Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Aplicación web para la gestión de alquileres de mobiliario para estudiantes

Autor:
Jaime Castro García

Tutor:
Mario Corrales Astorgano

A mis padres y a todas aquellas personas que me han acompañado durante estos meses. Gracias por vuestro apoyo incondicional.

Agradecimientos

Quiero agradecer en primer lugar a mis padres, cuyo apoyo, confianza y dedicación han sido fundamentales en todo este camino. También expreso mi agradecimiento a mi tutor, Mario Corrales Astorgano, por su ayuda, facilidades y constante adaptación, así como a mi familia y amigos por su respaldo incondicional.

Resumen

Debido a la ausencia de aplicaciones relacionadas con el alquiler temporal de muebles enfocadas para estudiantes y a un aumento en la concienciación sobre sostenibilidad y economía circular, surge la oportunidad de desarrollo de una aplicación que facilite el alquiler de muebles entre particulares. En este proyecto se plantea la creación de una aplicación web desarrollada con arquitectura hexagonal, aplicando tecnologías como Java y Angular, y enfocada en cubrir las necesidades específicas de estudiantes que buscan amueblar temporalmente sus viviendas. Además, se propone la implementación y evaluación de la aplicación mediante pruebas funcionales y de usabilidad, ofreciendo así una solución práctica, escalable y fácil de interactuar para el usuario.

Abstract

The lack of applications aimed specifically at the temporary rental of furniture for students and the increasing awareness of sustainability and circular economy creates the opportunity to develop an application that facilitates furniture rental between individuals. This project proposes the creation of a web application developed with hexagonal architecture, using technologies such as Java and Angular, specifically addressing the needs of students who require temporary furniture solutions. Additionally, it suggests the implementation and evaluation of the application through functional and usability tests, thus providing a practical, scalable, and user-friendly solution.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

La industria del mobiliario genera un impacto ambiental significativo debido a su dependencia de materias primas, energía y recursos naturales. Según datos de la Agencia Europea del Medio Ambiente (EEA) [1], los componentes utilizados en la fabricación de muebles representan el cuarto mayor impacto ambiental en Europa, destacando el elevado consumo de agua, energía y productos químicos. Por ejemplo, la producción de fibras sintéticas para tapicerías consume más de 70 millones de barriles de petróleo anuales.

Asimismo, la gestión ineficiente de los residuos de mobiliario genera hasta 5,6 millones de toneladas de CO_2, lo que equivale a las emisiones anuales netas de países como Suecia. Por tanto, prolongar la vida útil de los muebles mediante su alquiler y reutilización no solo es una solución sostenible, sino también una necesidad para mitigar este impacto.

En este contexto surge GoAway, una solución innovadora diseñada para simplificar el proceso de amueblar viviendas temporales, especialmente dirigida a estudiantes que se enfrentan al reto frecuente de mudarse al inicio de cada curso académico. GoAway ofrece una plataforma práctica, accesible y económica para alquilar muebles de forma temporal, evitando el compromiso y el gasto de adquirir mobiliario nuevo, promoviendo así un modelo de consumo consciente y sostenible.

La plataforma, a través de una interfaz intuitiva y segura, permite a los estudiantes explorar un catálogo diverso de muebles proporcionados por particulares que buscan dar salida a mobiliario que ya no utilizan, incentivando un ciclo responsable de reutilización que fortalece la economía circular.

De este modo, GoAway no solo facilita el acceso a mobiliario temporal, sino que también promueve la colaboración entre usuarios y ayuda a reducir la carga medioambiental generada por la fabricación y gestión ineficiente del mobiliario.

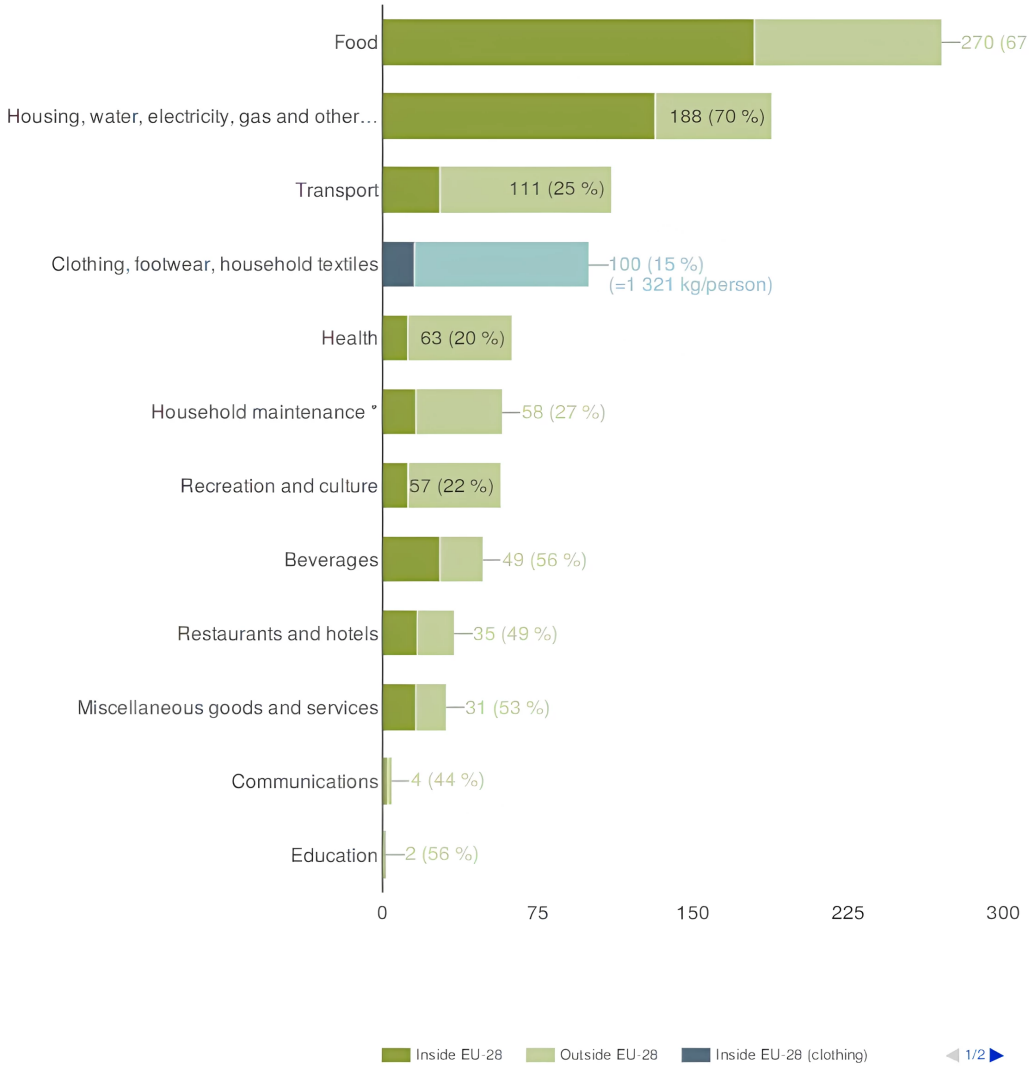


Figura 1.1: Uso de materias primas primarias en la cadena de suministro ascendente de los dominios de consumo doméstico de la UE-28, valores indexados de 2017 [1]

1.2. Motivación

La motivación principal para desarrollar GoAway surge de una necesidad observada en el entorno universitario, donde los estudiantes suelen enfrentarse a constantes cambios de residencia. La compra de muebles nuevos o de trasladarlos todos los años supone no solo un coste considerable, sino que también resulta insostenible desde el punto de vista del medio ambiente.

Además, tras un pequeño análisis del estado actual del mercado, no se han identificado plataformas concretas que resuelvan este problema concreto de alquiler temporal de mobiliario de una manera cómoda, accesible y orientada a estudiantes.

Esta necesidad se completa con un interés personal y profesional por ayudar a estudiantes a pasar una estancia más agradable con muebles de calidad y económicos por un período de tiempo limitado,

así como contribuir al cambio hacia una economía circular y sostenible, aprovechando mis conocimientos adquiridos en el grado de Ingeniería Informática con mención en desarrollo de software, utilizando tecnologías que se mencionan más adelante.

Finalmente, la posibilidad de generar un impacto real en la reducción del desperdicio y fomentar un consumo más responsable entre la comunidad estudiantil refuerza aún más mi motivación personal y académica para llevar a cabo este proyecto.

1.3. Estado de la cuestión

A continuación se explica la necesidad existente y la posición de GoAway frente a otras soluciones disponibles. Se examinan factores como la demanda de los estudiantes, las políticas de sostenibilidad y las plataformas competidoras.

1.3.1. Necesidad de mercado

La movilidad nacional entre los estudiantes universitarios es una realidad consolidada en el sistema universitario español. Según aparece en el informe *Datos y cifras del Sistema Universitario Español 2020-2021 [2]*, publicado por el Ministerio de Universidades [3], un 16,8 % de los estudiantes matriculados en titulaciones de Grado en universidades presenciales residen de manera habitual en una comunidad autónoma diferente a la que cursan sus estudios. Este porcentaje es aún mayor cuando hablamos de cambio de provincia como criterio de movilidad, que asciende hasta un 29,6 %.

Además, algunas comunidades autónomas como Navarra, Castilla y León o Madrid tienen tasas especialmente altas de captación de estudiantes de fuera, superando el 30 % de matriculados procedentes de otras comunidades. Por otro lado, comunidades como Castilla-La Mancha, Baleares o Extremadura registran que aproximadamente la mitad de sus estudiantes optan por estudiar en una universidad fuera de su comunidad de origen.

Estos datos revelan un nicho de mercado claro: miles de estudiantes necesitan trasladarse temporalmente a otras regiones para cursar sus estudios, lo que genera una necesidad de amueblar sus hogares temporales de forma barata y sencilla, especialmente para aquellos estudiantes que residan de forma provisional en pisos compartidos o de alquiler.

Una plataforma web orientada al alquiler de muebles entre particulares cubre esta necesidad de manera eficiente y sostenible, fomentando la economía circular, el reaprovechamiento de recursos y facilitando la instalación rápida y asequible de estudiantes en sus nuevas ciudades de residencia. Esta propuesta se alinea, además, con la creciente conciencia social hacia el consumo responsable y la reutilización de bienes, que se comenta a continuación.

	Estudiantes matriculados	Tasa de movilidad nacional (cambio de CCAA)	Tasa de movilidad nacional (cambio de provincia)
Comunidad autónoma del centro universitario			
Total	**1.099.555**	**16,8%**	**29,6%**
Andalucía	203.595	8,1%	30,6%
Aragón	29.061	17,5%	34,9%
Asturias (Principado de)	17.533	10,0%	10,0%
Balears (Illes)	11.749	4,3%	4,3%
Canarias	34.212	2,6%	11,5%
Cantabria	10.318	16,5%	16,5%
Castilla - La Mancha	24.840	18,2%	35,7%
Castilla y León	63.931	31,2%	53,0%
Cataluña	173.231	10,7%	23,0%
Comunitat Valenciana	119.752	14,6%	26,8%
Extremadura	17.493	10,0%	35,6%
Galicia	49.734	5,9%	38,1%
Madrid (Comunidad de)	231.313	30,8%	30,8%
Murcia (Región de)	42.334	21,1%	21,1%
Navarra (Comunidad Foral de)	14.594	36,8%	36,8%
País Vasco	49.501	11,6%	34,5%
Rioja (La)	3.562	28,1%	28,1%
Ceuta	1.291	40,8%	40,8%
Melilla	1.511	35,7%	35,7%

Figura 1.2: Movilidad nacional de estudiantes universitarios por comunidad autónoma. Fuente: Ministerio de Universidades [2]

1.3.2. Políticas y sostenibilidad

El modelo de GoAway también está alineado con políticas nacionales como la Ley de Residuos y Suelos Contaminados [4], que fomenta la reutilización y gestión responsable de bienes. Este marco legislativo, que entra en vigor en 2025, busca minimizar el impacto ambiental y promover el reciclaje. GoAway contribuye directamente a estos objetivos al ofrecer una solución práctica para extender la vida útil del mobiliario.

1.3.3. Competencia y diferenciación

Aunque existen otras plataformas de alquiler en el mercado, GoAway se diferencia por su enfoque en estudiantes y su modelo de economía circular. Para este caso concreto, solo se ha encontrado esta página que podría cubrir esta necesidad:

Amuebla Rent [5]

Amuebla Rent se especializa en el alquiler de muebles de diseño para una variedad de usos, incluyendo la creación de sets de grabación, eventos temporales y actividades empresariales. Su modelo de negocio es B2C (Business-to-Consumer), donde es la empresa la que proporciona directamente los muebles a los clientes finales, ofreciendo servicios adicionales como home staging y diseño de interiores.

Por otro lado, GoAway se centra en un modelo C2C (Consumer-to-Consumer), facilitando el alquiler de muebles entre particulares, específicamente orientado a estudiantes universitarios que se trasladan temporalmente para cursar sus estudios. Este enfoque promueve la economía circular y la reutilización de recursos, ofreciendo soluciones asequibles y flexibles adaptadas a las necesidades de los estudiantes.

Además, mientras que Amuebla Rent opera principalmente en Madrid y se enfoca en proyectos inmobiliarios y eventos, GoAway está diseñada para tener un alcance más amplio, abarcando diversas ciudades universitarias y proporcionando una plataforma accesible para estudiantes en todo el país.

Característica	Amuebla Rent	Nuestra Aplicación
Modelo de negocio	B2C	C2C
Público objetivo	Propietarios y empresas	Estudiantes universitarios
Duración del alquiler	1 mes a 1 año	Semestre o curso académico
Servicios adicionales	Home staging, diseño	Plataforma de alquiler entre particulares
Alcance geográfico	Principalmente Madrid	Ciudades universitarias a nivel nacional

Tabla 1.1: Comparativa entre Amuebla Rent y nuestra propuesta

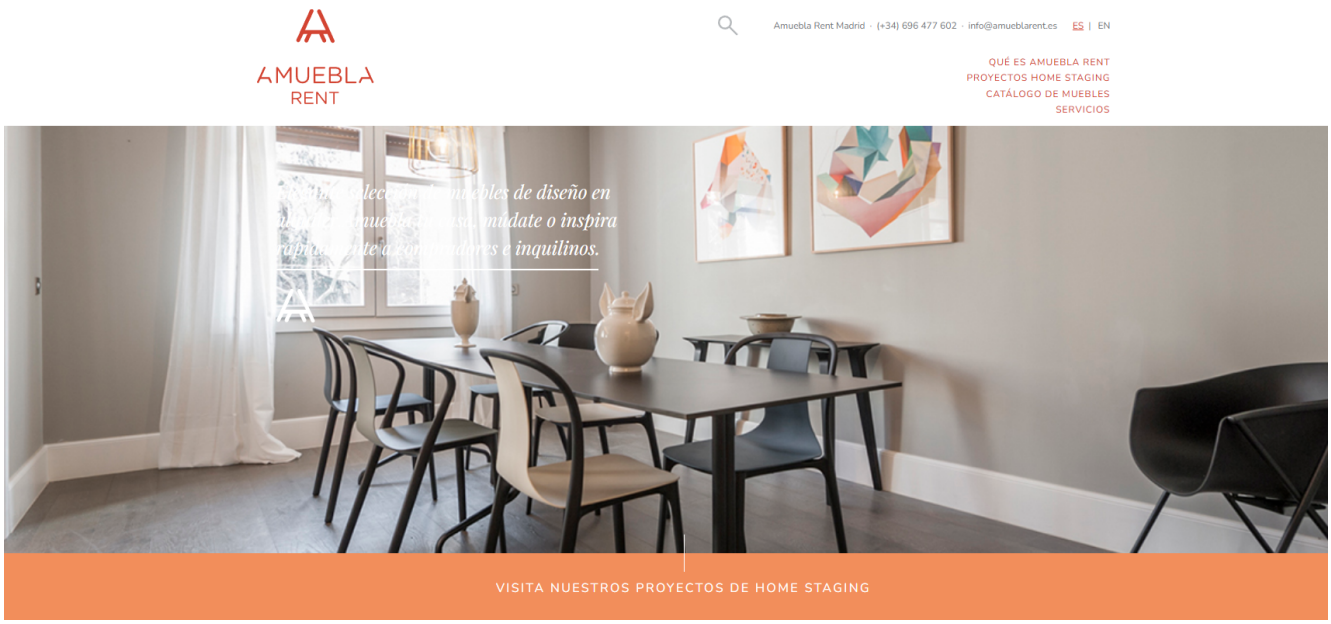


Figura 1.3: Aplicación de alquiler de muebles

1.4. Objetivos

1.4.1. Objetivos del desarrollo

El objetivo principal de este proyecto es el desarrollo de una aplicación web que permita la gestión de alquileres de muebles entre particulares, especialmente orientada a estudiantes universitarios que se trasladan temporalmente para cursar sus estudios. Esta plataforma pretende ser eficiente, accesible y escalable, con la posibilidad de evolucionar hacia una solución multiplataforma en futuras iteraciones.

Además, se plantean también una serie de objetivos específicos:

- Ofrecer una solución funcional que facilite la publicación, búsqueda y gestión de anuncios de muebles en alquiler.
- Garantizar la seguridad en la gestión de usuarios mediante sistemas de autenticación y control de permisos.
- Permitir la interacción fluida entre los usuarios mediante funcionalidades como solicitud de alquiler, aceptación, contacto y seguimiento.
- Implementar una arquitectura hexagonal que permita el mantenimiento y la ampliación de funcionalidades en el futuro.
- Sentar las bases para una futura extensión multiplataforma (por ejemplo, aplicaciones móviles).

1.4.2. Objetivos personales de aprendizaje

- **Fortalecer competencias en desarrollo full-stack:** El desarrollo de esta aplicación web proporcionará una formación integral en tecnologías clave como Angular para el frontend y Spring Boot para el backend, utilizando una arquitectura hexagonal que asegura una clara separación de responsabilidades, escalabilidad y facilidad de mantenimiento.

- **Adquirir experiencia en el ciclo completo de desarrollo de software:** Este proyecto permite abordar todas las fases del desarrollo de un sistema, desde su conceptualización inicial hasta la implementación y evaluación final. Esto incluye la aplicación de metodologías ágiles, el análisis de riesgos, la planificación estratégica y la integración de principios de diseño sostenible.

- **Capacitarse en la creación de soluciones escalables:** Mediante la implementación de la arquitectura hexagonal, se busca desarrollar una solución técnica robusta que pueda evolucionar en el tiempo y adaptarse a nuevos requisitos funcionales y tecnológicos.

- **Dominar herramientas de despliegue y control de versiones:** El uso de contenedores Docker para la ejecución del entorno, así como la gestión del código mediante Git y GitHub, permitirá adquirir competencias en el despliegue de aplicaciones, integración continua y colaboración en entornos profesionales.

1.5. Tecnologías empleadas

En esta sección se describen las tecnologías que se han utilizado para desarrollar el proyecto y la justificación de su uso.

1.5.1. Java

Java[6] es conocido por ser un lenguaje robusto y confiable. Permite el lenguaje multiplataforma, garantizando que pueda ejecutarse en diferentes entornos sin problemas de compatibilidad. Tiene gran capacidad para manejar datos eficientemente con una fácil mantenibilidad a largo plazo.

Para mejorar los aspectos de escalabilidad y modularidad en el código backend, se ha utilizado el framework Spring Boot[7], permitiendo que la lógica de acceso a datos sea más sencilla, gracias a su sistema de inyección de dependencias y patrones. Esto facilita la creación de aplicaciones web.

Figura 1.4: Logo de Spring Boot

1.5.2. Angular

Angular[8] es un framework basado en TypeScript[9] que nos permite desarrollar aplicaciones de una sola página. Se eligió Angular por su arquitectura basada en componentes, lo que nos permite crear un código más reutilizable y organizar correctamente el proyecto.

La curva de aprendizaje de Angular es más pronunciada en comparación con otros frameworks. Otro factor clave en su elección es que Angular promueve buenas prácticas como la separación de responsabilidades, la modularidad y el uso de servicios e inyección de dependencias, lo cual está alineado con el backend basado en arquitectura hexagonal. Aunque el frontend no sigue literalmente esta arquitectura, Angular permite estructurar el código de forma coherente con dichos principios, lo que facilita su integración y escalabilidad.

Figura 1.5: Logo de Angular

1.5.3. MySQL

La decisión de utilizar MySQL[10] como motor de base de datos es por su fiabilidad, rendimiento y facilidad de uso. Cuenta con una amplia documentación y una comunidad activa, debido a su popularidad.

Si bien PostgreSQL[11] es otra herramienta muy potente, se ha descartado por su curva de apren-

dizaje más pronunciada debido a su enfoque en estándares SQL más estrictos, con una configuración algo más compleja y su falta de uso en asignaturas anteriores.

Figura 1.6: MySQL

1.5.4. Docker

Docker[12] permite empaquetar la aplicación al completo en contenedores ligeros, incluyendo todo lo necesario del BackEnd, FrontEnd y base de datos. Su configuración es extremadamente sencilla, definiendo todo el entorno de la aplicación en un fichero docker-compose.yml y dos Dockerfile.

Al igual que con las plataformas anteriores, con Docker ya se ha trabajado con anterioridad, lo que facilita su configuración y puesta en funcionamiento.

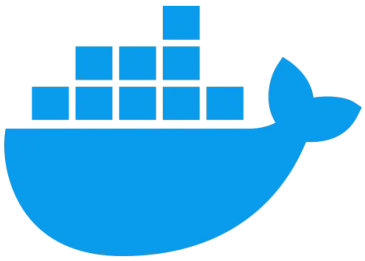


Figura 1.7: Docker

1.6. Metodología

Se ha optado por una metodología iterativa e incremental debido a su flexibilidad y adaptabilidad a los cambios, lo que es crucial en proyectos que pueden sufrir cambios durante su ejecución. Se divide el proyecto en una serie de iteraciones, siendo cada una de ellas funcional al finalizarlas para ser probada en cada ciclo.

Con este enfoque se consigue una mejora continua del producto. Se va desarrollando el producto de manera gradual, añadiendo un valor a cada versión. Se empezaría desarrollando las funciones básicas, como por ejemplo la gestión de los usuarios y, más adelante, ir añadiendo funcionalidades más complejas.

Una gran ventaja que tiene esta metodología es la facilidad para detectar y corregir los errores de manera temprana, reduciendo los riesgos y mejorando la calidad del producto.

Enfoques como, por ejemplo, el modelo cascada son más rígidos, debido a que requieren completar una fase antes de pasar a la siguiente y, pese a que Scrum también promueve la metodología ágil, es más formal que la metodología iterativa e incremental y se ajusta peor a equipos pequeños, con una gestión tan específica de sprints.

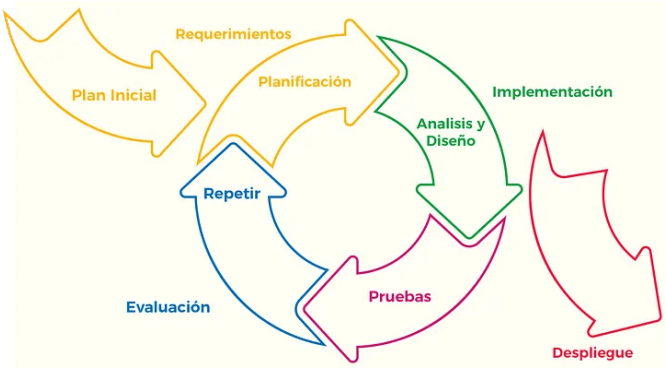


Figura 1.8: Metodología Iterativa e Incremental [13]

1.7. Estructura de la memoria

- **Introducción:** En esta sección se presenta el proyecto de forma general. Primero, se describe el contexto en el que surge la idea y las motivaciones que han llevado a su realización. A continuación, se hace un repaso del estado del arte, mencionando brevemente aplicaciones o soluciones similares. Luego, se exponen los objetivos principales que se pretenden alcanzar, junto con las tecnologías y la metodología que se utilizarán para desarrollar el proyecto. Finalmente, se ofrece una breve descripción de la estructura de la memoria, explicando las distintas secciones que la componen.
- **Planificación:** La planificación inicial del proyecto describe las tareas a realizar en cada iteración, considerando los riesgos potenciales y las herramientas necesarias, tanto de software como de hardware. Además, se define el entorno tecnológico utilizado para llevar a cabo el desarrollo. Al final, se compara esta planificación con la organización final del proyecto, analizando los ajustes realizados y cómo se ha llevado a cabo en la práctica.
- **Iteraciones:** En este capítulo se detalla el trabajo realizado en cada iteración, lo que permite contrastar con la planificación inicial y facilita el entendimiento del desarrollo del proyecto. Se ofrece una explicación del avance en cada iteración, describiendo con mayor detalle las tareas llevadas a cabo, los obstáculos encontrados y las soluciones implementadas.

- **Estado de la Aplicación:** Este capítulo está dedicado a la descripción del análisis, diseño y estructura del proyecto, acompañado de los diagramas que lo ilustran y documentan.
- **Pruebas:** Contiene la descripción de las pruebas que se han hecho.
- **Conclusiones:** Descripción de las conclusiones obtenidas al finalizar el proyecto.
- **Apéndices:** Documentación complementaria al proyecto que incluye los acrónimos, el cuestionario inicial, el cuestionario de evaluación, y los manuales de despliegue y uso de la aplicación.
- **Bibliografía:** Referencias bibliográficas utilizadas en el proyecto.

Capítulo 2

Planificación

2.1. Planificación inicial

En este capítulo se especifica la planificación temporal y las iteraciones que se han realizado cumpliendo los principios de la metodología iterativa e incremental vistos en la sección 1.6.

2.1.1. Gestión del tiempo

El Trabajo de Fin de Grado se ha organizado para cumplir con las 300 horas estipuladas por la normativa universitaria, excediendo en 20 horas. El proyecto comenzó el 23 de septiembre de 2024 y se espera finalizar el 2 de marzo de 2025. Se trabajará 5 días a la semana, dedicando aproximadamente 3,2 horas diarias, lo que asegura un total de 16 horas semanales. Este ritmo de trabajo permitirá alcanzar las 320 horas en el tiempo estipulado. En caso de ser necesario, se podrán dedicar horas adicionales durante los fines de semana para compensar posibles retrasos o cubrir cualquier imprevisto.

El período total de trabajo se extiende a aproximadamente 20 semanas, proporcionando el tiempo necesario para abordar cada fase del proyecto de manera adecuada. Estas incluyen una fase inicial de planificación, investigación de proyectos similares, análisis del software base y la selección de los dispositivos donde se implementará el proyecto.

Las reuniones con el profesor se realizarán conforme a los objetivos alcanzados o las dudas que surjan a lo largo del desarrollo. Este sistema garantiza un seguimiento personalizado y adaptado a las necesidades del proyecto, permitiendo ajustes en función del progreso y resolviendo posibles inconvenientes de manera eficiente. Este enfoque flexible asegura que el proyecto se desarrolla según lo planificado y que los objetivos se logran dentro del tiempo revisado.

El proyecto se llevará a cabo utilizando la metodología iterativa e incremental, estructurándose en las siguientes iteraciones:

- **Primera iteración:** *Duración estimada: 3 semanas.* Durante esta fase inicial, se desarrollará el

documento introductorio del proyecto, incluyendo el contexto, la motivación y los objetivos. Se investigará el estado actual del problema y se analizarán aplicaciones similares para identificar fortalezas y carencias. Además, se definirá una metodología ágil adaptada y se estructurará la planificación de las iteraciones futuras.

- **Segunda iteración:** *Duración estimada: 5 semanas.* Se realizará una investigación exhaustiva sobre la arquitectura hexagonal para consolidar su comprensión y aplicación en el proyecto. Además, se elaborarán los casos de uso, el modelo de dominio y los diagramas de actividad. También se documentará la estimación de costes y se definirá el entorno de trabajo.

- **Tercera iteración:** *Duración estimada: 7 semanas.* Implementación completa del backend. Esto incluirá las capas de dominio, aplicación, adaptadores de entrada/salida y la integración de todos los componentes según la arquitectura hexagonal. Durante esta fase, se llevará acabo además la documentación relacionada.

- **Cuarta iteración:** *Duración estimada: 6 semanas.* Desarrollo del frontend, incluyendo su integración con los adaptadores de entrada. Validación completa del sistema mediante un ejemplo de pruebas de integración. Además se realizará la documentación final del proyecto, añadiendo lo relacionado con el desarrollo front-end, las pruebas y la parte final de la documentación.

