UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA INFORMÁTICA

Desarrollo de una aplicación Android/iOS para optimizar la gestión de viviendas y facilitar la búsqueda de convivientes.

Development of an Android/iOS application to optimize housing management and facilitate the search for cohabitants.

Realizado por
Ignacio Pascual Gutiérrez

Tutorizado por
David Santo Orcero

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE 2024

Fecha defensa: Septiembre de 2024

Resumen

En la actualidad, la dificultad para encontrar un alojamiento adecuado es un problema creciente para jóvenes y adultos en distintas regiones del mundo. El elevado costo de los alquileres ha convertido al alojamiento compartido en una opción cada vez más necesaria, más que simplemente deseada. Esta tendencia responde a la presión económica creciente y a la dificultad de mantenerse al día con los altos costos de vida en numerosas áreas urbanas.

Aparte del desafío económico, surge otro problema crucial: la necesidad de compatibilidad con los compañeros de piso. La convivencia exitosa depende en gran medida de la afinidad en términos de hábitos, intereses y valores. Encontrar personas con las que se comparta una conexión puede transformar una experiencia de vida en una experiencia positiva, en lugar de una situación estresante.

Este Trabajo de Fin de Grado aborda estas dos problemáticas: la búsqueda de una vivienda adecuada y la identificación de compañeros de piso compatibles. Más allá de las consideraciones económicas, el alojamiento compartido también proporciona beneficios sociales y emocionales, especialmente para las personas mayores, para quienes la soledad puede ser un problema creciente, y para jóvenes, quienes tienen dificultades para encajar en determinados grupos y por eso no tengan personas de confianza con las que independizarse.

El proyecto propuesto consiste en desarrollar una aplicación móvil, disponible para Android e iOS, que permita a los usuarios publicar sus propiedades y encontrar compañeros de alquiler con intereses y situaciones similares. El sistema de afinidad integrado en la aplicación facilitará la búsqueda y selección de compañeros de vivienda, promoviendo experiencias de convivencia más satisfactorias.

Palabras clave:

alojamiento compartido, compatibilidad de compañeros de piso, aplicación móvil, búsqueda de vivienda, beneficios sociales, presión económica, convivencia, afinidad entre usuarios.

Abstract

Currently, finding suitable accommodation is an increasing challenge for both young people and adults across various regions of the world. The high cost of rent has made shared housing not just a desirable option but a necessary one. This trend is driven by the growing economic pressure and the difficulty of keeping up with high living costs in expensive urban areas.

Beyond the economic challenge, another crucial issue arises: the need for compatibility with housemates. Successful cohabitation largely depends on shared habits, interests, and values. Finding individuals with whom one has a connection can transform a living situation into a positive experience rather than a stressful one.

This thesis addresses these two issues: the search for suitable housing and the identification of compatible housemates. Besides the economic considerations, shared housing also offers social and emotional benefits, particularly for older adults, who may face increasing loneliness, and for young people, who may struggle to find trusted individuals to live with as they seek independence.

The proposed project involves developing a mobile application, available for Android and iOS, that allows users to list their properties and find rental companions with similar interests and circumstances. The integrated affinity system within the application will facilitate the search and selection of housemates, promoting more satisfactory living experiences.

Key words:

shared housing, housemate compatibility, mobile application, housing search, social benefits, economic pressure, cohabitation, user affinity.

Índice

1

Introducción

1.1 Motivación

A mis 25 años, he experimentado de primera mano los retos asociados con la búsqueda de un piso compartido. A lo largo de mi experiencia personal, me he enfrentado a problemas significativos al intentar encontrar compañeros de piso que compartieran mis intereses y valores, lo que resultó en varias malas experiencias. Estas situaciones no solo hicieron que la convivencia fuera incómoda, sino que también afectaron mi bienestar emocional y calidad de vida.

La dificultad para encontrar compañeros de piso compatibles es un problema cada vez más común, especialmente entre los jóvenes que buscan equilibrar la independencia con la necesidad de compartir gastos. Las altas tarifas de alquiler y los costos de vida en áreas urbanas han llevado a un aumento en la demanda de alojamiento compartido, convirtiéndolo en una opción prácticamente necesaria en lugar de una mera preferencia.

Para comprender la magnitud de este problema y la creciente demanda de soluciones efectivas, es útil examinar datos y estadísticas recientes sobre el alojamiento compartido. A continuación, se presentan algunos datos y estadísticas clave que motivan el desarrollo de la aplicación propuesta:

Tendencia de crecimiento en el alojamiento compartido: En los últimos años, el alojamiento compartido ha visto un aumento significativo. Según un estudio publicado por el periódico **El Mundo**, más del 30% de los hogares en áreas urbanas en países desarrollados son compartidos, un incremento del 20% en la última década. Este crecimiento refleja tanto el aumento en los costos de vivienda como la búsqueda de soluciones económicas por parte de jóvenes y adultos.

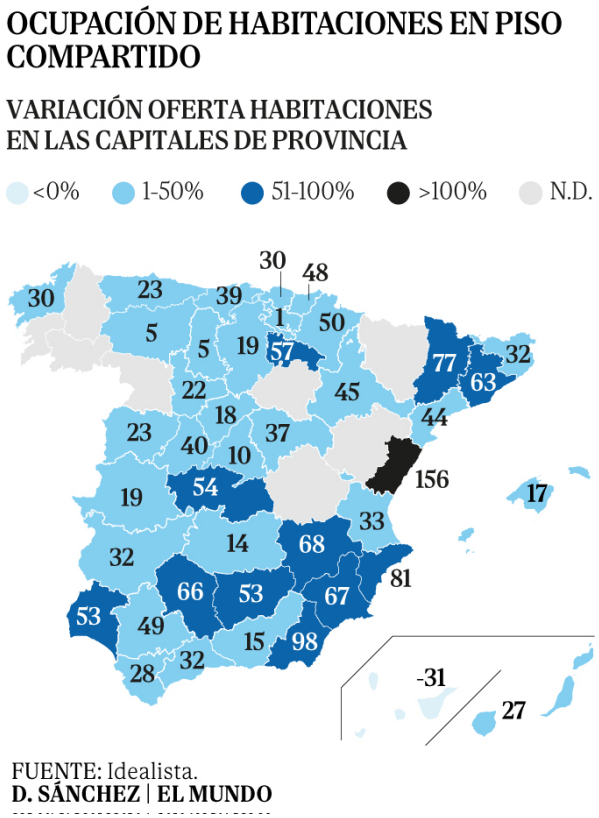


Figura 1.1. Ocupación de habitaciones en pisos compartidos (extraída de https://www.elmundo.es/economia/vivienda/2023/08/16/64dcbcb8e9cf4ac05d8b459f.html)

Además, otros portales web de alquileres como Idealista, apuntan a una continua tendencia al alta, aumentando la demanda de pisos compartidos debido principalmente a factorores como la escasa oferta de viviendas y unos precios inaccesibles (https://www.idealista.com/news/inmobiliario/vivienda/2023/11/16/809286-compartir-piso-una-tendencia-al-alza-la-demanda-crecera-hasta-un-20-en-2024)

Demografía del alojamiento compartido: La demanda de pisos compartidos está muy influenciada por la demografía. En ciudades con muchos jóvenes, como en las zonas universitarias, y en lugares con profesionales de corta edad, la gente busca compartir pisos para ahorrar dinero y tener más flexibilidad. También, cuando hay muchas personas que viven solas, la demanda de compartir pisos aumenta porque buscan opciones más económicas y sociables.

Por otro lado, la migración y el crecimiento urbano también juegan su parte. En ciudades con muchos inmigrantes, estos a menudo eligen compartir pisos para adaptarse mejor al nuevo entorno económico. Y en áreas que están creciendo

rápido, la gente recurre más a los pisos compartidos porque los precios de la vivienda son altos.

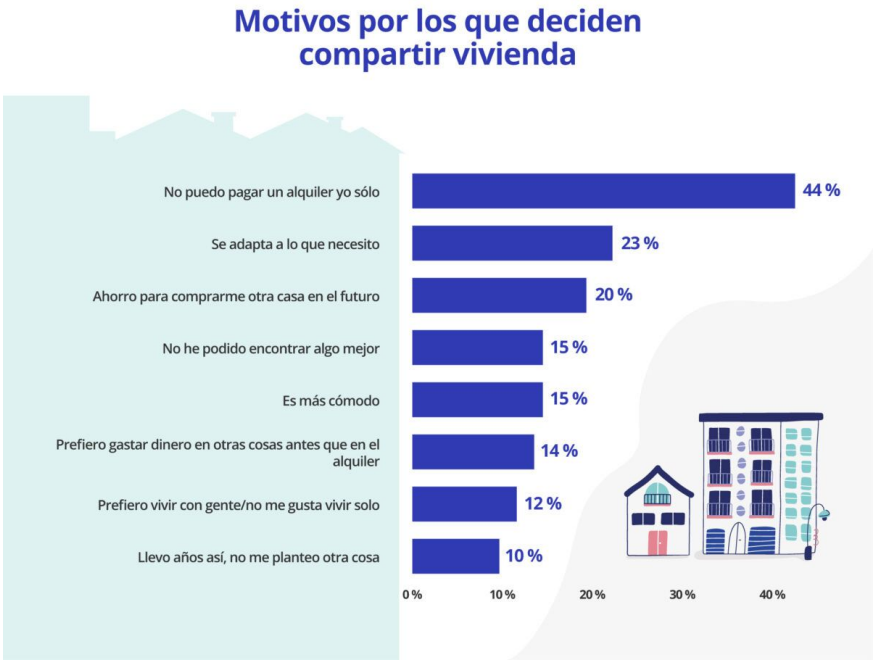


Figura 1.2. Motivos por los que las personas en España deciden compartir vivienda (extraída de https://www.fotocasa.es/fotocasa-life/compartir-piso/4-de-cada-10-de-los-espanoles-que-comparte-piso-lo-hace-porque-no-puede-pagar-un-alquiler-entero/)

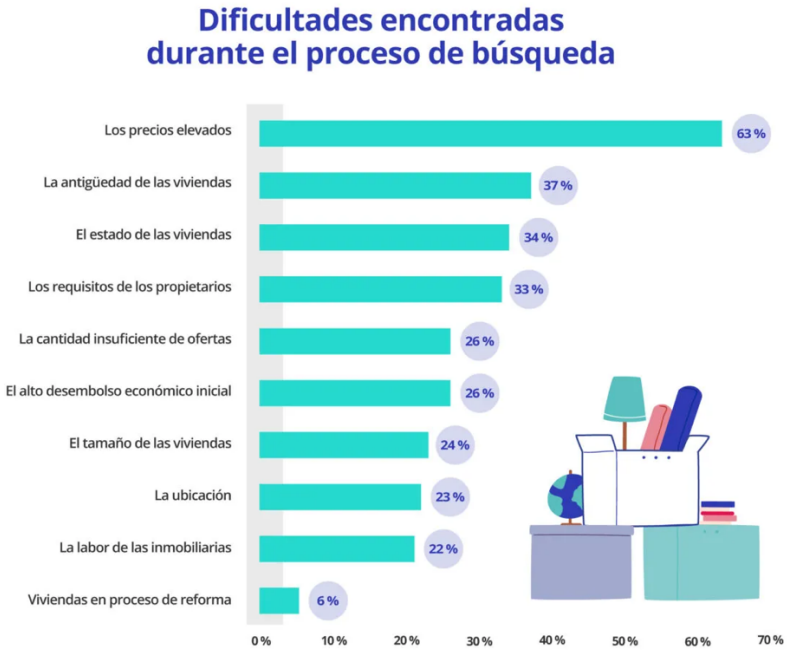


Figura 1.3. Dificultades encontradas durante el proceso de búsqueda (extraída de https://www.fotocasa.es/fotocasa-life/compartir-piso/4-de-cada-10-de-los-espanoles-que-comparte-piso-lo-hace-porque-no-puede-pagar-un-alquiler-entero/)

Impacto en la convivencia: Como el principal motivo por el que las personas buscan pisos compartidos es por un tema méramente económico (en la mayoría de casos), convivir con otras personas con las que no congenies puede producir un estado de impotencia y frustración durante toda tu estancia. Esto subraya la importancia de encontrar compañeros de piso que no solo compartan un espacio físico, sino también intereses y valores similares para asegurar una convivencia armoniosa.

Estos datos y mi propia experiencia personal, aflora la necesidad de una solución que facilite la conexión entre personas con intereses y hábitos compatibles, ayudando a mitigar los problemas derivados de la falta de afinidad en las convivencias. Por esta razón, el desarrollo de una aplicación móvil destinada a mejorar la búsqueda de compañeros de piso y la gestión de espacios compartidos se convierte en una solución ideal a los desafíos actuales del mercado inmobiliario.

1.2 Objetivos

El objetivo principal de este trabajo ha sido la elaboración de una aplicación móvil que sirva como base para subsanar los problemas definidos anteriormente, principalmente tanto la búsqueda de pisos compartidos como la de personas afines a tu estilo de vida y personalidad. Usando este concepto general, podemos granular diferentes metas a conseguir:

Diseño y desarrollo de la aplicación: Diseñar y desarrollar una plataforma intuitiva y funcional que permita a los usuarios publicar propiedades disponibles para alquiler compartido y encontrar compañeros de piso adecuados. Esto incluye la implementación de características clave como perfiles detallados, filtros de afinidad y un sistema de compatibilidad que ayude a los usuarios a identificar posibles compañeros con intereses y valores similares. El diseño debe asegurar una experiencia de usuario fluida y atractiva, optimizando la navegación y la interacción dentro de la aplicación.

Creación de un sistema de afinidad efectivo Un componente esencial del proyecto ha sido el desarrollo de un sistema de afinidad que permita a los usuarios conectar con compañeros de piso que compartan intereses, hábitos de vida y valores similares. Este sistema debe ser capaz de procesar y analizar la

información proporcionada por los usuarios para generar coincidencias precisas. La implementación de algoritmos de compatibilidad y la integración de encuestas o cuestionarios serán necesarios para garantizar que las recomendaciones sean relevantes y satisfactorias.

Validación de funcionalidad y usabilidad: El tercer objetivo será validar la funcionalidad y usabilidad de la aplicación desarrollada. Esto implica realizar pruebas exhaustivas para garantizar que todas las características y funcionalidades operen correctamente y cumplan con las expectativas de los usuarios. Se llevará a cabo una serie de pruebas piloto con usuarios reales para recoger feedback sobre la experiencia de uso, la eficacia del sistema de afinidad y la satisfacción general con la plataforma.

1.3 Estructura de la memoria

El proyecto ha sido dividido en 5 capítulos. A continuación, se resumen de manera breve cada uno de los capítulos que componen el trabajo realizado:

- **Capítulo 1. Introducción.**

Donde explicamos la motivación del proyecto, los objetivos establecidos y la estructura que compone el propio desarrollo del proyecto.

- **Capítulo 2. Tecnologías.**

Definición de conceptos básicos y de las tecnologías empleadas en el prototipo.

- **Capítulo 3. Análisis y diseño.**

Análisis y especificación de requisitos del prototipo, además de su diseño y arquitectura, añadiendo diagramas que reflejen la lógica elegida.

- **Capítulo 4. Desarrollo e implementación.**

Desarrollo del prototipo definido en la fase previa. En este caso, desde el sistema de afinidad propuesto hasta la elaboración de la interfaz del usuario de la aplicación móvil.

- **Capítulo 5. Conclusiones y líneas futuras.**

Apartado que incluye a modo de resumen las conclusiones extraídas del trabajo realizado así como la posibilidad de continuar con la línea de investigación y posibles mejoras e implementaciones del prototipo.

2

Tecnologías

En este segundo capítulo se describirán las tecnologías empleadas en el proyecto presentado en esta memoria. Se explicarán las principales características de cada tecnología, las razones de su selección y cómo contribuyen al logro de los objetivos del Trabajo de Fin de Grado.

2.1 Backend

2.1.1 Python

Figura 2.1 Logo de Python (extraída de https://www.python.org/community/logos/)

Descripción: Python es un lenguaje de programación de alto nivel, ampliamente utilizado en el desarrollo de aplicaciones web, ciencia de datos, y automatización, entre otros.

Razón de Uso: Python ha sido elegido para el desarrollo del backend de la aplicación debido a su simplicidad, legibilidad y amplia comunidad de soporte. Su sintaxis clara facilita el desarrollo rápido y eficiente de código, y su extensa biblioteca de módulos y frameworks proporciona herramientas robustas para construir y gestionar APIs y servicios backend.

Principales librerías utilizadas:

FastAPI

Se ha utilizado FastAPI para desarrollar la API del **backend** de la aplicación. Su diseño asíncrono y su capacidad para generar documentación automática y validaciones de entrada hacen que sea ideal para construir servicios web rápidos y eficientes. FastAPI facilita la creación de **endpoints** de manera rápida y con un rendimiento optimizado, lo que contribuye a una experiencia de usuario fluida.

Documentación oficial: https://fastapi.tiangolo.com

Uvicorn

Uvicorn se emplea para ejecutar la API desarrollada con FastAPI. Uvicorn es compatible con varios frameworks de Python basados en ASGI (Asynchronous Server Gateway Interface), lo que facilita su uso en proyectos que utilizan estos frameworks para construir aplicaciones web.

Documentación oficial: https://www.uvicorn.org

Poetry

Poetry se utiliza para gestionar las dependencias del proyecto y para el empaquetado de la aplicación. Permite definir y controlar las versiones de las bibliotecas utilizadas, así como crear un entorno aislado para el desarrollo. Esto asegura que las dependencias estén bien gestionadas y que el proyecto sea reproducible en diferentes entornos.

Documentación oficial: https://python-poetry.org/docs/

Psycopg2-binary

Se utiliza para conectar la API con la base de datos PostgreSQL. Esta librería facilita la ejecución de consultas SQL y la manipulación de datos almacenados en la base de datos.

Documentación oficial: https://www.psycopg.org/docs/install.html

2.1.2 PostgreSQL

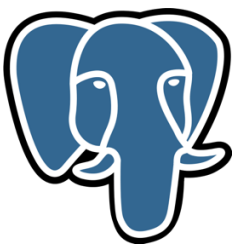


Figura 2.2 Logo de PostgreSQL (extraído de https://www.postgresql.org/docs/)

PostgreSQL es un sistema de gestión de bases de datos relacional (RDBMS) de código abierto, conocido por su robustez, flexibilidad y cumplimiento de estándares SQL. Trabajaremos con una base de datos relacional ya que tenemos un set de datos estructurados y bien definidos, siendo capaces de crear un esquema de base de datos que relacione correctamente nuestra entidades.

2.1.3 TablePlus

Figura 2.3 Logo de TablePlus (extraído de https://tableplus.com/)

TablePlus es una herramienta de administración de bases de datos que proporciona una interfaz gráfica para gestionar y consultar bases de datos SQL y NoSQL. Se ha elegido para facilitar la gestión y visualización de la base de datos PostgreSQL.

2.1.4 Postman

Figura 2.4 Logo de Postman (extraído de https://www.postman.com/)

Postman es una herramienta para el desarrollo y prueba de APIs que facilita la creación, envío y análisis de solicitudes HTTP. Se ha elegido para probar y depurar la API desarrollada en el backend de la aplicación.

2.2 Frontend

2.2.1 Javascript

Figura 2.5 Logo de Javascript (extraído de https://es.wikipedia.org/wiki/Archivo:JavaScript-logo.png)

JavaScript es un lenguaje de programación versátil y ampliamente utilizado en el desarrollo web y móvil. En el contexto de React Native, JavaScript se utiliza para escribir la lógica y los componentes de la aplicación.

2.2.2 React Native

Figura 2.6 Logo de React Native (extraído de https://reactnative.dev)

React Native es un framework de desarrollo de aplicaciones móviles creado por Facebook. Permite desarrollar aplicaciones para Android e iOS utilizando JavaScript y el mismo código base. React Native se basa en React, la biblioteca de JavaScript para la construcción de interfaces de usuario, y utiliza componentes nativos, lo que significa que las aplicaciones tienen un rendimiento cercano al nativo y una experiencia de usuario fluida.

Documentación oficial: https://reactnative.dev

2.3 Backend & Frontend

2.3.1 Visual Studio Code

Figura 2.7 Logo de VSCode (extraído de https://code.visualstudio.com/brand)

Visual Studio Code es un editor de código fuente multiplataforma desarrollado por Microsoft. Ofrece diversas características, como depuración de código, integración con Git, refactorización y autocompletado inteligente.

Aunque no se requiere un entorno específico para desarrollar clientes y servidores de aplicaciones Node.js, Visual Studio Code facilita el trabajo al proporcionar herramientas que optimizan la eficiencia y la simplicidad. Por ejemplo, su terminal integrado simplifica la ejecución del servidor, lo que agiliza el proceso de desarrollo.

3

Análisis y diseño

En este capítulo se detallan los objetivos y el alcance inicial del prototipo, junto con los requisitos que debe cumplir. Los requisitos se clasificarán en dos categorías principales: **funcionales** y **no funcionales**. Los requisitos funcionales especifican los servicios y funcionalidades que el sistema debe proporcionar, mientras que los no funcionales se centran en aspectos relacionados con el diseño y la implementación del sistema. Además, se describirán los **actores** involucrados en el uso y desarrollo del prototipo.

