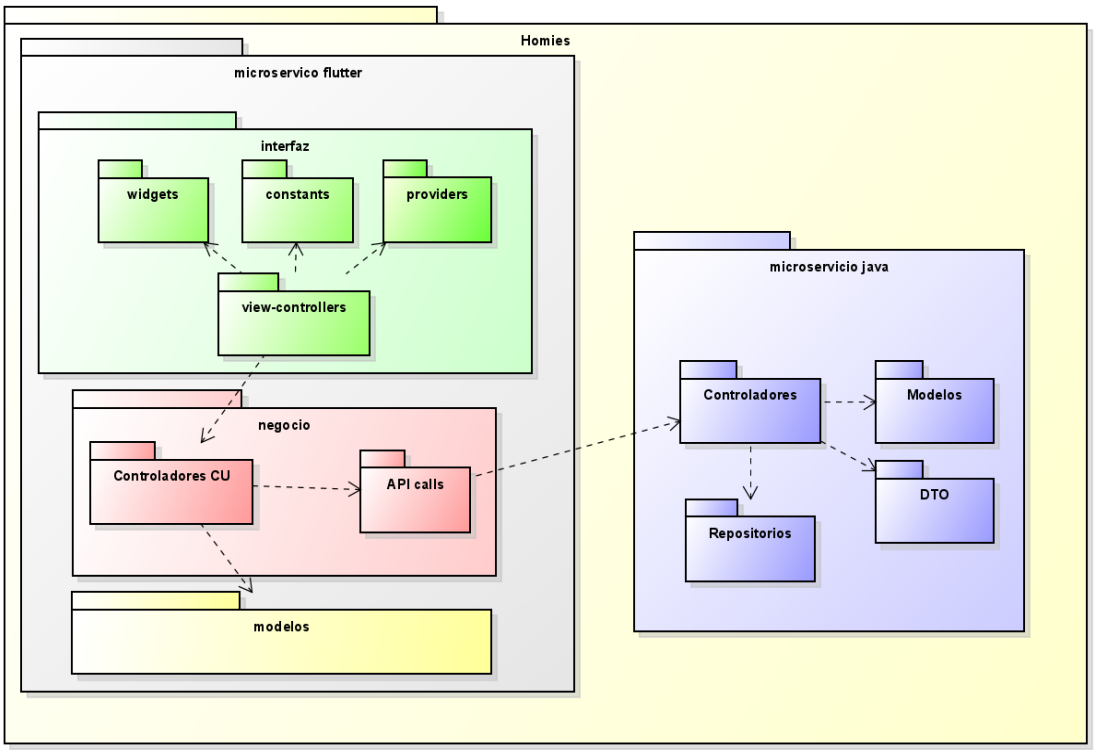


Figura 11. Flujo arquitectura de microservicios con MVVM.

Para representar la estructura del proyecto, se han creado diagramas UML en estilo Decomposition and Uses para cada uno de los componentes que conforman la aplicación. Estos diagramas permiten visualizar de manera gráfica la división del proyecto en capas y cómo se comunican entre sí, lo que facilita la comprensión de su composición. Cada diagrama muestra las relaciones y dependencias entre los diferentes componentes, brindando una representación clara de la estructura del proyecto y cómo interactúan sus elementos.

Figura 12.