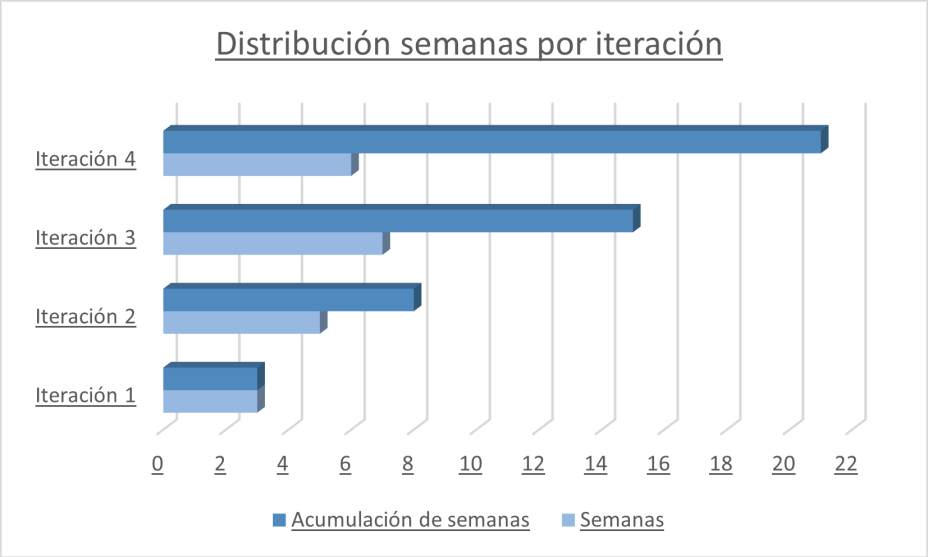


Figura 2.1: Distribución semanal por cada iteración

2.1.2. Análisis de riesgos

El análisis de riesgos detalla los posibles problemas que podrían surgir durante el desarrollo del proyecto. A continuación, se describen los riesgos principales con su identificación, un título breve, una descripción detallada y las probabilidades e impactos estimados de cada uno.

Riesgos Identificados

ID	Riesgo	Descripción	Probabilidad	Impacto
1	Enfermedad o indisposición	Situaciones de enfermedad que afecten al ritmo de trabajo y reduzcan la capacidad para avanzar en el proyecto.	Media	Alto
2	Carga de trabajo elevada	Sobrecarga laboral que dificulte dedicar tiempo al TFG, generando retrasos en el cronograma.	Media	Medio
3	Falta de experiencia con la arquitectura hexagonal	Desconocimiento técnico que dificulte implementar correctamente la arquitectura hexagonal, aumentando el riesgo de errores.	Media	Alto
4	Error en la elección de tecnologías	Selección de tecnologías inadecuadas que no se ajusten a las necesidades del proyecto, generando retrabajos.	Baja	Medio
5	Pérdida de datos por fallo del equipo	Fallos técnicos que provoquen la pérdida de información crítica para el proyecto.	Baja	Alto
6	Retraso por problemas con el tutor	Dificultades en la disponibilidad del tutor para resolver dudas o validar entregas, generando retrasos en las tareas.	Media	Medio
7	Incompatibilidad entre módulos	Problemas de integración técnica entre los módulos desarrollados, afectando al correcto funcionamiento del sistema.	Media	Medio
8	Cambios en los requisitos del proyecto	Modificaciones inesperadas en los requisitos que afecten a las funcionalidades ya desarrolladas.	Media	Alto

Tabla 2.1: Detalle de riesgos identificados.

Medidas de Mitigación y Prevención

Para cada riesgo identificado, se han propuesto medidas específicas de mitigación y prevención. Estas medidas están orientadas a reducir el impacto en caso de que el riesgo se materialice (mitigación) y a evitar que el riesgo ocurra (prevención).

ID	Título	Medidas de Mitigación	Medidas de Prevención
1	Enfermedad o indisposición	Reorganizar tareas para recuperar el tiempo perdido. Priorizar actividades críticas.	Planificar margen adicional en el cronograma.
2	Carga de trabajo elevada	Dividir la carga en bloques más pequeños. Planificar descansos para evitar el agotamiento.	Organizar mejor el tiempo dedicando horas específicas al TFG. Establecer límites para evitar sobrecarga laboral.
3	Falta de experiencia con la arquitectura hexagonal	Realizar pruebas y crear pequeños prototipos previos. Consultar documentación oficial y ejemplos.	Estudiar la arquitectura previamente y realizar un esquema de diseño para minimizar errores.
4	Error en la elección de tecnologías	Validar tecnologías con pruebas simples antes de adoptarlas. Consultar con el tutor y revisar buenas prácticas.	Investigar y comparar tecnologías en la fase inicial del proyecto. Elegir herramientas con comunidades activas y documentación.
5	Pérdida de datos por fallo del equipo	Realizar copias de seguridad diarias y usar almacenamiento en la nube. Verificar la integridad de los datos periódicamente.	Utilizar herramientas confiables y mantener sistemas de respaldo automáticos.
6	Retraso por problemas con el tutor	Asegurar reuniones programadas y adelantar temas importantes con tiempo. Tener un plan de contingencia para consultas críticas.	Establecer un cronograma claro de revisiones con el tutor desde el inicio del proyecto.
7	Incompatibilidad entre módulos	Realizar pruebas de integración continuas. Validar el funcionamiento de cada módulo antes de integrarlo en el sistema final.	Diseñar interfaces claras entre módulos y utilizar estándares de desarrollo.
8	Imposibilidad de dedicar las horas planificadas por carga laboral extra	Compensar las horas no trabajadas entre semana dedicando tiempo adicional al TFG durante los fines de semana.	Reajustar la planificación semanal para adaptarse a imprevistos y priorizar horas del TFG.
9	Cambios en los requisitos del proyecto	Revisar periódicamente los requisitos con el tutor o partes interesadas para minimizar malentendidos. Priorizar los cambios que aporten mayor valor al proyecto.	Documentar y validar los requisitos desde el inicio del proyecto para evitar confusiones.

Tabla 2.2: Medidas de mitigación y prevención.

2.2. Entorno de trabajo

Esta sección tiene como finalidad detallar tanto los recursos empleados como las especificaciones de los dispositivos que se hayan utilizado para desarrollar el proyecto.

2.2.1. Recursos empleados

Se describen las herramientas y software utilizados a lo largo del desarrollo del proyecto, especificando aplicaciones, plataformas y servicios que hayan facilitado las tareas.

- **Brave[14]**: Navegador web utilizado en múltiples circunstancias.
- **Astah Profesional[15]**: Herramienta de modelado UML para la creación de diagramas de clases, casos de uso y más. Utilizando la licencia de la escuela.
- **Visual Studio Code[16]**: Editor de código ligero y extensible, con soporte para múltiples lenguajes y herramientas de depuración.
- **Overleaf[17]**: Plataforma colaborativa para la escritura y edición de documentos LaTeX en línea.
- **Microsoft Excel[18]**: Software de hojas de cálculo para análisis de datos, creación de gráficos y gestión de información.
- **Microsoft Teams[19]**: Herramienta de colaboración para videollamadas, chats y gestión de equipos en entornos laborales.
- **GitHub[20]**: Plataforma para control de versiones en proyectos de software utilizando Git.
- **ChatGPT[21]**: Asistente virtual basado en IA para la generación de ideas, corrección de textos y resolución de dudas técnicas.
- **Postman[22]**: Herramienta para pruebas y desarrollo de APIs, permitiendo enviar, recibir y validar solicitudes HTTP.
- **MySQL Workbench[23]**: Herramienta de administración de bases de datos gratuita. Permite administrar y gestionar bases de datos de manera gráfica y sencilla.
- **Docker[12]**: Plataforma que empaqueta aplicaciones en contenedores aislados, facilitando su despliegue y ejecución en diferentes entornos

2.2.2. Especificaciones del ordenador de trabajo

Se detallan las características técnicas del equipo utilizado durante el desarrollo del proyecto, incluyendo información sobre el hardware y el sistema operativo utilizados.

Componente	Especificación
CPU	Ryzen 5 7600 X 4.7 GHz
Placa Base	Gigabyte B650M D3HP
GPU	PowerColor AMD Radeon RX 7700 XT Fighter OC 12GB GDDR6
RAM	Corsair Vengeance RGB DDR5 6000MHz 32GB
Caja	Nox Hummer Quantum Cristal Templado USB 3.0 ARGB
Refrigeración	Noctua NH-U12S chromax.black
Fuente de Alimentación	Nox Hummer 750w 80 plus gold
Disco Duro	Kioxia Exceria G2 Unidad SSD 1T NVMe M.3

Tabla 2.3: Especificaciones del ordenador de trabajo

Para una mayor comodidad y eficiencia durante el desarrollo del proyecto, se han utilizado dos monitores conectados a este equipo:

- **Monitor principal**: MSI con pantalla de 27” con resolución 1920x1080.
- **Monitor secundario**: LG con pantalla de 24” con resolución 1920x1080.

2.3. Estimación de costes

En este apartado se van a desglosar los gastos del proyecto. Entre los costes se podría hacer una diferenciación entre 4. Costes en cuanto a recursos humanos, software, hardware y costes del espacio de trabajo.

2.3.1. Costes laborables

Dado que este Trabajo Fin de Grado no implica una remuneración directa, se puede calcular el costo equivalente de un desarrollador junior en un contexto profesional. El sueldo bruto anual promedio para un desarrollador *full-stack* junior es de aproximadamente 19,700 € [24]. Esto corresponde a un salario mensual bruto de:

$$\text{Salario bruto mensual} = \frac{19,700\,€}{12\,\text{meses}} \approx 1,641.67\,€$$

Asumiendo una jornada laboral de 40 horas semanales y un total de 160 horas al mes, el costo por hora sería:

$$\text{Costo por hora} = \frac{1,641.67\,€}{160\,\text{h}} \approx 10.26\,€/\text{h}$$

Si el proyecto requiere 320 horas de trabajo, el costo total a asumir sería:

$$\text{Costo total} = 320\,\text{h} \times 10.26\,€/\text{h} = 3,283.20\,€$$

2.3.2. Costes de software

En cuanto a los costes de software, el único programa que precisa de licencia para ser utilizado es el Astah, la cual tiene un precio de 11,99 $ al mes, que corresponde a aproximadamente 11,50 € al mes [25]. Esto daría un total de 46 € por un total de 4 meses de trabajo. Pero en este caso, la licencia la obtenemos por la UVA de manera gratuita.

2.3.3. Costes de hardware

Se utilizarán recursos personales para el desarrollo, con los siguientes costos estimados:

- **Ordenador:** PC construido a piezas = 1,170.63 €
- **Pantalla 1:** Pantalla MSI 27- 170 €
- **Pantalla 2:** Pantalla LG 24- 140 €

Dado que estos dispositivos ya han sido adquiridos con anterioridad y tienen múltiples usos, se calcula su coste amortizado para reflejar un coste más preciso relacionado con este proyecto. Utilizando una vida útil de 7 años para el ordenador y 4 años para las pantallas (con un uso promedio de 8 horas diarias), las fórmulas que se van a utilizar son las siguientes:

$$\text{Coste amortizado por hora} = \frac{\text{Coste total}}{\text{Vida útil total (Horas)}}$$

Y para obtener el coste total por dispositivo amortizado asociado al proyecto:

$$\text{Coste Amortizado Proyecto} = \text{Coste Amortizado por Hora} \times \text{Horas del Proyecto}$$

- **Ordenador:** 18,33 € (320 horas a 0,057 €/hora).
- **Pantalla MSI 27'':** 4,66 € (320 horas a 0,015 €/hora).
- **Pantalla LG 24'':** 3,84 € (320 horas a 0,012 €/hora).

Estos valores reflejan únicamente el coste asociado al uso específico de los dispositivos durante las 320 horas del proyecto [26].

Esto nos da un total de coste amortizado de hardware de 26,83 €

2.3.4. Costes del espacio de trabajo

Dado que el proyecto se desarrollará desde casa, los gastos incluirán el consumo de luz e internet, con un coste estimado de 40 € mensuales. Al proyectarse una duración total de 5 meses, el gasto total en espacio de trabajo será de 200 €.

2.3.5. Resumen de costes

En la siguiente tabla se resumen los costos totales del proyecto:

Concepto	**Coste (€)**
Costes Laborales	3.283,20
Costes de Software	0,00
Costes de Hardware (Amortizado)	26,83
Costes del Espacio de Trabajo	200,00
Total	**3,510.03**

Tabla 2.4: Resumen de Costes del Proyecto

2.4. Comparativa entre lo planificado y lo realizado

En esta sección se especifica cómo ha quedado finalmente la planificación del proyecto respecto a la planificación inicial descrita en la sección 2.1, señalando los contratiempos y retrasos que han podido ir surgiendo.

Durante el desarrollo del proyecto, la planificación inicial ha sufrido algunos ajustes debido principalmente a que se subestimó el tiempo necesario para la implementación del backend 3.3 y a la incorporación de pequeñas funcionalidades adicionales que no habían sido previstas inicialmente, como mejoras en la seguridad y ajustes en los endpoints 3.3. Además, surgieron imprevistos relacionados con la vida laboral, en particular, la realización de horas extras en mi trabajo. Este riesgo se tenía contemplado en la tabla de riesgos 2.1, lo que ha limitado considerablemente el tiempo disponible para dedicarle a este proyecto.

La tercera iteración 3.3, correspondiente a la implementación del backend, se completó dentro del plazo estimado. Sin embargo, al comenzar con el desarrollo del frontend en la cuarta iteración 3.4, surgieron una serie de ajustes imprevistos que afectaron a ambos componentes del sistema. Al hacer la integración de ambos lados, se detectaron pequeños errores en algunos controladores y casos de uso del backend que, pese a ser funcionales, no se ajustaban completamente a las necesidades reales del frontend.

Esto obligó a modificar ciertas rutas y comportamientos, como las requests y las responses, para garantizar una integración coherente y fluida, así como la creación de nuevos endpoints que inicialmente no se habían considerado necesarios para el backend, pero que resultaron imprescindibles para el co-

rrecto funcionamiento del frontend. Estos cambios, aunque no han resultado excesivamente complejos, requirieron tiempo adicional no contemplado en la planificación inicial, lo que provocó un ligero retraso en la finalización del frontend y su correspondiente integración con el sistema completo.

En total, el proyecto estaba planificado para finalizar el 2 de marzo de 2025, pero debido a las circunstancias mencionadas, se prolongó aproximadamente un mes adicional. Finalmente, y previendo posibles ajustes o correcciones adicionales, se ha establecido un margen adicional de una semana, estimando la finalización definitiva del proyecto aproximadamente para la primera semana de mayo de 2025.

A pesar de estos contratiempos, la metodología ágil adoptada desde el principio 3.1 ha permitido gestionar estos cambios con cierta flexibilidad, lo que ha facilitado la adaptación a los imprevistos surgidos durante el desarrollo del proyecto. Las reuniones periódicas y la comunicación constante en caso de dudas o problemas con el tutor del proyecto también han sido de gran ayuda, permitiendo reajustes rápidos y efectivos ante cualquier eventualidad o incertidumbre 2.1.1.

Finalmente, aunque el proyecto no se ha completado en el tiempo inicialmente estipulado, se ha logrado finalizar con una desviación de algo más de un mes respecto al tiempo inicialmente marcado. Este retraso ha sido producido tanto por los ajustes técnicos necesarios durante la integración del frontend y backend, como por circunstancias personales puntuales, como por ejemplo, enfermedades o ausencias inevitables.

A pesar de estos problemas en el desarrollo del proyecto, se han implementado todas las funcionalidades previstas, cumpliendo con los objetivos establecidos y entregando un producto completo y funcional.

Capítulo 3

Descripción de las iteraciones

3.1. Iteración 1

Esta primera fase del proyecto corresponde con el análisis de herramientas disponibles y con la elección final de las mismas (ver secciones 1.5 y 2.2). En este período inicial se ha querido asentar una base firme y correcta de cómo va a ser el desarrollo completo del proyecto. Para ello, se comenzó con la redacción del documento, en especial en la introducción, explicando la motivación, los objetivos tanto personales como profesionales.

Además, se realizó un estudio de webs y aplicaciones con funcionalidad similar, comparando las soluciones que estas plataformas proponen, analizando sus ventajas y limitaciones. Este proceso ayudó a establecer unos objetivos claros y a facilitar la alineación con una necesidad detectada.

Se optó por una metodología de trabajo ágil, enfocada en el desarrollo individual. Esto facilita poder realizar ajustes continuos a medida que se van desarrollando las necesidades del sistema. Asimismo, se planificaron las iteraciones que iban a ejecutarse, estableciendo hitos y recursos técnicos para cada fase, lo que garantiza un avance ordenado y eficaz.

También se definió el conjunto de tecnologías y la arquitectura que se utilizarán a lo largo del proyecto. Se escogió Java con Spring Boot como tecnología principal para el backend, implementando una arquitectura hexagonal que permite una separación clara entre el dominio y los adaptadores. Esta elección, que se detalla previamente en la sección 1.5 y más adelante en la sección 4.2.1, responde a la necesidad de construir un sistema modular, escalable y fácilmente testeable.

3.2. Iteración 2

Durante esta iteración, el objetivo principal fue asentar una base sólida de la arquitectura hexagonal, comprendiendo completamente sus principios y cómo aplicarlos de manera efectiva al proyecto. Esta arquitectura es también conocida como arquitectura de puertos y adaptadores, organizando el sistema

en capas independientes, permitiendo que la lógica de negocio y los casos de uso sean completamente autónomos respecto a tecnologías externas como bases de datos, frameworks o APIs. Esta separación por capas no solo facilita el mantenimiento del código, sino que también mejora la escalabilidad y permite futuras adaptaciones sin comprometer la integridad.

Para reforzar este aprendizaje, se recurrió a dos recursos clave que proporcionaron tanto una base teórica como una guía práctica:

- **Platzi: Arquitectura hexagonal [27]**, que ofrece una visión teórica detallada de los conceptos fundamentales y cómo fomenta un diseño desacoplado y modular.
- **LeanMind: Arquitectura hexagonal en Spring [28]**, un recurso práctico que ayuda a visualizar un ejemplo de código, para luego aplicarlo correctamente.

Además, se desarrollaron los principales artefactos de diseño que servirán de base para el sistema:

- **Casos de uso:** Identificación y descripción detallada de las interacciones entre el usuario y el sistema, definiendo los flujos principales y las tareas necesarias para cumplir con los objetivos funcionales.
- **Modelo de dominio:** Creación de las entidades con sus atributos y las relaciones y multiplicidades que tienen.
- **Diagramas de actividad:** Representación visual de los flujos principales del sistema, ayudando a la comprensión de como se conectan los componentes y como es el flujo de los casos de uso.

Adicionalmente, se realizó una estimación de costes teniendo en cuenta los recursos necesarios, como herramientas de desarrollo, licencias y el tiempo estimado para cada fase del proyecto. También se definió un entorno de trabajo optimizado, que incluyó el uso de herramientas como *Visual Studio Code* [16], *Docker* [12], *MySQL Workbench* [23] y *Postman* [22]. Todas estas herramientas fueron configuradas para trabajar de manera integrada y siguiendo con los principios de la arquitectura hexagonal.

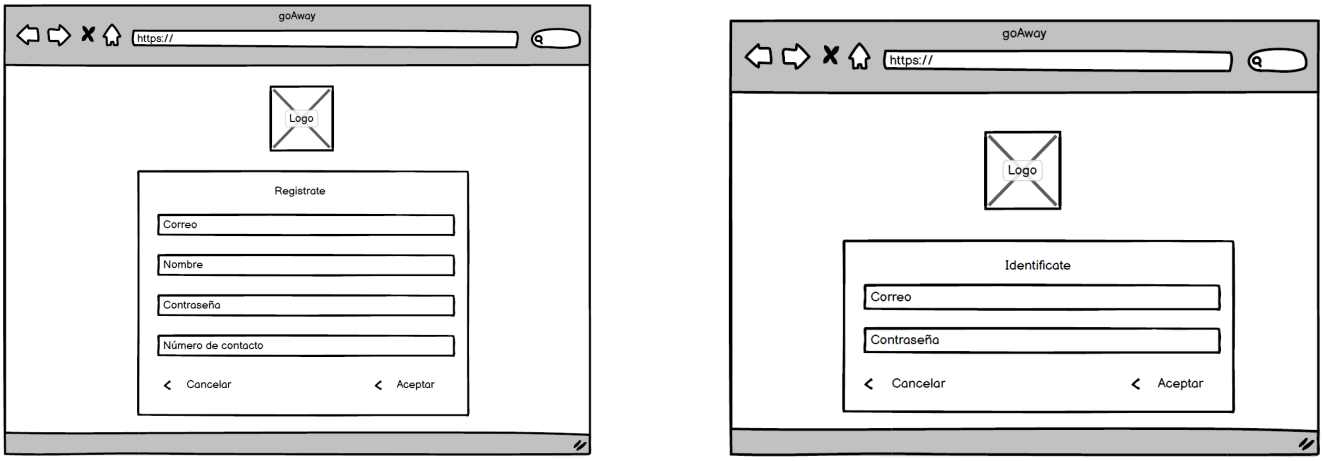
3.3. Iteración 3

En esta tercera iteración, se ha realizado el código del backend con Spring Boot, además de la configuración de la base de datos en MySQL. Paralelamente, se han ido creando los bocetos de la interfaz con la herramienta Balsamiq [29].

Lo primero a realizar ha sido la configuración de la base de datos en MySQL, donde se definió el modelo de datos, creando las tablas necesarias para la aplicación, utilizando la herramienta MySQL Workbench, en donde se generaron los scripts necesarios para su creación y relación, siguiendo un diagrama de entidad-relación (ERD). Una vez configurada, se procedió a habilitar la conexión entre Spring Boot y la base de datos a través del archivo de propiedades (application.properties), especificando las credenciales para acceder, así como la URL de conexión a la base de datos y su puerto.

Una vez terminada la configuración de la base de datos y su conexión, se llevó a cabo la implementación del backend con Spring Boot siguiendo la arquitectura hexagonal, donde se separaron correctamente las capas entre dominio, aplicación e infraestructura. En la parte de dominio se definieron las entidades y casos de uso principales y en la parte de infraestructura los controladores REST necesarios. Además, se añadieron las dependencias necesarias en el archivo pom.xml, como, por ejemplo, las librerías de conexión MySQL, el starter web de Spring y demás componentes.

Con toda la configuración ya terminada, se comenzó con el módulo de autenticación, el cual valida las credenciales de usuario y, tras un inicio correcto, genera un token JWT (JSON Web Token). Este token, posteriormente, se utiliza para asegurar las rutas y endpoints restringidos. De esta manera, las peticiones que requieran una autenticación deberán presentar el token para que el servidor verifique la validez.



(a) Boceto registro

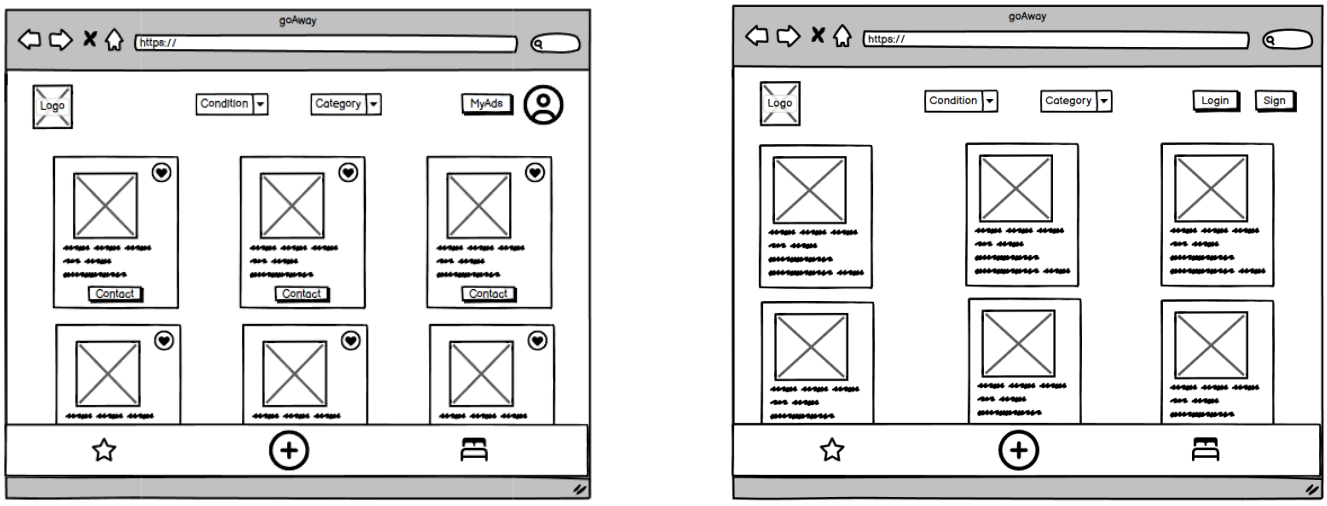
(b) Boceto inicio sesión

Figura 3.1: Bocetos de registro e inicio de sesión

En la figura 3.1a se presenta el boceto para el registro de usuario. Un registro sencillo, en el que se le solicita al usuario un correo, que servirá para la comunicación entre los usuarios, un nombre, una contraseña y, por último, un número de teléfono.

Para el inicio de sesión que se muestra en la figura 3.1b únicamente se necesita el correo y la contraseña con los que se registró el usuario.

Una vez se comprueba que el usuario se ha registrado o iniciado sesión correctamente, se recibe el token JWT con el que podrá acceder a toda la funcionalidad implementada.



(a) Boceto lista principal

(b) Boceto lista principal no registrado

Figura 3.2: Comparativa de la vista principal según el estado de sesión

En el boceto de la figura 3.2a se ve la diferencia que hay entre la lista principal de anuncios estando registrado y sin estarlo. Sin estar registrado, el usuario puede únicamente aplicar los filtros, pero no puede ni guardar un anuncio en favorito ni solicitar el alquiler de un anuncio ni utilizar la funcionalidad que se encuentra en la barra inferior.

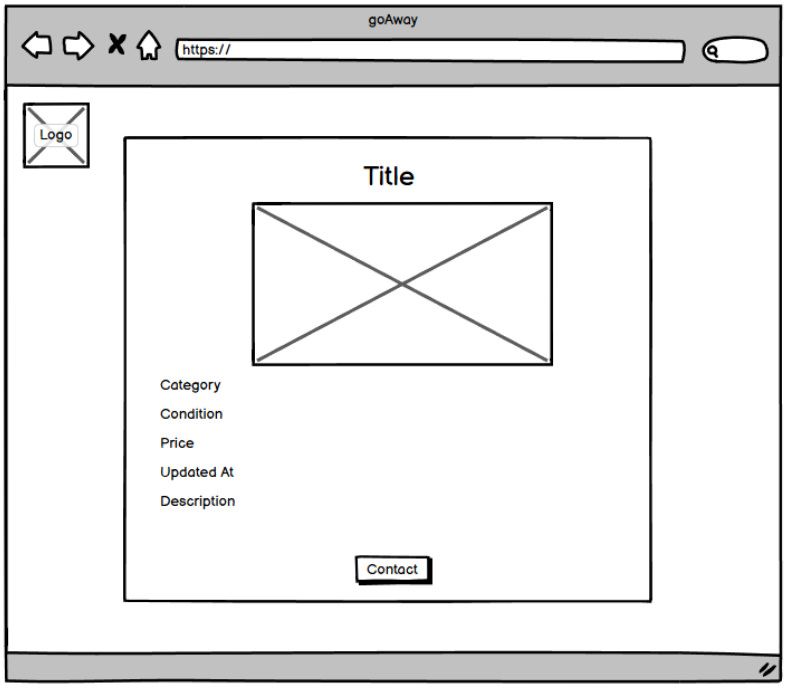


Figura 3.3: Boceto anuncio unitario

En la figura 3.3 se representa la visualización de un anuncio en concreto, ya sea de la lista principal de anuncios, de tus anuncios favoritos o bien, de las transacciones pendientes que tiene un usuario abiertas.

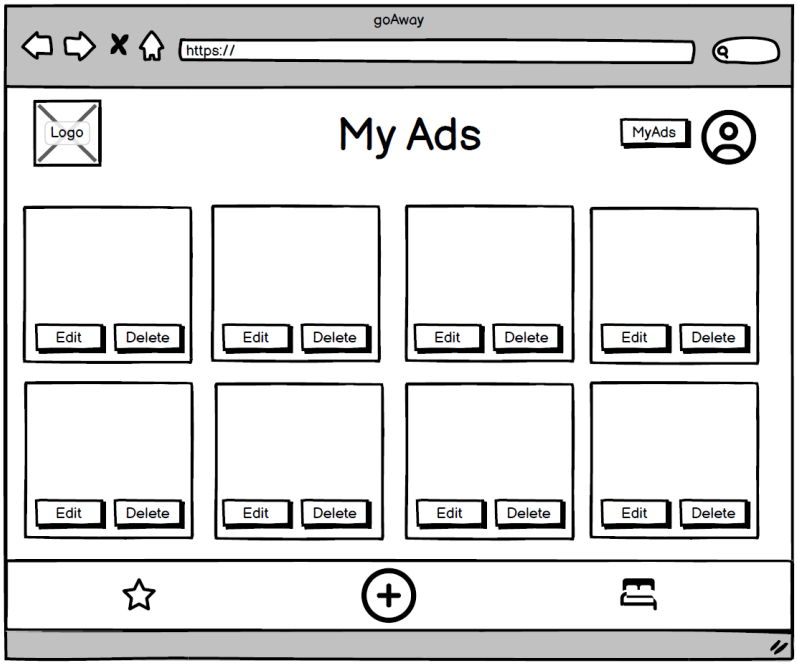


Figura 3.4: Boceto lista mis anuncios

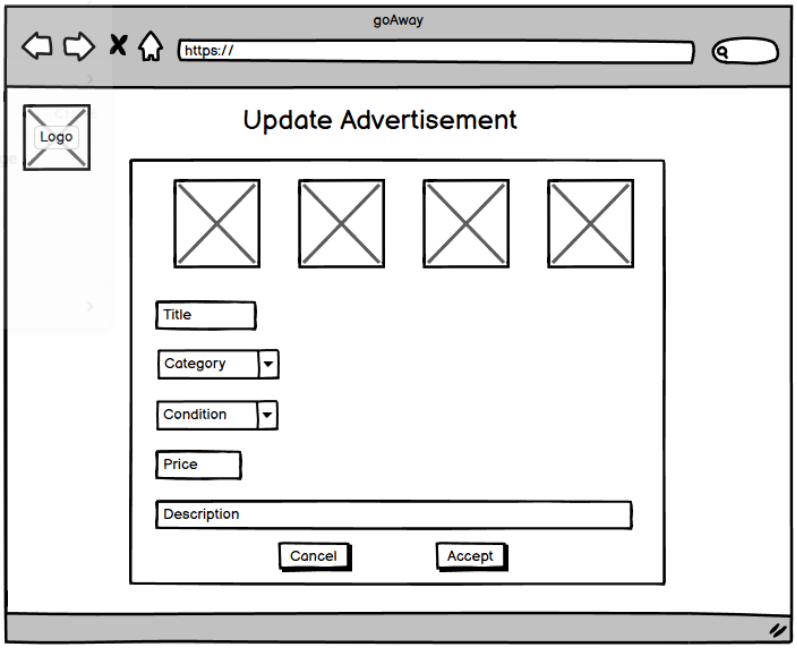


Figura 3.5: Boceto editar anuncio

En la lista de anuncios que ha creado el usuario que se muestra en la figura 3.4, se ha implementado la funcionalidad de poder eliminar o poder editar cualquiera de los anuncios que haya creado. La funcionalidad de editar, como se ve en la figura 3.5 se podrá editar el título, la categoría, la condición, el precio y la descripción de un anuncio previamente creado. A esta lista se accede a través del botón que se encuentra en la parte superior derecha, que aparece solo si el usuario está registrado.