3.1 Actores

Los actores son los usuarios o sistemas que interactuarán con la aplicación y que influirán en su funcionamiento. Para esta aplicación, los actores identificados son:

Usuarios Registrados

- **Descripción:** Personas que crean una cuenta en la aplicación para buscar o ofrecer alojamiento compartido.
- **Responsabilidades:** Publicar propiedades, buscar compañeros de piso, gestionar su perfil, y comunicarse con otros usuarios.

Propietarios de Viviendas

- **Descripción:** Usuarios que poseen propiedades y desean alquilarlas en régimen de compartición.
- **Responsabilidades:** Publicar anuncios de sus propiedades, proporcionar detalles y fotos, y gestionar solicitudes de potenciales compañeros de piso.

Compañeros de Piso Potenciales

- **Descripción:** Usuarios en busca de un lugar para vivir en compañía.
- **Responsabilidades:** Buscar propiedades, evaluar compatibilidad con posibles compañeros de piso, y comunicarse con propietarios o actuales inquilinos.

3.2 Requisitos funcionales

Para esta aplicación, los requisitos funcionales clave incluyen:

R.F. 1 Registro de usuarios: La aplicación debe permitir a los usuarios crear una cuenta.

R.F. 2 Autenticación de usuarios: La aplicación debe permitir a los usuarios con una cuenta autenticar su identidad mediante un identificador, como el correo electrónico, y usando contraseñas.

R.F. 3 Publicación de propiedades: Los usuarios deben poder publicar anuncios de propiedades disponibles para alquiler compartido, incluyendo detalles como ubicación, precio, descripción y fotos.

R.F. 4 Creación de perfiles de usuarios: Los usuarios deben poder crear y gestionar sus perfiles, incluyendo información personal, intereses, hábitos de vida y preferencias para la convivencia.

R.F. 5 Creación de preferencias de usuario: Los usuarios deben poder crear y gestionar sus preferencias en base a qué buscan de sus futuros compañeros de piso.

R.F. 6 Sistema de afinidad: La aplicación debe ofrecer un sistema de afinidad que permita a los usuarios buscar y conectar con posibles compañeros de piso que compartan intereses y valores similares.

R.F. 7 Búsqueda y filtros: Los usuarios deben tener la capacidad de buscar propiedades y compañeros de piso utilizando filtros basados en ubicación, precio, tipo de propiedad y otras características relevantes.

3.3 Requisitos no funcionales

Para esta aplicación, se han identificado los siguientes requisitos no funcionales:

R.N.F. 1 Usabilidad: La aplicación debe ser intuitiva y fácil de usar, con una interfaz de usuario clara y accesible. Se debe garantizar una experiencia de usuario fluida tanto en dispositivos Android como en iOS.

R.N.F. 2 Rendimiento: La aplicación debe funcionar de manera eficiente, con tiempos de respuesta rápidos y sin problemas de rendimiento, incluso con una base de usuarios creciente.

R.N.F. 3 Seguridad: La aplicación debe implementar medidas de protección adecuadas, como cifrado de datos, autenticación segura y protección contra accesos no autorizados.

R.N.F. 4 Compatibilidad: La aplicación debe ser compatible con las versiones actuales de los sistemas operativos Android e iOS, y debe adaptarse a diferentes tamaños y resoluciones de pantalla.

Este capítulo describe el prototipo creado, su diseño y arquitectura así como una serie de diagramas que permiten comprender la implementación realizada.

3.4 Estructura de la base de datos

Vamos a dividir la estructura en tres partes claramente diferenciadas: **Usuarios**, **Apartamentos** y **Peticiones**.

3.4.1 Usuarios

La **Figura 3.1.1** representa la estructura de una base de datos orientada a gestionar información de usuarios, sus preferencias y afinidades.

Cada **usuario** está vinculado a un **rol** a través de una clave foránea, lo que permite definir y controlar el acceso y las funciones que cada usuario puede desempeñar dentro del sistema.

La información adicional de los usuarios se guarda en la tabla **USER_EXTRA_INFO**, proporcionando un perfil detallado que incluye datos personales, estilos de vida y preferencias. Estas preferencias están bien definidas y almacenadas en **USER_PREFERENCES**, permitiendo a la plataforma gestionar las afinidades con otros usuarios.

La relación entre usuarios se gestiona mediante **USER_AFFINITY**, que registra las afinidades utilizando una puntuación específica, lo que facilita la identificación de conexiones significativas entre usuarios y mejora la recomendación y la búsqueda de coincidencias.

Además, los atributos relacionados con estilos de vida están organizados en tablas específicas como **PHYSICAL_LIFESTYLES**, **ECONOMY_LIFESTYLES**,

RESIDENCE_LIFESTYLES, y **SOCIAL_LIFESTYLES**, lo que permite una categorización precisa y flexible de los diversos aspectos del estilo de vida de los usuarios.

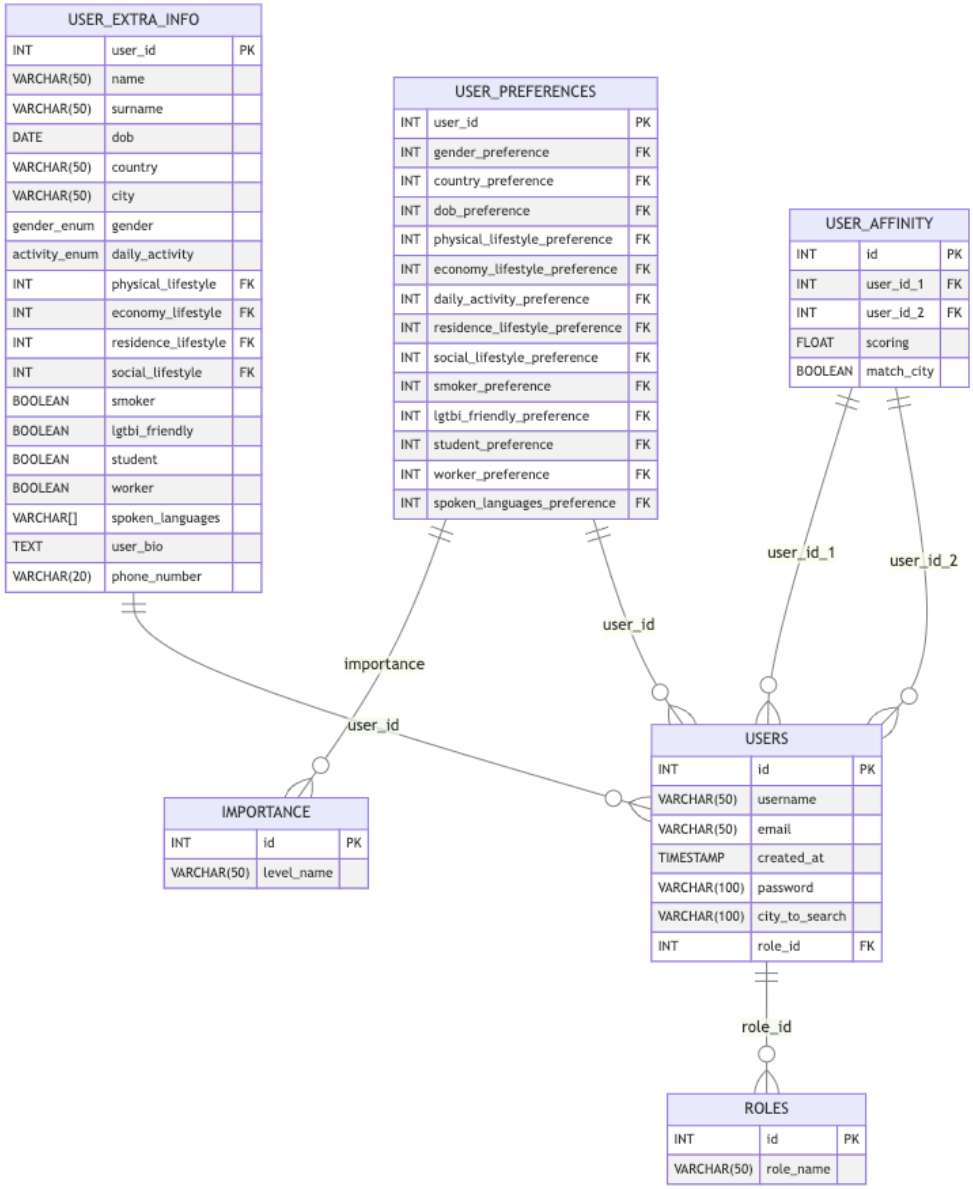


Figura 3.1.1 Modelo de datos relacionado con la entidad Usuarios

Entidades

1. **ROLES:** Información sobre los roles disponibles para los usuarios

- id: Identificador único del rol (clave primaria).
- role_name: Nombre del rol.

2. **USERS:** Datos básicos de los usuarios, como nombre de usuario, correo electrónico, contraseña y rol.

- id: Identificador único del usuario (clave primaria).
- username: Nombre de usuario (único).
- email: Correo electrónico del usuario (único).
- created_at: Fecha y hora de creación de la cuenta del usuario.
- password: Contraseña **encriptada** del usuario.
- role_id: Identificador del rol del usuario (clave foránea referenciando a ROLES).

3. **USER_EXTRA_INFO:** Información adicional del usuario, como nombre completo, fecha de nacimiento, ubicación, y características personales.

- user_id: Identificador único del usuario (clave primaria y clave foránea referenciando a USERS).
- name: Nombre del usuario.
- surname: Apellido del usuario.
- dob: Fecha de nacimiento.
- country: País de residencia.
- city: Ciudad de residencia.
- gender: Género del usuario.
- daily_activity: Actividad diaria del usuario.
- physical_lifestyle: Estilo de vida físico del usuario (clave foránea referenciando a PHYSICAL_LIFESTYLES).
- economy_lifestyle: Estilo de vida económico del usuario (clave foránea referenciando a ECONOMY_LIFESTYLES).
- residence_lifestyle: Estilo de vida de residencia del usuario (clave foránea referenciando a RESIDENCE_LIFESTYLES).
- social_lifestyle: Estilo de vida social del usuario (clave foránea referenciando a SOCIAL_LIFESTYLES).
- smoker: Indicador de si el usuario es fumador.
- lgtbi_friendly: Indicador de si el usuario es amigable con la comunidad LGTBI.
- student: Indicador de si el usuario es estudiante.
- worker: Indicador de si el usuario es trabajador.
- spoken_languages: Idiomas hablados por el usuario (almacenados como un array de strings).
- user_bio: Biografía del usuario.
- phone_number: Número de teléfono del usuario.

4. **USER_PREFERENCES:** Preferencias del usuario usando diferentes métricas, vinculadas a niveles de importancia.

- user_id: Identificador del usuario (clave primaria y clave foránea referenciando a USERS).
- gender_preference: Preferencia de género (clave foránea referenciando a IMPORTANCE).

- country_preference: Preferencia de país (clave foránea referenciando a IMPORTANCE).
- dob_preference: Preferencia de fecha de nacimiento (clave foránea referenciando a IMPORTANCE).
- physical_lifestyle_preference: Preferencia de estilo de vida físico (clave foránea referenciando a IMPORTANCE).
- economy_lifestyle_preference: Preferencia de estilo de vida económico (clave foránea referenciando a IMPORTANCE).
- daily_activity_preference: Preferencia de actividad diaria (clave foránea referenciando a IMPORTANCE).
- residence_lifestyle_preference: Preferencia de estilo de vida de residencia (clave foránea referenciando a IMPORTANCE).
- social_lifestyle_preference: Preferencia de estilo de vida social (clave foránea referenciando a IMPORTANCE).
- smoker_preference: Preferencia sobre si el usuario es fumador (clave foránea referenciando a IMPORTANCE).
- lgtbi_friendly_preference: Preferencia sobre si el usuario es amigable con la comunidad LGTBI (clave foránea referenciando a IMPORTANCE).
- student_preference: Preferencia sobre si el usuario es estudiante (clave foránea referenciando a IMPORTANCE).
- worker_preference: Preferencia sobre si el usuario es trabajador (clave foránea referenciando a IMPORTANCE).
- spoken_languages_preference: Preferencia sobre los idiomas hablados (clave foránea referenciando a IMPORTANCE).

5. **IMPORTANCE:** Niveles de importancia para preferencias del usuario.

- id: Identificador único del nivel de importancia (clave primaria).
- level_name: Nombre del nivel de importancia.

6. **PHYSICAL_LIFESTYLES:** Estilos de vida físicos con sus categorías.

- id: Identificador único del estilo de vida físico (clave primaria).
- physical_lifestyle_enum: Enum que define el estilo de vida físico.
- category: Categoría del estilo de vida físico.

7. **ECONOMY_LIFESTYLES:** Estilos de vida económicos disponibles con sus categorías.

- id: Identificador único del estilo de vida económico (clave primaria).
- economy_lifestyle_enum: Enum que define el estilo de vida económico.
- category: Categoría del estilo de vida económico.

8. **RESIDENCE_LIFESTYLES:** Estilos de vida residenciales disponibles con sus categorías.

- id: Identificador único del estilo de vida de residencia (clave primaria).
- residence_lifestyles_enum: Enum que define el estilo de vida de residencia.
- category: Categoría del estilo de vida de residencia.

9. **SOCIAL_LIFESTYLES:** Estilos de vida social disponibles con sus categorías.

- id: Identificador único del estilo de vida social (clave primaria).
- social_lifestyles_enum: Enum que define el estilo de vida social.
- category: Categoría del estilo de vida social.

10. **USER_AFFINITY:** Afinidades entre pares de usuarios, con puntuaciones de similitud y coincidencia de ciudad.

- id: Identificador único de la afinidad entre usuarios (clave primaria).
- user_id_1: Identificador del primer usuario (clave foránea referenciando a USERS).
- user_id_2: Identificador del segundo usuario (clave foránea referenciando a USERS).
- jaccard_scoring: Puntuación basada en el índice de Jaccard entre los usuarios.
- match_city: Indicador de si los usuarios están en la misma ciudad.

Relaciones

- **USER_PREFERENCES** tiene una relación uno a uno con **USERS**, ya que cada usuario tiene un conjunto específico de preferencias.
- **USER_EXTRA_INFO** también tiene una relación uno a uno con **USERS**, almacenando información adicional sobre el usuario.
- **USER_AFFINITY** representa las relaciones entre pares de usuarios, con puntajes de afinidad y un indicador de coincidencia de ciudad.

Diagramas de Relaciones

- **USER_PREFERENCES** se relaciona con **IMPORTANCE** a través de las claves foráneas de preferencias.
- **USERS** se relaciona con **ROLES** a través de role_id.
- **USER_EXTRA_INFO** y **USER_PREFERENCES** se relacionan con **USERS** a través de user_id.

- **USER_AFFINITY** se relaciona con **USERS** a través de user_id_1 y user_id_2.

3.4.2 Apartamentos

La **Figura 3.2** representa la estructura de la base de datos relacionada con las viviendas, además de su relación las entidades de los usuarios.

La entitdad **APARTMENTS** está diseñada para almacenar información completa sobre cada apartamento, incluyendo detalles sobre la ubicación, características y condiciones de alquiler, y está vinculada a los usuarios que gestionan o poseen estos apartamentos.

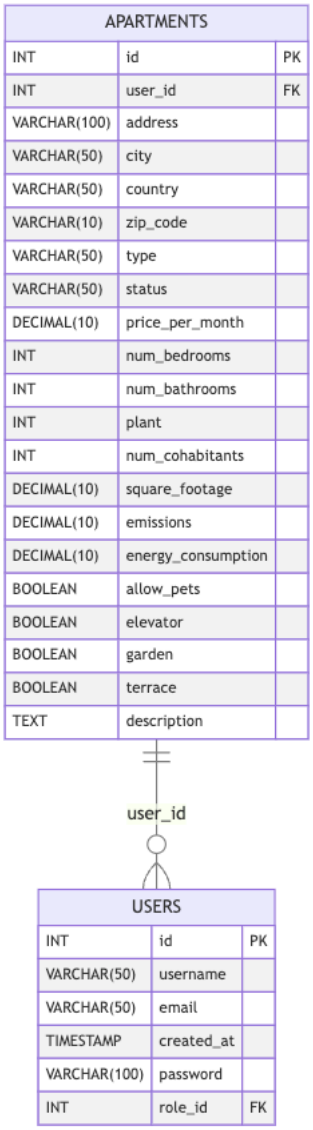


Figura 3.2 Modelo de datos relacionado con la entidad Apartamentos

1. APARTMENTS: Esta entidad proporciona un registro detallado de los apartamentos disponibles o gestionados, cubriendo desde detalles básicos hasta características específicas que pueden ser relevantes para los usuarios que buscan alquilar o comprar.

- id: Identificador único del apartamento (clave primaria).
- user_id: Identificador del usuario que posee o gestiona el apartamento (clave foránea referenciando a USERS).
- address: Dirección completa del apartamento.
- city: Ciudad donde se encuentra el apartamento.
- country: País donde se encuentra el apartamento.
- zip_code: Código postal del apartamento.
- type: Tipo de apartamento (por ejemplo, "apartamento", "piso", "estudio").
- status: Estado del apartamento (por ejemplo, "disponible", "ocupado", "en renovación").
- price_per_month: Precio mensual del alquiler del apartamento.
- num_bedrooms: Número de habitaciones del apartamento.
- num_bathrooms: Número de baños del apartamento.
- plant: Planta en la que se encuentra el apartamento
- max_cohabitants: Número máximo de personas que pueden habitar el apartamento.
- square_footage: Superficie del apartamento en metros cuadrados.
- emissions: Emisiones del apartamento en relación con su impacto ambiental.
- energy_consumption: Consumo energético del apartamento.
- allow_pets: Indica si se permiten mascotas en el apartamento.
- elevator: Indica si el edificio tiene ascensor.
- garden: Indica si el apartamento tiene jardín.
- terrace: Indica si el apartamento tiene terraza.
- description: Descripción del apartamento publicado.

Relaciones

- **USERS** tiene una relación uno a muchos con **APARTMENTS**, ya que cada usuario puede tener uno, ninguno o muchos apartamentos.

Diagramas de Relaciones

- **APARTMENTS** se relaciona con **USERS** a través de la clave user_id.

3.4.3 Peticiones

La **Figura 3.3** representa la estructura de la base de datos relacionada con las peticiones, además de su relación las entidades de los usuarios y los apartamentos.

La entidad **REQUESTS** está diseñada para almacenar información sobre cada petición, incluyendo el remitente de la petición, el apartamento sobre el que se basa la petición y el conjunto de usuarios implicados, entre otros campos.

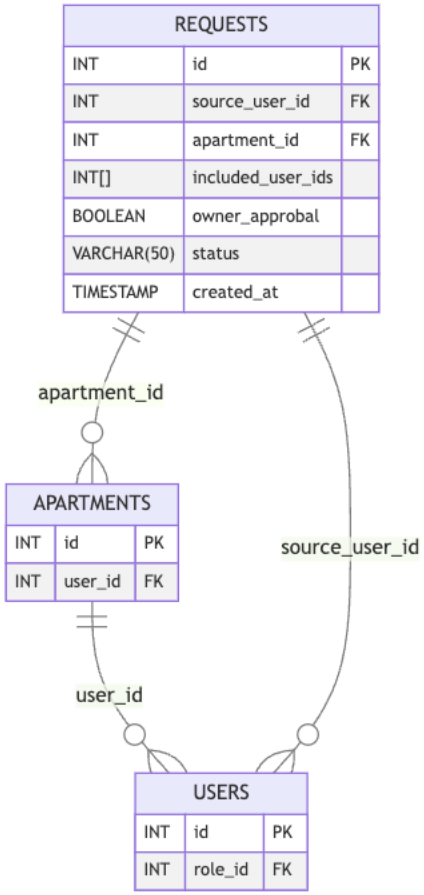


Figura 3.3 Modelo de datos relacionado con la entidad Peticiones

1. REQUESTS: Esta entidad proporciona un registro detallado de las peticiones emitidas por cada usuario, cubriendo detalles básicos para registrar y procesar cada petición.

- id: Identificador único de la petición (clave primaria).
- source_user_id: Identificador del usuario remitente de la petición (clave foránea referenciando a USERS).
- apartment_id: Identificador del aparte sobre el que se procesa la petición (clave foránea referenciando a APARTMENTS).
- included_user_ids: Identificadores de los usuarios involucrados en la petición.
- owner_approbal: Indica si la petición ha sido aprobada por el propietario del apartamento.
- status: Estado de la petición (se contemplan los estados "PENDIENTE" o "APROBADO")
- created_at: Fecha de creación de la petición.

Relaciones

- **REQUESTS** tiene una relación muchos a uno con **USERS**, ya que un usuario puede estar asociado con múltiples solicitudes, pero cada solicitud solo puede estar asociada al usuario que la creó.
- **REQUESTS** tiene una relación muchos a uno con **APARTMENTS**, ya que un apartamento puede estar asociado con múltiples solicitudes, pero cada solicitud solo puede estar asociada a un único apartamento.

Diagramas de Relaciones

- **REQUESTS** se relaciona con **USERS** a través de la clave user_id.
- **REQUESTS** se relaciona con **APARTMENTS** a través de la clave apartment_id.