Descomposition And Uses Style

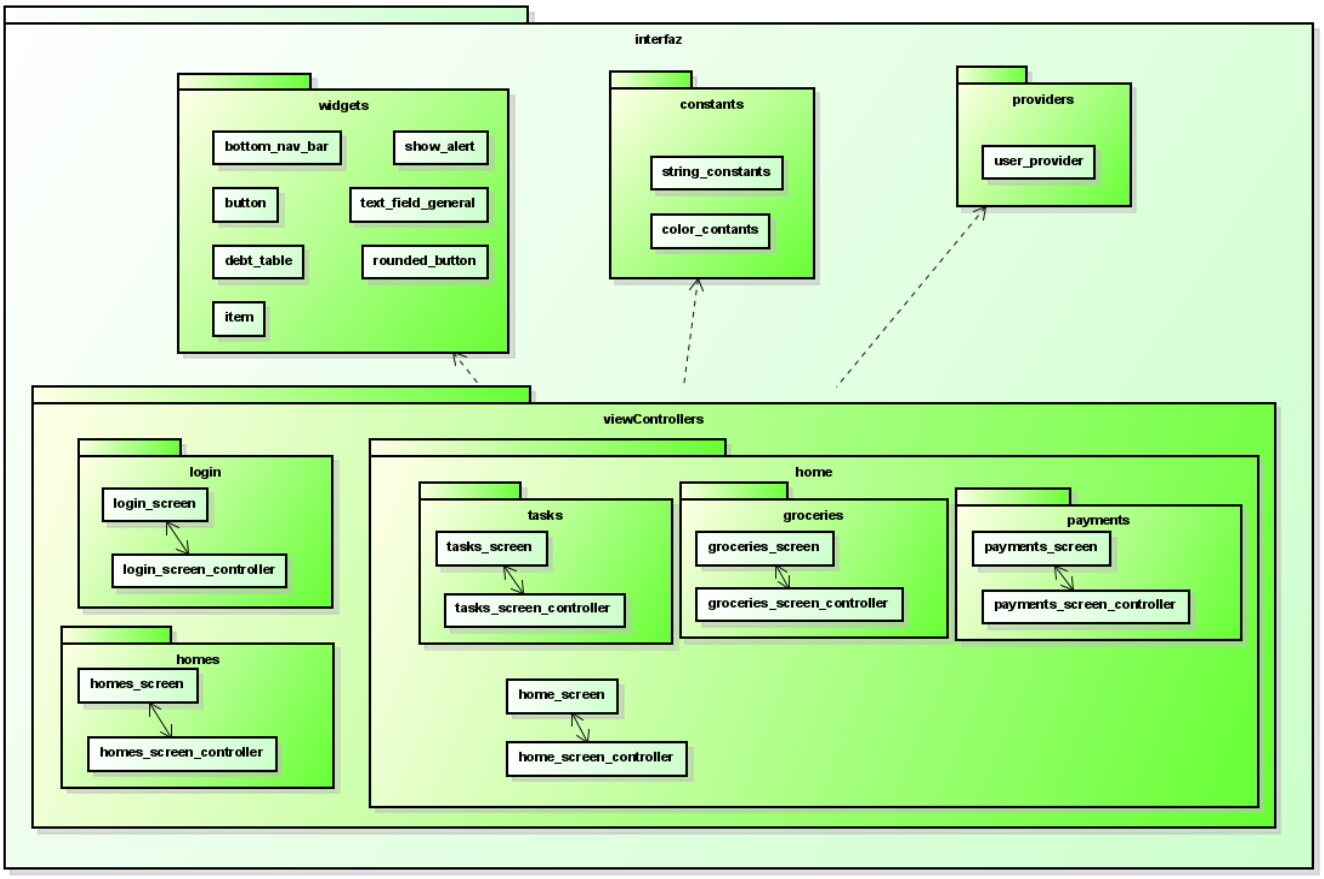


Figura 13. Descomposition And Uses Style Interfaz

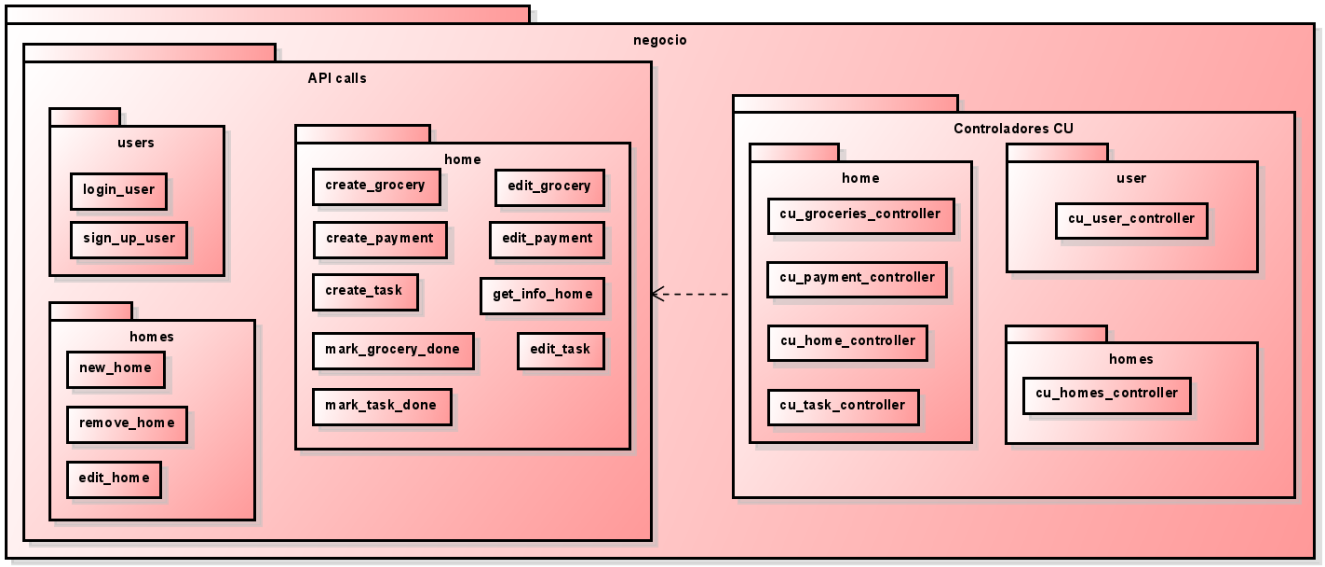


Figura 14. Descomposition And Uses Style Negocio

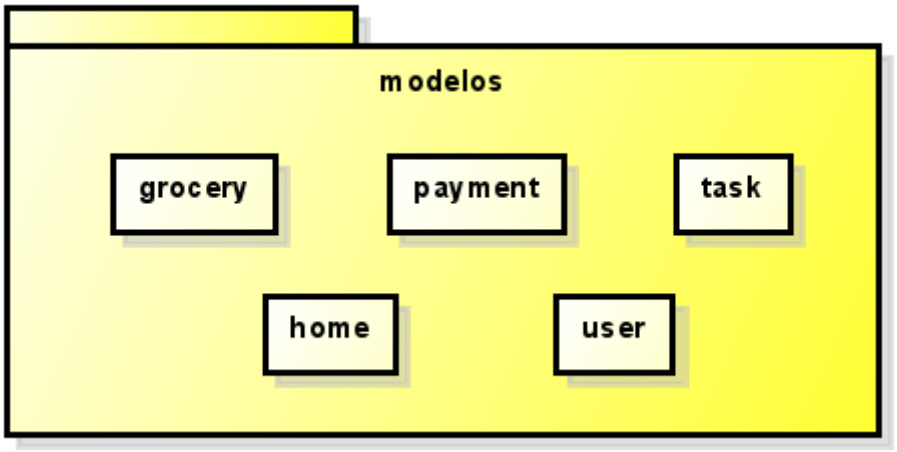


Figura 15. Descomposition And Uses Style Modelos

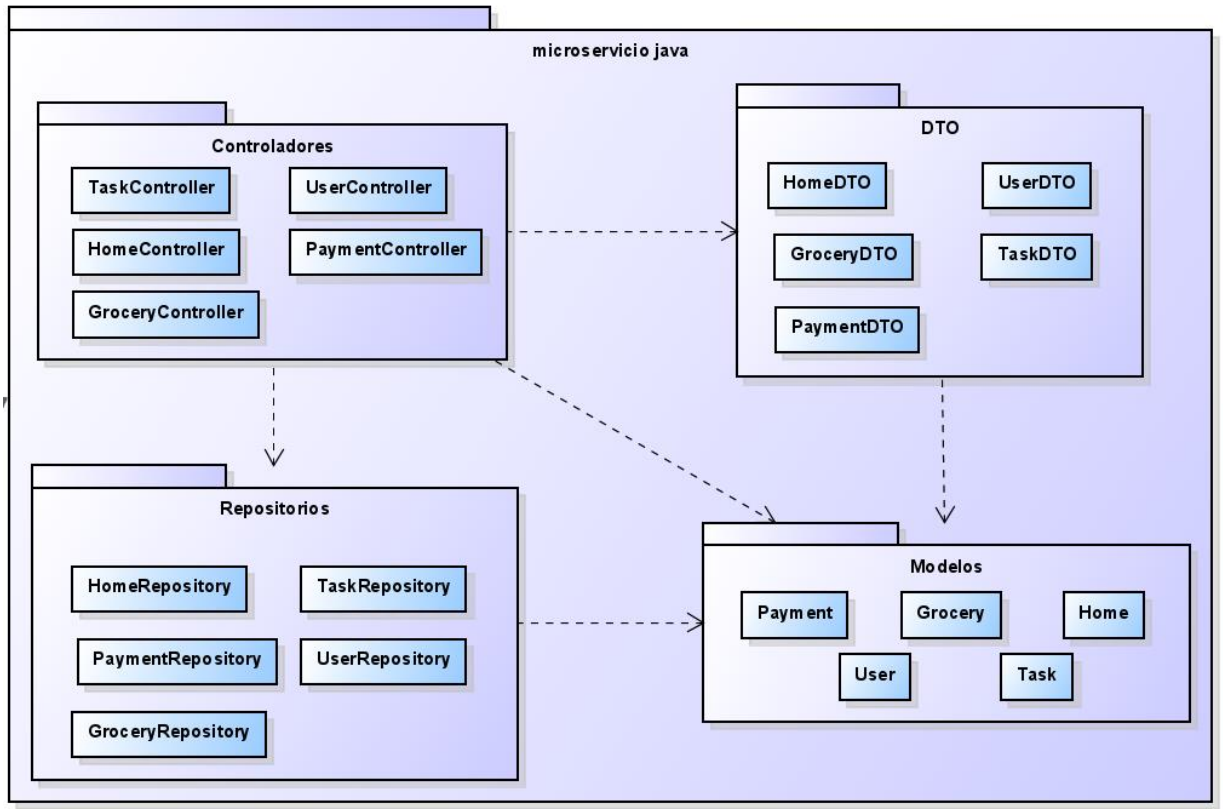


Figura 16. Descomposition And Uses Style Microservicio Java

5.4. Arquitectura física del sistema

La arquitectura física del sistema representa los elementos necesarios para desplegar la aplicación. En este caso, la aplicación utiliza tres artefactos: un servicio Flutter que contiene la interfaz de usuario, un servicio Java que se encarga de acceder a la base de datos, esto siendo el último servicio de la aplicación. Esta configuración se puede observar en la siguiente figura, que ilustra la arquitectura física del sistema.

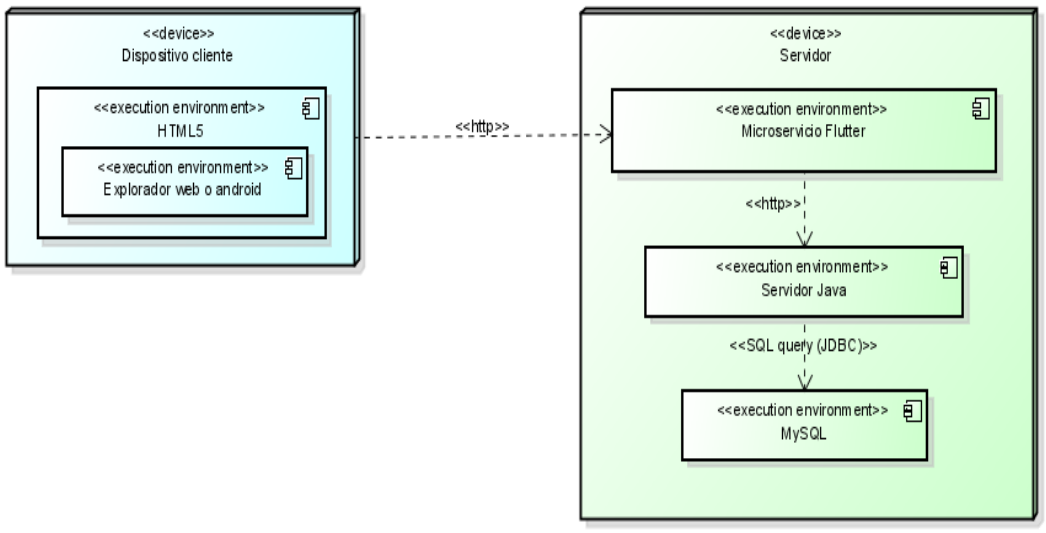


Figura 17. Diagrama de despliegue

5.5. Realización de casos de uso de diseño

Con el objetivo de explicar la interacción entre los objetos de la aplicación y para representar de manera más clara la arquitectura del sistema, se han creado los diagramas de secuencia correspondientes al caso de uso **CU-14 Crear un pago**. Dado que este caso de uso es el más representativo, se han generado los diagramas únicamente para este caso, ya que el resto de los casos de uso siguen una estructura similar y se pueden derivar a partir de este. Para evitar complicaciones, la secuencia se ha dividido en varios diagramas distintos con el fin de facilitar su comprensión.

Antes de mostrar los diagramas, se ha aplicado un esquema de colores para resaltar las diferentes partes de la arquitectura, esto ayuda a visualizar claramente las diferentes componentes de la arquitectura del sistema:

- Los elementos relacionados con la interfaz gráfica y su creación se han coloreado en verde.
- La parte del negocio, que incluye los objetos relacionados con los casos de uso, las APIs y otras clases auxiliares, se ha coloreado en rojo.
- La parte del microservicio Java, que abarca los controladores REST, los repositorios y los modelos de clase, se ha coloreado en azul.

El primer diagrama de secuencia, aunque no forma parte directamente del diagrama de secuencia del caso de uso a desarrollar, resulta muy útil, ya que muestra cómo se crean todos los objetos que serán utilizados en dicho caso de uso. Esta representación es importante ya que, sin este, los elementos utilizados podrían parecer no relacionados entre sí, lo que dificultaría la comprensión del flujo de ejecución del caso de uso.

En él, se puede observar cómo al iniciarse la pantalla de pagos se crea su controlador con el identificador de la casa, que ha sido pasado al cambiar de pantalla, y el objeto *payment_screen*. En este caso de uso en particular, nos interesa también la creación del botón de añadir pedido en la pantalla.

Durante la creación del controlador de la pantalla, se obtienen los controladores singleton del caso de uso y el director de interfaz. Además, se crean varias variables que son útiles para recopilar los datos introducidos por el usuario al crear el pago. Este diagrama de secuencia inicial es fundamental para comprender cómo se establecen los objetos y las variables necesarios para el caso de uso en cuestión.

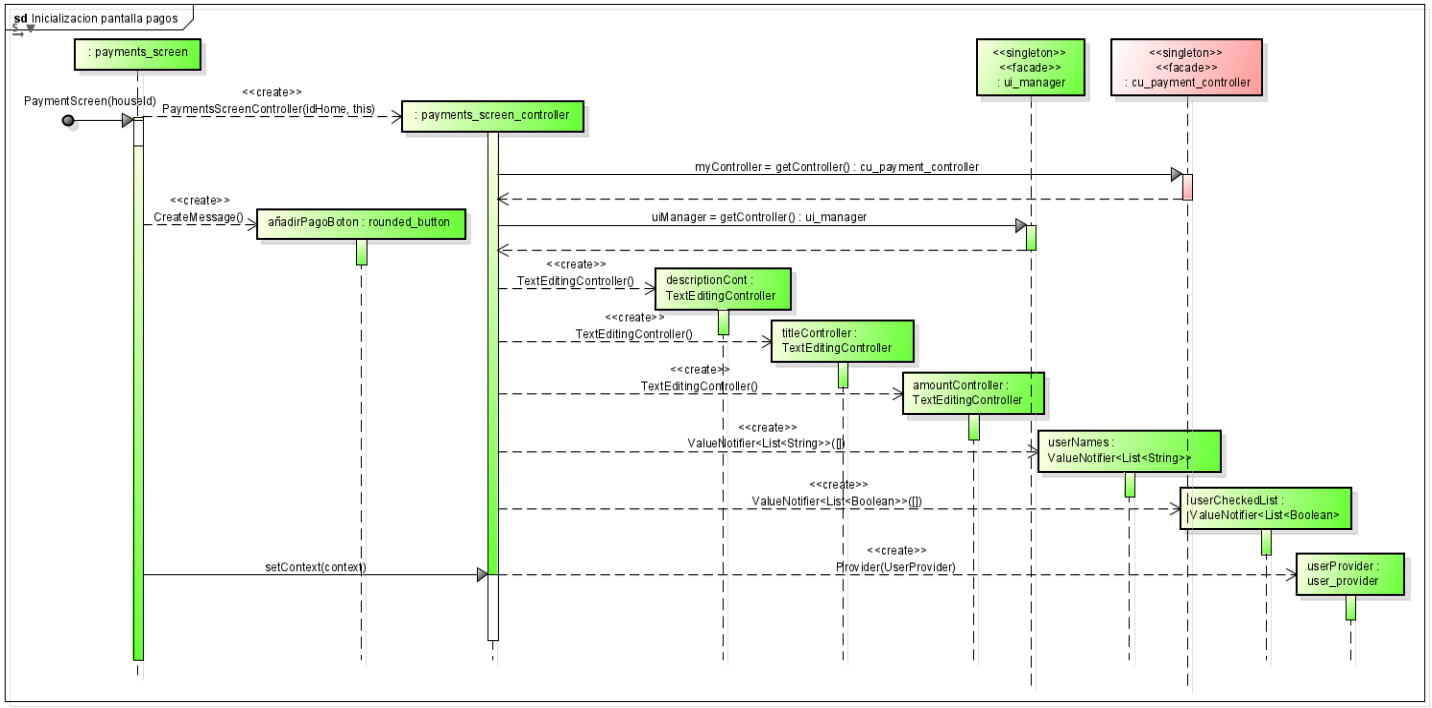


Figura 18. Diagrama de secuencia Inicialización pantalla de pagos

El diagrama siguiente representa la secuencia de creación del formulario de nuevo pedido. Este formulario también se utilizará para visualizar y editar los pedidos existentes. Para este propósito, el método showPaymentsDetailDialog recibe un argumento que corresponde al ID del pedido. En el ejemplo dado, se proporciona un valor de cero, el cual no es un ID válido, por lo que es el marcador de nuevo pedido.

En el diagrama, se puede apreciar la creación de los campos *text_field_general* para cada campo que debe ser introducido en el formulario. Además, se generan *checkboxes* individuales para cada usuario de la casa, permitiendo marcar aquellos usuarios que deben estar asociados al pedido.