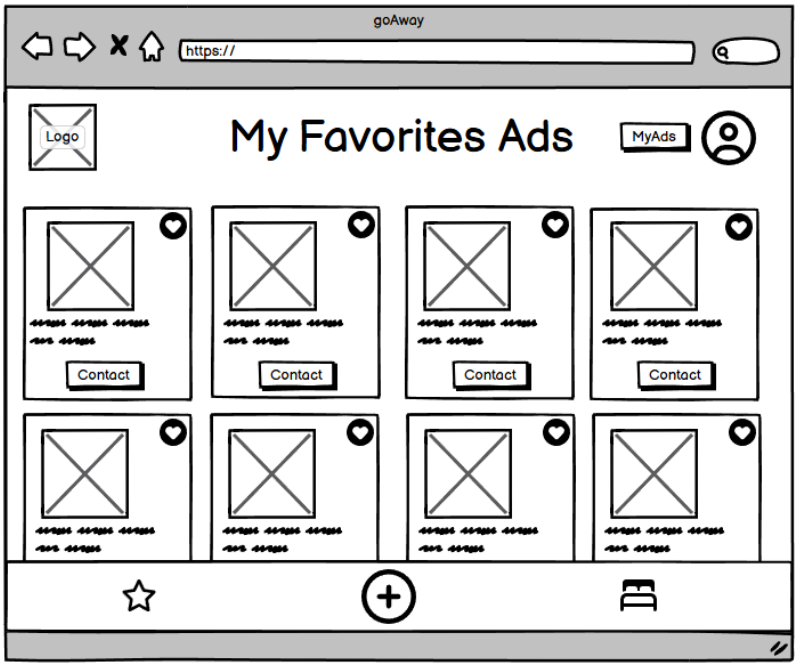


Figura 3.6: Boceto lista favoritos

Interactuando con la estrella que se encuentra en la parte inferior izquierda, se accede a la funcionalidad que muestra la lista de los anuncios favoritos de un usuario registrado. Como se puede ver en la figura 3.6 el usuario podrá eliminar un anuncio de favorito y realizar un alquiler desde esta misma lista.

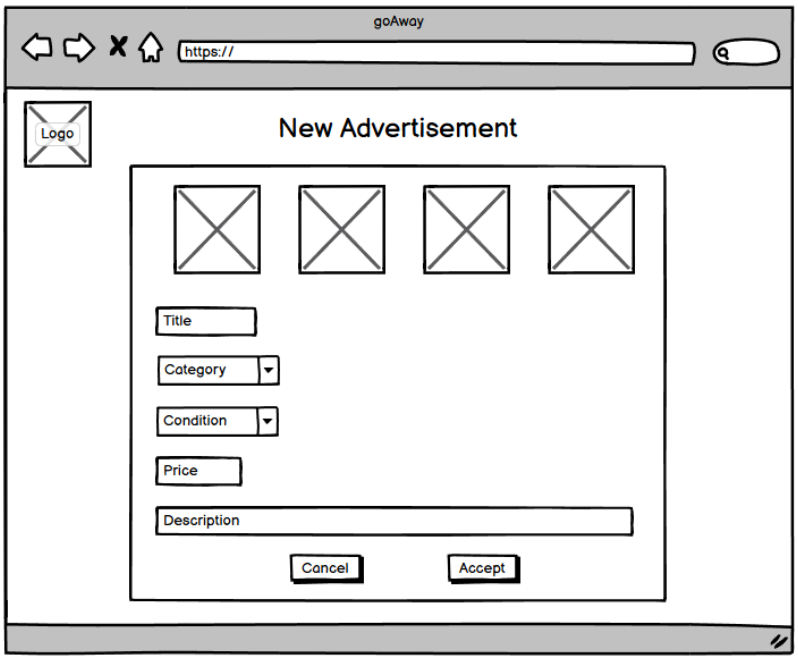
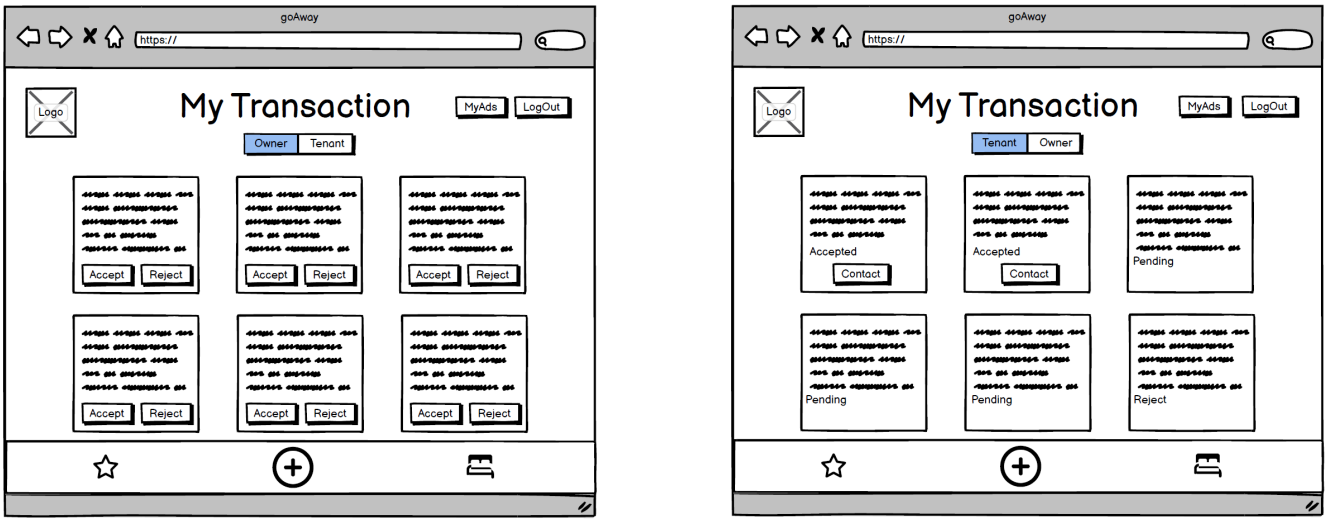


Figura 3.7: Boceto nuevo anuncio

Para la creación de un anuncio será necesario, como se ve en la figura 3.7 añadir un título, una categoría, la condición en la que se encuentra, el precio que cuesta al mes el alquiler, una descripción e imágenes.



(a) Boceto lista transacciones como propietario

(b) Boceto lista transacciones como arrendatario

Figura 3.8: Comparativa de la lista de transacciones según el rol de usuario

El usuario podrá visualizar la lista de las transacciones que tiene siendo arrendatario y arrendador. Por un lado, la lista como arrendador tiene la funcionalidad de aceptar o denegar las peticiones de alquiler que recibe de sus anuncios. Por otro lado, lista como arrendatario le aparecerá el estado que tiene la petición de alquiler que ha hecho sobre un anuncio de otro usuario. En caso de que una petición haya sido aceptada, podrá ponerse en contacto con el propietario del alquiler para terminar con el proceso de transacción que aparece en la figura 3.9, mostrando una etiqueta para enviar un mensaje por correo electrónico al propietario.

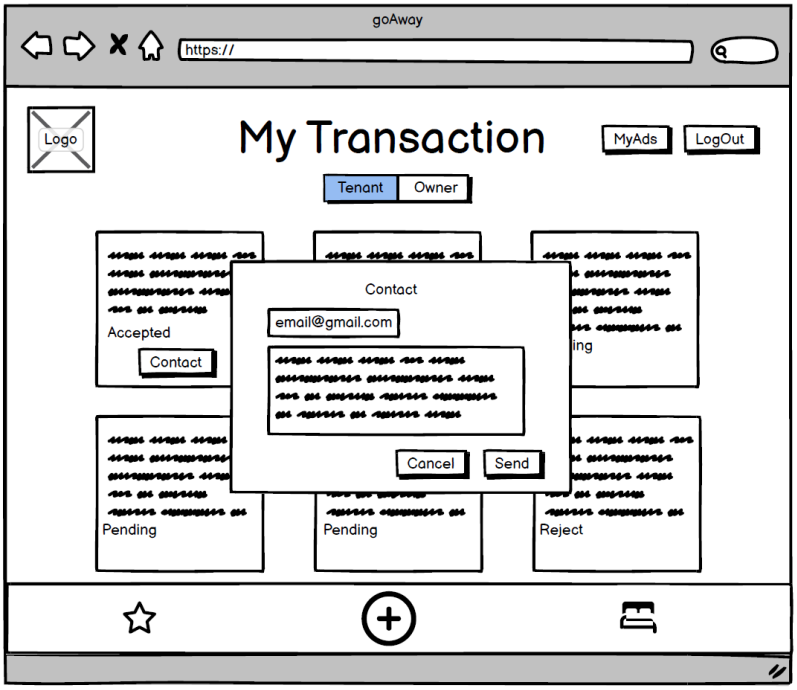


Figura 3.9: Boceto contacto con propietario

A medida que se iba avanzando con la funcionalidad y terminando los end points se ha ido comprobando que todo funcione correctamente, de manera unitaria y de manera conjunta usando la herramienta Postman[22]. Hasta que una funcionalidad no estaba terminada y validada, no se pasaba a la siguiente, para evitar acumulación de código incorrecto o sin finalizar.

3.4. Iteración 4

Tras finalizar la implementación del *backend* con Spring Boot, se realizó el código del *front-end* en **Angular 19**, con el propósito de conectar la lógica del sistema a una interfaz más visual y accesible para el usuario.

Lo primero fue crear la estructura del proyecto utilizando la CLI de Angular, definiendo los módulos, componentes y rutas que sostendrían la navegación de la aplicación. Seguidamente, se configuraron los *services* necesarios para comunicarse con los *endpoints* del *backend*, de modo que cada componente pudiera acceder a la información proporcionada por los controladores *REST* sin acoplar la lógica de negocio en la interfaz.

En cuanto a la seguridad, se optó por un *interceptor* para incluir el token JWT (*JSON Web Token*) en las cabeceras de las peticiones HTTP que requiriesen autenticación, almacenando el token en Session storage. De esta manera, el usuario solo podrá realizar cierta funcionalidad si está identificado en la plataforma.

Para los formularios de registro, inicio de sesión y creación de anuncios, se empleó el módulo de *Reactive Forms* de Angular. Esto permitió disponer de validaciones tanto síncronas como asíncronas, así como la posibilidad de mostrar mensajes claros ante posibles errores de formato o campos obligatorios no rellenados.

Se realizaron revisiones manuales en el navegador para asegurar que la comunicación entre el *front-end* y el *backend* se comportase correctamente. Con ello, la fase de integración visual de la aplicación quedó completada, ofreciendo al usuario una experiencia coherente y fluida al interactuar con todas las funcionalidades desarrolladas.

Finalmente, se han realizado unos códigos de test de prueba de uno de los casos de uso, para dejar el camino abierto a realizar más tests tanto para el backend como para el frontend.

Capítulo 4

Estado final de la aplicación

4.1. Análisis

4.1.1. Análisis de requisitos

En esta sección se desarrollan los diferentes requisitos, un proceso fundamental en el desarrollo que nos permite identificar y documentar necesidades de los usuarios. Proporcionando una base clara y completa para el diseño y desarrollo del sistema, permitiendo detectar posibles problemas o limitaciones tempranas.

Requisitos funcionales

Los requisitos funcionales son aquellos que expresan las capacidades esenciales que debería de tener el sistema para cumplir con el propósito inicial. Se corresponden con acciones específicas que el usuario puede realizar dentro de la aplicación, como la identificación y el registro.

Tabla 4.1: Requisitos Funcionales

ID	Nombre	Descripción
RF01	Identificación en el sistema	El sistema deberá permitir al usuario ya registrado identificarse en el sistema.
RF02	Registro de usuarios	El sistema deberá permitir a los usuarios registrarse en la plataforma mediante correo electrónico y contraseña.
RF03	Publicación de anuncio	El sistema deberá permitir a los usuarios publicar muebles para alquiler, especificando detalles relevantes.

RF04	Mostrar lista de anuncios	El sistema deberá permitir a los usuarios visualizar la lista de los anuncios que hayan publicado previamente.
RF05	Eliminación de anuncios	El sistema deberá permitir a los usuarios eliminar los anuncios que hayan publicado previamente.
RF06	Modificación de anuncios	El sistema deberá permitir a los usuarios modificar los anuncios que hayan publicado previamente.
RF07	Guardar anuncios favoritos	El sistema deberá permitir guardar los anuncios favoritos del usuario.
RF08	Mostrar lista de anuncios favoritos	El sistema deberá permitir mostrar una lista de los muebles favoritos del usuario.
RF09	Alquiler de mueble	El sistema deberá permitir a los usuarios realizar un alquiler de un mueble de otro usuario.
RF10	Contacto por correo electrónico	El sistema deberá permitir a los usuarios comunicarse mediante correo electrónico.
RF11	Mostrar lista de peticiones de alquiler	El sistema deberá permitir a los usuarios visualizar la lista de peticiones pendientes de aceptar/rechazar.
RF12	Gestionar las peticiones de alquiler de un mueble	El sistema deberá permitir a los usuarios gestionar las peticiones de alquiler de un anuncio.
RF13	Búsqueda y visualización de muebles	El sistema deberá permitir a los usuarios buscar muebles mediante filtros (categoría, ubicación, etc.) y mostrar anuncios aleatorios en la pantalla principal.

Requisitos no funcionales

Los requisitos no funcionales son aquellos que determinan la calidad y la eficiencia del sistema, teniendo en cuenta aspectos de seguridad y escalabilidad. Además, se incluyen requisitos de cumplimiento normativo y soporte técnico.

Tabla 4.2: Requisitos No Funcionales

ID	**Nombre**	**Descripción**
RNF01	Seguridad de datos	El sistema deberá proteger la contraseña de los usuarios mediante cifrado.
RNF02	Interfaz responsiva	La aplicación deberá ser eficiente, fácil de usar y con una gestión de errores en dispositivos móviles y de escritorio.
RNF03	Tiempo de respuesta	Las principales acciones del sistema deberán tener un tiempo de respuesta menor a 2 segundos.

RNF04	Mantenimiento	El sistema deberá permitir mantenimiento regular sin interrumpir el servicio.
RNF05	Cumplimiento de GDPR	La aplicación deberá cumplir con la normativa GDPR para la protección de datos personales.
RNF06	Escalabilidad	El sistema deberá poder adaptarse al crecimiento de usuarios sin perder rendimiento.
RNF07	Control de versiones	El desarrollo del sistema deberá realizarse mediante un sistema de control de versiones para mantener la trazabilidad.

Requisitos de información

Los requisitos funcionales de información son aquellos que establecen qué datos debe gestionar la aplicación, como registros de transacciones. Este tipo de requisitos garantizan un acceso adecuado y seguro a la información.

Tabla 4.3: Requisitos de Información

ID	Nombre	Descripción
RI01	Gestión de datos del usuario	El sistema almacenará información básica de los usuarios, como correo electrónico, nombre, contraseña, número de contacto, tipo de usuario (inquilino o propietario) y sus direcciones asociadas.
RI02	Gestión de anuncios	El sistema permitirá la publicación de anuncios, incluyendo nombre del mueble, descripción, categoría, condición (bueno, regular, excelente) y enlaces a fotos del mueble.
RI03	Estado de los anuncios	El sistema deberá reflejar el estado de disponibilidad de los anuncios y permitir que estén asociados a transacciones específicas.
RI04	Gestión de transacciones	El sistema almacenará datos de transacciones, como fechas de inicio y finalización, precio total, método de pago y estado de la transacción (pendiente, en proceso, completada o cancelada).
RI05	Gestión de favoritos	El sistema permitirá a los usuarios guardar anuncios como favoritos, facilitando su acceso posterior.
RI06	Gestión de direcciones	El sistema almacenará una o más direcciones asociadas a los usuarios, incluyendo calle, número, código postal, ciudad y país.

RI07	Clasificación de muebles	El sistema deberá clasificar los muebles por categorías como silla, mesa, sofá, televisión, entre otros, y reflejar su estado de conservación.

4.1.2. Casos de uso

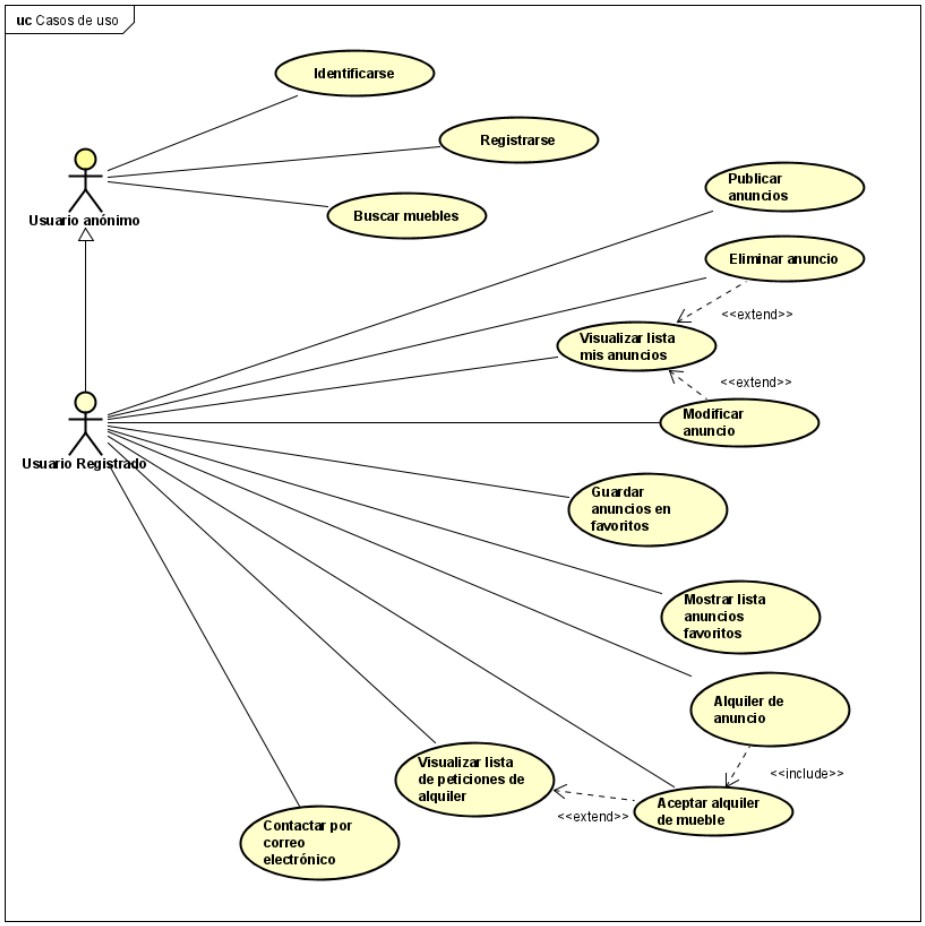


Figura 4.1: Casos de uso

Tabla 4.4: Caso de uso CU01

ID	CU01
Nombre	Identificarse
Prioridad	Alta
Riesgo	Bajo
Descripción	El usuario deberá poder identificarse en el sistema
Precondición	Ninguna

Secuencia normal	1. El usuario introduce nombre y contraseña. 2. El sistema comprueba que las credenciales del usuario son correctas. 3. El sistema muestra un mensaje de que se ha iniciado correctamente la sesión.
Secuencia alternativa	2.a. El sistema comprueba que el usuario y la contraseña no son correctos, muestra un error al usuario y el caso de uso queda sin efecto.
Postcondición	Ninguna
Frecuencia	Media

Tabla 4.5: Caso de uso CU02

ID	CU02
Nombre	Registrarse
Prioridad	Alta
Riesgo	Bajo
Descripción	El usuario deberá poder registrarse en el sistema
Precondición	Ninguna
Secuencia normal	1. El usuario introduce correo, nombre, contraseña y datos de dirección. 2. El sistema comprueba que no existe otro usuario con esa contraseña y ese nombre. 3. El sistema guarda el usuario que se ha registrado.
Secuencia alternativa	2.a. El sistema comprueba que el nombre de usuario o el correo electrónico ya existe en el sistema y el caso de uso queda sin efecto.
Postcondición	Ninguna
Frecuencia	Media

Tabla 4.6: Caso de uso CU03

ID	CU03
Nombre	Publicación de muebles

Prioridad	Alta
Riesgo	Bajo
Descripción	El usuario deberá poder publicar anuncios de muebles
Precondición	El usuario debe estar identificado
Secuencia normal	1. El usuario introduce el título, categoría, descripción, fotos y precio del mueble. 2. El sistema comprueba que la información introducida está completa. 3. El sistema crea el anuncio.
Secuencia alternativa	2.a. El sistema comprueba que el título del anuncio está vacío y muestra un mensaje de que debe rellenarlo para crear el anuncio. 2.b. El sistema comprueba que la categoría del anuncio está vacía y muestra un mensaje de que debe rellenarlo para crear el anuncio. 2.c. El sistema comprueba que la descripción del anuncio está vacía y muestra un mensaje de que debe rellenarlo para crear el anuncio. 2.d. El sistema comprueba que no se han introducido fotos en el anuncio y muestra un mensaje de que debe añadirlas para crear el anuncio. 2.e. El sistema comprueba que el precio del anuncio está vacío y muestra un mensaje de que debe rellenarlo para crear el anuncio.
Postcondición	Ninguna
Frecuencia	Media

Tabla 4.7: Caso de uso CU04

ID	CU04
Nombre	Mostrar anuncios publicados
Prioridad	Alta
Riesgo	Bajo
Descripción	El sistema deberá permitir al usuario visualizar los anuncios que previamente haya publicado
Precondición	El usuario debe estar identificado en la aplicación.

Secuencia normal	1. El usuario selecciona para ver todos los anuncios que ha publicado previamente. 2. El sistema muestra una lista de los anuncios publicados, incluyendo detalles como título, categoría y precio.
Secuencia alternativa	2a. Si no se encuentran anuncios publicados el sistema muestra un mensaje.
Postcondición	El usuario puede visualizar la lista de anuncios publicados y acceder a opciones para modificarlos o eliminarlos mediante los respectivos casos de uso.
Frecuencia	Media

Tabla 4.8: Caso de uso CU05

ID	CU05
Nombre	Eliminar un anuncio publicado
Prioridad	Media
Riesgo	Bajo
Descripción	El sistema permitirá al usuario eliminar un anuncio que haya publicado previamente.
Precondición	▪ El usuario debe estar identificado en el sistema. ▪ Debe haber anuncios publicados por el usuario.
Secuencia normal	1. El usuario selecciona el anuncio que desea eliminar. 2. El sistema solicita confirmación para eliminar el anuncio. 3. El usuario confirma que desea borrar el anuncio. 4. El sistema elimina el anuncio. 5. El sistema muestra un mensaje indicando que el anuncio ha sido eliminado correctamente.
Secuencia alternativa	2.a. El usuario cancela la confirmación, por lo que el anuncio no es eliminado y el caso de uso queda sin efecto.
Postcondición	El anuncio ya no estará disponible en la plataforma.

Frecuencia	Media

Tabla 4.9: Caso de uso CU06

ID	CU06
Nombre	Modificar un anuncio publicado
Prioridad	Alta
Riesgo	Bajo
Descripción	El sistema permitirá al usuario modificar un anuncio que haya publicado previamente desde la lista de "Mis Anuncios".
Precondición	▪ El usuario debe estar identificado en el sistema. ▪ Debe haber anuncios publicados por el usuario.
Secuencia normal	1. El usuario selecciona el anuncio que desea modificar. 2. El sistema muestra los detalles actuales del anuncio. 3. El usuario actualiza los datos del anuncio. 4. El sistema actualiza la información del anuncio. 5. El sistema muestra un mensaje indicando que los cambios se han guardado correctamente.
Secuencia alternativa	3.a. Si el usuario deja campos obligatorios sin rellenar, el sistema muestra un mensaje de error indicando que debe completarlos antes de guardar.
Postcondición	El anuncio se actualiza con la nueva información proporcionada por el usuario.
Frecuencia	Alta

Tabla 4.10: Caso de uso CU07

ID	CU07
Nombre	Guardar Muebles Favoritos
Prioridad	Media
Riesgo	Bajo
Descripción	El sistema deberá permitir guardar los muebles favoritos del usuario.
Precondición	El usuario debe estar identificado en el sistema.

Secuencia normal	1. El usuario selecciona un anuncio para añadirlo a favoritos. 2. El sistema comprueba si el anuncio esta ya en favoritos o no. 3. El sistema muestra un mensaje confirmando que el mueble se ha guardado en favoritos.
Secuencia alternativa	2.a. Si el mueble ya está en la lista de favoritos, elimina el anuncio de la lista. 2.b. Si el mueble no está en la lista de favoritos, añade el anuncio de la lista.
Postcondición	El mueble seleccionado se guarda en la lista de favoritos del usuario.
Frecuencia	Media

Tabla 4.11: Caso de uso CU08

ID	CU08
Nombre	Mostrar lista de anuncios favoritos
Prioridad	Media
Riesgo	Bajo
Descripción	El sistema deberá permitir listar los muebles favoritos del usuario.
Precondición	El usuario debe estar identificado en el sistema.
Secuencia normal	1. El usuario solicita ver su lista de favoritos 2. El sistema recupera la lista de anuncios marcados como favoritos. 3. El sistema muestra la lista de anuncios favoritos en una interfaz visual ordenada. 4. El usuario selecciona uno de los anuncios. 5. El sistema muestra las especificaciones del anuncio.
Secuencia alternativa	3.a. Si no hay muebles guardados en favoritos el sistema muestra un mensaje al respecto
Postcondición	Ninguna
Frecuencia	Media

Tabla 4.12: Caso de uso CU09

ID	CU09
Nombre	Realizar un alquiler de un anuncio
Prioridad	Alta
Riesgo	Medio
Descripción	El sistema deberá permitir a los usuarios realizar un alquiler de un anuncio.
Precondición	Los usuarios deben estar identificados en el sistema.
Secuencia normal	1. El usuario arrendatario selecciona un anuncio. 2. El sistema muestra el anuncio seleccionado. 3. El usuario solicita alquilar un mueble. 4. El sistema crea una transacción con estado “Pending” y envía una notificación al propietario del anuncio. 5. El propietario del anuncio acepta la solicitud. 6. El sistema cambia el estado de la transacción a “InProcess”. 7. El usuario arrendatario completa la transacción. 8. El sistema actualiza el estado de la transacción a “Completed”.
Secuencia alternativa	5.a. Si el propietario no acepta la solicitud de alquiler, la transacción pasa a estado “Canceled”.
Postcondición	Ninguna.
Frecuencia	Media

Tabla 4.13: Caso de uso CU10

ID	CU10
Nombre	Contacto por correo electrónico
Prioridad	Media
Riesgo	Bajo
Descripción	El sistema deberá permitir a los usuarios comunicarse mediante correo electrónico
Precondición	El usuario debe estar registrado en la aplicación.

Secuencia normal	1. El usuario solicita la comunicación con otro usuario. 2. El sistema muestra un formulario a rellenar los datos. 3. El usuario rellena el formulario de contacto con los datos requeridos. 4. El usuario confirma el envío del mensaje. 5. El sistema genera un correo electrónico con los datos introducidos y lo envía automáticamente al destinatario correspondiente.
Secuencia alternativa	4.a. Si el usuario no completa los campos requeridos, el sistema muestra un mensaje indicando que debe rellenar todos los campos obligatorios.
Postcondición	El correo electrónico es enviado al destinatario especificado y el usuario es notificado del resultado del envío.
Frecuencia	Media.