3.4 Estructura del backend

Base de datos

La arquitectura del backend de la aplicación está diseñada para ofrecer un sistema robusto y eficiente para la gestión de datos y la interacción con el frontend. La base de datos como bien se ha comentado anteriorment es PostgreSQL.

Para conectar la base de datos PostgreSQL con la API del backend, se utiliza la librería psycopg2, que es una librería de Python eficiente y estable para la comunicación con PostgreSQL. Esta librería facilita la ejecución de consultas SQL, la gestión de transacciones y el manejo de resultados de manera eficaz. Es ampliamente utilizada en la comunidad Python por su rendimiento y su capacidad para manejar conexiones de manera robusta.

Gestor de dependencias y entornos virtuales

La gestión de dependencias en el proyecto se realiza mediante **Poetry**, un gestor de dependencias y herramienta de empaquetado que simplifica la administración de versiones y dependencias en proyectos Python. Poetry permite definir y manejar las dependencias del proyecto de forma clara y estructurada en el archivo pyproject.toml, lo que evita conflictos de versiones y asegura la coherencia entre entornos de desarrollo y producción. Además, Poetry automatiza la creación de entornos virtuales, lo que mejora el aislamiento de las dependencias y facilita la gestión del proyecto.

Estándar de código Python: PEP8

Para mantener la calidad del código y asegurar su consistencia, se emplea **flake8** como herramienta de análisis estático. Flake8 combina el chequeo del estilo de código (según PEP 8), la detección de errores y la verificación de calidad del código. Su uso permite mantener un código limpio y bien estructurado. Al integrar flake8 en el proceso de desarrollo, se asegura que el código cumple con los estándares establecidos y se reduce el riesgo de errores y problemas de estilo.

Estructura de código

En cuanto a la estructura del código, se adopta el **patrón por repositorio**, que organiza el acceso a los datos mediante repositorios especializados. Este patrón permite separar claramente la lógica de negocio del acceso a datos, promoviendo una arquitectura modular y escalable. Al centralizar la lógica de acceso a datos en los repositorios, se facilita la prueba y el mantenimiento del código, y se promueve la reutilización y coherencia en las operaciones de datos. Este enfoque no solo mejora la organización del código, sino que también facilita su evolución y adaptabilidad a cambios futuros.

En este proyecto, este patrón se aplica tanto a la gestión de usuarios y sus preferencias como a la administración de apartamentos y peticiones.

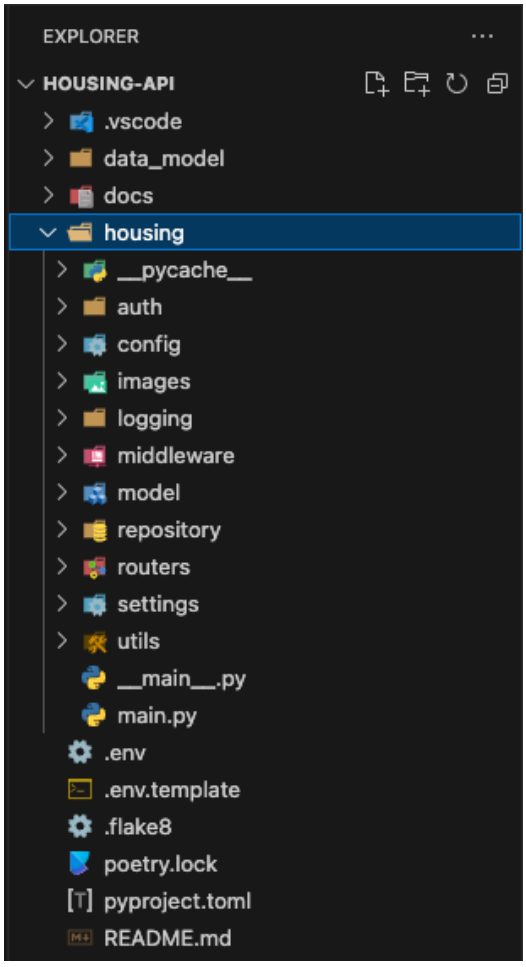


Figura 3.4.1 Estructura del código de **Housing API**

Tecnologías y configuración que rodean al servicio backend:

.vscode: Configuraciones específicas del editor de código Visual Studio Code.
docs: Documentación del proyecto y diagramas.
auth: Módulo encargado de la autenticación y autorización de usuarios.
config: Configuración general de la aplicación, generalmente para la inicialización de las variables de entorno.
images: Almacena las imágenes de las viviendas guardadas en la aplicación.
logging: Configura el sistema de logging para registrar logs de información y/o error.
middleware: Contiene middleware para procesar solicitudes HTTP (Hypertext Transfer Protocol), elevando errores controlados antes de que lleguen a la capa de servicio.
model: Contiene los modelos de respuesta de nuestro API (Application Programming Interface).
repository: Implementa el patrón de repositorio, encapsulando la lógica de acceso a datos.
routers: Define las rutas de la API y los controladores que manejan las solicitudes HTTP.
utils: Contiene funciones utilitarias que se utilizan en diferentes partes del código. Utilizada sobre todo para el calculo de afinidad entre usuarios dentro del sistema.
***__main__*.py:** Punto de entrada principal de la aplicación.
main.py: Inicialización de repositorios, configuración del logger, definición de routers…
.env: Archivo de configuración de entorno.
.env.template: Plantilla para el archivo .env.
.flake8: Configuración del linter flake8.
poetry.lock: Archivo generado por Poetry que lista las versiones exactas de las dependencias.
pyproject.toml: Archivo de configuración de Poetry para el proyecto.
README.md: Archivo que contiene información general sobre el proyecto, como instrucciones de instalación y uso.

3.5 Estructura del frontend

El frontend de la aplicación está desarrollado utilizando **React Native**, una plataforma para la creación de aplicaciones móviles nativas con JavaScript. En este apartado, se detalla la estructura del frontend, abarcando la gestión de dependencias, la conexión con la API y la organización del código.

3.5.1 Gestor de Dependencias

La gestión de dependencias en el frontend se realiza a través de **npm** (Node Package Manager), que es el administrador de paquetes estándar para proyectos JavaScript. El archivo package.json es el núcleo de la gestión de dependencias, donde se definen todas las librerías necesarias para el proyecto, así como sus versiones específicas.

Para instalar todas las dependencias necesarias, se utiliza el siguiente comando:

- npm install

Entre las principales dependencias gestionadas por **npm** en este proyecto se incluyen:

- **React Navigation**: Para gestionar la navegación entre las diferentes pantallas de la aplicación.
- **React Native Elements**: Para el uso de componentes de interfaz de usuario estilizados y predefinidos.
- **Expo**: Aunque no es una dependencia directa de **npm**, Expo CLI (Expo Command Line Interface) se utiliza para ejecutar, desarrollar y probar la aplicación.

Estas dependencias aseguran que el proyecto esté bien organizado y que las herramientas necesarias estén disponibles para el desarrollo continuo.

3.5.2 Conexión con la API

Para la interacción con el backend, la aplicación utiliza la librería **requests** para realizar solicitudes HTTP a la API. Las solicitudes a la API se configuran para comunicarse con los endpoints definidos en el backend, utilizando la URL (Uniform Resource Locator) base configurada en las variables de entorno (.env).

En cada solicitud, se incluye el token de autenticación en los encabezados para garantizar que solo los usuarios autorizados puedan acceder a los datos. La respuesta de la API se maneja mediante promesas, y se implementa un manejo de errores adecuado para gestionar cualquier problema en la comunicación.

3.5.3 Estructura de Código

La estructura del código en el frontend sigue un enfoque modular, lo que facilita la gestión, mantenimiento y escalabilidad de la aplicación. A continuación, se detalla la estructura principal del código:

- **/assets/**: Este directorio incluye los recursos estáticos como imágenes.
- **/components/**: Este directorio se compone componentes reutilizables que pueden ser utilizados en múltiples pantallas. Concretamente, los formularios para el registro, el inicio de sesión y el relativo a la inclusión los datos del usuario.
- **/providers/**: Contiene los contextos de React que gestionan el estado global de la aplicación, como la autenticación del usuario. Estos contextos permiten compartir información importante entre diferentes componentes de la aplicación sin necesidad de pasar props manualmente a lo largo de la jerarquía de componentes.
- **/screens/**: Contiene los componentes principales de la aplicación, cada uno representando una pantalla específica, como LoginScreen.js, HomeScreen.js, EditApartmentScreen.js. Estos componentes gestionan la interfaz y la lógica correspondiente a cada funcionalidad principal de la aplicación.

- **/styles/**: Centraliza los estilos globales y comunes que se aplican en toda la aplicación.
- **App.js**: Es el punto de entrada principal de la aplicación, donde se inicializan los contextos globales, la configuración de la navegación y otros parámetros fundamentales para el funcionamiento de la aplicación.

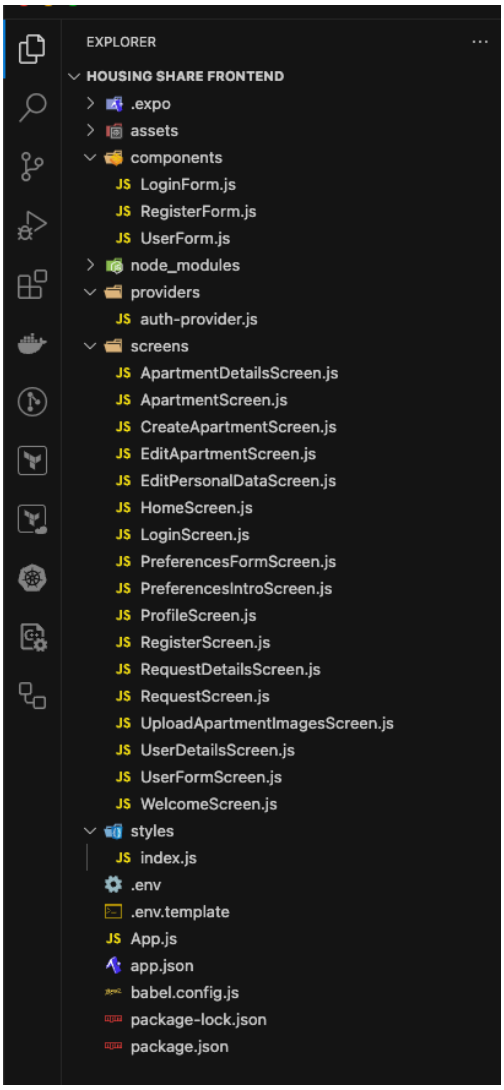


Figura 3.5.2 Estructura del código de **Housing Share Frontend**

3.6 Casos de uso

A continuación se presentan los escenarios posibles de los casos de uso identificados junto a un escenario alternativo que completa la acción ejecutada.

A su vez, todos estos casos de uso son apoyados por sus correspondientes diagramas de secuencia para poder visualizar el comportamiento de la aplicación.

Título	Inicio de sesión.
Descripción	Caso de uso que contempla un inicio de sesión de un usuario registrado en el sistema.
Precondición	El usuario ya se encuentra registrado.
Postcondición	El usuario inicia sesión correctamente.
Escenario principal	1. El usuario introduce su email y su contraseña. 2. El usuario hace click en 'Iniciar sesión'. 3. El sistema verifica los datos de entrada. 4. El sistema codifica la contraseña en base a una clave secreta. 5. El sistema compara la contraseña guardada (y codificada) en base de datos con la insertada por el usuario y valida la operación. 6. El usuario es redirigido a la vista principal "Explorar".
Escenario alternativo	3.B El sistema verifica que el email o la contraseña no vienen rellenos. 4.B El sistema no codifica la contraseña. 5.B El sistema invalida la operación. 6.B El usuario se mantiene en la vista de "Inicio de sesión" y recibe un error controlado. 5.C El sistema compara ambas contraseñas pero no coinciden, invalidando la operación. 6.C El usuario se mantiene en la vista de "Inicio de sesión" y recibe un error controlado.

Diagramas de secuencia.

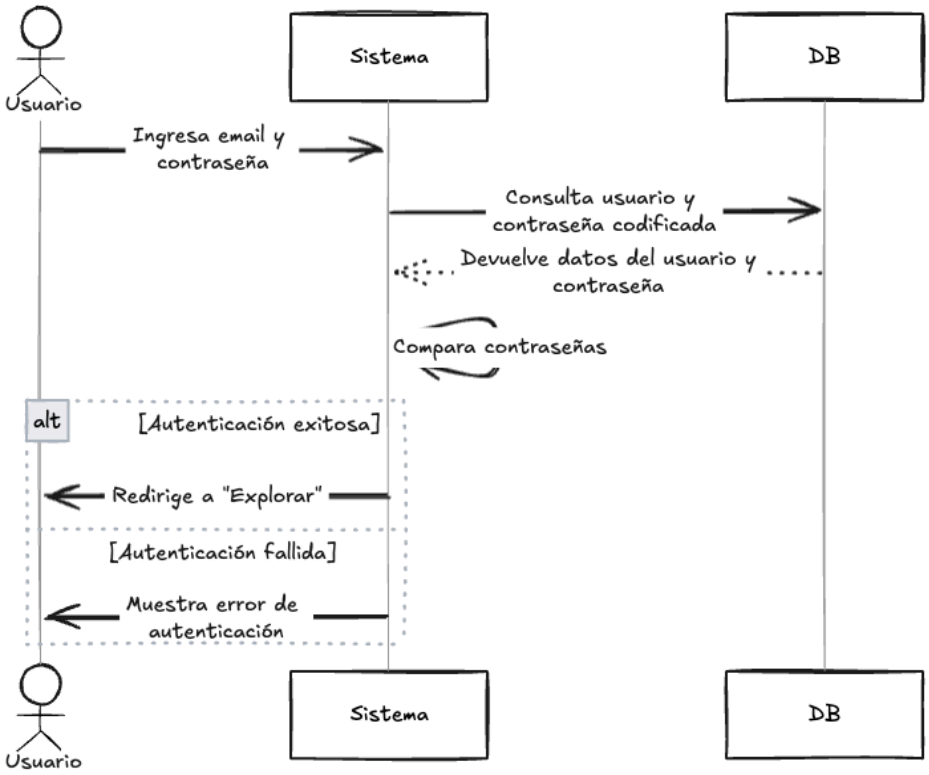


Figura 3.6.1 Diagrama de secuencia de Inicio de sesión.

Título	Registro
Descripción	Caso de uso que contempla un nuevo registro en la aplicación.
Precondición	-
Postcondición	El usuario se registra correctamente.
Escenario principal	1. El usuario hace click en ‘Crear cuenta’. 2. El usuario inserta su email, nombre de usuario, contraseña y la confirmación de la contraseña. 3. El usuario hace click en ‘Crear cuenta’. 4. El sistema compara ambas contraseñas para ver si coinciden y valida la petición. 5. El sistema codifica la contraseña en base a una clave secreta y valida la primera parte del registro. 6. El usuario recibe la pantalla de Bienvenida y hace click en “Crear mi perfil”. 7. El usuario introduce sus datos personales (nombre, apellido, fecha de nacimiento…) y hace click en “Añadir mis datos”. 8. El usuario recibe otra pantalla de transición antes de rellenar sus preferencias a la hora de buscar compañeros de piso y hace clicl en “Continuar”. 9. El usuario introduce sus preferencias de afinidad y hace click en “Guardar preferencias”. 10. El sistema calcula y guarda las afinidades con el resto de usuarios registrados. 11. El usuario es redirigido a la vista principal “Explorar”.
Escenario alternativo	4.B El sistema compara ambas contraseñas para ver si coinciden e invalida la petición. 5.B El usuario se mantiene en la vista de “Registro” y recibe un error controlado. 2.C / 7.C / 9.C El sistema verifica que alguno de los campos obligatorios no vienen rellenos. 3.C / 8.C / 10.C El sistema invalida la operación. 4.C / 9.C / 11.C El usuario se mantiene en la vista actual y recibe un error controlado.

Diagramas de secuencia.

- Escenario normal.

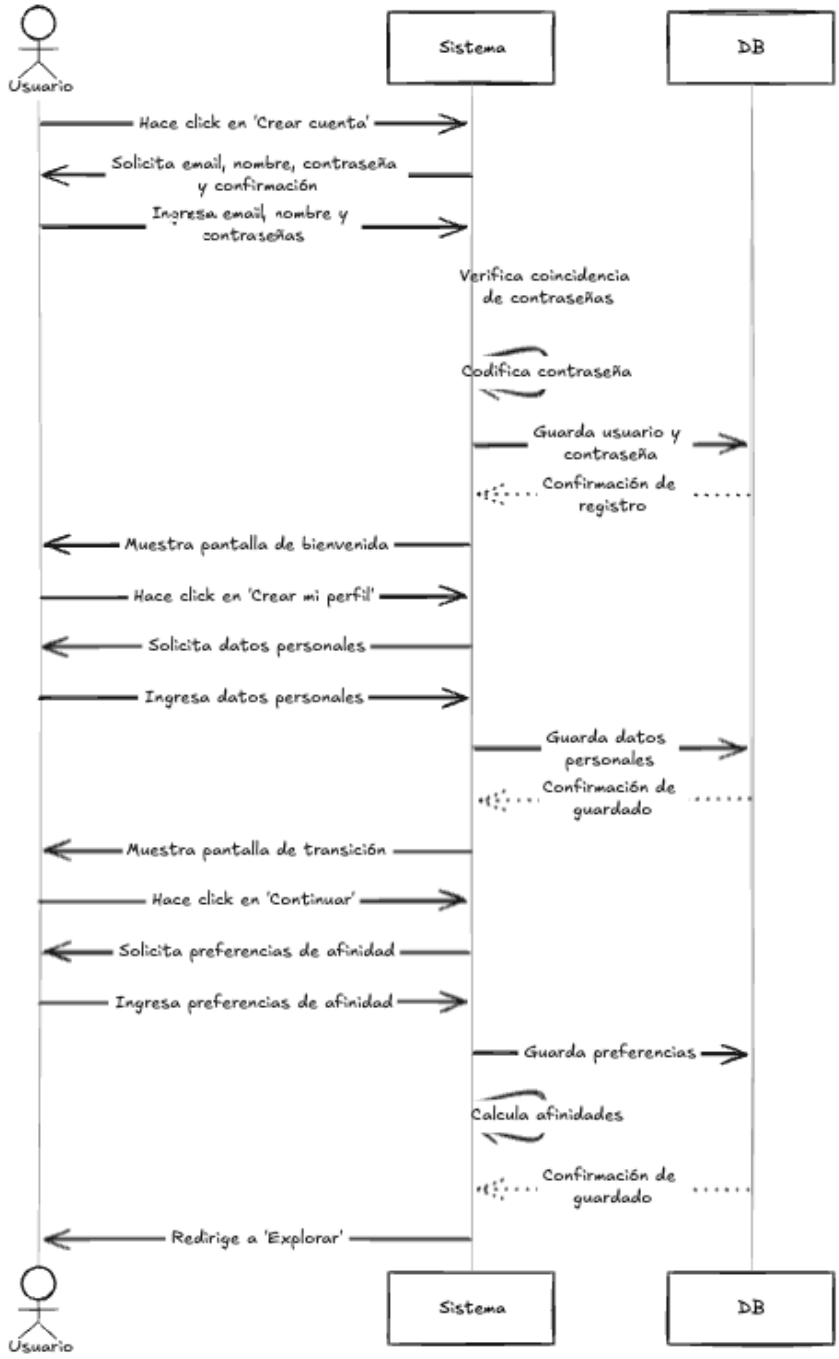


Figura 3.6.2 Diagrama de secuencia de Registro

- Escenario alternativo.