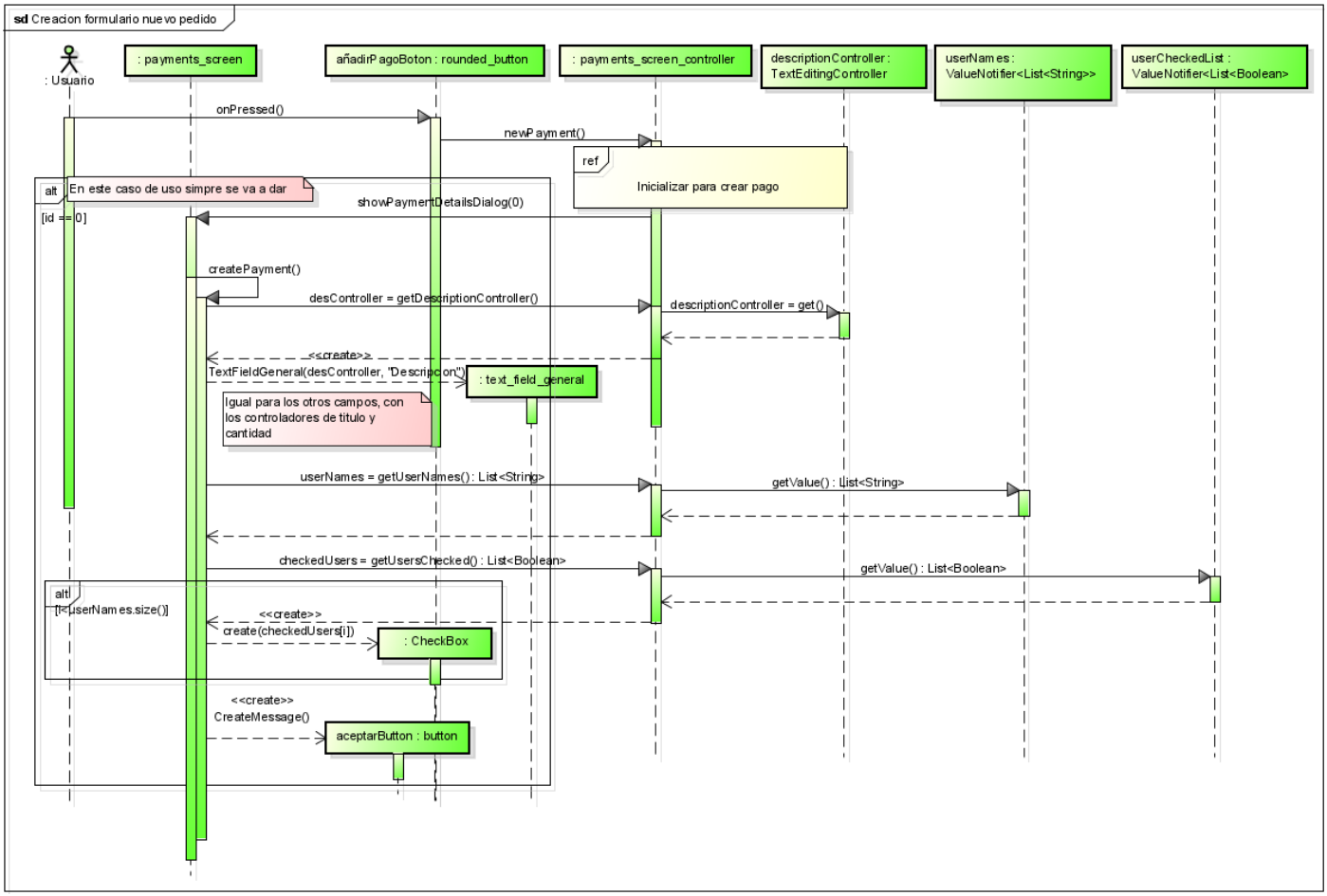


Figura 19. Diagrama de secuencia Creación del formulario de pago

En el siguiente diagrama, se muestra la secuencia de inicialización de los valores utilizados en la creación de un pago, con el propósito de evitar errores. Además, se obtienen los usuarios de la casa para poder asignar sus nombres a las *checkboxes*, permitiendo al usuario seleccionar quiénes deben realizar el pago en cuestión.

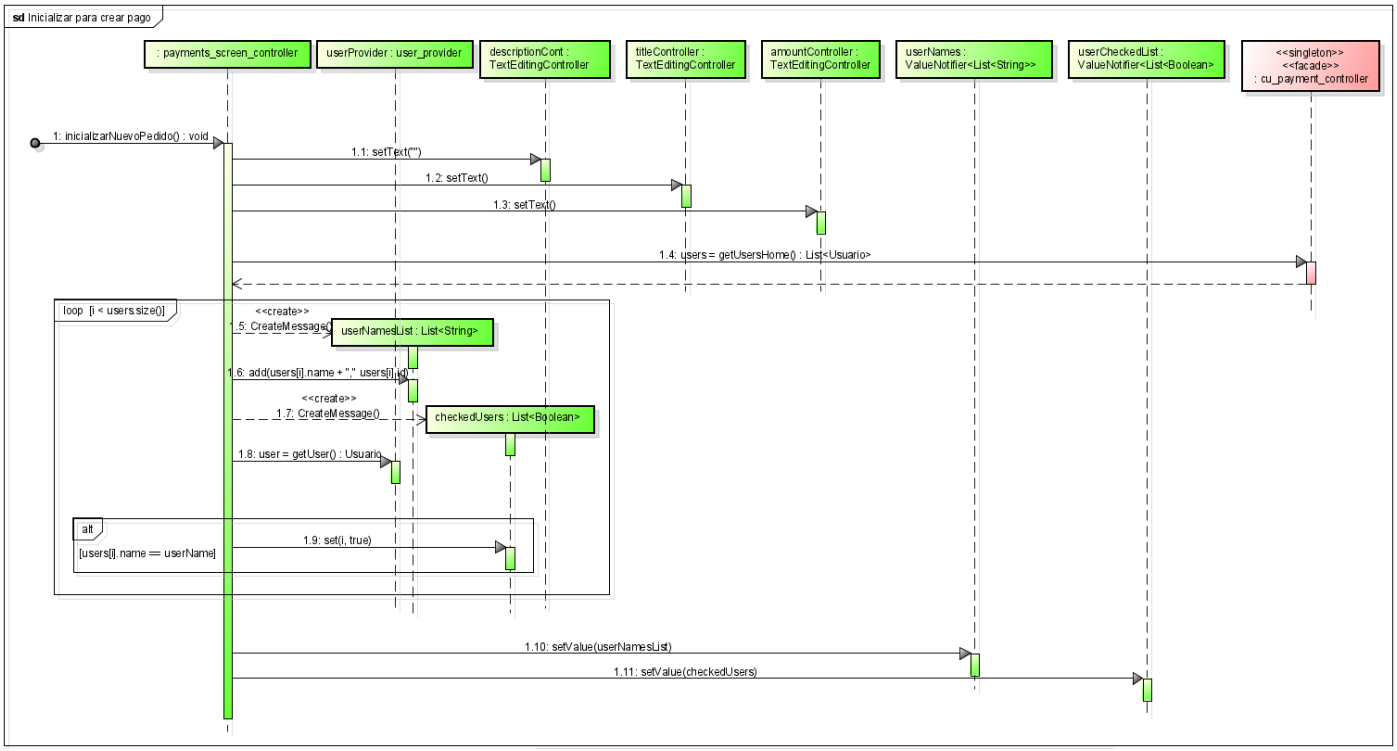


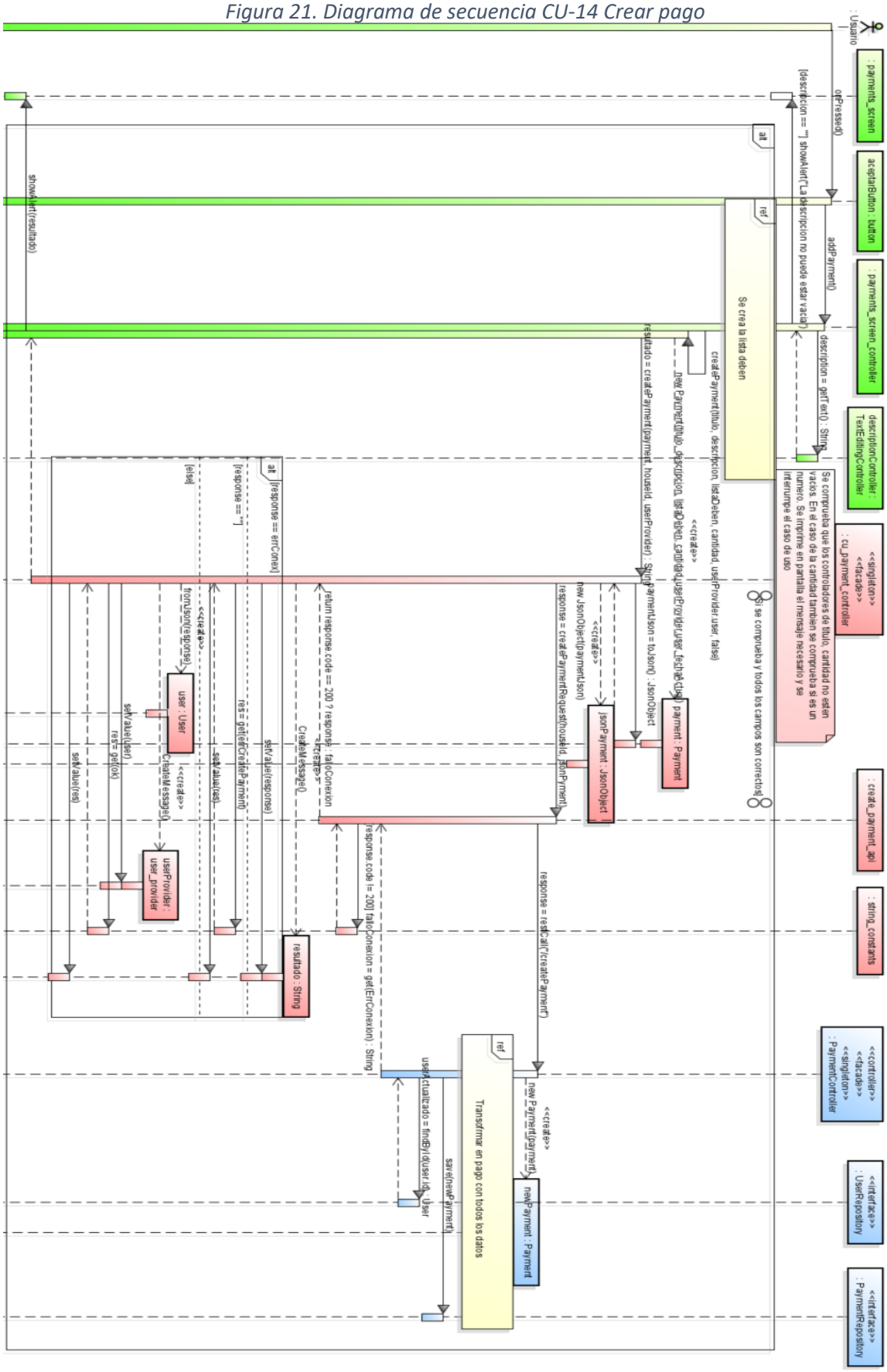
Figura 20. Diagrama de secuencia Inicialización para crear pago

En el siguiente diagrama se muestra la secuencia principal del caso de uso. Cuando el usuario ha completado el formulario y presiona el botón "Aceptar", se activa el controlador de la pantalla de pagos (payment_screen_controller) para recopilar los datos introducidos. Luego, se realiza una serie de comprobaciones, como verificar que no haya campos vacíos y que la cantidad sea un número válido. En caso de que alguna de estas comprobaciones falle, se muestra un mensaje de error y se detiene el caso de uso.