Tabla 4.14: Caso de uso CU11

ID	CU11
Nombre	Mostrar lista de peticiones de alquiler
Prioridad	Alta
Riesgo	Medio
Descripción	El sistema deberá permitir visualizar las peticiones de alquiler pendientes.
Precondición	El usuario propietario debe estar registrado y autenticado en la aplicación.
Secuencia normal	1. El usuario solicita visualizar la lista de peticiones. 2. El sistema muestra la lista.
Secuencia alternativa	2.a. Si no existen solicitudes pendientes, el sistema muestra un mensaje indicando que no hay solicitudes disponibles.
Postcondición	Ninguna
Frecuencia	Media.

Tabla 4.15: Caso de uso CU12

ID	CU12
Nombre	Aceptar el alquiler de un mueble
Prioridad	Alta
Riesgo	Medio
Descripción	El sistema deberá permitir gestionar las peticiones de alquiler de un anuncio publicado.
Precondición	▪ El usuario propietario debe estar registrado y autenticado en la aplicación. ▪ Debe existir al menos una solicitud de alquiler pendiente asociada a un mueble publicado por el propietario.
Secuencia normal	1. El usuario selecciona una solicitud específica de la lista de peticiones. 2. El usuario acepta la solicitud de alquiler.
Secuencia alternativa	2.a. El usuario rechaza la solicitud.
Postcondición	▪ El sistema actualiza el estado de la solicitud como aceptada o cancelada. ▪ Se notifica al solicitante sobre el resultado de su solicitud.
Frecuencia	Media.

Tabla 4.16: Caso de uso CU13

ID	CU13
Nombre	Visualización y búsqueda de muebles
Prioridad	Alta
Riesgo	Bajo
Descripción	Este caso de uso permite al usuario visualizar automáticamente los anuncios de muebles disponibles tras identificarse en la aplicación. Además, ofrece la funcionalidad de búsqueda mediante filtros como categoría, precio o ubicación, así como la visualización de detalles de un mueble específico.
Precondición	Deben existir anuncios de muebles en la base de datos.

Secuencia normal	1. El usuario puede aplicar filtros (categorías, precio, ubicación, etc.) para ajustar los resultados. 2. El sistema actualiza y muestra los anuncios según los filtros aplicados. 3. El usuario selecciona un anuncio para ver sus detalles. 4. El sistema muestra información relevante del mueble seleccionado.
Secuencia alternativa	2.a. Si no hay anuncios disponibles, el sistema muestra un mensaje indicando que no se encontraron resultados. 2.b. Si no se aplican filtros, el sistema muestra todos los anuncios.
Postcondición	El usuario puede proceder a contactar con el anunciante o guardar un anuncio como favorito.
Frecuencia	Alta (se espera un uso frecuente dado que es una funcionalidad principal de la aplicación).

4.1.3. Modelo de Dominio

Entidades del Modelo de Dominio

El modelo de dominio, representado en la Figura 4.2, describe las principales entidades y sus relaciones dentro de la plataforma, estableciendo una estructura clara para la gestión de los datos y sus interacciones.

La clase **User** representa a los usuarios registrados en el sistema, ya sean propietarios o inquilinos. Estos usuarios pueden publicar anuncios de muebles y gestionar transacciones de alquiler. También pueden interactuar con los anuncios marcándolos como favoritos. Cada usuario cuenta con información básica como su correo electrónico, nombre, contraseña, número de contacto y rol dentro de la plataforma. Además, para agregaciones futuras se deja planteada la asociación con una **Direction**, que almacena datos sobre la dirección del usuario, incluyendo calle, número, código postal, ciudad y país.

La entidad **Advertisement** modela los anuncios de muebles disponibles para alquiler. Cada anuncio tiene atributos como título, descripción, precio, estado (definido por el enum **FurnitureCondition**), categoría (gestionada por el enum **FurnitureCategory**) y la fecha de creación y actualización del anuncio. Cada anuncio está vinculado a un usuario a través de su correo electrónico.

La clase **Transaction** es fundamental en la plataforma, ya que registra los procesos de alquiler de muebles. Contiene información clave como el identificador del anuncio alquilado, las fechas de inicio y finalización del alquiler, el precio total, el método de pago utilizado y los correos electrónicos del inquilino y del propietario. Además, incorpora el estado de confirmación del propietario mediante el enum **OwnerConfirmation**, que puede tomar valores como `Pending`, `Accepted` o `Rejected`.

Los valores de los atributos categóricos están definidos en las siguientes enumeraciones:

- **FurnitureCategory**: clasifica los muebles disponibles en categorías como `chair`, `table`, `TV`, `chestOfDrawers`, `sofa`, `bookshelf` y `other`.
- **FurnitureCondition**: indica el estado del mueble con valores como `Good`, `Fair` y `Excellent`.

Este modelo de dominio proporciona una estructura robusta para la gestión eficiente de los anuncios, usuarios y transacciones dentro de la plataforma, garantizando coherencia en la información y facilitando las interacciones entre los participantes del sistema.

Diagrama del Modelo de Dominio

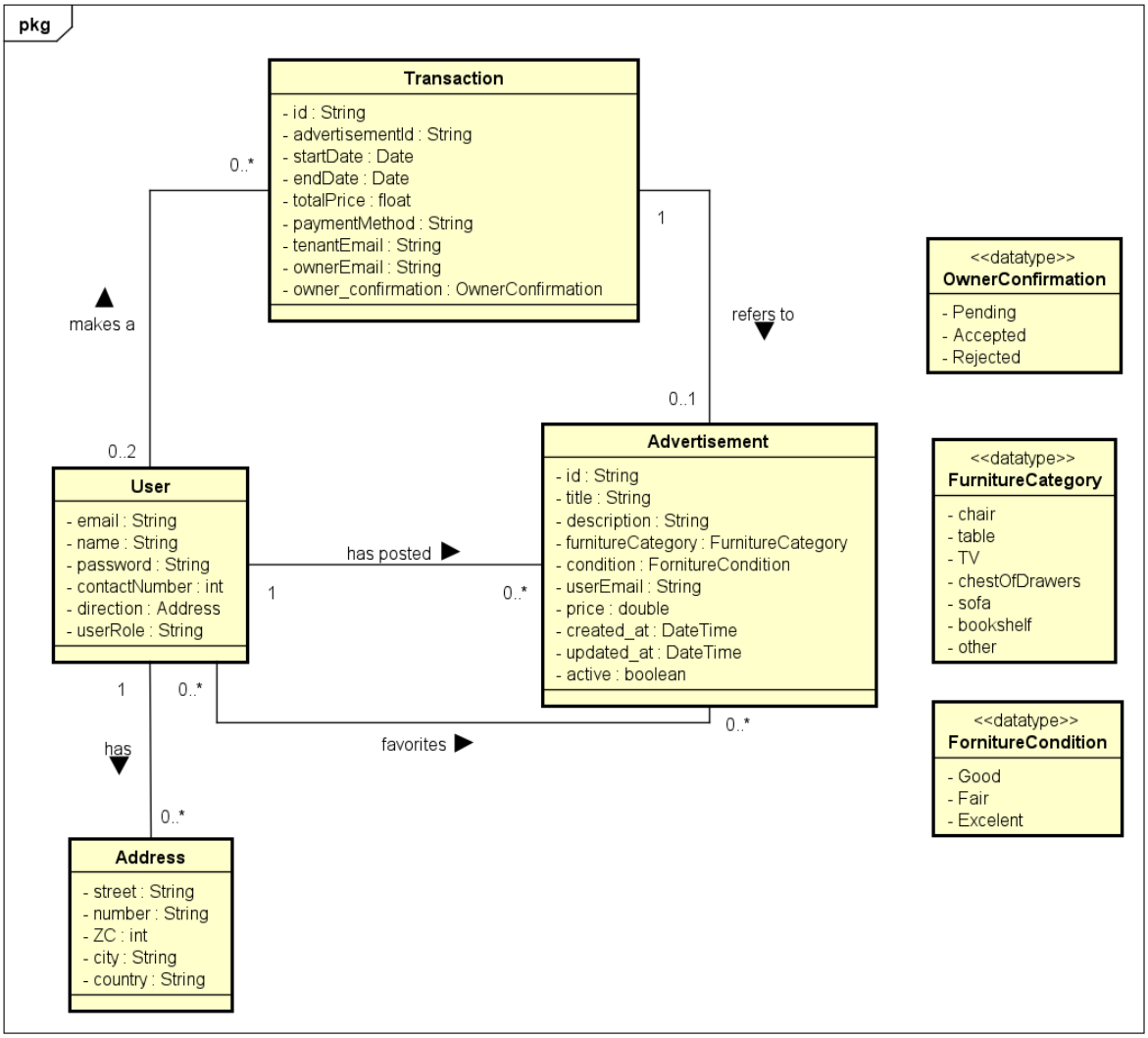


Figura 4.2: Modelo de dominio del sistema.

4.1.4. Diagrama de actividad

CU01. Identificarse

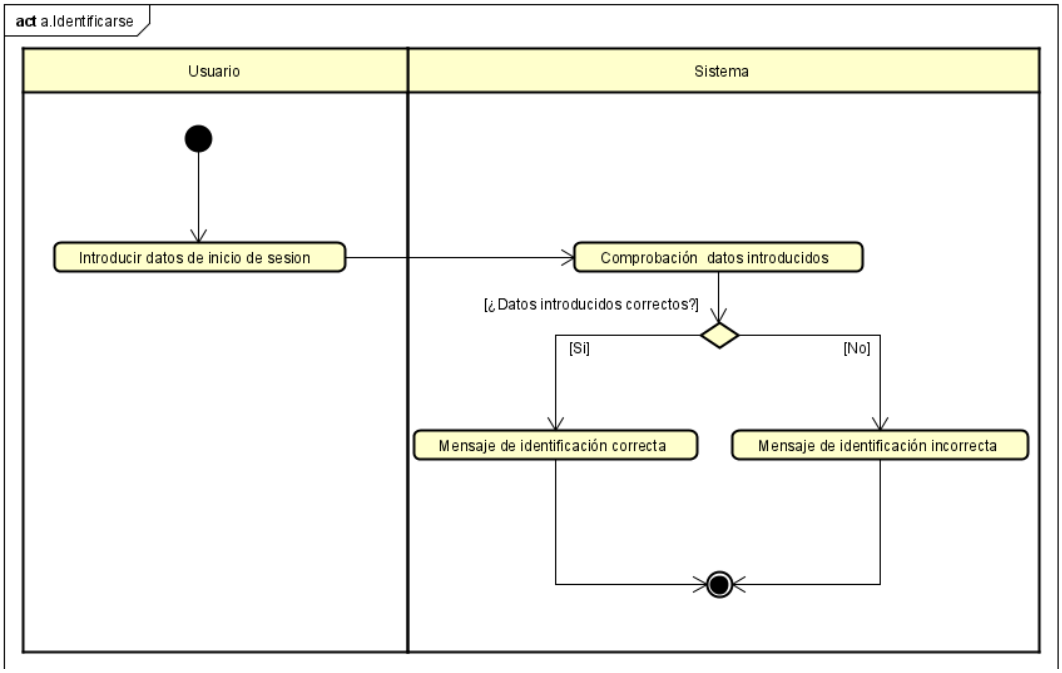


Figura 4.3: Secuencia CU identificarse.

CU02. Registrarse

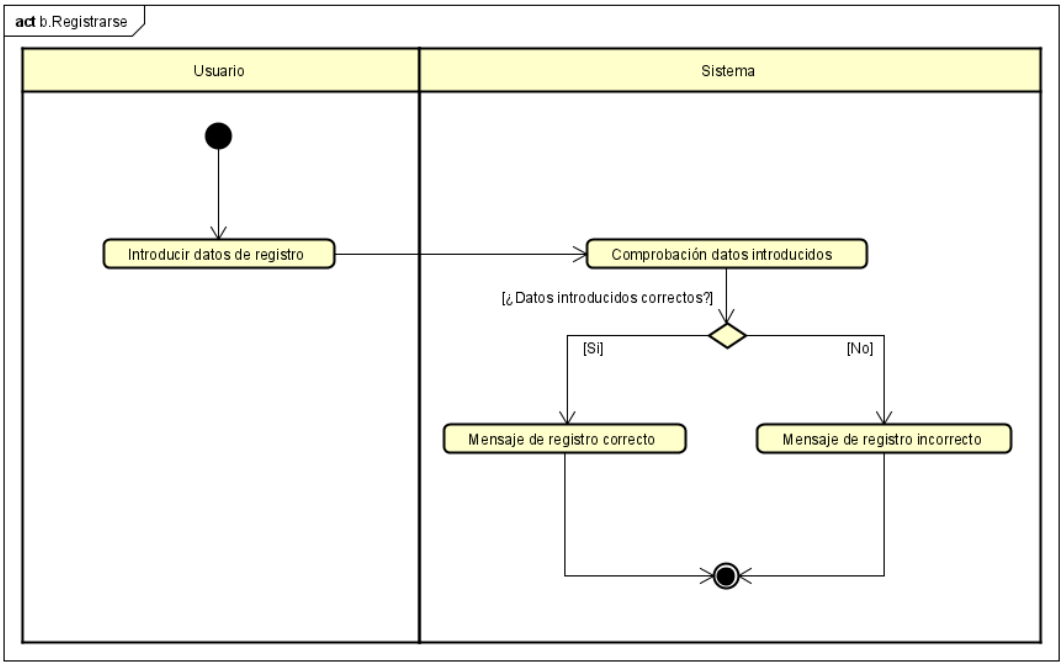


Figura 4.4: Secuencia CU registrarse.

CU03. Publicación muebles

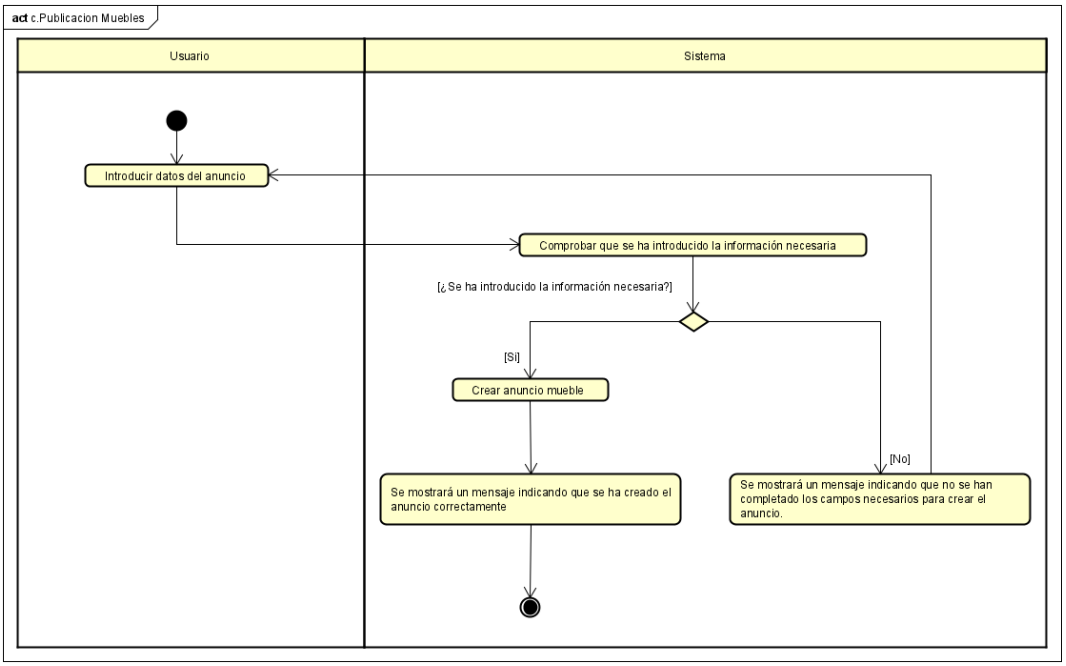


Figura 4.5: Secuencia CU publicar anuncio.

CU04. Visualizar anuncios.

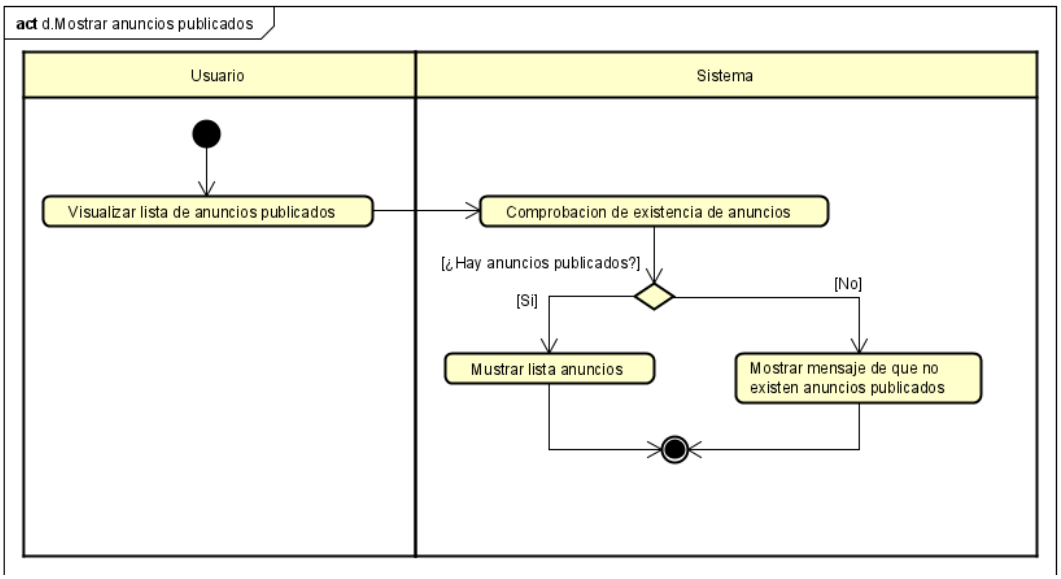


Figura 4.6: Secuencia CU visualizar anuncios.

CU05. Eliminar anuncio publicado

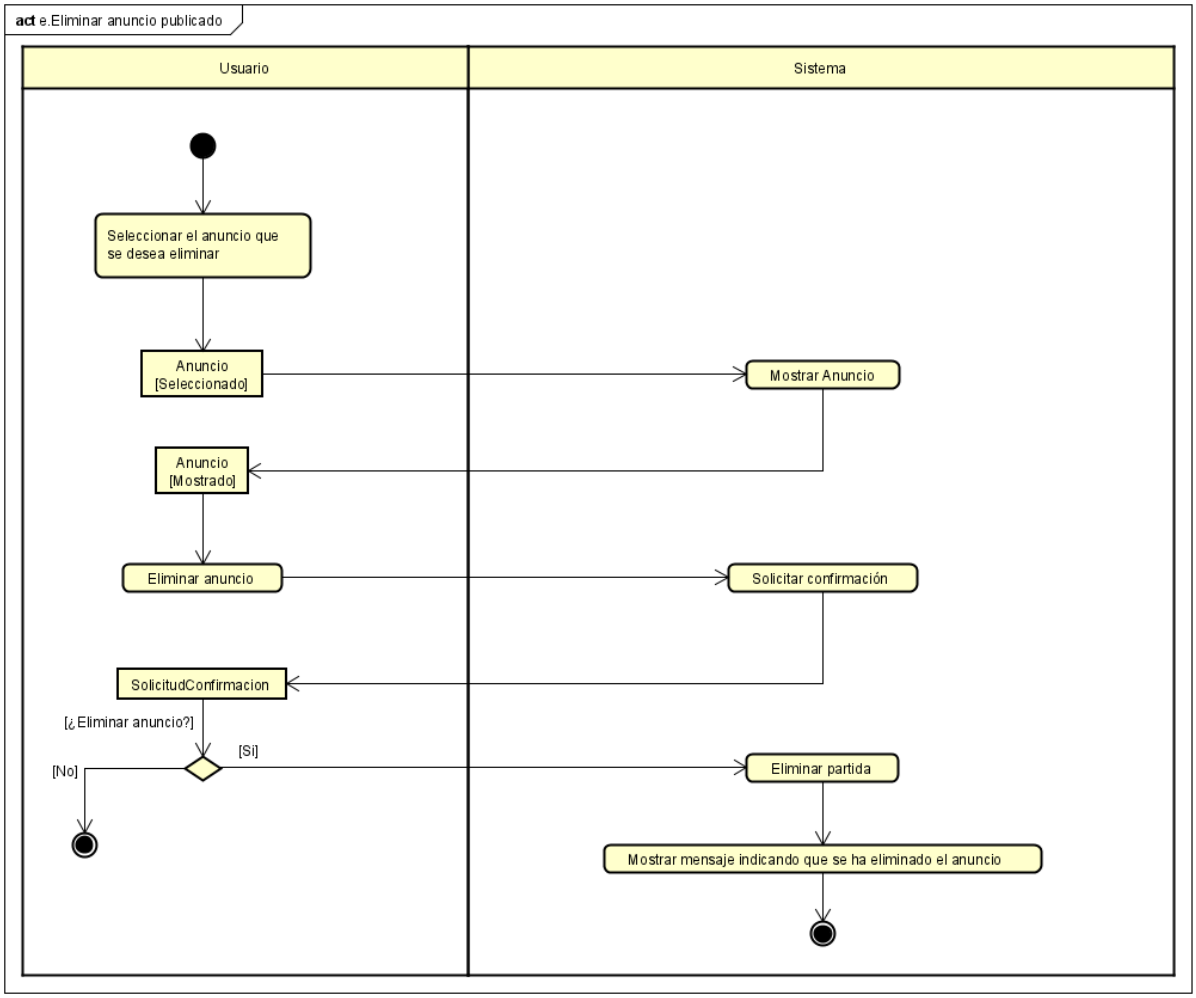


Figura 4.7: Secuencia CU Eliminar anuncio publicado.

CU06. Modificar anuncio publicado

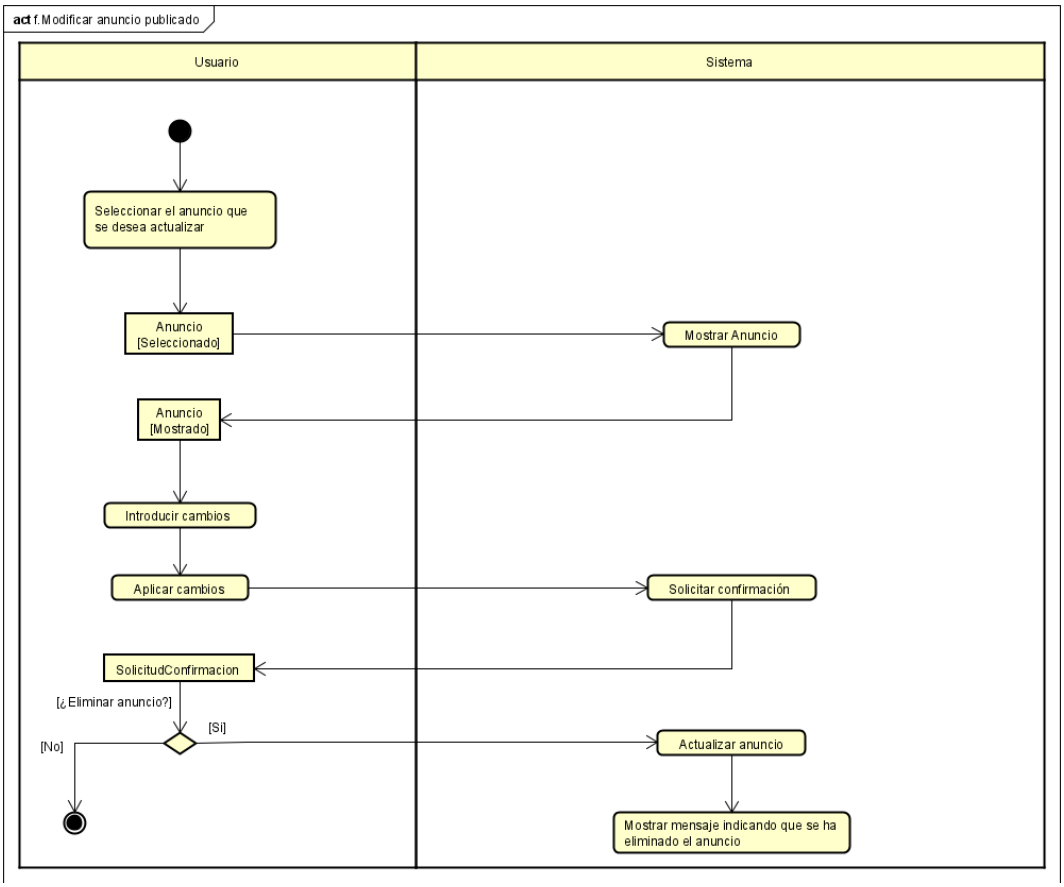


Figura 4.8: Secuencia CU Modificar anuncio publicado.

CU07. Guardar mueble en favoritos

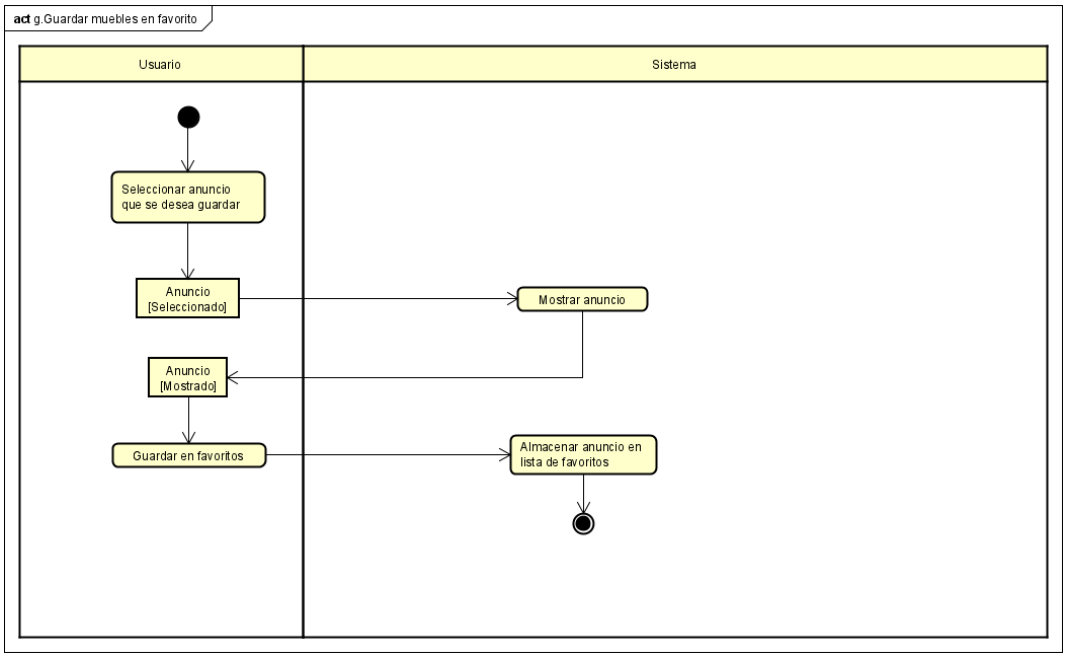


Figura 4.9: Secuencia CU Guardar muebles en favoritos.

CU08. Mostrar lista de favoritos

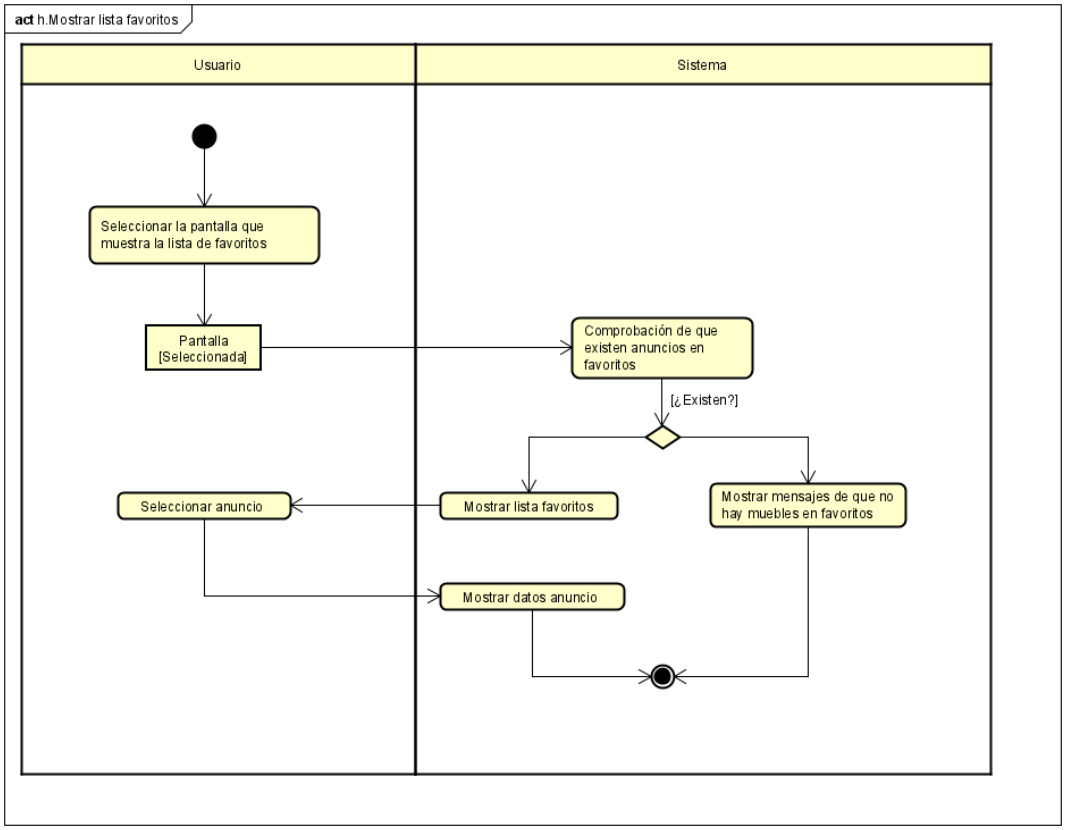


Figura 4.10: Secuencia CU Mostrar lista de favoritos.