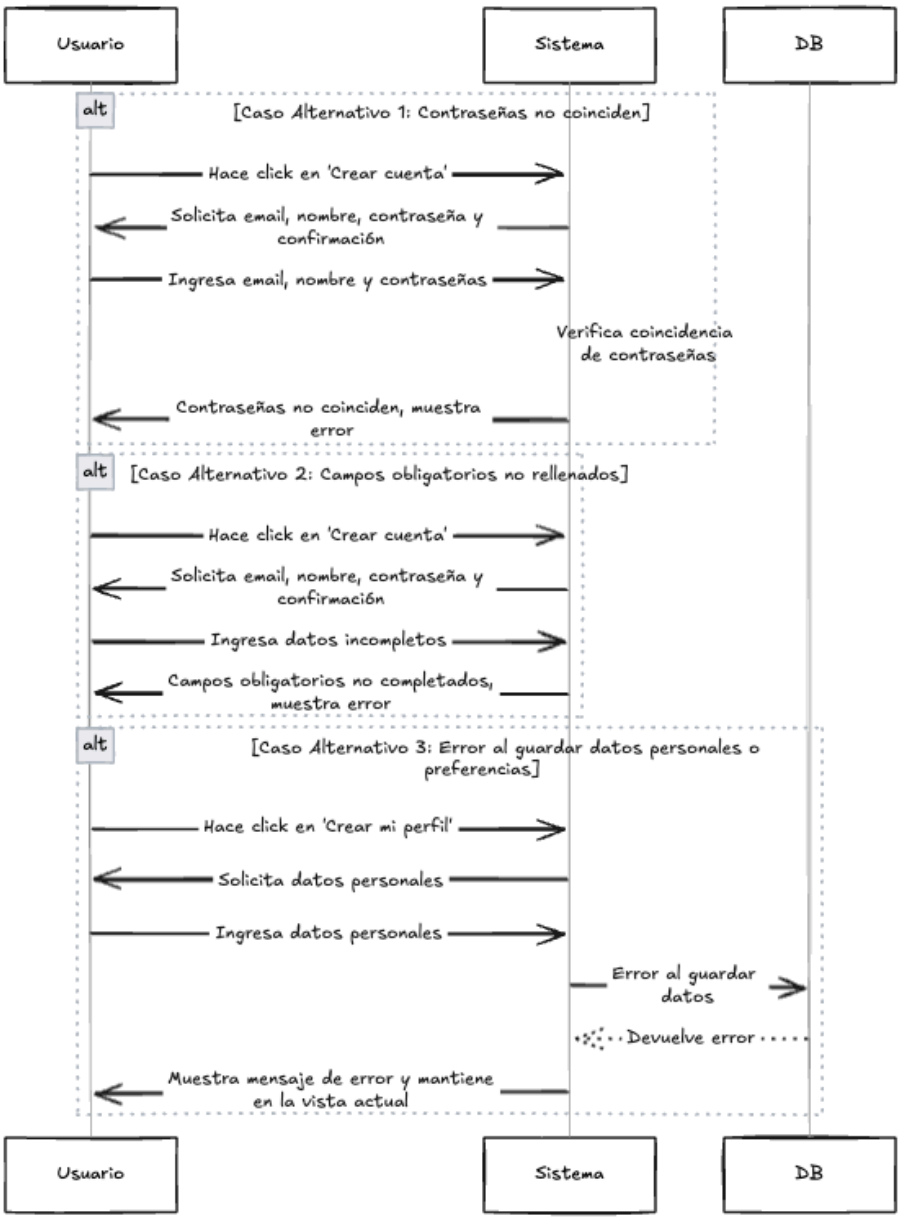


Figura 3.6.3 Diagrama de secuencia alternativo de Registro

Título	Añadir un apartamento.
Descripción	Caso de uso que contempla la creación de un apartamento.
Precondición	El usuario que añade el apartamento ya se encuentra registrado y logeado en la aplicación.
Postcondición	El apartamento es creado correctamente y visible para el resto de usuarios.
Escenario principal	1. El usuario hace click en la pestaña 'Mis apartamentos'. 2. El usuario hace click en 'Nuevo apartamento'. 3. El usuario introduce los datos del apartamento (dirección, ciudad, código postal…) y hace click en "Guardar apartamento". 4. El sistema valida el apartamento. 5. El sistema manda un mensaje de confirmación al usuario. 6. El usuario hace click en "Seleccionar imagen" y selecciona las imágenes de su apartamento. 7. El usuario recorta la/las imagen/imágenes seleccionadas y hace click en "Recortar". 8. El sistema redirige al usuario a la vista de selección de imágenes. 9. El usuario hace click en "Subir imágenes". 10. El sistema valida las imágenes y las almacena. 11. El usuario hace click en "Finalizar". 12. El usuario es redirigido a la vista "Mis apartamento".
Escenario alternativo	4.B El sistema no valida los datos del apartamento. 5.B El sistema manda un mensaje de error controlado al usuario. 6.B El usuario se mantiene en la vista actual "Crear apartamento"

Diagramas de secuencia.

- Escenario normal.

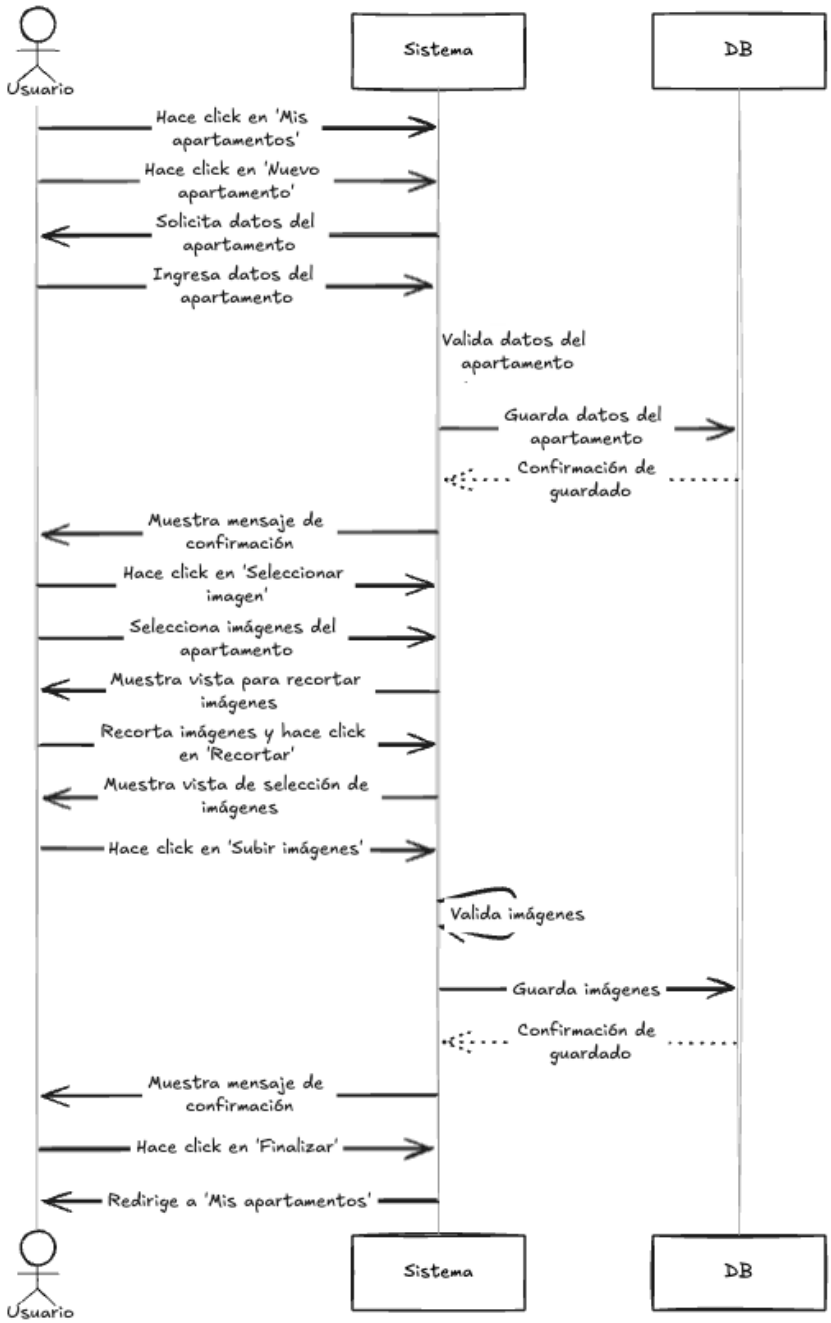


Figura 3.6.4 Diagrama de secuencia de Crear apartamento.

- Escenario alternativo.

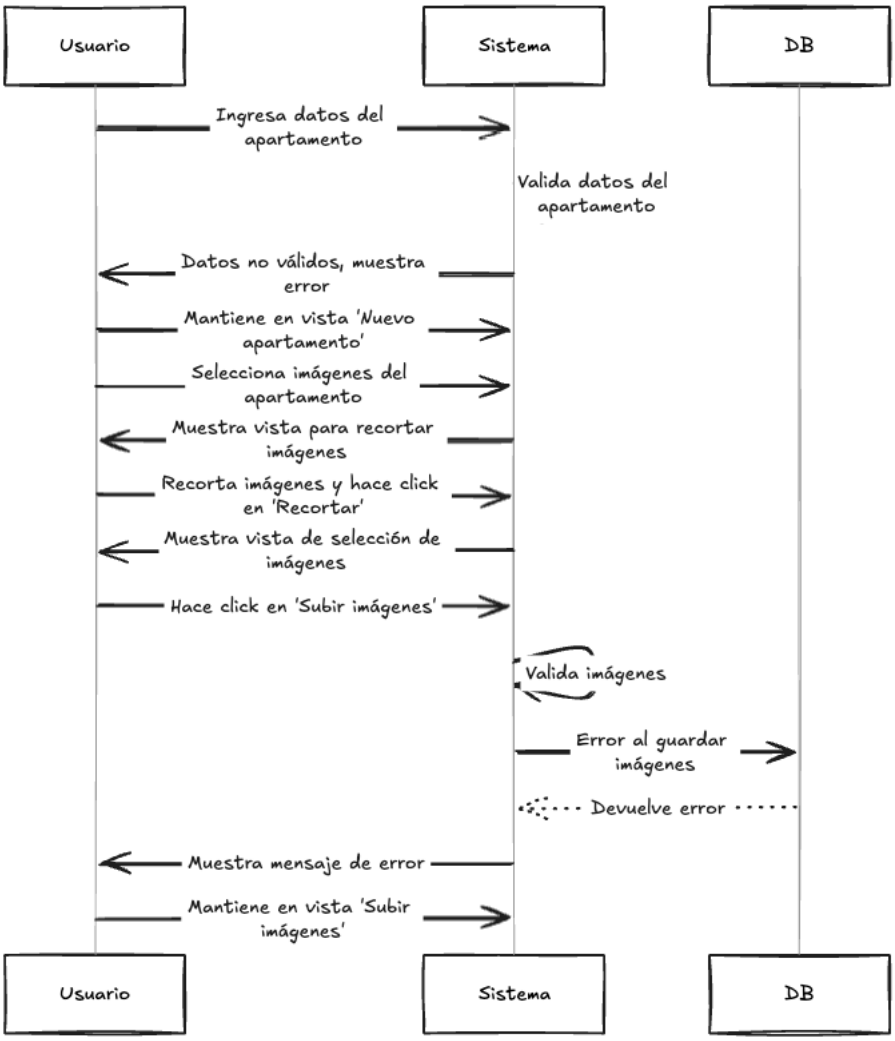


Figura 3.6.5 Diagrama de secuencia alternativo de Crear apartamento.

Título	Editar un apartamento.
Descripción	Caso de uso que contempla la modificación de un apartamento.
Precondición	El apartamento ha sido creado anteriormente. El usuario que modifica el apartamento ya se encuentra registrado y logeado en la aplicación. El apartamento fue creado por el mismo usuario que va a modificarlo.
Postcondición	El apartamento es modificado correctamente y visible para el resto de usuarios.
Escenario principal	1. El usuario hace click en la pestaña 'Mis apartamentos'. 2. El usuario hace click en 'Editar' dentro de la vista del apartamento. 3. El sistema redirige al usuario a la vista de modificación del apartamento. 4. El usuario cambia los datos del apartamento (dirección, ciudad, código postal…) que desea modificar y hace click en "Guardar cambios". 5. El sistema valida los nuevos cambios del apartamento. 6. El sistema manda un mensaje de confirmación al usuario.
Escenario alternativo	5.B El sistema no valida los nuevos datos del apartamento. 5.B El sistema manda un mensaje de error controlado al usuario. 6.B El usuario se mantiene en la vista actual "Editar apartamento"

Diagramas de secuencia.

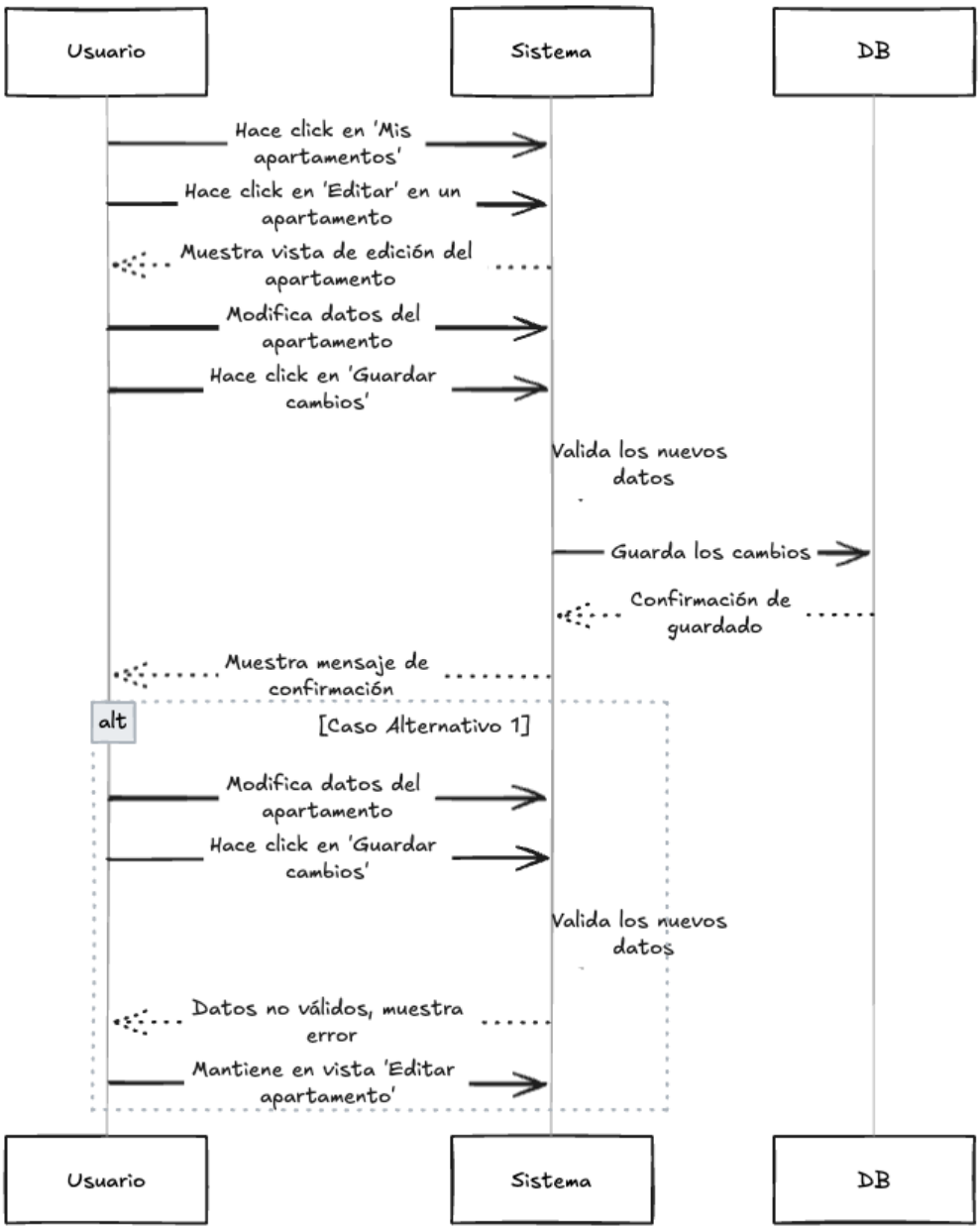


Figura 3.6.6 Diagrama de secuencia de Editar apartamento.

Título	Eliminar un apartamento.
Descripción	Caso de uso que contempla la eliminación de un apartamento.
Precondición	El apartamento ha sido creado anteriormente. El usuario que elimina el apartamento ya se encuentra registrado y logeado en la aplicación. El apartamento fue creado por el mismo usuario que va a eliminarlo.
Postcondición	El apartamento es eliminado correctamente y visible para el resto de usuarios.
Escenario principal	1. El usuario hace click en la pestaña 'Mis apartamentos'. 2. El usuario hace click en 'Eliminar' dentro de la vista del apartamento. 3. El sistema manda un mensaje de confirmación al usuario antes de borrar el apartamento. 4. El usuario confirma que desea borrar el apartamento. 5. El sistema manda un mensaje de confirmación al usuario.
Escenario alternativo	4.B. El usuario confirma que desea borrar el apartamento. 5.B El sistema manda un mensaje de error controlado al usuario. 6.B El usuario se mantiene en la vista actual "Mis apartamentos"

Diagramas de secuencia.

Figura 3.6.7 Diagrama de secuencia de Eliminar apartamento.

Título	Mandar una petición.
Descripción	Caso de uso que contempla el envío de una petición a otros usuarios sobre un apartamento en común.
Precondición	El apartamento ha sido creado anteriormente. El usuario que manda la petición ya se encuentra registrado y logeado en la aplicación. Existen otros usuarios registrado en la aplicación.
Postcondición	La petición es enviada correctamente y visible tanto para el destinario como para el resto de usuarios implicados.
Escenario principal	1. El usuario se encuentra en la pestaña principal 'Explorar'. 2. El usuario hace click en 'Ver detalles' dentro de la vista del apartamento. 3. El sistema muestra los detalles del apartamento. 4. El sistema muestra las personas más afines a ti debajo de los detalles del apartamento. 5. El usuario hace click en "Ver detalles" a través del icono al lado del usuario recomendado. 6. El sistema redirige a la pestaña "Detalles del usuario" y muestra la información del usuario. 7. El usuario hace click en "Volver". 8. El sistema redirige a la pestaña "Detalles del apartamento" de nuevo. 9. El usuario selecciona el usuario con el que quiere enviar la petición. 10. El usuario hace click en "Enviar petición". 11. El sistema valida la petición. 12. El sistema registra la petición. 13. El sistema manda un mensaje de confirmación al usuario.
Escenario alternativo	10.B El usuario hace click en "Enviar petición". 11.B El sistema no valida la petición. 12.B El sistema manda un mensaje de error controlado al usuario.

Diagramas de secuencia.

- Escenario normal.

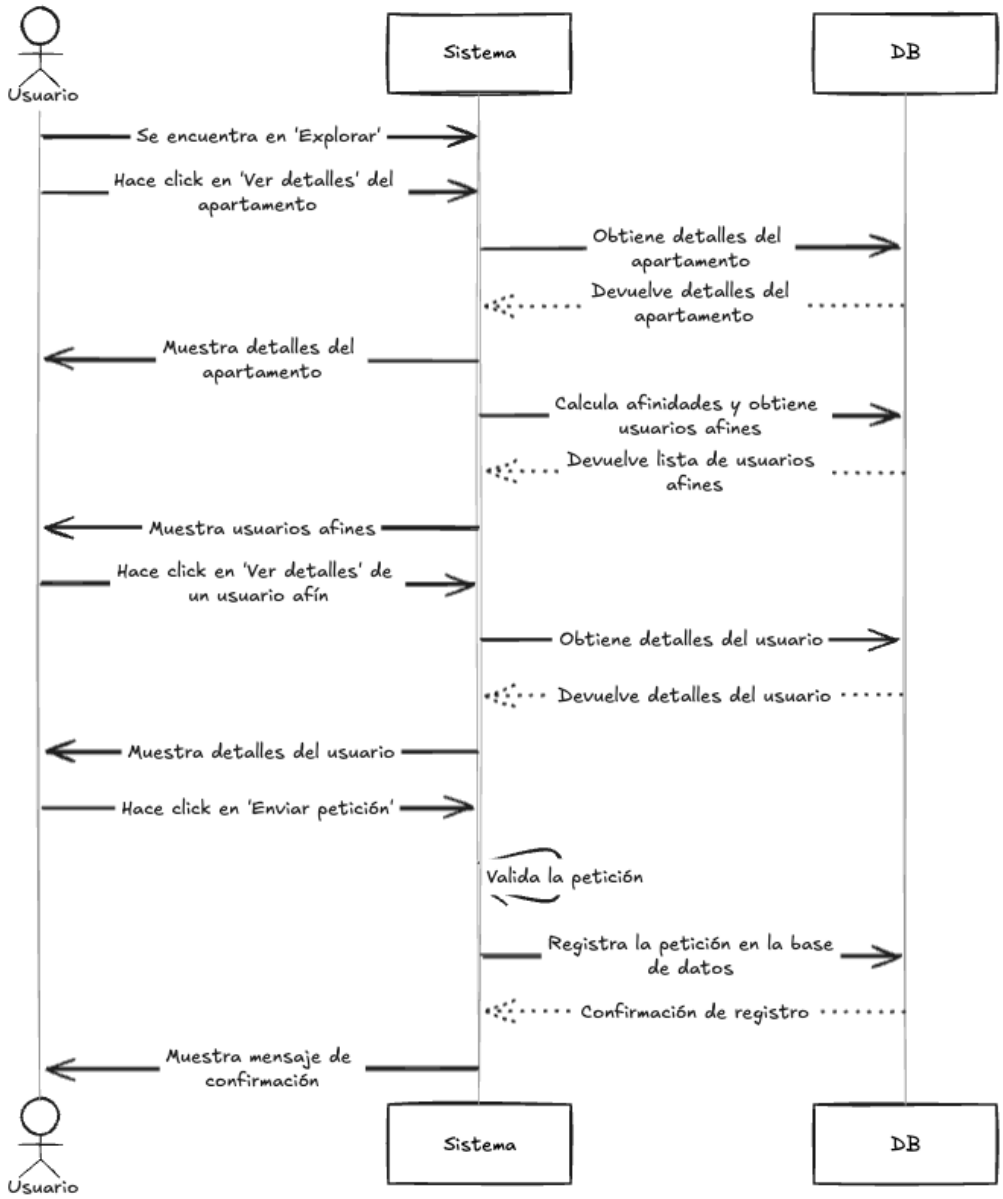


Figura 3.6.8 Diagrama de secuencia de Mandar una petición.

- Escenario alternativo.