Si todas las comprobaciones son exitosas, se procede a crear el pedido. Antes de eso, se genera una lista de usuarios que deben realizar el pago, lo cual se detalla en otro diagrama para mayor claridad. Una vez que se tiene la lista, se envía el pedido completo al controlador de pagos (cu_payment_controller), quien utiliza una API para realizar una llamada REST al microservicio Java, donde se guarda la información del pedido (más detalles se encuentran en la descripción del diagrama referenciado).

Después de este proceso, la API recibe un objeto de respuesta, en caso de fallo, se devolvera un mensaje de error almacenado en la clase de constantes. Si no hay errores, el pedido se crea y se actualiza la información en la interfaz, marcando así el final del caso de uso.

Figura 21. Diagrama de secuencia CU-14 Crear pago



En el diagrama se presenta el diagrama referenciado "Crear lista deben", donde se muestra el proceso de creación de la lista de usuarios que deben realizar el pago. En este caso, el controlador de la página de pagos recopila los ID de los usuarios y verifica si están marcados mediante la comparación de las dos listas que están en el mismo orden.

Si en la lista "userCheckedList" se encuentra un valor booleano "true" en una posición determinada, se crea un objeto Usuario con el respectivo ID y se agrega a la lista "listaDeben" que contiene los usuarios que deben realizar el pago. Esta lista será posteriormente agregada al nuevo pedido para asociar los usuarios correspondientes.

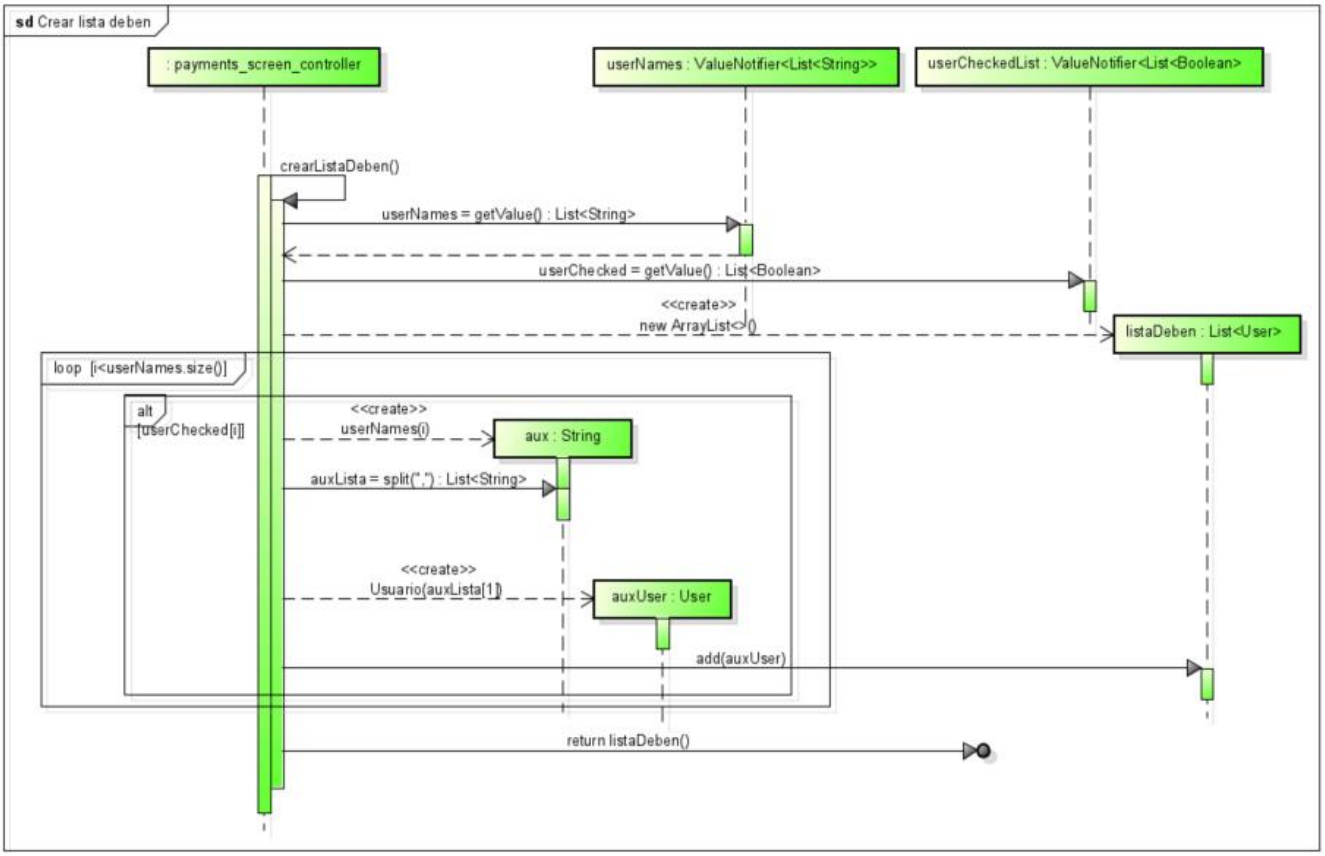


Figura 22. Diagrama de secuencia Creación lista de deudas

En el último diagrama, se muestra la transformación realizada por el microservicio Java al pedido antes de guardarlo. El pedido que llega al microservicio contiene la información necesaria para un pedido, pero no incluye todos los datos disponibles (como los datos adicionales del usuario). Por lo tanto, el controlador del microservicio realiza consultas a la base de datos utilizando los repositorios JPA para obtener toda la información de la casa, del usuario que ha realizado el pago y de todos los usuarios que deben pagar la deuda.

Una vez obtenida toda esta información adicional, se agrega al pago, enriqueciendo así sus datos. Después de completar este proceso de adición de datos, el pago está listo para ser guardado, como se muestra en el diagrama principal.

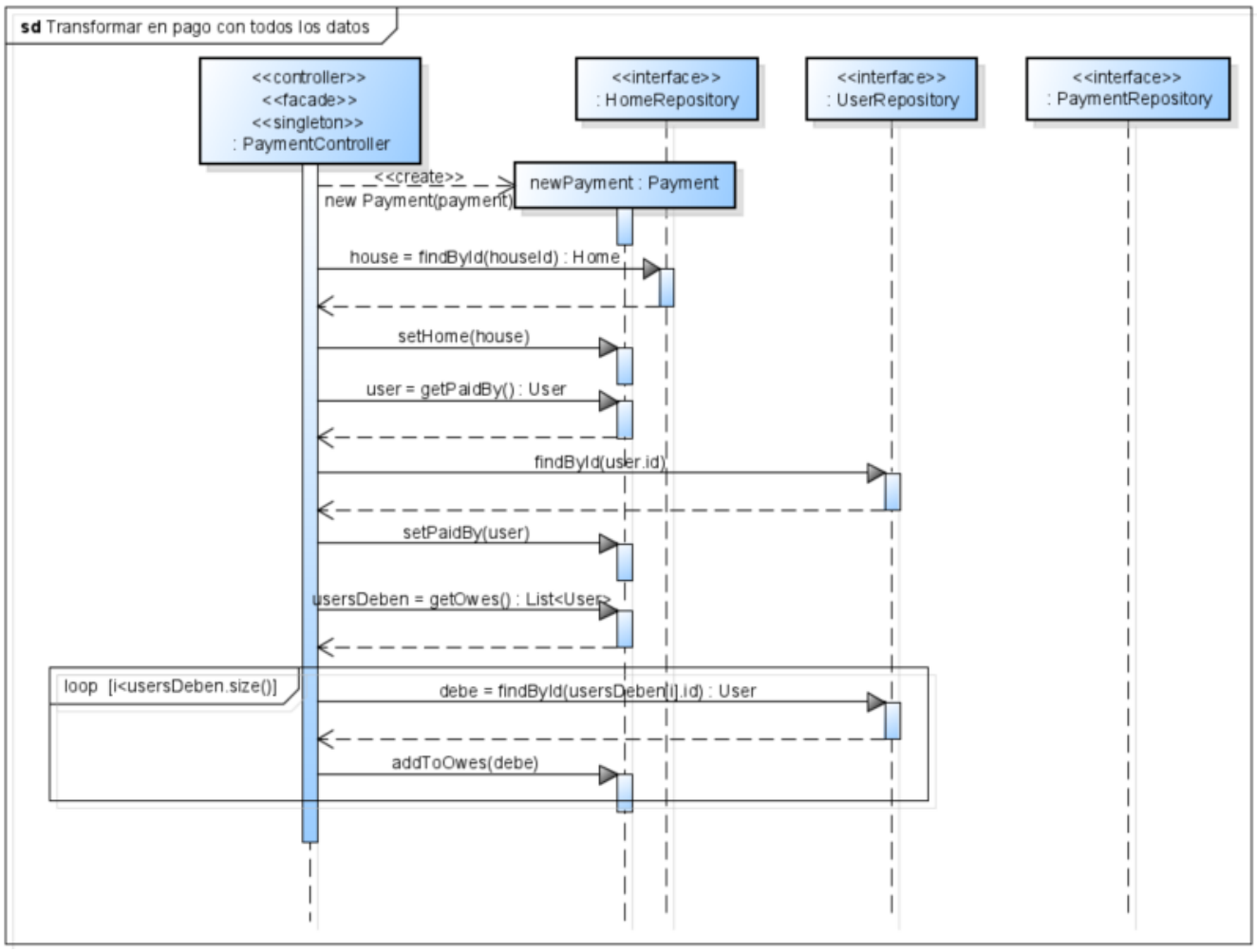


Figura 23. Diagrama de secuencia Transforamción a pago con todos los datos

5.6. Patrones utilizados

A la hora de diseñar la aplicación se han utilizado un numero de patrones para facilitar las implementacion más adelante. Los patrones utilizados han sido:

5.6.1. Patrón Observador

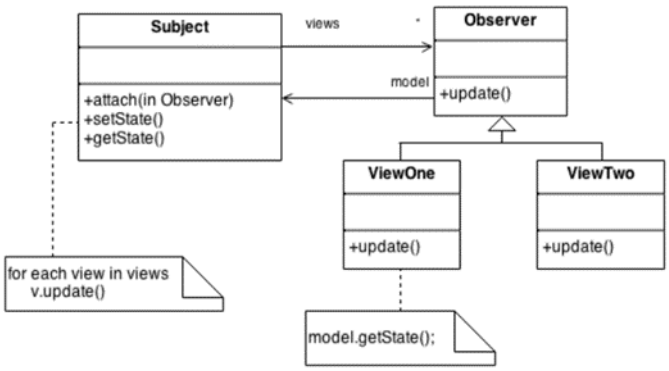


Figura 24. Patrón Observer

El patrón Observer establece una relación de dependencia uno a muchos entre un sujeto y varios observadores. El sujeto mantiene una lista de observadores y les notifica automáticamente cuando hay cambios en su estado. Los observadores se registran con el sujeto y reciben actualizaciones cuando ocurren cambios. Esto permite una comunicación flexible y desacoplada, donde los observadores pueden tomar acciones basadas en los cambios del sujeto. El patrón Observer se utiliza en situaciones donde se necesita una notificación eficiente y automática de eventos o cambios en un objeto a varios observadores.

Esto se ha usado en las interfaces para cambiar los datos que se están mostrando sin tener que refrescar toda la página.

5.6.2. Patrón State

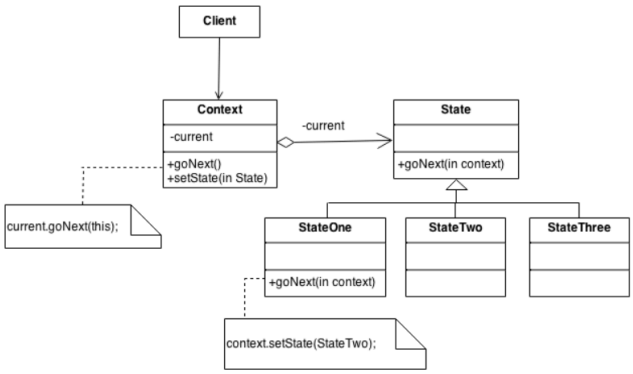


Figura 25. Patrón State

El patrón State permite que un objeto altere su comportamiento cuando su estado interno cambia. Proporciona una forma de cambiar dinámicamente el comportamiento de un objeto en función de su estado actual sin que el objeto sea consciente de los detalles específicos de cada estado. El patrón State se compone de diferentes clases que representan los distintos estados posibles y una clase de contexto que mantiene una referencia al estado actual. El contexto delega el comportamiento a la instancia de estado actual, lo que permite que el objeto se comporte de manera diferente según su estado. Esto facilita la adición de nuevos estados y cambios en el comportamiento del objeto sin afectar directamente a la lógica existente. El patrón State se utiliza cuando un objeto debe cambiar su comportamiento según su estado interno y cuando existen múltiples comportamientos diferentes asociados con diferentes estados.

El patrón State se ha utilizado ampliamente en la aplicación, especialmente en el caso de uso mencionado anteriormente, como el formulario de pago, donde hay tres estados: creando, viendo y editando el pedido. También se aplica en la pantalla de inicio de sesión, donde hay un estado para iniciar sesión y otro para registrarse.

5.6.3. Patrón Fachada

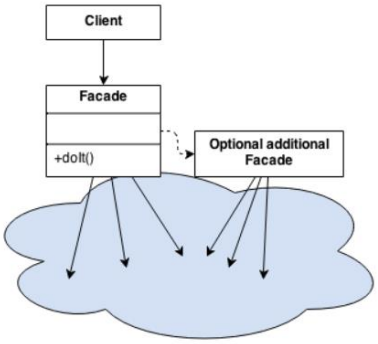


Figura 26. Patrón Fachada

La Fachada es un patrón de diseño que actúa como una capa intermedia entre los clientes y los subsistemas más complejos de un sistema. Su objetivo principal es proporcionar una interfaz unificada y simplificada, ocultando la complejidad interna y los detalles de implementación. Esto facilita la interacción de los clientes con el sistema, reduciendo la curva de aprendizaje y mejorando la mantenibilidad.

En términos prácticos, la Fachada se implementa como una clase o componente que encapsula la lógica de interacción con los subsistemas. Actúa como un punto de entrada único para los clientes y dirige sus solicitudes hacia los componentes apropiados del sistema. Además, puede coordinar las interacciones entre múltiples subsistemas, brindando una interfaz coherente y simplificada para los clientes. Un ejemplo en este proyecto son los controladores de casos de uso.

5.6.4. Patrón Singleton

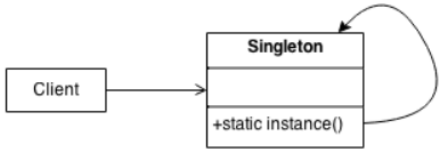


Figura 27. Patrón Singleton

El patrón Singleton es un patrón de diseño creacional que garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a dicha instancia. Esto se logra mediante la implementación de un constructor privado y un método estático para obtener la instancia única. Cuando se solicita la instancia, se comprueba si ya existe y se devuelve, evitando así la creación de múltiples instancias. Este patrón es útil en situaciones en las que se necesita compartir recursos o datos en todo el sistema y se requiere un único punto de acceso para garantizar la coherencia y evitar la duplicación de datos. Sin embargo, se debe tener cuidado al utilizar el patrón Singleton, ya que puede introducir acoplamiento y dificultar la prueba unitaria.El patrón sigleton se utiliza en todos los controladores de uso y en el director de la interfaz.

5.6.5. Patrón Builder

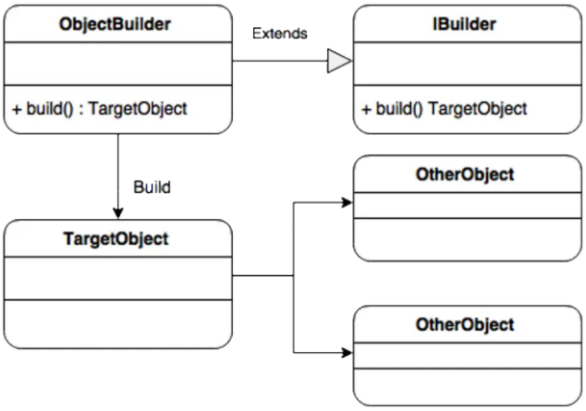


Figura 28. Patrón Builder

El patrón Builder es un patrón de diseño creacional que se utiliza para construir objetos complejos paso a paso. Permite separar la creación de un objeto de su representación final, lo que proporciona flexibilidad y facilidad de uso. En lugar de tener un constructor con numerosos parámetros, el patrón Builder utiliza un objeto Builder para construir el objeto final paso a paso, añadiendo los componentes necesarios en cada etapa. Esto facilita la creación de objetos con configuraciones diferentes o personalizadas, ya que se pueden utilizar diferentes constructores de Builder para proporcionar diferentes implementaciones de cada paso de construcción. Este patrón es usado al crear todas las pantallas de la aplicación, junto con todos los *widgets* dado que es la manera de crear que tiene *Flutter*.

5.7. Diseño de la interfaz gráfica

Antes de comentar la implementación del proyecto, se realizaron varios prototipos o mockups de algunas pantallas. Estos bocetos se crearon con el fin de alinear la visión del cliente y el desarrollador en cuanto al diseño de la interfaz, y también para servir como base durante el proceso de desarrollo.

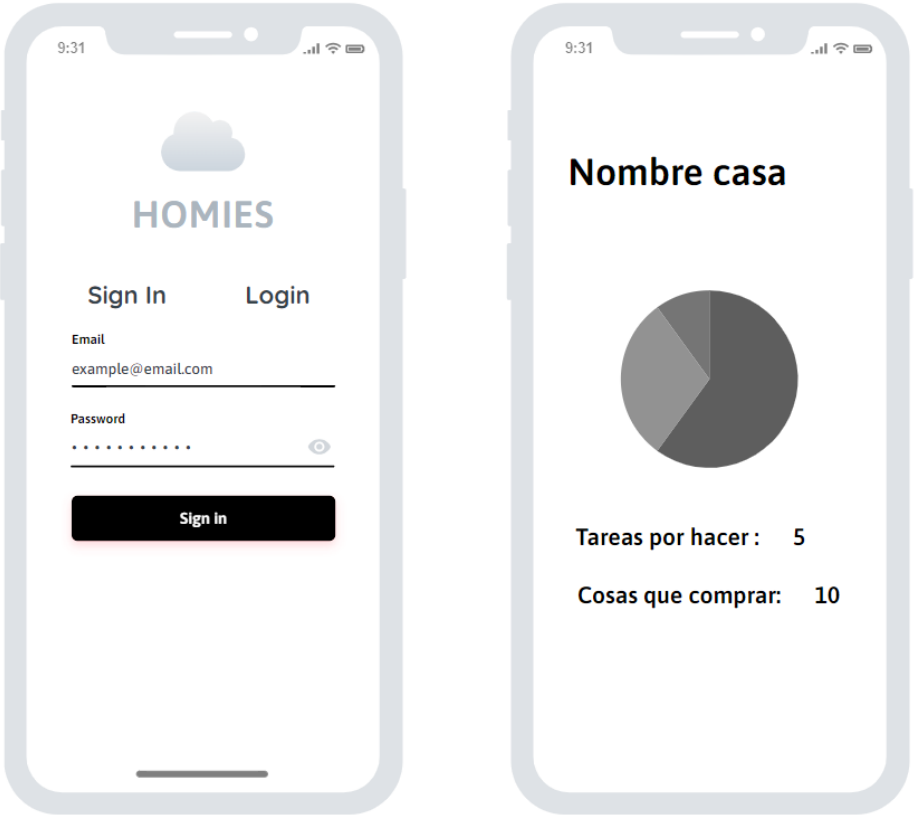


Figura 29. Boceto de pantalla de registro/login y pantalla de casa principal

Figura 30. Boceto de pantalla de crear pedido y lista de tareas

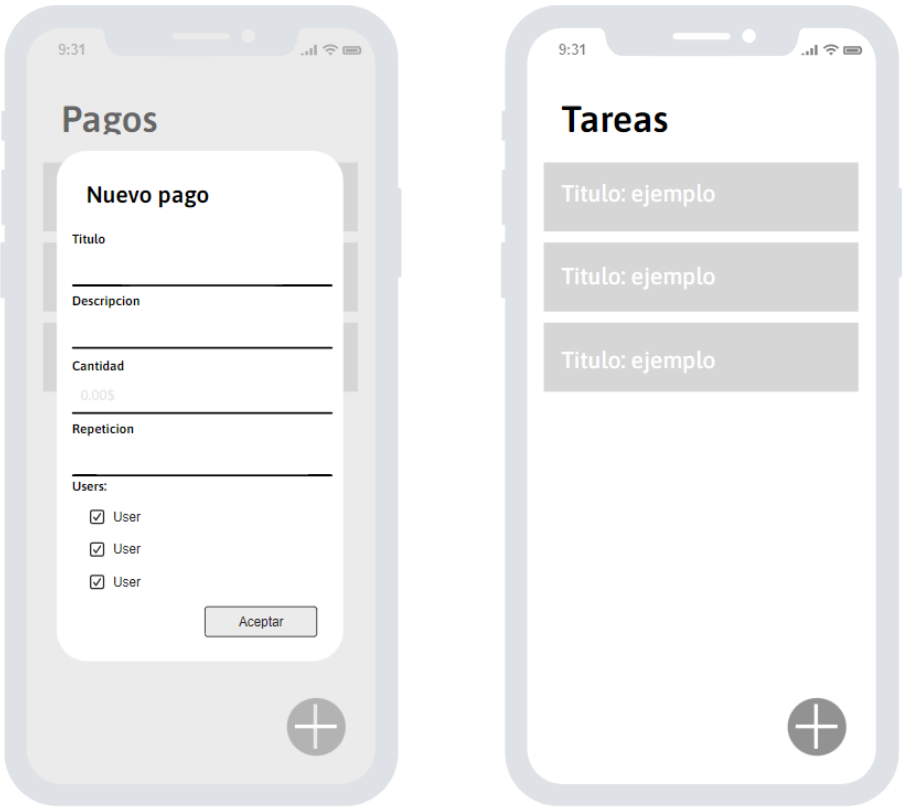
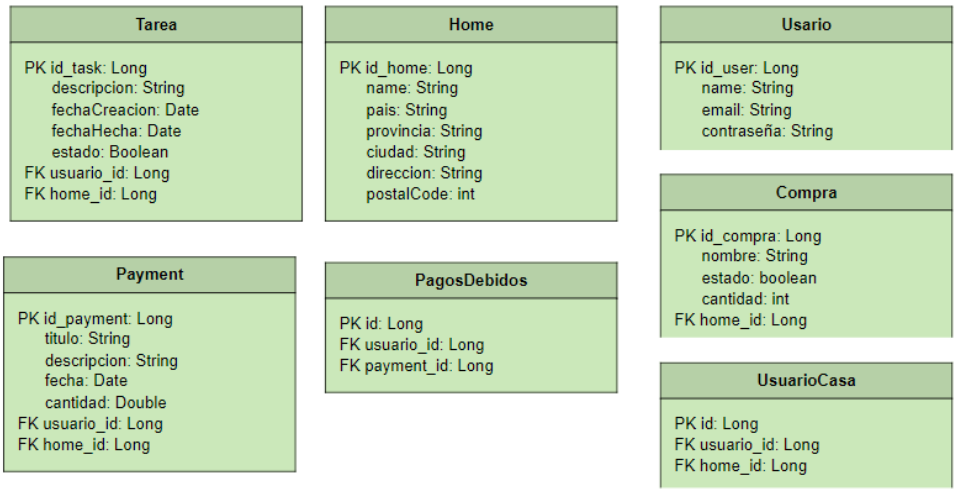


Figura 31. Boceto de pantalla de crear pedido y lista de tareas

5.8. Diseño de la base de datos

Para el diseño de este proyecto se utilizó una base de datos MySQL, un sistema de gestión de bases de datos relacional ampliamente utilizado. MySQL organiza la información en tablas estructuradas y permite establecer relaciones entre ellas mediante claves primarias y foráneas. Es compatible con SQL y ofrece funcionalidades completas para el almacenamiento, gestión y consulta de datos, además de opciones de escalabilidad y rendimiento.



CAPÍTULO 6

6. Implementación

En este capítulo, se aborda la fase de implementación del proyecto, donde se detalla la selección de las tecnologías y herramientas utilizadas para su desarrollo. Además, se describe la organización del código, proporcionando información relevante para el usuario o programador.

6.1. Herramientas utilizadas

Para la realización de este proyecto, se han utilizado una serie de herramientas. Entre ellas se encuentran aquellas consideradas necesarias para su desarrollo, así como otras que han proporcionado un apoyo adicional. A continuación, se detallan dichas herramientas y se explica para qué se ha empleado cada una de ellas.

6.1.1. Flutter

Flutter es un framework de desarrollo de aplicaciones móviles de código abierto desarrollado por *Google*. Con *Flutter*, los desarrolladores pueden crear aplicaciones nativas para *iOS*, *Android* y web utilizando un único código base. Su enfoque en el desarrollo rápido se destaca gracias a la función de "hot reload", que permite ver los cambios realizados en tiempo real, acelerando el proceso de desarrollo y facilitando la corrección eficiente de errores.

Una de las fortalezas de *Flutter* radica en la creación de interfaces de usuario atractivas y de alto rendimiento. El framework proporciona una amplia gama de widgets personalizables y predefinidos que se adaptan automáticamente al diseño y estilo de cada plataforma, ofreciendo una apariencia nativa en todas ellas.

Otra ventaja significativa de *Flutter* es su capacidad para mantener la consistencia en diferentes plataformas. El código escrito una vez en *Flutter* puede ejecutarse en *iOS*, *Android* y web sin necesidad de modificaciones, lo que simplifica el desarrollo, reduce los costos y el tiempo asociados con la creación de aplicaciones multiplataforma. En este proyecto se prevé que la mayoría de los usuarios sean a través de dispositivos *Android*, y *Flutter* puede ser ejecutado desde *Android* 4.1, versión que tiene más del 99% de usuarios *Android*.

Flutter cuenta con una comunidad activa de desarrolladores y un sólido soporte por parte de *Google*, lo que brinda acceso a una amplia gama de recursos y documentación. *Además*, *Google* continúa invirtiendo en el desarrollo y mejora de *Flutter*, asegurando su actualización constante y mejora continua.

6.1.2. Visual Studio Code

Visual Studio Code es un entorno de desarrollo integrado (IDE) ampliamente utilizado por los ingenieros informáticos. Se destaca por su ligereza, rapidez y amplia compatibilidad con diferentes lenguajes de programación. Además, ofrece una amplia gama de extensiones y temas para personalizar el entorno de trabajo. Estas características lo convierten en una opción atractiva para mejorar la productividad y adaptarse a las necesidades individuales de los desarrolladores. En resumen, Visual Studio Code es un IDE popular debido a su eficiencia, versatilidad y capacidad de personalización.

6.1.3. Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial de Google para la creación de aplicaciones móviles en el sistema operativo Android. Es ampliamente utilizado por los ingenieros informáticos debido a sus características específicas para el desarrollo en esta plataforma. En este proyecto, se ha utilizado Android Studio debido a su facilidad para crear y emular un dispositivo Android, que ha sido utilizado para probar el proyecto en desarrollo.

6.1.4. JDBC

JDBC (Java Database Connectivity) es una API estándar de Java utilizada para conectarse y operar con bases de datos relacionales. He utilizado JDBC en el proyecto para establecer la conexión con la base de datos y realizar operaciones de lectura y escritura. Proporciona una interfaz uniforme y flexible para interactuar con diferentes sistemas de gestión de bases de datos, lo que permite la persistencia y manipulación de datos de manera eficiente en la aplicación Java.

6.1.5. Gitlab

GitLab es una plataforma de gestión de repositorios de código basada en Git. Se utiliza en proyectos de desarrollo de software para facilitar el control de versiones, la colaboración en equipo y la entrega continua. En este proyecto, se ha utilizado GitLab como una herramienta esencial para alojar y gestionar el repositorio de código del proyecto, permitiendo un seguimiento detallado de los cambios, facilitando la colaboración entre los miembros del equipo y agilizando el proceso de desarrollo y entrega del software.

6.1.6. Astah

Astah es una herramienta de modelado y diseño de software utilizada por los ingenieros informáticos para crear diagramas UML y visualizar ideas de diseño de software de manera clara y precisa.

Ha sido utilizado para realizar todos los diagramas de este proyecto

6.1.7. BalsamiQ

Balsamiq es una herramienta de prototipado y diseño de interfaces de usuario que se utiliza para crear bocetos de manera rápida y sencilla. En este proyecto, hemos utilizado Balsamiq para desarrollar y visualizar los bocetos iniciales de la interfaz de usuario de la aplicación. Con Balsamiq, hemos podido representar las ideas y conceptos de diseño de manera visual, lo que ha facilitado la comunicación y colaboración entre el equipo de desarrollo y los stakeholders del proyecto.

6.2. Organización de código

En el proyecto, se ha adoptado una arquitectura de microservicios, donde se tienen dos microservicios: uno implementado en Flutter para la interfaz gráfica y el negocio, y otro en Java encargado de la conexión con la base de datos.

El microservicio de Flutter sigue una arquitectura MVVM (Modelo-Vista-VistaModelo) y presenta la siguiente estructura:

- Negocio: En esta carpeta se encuentran los controladores de los casos de uso, los DTOs (Objetos de Transferencia de Datos) y las APIs que se conectan con el microservicio Java. Los archivos se organizan en carpetas según los diferentes casos de uso.
- Interfaz: Aquí se encuentra el director de la interfaz (ui_manager), que utiliza rutas para redirigir las pantallas de manera conveniente. Además, se encuentran las pantallas con sus controladores correspondientes, los "provider" que son clases que almacenan datos necesarios en la memoria caché de la aplicación, y los "widget" que son partes de código reutilizables en varias pantallas, como la barra de navegación, botones o campos de texto.
- Modelo: En esta carpeta se encuentran las clases correspondientes al modelo del proyecto. Estas clases tienen un método "fromJson" para convertirlas de formato JSON a la clase correspondiente.

Por otro lado, en el microservicio de Java se sigue una arquitectura MVC (Modelo-Vista-Controlador) y se estructura de la siguiente manera:

- Modelo: Representa los datos y la lógica de negocio de la aplicación. Los modelos son clases o estructuras que mapean directamente a las tablas de la base de datos. Se encargan de realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos y contienen la lógica para validar y manipular los datos.
- Vista: Representa la interfaz de usuario de la aplicación. En el contexto de una API REST, como es en este proyecto, la vista se refiere a la respuesta JSON generada por el controlador.
- Controlador: Actúa como intermediario entre las solicitudes del cliente y los modelos/repositorios correspondientes. El controlador maneja las solicitudes HTTP entrantes, extrae los parámetros necesarios y realiza llamadas a los modelos o repositorios adecuados. También puede encargarse de la validación de datos y la gestión de errores.

- Repositorio: Es responsable de la interacción directa con la base de datos. Proporciona métodos para realizar consultas y operaciones en la base de datos, como obtener datos, crear, actualizar o eliminar registros. En este proyecto se utiliza JDBC para manejar estas operaciones.

CAPÍTULO 7

7. Pruebas

En este capítulo se documentan las pruebas realizadas a lo largo del proyecto para verificar los requisitos solicitados y garantizar el correcto funcionamiento del código. Estas pruebas se han llevado a cabo con el fin de evaluar aspectos diversos como la funcionalidad, el rendimiento y la usabilidad de la aplicación.

Las pruebas unitarias y de integración que ha hecho el autor durante el desarrollo no han sido documentadas para no hacer esta sección demasiado voluminosa.

7.1. Pruebas unitarias y de integración

Las pruebas unitarias son pruebas que se realizan a nivel de componentes individuales de software, como funciones, métodos o clases, para verificar su correcto funcionamiento de forma aislada. Estas pruebas se centran en comprobar que cada unidad de código cumple con los requisitos y produce los resultados esperados.

Por otro lado, las pruebas de integración se realizan para evaluar la interacción y la correcta integración de diferentes componentes de software en conjunto. Estas pruebas se enfocan en verificar que los distintos módulos o unidades de código trabajen adecuadamente cuando se combinan, y que se cumplan las interfaces y la comunicación entre ellos.

En este proyecto, aunque se han llevado a cabo pruebas unitarias y de integración por parte del desarrollador, no se documentarán detalladamente en este documento para evitar que se vuelva demasiado extenso.

7.2. Pruebas de aceptación

Las pruebas de aceptación son un conjunto de pruebas realizadas para verificar si un sistema cumple con los requisitos y expectativas del cliente o usuario final. Estas pruebas se llevan a cabo en un entorno cercano al ambiente de producción y tienen como objetivo principal asegurar que el sistema cumpla con los criterios de aceptación previamente establecidos.

En el proceso de pruebas de aceptación, se definen los criterios de aceptación que determinan los requisitos y expectativas del sistema. A partir de estos criterios, se diseñan casos de prueba que cubren diferentes escenarios y se ejecutan las pruebas correspondientes. Los resultados de las pruebas se validan y se comparan con los criterios de aceptación establecidos. Finalmente, se genera un informe que resume los resultados de las pruebas y cualquier problema identificado durante el proceso.

Las pruebas de aceptación son fundamentales para garantizar la calidad y el cumplimiento de los requisitos del sistema. Al realizar estas pruebas, se asegura que el producto final cumpla con las expectativas del cliente y

funcione de acuerdo a lo acordado. Esto ayuda a evitar problemas y desviaciones en la etapa de implementación y entrega del proyecto.

7.2.1. Casos de prueba

Para comprobar el correcto funcionamiento de la aplicación, se han llevado a cabo casos de prueba para todos los casos de uso. Sin embargo, en este documento se documentarán solo una parte de ellos para evitar que esta sección sea demasiado extensa.

Para comprobar el correcto funcionamiento de la aplicación, se han realizado casos de prueba para todos los casos de uso, incluyendo la creación de tareas y pagos. Sin embargo, en este documento se documentará solo una selección de dichos casos de prueba para evitar la repetición innecesaria de información, dado que los casos de prueba como la creación de tareas y pagos serían muy similares, igual que los casos de prueba entre editar y crear un pago.

Además, algunos casos de uso no requieren pruebas adicionales, ya que han sido validados en las pruebas unitarias y de integración realizadas por el desarrollador. Por ejemplo, en el caso de "Mostrar lista de la compra", el usuario no puede realizar acciones que afecten su funcionamiento, siempre y cuando el desarrollador haya realizado la prueba unitaria correspondiente para asegurar su correcto desempeño.

CP-01	**Registrarse**
Descripcion	Se introduce un correo que ya ha sido utilizado por otro usuario
Pasos de ejecución	1. Abrir página de registrarse 2. Introducir nombre “Miguel” 3. Introducir email “miguel@gmail.com” (correo ya utilizado) 4. Introducir contraseña “1234” 5. Hacer click en el botón de registro
Resultados esperados	El sistema muestra el error “Error de Registro, el email ya está en uso” El sistema sigue en la pantalla de registro
Estado de prueba	Aprobado

Tabla 28. Caso de Prueba 1 – Registrarse

CP-02	**Registrarse**
Descripcion	Se introduce un campo vacío
Pasos de ejecución	1. Abrir página de registrarse 2. Introducir nombre “” 3. Introducir email “miguel@gmail.com” 4. Introducir contraseña “1234” 5. Hacer click en el botón de registro
Resultados esperados	El sistema muestra el error "Algún campo está vacío" El sistema sigue en la pantalla de registro
Estado de prueba	Aprobado

Tabla 29. Caso de Prueba 2 – Registrarse

CP-03	Identificarse
Descripcion	Se introduce un email que no ha sido registrado
Pasos de ejecución	1. Abrir página de inicio sesión 2. Introducir email “miguel@gmail.com” (no registrado) 3. Introducir contraseña “1234” 4. Hacer click en el botón de inicio de sesión
Resultados esperados	El sistema muestra el error "Error de Inicio de sesión, incorrecto email o contraseña" El sistema sigue en la pantalla de inicio de sesión
Estado de prueba	Aprobado

Tabla 30. Caso de Prueba 3 – Identificarse

CP-04	Identificarse
Descripcion	Se introduce una contraseña incorrecta
Pasos de ejecución	1. Abrir página de inicio sesión 2. Introducir email “miguel@gmail.com” 3. Introducir contraseña “12345” (correcta: 1234) 4. Hacer click en el botón de inicio de sesión
Resultados esperados	El sistema muestra el error "Error de Inicio de sesión, incorrecto email o contraseña" El sistema sigue en la pantalla de inicio de sesión
Estado de prueba	Aprobado

Tabla 31. Caso de Prueba 4 – Identificarse

CP-05	Crear Pago
Descripcion	Se crea con un campo vacío
Pasos de ejecución	1. Abrir página de creación de tarea 2. Introducir título “Alquiler” 3. Introducir descripción “” 4. Introducir cantidad “600” 5. Pulsar los checkbox de todos los usuarios 6. Hacer click en el botón de crear pago
Resultados esperados	El sistema muestra el error "Algún campo está vacío" El sistema sigue en la pantalla de creación de pago
Estado de prueba	Aprobado

Tabla 32. Caso de Prueba 5 – Crear Pago

CP-06	Crear Pago
Descripción	Se crea con la cantidad no siendo solo un numero
Pasos de ejecución	1. Abrir página de creación de tarea 2. Introducir título “Alquiler” 3. Introducir descripción “Pago del alquiler” 4. Introducir cantidad “600x” 5. Pulsar los checkbox de todos los usuarios 6. Hacer click en el botón de crear pago
Resultados esperados	El sistema muestra el error "La cantidad debe ser un numero" El sistema sigue en la pantalla de creación de pago
Estado de prueba	Aprobado

Tabla 33. Caso de Prueba 6 – Crear Pago

CP-07	Crear Pago
Descripción	No se pulsa ningún checkbox de usuario
Pasos de ejecución	1. Abrir página de creación de tarea 2. Introducir título “Alquiler” 3. Introducir descripción “Pago del alquiler” 4. Introducir cantidad “600” 5. Hacer click en el botón de crear pago
Resultados esperados	El sistema muestra el error "Se debe marcar al menos un usuario" El sistema sigue en la pantalla de creación de pago
Estado de prueba	Aprobado

Tabla 34. Caso de Prueba 7 – Crear Pago

CP-08	Editar una compra
Descripción	Se edita con una cantidad no
Pasos de ejecución	7. Abrir página de creación de tarea 8. Introducir título “Alquiler” 9. Introducir descripción “Pago del alquiler” 10. Introducir cantidad “600x” 11. Pulsar los checkbox de todos los usuarios 12. Hacer click en el botón de crear pago
Resultados esperados	El sistema muestra el error "La cantidad debe ser un numero" El sistema sigue en la pantalla de creación de pago
Estado de prueba	Aprobado

Tabla 35. Caso de Prueba 8 – Editar una compra

7.3. Prueba de usabilidad

Las pruebas de usabilidad son pruebas diseñadas para evaluar la facilidad de uso, la eficiencia y la satisfacción del usuario al interactuar con un producto o sistema. Estas pruebas se centran en recopilar comentarios y observaciones directas de los usuarios mientras realizan tareas específicas, con el objetivo de identificar problemas de usabilidad y mejorar la experiencia del usuario.

En el caso de este proyecto, se ha introducido la aplicación a un grupo de cuatro personas seleccionadas como usuarios para realizar pruebas de usabilidad. Estas personas representan diferentes perfiles de usuario y se les ha proporcionado acceso a la aplicación para que la utilicen en situaciones reales. Durante este proceso, se les ha solicitado que realicen una variedad de tareas y se ha registrado su feedback y observaciones.

La introducción de la aplicación a estos usuarios permite recopilar información valiosa sobre la facilidad de uso, la eficiencia y la satisfacción del usuario. Los comentarios y las observaciones obtenidas de estos usuarios servirán para identificar áreas de mejora y realizar ajustes necesarios en la aplicación con el objetivo de ofrecer una experiencia óptima a todos los usuarios.

7.4. Escenarios ha probar

A continuación, se presentarán tablas que explicarán los escenarios que los usuarios deben realizar, así como los resultados y observaciones obtenidos de estas pruebas.

Prueba usabilidad 1		**Crear una casa**
Descripción		Te acabas de independizar con unos amigos y quieres crear una casa en la que se encuentren todos los convivientes.
User 1	**Resultado**	Éxito en 30s
	Observaciones	Sin problemas
User 2	**Resultado**	Éxito en 2min
	Observaciones	Genial
User 3	**Resultado**	Éxito en 2min
	Observaciones	Al crear una cuenta la lista de casas no debería estar vacía, aunque sea un mensaje.
User 4	**Resultado**	Éxito en 3min
	Observaciones	No me gustan mucho los colores

Tabla 36. Prueba de usabilidad 1 – Crear una casa

Prueba usabilidad 2		**Marcar toda la lista de compras como hecha**
Descripción		Acabas de ir a la compra y quieres vaciar la lista de la compra para que tus compañeros no compren lo mismo.
User 1	**Resultado**	Éxito en 20s
	Observaciones	Sin problemas
User 2	**Resultado**	Éxito en 15s
	Observaciones	Muy fácil
User 3	**Resultado**	Éxito en 30s
	Observaciones	La pantalla debería decir “Todo comprado” o algo parecido
User 4	**Resultado**	Éxito en 1m
	Observaciones	El icono de la bolsa me gusta lo hace muy claro

Tabla 37. Prueba de usabilidad 2 – Marcar toda la lista de la compra como hecha

Prueba usabilidad 3		**Pagar una deuda**
Descripción		Tu compañero es el que paga el alquiler todos los meses, por lo que le debes dinero
User 1	**Resultado**	Éxito en 20s
	Observaciones	Fácil de hacerlo con el botón en la lista
User 2	**Resultado**	Éxito en 15s
	Observaciones	Muy fácil
User 3	**Resultado**	Éxito en 30s
	Observaciones	Debería haber una confirmación después de pulsar el botón
User 4	**Resultado**	Éxito en 1.5m
	Observaciones	Empezó creando un pago, pero después pulso el botón de devolver deuda

Tabla 38. Prueba de usabilidad 3 – Pagar una deuda

Prueba usabilidad 4		Marcar tarea como hecha
Descripción		Has fregado las cazuelas, por lo que quieres marcar que lo has hecho
User 1	Resultado	Éxito en 1m
	Observaciones	Es intuitivo deslizar para borrar, pero no está claro
User 2	Resultado	Éxito en 40s
	Observaciones	No muy visible
User 3	Resultado	Fracaso
	Observaciones	Las primeras veces con la aplicación debería haber un pequeño tutorial sobre ello
User 4	Resultado	Éxito en 2m
	Observaciones	Edito la tarea en vez de deslizar la tarea

Tabla 39. Prueba de usabilidad 4 – Marcar tarea como hecha

7.4.1. Encuesta

Después de esta prueba de usabilidad, se les ha proporcionado el siguiente cuestionario a los usuarios para que registren su opinión sobre la aplicación. El cuestionario incluye preguntas relacionadas con la facilidad de uso, la satisfacción general, la claridad de la interfaz y cualquier sugerencia o comentario adicional que los usuarios deseen compartir. Los resultados de este cuestionario serán analizados para obtener información valiosa sobre la experiencia del usuario y ayudar a mejorar la aplicación en futuras iteraciones.

Pregunta
¿Qué tan fácil fue para ti utilizar la aplicación?
¿Qué nivel de intuitividad encontraste en la interfaz de la aplicación?
¿Qué tan eficiente fue la aplicación para ayudarte a completar tus tareas?
¿Qué tan claro y comprensible encontraste el diseño de la interfaz de la aplicación?
¿Qué opinas de la apariencia visual de la aplicación?
¿Cuál es tu nivel de satisfacción con el rendimiento y la funcionalidad de la aplicación?
En general, ¿cómo evaluarías tu satisfacción con la aplicación?

Tabla 40. Encuesta

CAPÍTULO 8

8. Resultados

En este capítulo se proporcionará un análisis detallado de los hallazgos y observaciones derivados de las pruebas y evaluaciones realizadas. Estos resultados son fundamentales para comprender la usabilidad, funcionalidad y satisfacción del usuario en relación con la aplicación desarrollada, y servirán como base para futuras decisiones y mejoras.

8.1. Resumen de hallazgos

Durante las pruebas de este proyecto, se ha observado que la mayoría de las funcionalidades de la aplicación funcionan correctamente y cumplen con su propósito. Sin embargo, se ha identificado que algunas funcionalidades específicas presentan cierta complejidad en cuanto a su descubrimiento y accesibilidad. Esto se debe a la estrategia de diseño minimalista y simplificado adoptada, que ha buscado priorizar la estética y la facilidad de uso global de la aplicación.

Se ha notado que, en ocasiones, los usuarios han experimentado dificultades para encontrar ciertas funcionalidades, lo que ha generado un ligero impacto en la eficiencia y la satisfacción del usuario. No obstante, es importante destacar que una vez que los usuarios han descubierto y comprendido cómo acceder a esas funcionalidades, han logrado utilizarlas de manera exitosa y han apreciado la simplicidad y la fluidez de su funcionamiento.

Estos hallazgos brindan información valiosa para futuras iteraciones del diseño, ya que permiten identificar áreas de mejora en términos de accesibilidad y descubrimiento de funcionalidades. Se sugiere considerar la implementación de soluciones como una mayor claridad en la navegación, indicadores visuales o elementos de ayuda contextual que faciliten la localización y comprensión de las funcionalidades menos evidentes.

En resumen, si bien la aplicación ha sido evaluada como generalmente fácil de usar, se ha observado la necesidad de realizar mejoras específicas para optimizar la accesibilidad y el descubrimiento de todas las funcionalidades por parte de los usuarios. Estos resultados proporcionan una base sólida para futuras decisiones de diseño y desarrollo que contribuyan a mejorar la experiencia global de los usuarios.

8.2. Datos cuantitativos

En este apartado se presentará un resumen de los resultados de las pruebas de casos de uso, las pruebas de usabilidad y el cuestionario realizado a los usuarios.

En las pruebas de casos de uso, se ha verificado que todas las funcionalidades de la aplicación están funcionando correctamente y se han controlado las acciones que podrían llevar a posibles fallos. Esto demuestra que el desarrollo del proyecto ha sido exitoso y las funcionalidades implementadas cumplen con sus objetivos.

En la siguiente tabla se muestran los resultados de la encuesta de usabilidad respondida por los usuarios que participaron en las pruebas. Se puede observar que la mayoría de las puntuaciones son muy altas, con una media de 4.46 sobre 5. Esto indica que los usuarios han percibido la aplicación como fácil de usar, intuitiva, eficiente y con un diseño claro y comprensible. Además, la apariencia visual de la aplicación también ha sido bien valorada. En general, los usuarios han expresado un alto nivel de satisfacción con el rendimiento y la funcionalidad de la aplicación.

Pregunta	**Puntuación 0-5**			
	User 1	**User 2**	**User 3**	**User 4**
¿Qué tan fácil fue para ti utilizar la aplicación?	5	5	5	4
¿Qué nivel de intuitividad encontraste en la interfaz de la aplicación?	4	5	4	3
¿Qué tan eficiente fue la aplicación para ayudarte a completar tus tareas?	5	5	5	5
¿Qué tan claro y comprensible encontraste el diseño de la interfaz de la aplicación?	5	5	3	3
¿Qué opinas de la apariencia visual de la aplicación?	4	5	4	3
¿Cuál es tu nivel de satisfacción con el rendimiento y la funcionalidad de la aplicación?	5	5	5	5
En general, ¿cómo evaluarías tu satisfacción con la aplicación?	5	5	4	4

Tabla 41. Resultados encuesta

CAPÍTULO 9

9. Conclusiones

El capítulo de conclusiones ofrece un cierre a todo el proceso llevado a cabo en el proyecto, brindando un análisis reflexivo sobre los resultados obtenidos, las lecciones aprendidas y las recomendaciones para futuras mejoras. A través de estas conclusiones, se busca resumir de manera concisa y precisa los aspectos más relevantes del proyecto y sentar las bases para posibles futuros trabajos y optimizaciones.

9.1. Importancia de los resultados

Al analizar los resultados, se ha evidenciado la importancia de comprender el funcionamiento de la aplicación y de identificar áreas de mejora para considerar futuros proyectos.

Los resultados obtenidos destacan la satisfactoria usabilidad de la aplicación, al mismo tiempo que señalan áreas específicas en las que se pueden implementar mejoras. Estos hallazgos brindan una base sólida para futuras iteraciones y el desarrollo de nuevas versiones, con el objetivo de ofrecer una experiencia aún más fluida y satisfactoria para los usuarios.

9.2. Cumplimiento de objetivos

En el comienzo de este documento, se establecieron objetivos claros para este proyecto. Los objetivos se dividieron en dos categorías: los objetivos de la aplicación, centrados en funcionalidades y usabilidad, y los objetivos de formación, que buscaban el aprendizaje y desarrollo del autor durante el proceso.

Los objetivos de la aplicación se enfocaron en lograr que esta fuera una herramienta eficiente para la organización de tareas entre compañeros de vivienda, facilitar la gestión de la lista de la compra compartida, almacenar los pagos y deudas de los usuarios, realizar un seguimiento de las actividades individuales y fomentar el esfuerzo y compañerismo mediante comparaciones de esfuerzo. Como hemos constatado en secciones anteriores, todos estos objetivos se han cumplido satisfactoriamente, y la aplicación está funcionando correctamente.

En cuanto a los objetivos de formación, el primer objetivo era desarrollar una aplicación multiplataforma, abarcando dispositivos Android y la web. Este objetivo ha sido alcanzado, demostrando la capacidad del autor para crear una aplicación que funcione en distintos entornos. Además, se llevó a cabo una planificación

exhaustiva durante todo el proceso de desarrollo, tal como se documenta en el segundo capítulo de esta memoria. La planificación y supervisión adecuadas permitieron un avance eficiente en el proyecto.

El tercer objetivo de formación abarcaba todas las fases de creación de la aplicación, desde el análisis de requisitos hasta el diseño, implementación y pruebas. Cada una de estas etapas ha sido cuidadosamente documentada en esta memoria, cumpliendo con el objetivo de registrar todo el proceso de desarrollo.

En conclusión, este proyecto ha alcanzado los objetivos propuestos para la aplicación y ha sido una valiosa oportunidad para el crecimiento y aprendizaje del autor como ingeniero informático, al enfrentarse a nuevos retos y desafíos en el desarrollo de una aplicación multiplataforma completa y funcional.

9.3. Lecciones aprendidas

Durante este proyecto, el autor ha tenido la oportunidad de abordar un proyecto completo en solitario, lo que ha permitido comprender y valorar el costo asociado a esta tarea. Además, se han aprendido valiosas lecciones en términos de planificación y documentación de proyectos, aspectos fundamentales tanto para el futuro laboral de un ingeniero informático como para cualquier persona en general.

Se destaca como un aspecto relevante el hecho de que el autor no contaba con experiencia previa en Flutter ni en Dart, el lenguaje en el cual se basa esta tecnología. Esto implicó un desafío inicial, pero también representó una valiosa oportunidad para adquirir conocimientos en estos lenguajes de programación emergentes. Además, se exploraron nuevas formas de programar, ya que el autor tenía experiencia previa en el uso del patrón MVVM, pero nunca había trabajado con bloques y widgets, que son fundamentales en el entorno de Flutter.

Esta situación permitió al autor expandir sus habilidades y competencias en el ámbito del desarrollo de aplicaciones, ampliando su repertorio de herramientas y enriqueciendo su comprensión de las distintas metodologías y técnicas de programación. A través de esta experiencia, el autor pudo enfrentarse a los desafíos inherentes a una tecnología nueva y adquirir conocimientos prácticos que serán de gran valor en proyectos futuros.

El abordaje de un proyecto de esta magnitud ha brindado al autor la oportunidad de poner en práctica habilidades previas y adquirir nuevas destrezas, logrando una mayor eficiencia y efectividad en el manejo de los distintos componentes y procesos involucrados en el desarrollo de la aplicación. Esta experiencia enriquecedora ha permitido al autor refinar su enfoque y capacidad de resolución de problemas, así como también ha incentivado la exploración de nuevas soluciones y enfoques innovadores.

Como resultado, el autor ha consolidado su expertise en múltiples entornos y ha adquirido un conjunto de habilidades más completo y versátil. Esta mejora en las técnicas empleadas representa un valioso activo para el futuro profesional del autor, permitiéndole abordar con mayor solidez y confianza proyectos de mayor envergadura y complejidad en el campo de la ingeniería informática.

9.4. Futuras mejoras

Como se ha evidenciado a lo largo del proyecto, debido a su utilidad y a la falta de una competencia directa, existe un gran potencial para un futuro prometedor. Desde el inicio, se han considerado funcionalidades adicionales, como un chat entre compañeros y un sistema de organización temporal para eventos como fiestas, cumpleaños u otros objetivos. Aunque la usabilidad de la aplicación es alta, siempre hay margen para mejoras. Se podría implementar una función para compartir casas mediante un enlace o un código QR, así como enviar notificaciones por correo o dispositivo cuando haya movimientos relevantes en la casa. En resumen, este proyecto tiene un amplio espacio para crecer y evolucionar en el futuro.

Bibliografía

1. SourceMaking. (n.d.). Design Patterns. Retrieved from https://sourcemaking.com/design_patterns

2. Atlassian. (n.d.). TFGMIG. Retrieved from https://tfgmiguelimaz.atlassian.net/jira/software/projects/TFGMIG/boards/1

3. Talent.com. (n.d.). Salario Ingeniero Informático en España. Retrieved from https://es.talent.com/salary?job=ingeniero+inform%C3%A1tico

4. Syncfusion. (n.d.). Pie Chart - Flutter Circular Charts. Retrieved from https://help.syncfusion.com/flutter/circular-charts/chart-types/pie-chart

5. XurxoDev. (n.d.). El patrón BLoC junto a Clean Architecture en Flutter. Retrieved from https://xurxodev.com/el-patron-bloc-junto-a-clean-architecture-en-flutter/

6. Hussain, M. (n.d.). Flutter BLoC Clean Architecture Best Practice News APIs. Medium. Retrieved from https://marajhussain.medium.com/flutter-bloc-clean-architecture-best-practice-news-apis-3adb0e2012cc

7. CodeChai. (n.d.). Architecting your Flutter project. Medium. Retrieved from https://medium.com/codechai/architecting-your-flutter-project-bd04e144a8f1

8. Flutter Community. (n.d.). Scalable folder structure for Flutter applications. Medium. Retrieved from https://medium.com/flutter-community/scalable-folder-structure-for-flutter-applications-183746bdc320

9. Codewithandrea.com. (n.d.). Flutter Project Structure: Best Practices & Tips for Building Scalable Apps | Code With Andrea. Retrieved from https://codewithandrea.com/articles/flutter-project-structure/

10. Codewithandrea.com. (n.d.). Flutter App Architecture: Riverpod Introduction | Code With Andrea. Retrieved from https://codewithandrea.com/articles/flutter-app-architecture-riverpod-introduction/

11. Kodeco.com. (n.d.). UML for Android Engineers. Retrieved from https://www.kodeco.com/21792733-uml-for-android-engineers

12. Flutter.dev. (n.d.). Write Your First Flutter App, part 1. Retrieved from https://docs.flutter.dev/get-started/codelab

13. Google Developers Codelabs. (n.d.). Build your first Flutter app. Retrieved from https://codelabs.developers.google.com/codelabs/flutter-codelab-first#1

14. Statista. (n.d.). Porcentaje de compras y ventas vía comercio electrónico España 2008-2019. Retrieved from https://es.statista.com/estadisticas/479921/porcentaje-de-compras-y-ventas-via-comercio-electronico-espana/

15. INEbase. (n.d.). Comercio electrónico por parte de las empresas - INEbase. Retrieved from https://www.ine.es/dyngs/INEbase/es/operacion.htm?c=estadistica_C&cid=1254736176741&menu=ultiDatos&idp=1254735976608

16. OCU.org. (n.d.). Subida de precios en alimentación: ¿qué productos se han encarecido más? | OCU. Retrieved from https://www.ocu.org/consumo-familia/supermercados/noticias/subida-precios-alimentacion

17. INEbase. (n.d.). Índices de Precios de Consumo (IPC) - INEbase. Retrieved from https://www.ine.es/dyngs/INEbase/es/operacion.htm?c=Estadistica_C&cid=1254736152838&menu=ultiDatos&idp=1254735976607

18. GeeksforGeeks. (n.d.). Dart Null Safety - GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/dart-null-safety/

19. Flutter.dev. (n.d.). TextEditingController class - widgets library - Dart API - Flutter API docs. Retrieved from https://api.flutter.dev/flutter/widgets/TextEditingController-class.html

20. Syncfusion Help Documentation. (n.d.). Data Label - Flutter Circular Charts | Syncfusion Help Documentation. Retrieved from https://help.syncfusion.com/flutter/circular-charts/datalabel

21. Pub.dev. (n.d.). syncfusion_flutter_charts | Flutter Package. Retrieved from https://pub.dev/packages/syncfusion_flutter_charts

22. Coding with Flutter. (n.d.). Flutter Case Study: Multiple Navigators with BottomNavigationBar. Medium. Retrieved from https://medium.com/coding-with-flutter/flutter-case-study-multiple-navigators-with-bottomnavigationbar-90eb6caa6dbf