CU09.Realizar alquiler de anuncio.

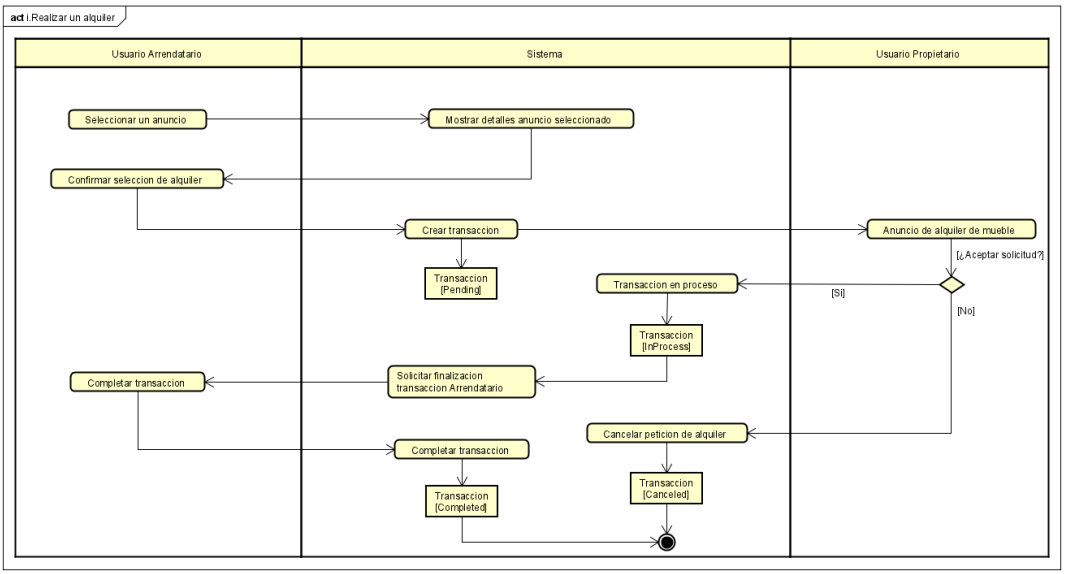


Figura 4.11: Secuencia CU Realizar alquiler de anuncio.

CU10.Contactar por correo electrónico

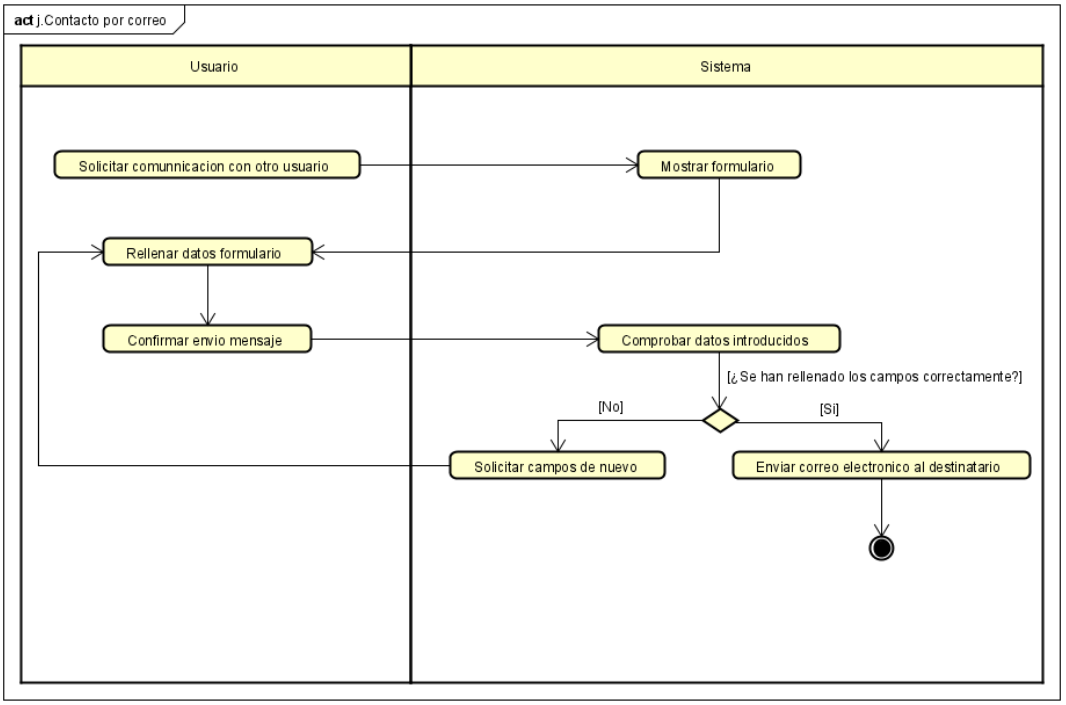


Figura 4.12: Secuencia CU Contactar por correo electrónico.

CU11.Mostrar lista de peticiones de alquiler

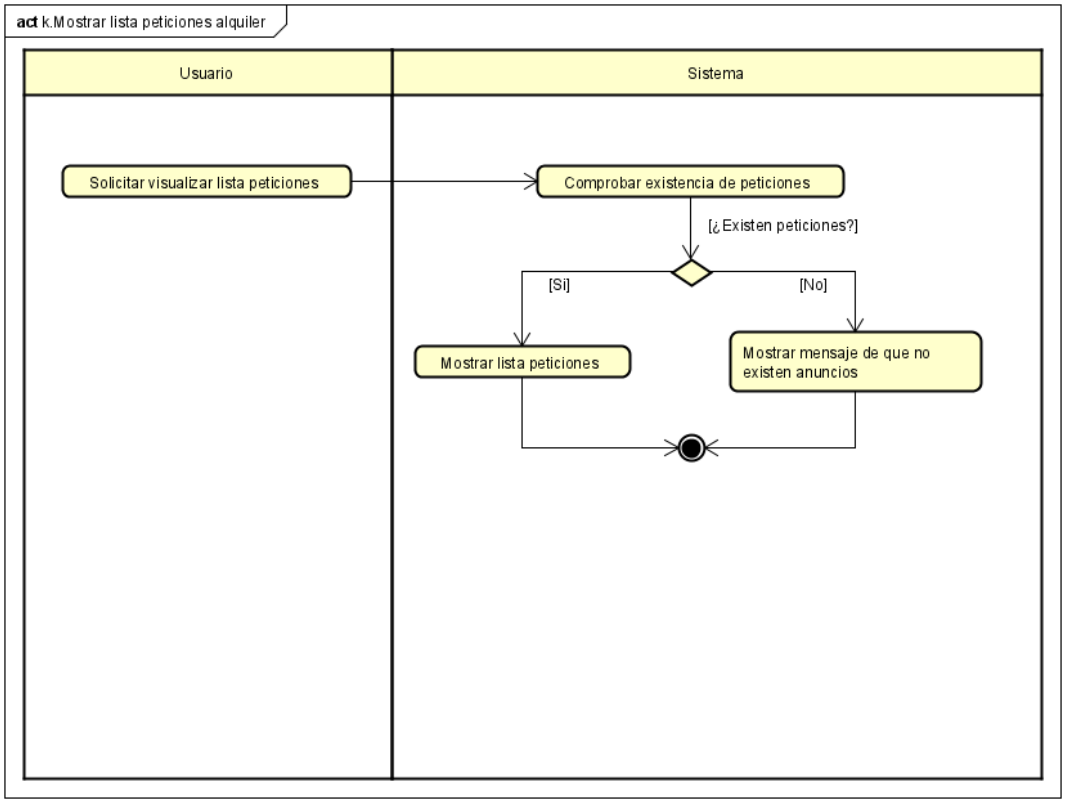


Figura 4.13: Secuencia CU Mostrar lista de peticiones de alquiler.

CU12.Aceptar alquiler de mueble

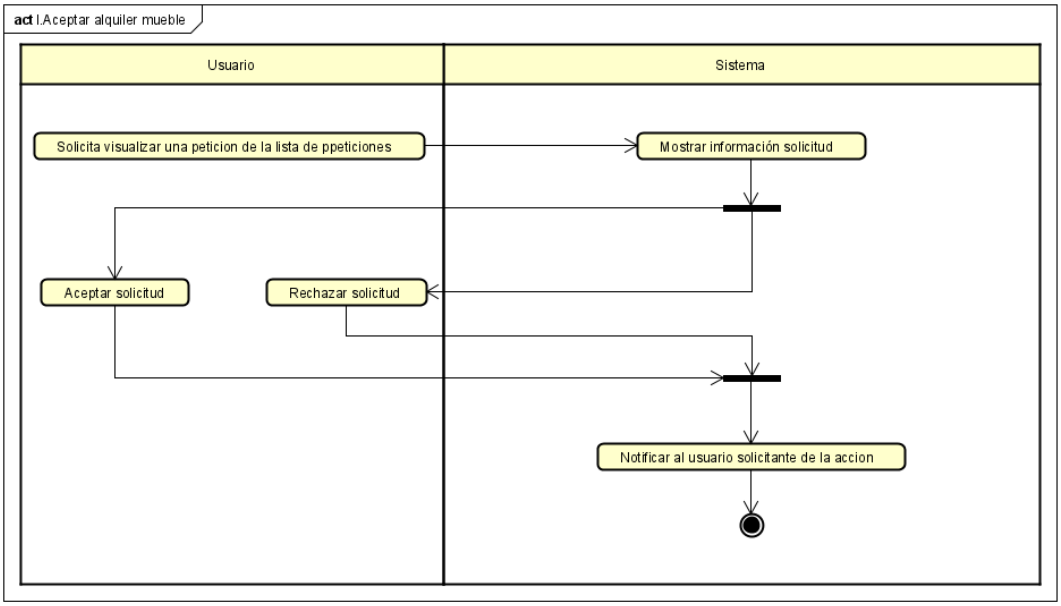


Figura 4.14: Secuencia CU Aceptar alquiler de mueble.

CU13.Búsqueda de mueble

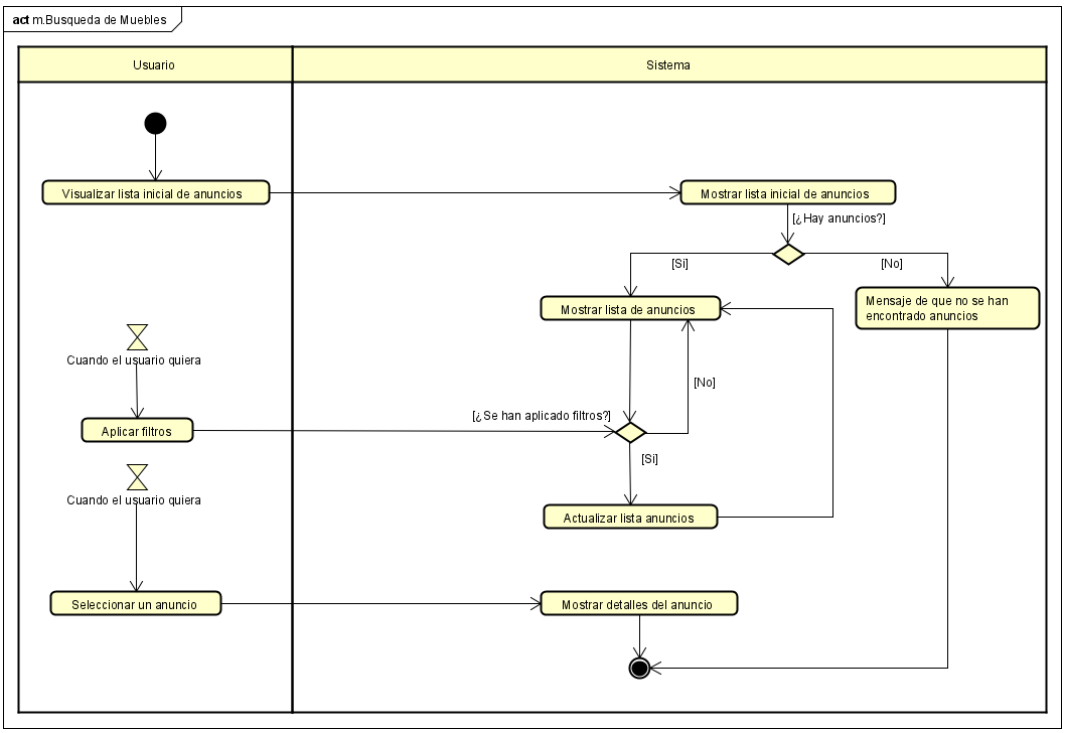


Figura 4.15: Secuencia CU Búsqueda de mueble.

4.2. Estructura del proyecto back-end

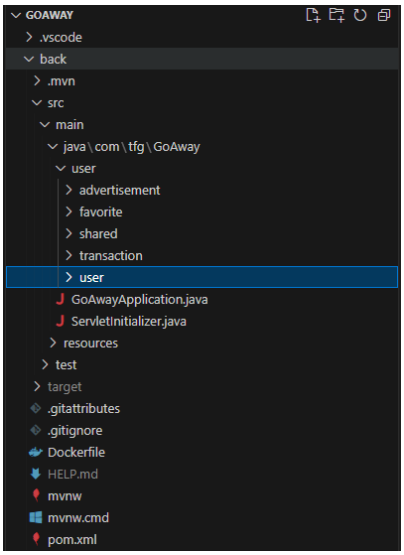


Figura 4.16: Estructura general del proyecto java desde Visual Studio Code.

En la Figura 4.16 se muestra una visión global de los directorios y ficheros más relevantes de la aplicación Java para la capa de servidor. Esta organización pretende facilitar la mantenibilidad del proyecto

y reforzar la separación de responsabilidades entre los distintos componentes.

La base del proyecto se compone de los siguientes elementos a nivel de directorios:

- **src/main/java/**: En esta carpeta se ubica el código fuente principal distribuido en un único módulo *user* con su funcionalidad separada en submódulos y con una carpeta denominada *shared* que contiene el código que se tenga que compartir entre la funcionalidad del módulo. Posteriormente se detallará de qué manera interactúan entre ellos y por qué se ha elegido esta organización, especialmente con la arquitectura utilizada. Lo que facilita la agregación de nuevas funcionalidades o módulos sin necesitar grandes modificaciones.

- **src/main/resources/**: Contiene recursos de configuración y archivos estáticos. Como en este caso *application.properties*, donde se definen parámetros de conexión a la base de datos, puertos o variables específicas de la aplicación, como el debugger.

- **src/test/java/**: No se han incluido pruebas automatizadas debido a la sencillez de la funcionalidad y la lógica se puede verificar manualmente.

- **.mvn/**: Directorio que guarda la configuración interna del *Maven Wrapper*. Por esto no es necesario que Maven esté instalado en el sistema para construir el proyecto.

- **target/**: Carpeta donde Maven guarda los resultados de la compilación (como el *.jar* generado) y directorios auxiliares (*classes*, *generated-sources*, *maven-archiver*, etc.). Este, se excluye del repositorio mediante *.gitignore* para evitar añadir ficheros binarios y temporales.

Respecto a los ficheros principales, **pom.xml** contiene los descriptores de Maven que definen las dependencias, la versión de Java, el nombre del proyecto y los *plugins* de construcción necesarios. **.gitignore** y **.gitattributes** corresponden a ficheros para el control de versiones. El primero indica qué elementos deben ignorarse y el segundo establece atributos específicos. Y por último **Dockerfile** contiene la creación de la imagen de contenedor de la API.

4.2.1. Arquitectura del proyecto back-end

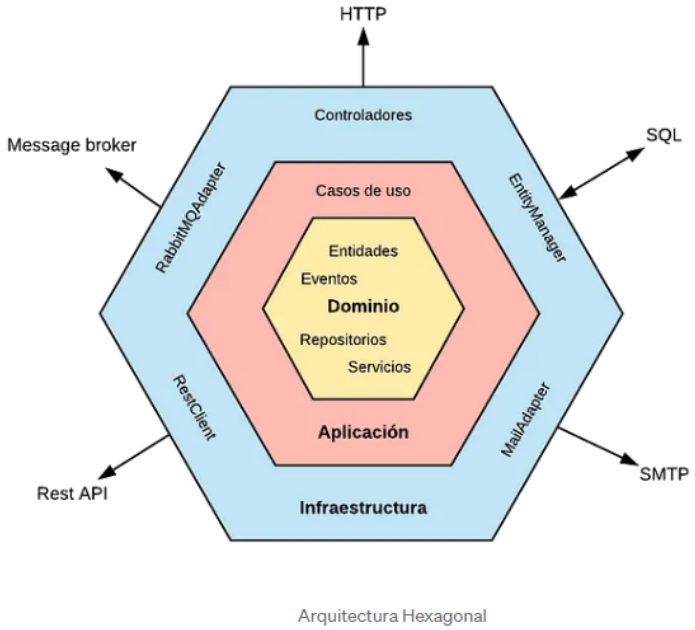


Figura 4.17: Capas de la Arquitectura Hexagonal. [30]

La arquitectura hexagonal es también conocida como *puertos y adaptadores* y la idea principal es aislar la lógica de negocio de cualquier detalle externo (bases de datos, frameworks, protocolos de comunicación, etc.). De esta manera, el núcleo del sistema se mantiene independiente, lo que hace que sea reutilizable y se pueda modificar sin tener un impacto en la infraestructura.

Esta forma de crear una aplicación tiene enormes ventajas en lo que se refiere a la escalabilidad, testeo y mantenimiento, debido a que las partes externas pueden ser sustituidas (por ejemplo, pasar de una base de datos SQL a otra NoSQL) sin modificar el código del dominio ni los flujos de negocio. Además, la arquitectura hexagonal hace hincapié en ubicar cada componente en el anillo que le corresponda, lo que evita mezclar responsabilidades.

- **Dominio (Núcleo de negocio):** Representa la parte central de la aplicación, donde se ubican las entidades y toda la lógica de negocio. Tiene una implementación totalmente aislada de detalles de infraestructura y librerías externas, de modo que cualquier validación o regla esencial se conserva aquí sin dependencia de elementos ajenos.
- **Aplicación (Casos de uso):** Define los flujos que coordinan las acciones sobre el dominio. Cada caso de uso encapsula un proceso concreto, como la creación de un nuevo usuario o la edición de su perfil. Desde aquí se invoca el dominio y siempre que sea necesario, se interactúa con otros servicios o adaptadores.
- **Adaptadores de entrada (Puertos de entrada):** Gestionan la comunicación que llega desde el exterior de la aplicación. Suelen ser controllers HTTP en el caso de peticiones REST, pero pueden existir otras vías de entrada, como colas de mensajería o interfaces de línea de comandos.

- **Adaptadores de salida (Puertos de salida):** Contemplan cualquier integración que la aplicación necesite hacia el exterior, como persistir información en la base de datos o conectarse a servicios de terceros. Transforman los datos según requiera la capa de dominio o la propia infraestructura.

Esta arquitectura favorece la evolución independiente de cada capa y la facilidad para reemplazar o modificar servicios externos sin alterar el núcleo de negocio. Aislando la lógica principal de los detalles de infraestructura, facilita la mantenibilidad, la escalabilidad y la realización de pruebas unitarias más precisas.

Con la creación de estos 4 anillos, esta arquitectura es más flexible que, por ejemplo, los diseños monolíticos o una arquitectura en capas, ya que cada componente se encuentra en el lugar adecuado sin generar dependencias. entre la lógica de negocio y la parte técnica.

4.3. Estructura del proyecto front-end

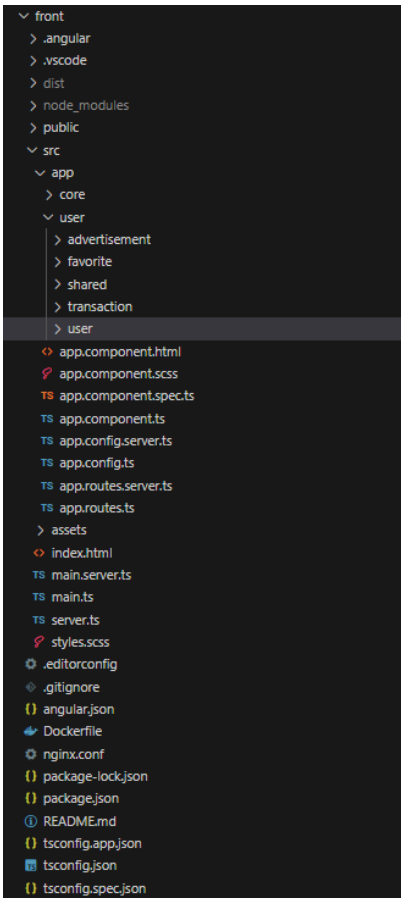


Figura 4.18: Estructura general del proyecto Angular desde Visual Studio Code.

En la Figura 4.18 se muestra una visión general de los directorios y ficheros más relevantes de la aplicación web, desarrollada con Angular 19. Esta organización favorece la mantenibilidad y facilita la incorporación de las nuevas funcionalidades o cambios que se realicen en el back-end.

La base del proyecto se compone de los siguientes elementos a nivel de directorios:

- **src**/**app**/: Contiene el código fuente principal dividido en distintas funcionalidades del módulo user (*advertisement*, *favorite*, *shared*, *transaction*, *user*, etc.). Además, incluye:
  - **app.component.ts** y **app.component.html**: Ficheros raíz del componente principal de la aplicación.
  - **app.routes.ts**: Donde se definen las rutas del proyecto, indicando para cada URL el componente o módulo correspondiente.
- **dist**/ (o **public**/): Se genera al compilar la aplicación para producción. Se excluye del repositorio ya que su contenido se regenera automáticamente.
- **node_modules**/: Esta es la carpeta con las dependencias instaladas para Angular y bibliotecas externas. Se ignora en `.gitignore` para evitar sobrecargar el repositorio.

En cuanto a los ficheros principales, **package.json** y **package-lock.json** especifican las dependencias y scripts de ejecución. **angular.json** define la configuración de Angular. Finalmente, **Dockerfile** y **nginx.conf** se emplean para generar la imagen de contenedor y ajustar el servidor, permitiendo desplegar la aplicación en entornos de producción.

4.4. Diseño de datos

4.4.1. Diseño de datos

La estructura de la base de datos está representada mediante un diagrama entidad-relación generado con MySQL Workbench, como se muestra en la Figura 4.19. Se utilizan distintos tipos de línea para este modelo para indicar la naturaleza de las relaciones entre entidades:

- **Línea sólida**: Indica una relación de uno a muchos (1:n). En este tipo de relación, una entidad en el extremo “uno” puede estar asociada con múltiples registros en la entidad del extremo “muchos”. Este es el caso, por ejemplo, de la relación entre user y advertisement, donde un usuario puede tener varios anuncios, pero cada anuncio pertenece a un único usuario.
- **Línea punteada**: Representa una relación opcional, es decir, la existencia de un registro en una entidad no obliga a que haya un registro relacionado en la otra. Un ejemplo en este modelo es la dirección de un usuario, ya que un usuario puede no haber registrado una dirección, pero si la tiene, estará vinculada a él.
- **Línea doble**: Este tipo de línea se usa para denotar relaciones de muchos a muchos (n:m). Múltiples registros en una entidad pueden estar asociados con múltiples registros en otra. Sin embargo, en el diseño de esta base de datos, con el diagrama de entidad relación, este tipo de

relación no está presente, ya que todas las relaciones de este tipo han sido descompuestas en entidades intermedias.

Para facilitar la identificación de claves primarias y foráneas dentro del modelo, se han aplicado las siguientes convenciones en el diagrama:

- **Claves primarias**: Son resaltadas en el diagrama mediante un icono en forma de llave de color amarillo. En el caso de claves primarias compuestas, cada atributo que forma parte de la clave está marcado con una llave de color naranja.
- **Claves foráneas**: Se representan de dos maneras en el diagrama. Por un lado, mediante un rombo de color naranja junto al campo correspondiente, indicando que dicho atributo es una clave foránea. Por otro lado, la relación entre tablas se ilustra con una línea, cuyo extremo en forma de triángulo identifica la entidad que contiene la clave foránea correspondiente.

Este diseño garantiza una estructura de datos clara y bien organizada, facilitando la integridad referencial y permitiendo una gestión eficiente de la información almacenada en la base de datos.

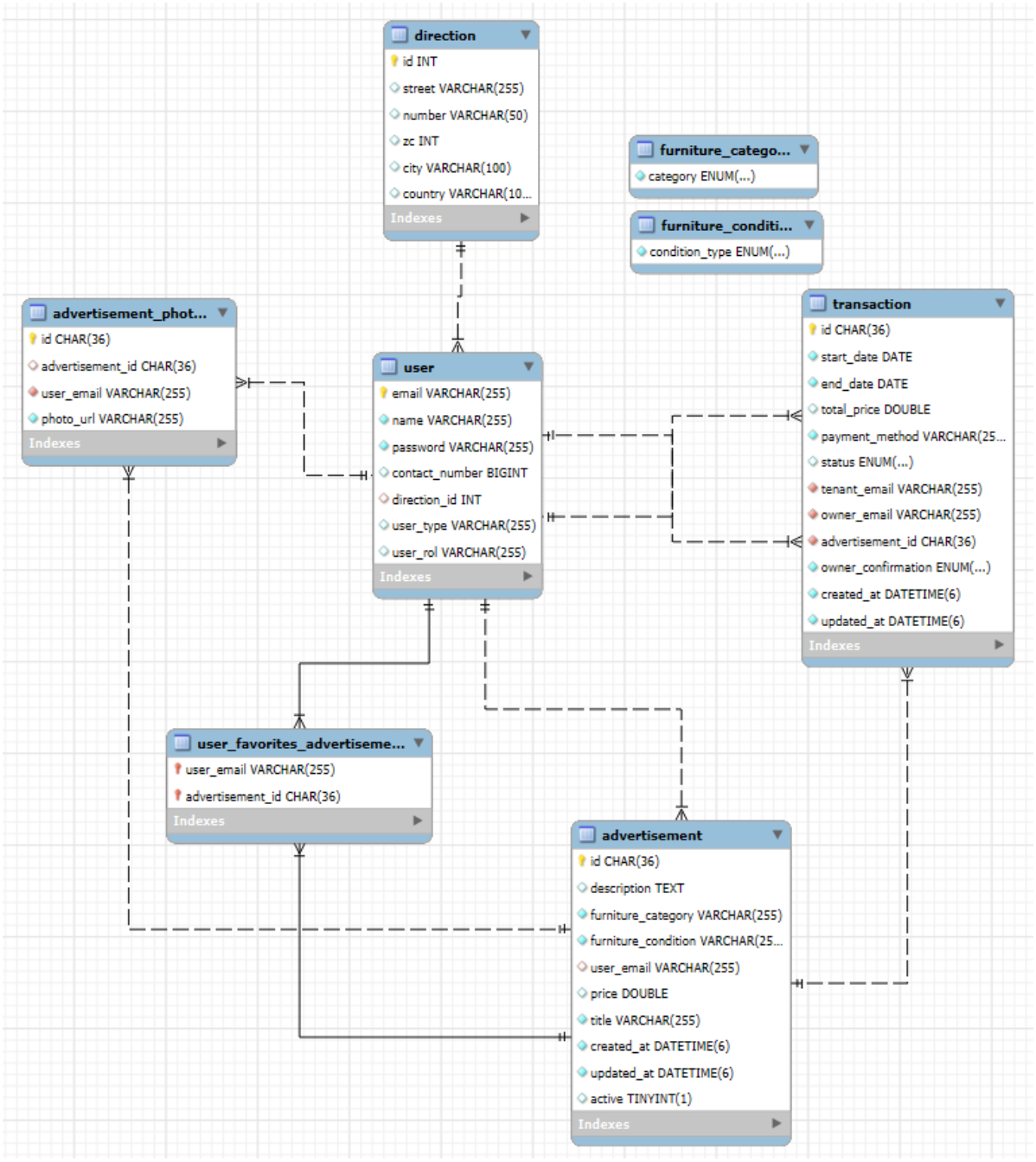


Figura 4.19: Diagrama Entidad-Relación

4.4.2. Patrones de diseño

Los patrones de diseño [31] son soluciones habituales a problemas que ocurren con frecuencia en el diseño de software. Son como planos prefabricados, pudiéndose personalizar para resolver un problema de diseño recurrente en el código.

Los patrones de diseño varían en su nivel de detalle, complejidad y escala de aplicabilidad al sistema completo que se diseña. Existen patrones básicos y de bajo nivel que normalmente se llaman *idioms*, que se suelen aplicar a un único lenguaje de programación. Y otros más universales y de más alto nivel llamados Patrones de Arquitectura. Estos últimos serán los que se explicarán a continuación.

Además, los patrones pueden clasificarse por su propósito, habiendo así tres clasificaciones:

- **Patrones creacionales:** se centran en la creación de objetos incrementando la flexibilidad y la reutilización de código existente. Dentro de esta categoría existen patrones como *Singleton* o *Factory*.
- **Patrones estructurales:** tratan la manera en que los objetos se conectan con otros objetos, para asegurar que los cambios del sistema no requieran cambiar esas conexiones. Patrones como *Adaptador*, *Compuesto* o *Fachada* se incluyen en esta categoría.
- **Patrones de comportamiento:** se centran en cómo los objetos interactúan y comunican entre sí, ayudando a definir las relaciones y responsabilidades entre ellos. Dentro de esta categoría se encuentran patrones como *Observador*, *Estrategia* o *Comando*.

A continuación se describen los patrones utilizados en este proyecto.

Patrón Adapter

El patrón *Adapter* es un patrón de diseño estructural que permite conectar interfaces incompatibles, actuando como un traductor entre dos componentes que no podrían interactuar directamente debido a diferencias en sus interfaces. Este patrón se utiliza para adaptar una interfaz existente a la interfaz esperada por el cliente, facilitando la interoperabilidad sin modificar el código original de los componentes involucrados.

- **Interfaz Objetivo:** Corresponde a la interfaz que el cliente espera utilizar para interactuar con un componente. Define el contrato que el adaptador debe cumplir.
- **Adaptador:** Es la clase o componente que implementa la interfaz objetivo, tomando una interfaz incompatible (la del componente adaptado) y traduciendo sus métodos o datos al formato esperado por el cliente.
- **Componente Adaptado:** Es el componente existente con una interfaz incompatible que el adaptador transforma para que sea usable por el cliente.

De esta manera, el **adaptador** permite que el cliente utilice el componente adaptado sin necesidad de conocer los detalles de su interfaz original. Al encapsular la lógica de conversión, el patrón evita que cambios en el componente adaptado o en la interfaz objetivo afecten al cliente, promoviendo un diseño flexible y mantenible.

Este patrón fomenta la reutilización de código y la interoperabilidad entre sistemas, facilitando la integración de tecnologías o bibliotecas existentes con el sistema actual. Además, mejora la testabilidad al permitir reemplazar componentes adaptados por simulaciones o implementaciones alternativas sin alterar la lógica del cliente.

Se ha utilizado de una manera fundamental para la implementación de la arquitectura hexagonal. De esta manera, se crea un puente entre el núcleo de la aplicación y la lógica de negocio de las interfaces, bases de datos y sistemas de archivos. Su uso ha sido aplicado tanto para adaptar las entradas (solicitudes HTTP) como las salidas (operaciones del dominio a tecnologías externas) lo que garantiza un modelo modular.

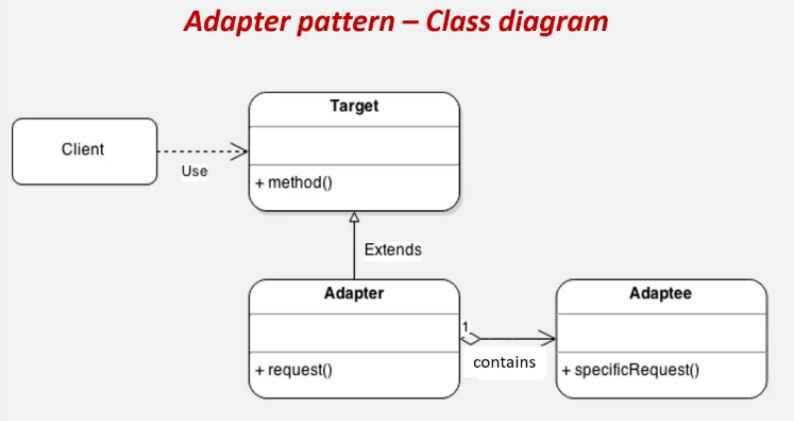


Figura 4.20: Esquema patrón adaptador [32].

Builder

El patrón de diseño *Builder* es un patrón creacional, el cual proporciona una solución flexible para la creación de objetos complejos. Sirve para separar la construcción de un objeto complejo de su representación, provocando que el mismo proceso de creación produzca diferentes representaciones. Con este patrón se evita la necesidad de tener un constructor con muchos parámetros, ofreciendo un enfoque paso a paso para la construcción del objeto.

Se ha utilizado para la simplificación en la creación de objetos complejos, para mejorar la legibilidad del código y para permitir la construcción paso a paso.

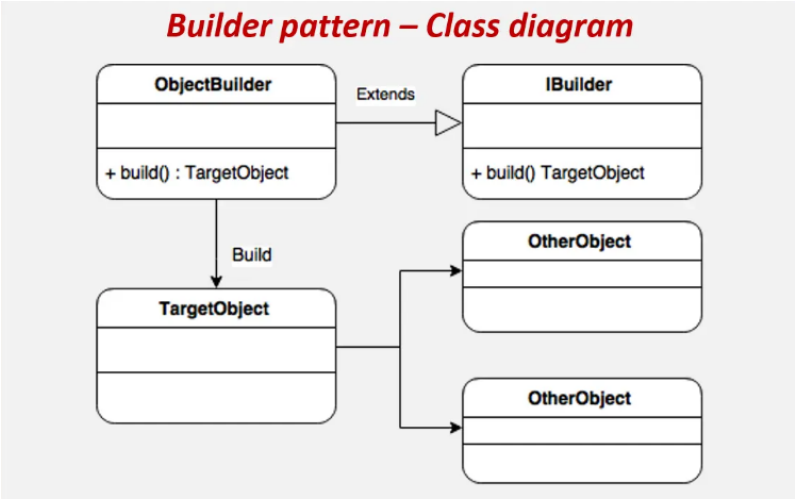


Figura 4.21: Esquema patrón adaptador [33].

Template Method

El patrón de diseño *Template Method* es un patrón de comportamiento que permite definir el esqueleto de un algoritmo en una clase base, dejando que los métodos específicos personalicen ciertos pasos sin alterar la estructura general.

El propósito principal para el uso de este patrón es definir una estructura común, proporcionando un esqueleto para un algoritmo que se mantenga para todas las implementaciones, permitiendo cierta personalización y evitando código duplicado.

En el contexto del proyecto se ha utilizado para una serie de querys custom, en la capa de infraestructura out, permitiendo realizar consultas personalizadas en el mismo código sin duplicado y de fácil mantenimiento. Utilizando el mismo esquema principal para todas las subconsultas.

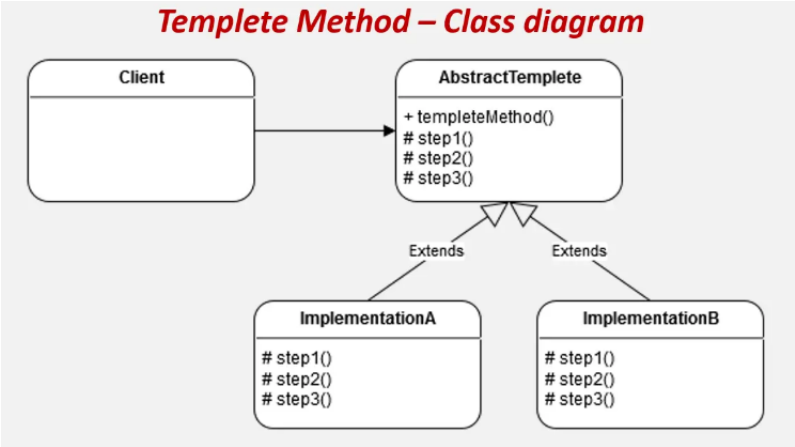


Figura 4.22: Esquema patrón Template Method [34].

4.5. Diagrama de paquetes

4.5.1. Diagrama de paquetes entre submódulos de user

En la Figura 4.23 se pueden observar las relaciones entre los diferentes módulos dentro del módulo `user`, mostrando las interacciones entre los módulos `advertisement`, `favorite`, `transaction` y `user`. A continuación, se detallan cómo se interrelacionan estos módulos y el porqué de estas relaciones.

Relación entre `advertisement` y `favorite`

Los módulos `advertisement` y `favorite`, ambos gestionan anuncios, pero con finalidades diferentes. El módulo `advertisement` es responsable de la creación, visualización y gestión de los anuncios, mientras que el módulo `favorite` gestiona los anuncios que un usuario ha marcado como favoritos. La relación entre estos módulos es bidireccional, debido a que el módulo `advertisement` consulta el estado de los anuncios favoritos a través de `favorite`, y el módulo `favorite` depende de los anuncios proporcionados por `advertisement` para determinar qué anuncios marcar como favoritos.

Relación entre `advertisement` y `transaction`

El módulo `advertisement` y el módulo `transaction` están relacionados debido a la necesidad de realizar una transacción sobre un anuncio. Cuando un usuario decide alquilar un anuncio, se crea una transacción asociada a ese anuncio en particular. El módulo `transaction` consulta el módulo `advertisement` para validar que el anuncio exista y que sea válido para realizar una transacción.

Relación entre `user` y los módulos `advertisement`, `favorite` y `transaction`

El módulo `user` presenta relación con el resto de módulos debido a que un usuario tiene que estar registrado para interactuar con cualquiera de los submódulos. En el caso de `advertisement`, para la creación de un anuncio; en el caso de `favorite` para poner un anuncio como favorito; y por último, en el caso de `transaction`, para generar una transacción a través de un anuncio. Por lo tanto, la relación de user con el resto de los submódulos es indispensable para un funcionamiento coherente.

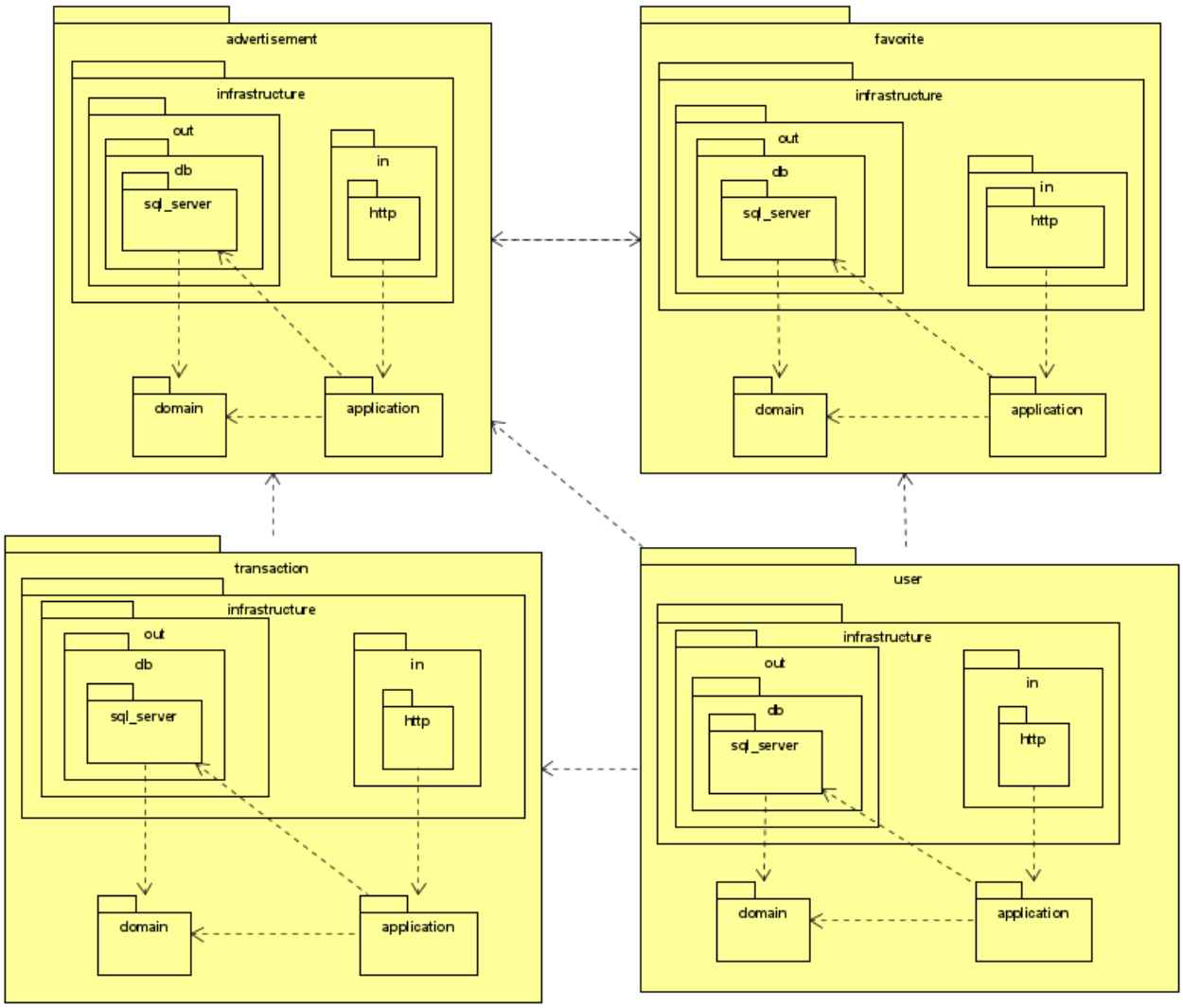


Figura 4.23: Diagrama de paquetes, relación entre submódulos user

4.5.2. Diagrama de capas en el módulo de `advertisement`

En la Figura 4.24 se muestra la estructura interna del módulo `advertisement`, que sigue el patrón de arquitectura hexagonal. En este diagrama se reflejan las diferentes capas dentro de la arquitectura. A continuación se muestran las diferentes relaciones entre las capas, las cuales son aplicables también a otros submódulos del sistema.

Relaciones entre capas

El módulo `advertisement` está dividido en diferentes capas, las cuales siguen el principio de separación de responsabilidades, una característica muy importante de la arquitectura hexagonal. Estas son sus capas:

- **Capa in:** Esta capa es responsable de recibir las peticiones externas. En este caso, incluye los con-

troladores HTTP dentro de la infraestructura del módulo `advertisement`. Los controladores gestionan las peticiones y las transmiten a las capas correspondientes para su procesamiento.

- **Capa out:** Esta capa maneja las salidas hacia componentes externos, como en este caso, la base de datos `sql_server`. Aquí se encuentra el repositorio `AdvertisementRepository`, que interactúa con la base de datos para almacenar, recuperar y modificar los anuncios. La capa `out` comunica las entidades del dominio con la infraestructura, en este caso, utilizando JPA y un repositorio específico para SQL Server.

- **Capa application:** En esta capa se encuentra la lógica de negocio. Aquí se manejan las operaciones y transformaciones de los datos que provienen de la capa de entrada. Es la encargada de la creación de anuncios y de interactuar con la capa `domain` para acceder a las entidades y persistir los datos.

- **Capa domain:** En esta capa se encuentran las entidades centrales del sistema. Aquí se definen los objetos que representan el modelo de negocio, así como las reglas y comportamientos que deben cumplirse para manipularlos. La capa de dominio es independiente de las tecnologías externas,lo que facilita posibles cambios de tecnología futuros o modificaciones internas.

Interacción con el módulo `security` y `password`

Además de las capas del módulo `advertisement`, se incluyen dos elementos importantes relacionados con la seguridad:

- **Capa security:** Esta capa es esencial para garantizar que solo los usuarios autenticados puedan interactuar con el sistema. Mediante esta capa se valida si el usuario está registrado, gestionando la autenticación y autorización de las peticiones, lo cual es crucial para la seguridad del sistema.

- **Capa password:** Aunque no se utiliza directamente en este ejemplo, la capa `password` se encarga de encriptar las contraseñas antes de almacenarlas en la base de datos una vez que el usuario se ha registrado en el sistema y comprobar que la contraseña introducida en el inicio y la encriptada en la base de datos son las mismas. Esto asegura que las contraseñas se mantengan seguras y que no se almacenen en texto plano.

Librerías utilizadas

El módulo de `advertisement` hace uso de varias librerías clave para su funcionamiento:

- **Spring Data JPA:** Esta librería es utilizada para interactuar con la base de datos, facilitando las operaciones de persistencia de datos mediante una interfaz simplificada que abstrae las consultas SQL.

- **Spring Core:** Base del framework Spring, utilizada para la inyección de dependencias y la gestión del ciclo de vida de los componentes del sistema.

- **Spring Security:** Esta librería se utiliza en la capa de seguridad para gestionar la autenticación y autorización de los usuarios.

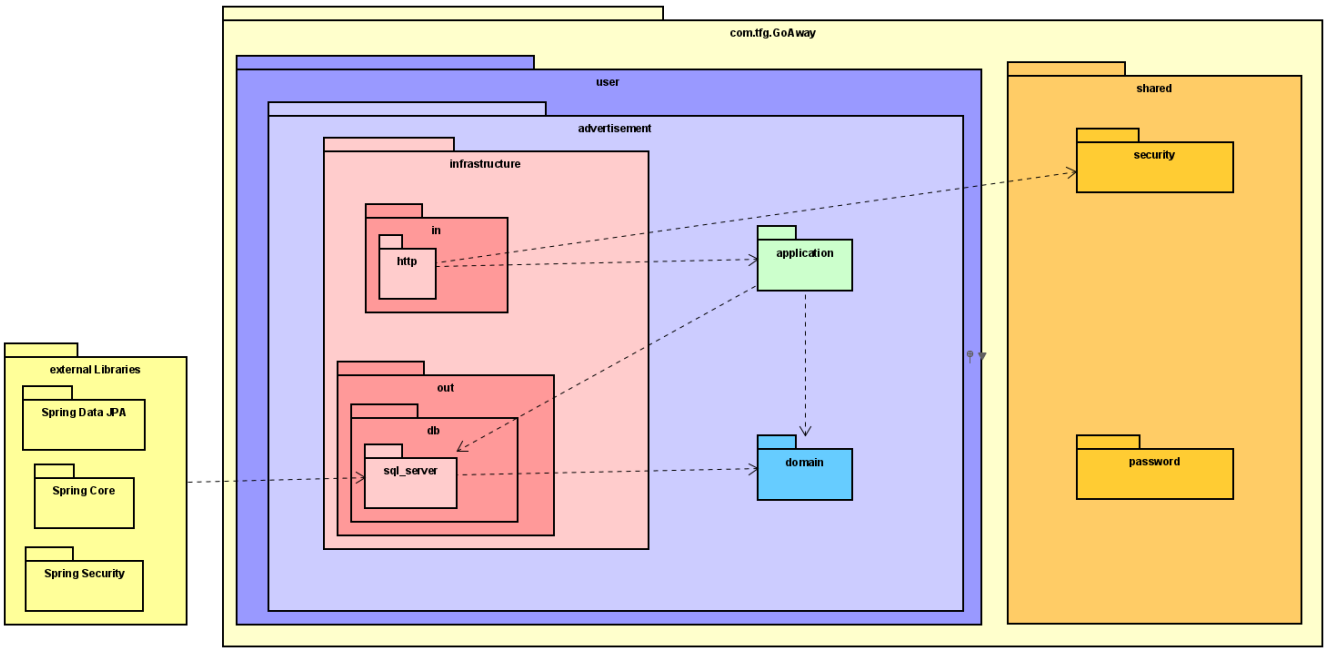


Figura 4.24: Diagrama de capas, módulo de `advertisement`

4.5.3. Diagrama de paquetes a nivel de clases

El diagrama de la figura 4.25 muestra las relaciones entre las diferentes clases del módulo `advertisement`. En este, se refleja la estructura y las relaciones de las diferentes clases. Se ha hecho únicamente el diagrama de este submódulo y de este método por falta de tiempo para desarrollar el resto. En los demás módulos y métodos se comporta de una manera similar.

Se encuentra dividido en varias capas que ya se han reflejado con anterioridad en la figura 4.23:

- **Infraestructura/out:** En esta capa están las clases responsables de la interacción con la base de datos, como `SqlServerAdvertisementRepository`, que maneja la persistencia de los anuncios en la base de datos, y `AdvertisementPhotoService`, que se encarga de gestionar las fotos asociadas a los anuncios. Además, se encuentra la clase `AdvertisementRepositoryMapper`, que realiza la conversión entre las entidades de la base de datos y los objetos de negocio.

- **Infraestructura/in:** En esta capa se encuentra la clase `AdvertisementCreatePostController`, encargada de manejar las solicitudes HTTP para la creación de anuncios, delegando la lógica en las clases de la capa de aplicación.

- **Aplicación:** En esta capa se encuentran las clases de lógica de negocio, como `AdvertisementCreate`, que gestiona la creación de un nuevo anuncio. La clase `AdvertisementCreateMapper` se encarga de mapear los objetos de entrada y salida, mientras que `AdvertisementCreateResponse` contiene la respuesta que se devuelve tras la creación del anuncio. La clase `AdvertisementCreateRecord` es la que

contiene la información necesaria para crear un anuncio.

- **Dominio:** Aquí se encuentran las clases que representan el dominio del negocio. `Advertisement` corresponde con la clase principal, la cual representa el anuncio. También tiene clases asociadas, como pueden ser `AdvertisementCategory`, `AdvertisementCondition`, y `AdvertisementPhoto`, que modelan los atributos y las relaciones específicas de un anuncio.

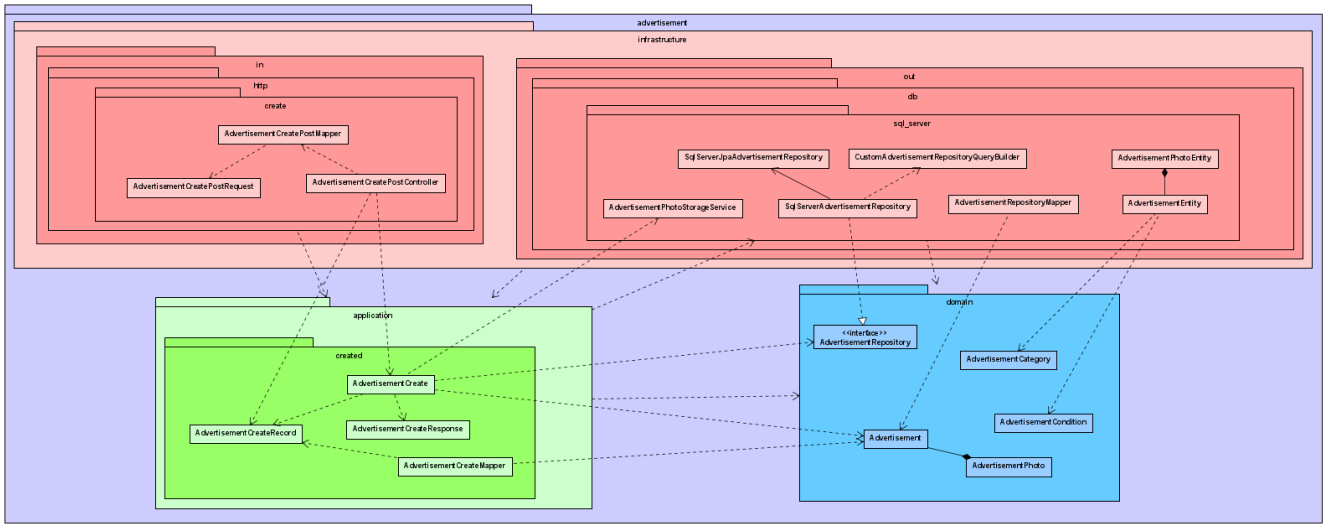


Figura 4.25: Diagrama de clases del submódulo user

4.6. Diagrama de despliegue

Un diagrama de despliegue [35] es un tipo de diagrama utilizado para modelar la disposición física de los artefactos de software en los componentes de hardware, conocidos como nodos. Además, representa las configuraciones físicas de software y hardware esenciales para la ejecución, el funcionamiento y la escalabilidad del sistema.

En el diagrama de despliegue mostrado en la Figura 4.26, se muestra la arquitectura del sistema en un entorno de desarrollo local. Se utiliza en este caso un dispositivo Windows, donde se ha instalado Docker Desktop para gestionar los contenedores Docker. Docker Desktop configura automáticamente Windows Subsystem for Linux (WSL), proporcionando un entorno Linux compatible con Docker, desde donde se ejecutan los contenedores del sistema.

El sistema está compuesto por dos contenedores Docker, los cuales son ejecutados dentro de WSL en el dispositivo Windows. El contenedor *back* utiliza Java 17 como entorno de ejecución para ejecutar el componente *GoAway API*, que presenta una API RESTful para la comunicación con el sistema. Dentro de este mismo contenedor, se ejecuta MySQL, que aloja la base de datos *goaway database*, encargada de gestionar el almacenamiento de datos del sistema. La comunicación entre *GoAway API* y la base de datos se realiza internamente dentro del mismo contenedor.

El otro contenedor es el contenedor *front* que utiliza Angular para desplegar el componente *GoAway*,

y que representa la interfaz de usuario de la aplicación web. Este componente se comunica con el back-end a través de conexiones HTTP, utilizando el servicio *APIService* para interactuar con la *GoAway API* del contenedor *back*.

Además, el diagrama incluye un dispositivo Windows (que puede ser el mismo dispositivo de desarrollo u otro) que actúa como cliente, ejecutando un navegador web etiquetado como *Navegador*. Desde este navegador, los usuarios acceden a la aplicación web desplegada en el contenedor *front* mediante conexiones HTTP, permitiendo la interacción con el sistema de manera local durante el desarrollo.

En resumen, el diagrama de despliegue representa un sistema distribuido que utiliza dos contenedores Docker para separar el back-end y el front-end, ejecutados dentro de WSL en un dispositivo Windows durante el desarrollo. El back-end, implementado con Java 17 y una base de datos MySQL integrada, expone una API que es consumida por el front-end, desarrollado en Angular. Los usuarios acceden a la aplicación web desde un navegador en un dispositivo Windows mediante conexiones HTTP.

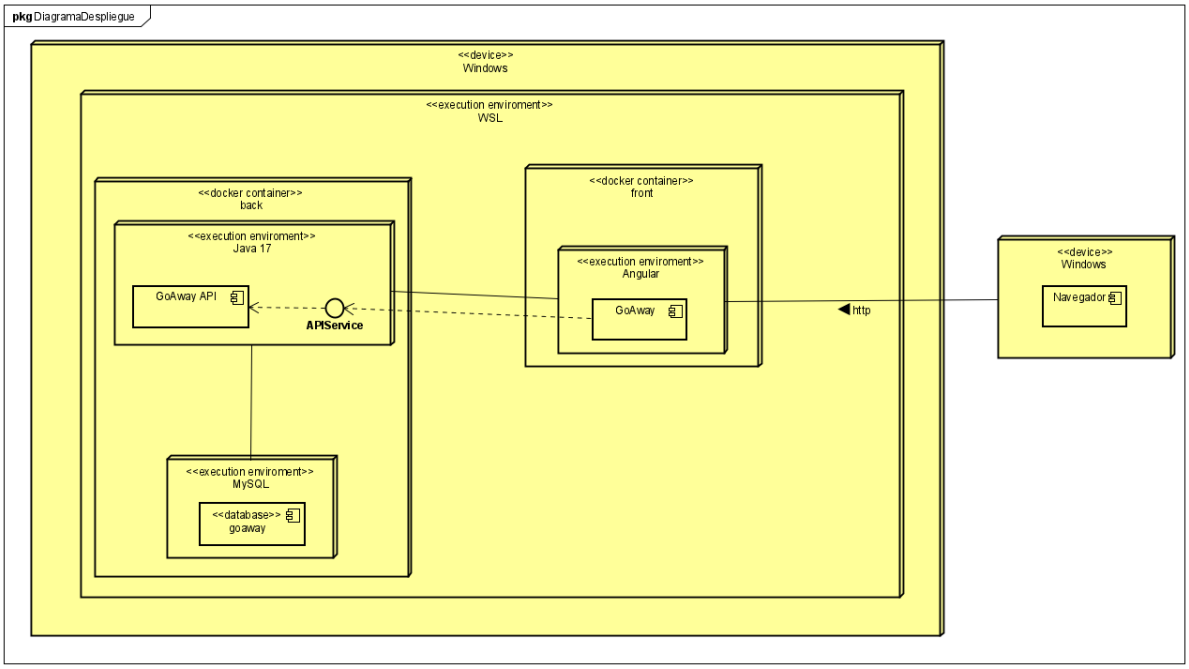


Figura 4.26: Diagrama despliegue [35].

Capítulo 5

Pruebas

5.1. Pruebas

A continuación se describirá la estrategia seguida para realizar las pruebas en las dos partes del proyecto, el backend y el frontend. Dado el tiempo limitado del proyecto, se ha optado por realizar un par de ejemplos de implementación. Desarrollar las pruebas para un sistema completo requiere una gran inversión de tiempo y recursos. Por este motivo, se priorizan pruebas automáticas que nos facilitarán su creación. En este caso, se emplea JUnit [36] para el backend desarrollado en Java con Spring Boot y Jasmine [37] con ayuda de Karma [38] para el código frontend desarrollado en Angular 19.

La elección de la arquitectura hexagonal ha facilitado en gran medida la creación y ejecución de las pruebas. Esta arquitectura separa las responsabilidades entre dominios, puertos y adaptadores, lo que nos ayuda a simplificar la simulación de dependencias y la validación de los componentes. En cuanto al frontend, la versión de Angular 19, que es con la que se ha trabajado, tiene una estructura modular con soporte nativo para pruebas, lo que también favorece este proceso.

Sin embargo, al tiempo limitado del proyecto, no se han podido realizar pruebas de aceptación del código, las cuales serían necesarias para verificar el sistema en su conjunto y asegurar que el código cumpla con los requisitos del usuario y del negocio.

5.1.1. Pruebas Backend

Para validar el correcto funcionamiento del backend, se ha utilizado el framework JUnit [36] junto con Mockito [39] para las pruebas unitarias. También se ha empleado Spring Boot Test para facilitar la ejecución de estas pruebas en un entorno controlado. JUnit permite definir y ejecutar tests de manera estructurada, mientras que Mockito facilita la simulación de dependencias, como repositorios o servicios externos.

Pruebas Unitarias

El objetivo de las pruebas unitarias es comprobar el comportamiento de componentes individuales, como servidores del dominio sin depender de recursos externos. Para realizar los tests correctamente se van a emplear *mocks*, cuya función es simular el comportamiento de las dependencias, como en este caso, el repositorio de usuarios (UserRepository) o el mapper (UserRegisterMapper).

En este proyecto se ha decidido centrarse únicamente en pruebas unitarias para la capa backend, descartando las pruebas de integración. Esta decisión responde a la necesidad de mantener el alcance del proyecto dentro de los límites temporales establecidos, priorizando una validación precisa de la lógica interna del dominio.

Un ejemplo de prueba unitaria se encuentra en la clase UserRegisterTest, donde se comprueba el correcto funcionamiento del servicio de registro de usuarios (UserRegister). Esta clase cubre varios escenarios relevantes:

- **El usuario ya existe**: el repositorio simula que devuelve un usuario al buscar por email. Se espera una excepción con el mensaje *"El usuario ya existe"* y se comprueba que ni el mapper ni el repositorio realizan acciones adicionales.

```java
@Test
void testRegisterUserAlreadyExists() throws Exception {
    User mockUser = mock(User.class);
    when(userRepository.finder(any(UserCriteria.class))).thenReturn(Optional.of(mockUser));

    Exception exception = assertThrows(Exception.class, () -> {
        userRegister.register(validRecord);
    });
    assertEquals("El usuario ya existe", exception.getMessage());
    verify(userRepository).finder(any(UserCriteria.class));
    verify(userRegisterMapper, never()).toDomain(any(), any());
    verify(userRepository, never()).save(any());
}
```

Figura 5.1: Test que verifica el caso en el que el usuario ya existe.

- **Registro con email nulo**: se prueba que, al recibir un email nulo, se lanza una excepción con el mensaje *"Usuario no encontrado"* sin llamar a ningún método del repositorio.

```java
@Test
void testRegisterNullEmail() throws Exception {
    UserRegisterRecord invalidRecord = UserRegisterRecord.builder()
            .email(email:null)
            .name(name:"Test User")
            .password(password:"password123")
            .contactNumber(contactNumber:1234567890L)
            .build();

    Exception exception = assertThrows(Exception.class, () -> {
        userRegister.register(invalidRecord);
    });
    assertEquals("Usuario no encontrado", exception.getMessage());
    verify(userRepository, never()).finder(any(UserCriteria.class));
    verify(userRegisterMapper, never()).toDomain(any(), any());
    verify(userRepository, never()).save(any());
}
```

Figura 5.2: Test que simula un registro con el campo email nulo.

- **Registro válido pero con error**: aunque el usuario no existe, el flujo se interrumpe antes de guardar. Se lanza una excepción y no se llama al mapper ni al método `save()`.

```java
@Test
void testRegisterSuccess() {
    when(userRepository.finder(any(UserCriteria.class))).thenReturn(Optional.empty());

    assertThrows(NoSuchElementException.class, () -> userRegister.register(validRecord));

    verify(userRepository).finder(any(UserCriteria.class));
    verify(userRegisterMapper, never()).toDomain(any(), any());
    verify(userRepository, never()).save(any());
}
```

Figura 5.3: Test que representa un registro aparentemente válido que falla antes de guardar.

- **Error al guardar el usuario**: test muy similar al anterior que simula un fallo interno. Se espera la misma excepción, confirmando que no se ha llegado al guardado.

```java
@Test
void testRegisterSaveError() {
    when(userRepository.finder(any(UserCriteria.class))).thenReturn(Optional.empty());

    assertThrows(NoSuchElementException.class, () -> userRegister.register(validRecord));

    verify(userRepository).finder(any(UserCriteria.class));
    verify(userRegisterMapper, never()).toDomain(any(), any());
    verify(userRepository, never()).save(any());
}
```

Figura 5.4: Test que simula un error al intentar guardar un nuevo usuario.

Estas pruebas aseguran que el servicio de registro se comporta correctamente ante distintos escenarios y errores, sin necesidad de depender de una base de datos real ni de otros servicios externos.

Pruebas de Rendimiento

Debido a limitaciones de tiempo, no se han implementado pruebas de rendimiento en esta versión del proyecto. Sin embargo, se contempla su incorporación futura con herramientas como Apache JMeter [40], que permitirían evaluar métricas clave como el tiempo de respuesta, el consumo de recursos y la escalabilidad bajo carga.

5.1.2. Pruebas Frontend

Por otra parte, las pruebas del frontend, desarrollado con Angular 19, se han implementado pruebas unitarias que han ayudado a verificar el correcto funcionamiento del componente de registro de usuario. Para realizar las pruebas se ha utilizado el entorno de pruebas de Angular, el cual combina Jasmine [37] para la sintaxis de tests y Karma [38] como ejecutor de navegador. Además, se ha utilizado el módulo HttpClientTestingModule, que simula las peticiones HTTP sin necesidad de comunicación real con el backend.

Justificación de pruebas unitarias

En un inicio, se contempló la posibilidad de incluir pruebas de tipo integración y end-to-end. Sin embargo, por falta de tiempo y alcance, se ha decidido optar por centrarse exclusivamente en las pruebas unitarias. Este tipo de pruebas son más eficientes en proyectos de desarrollo individual, debido a que permiten validar rápidamente la lógica interna del componente, lo que asegura su escalabilidad sin necesidad de infraestructura adicional.

Componente probado: `RegisterComponent`

El componente `RegisterComponent` es el encargado de gestionar el formulario de registro. Cuenta con validaciones reactivas, envío de datos al backend y redirección en caso de éxito o cancelación. Se han desarrollado un total de seis pruebas unitarias, las cuales cubren los siguientes escenarios:

- El componente se crea correctamente.
- Al pulsar el botón de cancelar, se redirige a la página principal.
- El formulario se inicializa con todos los campos vacíos.
- Si el formulario es inválido, no se envía la petición al backend.
- Si el registro es exitoso, se guarda el token y se redirige al inicio.
- Si el registro falla, se muestra el mensaje de error correspondiente.

```
it('should submit and navigate on successful register', fakeAsync(() => {
  component.registerForm.setValue({
    name: 'Juan',
    email: 'juan@example.com',
    password: '123456',
    contactNumber: '123456789'
  });

  component.onSubmit();

  const req = httpMock.expectOne('/auth/user/register');
  req.flush({ token: 'abc123' });

  tick();
  expect(sessionStorage.getItem('token')).toBe('abc123');
  expect(routerSpy.navigate).toHaveBeenCalledWith(['/']);
}));
```

Figura 5.5: Prueba que simula un registro exitoso y redirección a la pantalla principal.

En la Figura 5.5, se observa el código de una prueba que verifica el comportamiento esperado cuando el usuario introduce todos los datos válidos y el backend responde con un token. Además, se comprueba que el token se almacena correctamente en el `sessionStorage` y que se navega a la ruta principal mediante un `RouterSpy`.

```
it('should show error message on failed register', fakeAsync(() => {
  component.registerForm.setValue({
    name: 'Ana',
    email: 'ana@example.com',
    password: '123456',
    contactNumber: ''
  });

  component.onSubmit();

  const req = httpMock.expectOne('/auth/user/register');
  req.flush({ error: 'Error' }, { status: 400, statusText: 'Bad Request' });

  tick();
  expect(component.errorMessage).toBe('Error al registrar. Intenta con otro email.');
}));
```

Figura 5.6: Prueba que simula un error de registro y muestra el mensaje correspondiente.

La Figura 5.6 muestra el código de prueba de un caso donde el backend responde con un error 400. Esta prueba verifica que no se navega y que el mensaje de error que se muestra en la pantalla es *"Error al registrar. Intenta con otro email."*

Simulación de peticiones HTTP

Gracias al módulo `HttpClientTestingModule`, no es necesario realizar peticiones reales al backend debido a que las respuestas son interceptadas y simuladas mediante el servicio `HttpTestingController`. Esto permite testear la lógica del componente sin depender de un servidor activo.

Manejo del enrutamiento

Para no redirigir realmente durante los tests, se ha utilizado un `spy` del servicio `Router`. Este espía permite comprobar si el componente intenta redirigir al usuario a una ruta concreta sin ejecutar la navegación.

Cobertura y mantenimiento

Gracias a estas pruebas, se proporciona una base sólida para validar cambios futuros en el componente sin romper la funcionalidad ya existente. Además, al estar de manera aislada, son fáciles de mantener y rápidas de ejecutar.

Capítulo 6

Test de usabilidad

6.1. Test de usabilidad

La usabilidad de una aplicación es un pilar fundamental debido a que determina la facilidad que tienen los usuarios en interactuar con la aplicación realizando ciertas tareas. Este test tiene como objetivo evaluar la experiencia de nuevos usuarios de la aplicación web, centrándose en su facilidad de uso, navegabilidad y claridad.

El test se enfocará en medir la eficiencia, eficacia y satisfacción de los usuarios al realizar tareas fundamentales dentro de la aplicación, lo cual nos facilitará la identificación de posibles obstáculos en la interacción.

6.1.1. Objetivo

El objetivo principal es realizar un análisis de la capacidad que tienen los usuarios para completar tareas básicas en la aplicación sin necesidad de asistencia externa. En particular, se evalúan los siguientes aspectos:

- **Comprensión de la interfaz**: ¿Es un diseño intuitivo y fácil de entender?
- **Fluidez de la navegación**: ¿Los usuarios encuentran lo que buscan con dificultad?
- **Dificultades encontradas**: ¿Realizan pasos innecesarios o se sienten confusos en el proceso?
- **Tiempo y numero de acciones**: ¿Cuánto tiempo y cuantos clics requiere para realizar una acción?
- **Satisfacción del usuario**: ¿Cumple la aplicación con las expectativas y es agradable de usar?

6.1.2. Participantes y procedimiento

El test se va a llevar a cabo con dos usuarios, los cuales van a interactuar con la aplicación por primera vez. Se ha escogido a usuarios jóvenes debido a que es una aplicación enfocada a un perfil estudiantil que busca y alquila anuncios para decorar su vivienda temporal.

Estos usuarios interactuarán con la aplicación utilizando una base de datos preconfigurada con datos ficticios generados mediante seeders. Se les proporcionará una breve descripción sobre la aplicación, dándoles instrucciones detalladas sobre las tareas que deben realizar, permitiendo evaluar sus acciones.

Durante la prueba se tomarán notas de sus comportamientos, lo que permitirá analizar la accesibilidad y usabilidad del sistema.

6.1.3. Métricas a evaluar

Se han definido varias métricas que se utilizarán para analizar y estudiar la experiencia del usuario.

- **Tiempo empleado**: Tiempo que tarda el usuario en realizar cada tarea.
- **Número de clics**: Cantidad de interacciones que necesita para completar cualquier acción.
- **Errores encontrados**: Errores cometidos, clasificándolos por problemas de comprensión o de interfaz.
- **Comentarios y observaciones**: Feedback proporcionado por el usuario durante la prueba.
- **Encuesta de satisfacción**: Opiniones subjetivas del usuario al finalizar la prueba.

6.1.4. Tareas a realizar

Se han seleccionado dos tareas que son fundamentales dentro de la aplicación, como son publicar un anuncio y buscar un anuncio y solicitar su alquiler.

Tarea 1: Publicar un anuncio de alquiler de un mueble

1. Registrar o iniciar sesión.
2. Acceder a la sección de publicar un anuncio.
3. Rellenar el formulario para la creación del mismo (Titulo, Descripción, precio, tipo, estado y fotos).
4. Confirmar la publicación del anuncio.

Tarea 2: Buscar y solicitar el alquiler de un mueble

1. Navegar por el catálogo de anuncios.
2. Aplicar filtros de búsqueda.
3. Seleccionar un mueble y analizarlo con detalle, todas sus fotos, la descripción al completo, etc.
4. Enviar la solicitud de alquiler.
5. Comprobar el estado de la solicitud en la sección de transacciones.

6.2. Resultados del Test de Usabilidad

A continuación se presentan los resultados obtenidos del test de usabilidad realizado a la aplicación de alquiler de un anuncio. Este análisis sigue la estructura previamente definida en la sección 6.1, se tienen en cuenta métricas relacionadas con la eficiencia, eficacia y satisfacción del usuario.

6.2.1. Tarea 1: Publicar un anuncio de alquiler de un mueble

Publicar anuncio	Usuario 1	Usuario 2	Baremo
Éxito	Sí	Sí	-
Tiempo empleado	55s	48s	45s
Número de clics	13	13	12
Errores encontrados	1	0	0
Interfaz intuitiva	Sí	Sí	-
Necesidad de asistencia	Sí	No	-
Satisfacción general	Alta	Alta	-

Tabla 6.1: Resultados del test de usabilidad para la tarea de publicar un anuncio.

6.2.2. Tarea 2: Buscar y solicitar el alquiler de un mueble

Buscar y solicitar anuncio	Usuario 1	Usuario 2	Baremo
Éxito	Sí	Sí	-
Tiempo empleado	40s	35s	30s
Número de clics	12	10	9
Errores encontrados	1	0	0
Interfaz intuitiva	Sí	Sí	-
Necesidad de asistencia	No	No	-
Satisfacción general	Alta	Alta	-

Tabla 6.2: Resultados del test de usabilidad para la tarea de buscar y solicitar un anuncio.

6.2.3. Conclusiones de la evaluación

A partir de los resultados obtenidos en ambas tareas, se puede observar y concluir que los usuarios fueron capaces de completar correctamente las tareas propuestas y mostraron una satisfacción general con la aplicación. La navegabilidad resultó intuitiva y visualmente clara, facilitando la interacción con el sistema.

Sin embargo, los usuarios mencionaron algunos puntos específicos donde la aplicación podría mejorar:

- **Categorías más específicas y numerosas**: Los usuarios sugirieron que sería más útil tener la posibilidad de concretar más el tipo de producto que es contando con más categorías y subcategorías más concretas.
- **Filtros más avanzados**: En relación con la sugerencia anterior, los usuarios indicaron que añadir filtros más complejos mejoraría notablemente la experiencia de búsqueda. Propusieron incluir una lupa para buscar por palabras concretas así como un filtro de rango de precios.

Estas conclusiones serán útiles para priorizar futuras mejoras que optimicen aún más la usabilidad y accesibilidad general de la aplicación.

Capítulo 7

Conclusiones

Tras finalizar el Trabajo de Fin de Grado, se pueden obtener varias conclusiones importantes que reflejan tanto los conocimientos aprendidos como el valor que tiene el proyecto desarrollado.

En primer lugar, se ha logrado crear el código completo de una aplicación web útil para el alquiler de mobiliario para estudiantes. Esta aplicación resuelve una necesidad real: disponer de mobiliario temporal sin recurrir a la compra. Esto no solo promueve el ahorro económico por parte de los usuarios, sino que también fomenta la economía circular, lo que conlleva a una reutilización de recursos.

El desarrollo del código backend se ha realizado aplicando una arquitectura hexagonal, lo que, además de facilitar la creación del código por su simplicidad y separación entre la lógica del dominio y los elementos de infraestructura, también asegura en un futuro la mantenibilidad del mismo, así como la adaptación del código a posibles cambios y ampliaciones futuras de la aplicación. Por otro lado, esto ayuda a la integración fácil de tecnologías externas sin comprometer el núcleo.

Desde el punto de vista tecnológico, se ha trabajado con la última versión de Angular para el desarrollo del frontend y Java con SpringBoot para el backend, en donde se han aplicado patrones y buenas prácticas simulando un entorno profesional real. Esta combinación de tecnologías ha permitido la construcción de una solución moderna, robusta y escalable.

Durante el proyecto también se han presentado y superado una serie de desafíos y problemas, como por ejemplo el manejo de las imágenes en el sistema de anuncios, la validación de los formularios y el correcto funcionamiento del login y registro. Enfrentarse a estos desafíos ha contribuido al crecimiento profesional y personal.

Finalmente, el proyecto ha cumplido con los objetivos marcados al principio del documento (ver Sección 1.4): se ha creado un producto mínimo viable que resuelve un problema concreto, como es el de amueblar viviendas temporales de manera económica, de manera eficaz, trabajando con tecnologías actuales en un entorno estructurado y profesional. Todo en su conjunto ha supuesto una experiencia formativa integral, permitiendo afrontar con mayor preparación futuros retos profesionales y personales del desarrollo de software.

7.1. Trabajo futuro

Nuevas funcionalidades:

- **Implementación de un sistema de chat interno:** Un chat interno facilitaría la comunicación entre los usuarios, siendo directa y en tiempo real.
- **Sistema avanzado de valoraciones y reseñas:** Las valoraciones tanto de anuncios como de muebles mejoraría la experiencia de los usuarios.
- **Geolocalización:** La posibilidad de ver la ubicación de un anuncio sería de gran utilidad, añadiendo filtros por cercanía para una mayor interacción del usuario.
- **Sistema de notificaciones:** Añadir un sistema de notificaciones, que informe en tiempo real sobre las solicitudes de alquiler, cambios en alguna transacción o mensajes de recibidos de otros usuarios.

Mejoras en funcionalidades ya existentes:

- **Nuevos filtros:** Añadir diferentes tipos de filtro, como pueden ser por rango de precios, búsqueda por palabras clave y categorías más especificas para mejorar la búsqueda de muebles.

Integración con sistemas reales:

- Proponer colaboraciones con universidades, residencias estudiantiles o similares, estableciendo una plataforma de alquiler temporal de mobiliario para los estudiantes.
- Integrar un sistema de pagos que facilite y asegure las transacciones entre los usuarios.

Pruebas exhaustivas:

- Realizar pruebas del código completo para verificar su correcto funcionamiento. Además se podría involucrar a un gran número de usuarios reales que prueben la aplicación para detectar posibles fallos y así poder mejorar la funcionalidad.

Mejora de la interfaz de usuario:

- Teniendo en cuenta las sugerencias recogidas en el test de usabilidad y el feetback de los usuarios, hacer una mejora y optimización del sistema.
- Modificar el diseño para añadir uno más atractivo, intuitivo y moderno.

Apéndices

Apéndice A

Acrónimos

- **API**: Application Programming Interface.
- **HTTP**: Hypertext Transfer Protocol.
- **JSON**: JavaScript Object Notation.
- **JWT**: JSON Web Token.
- **REST**: Representational State Transfer.
- **URL**: Uniform Resource Locator.
- **CLI**: Command Line Interface.
- **ERD**: Entity Relationship Diagram.
- **WSL**: Windows Subsystem for Linux.

Apéndice B

Manual de despliegue

En este capítulo se describe el proceso para desplegar el sistema completo utilizando Docker en un entorno Windows. La aplicación está dividida en tres contenedores principales: base de datos MySQL, backend desarrollado en Spring Boot y frontend en Angular. Todo el código fuente se encuentra disponible públicamente en GitHub:

- **Repositorio completo:** `https://github.com/jaimec01/GoAway.git`

El sistema ha sido desarrollado utilizando Visual Studio Code como entorno principal, con soporte adicional de herramientas como Docker Desktop, MySQL Workbench y Postman para pruebas y consultas.

B.1. Instalación de Docker en Windows

Para ejecutar correctamente los contenedores, es necesario tener instalado Docker Desktop. Los pasos para su instalación son:

1. Descargar Docker Desktop desde su página oficial: `https://www.docker.com/products/docker-desktop/`.
2. Ejecutar el instalador y seguir los pasos del asistente.
3. Reiniciar el equipo si es requerido.
4. Verificar que Docker esté en funcionamiento observando el icono en la bandeja del sistema.

B.2. Clonación del proyecto

Con Docker instalado y funcionando, se debe clonar el proyecto desde GitHub. Para ello, desde una terminal o consola en Windows (o directamente desde Visual Studio Code), se ejecuta el siguiente comando:

```
git clone https://github.com/jaimec01/GoAway.git
```

Esto descargará todo el proyecto en el directorio local.

B.3. Despliegue del sistema

Desde la raíz del proyecto clonado, se puede levantar toda la aplicación (base de datos, backend y frontend) ejecutando:

```
docker-compose up
```

Esto creará y levantará automáticamente tres contenedores:

- `goaway-db`: Servicio de base de datos MySQL.
- `goaway-backend`: Aplicación Spring Boot expuesta en el puerto 8080.
- `goaway-frontend`: Aplicación Angular servida mediante Nginx en el puerto 4200.

B.4. Ficheros Docker

B.4.1. Dockerfile del backend

El backend está construido con Spring Boot y Java 17, empaquetado en un contenedor listo para producción y depuración:

B.4.2. Dockerfile del frontend

El frontend utiliza una imagen de Node para construir el proyecto Angular, y Nginx para servirlo en producción:

```
back > Dockerfile > ...
1   # Usa una imagen oficial de Java con soporte para Java 17
2   FROM amazoncorretto:17-alpine
3
4   # Establecer opciones de Java para habilitar el depurador remoto
5   ENV JAVA_TOOL_OPTIONS="-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:5005
6
7   # Copia el archivo JAR a la imagen
8   COPY target/GoAway-0.0.1-SNAPSHOT.jar app.jar
9
10  # Expone los puertos de la aplicación y del depurador
11  EXPOSE 8080 5005
12
13  # Ejecuta la aplicación
14  ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Figura B.1: Dockerfile backend

B.4.3. Archivo docker-compose.yml

El siguiente fichero define los servicios necesarios para ejecutar la aplicación completa:

B.5. Acceso a la aplicación

Tras el despliegue, se puede acceder a los siguientes servicios desde el navegador:

- **Frontend (aplicación web)**: `http://localhost:4200`
- **Backend (API REST)**: `http://localhost:8080`

Desde herramientas como Postman o MySQL Workbench también es posible hacer peticiones HTTP o consultar directamente la base de datos accediendo al puerto `3307` del host.

front > Dockerfile > ...

```dockerfile
# Etapa de build
FROM node:18 AS build
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm install
COPY . .
RUN npm run build --configuration=production

# Etapa de producción con Nginx
FROM nginx:alpine
COPY nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=build /app/dist/front /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Figura B.2: Dockerfile frontend

docker-compose.yml

```yaml
version: "3.9"
services:
  frontend:
    build:
      context: ./front
    container_name: goaway-frontend
    ports:
      - "4200:80"
    depends_on:
      - backend

  db:
    image: mysql:8.0
    container_name: goaway-db
    ports:
      - "3307:3306"
    environment:
      MYSQL_ROOT_PASSWORD: root
    volumes:
      - db_data:/var/lib/mysql

  backend:
    build:
      context: ./back
    container_name: goaway-backend
    ports:
      - "8080:8080" # Puerto de la aplicación
      - "5005:5005" # Puerto del depurador
    depends_on:
      - db
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://db:3306/goaway
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root
      SPRING_JPA_HIBERNATE_DDL_AUTO: update
    volumes:
      - D:/UNIVERSIDAD/TFG/GoAway/back/adsImages:/app/images

volumes:
  db_data:
```

Figura B.3: docker$_{compose.yml}$

Apéndice C

Manual de uso

Al abrir la aplicación, lo primero que ve el usuario es una pantalla inicial en la que aparecen todos los anuncios que hay publicados en la plataforma. (ver Figura C.1). En esta pantalla principal se puede realizar una ordenación de la fecha de modificación ascendente o descendente, filtrar por categoría o por estado del producto. Las demás acciones solo están habilitadas para los usuarios que están registrados en el sistema.

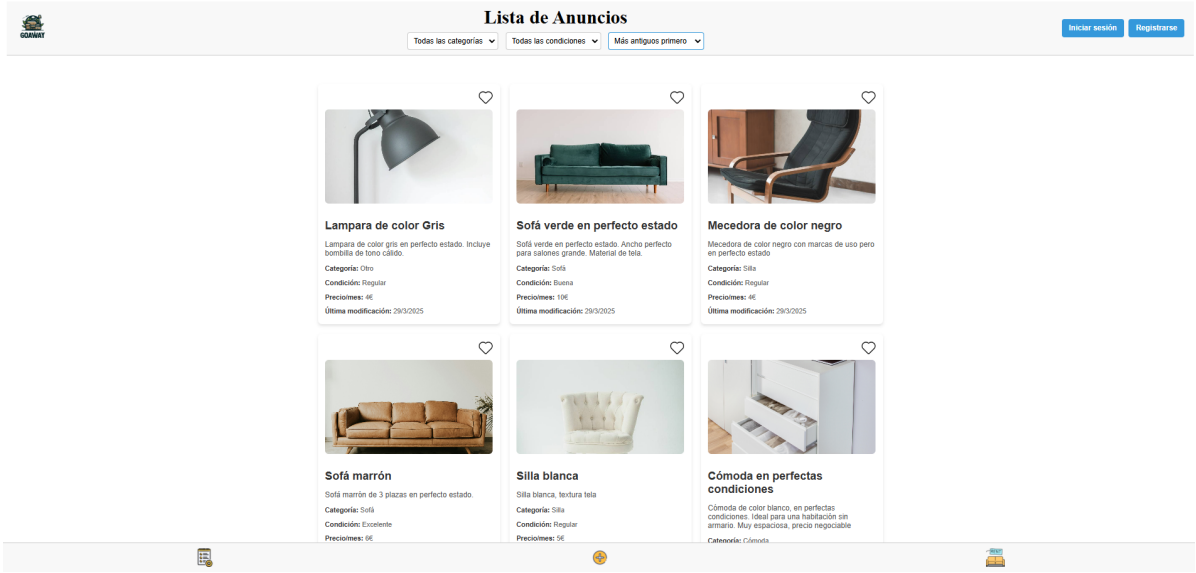
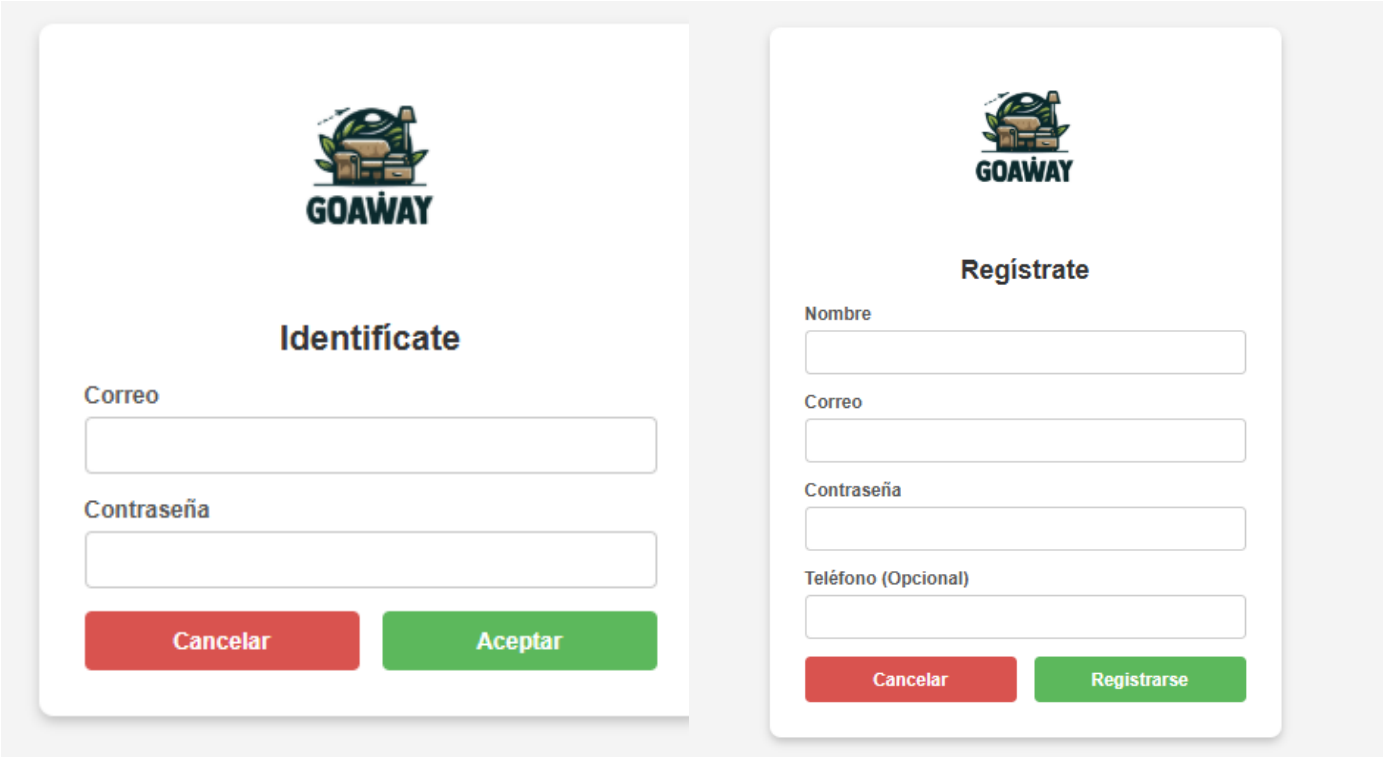


Figura C.1: Pantalla principal sin registro

El usuario puede registrarse o iniciar sesión si ya tiene una cuenta creada. Para el registro en el sistema, tal y como se muestra en la figura C.2b se precisa obligatoriamente del nombre de usuario, su correo y su contraseña, y de manera opcional el teléfono. Se valida que el correo no esté ya creado en la base de datos, que tenga un formato correcto y que la contraseña sea de al menos 4 caracteres. Por otro lado, en la figura C.2a es necesario únicamente el correo y la contraseña.

Una vez que el usuario se ha registrado o iniciado sesión en el sistema, se permite acceder al resto de la funcionalidad de la aplicación. Por un lado, la pantalla principal cambia ligeramente, como se



(a) Inicio de sesión (b) Registro

Figura C.2: Pantallas del proceso de acceso

puede observar en la figura C.3; se desbloquea la posibilidad de marcar como favorito los anuncios y, arriba a la derecha, se activa la posibilidad de ver los anuncios publicados por el usuario y cerrar sesión. Además, únicamente aparecen los anuncios que no pertenecen al usuario identificado, sino que se muestran el resto de anuncios publicados por los demás usuarios. Por otro lado, el botón de alquilar se hace visible, lo que permite a los usuarios crear una solicitud de alquiler de un anuncio.

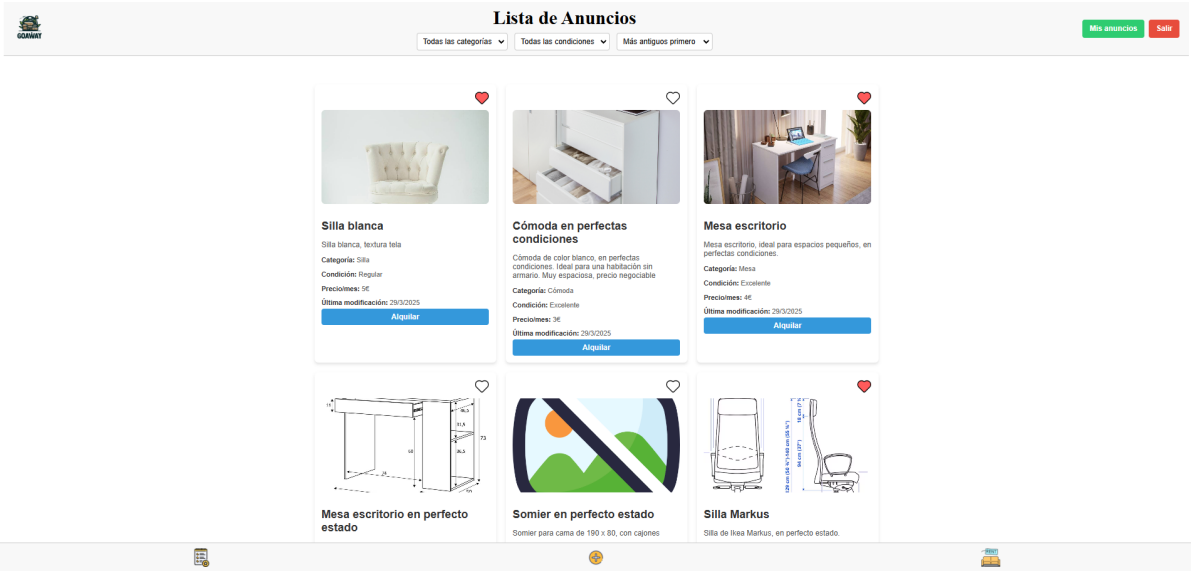
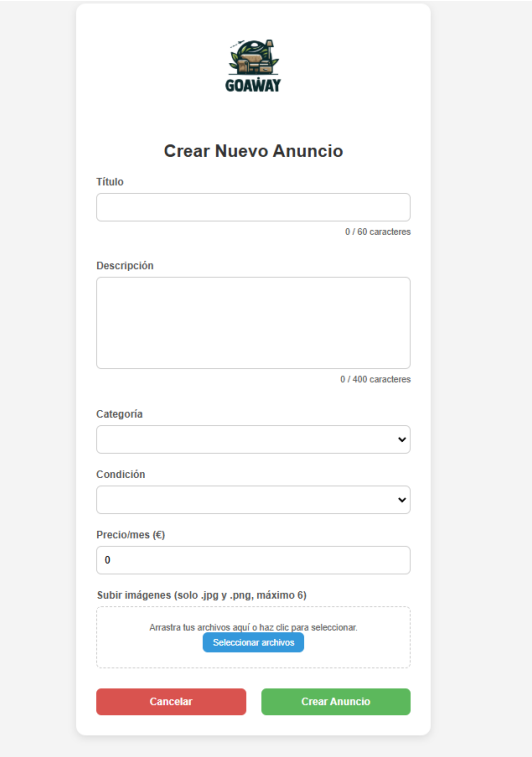
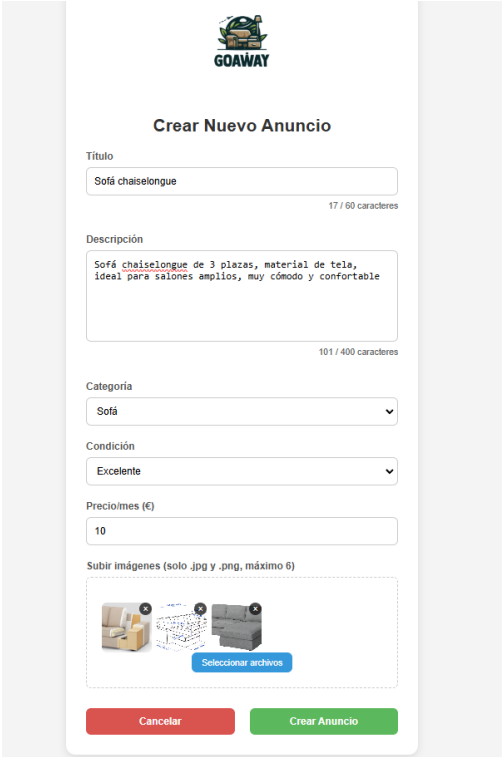


Figura C.3: Pantalla principal con registro

La figura C.4a muestra el formulario que hay que rellenar para llevar a cabo la creación de un anuncio y en la figura C.4b un ejemplo de cómo quedaría un anuncio listo para ser creado. No se permiten títulos de más de 60 caracteres y descripciones superiores a 400 caracteres, para no sobrecargar la base de datos en exceso. Por otro lado, el precio no permite que sea un valor negativo, siempre tiene que ser superior o igual a 0. Para agregar las imágenes, se pueden añadir o bien arrastrándolas al recuadro o bien abriendo el explorador de archivos pulsando en el botón "Seleccionar archivos".



(a) Creación de un anuncio



(b) Creación de un anuncio con datos

Figura C.4: Pantallas del proceso de creación de anuncio

En la lista de favoritos, como se puede observar en la figura C.5, es muy similar a la lista principal de anuncios, salvo que solo aparecen los anuncios que se tienen marcados como favoritos.

En el apartado de "Mis anuncios", aparecen todos los anuncios que ha publicado el usuario registrado, permitiéndole eliminarlos y/o modificarlos. (ver figura C.6).

Accediendo a la modificación de uno de los anuncios, se puede observar que se pueden modificar todos los campos, cambiar el título, la descripción, la categoría, la condición, el precio, añadir más imágenes y borrar alguna existente (hasta un máximo de 6 imágenes en total) y habilitar o deshabilitar el anuncio. Esta última acción nos permite hacer visible el anuncio para el resto de usuarios o no, lo cual es muy útil para cuando hay una transacción en curso, pero no se quiere eliminar el anuncio. (ver figura C.7).

Pulsando sobre cualquiera de los anuncios en cualquiera de las listas, ya sea la lista principal, la de favoritos, y la de mis anuncios se podrá acceder a la descripción completa de un anuncio concreto, lo que proporciona un texto descriptivo completo, y nos permite avanzar y retroceder entre las diferentes imágenes que tiene el anuncio. Ver figuras C.8.

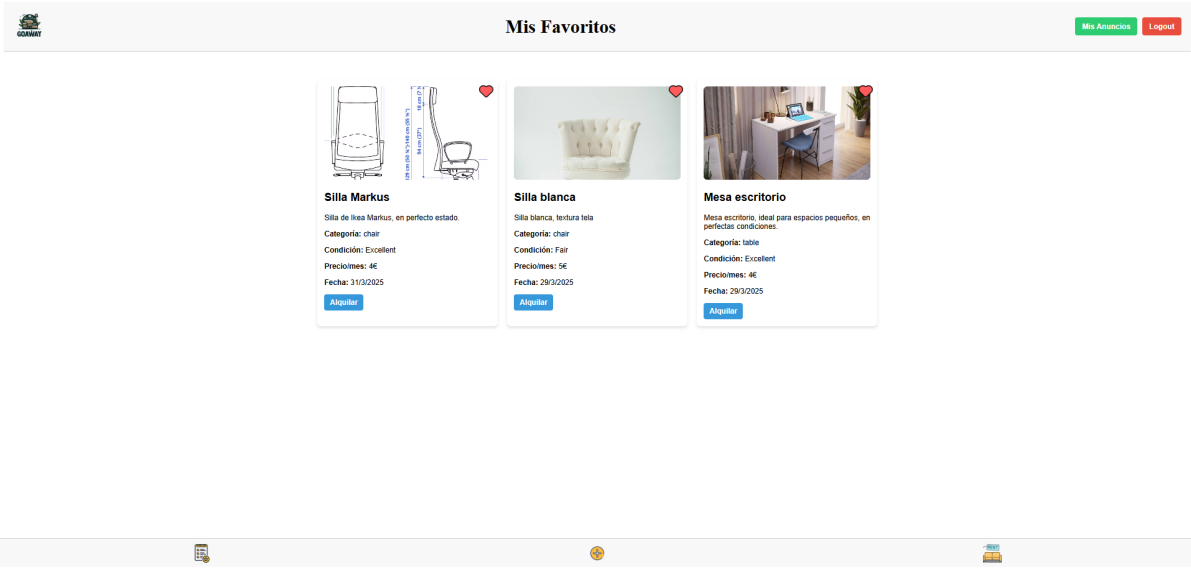


Figura C.5: Lista de anuncios favoritos

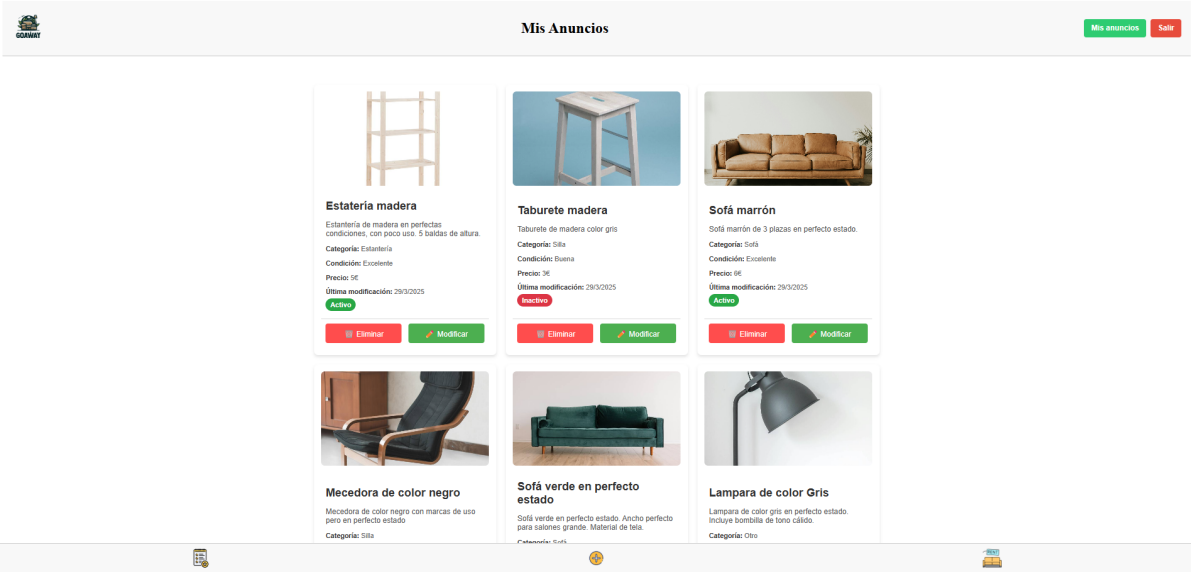


Figura C.6: Lista de mis anuncios

Figura C.7: Modificación de anuncio

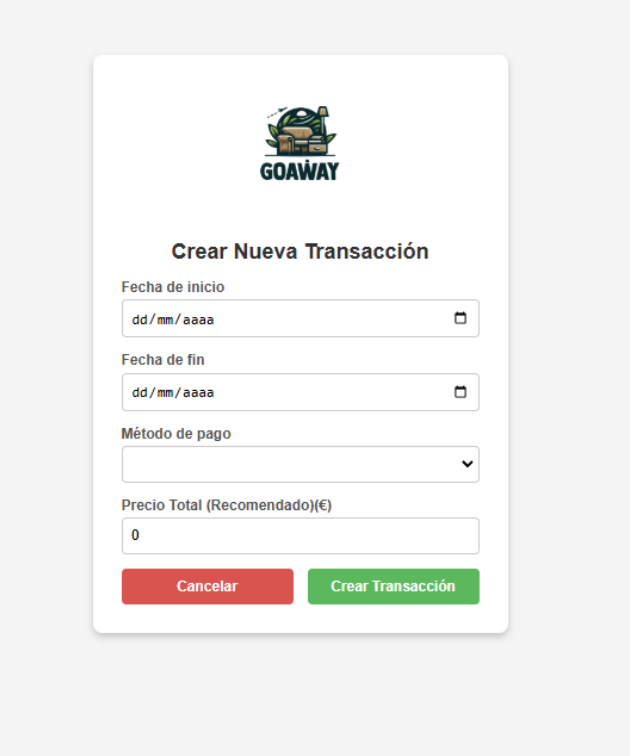
(a) Anuncio completo foto 1

(b) Anuncio completo foto 2

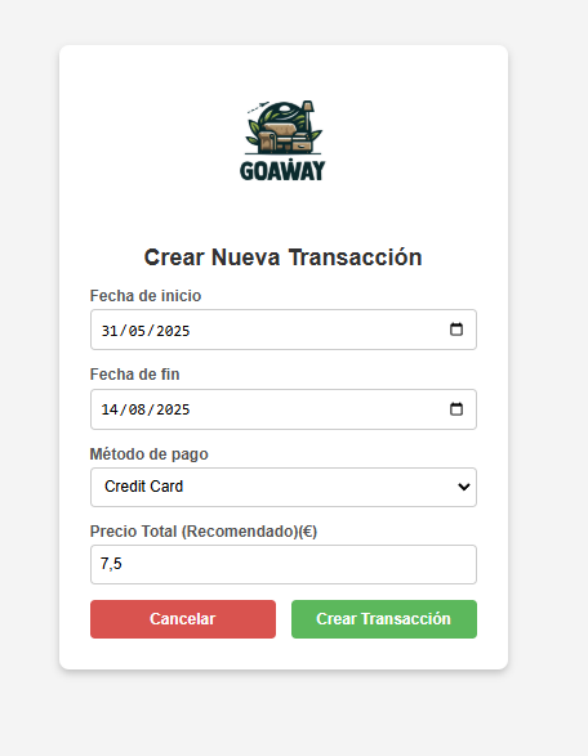
(c) Anuncio completo foto 3

Figura C.8: Anuncio concreto completo

Si se desea realizar el alquiler de uno de los anuncios, accediendo a través del botón de alquilar, podremos rellenar el formulario para la creación de una petición de alquiler. Como se observa en las figuras C.9 hay que rellenar los campos de la fecha de inicio y fin del alquiler, el método de pago y automáticamente se genera un precio (el cual se puede modificar), que nos recomienda el precio por el cual deberíamos alquilar el anuncio en función del precio/mes que tenga puesto el usuario propietario.



(a) Creación de una transacción



(b) Creación de una transacción con datos rellenos

Figura C.9: Creación de una transacción de anuncio

Una vez hemos creado una transacción de alquiler, podemos consultarlo en la figura C.10, donde en caso de que el propietario nos haya aceptado la petición de transacción podremos contactar con él vía email, como se muestra en la figura C.12. En caso en el que el propietario nos rechace la solicitud no podremos ponernos en contacto con él, se tendrá que crear una nueva solicitud de transacción.

Por otro lado, podemos ver la lista de las peticiones de transacciones (ver Figura C.11) que han hecho los usuarios sobre mis anuncios. Permitiendo aceptar o denegar una transacción. Una vez que la transacción se acepte o cancele, podremos ponernos en contacto con el solicitante.

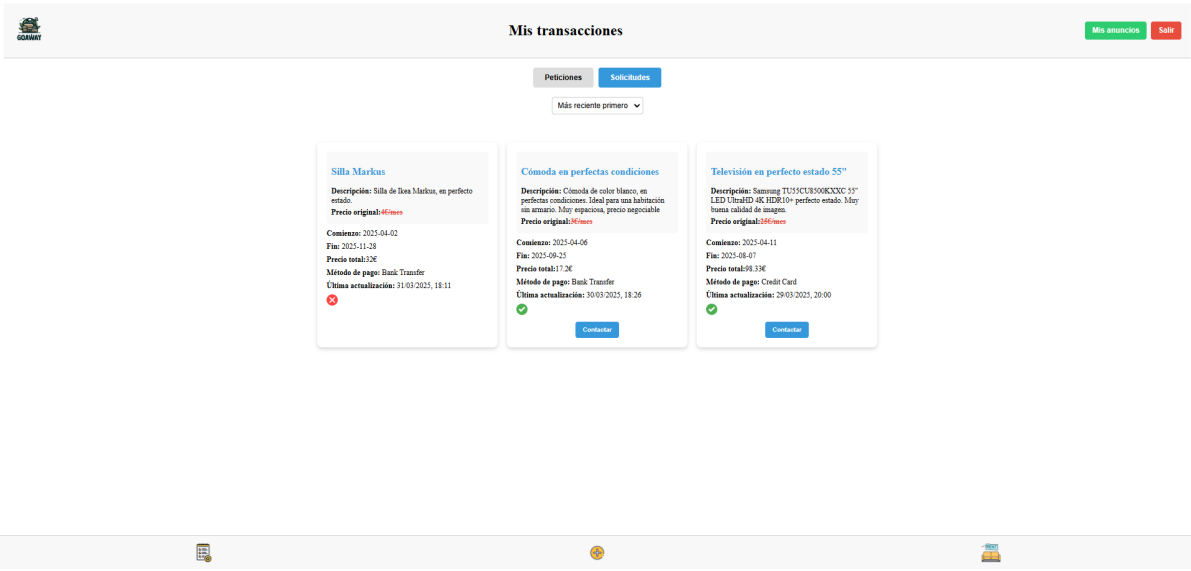


Figura C.10: Solicitudes de transacción

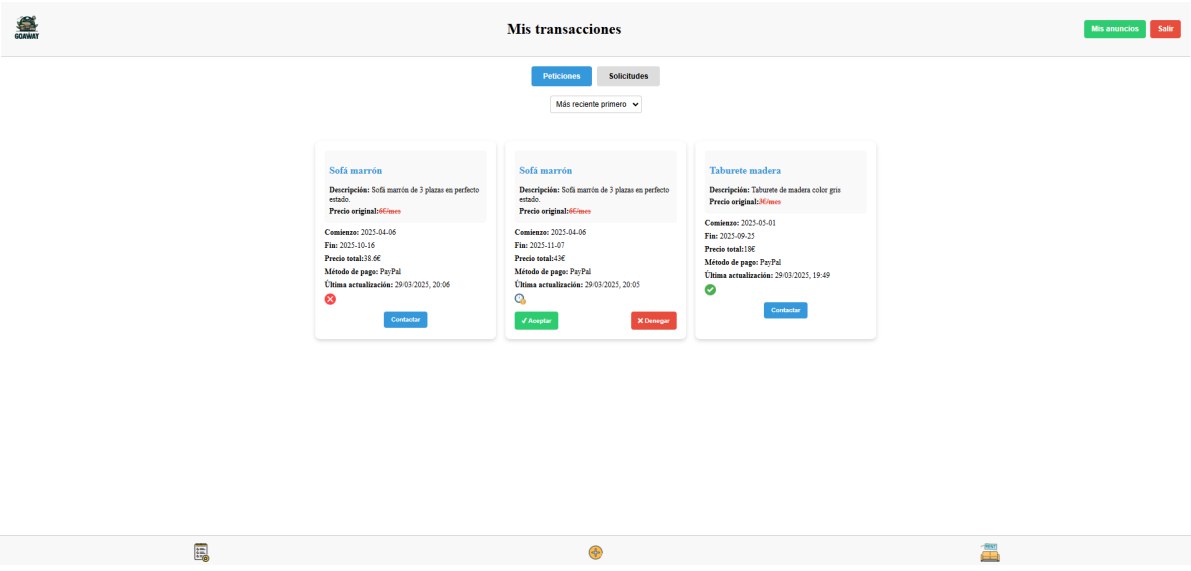


Figura C.11: Peticiones de alquiler

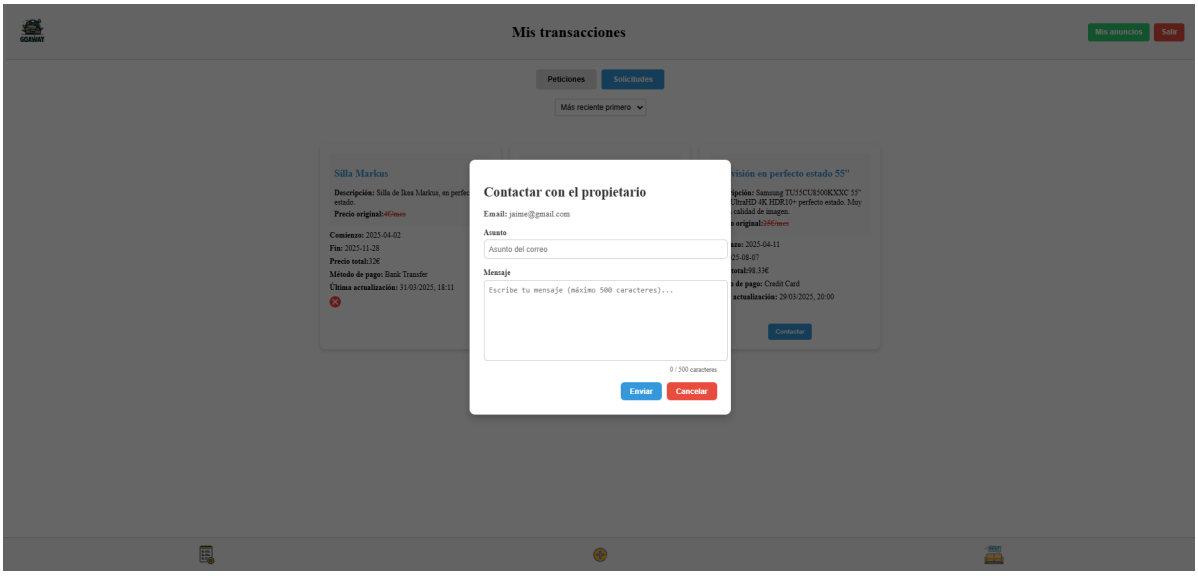


Figura C.12: Contacto por correo

Apéndice D

Contenido del TFG

El código de la aplicación desarrollado a lo largo del proyecto se encuentra en un repositorio público de GitHub. Se puede acceder mediante el siguiente enlace:

`https://github.com/jaimec01/GoAway/tree/main`

Bibliografía

[1] Eea. Visitado: 2025-03-16. [Online]. Available: https://www.eea.europa.eu/publications/textiles-in-europes-circular-economy

[2] Datos y cifras del sistema universitario español 2020-2021. Visitado: 2025-02-16. [Online]. Available: https://www.ciencia.gob.es/InfoGeneralPortal/detalle-publicacion/AGE/Datos-y-cifras-del-Sistema-Universitario-Espanol9.html

[3] Datos y cifras del sistema universitario español 2020-2021. Visitado: 2025-02-16. [Online]. Available: https://www.ciencia.gob.es/

[4] Ley de residuos y suelos contaminados. Visitado: 2025-01-16. [Online]. Available: https://www.lamoncloa.gob.es/consejodeministros/Paginas/enlaces/180521-enlace-residuos.aspx

[5] amueblarent. Visitado: 2025-01-16. [Online]. Available: https://amueblarent.es/alquiler-de-muebles/

[6] Java. Visitado: 2024-12-30. [Online]. Available: https://www.java.com/es/

[7] Springboot. Visitado: 2024-12-30. [Online]. Available: https://spring.io/projects/spring-boot

[8] Angular. Visitado: 2024-12-30. [Online]. Available: https://angular.dev/

[9] Tyscript. Visitado: 2024-12-30. [Online]. Available: https://www.typescriptlang.org/

[10] Mysql. Visitado: 2024-12-30. [Online]. Available: https://www.mysql.com/

[11] Postgresql. Visitado: 2024-12-30. [Online]. Available: https://www.postgresql.org/

[12] Docker. Visitado: 2024-12-30. [Online]. Available: https://www.docker.com/products/docker-desktop/

[13] Metodología iterativa e incremental. Visitado: 2025-01-16. [Online]. Available: https://medium.com/sue%C3%B1os-graficos/dise%C3%B1o-iterativo-la-metodolog%C3%ADa-que-perfeccionar%C3%A1-tus-proyectos-21034b0d277e

[14] Brave browser. Visitado: 2024-12-18. [Online]. Available: https://brave.com/

[15] Astah professional. Visitado: 2024-12-18. [Online]. Available: https://astah.net/products/astah-professional/

[16] Visual studio code. Visitado: 2024-12-18. [Online]. Available: https://code.visualstudio.com/

[17] Overleaf. Visitado: 2024-12-18. [Online]. Available: https://www.overleaf.com/

[18] Microsoft excel. Visitado: 2024-12-18. [Online]. Available: https://www.microsoft.com/en-us/microsoft-365/excel

[19] Microsoft teams. Visitado: 2024-12-18. [Online]. Available: https://www.microsoft.com/en-us/microsoft-teams

[20] Github. Visitado: 2024-12-18. [Online]. Available: https://github.com/

[21] Chatgpt. Visitado: 2024-12-18. [Online]. Available: https://openai.com/chatgpt

[22] Postman. Visitado: 2024-12-18. [Online]. Available: https://www.postman.com/

[23] Mysqlworkbench. Visitado: 2024-12-30. [Online]. Available: https://www.mysql.com/products/workbench/

[24] Sueldosprogramador. Visitado: 2024-12-28. [Online]. Available: https://www.hackaboss.com/blog/cuanto-gana-programador-salario-espana

[25] Costelicenciaastah. Visitado: 2024-12-28. [Online]. Available: https://astah.net/pricing/individual/

[26] Costeamortizado. Visitado: 2024-12-28. [Online]. Available: https://www.supercontable.com/informacion/Contabilidad/Metodo_de_Amortizacion_por_Trabajo_Realizado.html

[27] Platzi. Visitado: 2024-12-08. [Online]. Available: https://platzi.com/blog/arquitectura-hexagonal/

[28] Arquitectura hexagonal en spring. Visitado: 2024-12-08. [Online]. Available: https://leanmind.es/es/blog/arquitectura-hexagonal-en-spring/

[29] Balsamic. Visitado: 2024-12-30. [Online]. Available: https://balsamiq.com/

[30] Arquitectura hexagonal. Visitado: 2025-01-16. [Online]. Available: https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f

[31] Patrones de diseño. Visitado: 2025-03-14. [Online]. Available: https://refactoring.guru/es/design-patterns

[32] Patron de diseño adaptador. Visitado: 2025-03-14. [Online]. Available: https://refactoring.guru/es/design-patterns/adapter

[33] Patron de diseño builder. Visitado: 2025-03-16. [Online]. Available: https://reactiveprogramming.io/blog/es/patrones-de-diseno/builder

[34] Patron de diseño template method. Visitado: 2025-03-16. [Online]. Available: https://reactiveprogramming.io/blog/es/patrones-de-diseno/template-method

[35] Diagrama despliegue. Visitado: 2025-03-16. [Online]. Available: https://www.lucidchart.com/pages/es/tutorial-de-diagramas-de-despliegue

[36] Junit. Visitado: 2025-03-16. [Online]. Available: https://junit.org/junit5/

[37] Jasmine. Visitado: 2025-03-16. [Online]. Available: https://jasmine.github.io/

[38] Karma. Visitado: 2025-03-16. [Online]. Available: https://karma-runner.github.io/latest/index.html

[39] Mockito. Visitado: 2025-03-16. [Online]. Available: https://site.mockito.org/

[40] Jmeter. Visitado: 2025-03-16. [Online]. Available: https://jmeter.apache.org/