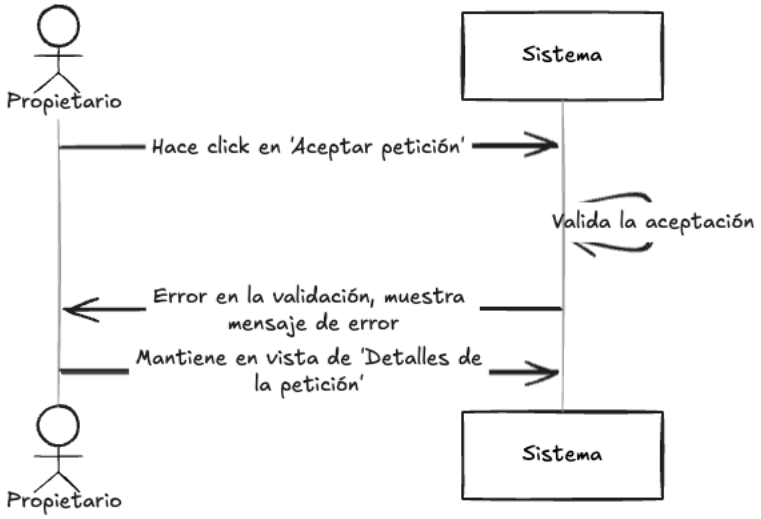


Figura 3.6.9 Diagrama de secuencia alternativo de Mandar una petición.

Título	Modificar las preferencias de afinidad
Descripción	Caso de uso que contempla la modificación de las preferencias de afinidad de un usuario
Precondición	El usuario que modifica sus preferencias ya se encuentra registrado y logeado en la aplicación.
Postcondición	Las preferencias de usuario son modificadas correctamente y sus registros de afinidad con el resto de usuarios se han actualizado.
Escenario principal	1. El usuario hace click en la pestaña 'Mi perfil'. 2. El sistema recibe la información del usuario. 3. El sistema muestra los datos del usuario. 4. El usuario hace click en 'Modifcar mis preferencias' dentro de la sección 'Mis preferencias de afinidad'. 5. El sistema redirige al usuario a la vista de modificación de las preferencias.. 6. El usuario cambia sus preferencias en base a los criterios definidos (edad, género, estilo de vida…) y hace click en "Guardar cambios". 7. El sistema valida las nuevas preferencias del usuario. 8. El sistema manda un mensaje de confirmación al usuario.
Escenario alternativo	5.B El sistema invalida los nuevos datos del usuario. 5.B El sistema manda un mensaje de error controlado al usuario. 6.B El usuario se mantiene en la vista actual "Modifcar mis preferencias"

Diagramas de secuencia.

- Escenario normal.

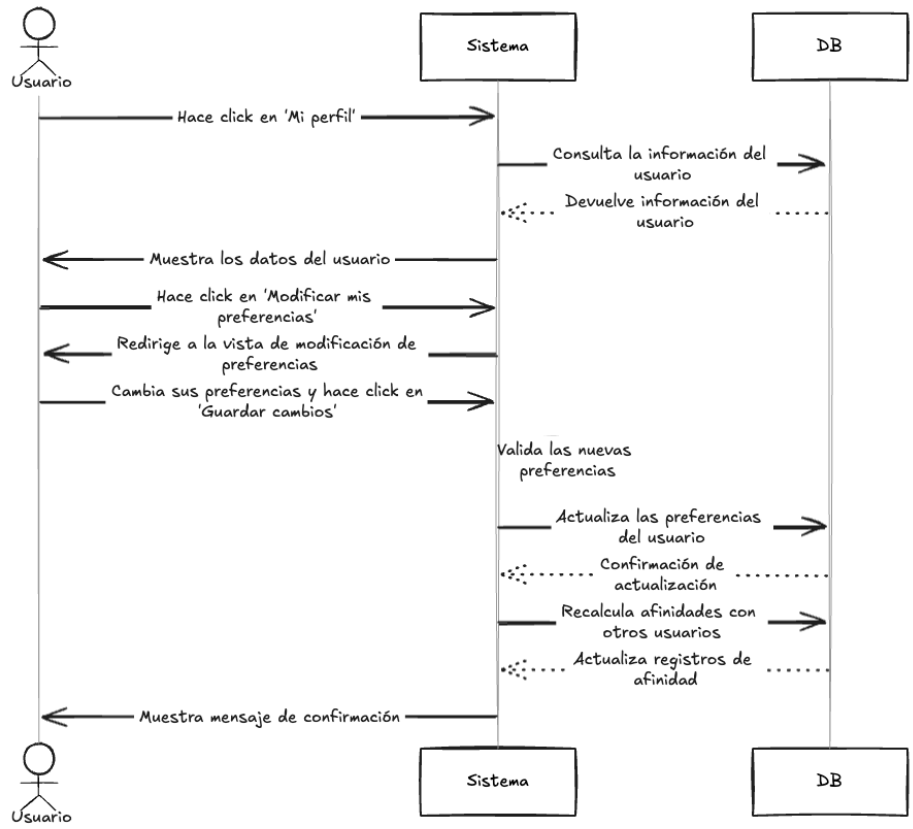


Figura 3.6.10 Diagrama de secuencia de Modificar preferencia de afinidad

- Escenario alternativo.

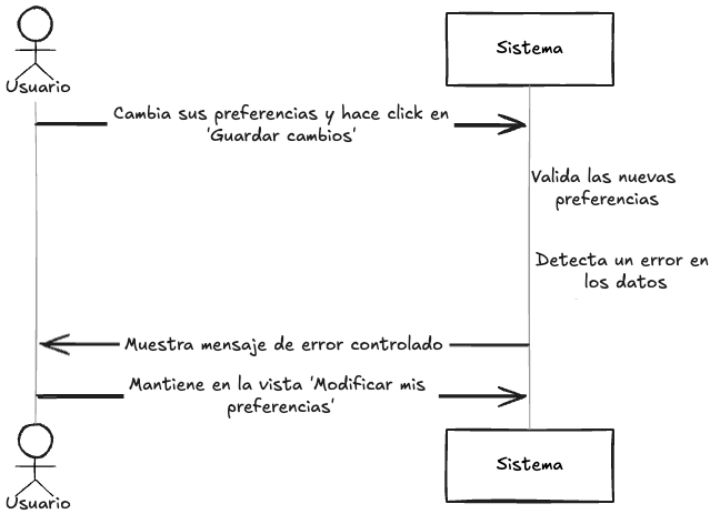


Figura 3.6.11 Diagrama de secuencia alternativo de Modificar preferencia de afinidad

4

Desarrollo e implementación

En este capítulo se detalla el proceso de desarrollo e implementación del prototipo de la aplicación, abarcando desde el diseño de la arquitectura inicial hasta la configuración del entorno de desarrollo y la construcción de los componentes frontend y backend. Cada fase se describe en detalle, destacando las decisiones tecnológicas y metodológicas adoptadas para asegurar un producto robusto y escalable.

6.1 Diseño de la Arquitectura

El primer paso en el desarrollo del prototipo fue diseñar la arquitectura de la aplicación. La arquitectura se dividió en dos componentes principales: el backend, responsable de la gestión de datos y lógica del negocio, y el frontend, encargado de la interacción con el usuario y la presentación de la información.

Para el backend, se eligió **Python** como lenguaje de programación, junto con **PostgreSQL** como base de datos relacional. La elección de PostgreSQL se debió a su robustez, escalabilidad y capacidad de manejar grandes volúmenes de datos de manera eficiente.

El frontend se desarrolló utilizando **React Native** y **JavaScript**, tecnologías que permiten la creación de aplicaciones móviles multiplataforma, garantizando una experiencia de usuario nativa en ambos sistemas operativos principales, iOS y Android.

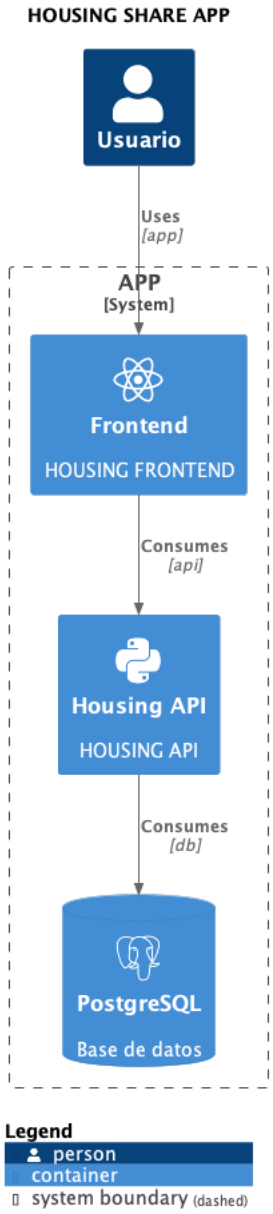


Figura 6.1 Arquitectura de la aplicación

6.2 Configuración del entorno de desarrollo

Con la arquitectura diseñada, el siguiente paso fue configurar el entorno de desarrollo. La gestión de dependencias y la configuración de un entorno virtual adecuado fueron cruciales para asegurar un desarrollo eficiente y sin problemas de compatibilidad.

2.2.1. Gestión de dependencias y entornos virtuales

Para la gestión de dependencias, se optó por **Poetry**, una herramienta moderna que no solo facilita la administración de dependencias en proyectos Python, sino que también maneja la creación y el manejo de entornos virtuales de manera

automática. **Poetry** permite definir las dependencias del proyecto en un archivo pyproject.toml, asegurando que todos los desarrolladores trabajen con las mismas versiones de las librerías, evitando conflictos y garantizando la coherencia entre los entornos de desarrollo y producción.

La estructura del archivo pyproject.toml se organizó para incluir todas las dependencias esenciales, tanto para el desarrollo como para la producción. Esto permitió un control detallado sobre las versiones de las bibliotecas, asegurando que las actualizaciones o cambios no introduzcan errores inesperados.

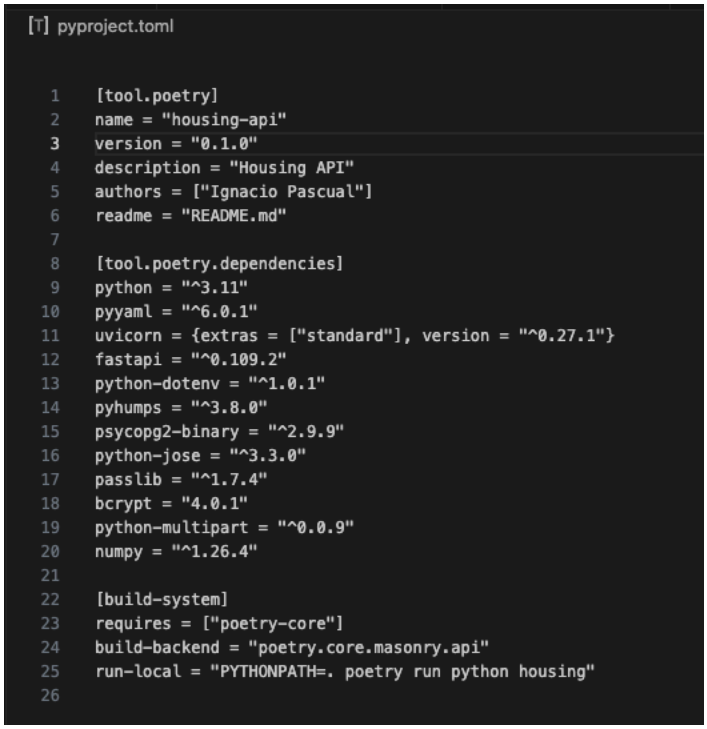


Figura 6.2 Librerías instaladas en Housing API

2.2.2. Estándares de Código y Análisis Estático

Para mantener la calidad del código y asegurar su consistencia, se implementó **flake8** como herramienta de análisis estático. **flake8** verifica que el código cumpla con los estándares de PEP8, el estándar de estilo de código para Python, lo que ayuda a mantener un código limpio y legible.

Integrar **flake8** en el proceso de desarrollo resultó ser una decisión acertada, ya que facilitó la detección temprana de errores de sintaxis, problemas de estilo y otras irregularidades que podrían haber causado problemas en fases posteriores del proyecto. Se configuró un archivo **.flake8** en el repositorio para definir las reglas específicas de estilo que debían seguirse, permitiendo un control centralizado sobre cómo se verifica el código.


```python
# USER PREFERENCES


@router.get(
    "/user/{user_id}/preferences",
    summary="Obtiene información detallada de las preferencias de un usuario dentro del sistema",
    responses={
        500: {
            "model": ResponseException,
            "description": "Database exception",
        },
    },
    tags=["user"],
)
async def get_user_preferences(user=Depends(get_current_user)):  # -> User:
    return await user_repository.get_user_preferences(conn, user)
```

Figura 6.3 Error detectado por el linter Flake8

6.3 Desarrollo del backend

Con el entorno de desarrollo completamente configurado, el desarrollo del backend comenzó con la configuración de la base de datos y la implementación de la API.

2.3.1. Estructura del Código y Patrón de Repositorio

El código del backend se organizó siguiendo el **patrón por repositorio**, que permite separar la lógica de negocio del acceso a los datos. Este patrón facilita el mantenimiento del código, la prueba de las funcionalidades y la escalabilidad del sistema. Cada repositorio encapsula las operaciones de acceso a los datos, permitiendo que la lógica de negocio interactúe con los datos de manera estructurada y coherente.

Por ejemplo, el repositorio encargado de gestionar los usuarios y sus preferencias maneja todas las operaciones de creación, lectura, actualización y eliminación (CRUD) de los datos del usuario, mientras que otro repositorio se encarga de las operaciones relacionadas con los apartamentos. Esta separación asegura que cualquier cambio en la lógica de datos no afecte otras partes del sistema.

A continuación, se desglosan los componentes y el flujo de datos, explicando cómo cada parte del sistema interactúa con las demás para procesar las solicitudes y entregar los datos necesarios.

Inicio del flujo: Solicitud del cliente

El flujo de datos en la aplicación comienza cuando un cliente realiza una solicitud a uno de los endpoints expuestos por la API de FastAPI. Estos endpoints están definidos en los routers, que actúan como controladores de las solicitudes HTTP. Por ejemplo, cuando un cliente desea obtener los detalles de una petición específica, realiza una solicitud GET al endpoint ***/request/{request_id}.***

Router: Gestión de la solicitud

El router, al recibir la solicitud, delega la operación al servicio correspondiente. En el ejemplo anterior, el router request_routes recibe la solicitud y llama al servicio request_service para procesarla. Los routers están diseñados para actuar como puntos de entrada a la lógica de negocio, canalizando las solicitudes a los servicios adecuados según la naturaleza de la operación solicitada.

En nuestro fichero principal 'main.py', inicializamos la app usando la librería de FastAPI y declaramos los routers en base a la lógica establecida.

Service: Lógica de negocio

El servicio es el encargado de implementar la lógica de negocio de la aplicación. Aquí es donde se aplican reglas de negocio, y se coordinan las operaciones entre diferentes repositorios si es necesario. En el servicio request_service, el método get_request se encarga de procesar la solicitud del cliente, delegando la obtención de datos al repositorio.

El servicio llama al repositorio request_repository para que este realice las operaciones necesarias en la base de datos, manteniendo así una clara separación entre la lógica de negocio y el acceso a datos.

En el caso anteriormente definido, en la función get_request dentro del request_service, se modela el formato de la respuesta del API, accediendo a distintos repositorios (request y user) para obtener la información de la petición y de los usuarios implicados en dicha petición.

Repository: Acceso a datos

El repositorio encapsula toda la lógica de acceso a datos, gestionando las consultas SQL, inserciones, actualizaciones y eliminaciones en la base de datos. En este caso, el repositorio tiene un método get_request_by_id que ejecuta una consulta en la base de datos para obtener los detalles de una petición específica.

El método get_request_by_id utiliza la conexión a la base de datos (conn) proporcionada por el servicio, ejecuta la consulta, y devuelve el resultado mapeado a un objeto Request que el servicio puede utilizar.

Base de Datos: Ejecución de la consulta

El repositorio interactúa con la base de datos PostgreSQL, que es la encargada de almacenar y gestionar los datos persistentes de la aplicación. La consulta SQL se ejecuta en la base de datos, y los resultados se devuelven al repositorio en un formato que puede ser procesado por la aplicación.

La utilización de PostgreSQL garantiza que la aplicación pueda manejar grandes volúmenes de datos de manera eficiente, proporcionando además soporte para consultas complejas y operaciones transaccionales.

En este caso, las queries se guardan en la carpeta repository/postgres/queries, también divididas por caso de uso de la aplicación:

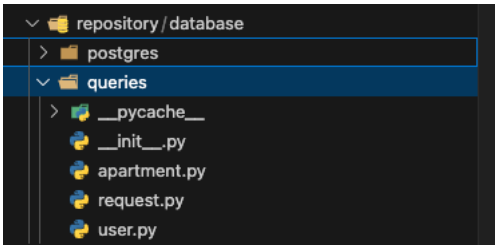


Figura 6.4 Estructura de la carpeta ***repository***

Respuesta del flujo: Devolución de datos al cliente

Una vez que el repositorio ha obtenido los datos de la base de datos, los devuelve al servicio, que a su vez los pasa al router. Finalmente, el router empaqueta los datos en una respuesta HTTP que se envía de vuelta al cliente.

Este flujo asegura que el cliente reciba la información solicitada de manera eficiente, manteniendo la separación de funcionalidad entre los diferentes componentes de la aplicación.

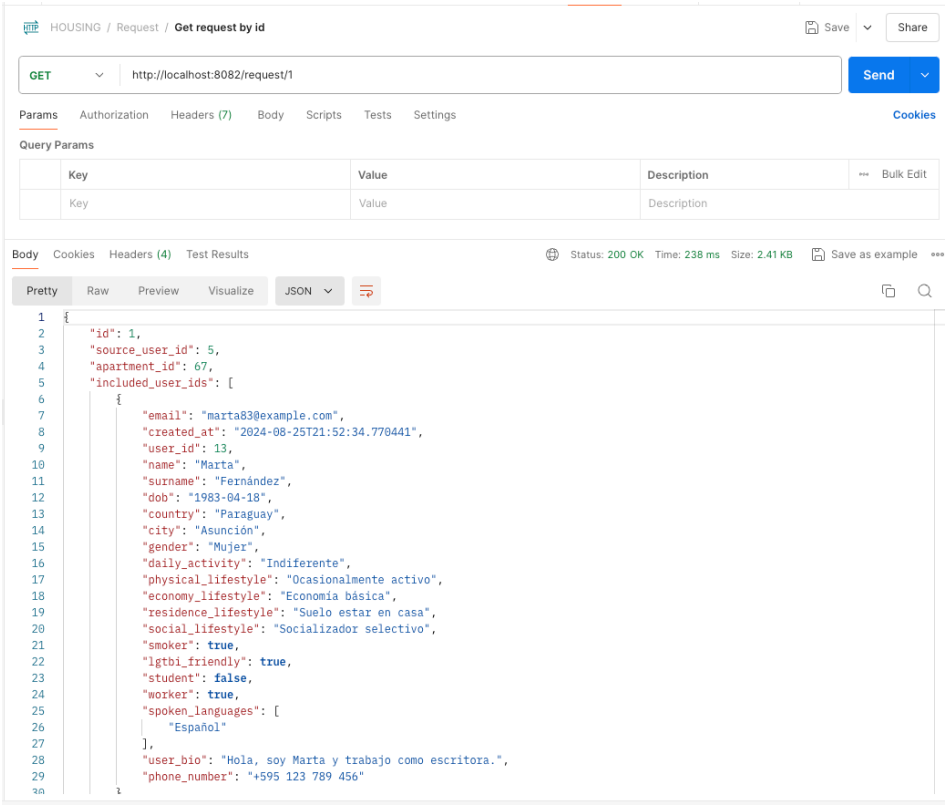


Figura 6.8 Captura de pantalla de la aplicación Postman (ver Apartado 2.1.4) donde se consiguen los detalles de una petición

2.3.2. Implementación de la API

La API del backend fue desarrollada utilizando FastAPI, un framework moderno y altamente eficiente que no solo facilita la creación de APIs robustas y de alto rendimiento, sino que también ofrece una ventaja crucial: la generación automática de documentación de la API mediante Swagger.

La documentación generada incluye información detallada sobre las rutas, que están definidas en el directorio routers, donde se organizan las distintas funcionalidades de la aplicación. Estas funcionalidades incluyen:

- Autenticación: Manejo de la autenticación y autorización de usuarios.
- Gestión de Usuarios: Operaciones CRUD para gestionar la información de los usuarios.
- Gestión de Apartamentos: Operaciones relacionadas con la creación, modificación, y eliminación de anuncios de apartamentos.
- Gestión de Peticiones: Manejo de solicitudes entre usuarios y el propietario del apartamento.

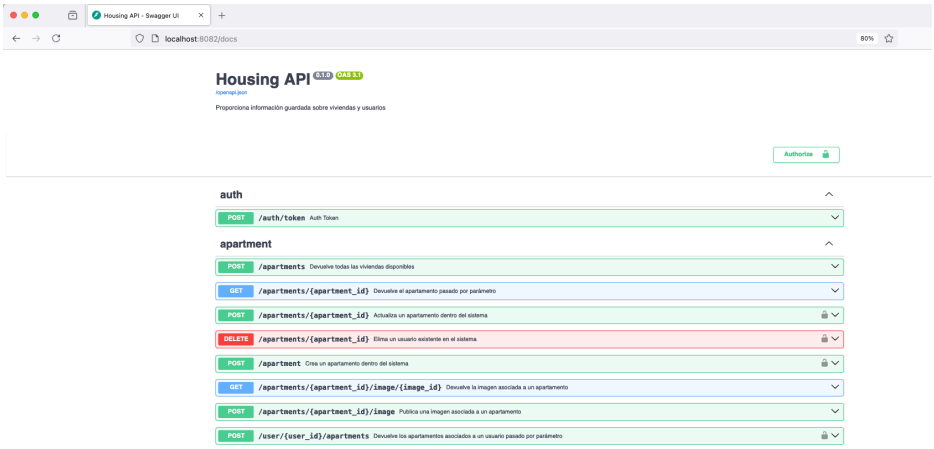


Figura 6.9 Endpoints de la aplicación - Parte 1 (obtenidos mediante Swagger)

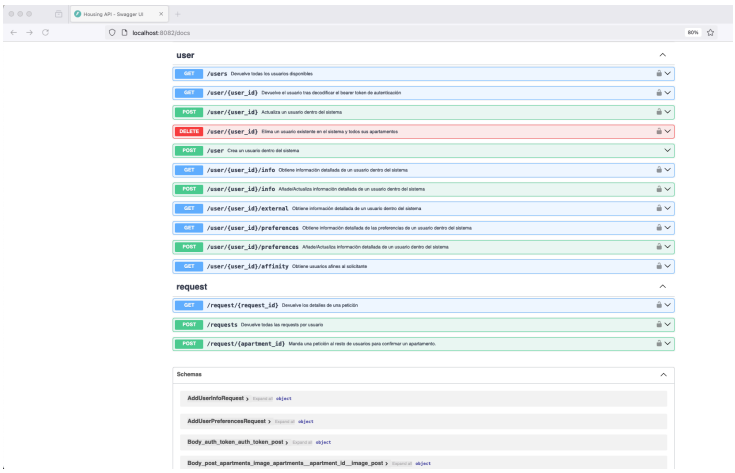


Figura 6.10 Endpoints de la aplicación - Parte 2 (obtenidos mediante Swagger)

2.3.3. Autenticación

La seguridad y autenticación de los usuarios es un aspecto crítico en cualquier aplicación moderna. En nuestra aplicación, la autenticación se maneja mediante la implementación de **JSON Web Tokens (JWT)**, utilizando el framework FastAPI.

Implementación del sistema de autenticación

La clase Auth es el núcleo del sistema de autenticación en nuestra aplicación. Esta clase encapsula toda la lógica relacionada con la autenticación, incluyendo la creación y verificación de contraseñas, la generación de tokens JWT, y la validación de usuarios durante el proceso de inicio de sesión.

Para asegurar que las contraseñas de los usuarios se almacenan de forma segura, la clase Auth utiliza la librería **passlib** y su contexto de hashing CryptContext, configurado para usar el algoritmo bcrypt.

A continuación, se describen las principales funcionalidades de la clase de autenticación:

- **Hashing de contraseñas:** Cuando un usuario se registra o cambia su contraseña, se utiliza el método ***hash_password*** para convertir la contraseña en un hash seguro antes de almacenarla en la base de datos.

- **Verificación de contraseñas:** Durante el proceso de inicio de sesión, se utiliza el método ***verify_password*** para comparar la contraseña proporcionada por el usuario con el hash almacenado en la base de datos.

El uso de **JSON Web Tokens (JWT)** permite a la aplicación autenticar a los usuarios de forma segura sin necesidad de mantener sesiones en el servidor.

Un token JWT se genera tras un inicio de sesión exitoso y se devuelve al frontend, que lo utilizará en las solicitudes subsecuentes para acceder a recursos protegidos.

Creación del token de acceso: El método create_access_token genera un token JWT que contiene información del usuario (como su correo electrónico) y una fecha de expiración. Este token es firmado utilizando una clave secreta y un algoritmo de hashing (HS256).

Decodificación del token de acceso: Para autenticar al usuario en las solicitudes, el método ***decode_access_token*** decodifica el token JWT utilizando la misma clave secreta y verifica su validez. Si el token es inválido o ha expirado, se lanza una excepción HTTP 401.

El método **authenticate_user** es el encargado de validar las credenciales del usuario durante el inicio de sesión. Este método primero consulta la base de datos para verificar que el correo electrónico del usuario existe. Luego, compara la contraseña proporcionada con la almacenada utilizando el método ***verify_password***.

Por último, el método login es llamado cuando un usuario intenta iniciar sesión en la aplicación. Este método valida las credenciales utilizando ***authenticate_user***, y si son correctas, genera un token JWT usando ***create_access_token*** que se devuelve al cliente.

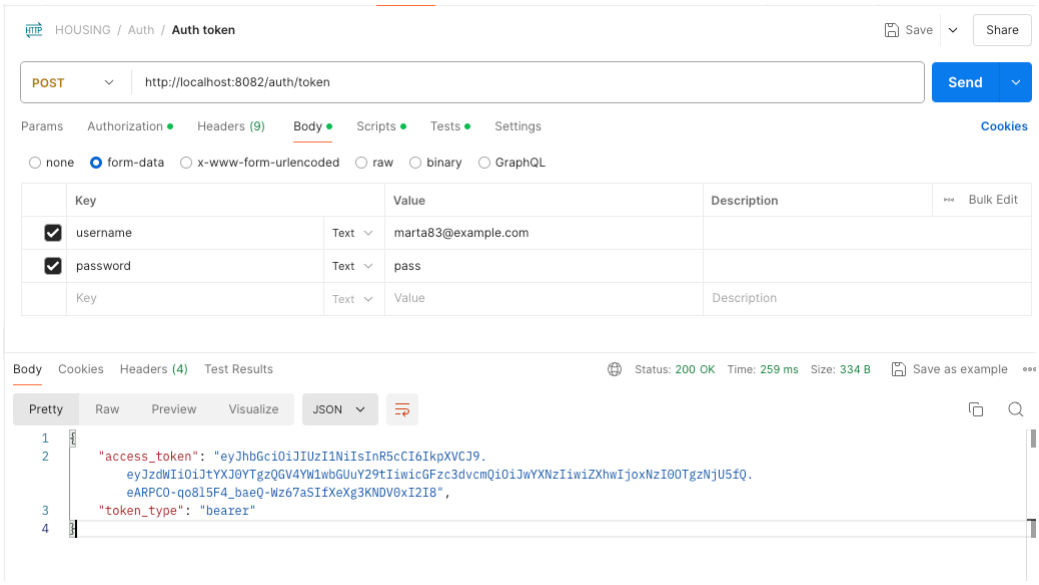


Figura 6.18 Captura de pantalla de la aplicación Postman (ver Apartado 2.4) donde se consigue un token de usuario al introducir las credenciales adecuadas

2.3.4. Gestión y cálculo de afinidades.

En el desarrollo de la aplicación, la gestión y el cálculo de afinidades se han convertido en uno de los componentes más esenciales para asegurar que los usuarios encuentren compañeros de piso compatibles. Este sistema de afinidades se basa en un conjunto de algoritmos diseñados para analizar y comparar las características y preferencias de los usuarios, utilizando técnicas de similitud ponderada para generar recomendaciones personalizadas.

Dividimos el proceso en 3 fases diferenciadas:

Fase 1: El primer paso en el cálculo de afinidades es la preparación y transformación de los datos del usuario. Por ejemplo:

- **Fecha de nacimiento:** Se transforma en un grupo de edad utilizando la función classify_dob_group. Esta función agrupa a los usuarios en uno de los diez rangos de edad predefinidos, lo que facilita la comparación entre usuarios de edades similares.
- **Valores booleanos:** Se convierten en valores numéricos (1 para verdadero y 0 para falso), lo que permite su inclusión en los cálculos de similitud.
- **Preferencias específicas:** Valores como la actividad diaria (diurno, nocturno, indistinto) y el género (Mujer, Hombre, Otro) también se

transforman en valores numéricos, lo que permite una comparación directa entre las preferencias de los usuarios.

Este proceso de transformación asegura que todos los datos relevantes estén en un formato uniforme y listo para el análisis de similitud.

Fase 2: Para el cálculo se utiliza principalmente el **Índice de Jaccard Ponderado** para medir la similitud entre las preferencias de dos usuarios. Este índice se adapta para considerar tanto valores numéricos como listas de preferencias, y su cálculo sigue los siguientes pasos:

Intersección y unión ponderada: El algoritmo compara cada característica de dos usuarios, calculando la intersección y la unión de sus preferencias. Estas comparaciones se ponderan según la importancia de cada característica definida en los pesos (weights).

Cálculo de similitud: La similitud final se expresa como el cociente entre la intersección ponderada y la unión ponderada de las preferencias, multiplicada por 100 para obtener un porcentaje.

Índice de Jaccard Ponderado

$$\text{Índice de Jaccard Ponderado} = \frac{\sum_{i=1}^{n} \min(a1_i, a2_i) \times w_i}{\sum_{i=1}^{n} \max(a1_i, a2_i) \times w_i} \times 100$$

Este enfoque permite obtener un valor de similitud que refleja cuán compatibles son dos usuarios en función de sus características y preferencias individuales.

Fase 3: Una vez calculadas las similitudes, el sistema procede a generar un listado de afinidades. Este proceso involucra los siguientes pasos:

1. **Preparación de pesos y datos:** Los pesos (weights) y los datos del usuario se procesan para eliminar cualquier identificador no relevante, como el user_id, y se reordenan para asegurar la coherencia en la comparación.
2. **Cálculo de similitud con otros usuarios:** El sistema compara al usuario objetivo con otros usuarios registrados en la base de datos, calculando la similitud utilizando el método descrito anteriormente.
3. **Listado de afinidades:** El resultado es un listado de usuarios junto con su porcentaje de similitud (jaccard_similarity), lo que permite sugerir aquellos compañeros de piso que tienen mayor compatibilidad con el usuario.

Este listado se ordena de mayor a menor similitud, proporcionando al usuario las mejores opciones de compañeros de piso en función de sus preferencias.

6.4 Desarrollo del frontend

Tras establecer las bases de la parte backend de la aplicación, se inició la construcción del frontend. Utilizando **React Native** y **JavaScript**, se diseñaron las vistas principales de la aplicación, asegurando una experiencia de usuario fluida y responsiva en dispositivos móviles.

2.4.1. Componentes y pantallas principales

AppNavigator

En la aplicación, la navegación se gestiona a través del componente AppNavigator, que utiliza **React Navigation**, una biblioteca popular en el ecosistema de React Native para la creación de navegadores.

El AppNavigator se encarga de determinar qué pantalla debe mostrarse al usuario en función de su estado de autenticación y el flujo de usuario específico (por ejemplo, si es la primera vez que inicia sesión). Este flujo se define en el archivo App.js, donde el componente AppNavigator se incluye dentro de un NavigationContainer y se conecta con un contexto de autenticación (AuthContext) que proporciona información sobre el estado de autenticación del usuario. A su vez, gestiona las rutas y las pantallas según el estado del usuario.

Si el usuario está autenticado (hasToken es verdadero) y es su primer inicio de sesión (isFirstLogin es verdadero), se le presenta un flujo de bienvenida. Si el usuario ya ha iniciado sesión anteriormente, se le dirige directamente a la pantalla principal (HomeScreen). Si el usuario no está autenticado, se le muestra la pantalla de inicio de sesión o registro.

HomeScreen y navegación tabulada

El HomeScreen es el corazón de la aplicación, donde se centralizan las principales funcionalidades a las que el usuario accede regularmente. La navegación en esta sección se gestiona mediante un **BottomTabNavigator**. Este tipo de navegación permite a los usuarios moverse fácilmente entre diferentes secciones de la aplicación, como "Explorar", "Mis Apartamentos", "Solicitudes", y "Mi perfil".

Cada tab está vinculado a un componente específico que maneja la interfaz y la lógica de esa sección. Por ejemplo:

- **Tab de "Explorar":** Conecta al usuario con la pantalla ApartmentScreen, que muestra una lista de apartamentos disponibles.
- **Tab de "Mis Apartamentos":** Muestra los apartamentos que el usuario ha publicado.

- **Tab de "Solicitudes":** Permite al usuario gestionar las peticiones, tanto enviadas como recibidas.
- **Tab de "Mi Perfil":** Ofrece acceso a la gestión de la información personal y preferencias del usuario.

Pantallas individuales y lógica de componentes

Cada pantalla de la aplicación está implementada como un componente individual que encapsula su propia lógica y diseño. Esta separación no solo mejora la organización del código, sino que también facilita la prueba y mantenimiento de cada parte de la aplicación.

Por ejemplo, la pantalla **EditApartmentScreen** permite a los usuarios editar los detalles de un apartamento. Esta pantalla gestiona múltiples estados locales para almacenar temporalmente los datos que el usuario introduce en los formularios, como la dirección, ciudad, número de habitaciones, etc. Cuando el usuario guarda los cambios, la pantalla realiza una llamada a la API para actualizar los datos en el backend.

5

Conclusiones y líneas futuras

8.1. Conclusiones

A lo largo del desarrollo de esta aplicación, la motivación personal y la experiencia directa fueron motores clave que impulsaron cada decisión de diseño y funcionalidad. Gracias a experiencias personales, he vivido en carne propia las dificultades de encontrar un piso compartido, y especialmente de hallar compañeros de piso con los que realmente me pudiera entender y convivir en armonía. Esta experiencia personal fue fundamental para dar forma a la visión de este proyecto: crear una herramienta que no solo facilite la búsqueda de alojamiento, sino que también mejore significativamente la calidad de vida de los usuarios al conectarles con compañeros de piso que compartan sus intereses y valores.

Al llegar al final de este Trabajo de Fin de Grado, podemos afirmar con satisfacción que se ha alcanzado el objetivo principal que me propuse al iniciar este proyecto: desarrollar una solución completa y funcional para la búsqueda y gestión de compañeros de piso y apartamentos, cumpliendo con todos los requisitos que me había planteado. A lo largo del desarrollo, se han implementado con éxito las funcionalidades clave que permiten a los usuarios realizar estas tareas de manera intuitiva y eficiente, tanto en el frontend como en el backend.

Uno de los logros más importantes fue construir un backend sólido que no solo gestiona de manera eficiente la información de usuarios, apartamentos y solicitudes, sino que también garantiza la seguridad y privacidad de los datos mediante un sistema de autenticación robusto.

El desarrollo del frontend fue otro desafío satisfactorio. Crear una interfaz móvil con React Native y aprender dos tecnologías nuevas, aplicando los conocimientos aprendidos y siendo a la vez tanto atractiva como funcional ha sido un proceso gratificante.

Finalmente, cabe destacar las ventajas que este Trabajo de Fin de Grado ofrece. Las tecnologías utilizadas, como la autenticación basada en JWT, el desarrollo móvil con React Native, y la arquitectura backend con FastAPI, son altamente demandadas en el mercado laboral actual debido a su relevancia en el desarrollo de aplicaciones modernas y escalables. Por ello, este proyecto no solo representa un logro académico significativo, sino que también constituye una base sólida para futuras oportunidades profesionales en el campo de la ingeniería informática.

8.2. Líneas Futuras

A pesar del éxito logrado, siempre hay espacio para la mejora y expansión del proyecto. A continuación, se describen algunas líneas futuras que podrían explorarse para continuar mejorando y evolucionando la aplicación:

- **Implementación de notificaciones push:** Incluir notificaciones push para alertar a los usuarios sobre nuevas solicitudes, respuestas a solicitudes existentes, o cambios importantes en los apartamentos de su interés.
- **Expansión de funcionalidades de afinidad:** Ampliar el sistema de afinidad para incluir más criterios y un algoritmo más sofisticado que pueda ofrecer recomendaciones más precisas de compañeros de piso.
- **Internacionalización y localización:** Adaptar la aplicación para soportar múltiples idiomas y configuraciones regionales, lo que ampliaría su alcance a nivel global.
- **Desarrollo de una Versión Web:** Aunque la aplicación móvil es el enfoque principal, una versión web complementaria podría atraer a un segmento más amplio de usuarios y proporcionar una experiencia más versátil.
- **Chat dentro de la aplicación:** Actualmente la aplicación no soporta ningún chat de mensajería interno, sino que, una vez se aprueba una petición, se muestra la información de contacto de todas las partes implicadas para que puedan formalizar el alquiler a través de un sistema externo (Whatsapp o email).

Referencias

[1] (Agosto de 2024) **FastAPI** - El framework moderno para desarrollo web con Python, (https://web.archive.org/web/20240825135325/https://fastapi.tiangolo.com/es/)

[2] (Agosto de 2024) **PostgreSQL** - La base de datos relacional de código abierto más avanzada del mundo, https://www.postgresql.org/

[3] (Agosto de 2024) **React Native** - Crea aplicaciones móviles nativas utilizando JavaScript y React, https://web.archive.org/web/20240826074556/https://reactnative.dev/

[4] (Agosto de 2024) **Expo** - Un framework y una plataforma para aplicaciones React universales, https://expo.dev/

[5] (Agosto de 2024) **npm** - Administrador de paquetes para Node.js, https://web.archive.org/web/20240823004019/https://www.npmjs.com/

[6] (Agosto de 2024) **npx** - Una herramienta para ejecutar paquetes de Node, https://web.archive.org/web/20240822093725/https://www.npmjs.com/package/npx

[7] (Agosto de 2024) **JWT (JSON Web Tokens)** - Estándar basado en JSON para crear tokens de acceso, https://jwt.io/

[8] (Agosto de 2024) **Passlib** - Hashing seguro de contraseñas para Python, https://passlib.readthedocs.io/es/stable/

[9] (Agosto de 2024) **Postman** - Simplifica el desarrollo de APIs, https://web.archive.org/web/20240826190332/https://www.postman.com/

[10] (Agosto de 2024) **React Navigation** - Enrutamiento y navegación para tus aplicaciones React Native, https://web.archive.org/web/20240826190332/https://www.postman.com/

[11] (Agosto de 2024) **CryptContext** (Passlib) - Documentación, https://web.archive.org/web/20240828213509/https://passlib.readthedocs.io/en/stable/narr/context-tutorial.html

[12] (Agosto de 2024) **Expo CLI** - Herramienta de línea de comandos para gestionar aplicaciones Expo, https://web.archive.org/web/20240826161913/https://docs.expo.dev/more/expo-cli/

Apéndice A

Manual de Instalación

Este apartado servirá como guía para poder utilizar el sistema implementado o realizar trabajos futuros sobre él. Inicialmente, será necesario disponer de unos requisitos previos que serán de vital importancia para instalar la arquitectura creada y configurar tanto el entorno de desarrollo como la aplicación de búsqueda de compañeros de piso y apartamentos.

Requisitos previos

- **Sistema Operativo:** Windows, macOS o Linux.
- **Python:** Versión 3.11 o superior.
- **Poetry:** Versión 1.8.0 o superior.
- **Node.js y npm:** Versión 20.x o superior.
- **Expo CLI:** Para gestionar y ejecutar la aplicación móvil.
- **PostgreSQL:** Base de datos relacional para almacenar la información de la aplicación (https://www.postgresql.org/download/)
- **Importar el código fuente.**

Instalación del Backend

Crear y Activar un Entorno Virtual: Crea un entorno virtual usando poetry para aislar las dependencias del proyecto.

```
-> poetry env use 3.11
```

Instalar Dependencias del Backend: Utiliza ***poetry*** para instalar las dependencias del backend.

```
-> poetry install
```

Configurar Variables de Entorno: Crea un archivo `.env` en el directorio raíz del proyecto y define las variables de entorno necesarias, como la conexión a la base de datos PostgreSQL.

```
-> cp .env.template .env
```

Luego, edita el archivo `.env` con la configuración adecuada:

```
.env
# APP CONFIG
PORT=8082
ENVIRONMENT="Development"

# DB CONFIG
DB_NAME = "housing_db"
DB_USER = <DB_USER>
DB_PASSWORD = <DB_PASSWORD>
DB_HOST = "localhost"
DB_PORT = "5432"

# LOGGING
LOGGER_CONFIG_PATH="housing/logging/logging.yaml"

# SWAGGER
SWAGGER_ENABLED="true"
DOCS_URL="/docs"
LOGGER_ENV="development"
```

Base de Datos:

Crea y configura la base de datos usando **psql**:

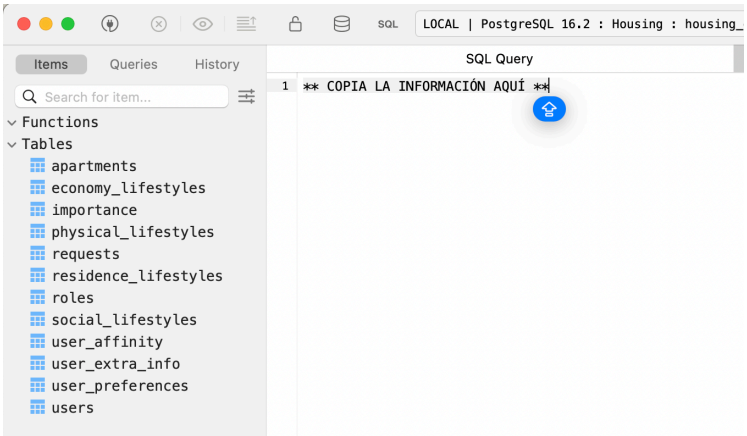
```
-> psql -U postgres

-> CREATE DATABASE housing_db;

-> CREATE USER nombre_de_usuario WITH PASSWORD 'tu_contraseña';

-> GRANT ALL PRIVILEGES ON DATABASE housing_db TO nombre_de_usuario;
```

Conéctate a la base de datos usando TablePlus (ver Figura 2.3) y ejecuta los ficheros alojados en /data_model/init_db.sql en tu base de datos creada ***housing_db.*** Este fichero contiene la creación e inicialización de las tablas necesarias para el modelo de datos establecido.



Iniciar el Servidor Backend: Inicia el servidor backend para manejar las solicitudes de la aplicación.

```
-> PYTHONPATH=./ poetry run python housing
```

Instalación del Frontend

Instalar Node.js y npm: Asegúrate de tener instalada la última versión de Node.js y npm. Puedes verificar la instalación con los siguientes comandos:

```
-> node -v
-> npm -v
```

Instalar Expo CLI: Expo CLI es necesario para ejecutar y desarrollar la aplicación móvil. Instálalo globalmente usando npm:

```
-> npm install -g expo-cli
```

Instalar Dependencias del Frontend: Desde el directorio del frontend del proyecto, instala las dependencias necesarias.

```
-> npm install
```

Configurar Variables de Entorno del Frontend: Crea un archivo .env en el directorio raíz del frontend y define las variables de entorno necesarias para conectar con la API del backend.

```
-> API_URL=http://<IP-DE-HOUSING-API>:8000
```

Iniciar la Aplicación Móvil: Una vez configurado todo, puedes iniciar la aplicación móvil en un emulador o dispositivo real usando el siguiente comando:

```
-> npx expo start --android  # O --ios para iOS
```

Probar la Conexión: Abre la aplicación en el emulador o dispositivo conectado y verifica que la aplicación puede interactuar correctamente con el backend.

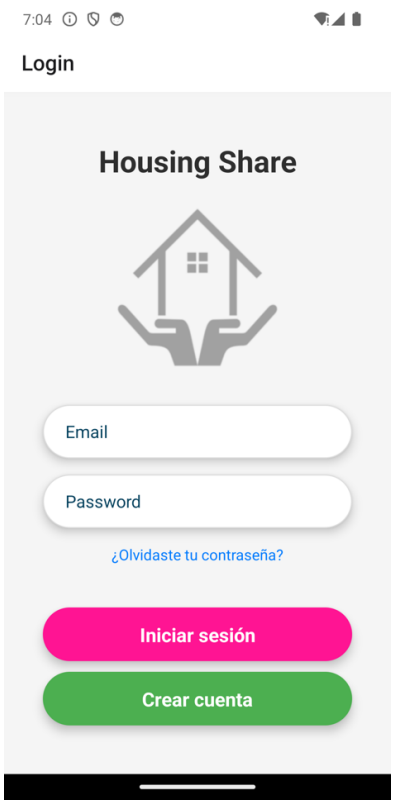
Apéndice B

Manual de usuario

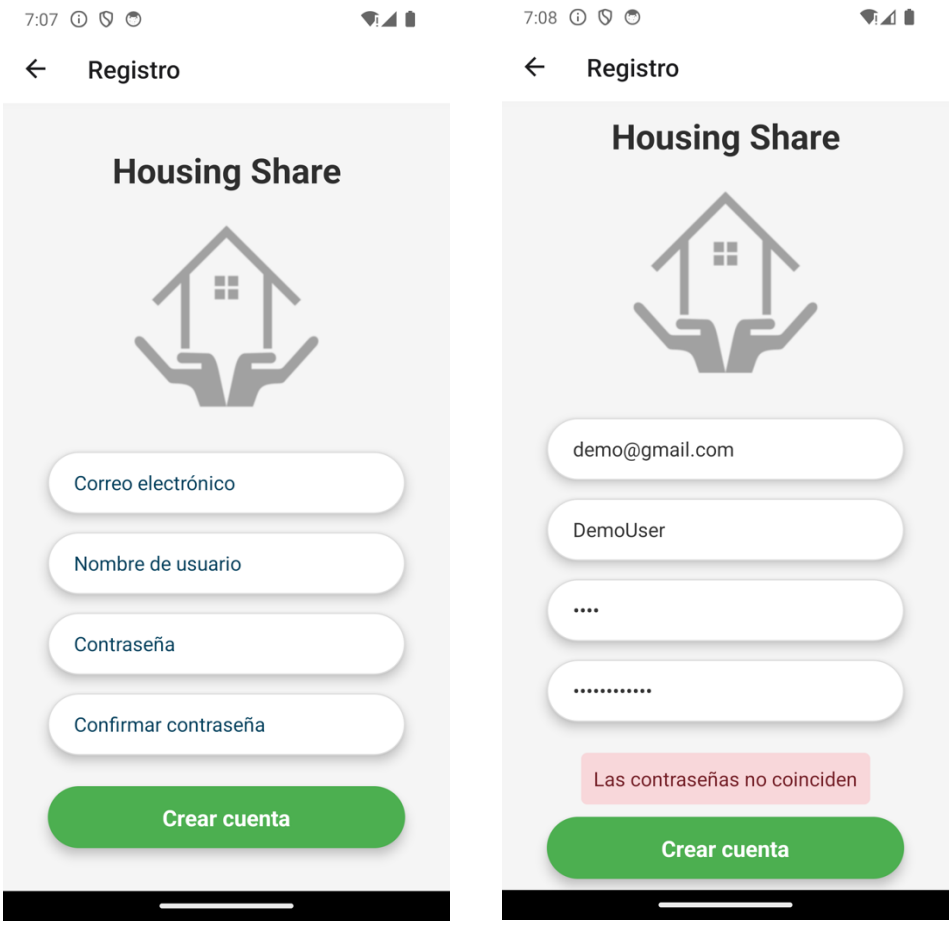
El presente manual pretende enumerar las posibilidades que brinda la aplicación desarrollada y cómo acceder a cada una de las funcionalidades. A continuación, se describen las principales funciones de la aplicación y cómo utilizarlas de manera efectiva.

1. Inicio de Sesión y Registro

Al abrir la aplicación, serás recibido por la pantalla de inicio de sesión. Aquí deberás ingresar tu correo electrónico y contraseña para acceder a tu cuenta. Si ya tienes una cuenta, introduce tus credenciales y haz clic en “Iniciar sesión”.



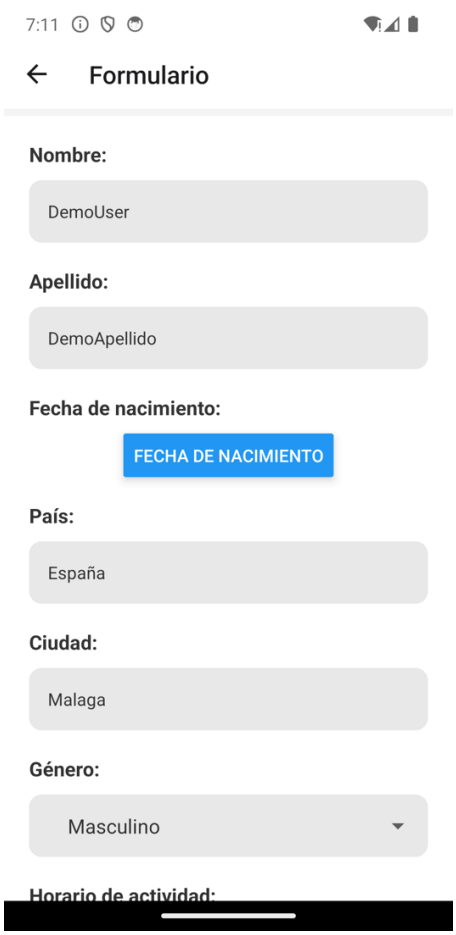
Si es la primera vez que utilizas la aplicación y no tienes una cuenta, selecciona la opción "Crear cuenta". Durante el registro, deberás ingresar tu correo electrónico, nombre de usuario y una contraseña segura. Si en el proceso introduces la confirmación de tu contraseña de manera errónea, no se te permitirá avanzar a la siguiente fase para registrar tu cuenta.

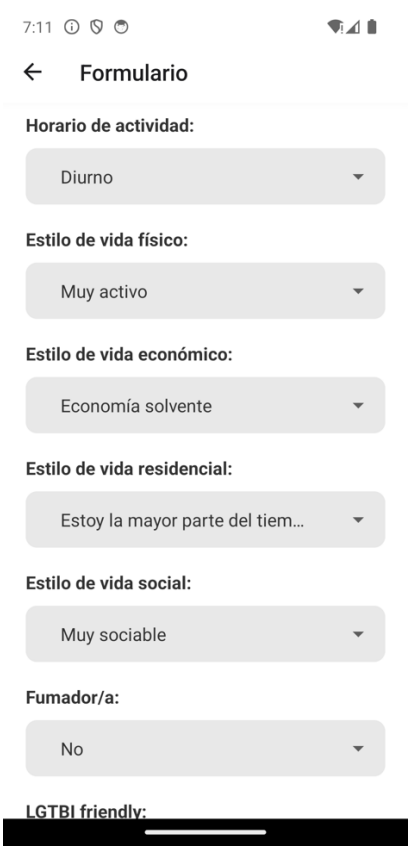


A continuación, se te pedirá que completes tu perfil inicial con información personal, como tu nombre, fecha de nacimiente, género y más información relevante para la aplicación, que incluyen aspectos como el estilo de vida y los hábitos de convivencia. Estos datos serán utilizadas para recomendarte posibles compañeros de piso.

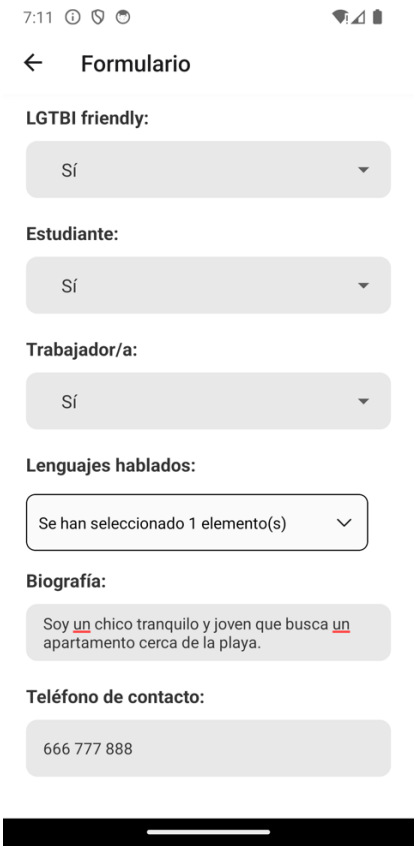

7:08
Bienvenido!
¡Bienvenido a Housing Share!
Antes de comenzar, tómate un momento para crear tu perfil.
Primero, te preguntaremos sobre tus datos personales. Luego, añadiremos un cuestionario para conocer tus preferencias respecto a tus futuros compañeros de piso.
CREAR MI PERFIL

-

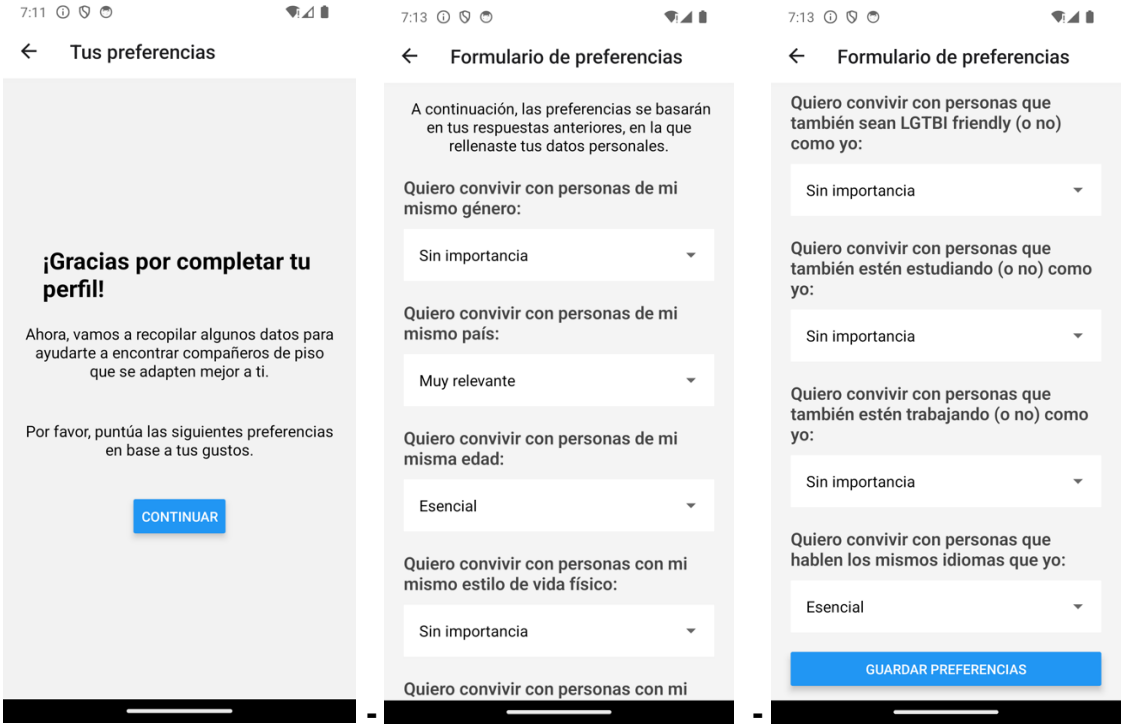

7:11
Formulario
Nombre:
DemoUser
Apellido:
DemoApellido
Fecha de nacimiento:
FECHA DE NACIMIENTO
País:
España
Ciudad:
Malaga
Género:
Masculino
Horario de actividad:


7:11
Formulario
Horario de actividad:
Diurno
Estilo de vida físico:
Muy activo
Estilo de vida económico:
Economía solvente
Estilo de vida residencial:
Estoy la mayor parte del tiem...
Estilo de vida social:
Muy sociable
Fumador/a:
No
LGTBI friendly:

-

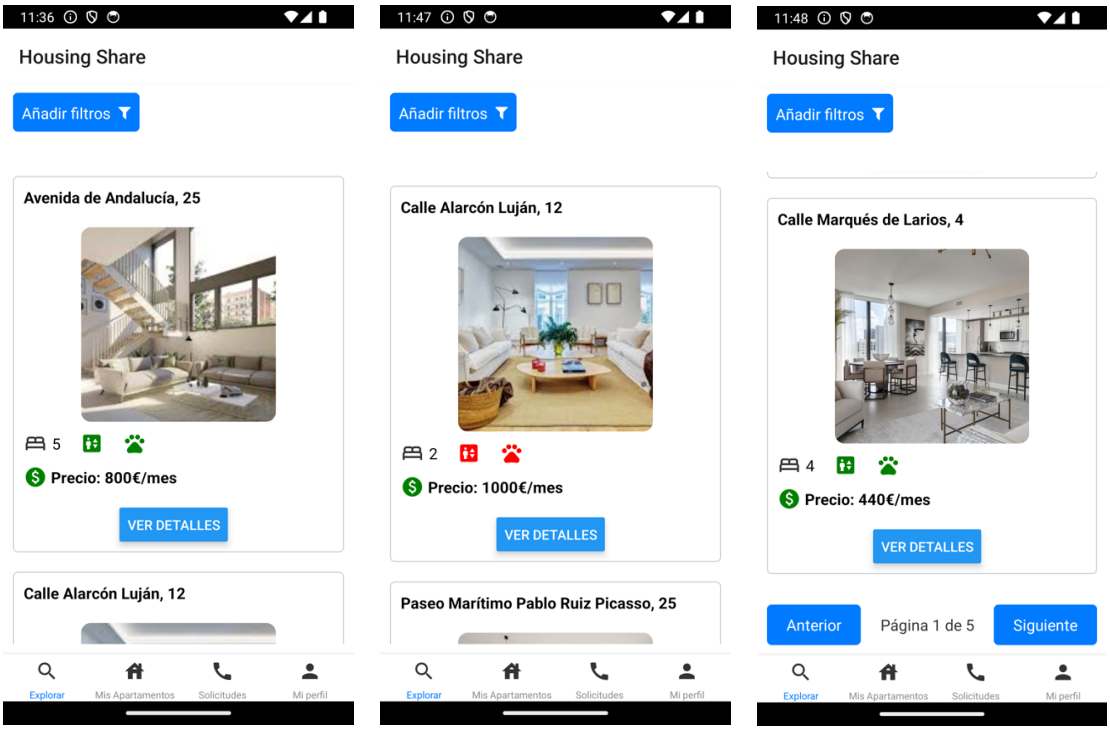

7:11
Formulario
LGTBI friendly:
Sí
Estudiante:
Sí
Trabajador/a:
Sí
Lenguajes hablados:
Se han seleccionado 1 elemento(s)
Biografía:
Soy un chico tranquilo y joven que busca un apartamento cerca de la playa.
Teléfono de contacto:
666 777 888

Por último, se te pedirá que rellenes una serie de preferencias de afinidad para que la aplicación sea capaz de recomendarte la gente más cercana a tus gustos, como por ejemplo cómo de importante es para ti convivir con persosas de tu mismo género o con gente de tu misma edad. Estos datos, que se combinan con tus datos personales, serán utilizados para recomendarte posibles compañeros de piso.

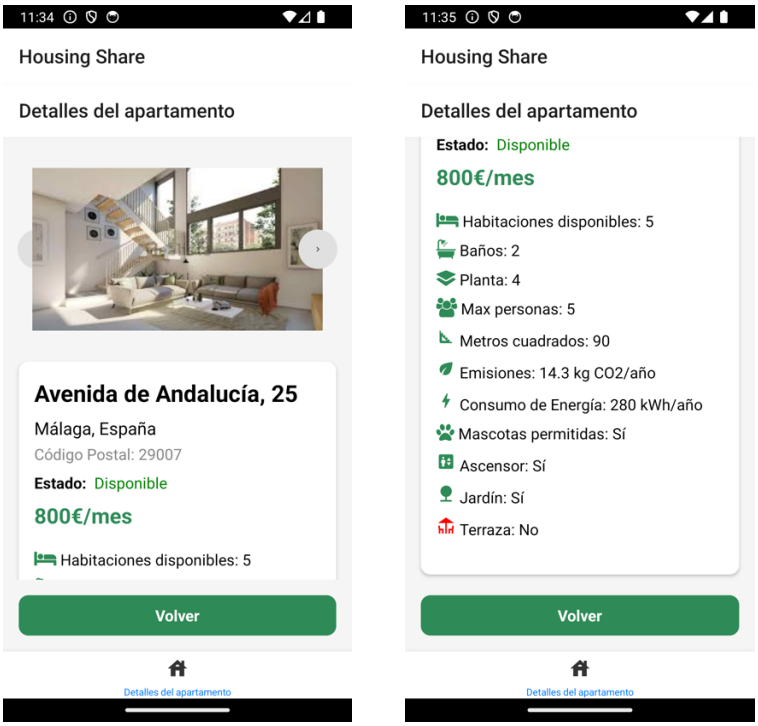


2. Explorar Apartamentos

Una vez que hayas iniciado sesión o te hayas registrado en la aplicación, serás dirigido a la pantalla principal de "Explorar". Aquí podrás ver una lista de apartamentos disponibles para alquiler compartido. Cada apartamento en la lista mostrará información básica como la ubicación, el precio y el número de habitaciones disponibles.

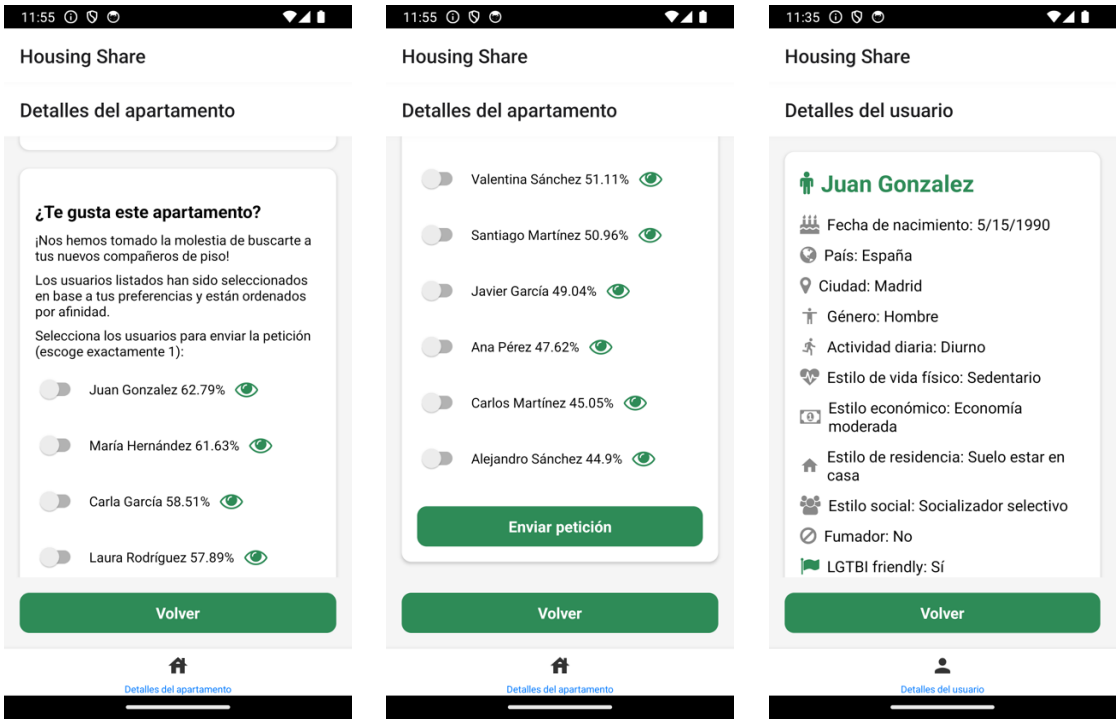


Para obtener más detalles sobre un apartamento específico, haz clic en “Mostrar detalles”. Esto te llevará a una nueva vista donde podrás ver fotos, una descripción completa del apartamento, y otras características relevantes. Esta vista te ayudará a evaluar si el apartamento cumple con tus necesidades y expectativas.



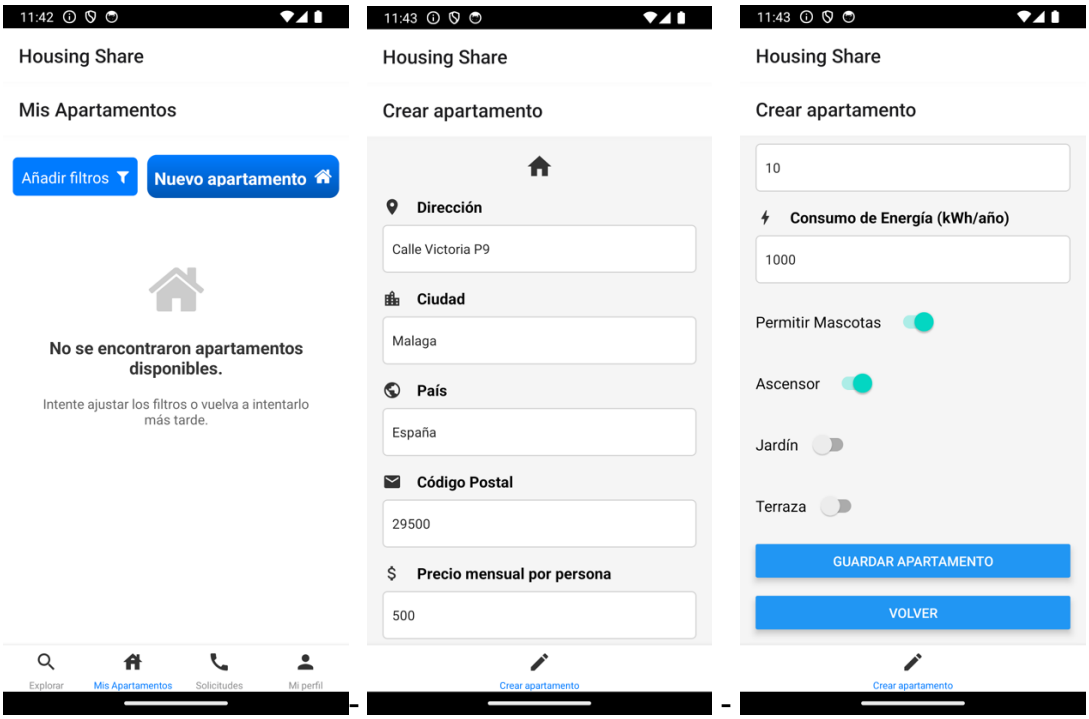
3. Recomendación de Compañeros de Piso

Debajo de los detalles del apartamento, la aplicación te sugerirá posibles compañeros de piso con los que podrías tener una alta afinidad. Estas recomendaciones se basan en las preferencias de afinidad que configuraste durante el registro y las comparará con las de otros usuarios. Al seleccionar un usuario recomendado, podrás ver más detalles sobre ellos y decidir si enviarles una solicitud para compartir el apartamento.

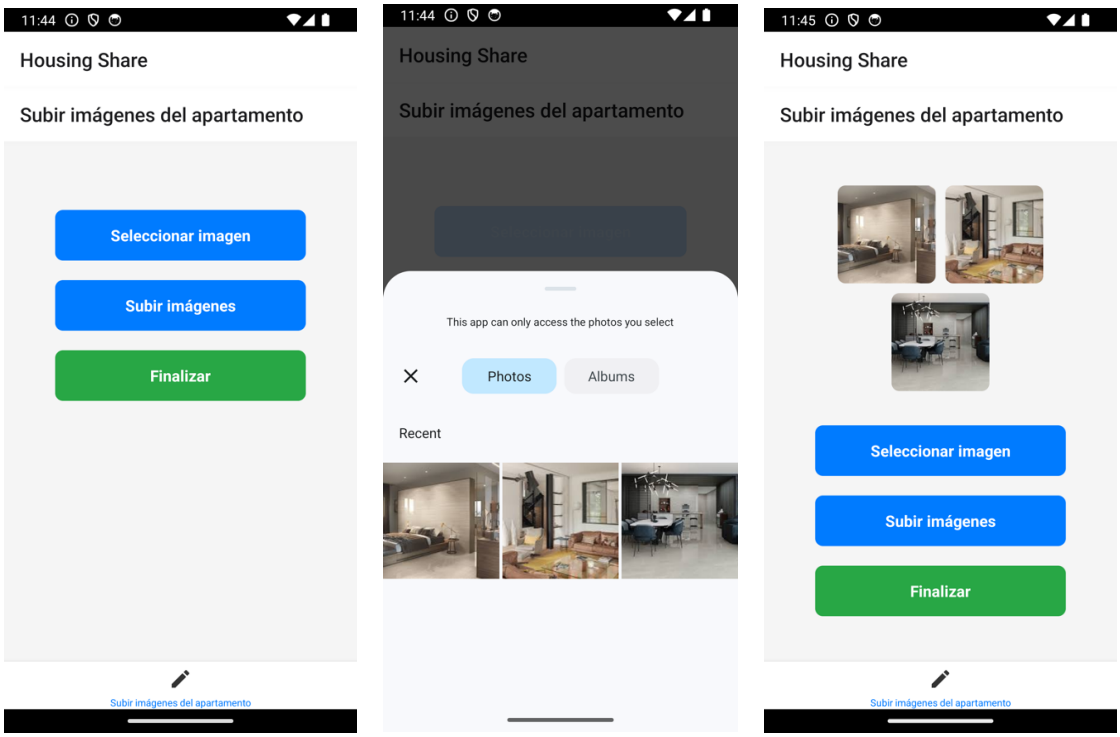


4. Gestión de Apartamentos

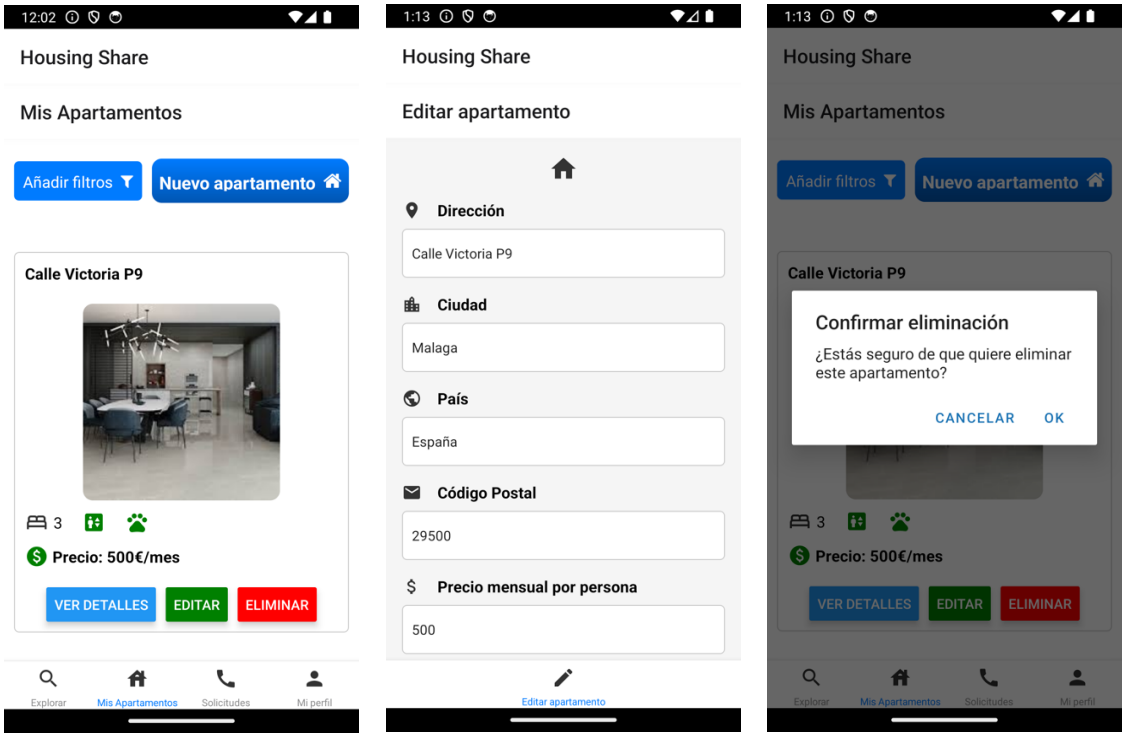
La sección "Mis Apartamentos" está diseñada para que puedas gestionar los apartamentos que has publicado. Si eres propietario o tienes un apartamento para compartir, puedes crear, editar o eliminar anuncios desde esta sección. Para añadir un nuevo apartamento, selecciona “Nuevo apartamento” y completa los campos solicitados, como la dirección, el precio y la descripción.



Luego, podrás subir imágenes del apartamento y recortarlas directamente en la aplicación para mejorar la presentación de tu anuncio.

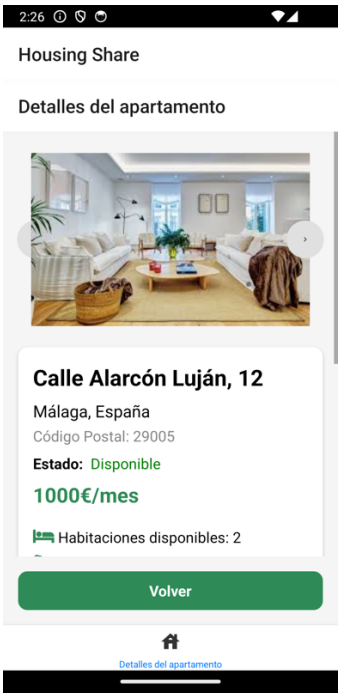


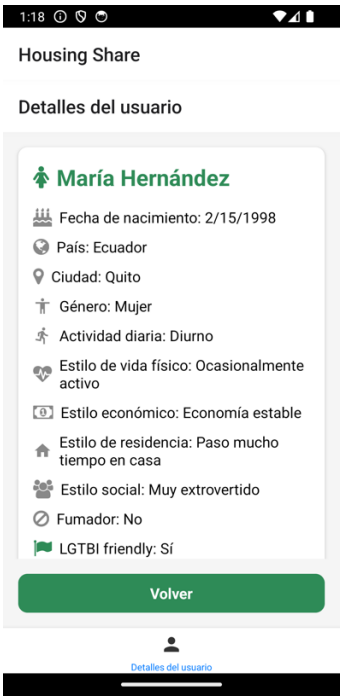
Una vez creado tu apartamento, podrás ver los detalles de tu apartamento, editarlo o eliminarlo en la pestaña de “Mis apartamentos”.



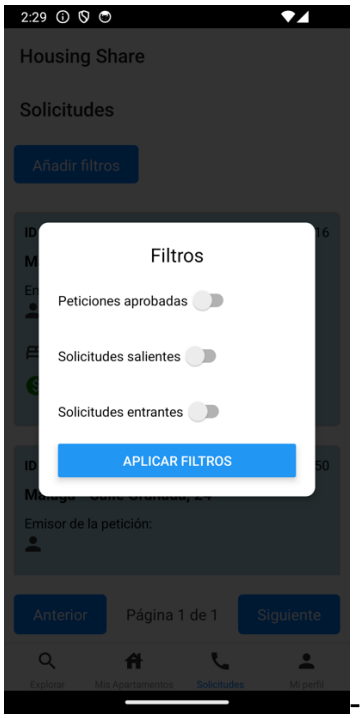
5. Gestión de Solicitudes

En la sección "Mis Peticiones", puedes ver y gestionar todas las solicitudes relacionadas con los apartamentos, tanto las que has enviado como las que has recibido. Aquí, podrás ver el estado de cada solicitud, los detalles de los usuarios implicados (haciendo click en cada uno de los iconos de personas) y la información del apartamento correspondiente (haciendo click en el icono del apartamento).

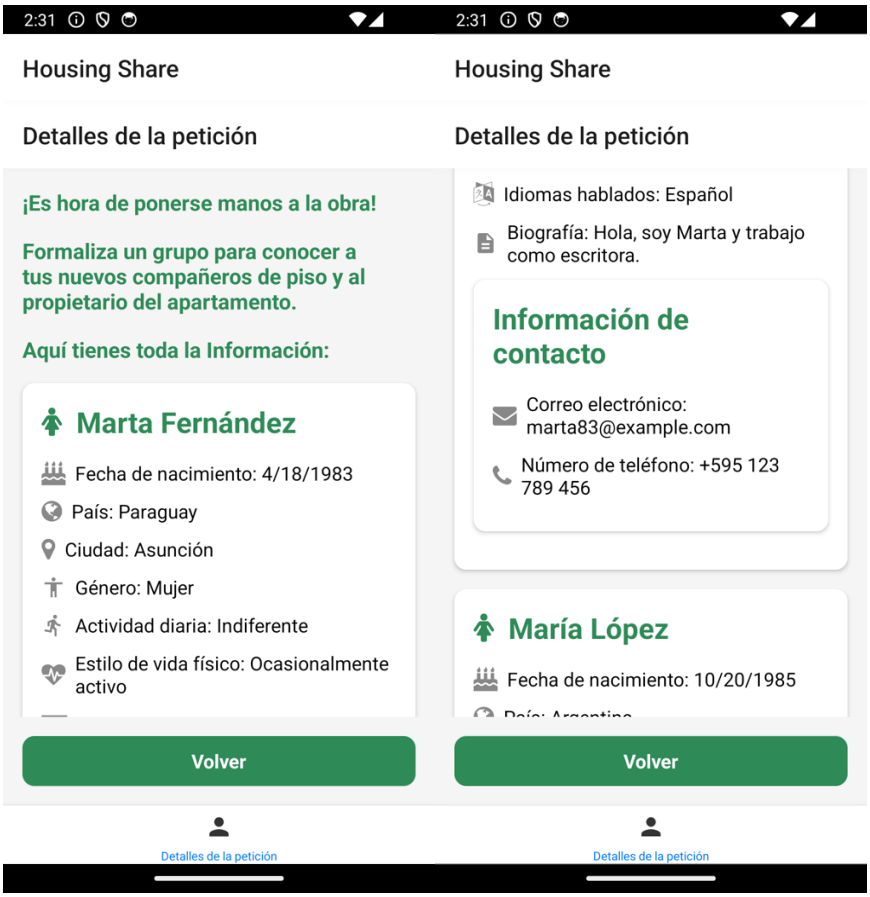




Desde esta vista, un usuario puede ver todas las solicitudes, tanto las entrantes (color azul) como las salientes (color verdes), pero no puede ver la información de contacto (email o teléfono) de ningún usuario implicado hasta que la petición sea aprobada. Además, el usuario tiene la opción de aplicar filtros para seleccionar las solicitudes que más le interesen.

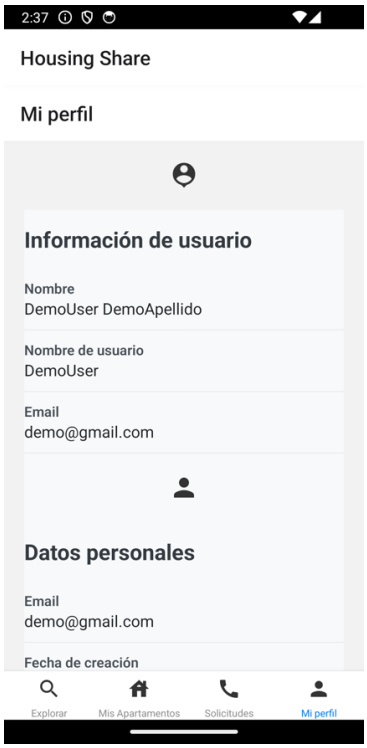


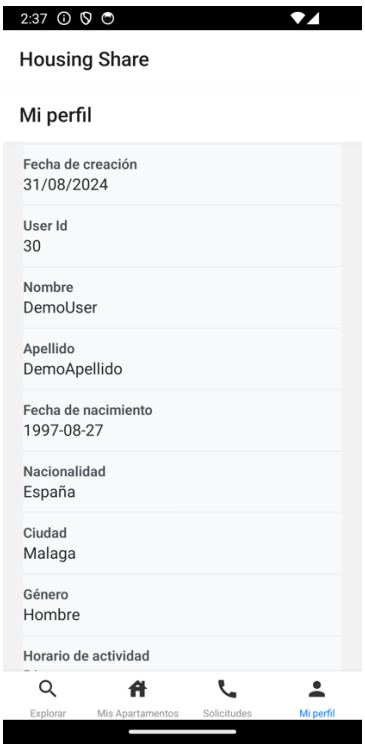
Los estados de las peticiones están marcados como “PENDIENTE” hasta que todos los usuarios implicados en una petición manden la misma solicitud de forma independiente. Solo en las peticiones marcadas como “APROBADO” se podrá ver toda la información de contacto tanto de los usuarios como del propietario del apartamento, para que así puedan procesar el alquier a partir del correo o email.



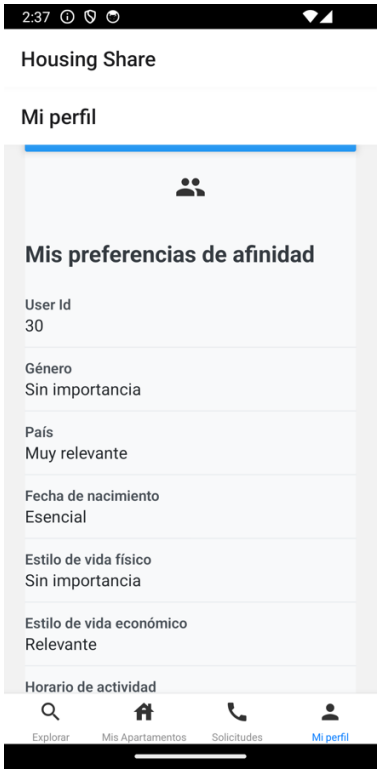
6. Gestión de Perfil y Preferencias

La sección "Mi Perfil" te permite actualizar y gestionar tu información personal. Puedes cambiar tu nombre, foto de perfil, y ajustar tus preferencias de afinidad, como si prefieres vivir con personas que fumen o no, el tipo de ambiente que buscas (tranquilo, social, etc.), y otros aspectos que son importantes para encontrar compañeros de piso compatibles. Mantener tu perfil actualizado es clave para recibir recomendaciones precisas y mejorar tu experiencia en la aplicación.


2:37
Housing Share
Mi perfil
Información de usuario
Nombre
DemoUser DemoApellido
Nombre de usuario
DemoUser
Email
demo@gmail.com
Datos personales
Email
demo@gmail.com
Fecha de creación
Explorar
Mis Apartamentos
Solicitudes
Mi perfil


2:37
Housing Share
Mi perfil
Fecha de creación
31/08/2024
User Id
30
Nombre
DemoUser
Apellido
DemoApellido
Fecha de nacimiento
1997-08-27
Nacionalidad
España
Ciudad
Malaga
Género
Hombre
Horario de actividad
Explorar
Mis Apartamentos
Solicitudes
Mi perfil


2:37
Housing Share
Mi perfil
Fumador
No
LGTBI friendly
Sí
Estudiante
Sí
Trabajador
Sí
Lenguajes hablados
Spanish
Biografía
Soy un chico tranquilo y joven que busca un apartamento cerca de la playa.
Número de teléfono
666 777 888
EDITAR MIS DATOS PERSONALES
Explorar
Mis Apartamentos
Solicitudes
Mi perfil


2:37
Housing Share
Mi perfil
Mis preferencias de afinidad
User Id
30
Género
Sin importancia
País
Muy relevante
Fecha de nacimiento
Esencial
Estilo de vida físico
Sin importancia
Estilo de vida económico
Relevante
Horario de actividad
Explorar
Mis Apartamentos
Solicitudes
Mi perfil


2:37
Housing Share
Mi perfil
Horario de actividad
Relevante
Estilo de vida residencial
Muy relevante
Estilo de vida social
Esencial
Fumador
Esencial
LGTBI friendly
Sin importancia
Estudiante
Sin importancia
Trabajador
Sin importancia
Lenguajes hablados
Esencial
EDITAR MIS PREFERENCIAS
Explorar
Mis Apartamentos
Solicitudes
Mi perfil

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga