Práctica Profesional Supervisada

Claros Maldonado, Brian Alejandro

MVP de sistema web para inmobiliarias

Instituto de Ingeniería y Agronomía

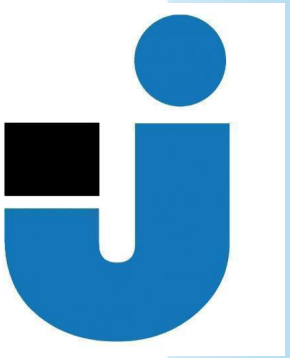
Carrera: Ingeniería en Informática

2025

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Carrera de Ingeniería en Informática



PRÁCTICA PROFESIONAL SUPERVISADA
Informe final

MVP de sistema web para inmobiliarias

Brian Alejandro, Claros Maldonado

Florencio Varela, Julio 2025

DATOS DEL ESTUDIANTE

Apellido y Nombres: Claros Maldonado, Brian Alejandro

DNI: 41150075

Nº de Legajo: 36799

Correo electrónico: brianclarosmaldonado@gmail.com

Cantidad de materias aprobadas al comienzo de la PPS: 44

PRÁCTICA PROFESIONAL SUPERVISADA enmarcada en el artículo 4 inciso 1.c de la Resolución (CS) 123/24

DOCENTE SUPERVISOR

Apellido y Nombres: Dr. Ing. Morales, Martin

Correo electrónico: martin.morales@unaj.edu.ar

DOCENTE TUTOR DEL TALLER DE APOYO A LA PRODUCCIÓN DE TEXTOS ACADÉMICOS DE LA UNAJ

Apellido y Nombres: Lic. Kelly, Carolina

Correo electrónico: kellygcarolina@gmail.com

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS

Nombre o Razón Social: COOPERATIVA DE TRABAJO DEVECOOP LTDA.

Dirección: Chile 1155, Ciudad Autónoma de Buenos Aires

Teléfono: 011 6841-2082

Sector: Desarrollo web

TUTOR ORGANIZACIONAL

Apellido y Nombres: Ing. Gonzalez, Federico

Correo electrónico: federico.gonzalez@devecoop.com

Resumen

El presente informe describe el desarrollo de un Producto Mínimo Viable (MVP) para una plataforma web orientada a inmobiliarias, con el objetivo de optimizar la gestión y publicación de propiedades en venta o alquiler mediante una solución tecnológica moderna, escalable y adaptable. Este trabajo se realizó en el marco de la Práctica Profesional Supervisada (PPS) del autor, en colaboración con la Cooperativa de Trabajo DEVECOOP Ltda.

La plataforma fue diseñada con una estructura modular y adaptable que permite a cada inmobiliaria personalizar aspectos visuales como el logotipo, la paleta de colores y los textos iniciales, dentro de una estructura general compuesta por secciones como Inicio, Información Institucional y Publicaciones de Propiedades. Entre las funcionalidades avanzadas se destaca un mapa interactivo de propiedades, un comparador basado en criterios clave (precio, ubicación y tamaño), un historial de precios por propiedad, y un calendario de turnos que permite coordinar visitas entre clientes y agentes inmobiliarios.

Desde el punto de vista técnico, el desarrollo se realizó utilizando Vue.js para el frontend, Django (con Django REST Framework) para el backend, y PostgreSQL como base de datos relacional. Se integraron servicios externos como Google Maps API para la geolocalización de propiedades y Firebase Cloud Messaging para notificaciones push en tiempo real. Además, se implementó un sistema automatizado de integración y despliegue continuo (CI/CD) mediante Docker y GitHub Actions, lo que permite crear instancias independientes del sistema para distintas inmobiliarias con mínimos ajustes técnicos.

El sistema incluye un back office desde donde las agencias pueden gestionar contenidos, administrar citas y usuarios, y personalizar la apariencia del sitio. Este desarrollo sentó las bases para futuras extensiones del producto, como la incorporación de diseño responsivo, pasarelas de pago y mejoras visuales. En conjunto, el MVP constituye una solución concreta y reutilizable para el sector inmobiliario, aportando valor real al cliente y experiencia profesional directa al estudiante.

Abstract

This report describes the development of a Minimum Viable Product (MVP) for a web platform aimed at real estate agencies, with the goal of optimizing the management and publication of properties for sale or rent through a modern, scalable, and adaptable technological solution. This project was carried out within the framework of the author's Supervised Professional Practice (PPS), in collaboration with the Cooperativa de Trabajo DEVECOOP Ltda.

The platform was designed with a modular and adaptable structure that allows each agency to customize visual elements such as the logo, color palette, and introductory texts, within a general layout composed of sections like Home, Institutional Information, and Property Listings. Among the advanced features are an interactive property map, a comparison tool based on key criteria (price, location, and size), a price history tracker for each property, and a scheduling calendar to coordinate visits between clients and real estate agents.

From a technical standpoint, the development was carried out using Vue.js for the frontend, Django (with Django REST Framework) for the backend, and PostgreSQL as the relational database. External services such as the Google Maps API for property geolocation and Firebase Cloud Messaging for real-time push notifications were integrated. Additionally, an automated continuous integration and deployment (CI/CD) system was implemented using Docker and GitHub Actions, allowing the creation of independent system instances for different agencies with minimal technical adjustments.

The system includes a back office from which agencies can manage content, appointments, users, and customize the appearance of their website. This development laid the foundation for future extensions of the product, such as the incorporation of responsive design, payment gateways, and visual enhancements. Overall, the MVP constitutes a concrete and reusable solution for the real estate sector, delivering real value to clients and hands-on professional experience to the student.

Dedicatorias y agradecimientos

Quiero agradecer profundamente a mi familia y a mi novia por su acompañamiento incondicional durante esta etapa final. Gracias por estar en los momentos de mayor duda, por insistir cuando me sentía estancado y por motivarme a seguir adelante hasta alcanzar esta meta.

A mis padres, gracias por la educación que me brindaron desde siempre, por enseñarme el valor del esfuerzo, la constancia y por transmitirme con el ejemplo que nunca hay que rendirse. Esta base fue fundamental para llegar hasta acá.

A mi novia, gracias por tu apoyo constante, por impulsarme a mejorar y acompañarme siempre que tenía que quedarme hasta tarde.

También quiero expresar mi agradecimiento a los docentes de la Universidad Nacional Arturo Jauretche, por su dedicación y compromiso. Siempre estuvieron dispuestos a brindar apoyo académico y humano, contribuyendo de forma clave a mi formación profesional.

Finalmente, a todos mis compañeros y compañeras con quienes compartí cursadas, proyectos, exámenes, alegrías y frustraciones: gracias por su compañía, por la colaboración mutua y por haber hecho de este camino una experiencia más llevadera.

¡Gracias!

Índice

Índice de figuras

1. Introducción

En un contexto cada vez más digitalizado y competitivo, el sector inmobiliario enfrenta múltiples desafíos relacionados con la gestión y promoción de propiedades. Muchas inmobiliarias continúan utilizando herramientas tradicionales o procesos manuales, lo que limita su eficiencia operativa y la experiencia de sus clientes. Ante esta problemática, surge la necesidad de contar con soluciones tecnológicas integrales, adaptables y de fácil implementación que permitan centralizar operaciones y mejorar la visibilidad online.

Este proyecto fue desarrollado en el marco de la Práctica Profesional Supervisada (PPS) del autor, como respuesta a una necesidad de un Cliente la Cooperativa DEVECOOP Ltda., que buscaba diseñar y comercializar una plataforma web orientada a la gestión inmobiliaria, capaz de ser replicada por múltiples clientes sin necesidad de desarrollos independientes.

Como solución, se desarrolló un Producto Mínimo Viable (MVP) de una plataforma web, personalizable y escalable, enfocada en permitir a las inmobiliarias gestionar y publicar propiedades en venta o alquiler. Entre sus funcionalidades principales se incluye la personalización visual institucional, visualización de propiedades en un mapa interactivo, herramienta de comparación por criterios clave, historial de precios por inmueble, agenda de visitas programadas y notificaciones en tiempo real. Entre las funcionalidades inicialmente previstas, se consideró un sistema de automatización de publicaciones en redes sociales. No obstante, esta característica fue reprogramada para una fase futura, fuera del alcance del MVP desarrollado en esta etapa.

La solución fue implementada con tecnologías modernas: Vue.js en el frontend, Python, Django y Django REST Framework en el backend, y PostgreSQL como base de datos. El sistema adaptó una arquitectura multi-tenant, permitiendo que múltiples inmobiliarias pudieran compartir una misma instancia con aislamiento lógico. Asimismo, se integraron servicios externos como Google Maps API y Firebase Cloud Messaging, y se automatizó el proceso de integración y despliegue continuo (CI/CD) mediante GitHub Actions y Docker.

Es importante destacar que, dado que se trató de un MVP, el diseño responsivo completo fue planificado para una fase posterior, priorizando en esta etapa la validación funcional del sistema en entornos de escritorio.

Este informe documenta en detalle todas las fases del desarrollo desde la identificación de requerimientos y diseño de la arquitectura, hasta la implementación técnica, pruebas, despliegue, evaluación del impacto y proyecciones futuras del sistema.

2. Marco teórico y fundamentación tecnológica

El proyecto se desarrolló bajo el enfoque metodológico de Producto Mínimo Viable (MVP), concepto introducido por Eric Ries en el marco del movimiento Lean Startup en 2008. Un MVP permite validar hipótesis de valor con la menor cantidad de desarrollo posible, priorizando funcionalidades esenciales que puedan ser testeadas por usuarios reales. Esta estrategia permite acelerar la retroalimentación, reducir costos iniciales y fomentar una evolución iterativa del producto.

Desde el punto de vista arquitectónico, se optó por un modelo multi-tenant, lo cual resulta particularmente adecuado para soluciones SaaS. Esta arquitectura permite que distintas inmobiliarias puedan operar sobre una única base de código, con configuraciones visuales y lógicas independientes, favoreciendo la reutilización, el mantenimiento centralizado y la escalabilidad horizontal.

El backend del sistema fue desarrollado utilizando Python, Django y su extensión Django REST Framework (DRF). Esta elección responde a la madurez del ecosistema Python, la seguridad y escalabilidad de Django, y la facilidad con la que DRF permite construir APIs RESTful con autenticación, control de permisos, serialización eficiente de datos y soporte completo para operaciones CRUD. Se implementó autenticación mediante tokens JWT para una gestión segura de sesiones.

La base de datos seleccionada fue PostgreSQL, un sistema relacional de código abierto robusto, que garantiza transacciones ACID, permite replicación y ofrece extensiones avanzadas como PostGIS para operaciones geoespaciales, útiles en la gestión de propiedades geolocalizadas. Su integración nativa con el ORM de Django permite consultas eficientes, migraciones estructuradas y validación a nivel de modelo.

El frontend fue desarrollado en Vue.js, un framework progresivo basado en componentes, ideal para construir interfaces tipo SPA (Single Page Applications, aplicaciones de página única). Vue ofrece una curva de aprendizaje accesible, alto rendimiento, y un ecosistema maduro que permite dividir la lógica visual en componentes reutilizables y desacoplados. Esta decisión favorece la escalabilidad del frontend y permite que el backend y frontend evolucionen de forma independiente.

El sistema fue desarrollado bajo un enfoque full stack desacoplado, con comunicación entre cliente y servidor exclusivamente a través de una API REST. Esta separación de capas facilita el mantenimiento modular, la escalabilidad futura y la posibilidad de migrar a otras interfaces (como apps móviles o paneles administrativos externos) sin modificar la lógica del backend.

Para garantizar entornos de ejecución homogéneos y facilitar la automatización, se utilizó Docker como tecnología de contenedorización. A su vez, se configuró un pipeline de CI/CD con GitHub Actions, que automatiza tareas como testeo, construcción de imágenes y despliegue en producción.
En conjunto, esta selección de herramientas y enfoques tecnológicos permitió construir una plataforma robusta, mantenible y preparada para su evolución futura, alineada con los estándares actuales de desarrollo profesional en ingeniería de software.

3. Objetivos

3.1. Objetivo general

Desarrollar un Producto Mínimo Viable (MVP) de una plataforma web orientada a inmobiliarias, basada en una arquitectura multi-tenant, que permita gestionar, publicar y promocionar propiedades en venta o alquiler de manera eficiente, escalable y personalizable. La solución será implementada utilizando tecnologías modernas como Vue.js, Django REST Framework, PostgreSQL, Docker y GitHub Actions, y se desarrollará bajo metodologías ágiles. Se busca entregar un sistema funcional, replicable y de fácil implementación técnica, que permita validar el modelo de producto en un entorno real con múltiples clientes.

3.2. Objetivos de proyecto

3.2.1. Arquitectura, backend y lógica del sistema

- Diseñar e implementar una arquitectura desacoplada, basada en API REST y modelo multi-tenant, que garantice el aislamiento lógico de los datos de cada cliente dentro de una misma instancia del sistema.
- Implementar una base de datos relacional (PostgreSQL), estructurada y optimizada para soportar operaciones complejas y escalables, incluyendo relaciones entre usuarios, propiedades, imágenes y citas.
- Desarrollar el backend con Django y Django REST Framework, utilizando vistas basadas en clases, serializers, autenticación con JWT, y control de permisos, aplicando principios de seguridad y buenas prácticas.
- Documentar la API REST mediante Swagger para asegurar su comprensión, uso y futura extensión.

3.2.2. Funcionalidades clave y valor para el usuario

- Permitir a las inmobiliarias gestionar propiedades, incluyendo múltiples imágenes, descripción, ubicación, precio, tamaño y otras características relevantes.
- Incorporar funcionalidades avanzadas orientadas a la experiencia del usuario como mapa interactivo (Google Maps API), comparador de propiedades por criterios clave, historial de precios y calendario de visitas programadas.
- Planificar la futura automatización de publicaciones en redes sociales como Facebook e Instagram, funcionalidad que fue identificada como deseable por el cliente, pero postergada para una próxima iteración fuera del alcance del MVP actual.
- Implementar un sistema de notificaciones push utilizando Firebase Cloud Messaging, para alertar a los usuarios sobre cambios o nuevas publicaciones relevantes.

3.2.3. Personalización y administración

- Desarrollar un back office que permita a cada inmobiliaria personalizar visualmente su sitio (colores, logo, textos institucionales) dentro de una estructura modular previamente diseñada.
- Incorporar un sistema de gestión de roles (administrador, agente, visitante), asegurando una experiencia diferenciada por perfil.

3.2.4. Automatización y despliegue

- Contenerizar los entornos del sistema (frontend, backend y base de datos) con Docker para asegurar entornos reproducibles.
- Automatizar el proceso de integración y despliegue continuo (CI/CD) con GitHub Actions, incluyendo ejecución de tests, generación de imágenes y despliegue en producción.
- Facilitar el despliegue de nuevas instancias del sistema para distintos clientes mediante configuración dinámica, sin necesidad de intervención manual intensiva.

3.2.5. Calidad del software y validación

- Implementar pruebas unitarias y de integración sobre la API REST para asegurar una base sólida del backend y detectar errores en etapas tempranas del desarrollo.
- Validar internamente el sistema y con el cliente real, recogiendo feedback que sirvió como base para planificar futuras mejoras.

3.2.6. Proyectar escalabilidad y evolución futura

- Diseñar la solución con principios de modularidad y extensibilidad, permitiendo escalar vertical y horizontalmente el sistema.
- Establecer las bases técnicas para incluir en fases posteriores un diseño totalmente responsivo, integración con pasarelas de pago y mejoras visuales orientadas al usuario final.

3.3. Objetivos personales

3.3.1. Técnicos y profesionales

- Fortalecer mis competencias en desarrollo full stack con tecnologías modernas como Vue.js, Django REST Framework y PostgreSQL, aplicadas en un entorno real de trabajo.
- Adquirir experiencia concreta en integración de servicios externos (Google Maps API, Firebase) y su configuración dentro de un sistema modular y mantenible.
- Incorporar herramientas profesionales para documentación técnica, incluyendo Swagger para API REST y Notion como soporte organizacional.
- Dominar herramientas de automatización del ciclo de vida del software (Docker, GitHub Actions), implementando pipelines CI/CD efectivos.

3.3.2. Organizacionales y metodológicos

- Aplicar metodologías ágiles en un proyecto real mediante un enfoque Scrumban, utilizando Trello para gestión de tareas y organización de flujos de trabajo en equipo.
- Participar activamente en la planificación y seguimiento del cronograma del proyecto, asegurando cumplimiento de los tiempos, definición de prioridades y entregables funcionales.

3.3.3. Liderazgo, comunicación y experiencia profesional

- Asumir un rol técnico central en el proyecto, participando en todas las decisiones clave de diseño, arquitectura e implementación.
- Ejercitar habilidades de comunicación efectiva con el cliente, entendiendo sus prioridades, y ajustando el desarrollo en función de sus necesidades.
- Priorizar funcionalidades bajo criterios de valor para el usuario y viabilidad técnica, practicando una toma de decisiones informada y profesional.

- Integrar la práctica académica con un entorno productivo real, desarrollando una solución concreta para un cliente y afianzando mi perfil profesional como futuro ingeniero en informática.

4. Tareas a ejecutar

Para alcanzar el desarrollo del Producto Mínimo Viable (MVP) de la plataforma web para inmobiliarias, se definió un conjunto de tareas agrupadas por áreas funcionales del sistema. Estas tareas fueron planificadas para validar las funcionalidades esenciales del sistema, siguiendo un enfoque iterativo, técnico y centrado en el valor funcional, sin profundizar en aspectos visuales avanzados ni escalamiento masivo.

Las tareas fueron ejecutadas en el marco de una metodología ágil híbrida (Scrumban), con seguimiento continuo a través de tableros de trabajo y entregas funcionales parciales validadas por el cliente en instancias de demostración mensuales. La planificación cronológica y la distribución de estas tareas se detallan en la sección siguiente (Diagrama de Gantt). A continuación, se presentan las tareas a ejecutar definidas para esta primera etapa del proyecto.

4.1. Análisis y planificación

- Relevamiento de requisitos funcionales y no funcionales junto al cliente.
- Análisis de plataformas inmobiliarias existentes para identificar buenas prácticas y funcionalidades claves a priorizar.
- Delimitación del alcance del MVP, excluyendo funcionalidades de etapas posteriores como el diseño responsivo avanzado o pasarelas de pago.
- Definición de la arquitectura general del sistema (modelo desacoplado multi-tenant).
- Selección de tecnologías y herramientas base para cada componente.
- Planificación técnica general y definición del flujo de trabajo iterativo.

4.2. Desarrollo del backend

- Configuración del entorno de desarrollo (Python, Django, PostgreSQL).
- Modelado de datos y relaciones: usuarios, propiedades, imágenes, citas, historial de precios, etc.
- Implementación de la API RESTful con autenticación JWT, serializers y permisos diferenciados por rol.
- Integración de Google Maps API para geolocalización.
- Integración de Firebase Cloud Messaging para envío de notificaciones push.

- Envío de emails a las casillas de correo de los clientes.
- Implementación de pruebas unitarias y de integración en endpoints críticos.
- Documentación de la API mediante Swagger.

4.3. Desarrollo del frontend

- Configuración del entorno de trabajo con Vue.js.
- Implementación de estructura general del sitio: navegación, rutas, vistas funcionales.
- Desarrollo de componentes dinámicos: mapa interactivo, filtros de búsqueda, comparador, historial de precios y calendario de visitas.
- Integración completa con la API REST para visualización y gestión de contenidos.
- Personalización visual por inmobiliaria (logo, paleta de colores, textos institucionales).
- Diseño visual funcional basado en referencias de plataformas existentes, adaptado a un enfoque de validación temprana del MVP.

4.4. Automatización, despliegue e infraestructura

- Contenerización del sistema completo con Docker (backend, frontend, base de datos).
- Implementación de flujos de integración y despliegue continuo (CI/CD) con GitHub Actions.
- Configuración de infraestructura de hosting en VPS (DigitalOcean).
- Aseguramiento del entorno de producción: variables de entorno, puertos, dominios, servicios de base de datos, firewall.
- Configuración básica de monitoreo, registros y validación de logs del sistema en contexto MVP.

4.5. Validación funcional y comunicación con el cliente

- Presentación mensual de avances funcionales mediante instancias de demostración al cliente.
- Recepción de feedback e incorporación de ajustes menores.
- Validación final de funcionalidades clave del MVP antes del cierre de etapa.

4.6. Documentación y entrega

- Redacción de documentación técnica del sistema.
- Elaboración de documentación de uso del MVP dirigida al cliente.

- Organización de recursos de documentación y notas técnicas en Notion.
- Redacción del informe académico final y entrega del proyecto completo.

La ejecución de estas tareas no sólo permitió alcanzar los objetivos funcionales del MVP, sino que constituyó un ejercicio integral de aplicación profesional. A lo largo del proceso, se pusieron en práctica conocimientos clave adquiridos durante la carrera, incluyendo diseño y arquitectura de software, desarrollo backend y frontend, automatización de procesos, integración de servicios externos y documentación técnica. La planificación, implementación y validación de estas tareas en un entorno real aportaron una experiencia valiosa, formativa y alineada con las competencias propias de un ingeniero en informática.

5. Diagrama de Gantt

El desarrollo del Producto Mínimo Viable (MVP) se organizó en cinco fases técnicas que fueron planificadas y ejecutadas de forma estructurada, siguiendo una estrategia de desarrollo progresivo y validación incremental. El siguiente diagrama representa gráficamente la planificación, duración y secuencia de las tareas del proyecto durante el desarrollo del MVP.

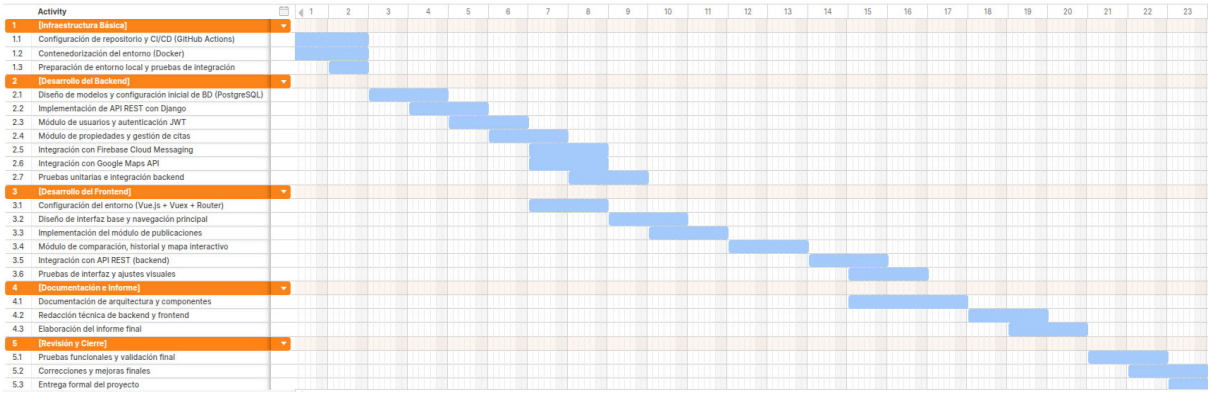


Figura 1. Diagrama de Gantt.

Fuente: Elaboración propia, basada en la práctica.

El Gantt fue concebido no sólo como una herramienta de planificación inicial, sino como un instrumento de gestión activa del proyecto, que permitió organizar prioridades, anticipar problemas, asignar recursos y facilitar la toma de decisiones en contextos reales de trabajo. Su estructura flexible habilitó ajustes en función de imprevistos y disponibilidad del equipo, manteniendo siempre los objetivos funcionales definidos.

El cronograma se dividió en cinco fases:

- Infraestructura básica: incluyó la configuración de integración continua (CI/CD) mediante GitHub Actions, la contenerización del sistema con Docker, y la preparación del entorno de desarrollo común.
- Desarrollo del backend: se abordó el diseño de modelos, la implementación de la API REST, la autenticación, la gestión de propiedades y citas, la integración con servicios externos y validación técnica mediante pruebas unitarias e integración.
- Desarrollo del frontend: se trabajó sobre una versión inicial no responsiva con Vue.js, enfocada en validar la interfaz y la comunicación con el backend a través de funcionalidades clave como filtros, mapa interactivo, comparador y calendario de visitas.
- Documentación técnica e informe final: abarcó la redacción del informe académico, documentación de la API (Swagger), y sistematización del conocimiento del sistema en herramientas colaborativas como Notion.
- Revisión, pruebas funcionales y entrega: incluyó la validación integral del MVP, ajustes finales a partir del feedback del cliente, y la entrega de una versión funcional cerrada, con posibilidad de expansión futura.

El proyecto fue ejecutado por dos desarrolladores con perfil semi senior. El autor del informe asumió la coordinación técnica general, liderando la planificación y supervisión de tareas, especialmente durante las semanas en las que el otro integrante del equipo estuvo ausente (semanas 18 a 20). En ese período se sostuvieron sin inconvenientes tareas clave como la documentación y la validación funcional.

Cabe destacar que el cronograma fue diseñado con márgenes de tolerancia en fases críticas como las integraciones externas y la redacción del informe, lo cual permitió absorber desviaciones sin comprometer los objetivos. Esta decisión estratégica, tomada desde un enfoque ingenieril, favoreció el cumplimiento de entregables, la detección temprana de riesgos y la optimización del tiempo disponible.

En conclusión, la utilización del diagrama de Gantt como herramienta de gestión permitió estructurar el trabajo en fases coherentes, asignar prioridades, visualizar dependencias y mantener una ejecución profesional del proyecto en sus distintas etapas. Su uso aportó claridad al equipo, orden al proceso de desarrollo y trazabilidad a cada avance técnico alcanzado.

6. Análisis

El análisis inicial del sistema constituyó una etapa fundamental para establecer las bases funcionales, técnicas y estratégicas del proyecto. A través del relevamiento de requerimientos, la delimitación del alcance, el diseño preliminar de la arquitectura y la gestión de riesgos, se definieron con precisión los componentes del Producto Mínimo Viable (MVP), así como las decisiones necesarias para llevar adelante su desarrollo de manera eficiente, estructurada y alineada con los objetivos reales del cliente.

Esta etapa permitió alinear las expectativas del proyecto con las posibilidades técnicas disponibles, priorizar funcionalidades críticas, anticipar desafíos y diseñar una solución que, aun en su primera versión, pueda ofrecer valor real y reutilizable para múltiples clientes. A continuación, se detalla el proceso llevado a cabo durante la fase de análisis.

6.1. Relevamiento de requisitos

Con el objetivo de identificar las necesidades operativas y funcionales del sistema, se llevó a cabo un proceso de relevamiento que incluyó entrevistas con el cliente responsable del producto, quien actuó como referente funcional durante todo el proyecto. Estas entrevistas permitieron obtener información sobre el flujo de trabajo habitual en inmobiliarias, las herramientas que se utilizan actualmente y las principales deficiencias que se busca resolver.

En paralelo, se realizó un análisis de referencia sobre plataformas inmobiliarias existentes en el mercado, tanto nacionales como internacionales, lo que permitió identificar funcionalidades comunes, características deseables y estándares actuales de experiencia de usuario. Esta combinación entre análisis competitivo y contacto directo con el cliente permitió elaborar una lista de requisitos iniciales clasificados como funcionales y no funcionales.

La priorización de los requisitos se realizó considerando tres factores principales: valor funcional para el cliente, viabilidad técnica para su implementación en el contexto de un MVP, y dependencia técnica entre funcionalidades.

6.2. Análisis del usuario final y perfiles de uso

Durante el relevamiento funcional se definieron distintos perfiles de usuario que interactúan con la plataforma, cuyas necesidades guiaron el diseño funcional y visual del sistema. Comprender estos perfiles permite alinear el desarrollo técnico con una experiencia de uso coherente, intuitiva y centrada en los objetivos de cada tipo de actor.

6.2.1. Usuario visitante (cliente potencial)

- Interactúa con el sitio sin registrarse.
- Utiliza filtros, mapa interactivo y comparador para encontrar propiedades.
- Puede solicitar visitas o dejar sus datos de contacto.
- Puede registrarse.
- El usuario registrado puede agregar propiedades a favoritos para recibir notificaciones push y emails sobre cambios sobre los mismos.
- Puede comparar las características de diferentes propiedades.
- Busca inmediatez, claridad visual y navegación fluida.

Diseño orientado: interfaz clara, filtros destacados, carga rápida, botón de contacto visible.

6.2.2. Agente inmobiliario

- Acceder al panel de administración.
- Operar como usuario autenticado con permisos intermedios.
- Gestiona propiedades (crear, editar, eliminar).
- Programa visitas y responde consultas.
- Necesita una interfaz rápida, confiable y con formularios simples.

Diseño orientado: acceso directo al listado de propiedades propias, agenda de visitas, validación de campos y carga de imágenes.

6.2.3. Administrador de inmobiliaria

- Tiene control completo del sitio.
- Personaliza el logo, colores, textos institucionales.
- Supervisa publicaciones, agenda y usuarios.

Diseño orientado: panel de configuración visual con vista previa, estructura jerárquica de accesos, experiencia fluida sin necesidad de conocimientos técnicos.

6.2.4. Administrador de sistema

- Tiene control completo de todo el sistema.
- Puede dar de alta nuevas inmobiliarias.
- Puede dar de baja inmobiliarias.

Diseño orientado: panel de configuración para el alta, baja o modificación de inmobiliarias.

6.2.5. Usuario desarrollador

- Mantener y extender el sistema, asegurando su correcto funcionamiento y la incorporación de nuevas funcionalidades.
- Necesita documentación clara y endpoints accesibles.

Diseño orientado: documentación Swagger, arquitectura modular y autenticación JWT para integraciones futuras.

6.3. Requisitos funcionales

- Gestión de propiedades: creación, edición y eliminación de propiedades, con carga de múltiples imágenes, ubicación, precio, superficie y características relevantes.
- Publicación de propiedades en una interfaz web accesible para clientes con filtros por ubicación, tipo y precio.
- Visualización de propiedades en un mapa interactivo con geolocalización.
- Formulario de contacto.
- Comparación de propiedades por criterios como precio, tamaño y ubicación.
- Visualización del historial de precios de cada propiedad.
- Coordinación de visitas mediante un calendario de turnos disponibles.
- Notificaciones push para informar a los usuarios sobre cambios en precios en publicaciones.
- Panel de administración (back office) para inmobiliarias: carga de propiedades, personalización visual del sitio (logo, colores, textos), gestión de publicaciones.
- Gestión de usuarios y roles con autenticación segura: administrador, agente inmobiliario y visitante.

6.4. Requisitos no funcionales

- Escalabilidad: soporte para múltiples inmobiliarias sin necesidad de replicar el sistema.
- Seguridad: separación lógica de datos por cliente (modelo multi-tenant), uso de JWT para autenticación.
- Mantenibilidad: arquitectura modular y desacoplada, código legible y documentado.
- Portabilidad: sistema ejecutable en entornos heterogéneos gracias al uso de contenedores Docker.
- Automatización del despliegue: uso de flujos CI/CD con GitHub Actions.
- Compatibilidad: acceso desde navegadores modernos en entornos de escritorio.
- Usabilidad: diseño funcional centrado en la validación del MVP.

6.5. Definición del alcance

Una vez definidos los requerimientos, se estableció el alcance específico del proyecto para esta primera etapa. El desarrollo se limitó al diseño y entrega de un MVP funcional, centrado en la validación de las funcionalidades básicas por parte del cliente. El objetivo fue entregar una versión operativa, estable y reproducible del sistema, que sirviera como base para futuras iteraciones y mejoras.

Quedaron fuera del alcance del MVP aquellas funcionalidades asociadas a etapas más avanzadas del proyecto, tales como:

- Diseño completamente responsivo para dispositivos móviles y tablets.
- Integración de pasarelas de pago.
- Personalización avanzada de contenidos o estructuras visuales.
- Implementaciones específicas por cliente fuera de la configuración visual básica.
- Módulos de analítica o reportes internos.
- Automatización de las publicaciones en redes sociales de la inmobiliaria.

Esta delimitación permitió enfocar los recursos disponibles en aquellas funcionalidades que ofrecían mayor valor inmediato y mayor impacto en la experiencia inicial del usuario.

6.6. Diseño de la arquitectura del sistema

Durante esta fase también se definió la arquitectura general del sistema, la cual se estructuró en tres componentes principales: Frontend, Backend y Base de datos, comunicados mediante una API REST.

- Frontend: desarrollado con Vue.js, orientado a componentes, con estructura modular. En esta etapa se implementó una versión funcional no responsiva, centrada en validar la experiencia de usuario con las funcionalidades clave.
- Backend: desarrollado con Django y Django REST Framework, encargado de la lógica de negocio, la autenticación con JWT, el control de permisos y la gestión de la API REST.
- Base de datos: utilización de PostgreSQL, estructurada con relaciones normalizadas para entidades como propiedades, usuarios, imágenes, citas y cambios de precios. Soporta operaciones geográficas mediante extensiones.

La arquitectura se diseñó bajo un modelo multi-tenant, donde una misma instancia lógica del sistema puede servir a múltiples inmobiliarias, manteniendo el aislamiento de datos y una personalización visual básica por cliente. Esta elección técnica permitió construir un sistema escalable, reutilizable y de bajo costo de mantenimiento.

Se integraron además dos servicios externos fundamentales: Google Maps API (para la geolocalización de propiedades en el mapa) y Firebase Cloud Messaging (para el sistema de notificaciones push en tiempo real).

6.7. Gestión de riesgos

Como parte del análisis se identificaron riesgos técnicos y operativos que podían impactar en el desarrollo o en la estabilidad del MVP. Se definieron estrategias específicas de mitigación para cada uno:

- Complejidad en la integración con servicios externos (Google Maps, Firebase): se asignó tiempo adicional para pruebas independientes antes de la integración final al sistema.
- Cambios de requerimientos durante el desarrollo: se realizaron demostraciones mensuales al cliente para validar los avances funcionales y ajustar la planificación sin afectar la base del sistema.
- Ausencia temporal de un integrante del equipo: el cronograma contempló esta posibilidad y se redistribuyeron tareas en semanas críticas para mantener el ritmo de trabajo y garantizar la entrega final.

6.8. Cierre de análisis

El análisis detallado del sistema permitió sentar una base técnica y funcional robusta para el desarrollo del MVP. Gracias a este proceso, se establecieron objetivos claros, se identificaron oportunidades de mejora y se redujeron al mínimo los desvíos críticos. La claridad lograda en esta etapa fue clave para garantizar que cada desarrollo posterior estuviera justificado, alineado con los objetivos del cliente y técnicamente sustentable.

7. Metodología de trabajo

En el desarrollo de software, la elección y correcta aplicación de una metodología de trabajo resulta fundamental para organizar el proceso, controlar el avance, adaptarse a imprevistos y garantizar entregas funcionales en tiempo y forma. Dado que este proyecto fue realizado en el marco de una Práctica Profesional Supervisada, con un equipo reducido y un objetivo acotado pero real (desarrollar un MVP funcional para un cliente), se decidió adoptar una metodología ágil híbrida basada en Scrumban.

Este enfoque combinó elementos de Scrum (entregas por fases funcionales, historias de usuario, retrospectivas) con la visualización continua del trabajo propia de Kanban,

utilizando herramientas como GitHub Projects y Trello para organizar tareas, priorizar entregables y dar seguimiento al flujo de desarrollo.
A continuación, se describe el ciclo de vida del desarrollo adoptado para el sistema, y posteriormente se detalla la metodología ágil aplicada (Scrumban) como estrategia de gestión del proyecto.

7.1. Ciclo de vida del desarrollo de software

El desarrollo del sistema web realizado en el marco de esta Práctica Profesional Supervisada se estructuró siguiendo un ciclo de vida incremental ágil y compatible con las metodologías modernas de desarrollo iterativo como Scrumban. Este enfoque permitió abordar el proceso de forma ordenada pero flexible, respondiendo tanto a los objetivos técnicos del proyecto como a las necesidades reales del cliente y a los recursos disponibles.
A diferencia del modelo en cascada, el modelo incremental ágil plantea una construcción del sistema por fases funcionales autónomas (incrementos), donde cada fase recorre las etapas clásicas del ciclo de vida (análisis, diseño, desarrollo, pruebas y despliegue), entregando versiones parciales pero funcionales del producto, que se validan y mejoran progresivamente.
Esta elección fue especialmente adecuada para el desarrollo de un Producto Mínimo Viable (MVP) en contexto real, con validaciones periódicas por parte del cliente, adaptaciones técnicas en tiempo real y una planificación abierta a cambios sin comprometer la estabilidad del proyecto.

a) Análisis y relevamiento

El ciclo se inició con una etapa de relevamiento de requisitos, en la cual se identificaron las necesidades operativas del cliente, los perfiles de usuario, las funcionalidades esperadas y las deficiencias de herramientas utilizadas previamente. Este análisis fue clave para delimitar el alcance del MVP, estableciendo una lista priorizada de requisitos funcionales y no funcionales. A partir de allí, se trazaron los objetivos técnicos que guiarían las siguientes fases del proyecto.

b) Diseño arquitectónico y planificación técnica

Sobre la base del análisis previo, se diseñó la arquitectura general del sistema, orientada a una estructura modular y desacoplada, bajo un modelo multi-tenant de instancia única. Esta arquitectura permitió garantizar el aislamiento lógico de los datos por cliente, facilitar la escalabilidad horizontal del sistema y habilitar una reutilización extensiva del código base.

Además, se planificó el ciclo de trabajo en fases funcionales que abarcaron infraestructura, backend, frontend, validación y documentación, organizadas en un cronograma representado mediante un diagrama de Gantt.

c) Desarrollo incremental

La etapa de codificación se llevó a cabo en ciclos sucesivos, cada uno centrado en una funcionalidad específica validable de forma independiente. Se desarrolló el backend utilizando Python, Django y Django REST Framework, con autenticación JWT, modelo multi-tenant y endpoints RESTful documentados. En paralelo, se implementó el frontend en Vue.js como una Single Page Application (SPA), comunicada mediante API y adaptable visualmente por cliente. Las funcionalidades se desarrollaron en entregas incrementales, que fueron validadas internamente y con el cliente.

d) Pruebas funcionales y validación continua

En cada fase se incorporaron pruebas unitarias e integraciones parciales para asegurar la estabilidad del backend y la correcta comunicación entre capas. A su vez, se organizaron validaciones funcionales con el cliente mediante entregas intermedias, permitiendo aplicar correcciones, mejoras o ajustes menores sin alterar la estructura central del sistema. Esta estrategia se alinea con el principio de mejora continua propuesto por el enfoque ágil adoptado.

e) Automatización, despliegue y documentación

El cierre del ciclo técnico incluyó la implementación de un pipeline de integración y despliegue continuo (CI/CD) mediante GitHub Actions y la contenerización completa del sistema con Docker. Esto permitió facilitar la replicación de nuevas instancias del sistema para distintos clientes, centralizando el mantenimiento operativo y garantizando entornos reproducibles. Asimismo, se documentó el sistema utilizando herramientas como Swagger y Notion, tanto para la API técnica como para la operatividad funcional del MVP entregado.

f) Aplicación del enfoque ágil con Scrumban

El ciclo de vida incremental fue organizado y ejecutado bajo la metodología ágil Scrumban, que combinó elementos de planificación por fases funcionales (Scrum) con visualización y control de flujo de trabajo (Kanban). Este enfoque híbrido permitió mantener una gestión clara de tareas, respetar prioridades funcionales, limitar el trabajo en progreso y facilitar la adaptación continua frente a cambios.

7.2. Scrumban

El desarrollo ágil de software ha transformado radicalmente la forma en que los equipos gestionan proyectos, particularmente en contextos donde los requerimientos pueden cambiar o ajustarse durante el proceso. En ese marco, Scrumban se presenta como una metodología ágil híbrida, que combina las estructuras iterativas de Scrum con los principios de visualización y mejora continua de Kanban.

Scrumban fue propuesto por Corey Ladas en su obra “Scrumban: Essays on Kanban Systems for Lean Software Development” (2009), como una respuesta práctica a las limitaciones de Scrum cuando se aplica de forma estricta, y a la necesidad de mantener control visual del flujo de trabajo sin perder la planificación por fases.

Esta metodología es especialmente adecuada para proyectos con requerimientos estables pero con necesidad de adaptación interna, equipos reducidos sin roles formales y entornos donde la entrega de valor incremental es más efectiva que los ciclos rígidos de sprints cerrados.

Entre sus características más destacadas se encuentran:

- Planificación flexible por fases técnicas o por prioridad, en lugar de iteraciones fijas de tiempo.
- Visualización continua del trabajo mediante tableros tipo Kanban.
- Límites de trabajo en progreso (WIP) para evitar la multitarea excesiva y garantizar foco.
- Priorización dinámica del backlog, sin necesidad de una definición cerrada al inicio del sprint.
- Adaptación progresiva en función de feedback y condiciones del equipo o del cliente.

El uso de Scrumban no implica descartar las buenas prácticas de Scrum, sino reinterpretarlas bajo un modelo más flexible, lo cual fue clave para este proyecto, dado su alcance, recursos disponibles y ritmo de avance.

7.3. Enfoque metodológico adoptado

Scrumban se presenta como una alternativa flexible a las metodologías ágiles tradicionales, especialmente adecuada para equipos reducidos, proyectos iterativos y productos con requerimientos bien definidos desde el inicio. En este caso, permitió planificar fases concretas (infraestructura, backend, frontend, documentación) sin la necesidad de roles formales o ceremonias estrictas.

Se utilizaron tableros con columnas clásicas (Pendiente, En Proceso, Bloqueado, Finalizado) y control de tareas simultáneas (WIP – Work In Progress), lo cual permitió mantener el enfoque, evitar la dispersión del esfuerzo y detectar bloqueos a tiempo.

7.4. Fundamentación de la elección

La decisión de utilizar Scrumban respondió a una serie de criterios objetivos:

- Requerimientos funcionales definidos desde el inicio, lo que permitió una planificación por fases en lugar de iteraciones abiertas.
- Fecha de entrega fija, lo cual demandaba un seguimiento riguroso y adaptable ante posibles imprevistos.
- Equipo reducido de trabajo, donde la coordinación podía resolverse de manera asincrónica y sin reuniones formales.
- Necesidad de visibilidad y trazabilidad, tanto para el seguimiento interno como para las validaciones con el cliente.

Desde una perspectiva de ingeniería del software, Scrumban permitió controlar la deuda técnica, mantener un backlog accesible, realizar ajustes durante el ciclo de vida del producto y priorizar entregables funcionales por valor.

Además de responder a características propias del contexto (equipo reducido, entregables bien definidos, fechas fijas), la elección de Scrumban sobre otras metodologías ágiles se justificó por su capacidad de adaptarse sin comprometer estructura.

Por ejemplo, adoptar Scrum puro habría requerido asignar formalmente los roles de Scrum Master y Product Owner, así como mantener rituales como dailies obligatorias, sprint plannings y retrospectivas programadas, que no se adecuaban al flujo real del trabajo ni a la comunicación asincrónica del equipo. Por otro lado, utilizar Kanban puro podría haber resultado en una falta de enfoque estratégico o planificación a mediano plazo, especialmente en tareas críticas como infraestructura o despliegue.

Scrumban permitió fusionar lo mejor de ambos enfoques: planificación por fases sin rigidez, visibilidad constante del avance, y un marco ágil de mejora continua basado en entregas con valor funcional. Esta metodología está ampliamente adoptada en empresas tecnológicas y startups que trabajan con equipos multidisciplinarios pequeños, lo que también refuerza su aplicabilidad al perfil profesional buscado en esta PPS.

7.5. Aplicación práctica de Scrumban al proyecto

La metodología se aplicó del siguiente modo:

Elemento	Aplicación concreta
Sprint Planning	Cada fase técnica (infraestructura, backend, frontend) se abordó como un sprint funcional.
Sprint Backlog	Historias de usuario priorizadas según dependencia y valor funcional.
Tablero Kanban	Usado en GitHub Projects y Trello para visualizar tareas, controlar WIP y detectar bloqueos.
Daily Meetings (flexibles)	Coordinación asincrónica mediante comentarios en issues y revisiones semanales de avance.
Sprint Review	Validaciones funcionales con entregables concretos y revisión mensual con el cliente.
Retrospectiva	Evaluaciones informales al cierre de cada fase, para identificar mejoras.

Uno de los aspectos más relevantes en la aplicación de Scrumban fue la gestión del WIP (Work In Progress), es decir, el número máximo de tareas que podían estar activas al mismo tiempo por desarrollador. Para evitar dispersión del esfuerzo y errores por multitarea, se estableció un límite informal de dos tareas activas por persona, permitiendo alternancia solo en caso de bloqueo técnico o dependencia externa. Esta estrategia, aunque simple, resultó efectiva para mantener el foco y garantizar que cada módulo técnico fuera entregado con un nivel de madurez suficiente antes de avanzar.

Además, se hizo uso de etiquetas y categorías dentro de los tableros para distinguir entre tareas técnicas (backend, frontend, infraestructura), tareas de documentación y tareas de validación. Esto permitió mantener una vista clara del tipo de actividad en curso y facilitó la redistribución de responsabilidades, especialmente durante las semanas en las que uno de los integrantes del equipo no pudo participar.

Este enfoque permitió organizar el trabajo en torno a entregas concretas, sin caer en la rigidez de metodologías tradicionales ni en la informalidad de un modelo ad hoc.

7.6. Beneficios esperados y observados

Durante el desarrollo del MVP, la aplicación de Scrumban permitió:

- Organizar el trabajo de manera progresiva, clara y flexible.
- Controlar el avance en tiempo real con alta visibilidad del flujo de tareas.
- Reducir errores por trabajo simultáneo no coordinado.
- Favorecer la entrega continua de valor y el enfoque en funcionalidades críticas.

- Adaptarse a imprevistos, como la ausencia temporal de un miembro del equipo, sin afectar el cumplimiento de entregables.

7.7. Valoración de la metodología aplicada

La elección de Scrumban como metodología de trabajo fue adecuada al contexto del proyecto, al perfil del equipo y al tipo de producto desarrollado. Este enfoque no solo facilitó la organización interna del trabajo, sino que también permitió entregar una solución real a un cliente, en tiempo y forma, validada funcionalmente en cada etapa. Su implementación representó una experiencia concreta de gestión ágil en un entorno real, fortaleciendo habilidades que serán directamente transferibles a futuros contextos profesionales.

7.8. Evaluación crítica de la experiencia con Scrumban

La aplicación de Scrumban durante este proyecto no solo permitió cumplir con los objetivos funcionales, sino que representó una experiencia concreta de gestión ágil en un entorno de trabajo real. La metodología demostró ser adecuada tanto para las necesidades técnicas del sistema como para el contexto operativo del equipo.

Entre los aspectos más valorados de la experiencia se destacan:

- Baja fricción en la implementación: no fue necesario invertir tiempo en definir ceremonias formales, ni establecer estructuras organizativas externas al proyecto.
- Alta visibilidad del avance: el uso de tableros y etiquetas permitió tomar decisiones informadas sobre prioridades y redistribución de carga.
- Autonomía y responsabilidad individual: cada integrante del equipo asumió sus tareas con claridad, favoreciendo una cultura de cumplimiento por compromiso más que por control externo.
- Capacidad de adaptación real: se pudieron reordenar fases, reemplazar tareas y redefinir alcances intermedios sin perder estabilidad ni control del proyecto.

No obstante, también se reconocen áreas de mejora. La informalidad de algunas decisiones (como la falta de una planificación visual del backlog futuro) pudo haber provocado cierta incertidumbre hacia el final del desarrollo. Asimismo, en momentos críticos, una mayor documentación interna sobre dependencias técnicas podría haber evitado retrabajos menores.

En síntesis, la experiencia con Scrumban fue altamente positiva y formativa. Permitió consolidar habilidades de autogestión, priorización de valor, trazabilidad de tareas, y entrega incremental de funcionalidades reales. Su aplicación dejó aprendizajes directamente

transferibles al entorno profesional de desarrollo ágil, donde los equipos interdisciplinarios, los cambios de contexto y la entrega continua son moneda corriente.

8. Historias de usuario

Como parte de la planificación funcional del proyecto, se adoptó el enfoque de historias de usuario para modelar los requerimientos del sistema desde la perspectiva de los distintos actores involucrados. Esta técnica permitió representar funcionalidades concretas de forma clara, comprensible y orientada al valor funcional aportado al usuario final, facilitando así la organización del trabajo y la priorización de entregas dentro del marco metodológico ágil utilizado (Scrumban).
Cada historia de usuario se redactó utilizando la estructura clásica:

> **Como [rol], quiero [acción], para [objetivo]**

Esta formulación permitió mantener el enfoque en los objetivos reales del sistema, organizando los requerimientos en bloques funcionales que corresponden a las distintas etapas del desarrollo. A continuación, se listan las historias priorizadas que guiaron la implementación del MVP.

8.1. Configuración inicial, despliegue e infraestructura

- Como desarrollador, quiero clonar el repositorio e iniciar el entorno Dockerizado con un solo comando, para facilitar la instalación local del sistema.
- Como desarrollador, quiero automatizar el flujo CI/CD con GitHub Actions, para asegurar pruebas, construcción de contenedores y despliegue automático.
- Como administrador del sistema, quiero tener la infraestructura correctamente configurada en un VPS, para alojar el sistema en un entorno accesible y seguro.

8.2. Personalización visual del sitio

- Como administrador de una inmobiliaria, quiero subir el logo de mi empresa para que se visualice en el encabezado del sitio.
- Como administrador de una inmobiliaria, quiero definir los colores principales y secundarios del sitio, para que reflejen mi identidad institucional.
- Como administrador de una inmobiliaria, quiero modificar los textos de la página principal (título, subtítulo y presentación), para adaptarlos a la comunicación de mi empresa.

8.3. Gestión de propiedades

- Como agente inmobiliario, quiero cargar una nueva propiedad con imágenes, precio, ubicación y características, para publicarla en el sitio.
- Como agente inmobiliario, quiero editar propiedades ya publicadas, para mantener la información actualizada.
- Como agente inmobiliario, quiero eliminar propiedades que ya no están disponibles, para evitar confusiones.

8.4. Visualización y navegación pública

- Como visitante, quiero ver las propiedades listadas con información clave y fotos, para explorar las ofertas disponibles.
- Como visitante, quiero aplicar filtros por ubicación, tipo y precio, para encontrar propiedades según mis criterios.
- Como visitante, quiero visualizar la ubicación de las propiedades en un mapa interactivo, para evaluar zonas de interés.
- Como visitante, quiero acceder a un formulario de contacto para enviar consultas sobre propiedades o servicios, para recibir respuesta de la inmobiliaria.

8.5. Funcionalidades avanzadas

- Como visitante, quiero visualizar el historial de precios de una propiedad, para analizar su evolución.
- Como visitante, quiero poder realizar comparaciones de propiedades por criterios como precio, tamaño y ubicación.

8.6. Solicitud de visitas y calendario

- Como visitante registrado, quiero solicitar una visita a una propiedad en un día y horario disponible, para coordinar un encuentro con el agente.
- Como agente inmobiliario, quiero visualizar un calendario de visitas agendadas, para organizar mi agenda laboral.

8.7. Notificaciones

- Como visitante registrado, quiero poder agregar propiedades a favoritos para poder recibir notificaciones de cambios en la publicación en mi email y mediante notificación push.

- Como agente inmobiliario, quiero enviar notificaciones a usuarios interesados, para mantenerlos informados de novedades o actualizaciones.
- Como desarrollador, quiero integrar un sistema de notificaciones push mediante Firebase, para enviar mensajes en tiempo real desde el backend.

8.8. Automatización de publicaciones

- Como agente inmobiliario, quiero compartir automáticamente mis publicaciones en redes sociales, para aumentar la visibilidad sin esfuerzo manual.

Esta historia de usuario fue documentada durante el desarrollo como una funcionalidad valiosa, pero no fue contemplada inicialmente como requisito esencial del MVP, y por lo tanto, no se priorizó en la primera iteración. Su inclusión quedó sujeta a futuras versiones del sistema, dada su complejidad técnica y el enfoque en funcionalidades críticas para el cliente.

8.9. Seguridad y gestión de usuarios

- Como administrador del sistema, quiero gestionar los accesos y permisos, para proteger la integridad del contenido.
- Como desarrollador, quiero definir roles diferenciados (administrador, agente, visitante), para adaptar la experiencia de uso según el perfil del usuario.

8.10. Documentación y validación

- Como desarrollador, quiero generar documentación técnica y de uso accesible, para que el sistema pueda ser utilizado y mantenido correctamente.
- Como cliente, quiero recibir demostraciones funcionales del sistema de forma periódica, para validar avances y sugerir ajustes antes de la entrega final.

El uso de historias de usuario permitió transformar los requerimientos detectados durante el análisis en funcionalidades concretas, operativas y priorizables. Su redacción clara y orientada a los distintos perfiles de usuario facilitó la organización del trabajo, la trazabilidad de tareas y la validación funcional de cada entrega del MVP. Esta técnica fue especialmente útil en un entorno de desarrollo incremental, en el que se buscó entregar valor de forma continua, alineando el desarrollo técnico con las necesidades reales del cliente.

9. Lenguajes y tecnologías utilizadas

La selección de tecnologías para el desarrollo del sistema se fundamentó en la necesidad de construir un producto mínimo viable (MVP) funcional, escalable y técnicamente robusto, dentro de un entorno controlado y con tiempos de desarrollo definidos. Dado que se trató de un proyecto real con validación por parte de un cliente externo, se priorizó el uso de herramientas con comunidades activas, documentación extensa, buenas prácticas consolidadas y facilidad de integración entre capas.

Para lograr un sistema desacoplado, modular y extensible, se implementó una arquitectura basada en servicios, separando la lógica de negocio (backend), la interfaz de usuario (frontend), la persistencia de datos (base de datos) y la infraestructura de despliegue (contenedores y servicios externos). A continuación se detallan las tecnologías empleadas en cada componente del sistema, junto con su función específica, justificación de elección y evaluación técnica.

9.1. Backend: tecnologías y lenguajes utilizados

9.1.1. Python

Python es un lenguaje de programación interpretado, orientado a objetos y de propósito general, ampliamente utilizado en el desarrollo web, científico y automatización de procesos. Su sintaxis clara y su bajo nivel de complejidad lo convierten en una herramienta robusta y versátil, adecuada tanto para prototipado ágil como para entornos productivos.

Fue elegido como lenguaje principal del backend por su integración directa con el framework Django, su sintaxis altamente legible y su ecosistema robusto. La posibilidad de utilizar entornos virtuales y su compatibilidad con PostgreSQL aseguraron una implementación confiable de la lógica de negocio, los modelos de datos y la exposición de endpoints.

Ventajas:

- Reducción del tiempo de desarrollo por su sintaxis concisa.
- Ecosistema maduro con bibliotecas bien documentadas.
- Portabilidad entre entornos y compatibilidad con herramientas DevOps.

Limitaciones:

- Rendimiento inferior a lenguajes compilados (no crítico para un MVP).
- No óptimo para tareas concurrentes intensivas o procesos multihilo complejos.

9.1.2. Django y Django REST Framework (DRF)

Django es un framework web de alto nivel basado en Python que promueve el desarrollo rápido y seguro de aplicaciones. Incluye un ORM integrado, sistema de autenticación, panel de administración y otras funcionalidades. DRF es una extensión que permite construir APIs RESTful de manera estructurada y escalable.
La combinación Django + DRF permitió estructurar el backend de forma modular, desacoplada del frontend, con una API REST consumible por cualquier cliente externo. Su integración nativa con el ORM y el sistema de serialización de datos permitió una rápida evolución del modelo de datos y una validación robusta.

Ventajas:

- API bien estructurada, con validaciones, permisos y autenticación.
- Modularidad para evolucionar el sistema hacia nuevas funcionalidades.
- Mantenibilidad del código y escalabilidad futura.

Limitaciones:

- Requiere configuración detallada de permisos y vistas.
- Necesita conocimientos sólidos del patrón MVC/MVT.

9.1.3. PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional (RDBMS), conocido por su estabilidad, extensibilidad y cumplimiento de estándares. Soporta transacciones ACID, tipos de datos complejos y funciones personalizadas.
Se eligió PostgreSQL por su integración nativa con Django ORM, su confiabilidad para sistemas multiusuario y su capacidad para manejar relaciones complejas entre entidades, incluyendo el modelo multi-tenant implementado para separar los datos de cada inmobiliaria.

Ventajas:

- Soporte completo para claves foráneas, integridad referencial y migraciones.
- Escalabilidad vertical y horizontal.
- Compatibilidad con entornos Dockerizados.

Limitaciones:

- Mayor consumo de memoria frente a motores más simples (SQLite, por ejemplo).
- Necesita mantenimiento y configuración del servidor de base de datos.

9.2. Frontend: tecnologías utilizadas

9.2.1. JavaScript

JavaScript es un lenguaje de programación interpretado, orientado a eventos, que se ejecuta en el cliente (navegador). Permite manipular el DOM y construir interfaces interactivas en tiempo real.
Fue utilizado a través de Vue.js para construir una SPA (Single Page Application) que se comunica de forma asincrónica con la API. Su uso es fundamental para la interactividad del sistema y la carga dinámica de contenido.

Ventajas:

- Alta compatibilidad con todos los navegadores modernos.
- Facilidad para consumir APIs mediante Axios.
- Flexibilidad para personalizar la interfaz por cliente.

Limitaciones:

- Mayor esfuerzo en la gestión de estado si el sistema crece.
- Posible complejidad de depuración sin herramientas adecuadas.

9.2.2. Vue.js

Vue.js es un framework progresivo para construir interfaces de usuario reactivas y modulares. Su estructura basada en componentes permite dividir la lógica de presentación en bloques reutilizables.
Se eligió Vue.js por su curva de aprendizaje amigable, su compatibilidad con la filosofía SPA y su integración con bibliotecas como Vue Router y Axios. Permitió construir un frontend funcional, mantenible y con separación clara de responsabilidades.

Ventajas:

- Código organizado por componentes.
- Reactividad automática ante cambios de datos.
- Buen rendimiento en aplicaciones medianas.

Limitaciones:

- No incluye gestión de estado por defecto (como Vuex).
- Requiere herramientas de construcción (Vite) para producción.

9.2.3. Google Maps API

Durante el desarrollo del sistema, se utilizó la API de Google Maps para implementar funcionalidades de geolocalización, como la visualización de propiedades sobre un mapa interactivo.
Sin embargo, en el entorno local de desarrollo se empleó la librería Leaflet como alternativa gratuita, con el fin de simular dicha funcionalidad sin incurrir en gastos asociados al consumo de créditos de la API de Google. Esta decisión permitió realizar pruebas funcionales completas de geolocalización y validación de datos, garantizando un desarrollo eficiente y sin costos innecesarios.

9.3. Infraestructura y herramientas DevOps

9.3.1. Docker

Docker es una plataforma de contenedorización que encapsula aplicaciones y sus dependencias en contenedores portables, aislados del sistema operativo anfitrión.
Fue utilizado para asegurar la portabilidad del sistema entre entornos de desarrollo y producción, reducir errores por diferencias de configuración y facilitar la replicación del entorno completo del proyecto (backend, frontend y base de datos).

Ventajas:

- Simplificación del despliegue.
- Aislamiento y control de versiones.
- Reutilización de imágenes y volúmenes.

Limitaciones:

- Curva de aprendizaje en configuración inicial.
- Requiere monitoreo del uso de recursos si se escala.

9.3.2. GitHub Actions

GitHub Actions es una herramienta de CI/CD que permite automatizar flujos de trabajo a partir de eventos en el repositorio (como push o pull request), mediante la definición de scripts en YAML.
Se implementaron pipelines de integración continua para ejecutar pruebas automáticas, construir imágenes Docker y desplegar actualizaciones en el entorno productivo sin intervención manual.

Ventajas:

- Aumento de la eficiencia y reducción de errores.
- Revisión continua de calidad del código.
- Automatización del proceso de entrega.

Limitaciones:

- Requiere diseño cuidadoso de los flujos.
- Las tareas complejas implican múltiples pasos interdependientes.

9.3.3. DigitalOcean (VPS)

DigitalOcean es una plataforma de infraestructura en la nube que ofrece instancias de servidores virtuales privados (VPS), con acceso completo al sistema operativo.
Se eligió DigitalOcean por su bajo costo inicial, su flexibilidad de configuración y su compatibilidad con tecnologías como Docker y PostgreSQL. Brinda un entorno controlado, ideal para desplegar y mantener el MVP de manera independiente.

Ventajas:

- Control total sobre la configuración del sistema.
- Escalabilidad según demanda.
- Compatibilidad con arquitecturas dockerizadas.

Limitaciones:

- Necesidad de administración manual de puertos, firewall y actualizaciones.
- No incluye servicios gestionados por defecto (como backups automáticos).

9.4. Servicios externos

9.4.1. Google Maps API

Google Maps API es un servicio desarrollado por Google que permite integrar mapas interactivos en aplicaciones web, incluyendo marcadores, rutas y geolocalización.
Permitió enriquecer la experiencia del usuario final, mostrando visualmente la ubicación de propiedades en venta o alquiler dentro del sistema. Su uso mejora la navegabilidad y permite una evaluación contextual de los inmuebles.

Ventajas:

- Visualización intuitiva y precisa.
- Personalización de la interfaz del mapa.
- Integración directa con coordenadas geográficas almacenadas.

Limitaciones:

- Requiere autenticación con clave API.
- Uso gratuito limitado a una cuota mensual.

9.4.2. Firebase Cloud Messaging (FCM)

Firebase Cloud Messaging es un servicio ofrecido por Google que permite enviar notificaciones push en tiempo real a navegadores web o dispositivos móviles.
Se integró para permitir la comunicación inmediata con los usuarios, enviando alertas sobre nuevas propiedades, confirmaciones de visitas o cambios relevantes.

Ventajas:

- Canal no invasivo de comunicación.
- Gratuito en niveles bajos de uso.
- Escalable para implementaciones futuras.

Limitaciones:

- No almacena historial de mensajes.
- Requiere gestión manual de tokens de suscripción.

La combinación tecnológica empleada en este sistema fue diseñada para ofrecer un equilibrio entre productividad, estabilidad, escalabilidad y mantenibilidad. La integración de lenguajes y herramientas modernas, junto con decisiones basadas en criterios de arquitectura desacoplada, permitió construir un producto funcional alineado con los objetivos de la PPS. Esta elección tecnológica no solo responde a las necesidades inmediatas del MVP, sino que sienta las bases para la evolución futura del sistema hacia versiones más complejas y entornos de alta demanda.

10. Diseño de la solución

El diseño de la solución técnica representó una etapa clave para transformar los requerimientos funcionales y no funcionales definidos durante el análisis en una arquitectura concreta, escalable y mantenible. Dado el carácter del proyecto, se definió una solución técnica moderna, desacoplada y modular, que permite la validación de funcionalidades esenciales, al tiempo que sienta las bases para futuras iteraciones.
La arquitectura fue pensada con el objetivo de garantizar compatibilidad, reutilización y facilidad de despliegue, utilizando tecnologías ampliamente adoptadas en el desarrollo web

profesional, como Vue.js para el frontend, Django REST Framework para el backend y PostgreSQL como sistema de persistencia de datos.

10.1. Arquitectura general del sistema

La arquitectura del sistema está basada en una estructura desacoplada, compuesta por tres capas principales: Frontend, Backend y Base de datos. Estas capas se comunican entre sí mediante una API REST, lo que favorece el mantenimiento modular, la posibilidad de escalar cada componente de forma independiente y la futura incorporación de otras interfaces (como aplicaciones móviles o dashboards administrativos externos).

Adicionalmente, se adoptó un modelo multi-tenant, en el cual múltiples inmobiliarias pueden operar dentro de una misma instancia lógica del sistema, con separación de datos, configuraciones visuales y control de accesos aislado para cada cliente. Esta estrategia permite una reutilización eficiente del código base, simplifica el despliegue de nuevas instancias y reduce significativamente el costo operativo del sistema.

10.2. Comparación de arquitecturas: Single-tenant, Multi-tenant, Single-instance y Multi-instance

En el proceso de diseño de la solución técnica para este proyecto, fue necesario evaluar distintos modelos de arquitectura que permitieran garantizar la escalabilidad, mantenibilidad y eficiencia operativa del sistema. En particular, se analizaron los enfoques más comúnmente utilizados en soluciones multiusuario: Single-tenant, Multi-tenant, Single-instance y Multi-instance. Comprender estas arquitecturas resultó esencial para fundamentar la estrategia adoptada en el desarrollo del sistema inmobiliario planteado en este trabajo.

a) Modelo Single-tenant

En una arquitectura single-tenant, cada cliente dispone de una instancia independiente de la aplicación y de su base de datos. Esto implica un aislamiento total de recursos, configuraciones y datos, lo cual puede resultar beneficioso en entornos con altos requisitos de seguridad o personalización específica.

Ventajas:

- Aislamiento completo entre clientes.
- Personalización estructural por cliente.
- Reducción del riesgo de fallos cruzados.

Desventajas:

- Costos operativos elevados.

- Multiplicación del esfuerzo de mantenimiento.
- Baja escalabilidad para productos SaaS.

b) Modelo Multi-tenant

Este enfoque permite que múltiples clientes compartan una misma instancia lógica de la aplicación, diferenciándose a través de identificadores internos (por ejemplo, tenant_id). El sistema debe garantizar aislamiento lógico de los datos y configuraciones entre clientes, sin necesidad de replicar el software ni la infraestructura.

Ventajas:

- Mayor eficiencia de recursos.
- Facilidad para desplegar nuevas cuentas de cliente.
- Mantenimiento centralizado.

Desventajas:

- Mayor complejidad en el diseño del modelo de datos.
- Requiere controles estrictos de aislamiento lógico.
- Menor personalización estructural individual.

c) Modelo Single-instance

En este modelo, una única instancia de la aplicación da servicio a todos los usuarios. Puede utilizarse tanto en arquitecturas multi-tenant como en entornos monocliente, y suele acompañarse de configuraciones flexibles que permiten distinguir usuarios o roles dentro del mismo entorno.

Ventajas:

- Sencillez de despliegue y monitoreo.
- Bajo consumo de infraestructura.
- Aplicación centralizada de cambios y mejoras.

Desventajas:

- Puntos únicos de fallo.
- Menor tolerancia a la personalización por cliente.

d) Modelo Multi-instance

Aquí se despliega una instancia independiente del sistema por cada cliente. A diferencia del modelo single-tenant, esta separación puede incluir también entornos de ejecución aislados (servidores, contenedores, etc.), dando lugar a una completa separación física entre clientes.

Ventajas:

- Máximo nivel de personalización y aislamiento.
- Adecuado para entornos regulados o de misión crítica.

Desventajas:

- Alta complejidad operativa.
- Escalabilidad limitada.
- Costos de infraestructura más elevados.

Cuadro comparativo general

Arquitectura	Aislamiento	Escalabilidad	Mantenimiento	Personalización	Costo
Single-tenant	Muy alto	Bajo	Alto	Alta	Alto
Multi-tenant	Medio	Muy alto	Bajo	Media	Bajo
Single-instance	Bajo	Alto	Bajo	Baja	Bajo
Multi-instance	Muy alto	Bajo	Alto	Alta	Alto

Este análisis fue clave para seleccionar el modelo más adecuado a los objetivos del proyecto y a las necesidades operativas del cliente.

10.3. Justificación de la arquitectura adoptada

Luego del análisis comparativo de modelos arquitectónicos, se adoptó una arquitectura multi-tenant basada en una única instancia lógica (single-instance). Esta elección responde a los objetivos definidos desde el inicio del proyecto: construir una solución escalable, reutilizable y de bajo costo operativo, que permita a múltiples inmobiliarias utilizar el sistema sin necesidad de replicar código ni infraestructura.

Esta estrategia permite que una misma instancia de la aplicación sirva a múltiples clientes de forma simultánea, diferenciando sus datos, configuraciones y estilos visuales a través de mecanismos de aislamiento lógico, sin que exista dependencia o interferencia entre ellos.

10.3.1. Fundamentos de la arquitectura adoptada

A continuación, se detallan las razones principales que fundamentan esta decisión, así como su implementación técnica concreta.

a) Adaptabilidad al modelo SaaS propuesto

El sistema fue concebido como un producto reutilizable, orientado a múltiples clientes del sector inmobiliario bajo un enfoque tipo Software-as-a-Service (SaaS). La arquitectura multi-tenant permite ofrecer la misma instancia del sistema a distintas inmobiliarias con mínima intervención técnica, lo que reduce significativamente los tiempos y costos de incorporación de nuevos clientes. Esta capacidad es fundamental para escalar comercialmente el producto sin comprometer su mantenibilidad.

b) Separación lógica efectiva

El sistema implementa un esquema de separación lógica robusto para garantizar que cada inmobiliaria opere de forma completamente aislada dentro de la misma instancia lógica de la aplicación. Esta separación se logra mediante la identificación dinámica del cliente en función del subdominio solicitado por el navegador. A partir de esta información, el backend determina qué configuración, datos y contenido deben cargarse para ese entorno específico.

Cada entidad clave del sistema (usuarios, propiedades, visitas, configuraciones visuales) está asociada internamente a un identificador único de cliente, y todas las operaciones están condicionadas por este vínculo, lo que garantiza el aislamiento completo de los datos entre inmobiliarias.

Este enfoque permite centralizar el código base y la infraestructura, mientras se mantiene una experiencia personalizada y segura para cada cliente. Además, la separación lógica se aplica no solo a nivel de base de datos, sino también a nivel visual: al acceder a su subdominio, cada inmobiliaria visualiza su logotipo, colores institucionales y textos propios, definidos previamente mediante un panel de configuración accesible desde el back office.

c) Escalabilidad y eficiencia

La utilización de una única base de código y un sistema centralizado permite aplicar actualizaciones, correcciones y nuevas funcionalidades de forma global, beneficiando a todos los clientes sin duplicar esfuerzos. Asimismo, la arquitectura desacoplada (frontend/backend/API REST) y la contenerización con Docker permiten escalar horizontalmente los componentes de forma independiente si fuera necesario, aumentando la capacidad del sistema sin comprometer su estabilidad.

d) Despliegue automatizado y flexible

Gracias a la incorporación de Docker y GitHub Actions, se logró una automatización completa del flujo de integración y despliegue continuo (CI/CD). Esto permite que nuevas instancias del sistema, configuradas para una inmobiliaria específica, puedan ser generadas

y desplegadas automáticamente con sólo definir las variables de entorno correspondientes. Esta estrategia reduce la carga técnica del equipo y permite escalar la plataforma con agilidad ante nuevos requerimientos.

e) Sostenibilidad a largo plazo

La arquitectura adoptada permite mantener bajos costos operativos sin comprometer la calidad del servicio. En lugar de replicar sistemas por cliente (como en el modelo multi-instance), se centraliza el mantenimiento, reduciendo la deuda técnica y aumentando la consistencia en la experiencia de usuario.

Además, este enfoque no impide una eventual evolución hacia esquemas híbridos si ciertos clientes llegaran a requerir un mayor nivel de aislamiento físico o configuraciones especiales. Gracias a la modularidad del sistema y a la utilización de contenedores, sería posible transformar una instancia lógica en una instancia dedicada sin reescribir la base del sistema.

10.3.2. Implementación de la arquitectura multi-tenant en el sistema

Para lograr el aislamiento lógico de datos por cliente en el sistema, se desarrolló un middleware personalizado en Django. Este componente se encarga de interceptar cada solicitud HTTP entrante, identificar el subdominio desde el cual se realiza la consulta y, a partir de ese dato, determinar a qué inmobiliaria (tenant) pertenece dicha solicitud.

El middleware extrae el dominio desde la cabecera Host de la petición, lo compara con el campo dominio del modelo Inmobiliaria, y en caso de encontrar una coincidencia válida, agrega la información del tenant a la petición, que luego es utilizada a lo largo de todo el ciclo de vida de la solicitud para filtrar datos de forma segura.

Este enfoque permite:

- Aislar la información de cada cliente sin necesidad de múltiples bases de datos.
- Reutilizar la misma lógica de negocio y endpoints en toda la aplicación.
- Cargar dinámicamente configuraciones visuales, contenido y datos según el tenant actual.

En caso de que no se identifique un subdominio válido, el middleware responde con un error 404, impidiendo el acceso no autorizado a datos de otros clientes o a entornos no configurados.

Esta implementación es fundamental para el modelo multi-tenant adoptado en el sistema, y permite una escalabilidad eficiente sin replicación de instancias.

```python
class TenantMiddleware:
    """
    Middleware to manage tenant context based on subdomain
    """

    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        # Extraer el dominio completo del request sin puerto
        domain = request.get_host().split(":")[0]

        if domain.startswith("localhost:8000"):
            return self.get_response(request)

        try:
            # Buscar el tenant por dominio
            tenant = Inmobiliaria.objects.get(dominio=domain)
            request.inmobiliaria = tenant
            request.inmobiliaria_id = tenant.id

            # Log para debugging
            logger.debug(f"Tenant encontrado: {tenant.nombre} para dominio: {domain}")

        except Inmobiliaria.DoesNotExist:
            # Log del error para debugging
            logger.warning(f"Tenant no encontrado para dominio: {domain}")
            raise Http404("Tenant not found")

        return self.get_response(request)
```

Figura 2. Fragmento de código del middleware utilizado para implementar el aislamiento lógico multi-tenant.
Fuente: Elaboración propia, basada en la práctica.

10.4. Componentes principales

Frontend

Desarrollado en Vue.js, el frontend actúa como la capa de presentación de la plataforma. Se estructuró como una SPA (Single Page Application) basada en componentes reutilizables, conectados al backend mediante API REST. La versión implementada para el MVP es funcional y no responsiva, centrada en validar la experiencia de usuario con las funcionalidades principales.

Backend

Implementado en Django con Django REST Framework (DRF), el backend gestiona la lógica de negocio, las operaciones CRUD, la autenticación y autorización mediante tokens

JWT, y la integración con servicios externos. Se diseñaron endpoints RESTful que permiten una comunicación fluida, segura y estructurada con el frontend.

Base de datos

PostgreSQL fue seleccionado como sistema de gestión de base de datos por su robustez, rendimiento y capacidad para manejar relaciones complejas entre entidades. Se utilizó el ORM nativo de Django para mapear modelos como propiedades, usuarios, imágenes, visitas y cambios de precio, permitiendo una gestión coherente y validación a nivel de aplicación.

Servicios Externos Integrados

Para ampliar las capacidades del sistema sin aumentar la complejidad del desarrollo, se integraron servicios externos que aportan funcionalidades clave:

- Google Maps API: para la visualización geográfica de propiedades mediante mapas interactivos.
- Firebase Cloud Messaging: para el envío de notificaciones push a usuarios registrados, facilitando la actualización en tiempo real de cambios importantes.

Estas integraciones se realizaron a nivel de backend, utilizando APIs oficiales, con autenticación segura y mecanismos de prueba para validar su comportamiento antes del despliegue.

10.5. Configuración de docker

10.5.1. Dockerfile del backend

Para contenerizar el servicio backend del sistema, se desarrolló un archivo Dockerfile que define la construcción del entorno de ejecución basado en Python y Django. Este archivo permite generar una imagen reproducible del backend, que puede ejecutarse en cualquier servidor compatible con Docker, facilitando el despliegue homogéneo en entornos de desarrollo, prueba o producción.
El proceso definido en el Dockerfile incluye:

- La selección de una imagen base ligera de Python que garantiza compatibilidad con Django y reduce el tamaño de la imagen final.
- La copia del archivo requirements.txt y su instalación mediante pip, asegurando que todas las dependencias del backend estén presentes dentro del contenedor.
- La incorporación del código fuente de la aplicación al contenedor.

- La ejecución de comandos para preparar el entorno (como migraciones de base de datos o recolección de archivos estáticos, si se requiere).
- La definición del comando de inicio, habitualmente el servidor de desarrollo de Django o una aplicación de producción como Gunicorn.

```
# Usa una imagen base de Python 3.9 en su versión slim (ligera)
FROM python:3.9-slim

# Establece el directorio de trabajo dentro del contenedor
WORKDIR /app

# Evita que Python genere archivos .pyc
ENV PYTHONDONTWRITEBYTECODE=1
# Fuerza la salida de Python a ser sin buffer (útil para logs)
ENV PYTHONUNBUFFERED=1

# Copia los archivos de dependencias al contenedor
COPY requirements.txt requirements-dev.txt ./
# Actualiza pip e instala las dependencias de desarrollo
RUN pip install --upgrade pip \
    && pip install -r requirements-dev.txt

# Copia todo el contenido del proyecto al contenedor
COPY . .

# Crea los directorios para archivos estáticos y de medios
RUN mkdir -p /app/staticfiles /app/media
```

Figura 3. Fragmento del archivo Dockerfile utilizado para contenerizar el backend del sistema.
Fuente: Elaboración propia, basada en la práctica.

10.5.2. Docker compose del backend

Para orquestar el backend junto con otros servicios complementarios se utilizó un archivo Docker compose, que permite definir, levantar y administrar múltiples contenedores de forma simultánea mediante un solo comando.

El archivo docker-compose.yml correspondiente al backend específica:

- Un servicio backend, que se basa en el Dockerfile previamente definido.
- Un servicio db, basado en una imagen oficial de PostgreSQL, configurado con nombre de base de datos, usuario, contraseña y persistencia de datos mediante volúmenes.

- Variables de entorno compartidas entre servicios, como credenciales de base de datos o configuraciones de entorno de Django.
- Una red compartida para garantizar la comunicación entre contenedores.
- Dependencias explícitas que aseguran que la base de datos esté disponible antes de iniciar el backend.

```yaml
version: "3.8"  # Versión de la sintaxis de Docker Compose

services:
  db:  # Servicio de base de datos PostgreSQL
    image: postgres:13  # Imagen de PostgreSQL versión 13
    environment:  # Variables de entorno para inicializar la base de datos
      POSTGRES_USER: inmobiliarias
      POSTGRES_PASSWORD: secret
      POSTGRES_DB: inmobiliarias_db
    volumes:
      - postgres_data:/var/lib/postgresql/data/  # Persistencia de datos en volumen nombrado
    restart: always  # Reinicia el contenedor siempre que sea necesario

  backend:  # Servicio de la aplicación backend (Django)
    build:
      context: .  # Directorio de contexto para construir la imagen
      dockerfile: Dockerfile  # Dockerfile a usar para la construcción
    ports:
      - "8000:8000"  # Expone el puerto 8000 del contenedor al host
    command: >  # Comando que se ejecuta al iniciar el contenedor
      sh -c "python manage.py migrate &&
             python manage.py collectstatic --noinput &&
             gunicorn inmob.wsgi:application --bind 0.0.0.0:8000"
      # Realiza migraciones, recolecta archivos estáticos y ejecuta Gunicorn
    volumes:
      - ./staticfiles:/app/staticfiles  # Monta carpeta de archivos estáticos
      - ./media:/app/media  # Monta carpeta de archivos multimedia
    env_file:
      - .env  # Archivo de variables de entorno
    depends_on:
      - db  # Espera a que el servicio de base de datos esté listo
    restart: always  # Reinicia el contenedor siempre que sea necesario

volumes:
  postgres_data:  # Volumen nombrado para persistencia de datos de PostgreSQL
```

Figura 4. Fragmento del archivo utilizado para orquestar los servicios del sistema backend y base de datos.
Fuente: Elaboración propia, basada en la práctica.

10.6. Automatización del despliegue

La solución técnica incluye una capa de automatización del proceso de integración y despliegue continuo (CI/CD), implementada mediante GitHub Actions y Docker. Cada componente del sistema (frontend, backend, base de datos) fue contenerizado, permitiendo su ejecución en entornos controlados, homogéneos y portables.

Los flujos CI/CD contemplan:

- Ejecución de pruebas automáticas sobre la API REST.
- Generación de imágenes Docker.
- Despliegue en entornos definidos (por ejemplo, VPS con configuración personalizada por cliente).

Esta infraestructura permite reducir errores de integración, acelerar la entrega de nuevas versiones y facilitar el mantenimiento post-despliegue.

10.6.1. CI/CD con GitHub Actions

Con el objetivo de automatizar el proceso de pruebas, construcción y despliegue del sistema, se implementó un flujo de Integración Continua y Despliegue Continuo (CI/CD) utilizando GitHub Actions. Esta herramienta permite definir flujos de trabajo (workflows) que se ejecutan automáticamente en función de eventos sobre el repositorio, como push, pull request o release, facilitando una entrega ágil y confiable del software.

En el contexto de este proyecto, se creó un archivo denominado deploy.yml, ubicado en la ruta .github/workflows/, el cual contiene la configuración necesaria para ejecutar tareas automatizadas sobre el servicio backend. Este flujo incluye las siguientes etapas clave:

- Trigger automático: el workflow se activa cada vez que se realiza un push a la rama principal (main o master), garantizando que cada cambio relevante sea validado y desplegado.
- Build del contenedor: el código se prueba y luego se construye una nueva imagen Docker del backend, utilizando el mismo Dockerfile que se emplea localmente. Esto asegura consistencia entre entornos de desarrollo y producción.
- Autenticación con el servidor remoto: mediante variables de entorno y llaves SSH configuradas previamente, GitHub Actions se conecta de forma segura al servidor donde se encuentra alojada la instancia productiva del sistema.
- Actualización remota del servicio: una vez validado y construido, el contenedor se despliega en el VPS de Digital Ocean, reiniciando el servicio con la nueva versión del código, sin intervención manual.

```yaml
# Nombre del workflow
name: 🚀 Deploy Django Backend to Production

# Evento que dispara el workflow: push a la rama main
on:
  push:
    branches:
      - main

jobs:
  deploy:
    # El job se ejecuta en una máquina virtual Ubuntu
    runs-on: ubuntu-latest

    steps:
      # 1. Descarga el código fuente del repositorio
      - name: 📥 Checkout code
        uses: actions/checkout@v3

      # 2. Configura la clave SSH para conectarse al servidor
      - name: 🔐 Set up SSH
        run: |
          mkdir -p ~/.ssh
          echo "${{ secrets.DO_SSH_KEY }}" > ~/.ssh/id_ed25519
          chmod 600 ~/.ssh/id_ed25519
          ssh-keyscan -H ${{ secrets.DO_HOST }} >> ~/.ssh/known_hosts

      # 3. Instala dependencias y corre los tests con pytest
      - name: 🧪 Run tests with pytest
        run: |
          python -m venv venv
          source venv/bin/activate
          pip install -r requirements.txt
          pytest

      # 4. Despliega la aplicación en DigitalOcean usando SSH
      - name: 🚀 Deploy to DigitalOcean
        run: |
          ssh ${{ secrets.DO_USER }}@${{ secrets.DO_HOST }} << 'EOF'
            cd /srv/inmob/inmob-back/
            git pull origin main
            docker compose -f docker-compose.prod.yml down
            docker compose -f docker-compose.prod.yml up -d --build
          EOF
```

Figura 5. Fragmento del archivo deploy.yml que define el flujo de CI/CD mediante GitHub Actions.

Fuente: Elaboración propia, basada en la práctica.

La implementación de este flujo permitió reducir errores humanos, mantener entornos consistentes, disminuir los tiempos de entrega y asegurar que cada versión del sistema pase por etapas controladas de validación antes de llegar a producción. Además, el uso de GitHub Actions elimina la necesidad de herramientas externas adicionales, integrando el proceso de desarrollo, control de versiones y despliegue dentro del mismo ecosistema.

10.7. Diagrama de la arquitectura general

La arquitectura general del sistema responde a un enfoque desacoplado entre frontend, backend y base de datos, organizado bajo un esquema multi-tenant con separación lógica de usuarios por inmobiliaria.

A continuación, se presenta el diagrama de arquitectura general del sistema, donde se visualiza la separación entre frontend, backend y base de datos, así como la interacción mediante API REST y la implementación del modelo multi-tenant:

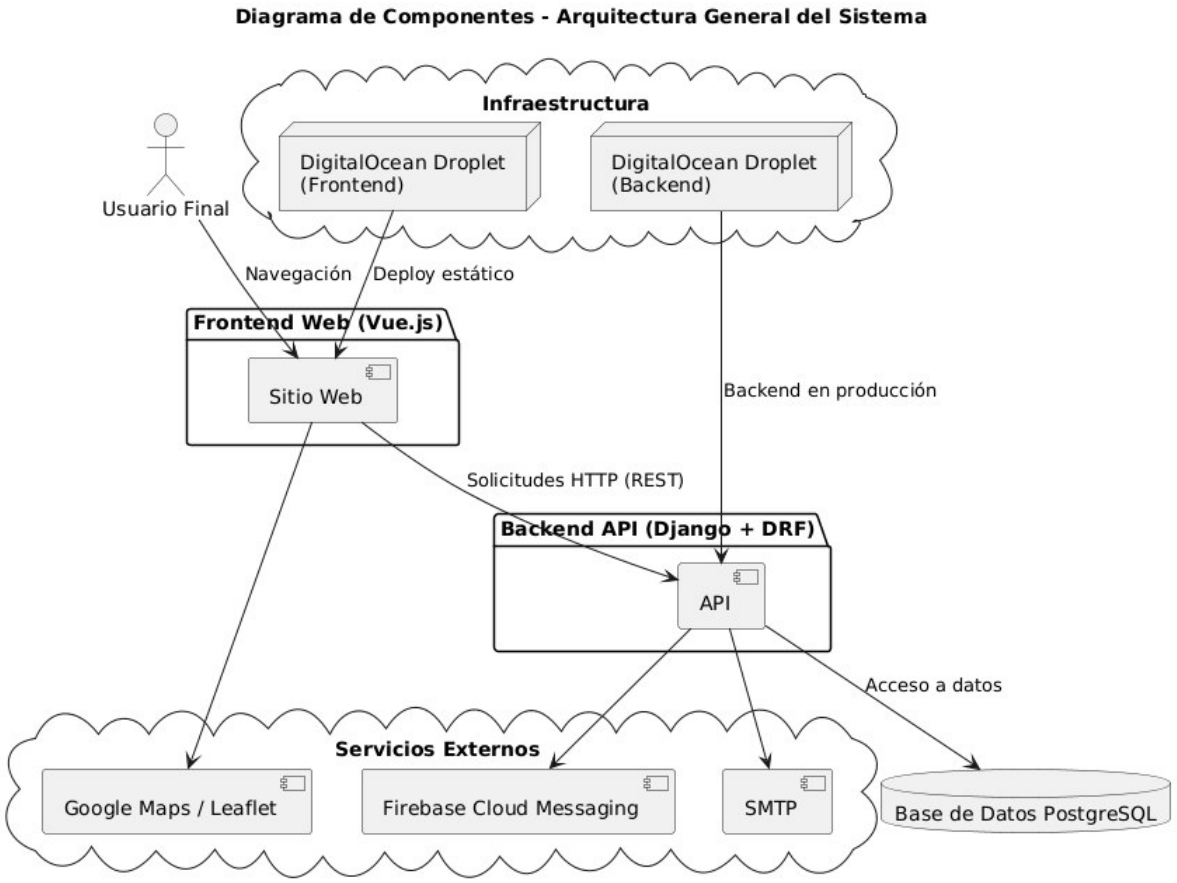


Figura 6. Diagrama de arquitectura general del sistema.

Fuente: Elaboración propia, basada en la práctica.

10.8. Flujos de datos

El flujo de datos sigue un esquema clásico cliente-servidor:

1. El usuario interactúa con la interfaz (frontend), ya sea como visitante, agente o administrador.
2. Las acciones generan solicitudes HTTP hacia el backend, a través de endpoints definidos en la API REST.
3. El backend procesa las solicitudes, consulta o modifica la base de datos según corresponda, y devuelve una respuesta estructurada.
4. En caso de acciones relevantes (cambios de precio, nuevas propiedades), el backend dispara notificaciones push hacia los dispositivos registrados mediante Firebase.
5. El frontend actualiza la interfaz de usuario en función de la respuesta obtenida.

10.9. Decisiones técnicas adicionales

Durante el diseño se tomaron decisiones clave que impactan positivamente en la calidad del sistema:

- Utilizar JWT para la autenticación segura de usuarios, evitando el almacenamiento de sesiones en el servidor.
- Desarrollar el frontend como SPA para permitir una navegación fluida sin recargas.
- Aplicar separación de roles (administrador, agente, visitante) con permisos diferenciados.
- Utilizar GitHub Actions para asegurar una trazabilidad completa del código y sus despliegues.

Estas decisiones favorecen la seguridad, el rendimiento y la mantenibilidad del sistema en el tiempo.

10.10. Escalabilidad y mantenimiento

El sistema fue diseñado desde el inicio para ser escalable y fácil de mantener. Algunas consideraciones clave en este aspecto:

- La arquitectura modular permite escalar horizontalmente los servicios.
- La contenedorización con Docker permite replicar entornos fácilmente.
- La automatización CI/CD reduce la carga operativa sobre el equipo de desarrollo.
- La estructura multi-tenant evita la necesidad de crear una instancia física por cada inmobiliaria, reduciendo costos y complejidad.

- La documentación generada (Swagger, Notion) garantiza la mantenibilidad del sistema incluso con cambios en el equipo técnico.

10.11. Diagrama de componentes

A continuación, se presentan los diagramas de componentes del sistema, diferenciando entre frontend y backend. En ellos se visualizan los principales módulos del sistema y sus interacciones.

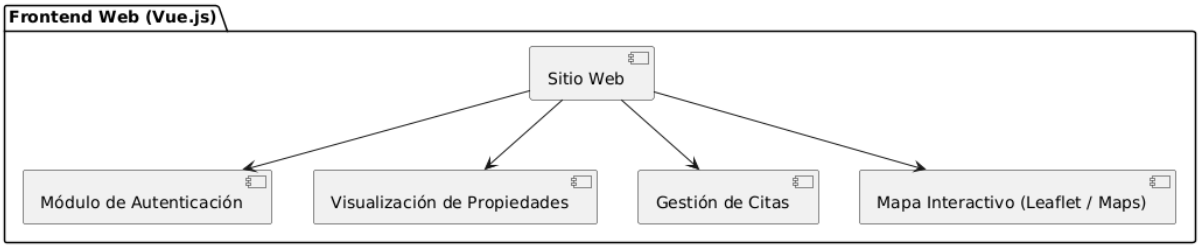


Figura 7. Diagrama de componentes del frontend.
Fuente: Elaboración propia, basada en la práctica.

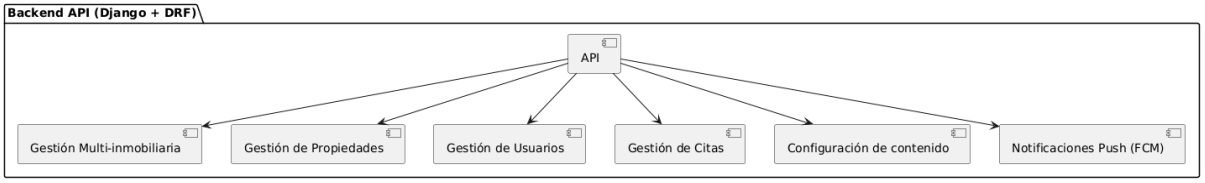


Figura 8. Diagrama de componentes del backend.
Fuente: Elaboración propia, basada en la práctica.

10.12. Diagrama UML del modelo de datos

Como parte del diseño técnico del sistema, se elaboró un diagrama UML de clases que representa la estructura principal de los modelos utilizados en el backend del sistema, así como sus relaciones fundamentales. Este tipo de diagrama permite visualizar de forma clara la relación entre entidades, atributos y dependencias, lo que resulta especialmente útil para comprender la lógica del dominio y su implementación en el ORM de Django.

El diagrama incluye clases representativas como Inmobiliaria, Propiedad, Usuario, Visita, entre otras, junto con sus atributos clave y las relaciones de asociación, herencia o dependencia entre ellas. Este esquema refleja tanto las decisiones tomadas en el modelado del sistema como el soporte necesario para implementar el enfoque multi-tenant adoptado.

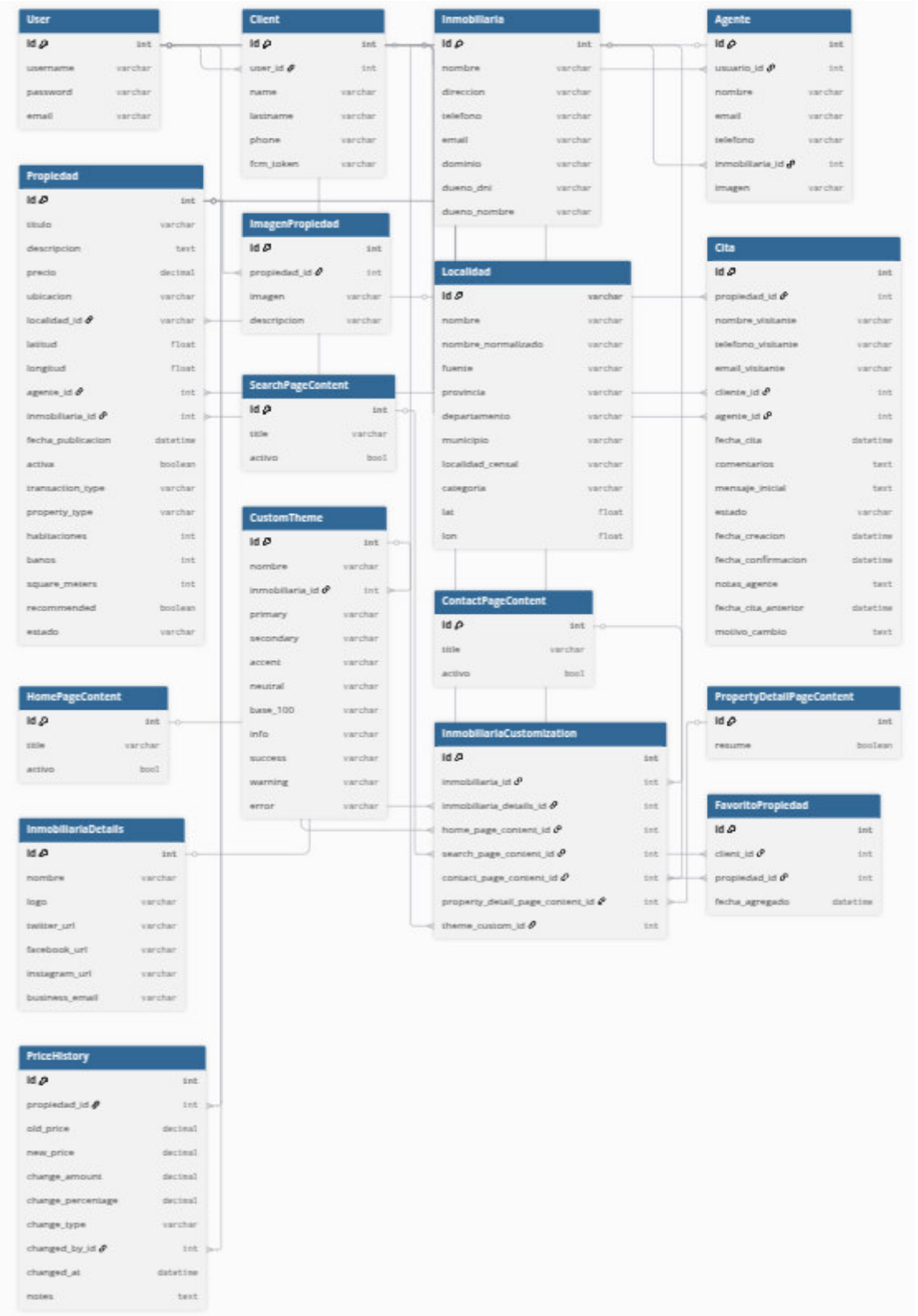


Figura 9. Diagrama UML de clases del modelo de datos del sistema.

Fuente: Elaboración propia, basada en la práctica.

Este diagrama fue utilizado como base de referencia durante el desarrollo del backend, permitiendo validar el diseño antes de codificar, y sirviendo como documento útil para la comprensión futura del sistema por parte de otros desarrolladores o equipos técnicos.

10.13. Pruebas y validación del sistema

Durante el desarrollo del MVP se llevaron a cabo diversas estrategias de validación funcional y técnica, con el objetivo de asegurar el correcto comportamiento del sistema antes de su despliegue.

10.13.1. Pruebas unitarias

Se implementaron pruebas unitarias sobre las funcionalidades más relevantes del backend utilizando el framework Pytest. Estas pruebas permitieron verificar la lógica de negocio, validaciones, y el comportamiento esperado de funciones clave sin necesidad de ejecutar el sistema completo. La ejecución regular de estas pruebas durante el desarrollo ayudó a detectar errores tempranamente y a mantener la estabilidad del código.

10.13.2. Documentación de la API en Postman

Los endpoints principales del backend fueron documentados y organizados en una colección de Postman, lo que permitió realizar pruebas manuales estructuradas, reproducibles y fácilmente compartibles entre miembros del equipo. Esta colección también sirvió como referencia para validar la correcta interacción del frontend con la API REST desarrollada en Django.

10.13.3. Pruebas manuales e integración parcial

Además de las pruebas automatizadas, se llevaron a cabo validaciones manuales sobre la interfaz web, verificando los flujos principales del sistema (registro, login, publicaciones, contacto, turnos). Se testearon también escenarios de error, inputs inválidos y la correcta visualización de datos.

En cuanto a la integración, se evaluó la comunicación entre el frontend (Vue.js) y el backend (Django REST Framework) a través de llamadas HTTP, verificando la coherencia de los datos entre la base de datos y la interfaz de usuario.

10.13. Evidencias de implementación: API y sistema en funcionamiento

Como parte de la validación técnica del Producto Mínimo Viable (MVP), se documentaron los principales puntos de acceso a la API REST y se capturaron distintas secciones del sistema en funcionamiento, como evidencia del cumplimiento de los objetivos funcionales definidos durante el análisis y el diseño.

10.13.1. Visualización de endpoints mediante Swagger

Para facilitar el uso, mantenimiento y validación de la API desarrollada en Django REST Framework, se integró la herramienta Swagger para documentar y exponer de forma interactiva los endpoints implementados.

Esta interfaz permite a desarrolladores, testers y futuros mantenedores del sistema consultar cada recurso disponible, sus métodos HTTP asociados (GET, POST, PUT, DELETE), parámetros requeridos, estructura de respuesta y posibles códigos de estado.

Entre los endpoints clave documentados se encuentran:

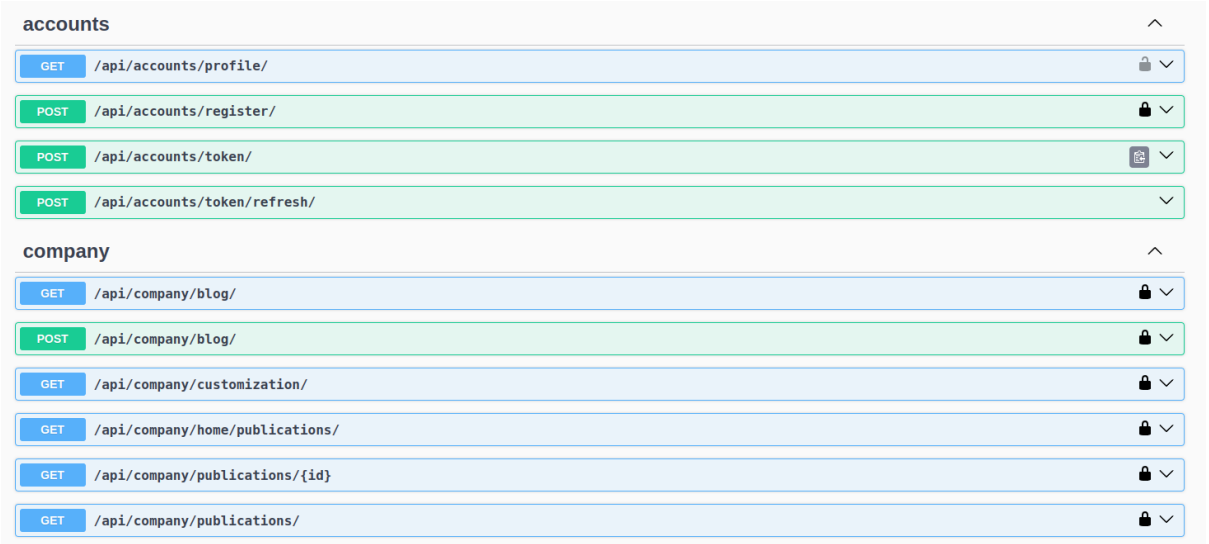


Figura 10. Visualización de la documentación de endpoints mediante Swagger.

Fuente: Elaboración propia, basada en la práctica.

10.13.2. Capturas del sistema en funcionamiento

A continuación, se presentan capturas del sistema ejecutándose en un entorno de pruebas, que ilustran la interfaz construida con Vue.js, adaptada a cada inmobiliaria según su configuración visual:

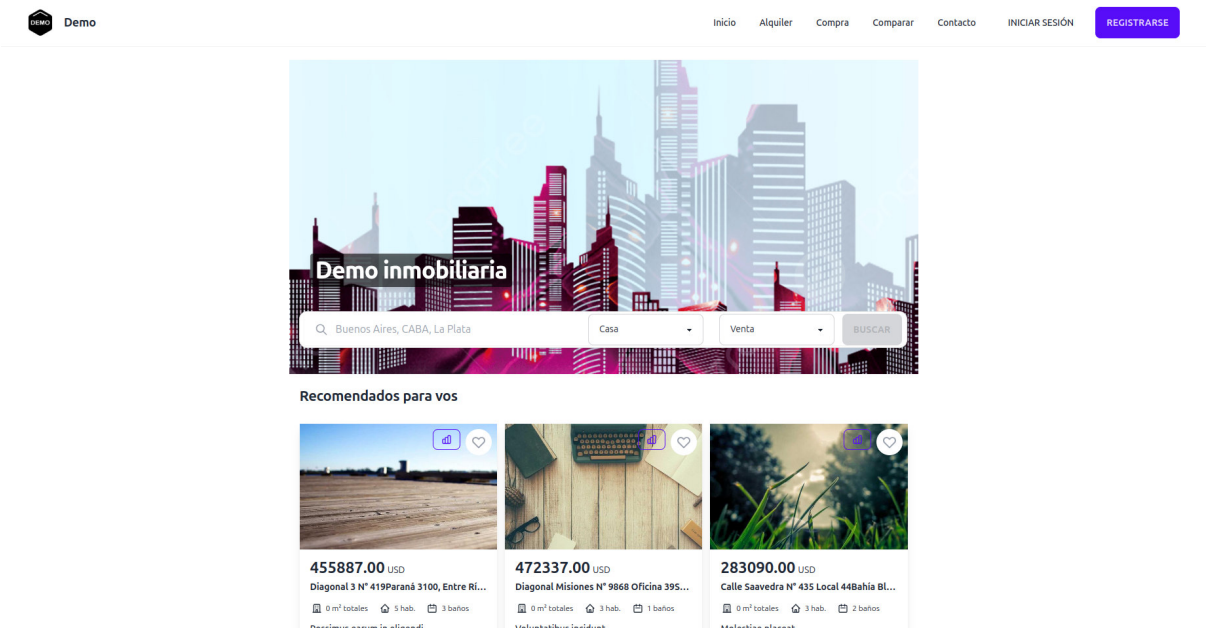


Figura 11. Página de inicio institucional, con logotipo personalizado y presentación de la empresa.

Fuente: Elaboración propia, basada en la práctica.

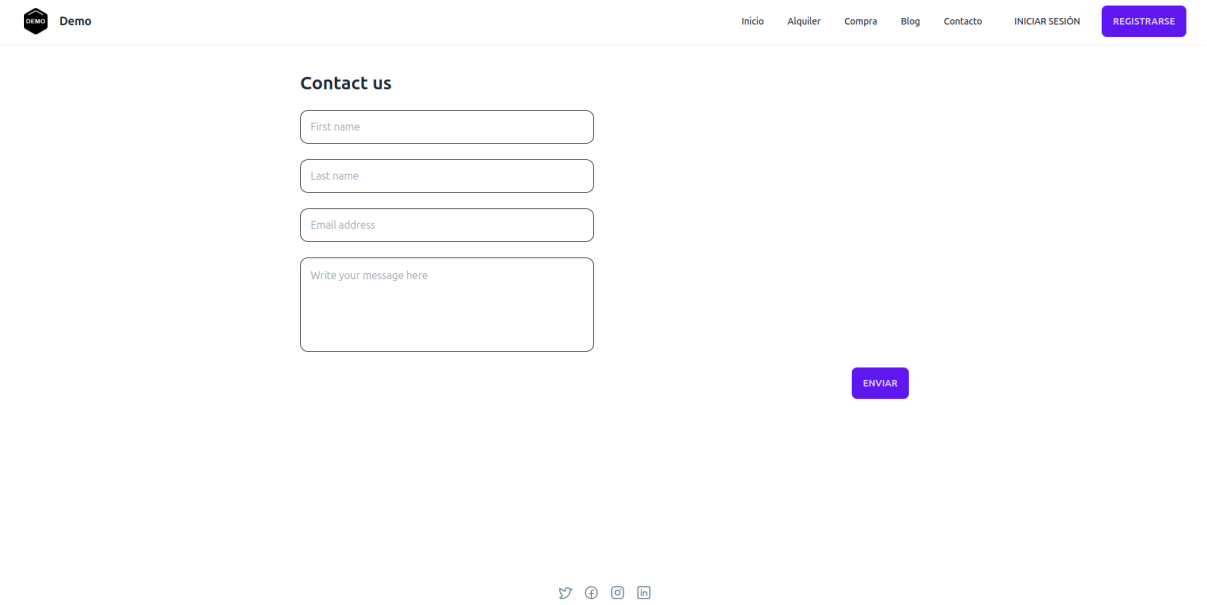


Figura 12. Sección de contacto con formulario de consulta.

Fuente: Elaboración propia, basada en la práctica.

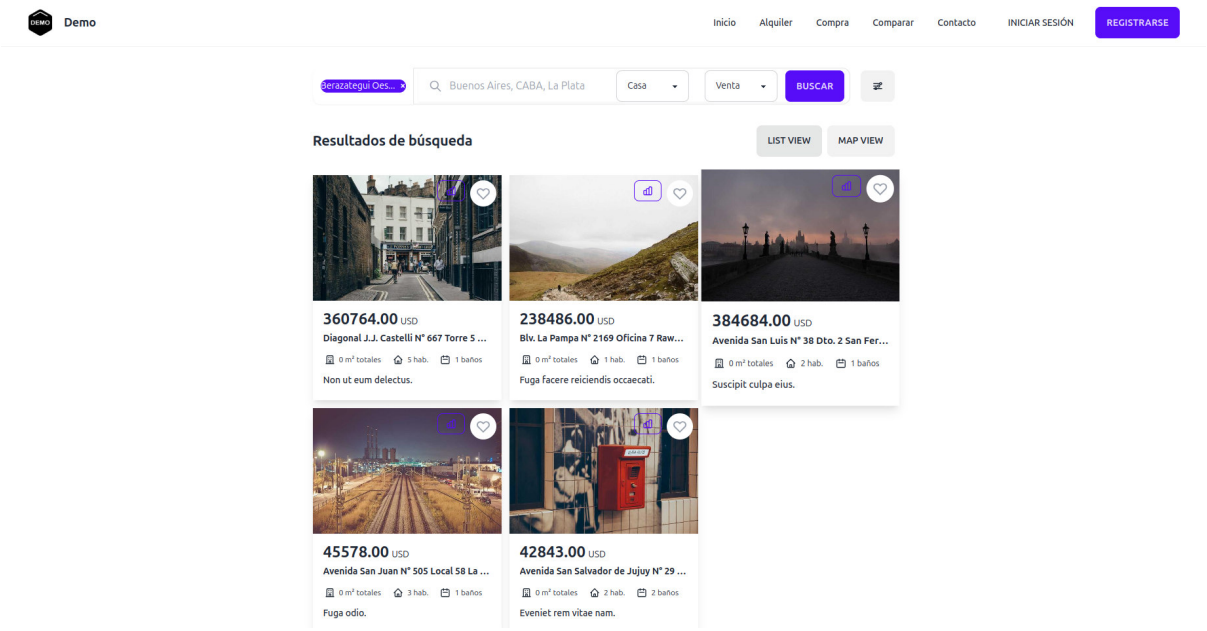


Figura 13. Listado de propiedades públicas, con filtros aplicados (ubicación y tipo).

Fuente: Elaboración propia, basada en la práctica.

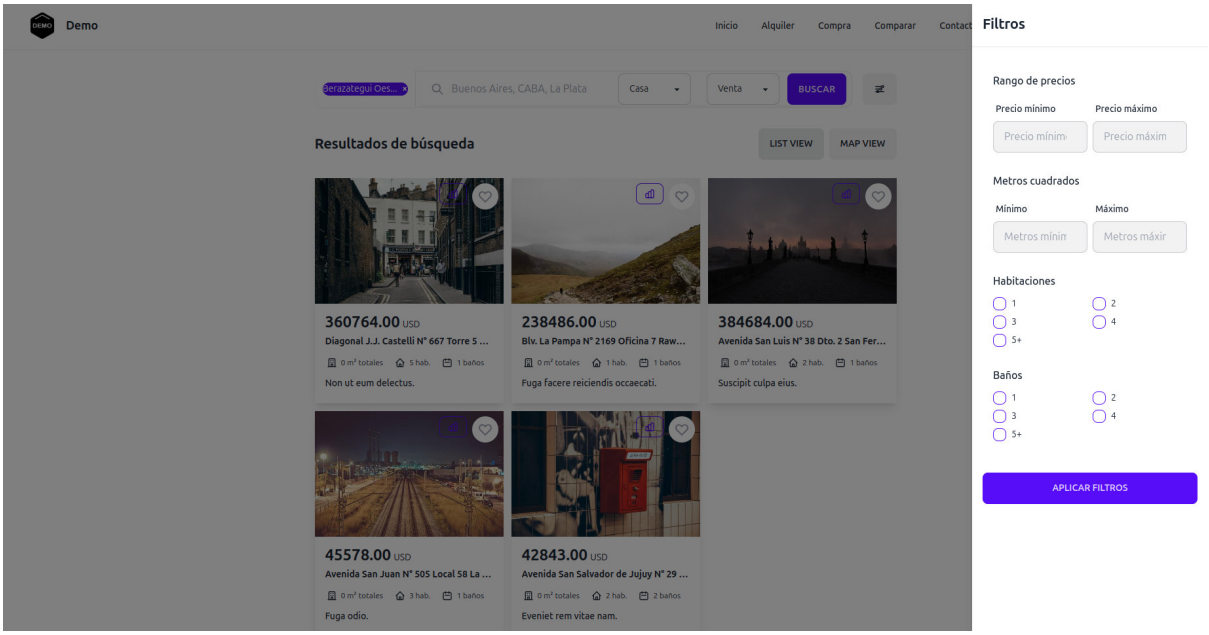


Figura 14. Filtros disponibles para realizar búsquedas.

Fuente: Elaboración propia, basada en la práctica.

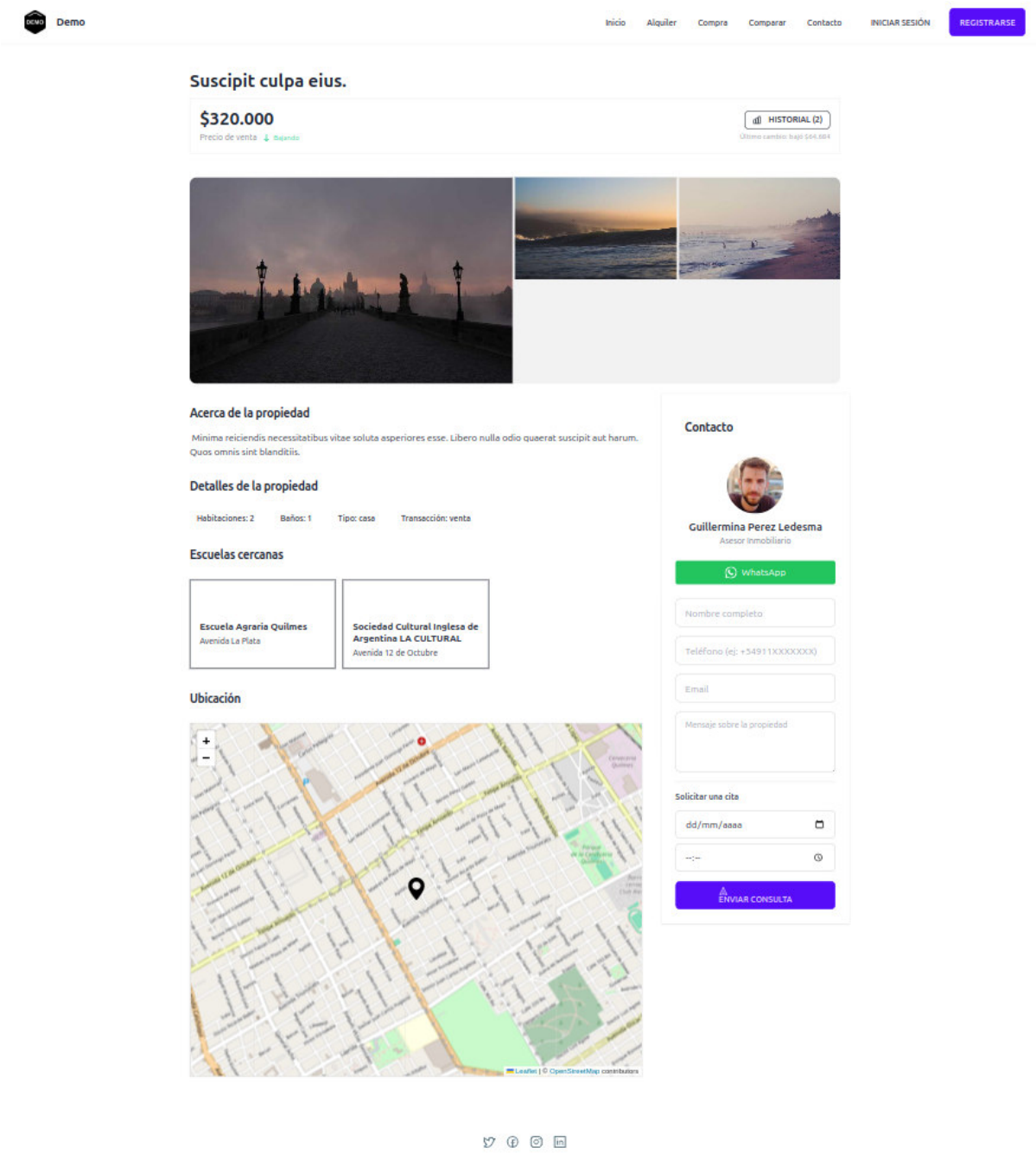


Figura 15. Detalle de una propiedad individual, incluyendo galería de imágenes, características y mapa.

Fuente: Elaboración propia, basada en la práctica.

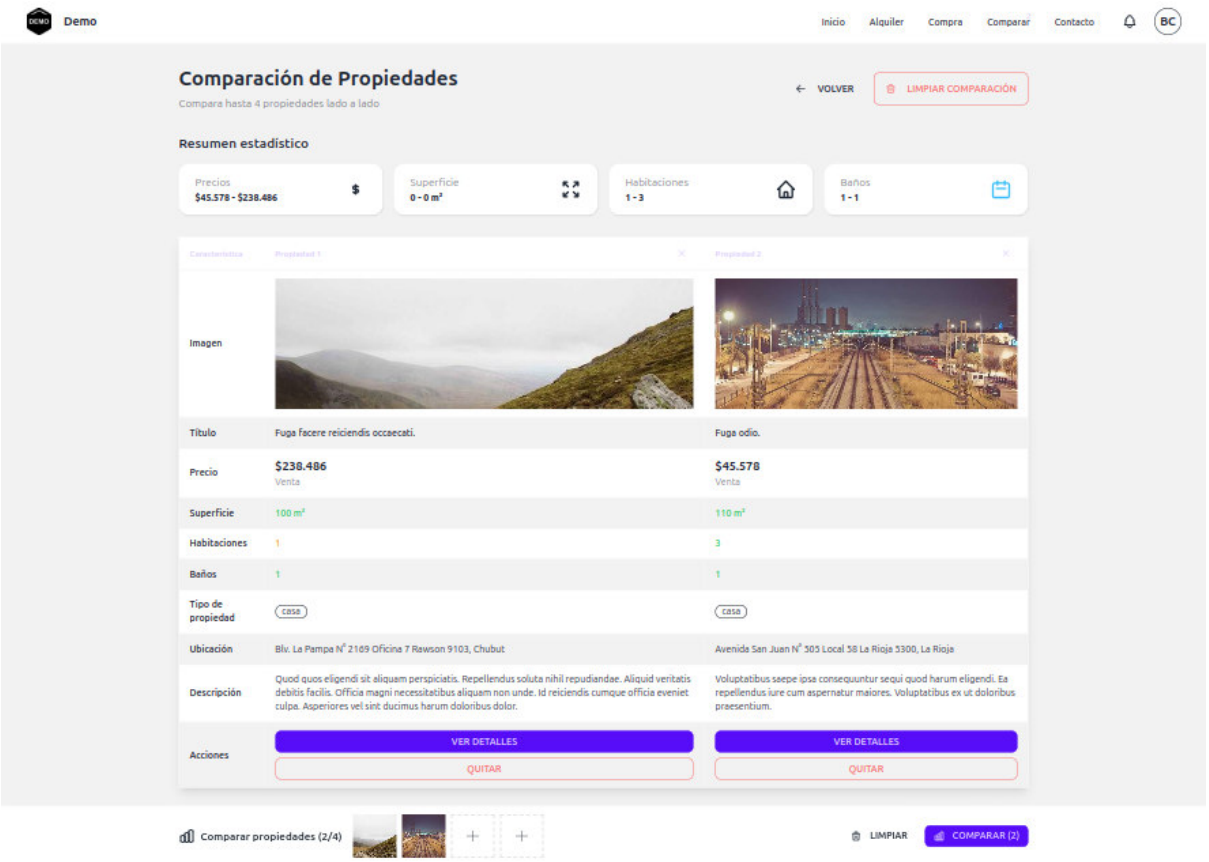


Figura 16. Sección para comparar diferentes propiedades.

Fuente: Elaboración propia, basada en la práctica.

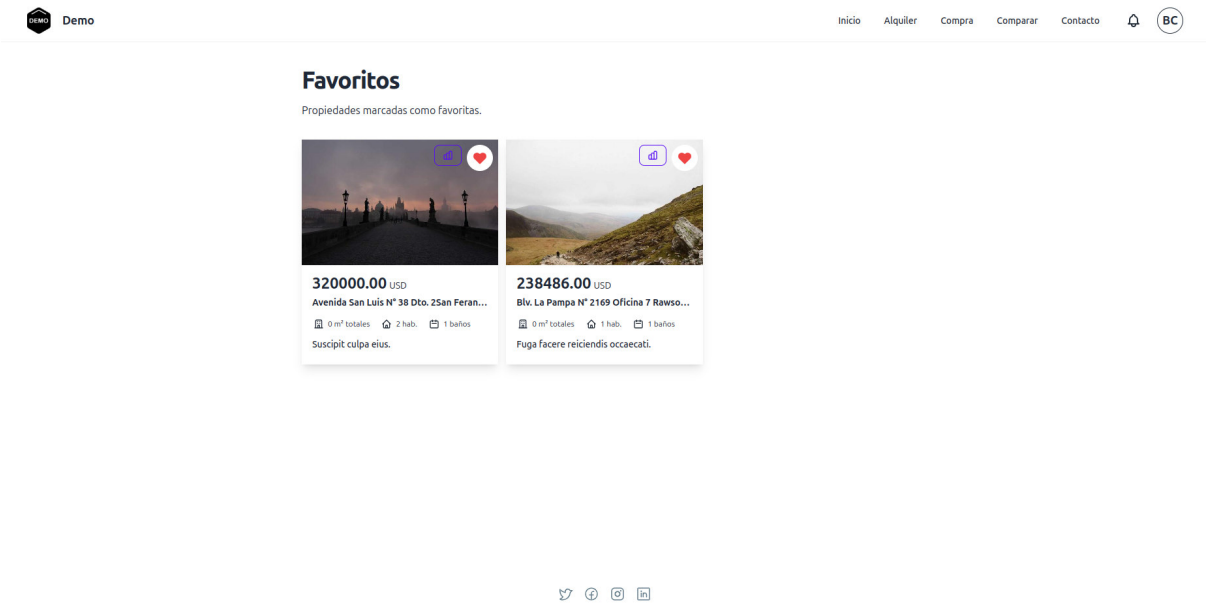


Figura 17. Sección para poder listar las propiedades marcadas como favoritas.

Fuente: Elaboración propia, basada en la práctica.

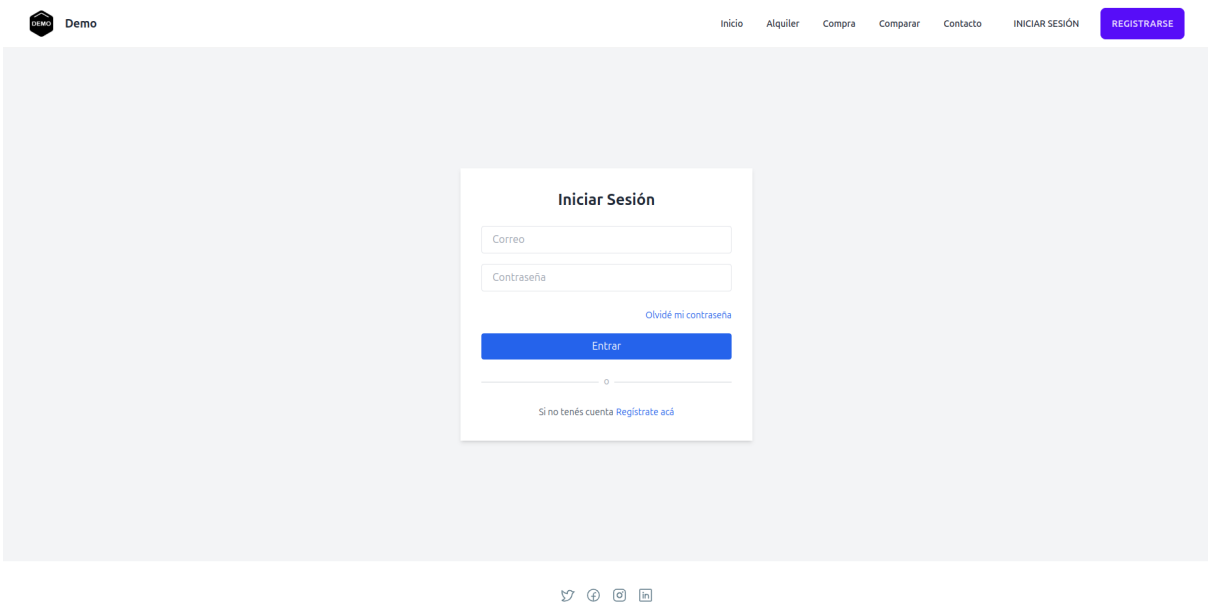


Figura 18. Formulario de inicio de sesión.

Fuente: Elaboración propia, basada en la práctica.

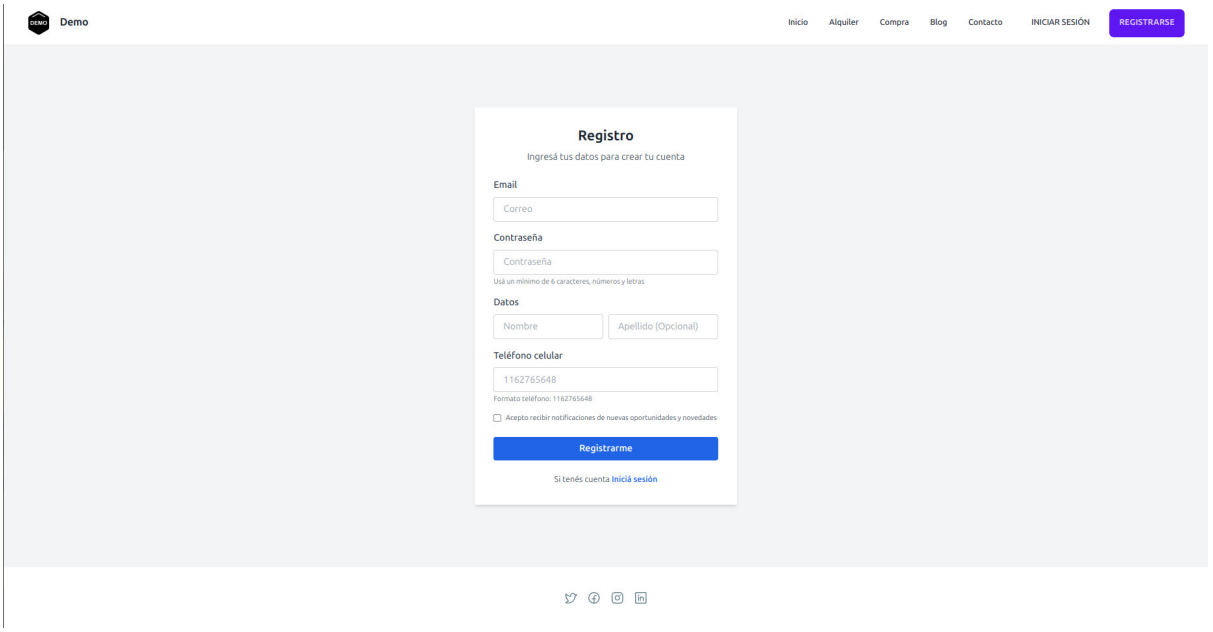


Figura 19. Formulario de registro para nuevos usuarios.

Fuente: Elaboración propia, basada en la práctica.

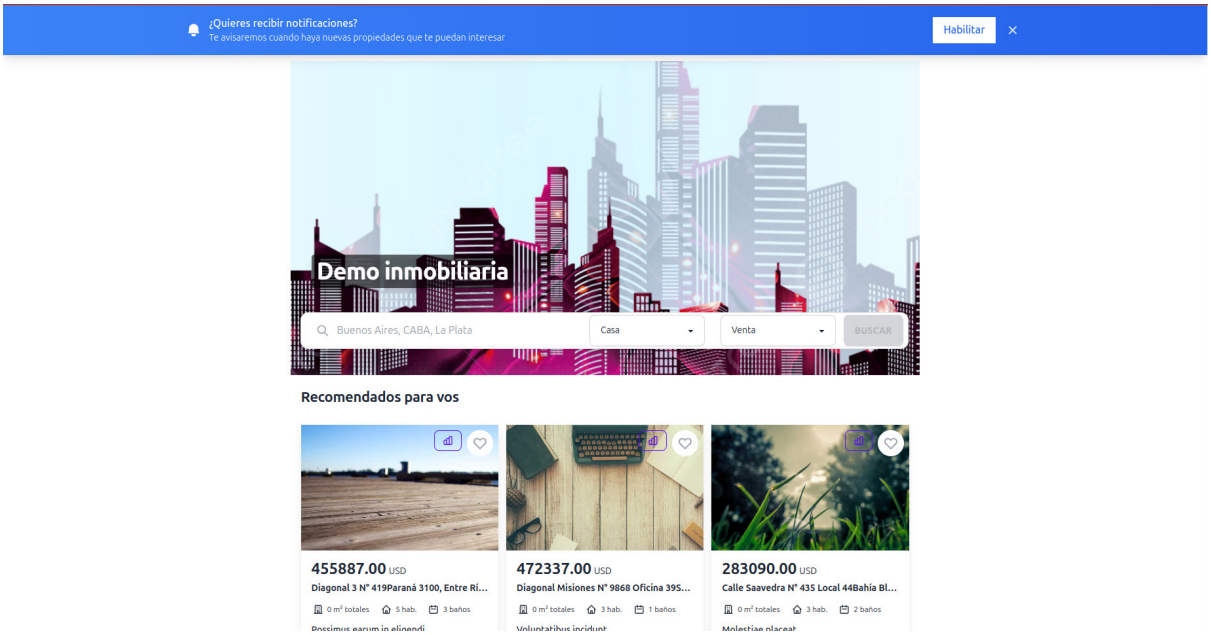


Figura 20. Notificación para habilitar notificaciones push para usuarios registrados.

Fuente: Elaboración propia, basada en la práctica.

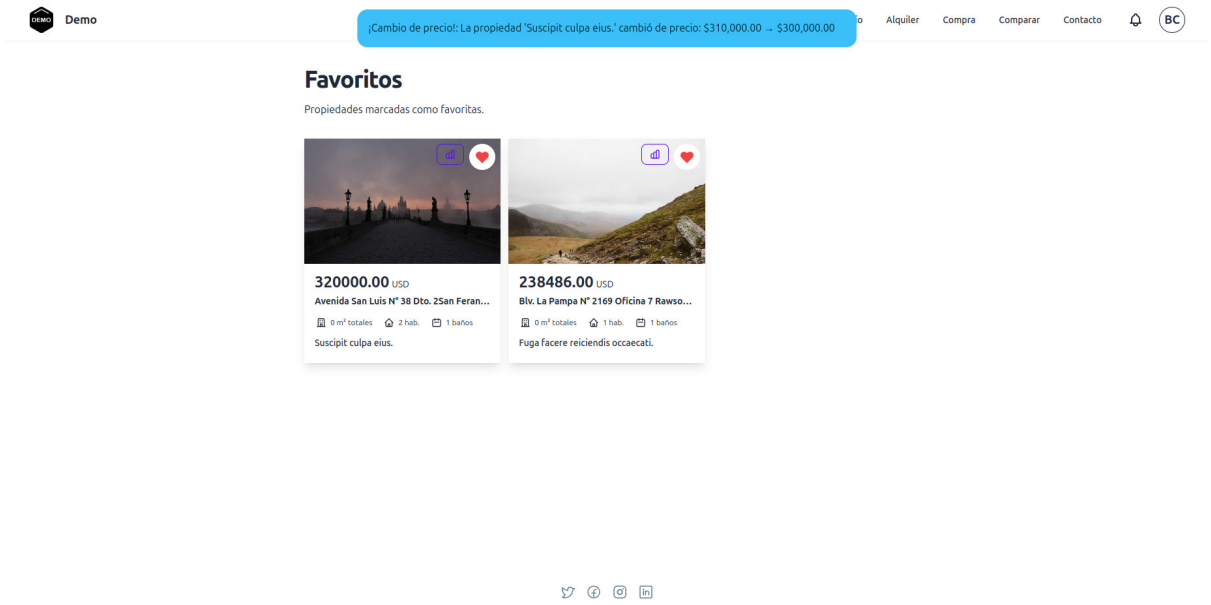


Figura 21. Notificación de actualización de precio.

Fuente: Elaboración propia, basada en la práctica.

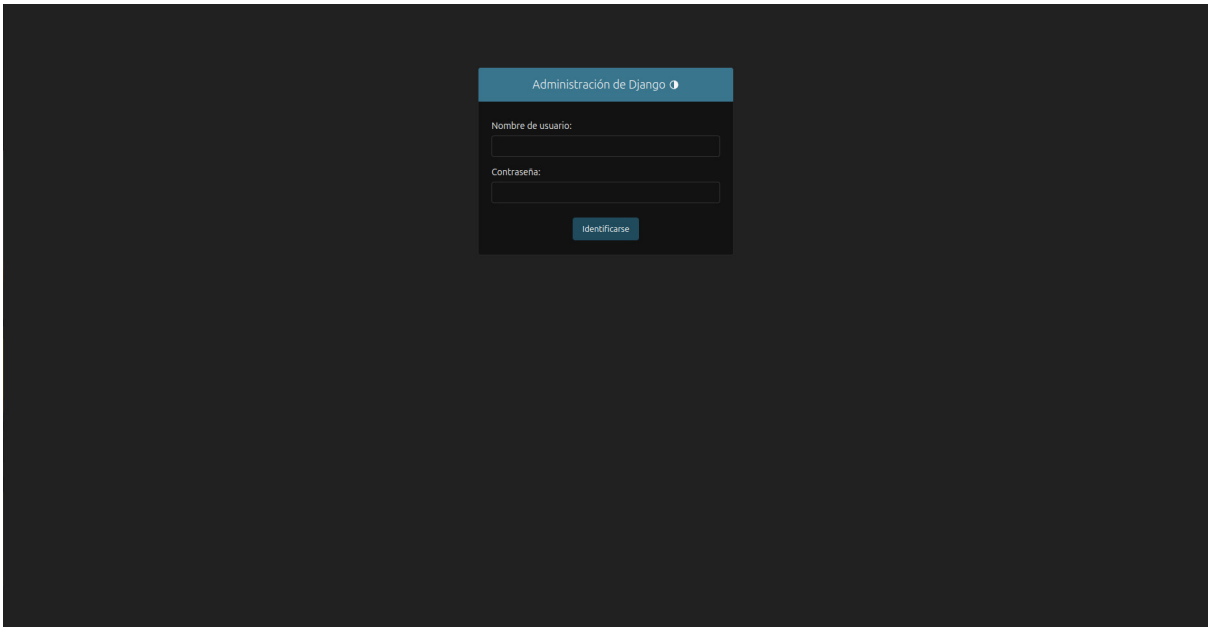


Figura 22. Pantalla de inicio de sesión del panel de administración.

Fuente: Elaboración propia, basada en la práctica.

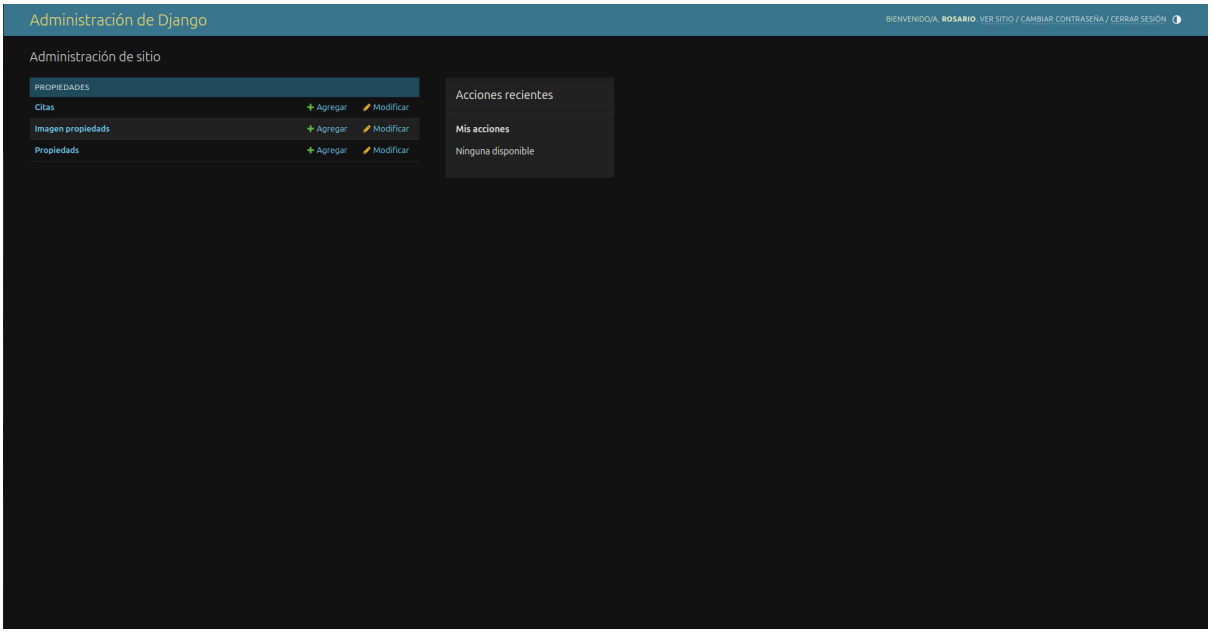


Figura 23. Panel de control para agentes inmobiliarios.

Fuente: Elaboración propia, basada en la práctica.

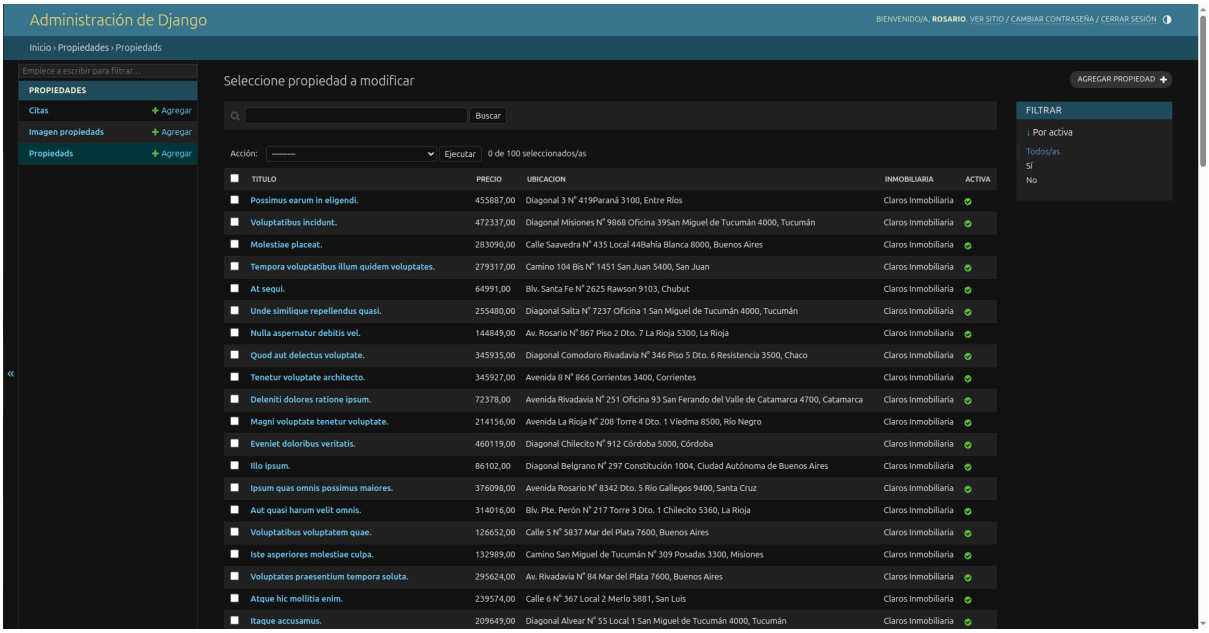


Figura 24. Sección de carga de lista y carga de propiedades.

Fuente: Elaboración propia, basada en la práctica.

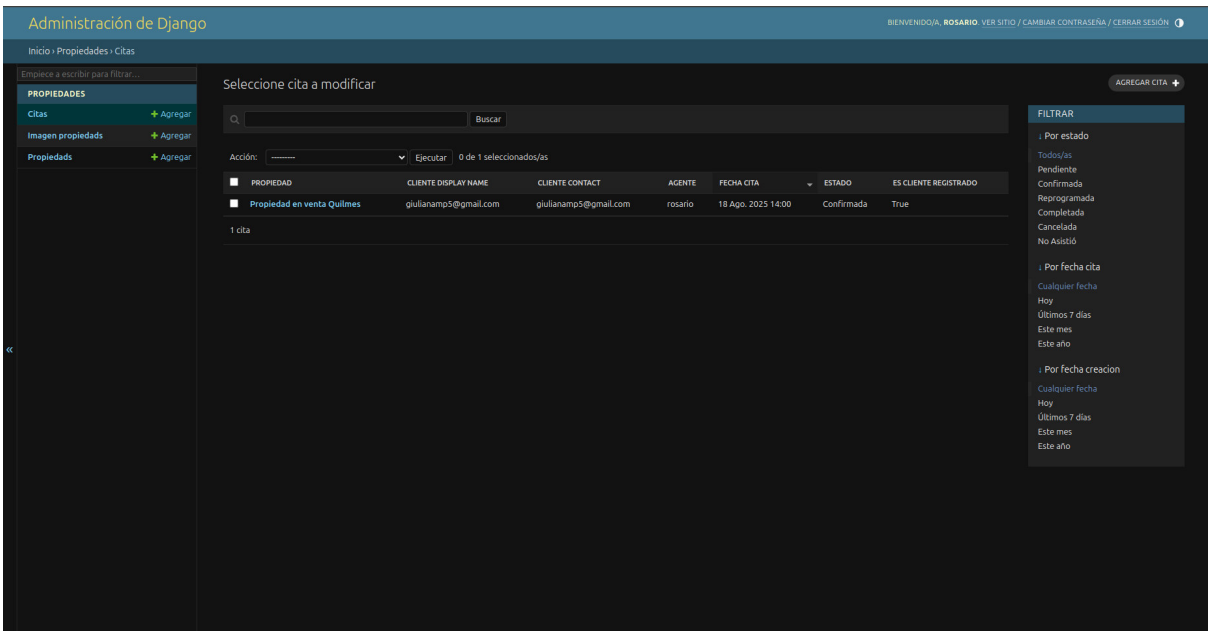


Figura 25. Sección para la gestión de citas.

Fuente: Elaboración propia, basada en la práctica.

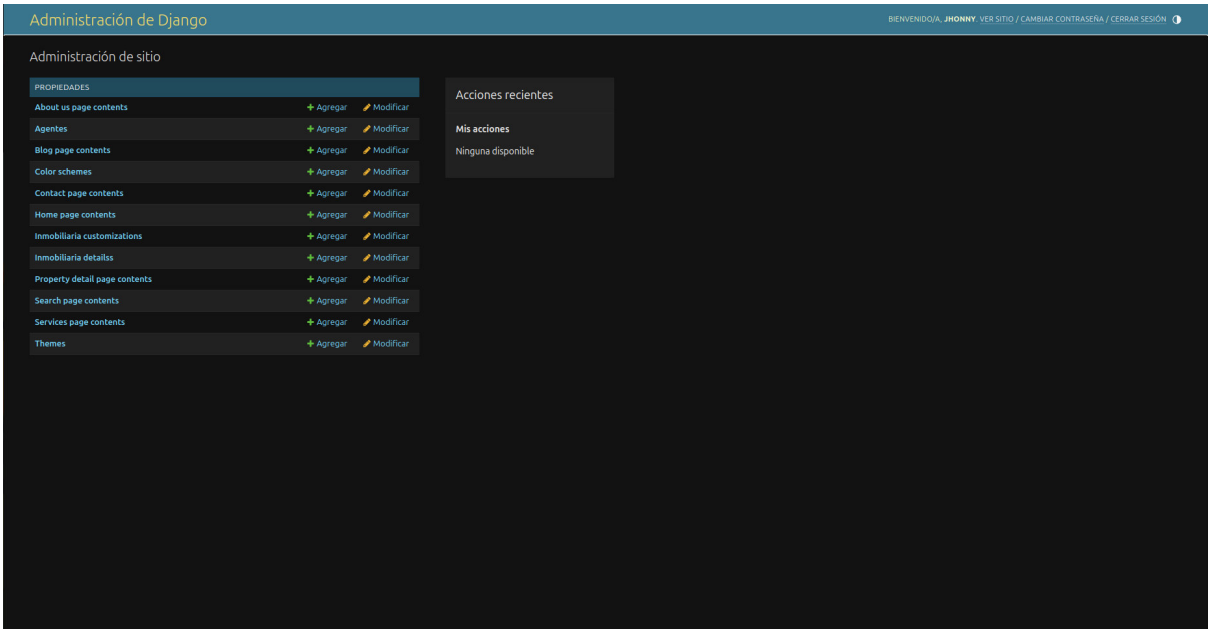


Figura 26. Panel de administración de un administrador de inmobiliaria con opciones disponibles para realizar modificaciones de la información de la inmobiliaria, gestión de agentes, gestión de contenido de páginas y estilos.

Fuente: Elaboración propia, basada en la práctica.

10.14. Reflexión

El diseño técnico del sistema permitió implementar una solución concreta, funcional y adaptable, alineada con los objetivos del MVP y con posibilidad de evolución futura. La elección de tecnologías, la arquitectura desacoplada y el enfoque multi-tenant no solo respondieron a las necesidades del cliente, sino que sentaron una base profesional sólida para futuras iteraciones del producto, mantenimiento escalable y adaptabilidad a nuevos contextos de negocio.

11. Propuesta de evolución futura del sistema

El desarrollo del MVP constituyó una base funcional sólida que permitió validar las principales características técnicas y operativas del sistema. Sin embargo, se definió desde el inicio como una primera versión, acotada en alcance y centrada en funcionalidades esenciales. A partir del feedback recibido, de la evolución natural de los productos SaaS, y de la experiencia obtenida durante la implementación, se plantean a continuación posibles líneas de evolución técnica y comercial del sistema:

11.1. Evolución funcional

- Diseño completamente responsivo: adaptación integral a móviles y tablets, manteniendo accesibilidad visual y usabilidad.
- Integración con pasarelas de pago (MercadoPago/Stripe): para permitir reservas de visitas, pagos de señas o alquileres.
- Sistema de estadísticas internas: con visualización de métricas (propiedades más visitadas, consultas enviadas, comportamiento de usuarios).
- Módulo de gestión de leads: con seguimiento de clientes interesados, embudos de conversión y contacto.
- Publicación automática en redes sociales: integración futura con APIs de Meta (Facebook, Instagram) y X (ex-Twitter) para compartir propiedades sin intervención manual, incluyendo plantillas personalizables y autenticación OAuth.

11.2. Evolución técnica y de arquitectura

- Separación en microservicios: para facilitar el mantenimiento, la escalabilidad horizontal y el despliegue modular.
- Contenedorización avanzada con Kubernetes: para entornos con múltiples inmobiliarias activas y balanceo de carga automático.
- Integración de un panel de administración externo (CMS): para permitir a usuarios no técnicos una mayor autonomía en la edición de contenidos.

11.3. Comercialización futura y empaquetado como producto

- Publicación del sistema como producto SaaS white-label, con onboarding guiado, precios escalonados y opción de soporte técnico.
- Desarrollo de una plataforma de gestión centralizada para el administrador del sistema, con control de múltiples clientes, dominios y configuraciones.

Estas líneas de evolución permitirán transformar el sistema actual en una plataforma integral, preparada para competir en el mercado como solución SaaS robusta, escalable y centrada en el usuario.

12. Consideraciones no previstas

Durante el desarrollo del proyecto surgieron diversas situaciones que no estaban contempladas en la planificación inicial. Algunas de ellas implicaron ajustes en el cronograma y en la distribución de tareas, mientras que otras fueron detectadas como oportunidades técnicas para futuras mejoras del sistema. Este tipo de imprevistos es

habitual en el desarrollo de productos reales, especialmente en contextos de validación temprana como el de un Producto Mínimo Viable (MVP), y su identificación oportuna permitió sostener la calidad del desarrollo sin comprometer los objetivos establecidos.
A continuación, se detallan las principales consideraciones no previstas, clasificadas por su naturaleza:

12.1. Ajustes técnicos y estratégicos

Reprogramación del diseño responsivo

Si bien en un principio se contempló desarrollar un frontend completamente responsivo, el cliente solicitó reprogramar esta funcionalidad para una segunda etapa del proyecto. Se priorizó en cambio la validación funcional del sistema en entornos de escritorio, acorde a los recursos disponibles y al objetivo principal del MVP.

Necesidad futura de integrar una pasarela de pagos

Durante las reuniones funcionales se identificó que, en etapas posteriores, el sistema podría beneficiarse con la incorporación de una pasarela de pagos, especialmente para permitir reservas online o cobros de alquileres. Esta funcionalidad quedó explícitamente fuera del alcance del MVP, pero su necesidad fue registrada para fases futuras.

Exploración técnica de soluciones de orquestación como Kubernetes

A medida que se avanzó en la arquitectura del sistema, se exploraron herramientas más avanzadas de despliegue y escalado, como Kubernetes. Aunque no fue implementado en esta etapa, se documentaron las bases para su eventual incorporación en versiones futuras con mayores requerimientos de escalabilidad.

12.2. Condicionamientos externos

Reducción de recursos técnicos por parte del cliente:

Por cuestiones presupuestarias externas al equipo de desarrollo, el cliente solicitó reducir la intensidad de trabajo planificada. Esto derivó en una extensión del cronograma original y en la reprogramación de algunas tareas de menor criticidad, sin afectar la entrega funcional del MVP.

12.3. Reprogramación de la automatización en redes sociales

Aunque inicialmente se había contemplado la automatización de publicaciones en redes sociales como Facebook e Instagram, esta funcionalidad fue diferida para una futura etapa del sistema. La decisión se tomó en conjunto con el cliente, priorizando funcionalidades clave del MVP que requerían mayor validación y menor integración con servicios externos.

12.4. Reflexión

Estas consideraciones no previstas reflejan la importancia de mantener una planificación flexible, adaptada a los cambios contextuales y a la validación progresiva del sistema. Lejos de representar obstáculos, estos elementos fueron gestionados activamente y permitieron mejorar el enfoque estratégico del desarrollo, reorganizando prioridades de manera realista y sentando las bases para futuras mejoras y expansiones del sistema.

13. Impacto de la práctica profesional supervisada

La realización de esta Práctica Profesional Supervisada (PPS) representó una instancia clave de integración de los conocimientos adquiridos durante la carrera y su aplicación concreta en un entorno de desarrollo real. A través del diseño, construcción, validación y documentación de un Producto Mínimo Viable (MVP) funcional, fue posible consolidar competencias técnicas, organizacionales y profesionales que forman parte esencial del perfil de un ingeniero en informática.

El proyecto se abordó de manera integral, desde el análisis de requerimientos hasta el despliegue final del sistema, incluyendo la toma de decisiones arquitectónicas, la integración de servicios externos, el diseño de una API REST, el desarrollo frontend, la implementación de pruebas, la automatización de procesos y la interacción continua con un cliente real. Esta experiencia permitió no solo aplicar tecnologías modernas como Django, Vue.js, Docker o GitHub Actions, sino también comprender su articulación en un ciclo completo de desarrollo de software.

12.1. Impacto en la formación como ingeniero en informática

Desde el punto de vista académico y formativo, la PPS brindó una oportunidad concreta para poner en práctica conceptos fundamentales de la carrera, como:

- Diseño de arquitecturas desacopladas y escalables.
- Modelado de datos relacionales y desarrollo de API RESTful.
- Implementación de CI/CD en entornos reales.
- Gestión de proyecto ágil con enfoque iterativo (Scrumban).

- Validación de requisitos con usuarios reales y ajustes sobre entregables funcionales.
- Documentación técnica, planificación con cronogramas y control del proceso de desarrollo.

Además, se fortalecieron habilidades transversales esenciales para el ejercicio profesional: planificación, organización del trabajo, toma de decisiones técnicas, trabajo en equipo, comunicación efectiva con el cliente, priorización de tareas y resolución de problemas bajo presión.

12.2. Impacto en la organización y en la solución entregada

El sistema desarrollado permitió a la organización contar con una herramienta funcional, escalable y reutilizable, orientada a gestionar propiedades de manera eficiente y moderna. La plataforma sentó las bases para una posible evolución comercial del producto y habilitó una solución adaptable a múltiples inmobiliarias mediante un modelo multi-tenant, sin necesidad de replicar instancias.

La posibilidad de validar esta herramienta en un entorno real, con interacción directa con el cliente y ajustes a partir del feedback recibido, generó un valor concreto tanto para la organización como para el proceso formativo del estudiante. Asimismo, el sistema desarrollado puede ser reutilizado o extendido en futuras etapas, ampliando el impacto más allá del marco académico.

Esta experiencia permitió conectar el rol del ingeniero con las necesidades reales del sector productivo. No se trató solo de cumplir con un proyecto académico, sino de entregar una solución profesional, técnicamente sólida y validada en el mundo real. La PPS no solo enriqueció mi formación, sino que me preparó con herramientas concretas para futuros desafíos profesionales, fortaleciendo mi perfil como desarrollador, analista y gestor de soluciones tecnológicas.

12.3. Evaluación del impacto económico, social y ambiental

El desarrollo del sistema tuvo un impacto positivo en múltiples dimensiones:

- Impacto económico: la plataforma ofrece una solución accesible y escalable para pequeñas y medianas inmobiliarias, que suelen carecer de recursos para implementar sistemas a medida. El modelo multi-tenant permite reducir significativamente los costos operativos, al reutilizar una misma instancia lógica del sistema para múltiples clientes, sin sacrificar personalización ni seguridad.
- Impacto social: al facilitar el acceso digital a ofertas de propiedades, con herramientas intuitivas y filtros precisos, la solución contribuye a mejorar la

experiencia de búsqueda de vivienda. Esto democratiza el acceso a la información y reduce barreras geográficas o tecnológicas, beneficiando tanto a los usuarios como a las inmobiliarias locales.

- Impacto ambiental: el sistema promueve una gestión 100% digital, reduciendo el uso de papel, folletos impresos y visitas innecesarias gracias a la visualización anticipada de las propiedades y la solicitud de turnos online. Asimismo, la digitalización del catálogo inmobiliario contribuye indirectamente a reducir la huella de carbono del proceso comercial tradicional.

14. Valoración crítica del trabajo realizado

La realización de este proyecto representó una experiencia integral que me permitió enfrentar un proceso completo de desarrollo de software, desde su concepción hasta su despliegue técnico en un entorno real. La posibilidad de liderar y coordinar el desarrollo de un Producto Mínimo Viable (MVP) funcional, con interacción directa con un cliente, me permitió integrar conocimientos técnicos, habilidades metodológicas y competencias profesionales que fueron evolucionando a lo largo de la carrera.

13.1. Fortalezas del proyecto

Uno de los principales logros fue la correcta identificación del alcance del MVP, lo cual permitió estructurar un plan de trabajo realista, centrado en funcionalidades clave con alto valor para el cliente. La adopción de una arquitectura desacoplada, el modelo multi-tenant y el uso de tecnologías modernas (Django, Vue.js, Docker, GitHub Actions) contribuyeron a la solidez del sistema y a su potencial escalabilidad.

Desde lo metodológico, la aplicación de un enfoque ágil híbrido (Scrumban) facilitó la organización del trabajo, la planificación por fases y la validación funcional continua. La implementación de CI/CD, la contenerización del sistema y la documentación técnica estructurada también reflejan buenas prácticas profesionales que resultaron fundamentales para el éxito del proyecto.

13.2. Dificultades y desafíos enfrentados

Uno de los principales desafíos fue la gestión del tiempo, especialmente en momentos en que debía asumir tareas clave sin la participación del segundo desarrollador. Esta situación requirió una redistribución efectiva del trabajo, una planificación más rigurosa y la toma de decisiones técnicas de forma autónoma.

Otro aspecto que presentó dificultades fue la necesidad de equilibrar el cumplimiento de los objetivos funcionales con las limitaciones de recursos y tiempo impuestas por el cliente. Esto implicó priorizar ciertos desarrollos, postergar funcionalidades no críticas (como el diseño responsivo o la pasarela de pagos), y sostener una comunicación clara para alinear expectativas.

13.3. Aprendizajes técnicos y profesionales

Esta experiencia me permitió consolidar habilidades técnicas como:

- Diseño y desarrollo de APIs RESTful con control de acceso y validación.
- Automatización de flujos CI/CD y contenedorización completa del sistema.
- Integración de servicios externos (Google Maps API, Firebase).
- Desarrollo frontend orientado a componentes reutilizables y comunicación asincrónica con backend.
- Planificación de tareas y seguimiento de proyectos en entornos reales.

Desde el punto de vista profesional, logré fortalecer competencias en gestión de proyectos, liderazgo técnico, comunicación efectiva con clientes y toma de decisiones bajo criterios de viabilidad, impacto y sostenibilidad del producto.

13.4. Áreas de mejora y oportunidades futuras

Si volviera a abordar este proyecto, planificaría con mayor anticipación la estructura de pruebas para el frontend, y dedicaría más tiempo a la definición de criterios de éxito para cada funcionalidad clave. También consideraría la implementación de métricas básicas de uso para validar el comportamiento del sistema en producción.

Asimismo, dejaría documentada una estrategia más formal de evolución del producto, incluyendo hitos funcionales futuros, pruebas automatizadas en frontend, implementación de diseño responsivo progresivo y monitoreo de métricas de salud del sistema.

El desarrollo de este proyecto no solo me permitió aplicar de forma integral lo aprendido en la carrera, sino que también me enfrentó a condiciones reales de trabajo en las que fue necesario priorizar, decidir y sostener técnicamente un producto con impacto real. La experiencia fortaleció mi perfil como desarrollador full stack, pero también como futuro profesional capaz de analizar, planificar y liderar soluciones tecnológicas que respondan a necesidades concretas del sector productivo.

15. Condiciones de seguridad e higiene

Dado que el desarrollo del proyecto se llevó a cabo en modalidad digital y remota, no se identificaron riesgos físicos ni sanitarios asociados al entorno de trabajo. No obstante, se adoptaron medidas de seguridad informática y buenas prácticas de desarrollo seguro, tales como:

- Uso de autenticación JWT para proteger las sesiones de usuario.
- Separación lógica de datos por cliente (multi-tenant) para evitar fugas de información.
- Configuración de firewall y cifrado de credenciales en el entorno de producción.
- Aislamiento de entornos mediante contenedores Docker, evitando conflictos entre servicios y facilitando el control de versiones.
- Gestión responsable del acceso al servidor y de las variables de entorno.

Estas prácticas garantizan no solo la integridad técnica del sistema, sino también un entorno de desarrollo seguro y profesional, acorde a los estándares de la industria.

16. Conclusión

La realización de esta Práctica Profesional Supervisada permitió integrar de forma concreta y sistemática los conocimientos adquiridos a lo largo de la carrera de Ingeniería en Informática, aplicándolos al desarrollo completo de un Producto Mínimo Viable (MVP) funcional, escalable y técnicamente sólido. A través de la construcción de una plataforma web orientada al sector inmobiliario, se abordaron de manera práctica múltiples dimensiones del proceso de desarrollo de software: Análisis de requerimientos, diseño arquitectónico, implementación backend y frontend, automatización del despliegue, pruebas, documentación y validación funcional con el cliente.

El proyecto permitió experimentar con herramientas y metodologías modernas en un entorno de trabajo real, fortaleciendo habilidades técnicas como la contenedorización con Docker, la automatización de flujos CI/CD con GitHub Actions, el diseño de API REST con Django, la estructuración modular del frontend con Vue.js y la integración de servicios externos como Google Maps API y Firebase. Asimismo, se profundizó en aspectos clave de la ingeniería del software como la organización de tareas, el trabajo por entregables, la trazabilidad de requisitos y la documentación estructurada.

Desde el punto de vista profesional, esta experiencia representó una oportunidad para validar no solo capacidades técnicas, sino también competencias transversales como la toma de decisiones autónoma, la comunicación efectiva con el cliente, la gestión de prioridades y la adaptación ante cambios e imprevistos. La PPS no solo facilitó el desarrollo

de una solución concreta con impacto real, sino que también consolidó una base técnica reutilizable, con potencial de evolución comercial y adaptación a otros escenarios.

Este proyecto no solo cumplió los objetivos establecidos, sino que también dejó planteadas futuras líneas de trabajo, como la incorporación de diseño responsivo, la integración de pasarelas de pago, el monitoreo en producción, el diseño iterativo de interfaz y el escalamiento sobre infraestructura distribuida. Considero que esta experiencia marcó una instancia de cierre significativa en mi formación como ingeniero en informática, y a la vez, un punto de partida para enfrentar nuevos desafíos profesionales con autonomía, criterio técnico y visión proyectual.

17. Bibliografía

- Bass, L., Clements, P., & Kazman, R. (2012). Software architecture in practice (3rd ed.). Addison-Wesley.
- Django Software Foundation. (s.f.). Django REST framework. Recuperado de https://www.django-rest-framework.org/ (Fecha de consulta: 3 de julio de 2025)
- Docker Inc. (s.f.). Docker documentation. Recuperado de https://docs.docker.com/ (Fecha de consulta: 25 de junio de 2025)
- Firebase. (s.f.). Firebase Cloud Messaging. Recuperado de https://firebase.google.com/docs/cloud-messaging (Fecha de consulta: 4 de julio de 2025)
- Fowler, M. (2002). Patterns of enterprise application architecture. Addison-Wesley.
- GitHub. (s.f.). GitHub Actions documentation. Recuperado de https://docs.github.com/en/actions (Fecha de consulta: 12 de junio de 2025)
- Google Developers. (s.f.). Maps JavaScript API documentation. Recuperado de https://developers.google.com/maps/documentation/javascript (Fecha de consulta: 8 de julio de 2025)
- ISO/IEC/IEEE 12207:2017. (2017). Systems and software engineering - Software life cycle processes. International Organization for Standardization. Recuperado de https://www.iso.org/standard/63712.html (Fecha de consulta: 17 de abril de 2025)
- ¿Qué es SMTP?. (s.f.). AWS. Recuperado de https://aws.amazon.com/what-is/smtp/ (Fecha de consulta: 18 de abril de 2025)
- ¿Qué es una API REST?. (s.f.). IBM. Recuperado de https://www.ibm.com/think/topics/rest-apis (Fecha de consulta: 18 de abril de 2025)
- Kniberg, H., & Skarin, M. (2010). Kanban and Scrum – making the most of both. C4Media.
- Ladas, C. (2009). Scrumban: Essays on Kanban Systems for Lean Software Development. Modus Cooperandi Press.
- Medium. (s.f.). Building Powerful Multi-Tenant SaaS Applications with Django: A Comprehensive Guide. Recuperado de https://medium.com/@pranavdixit20/building-powerful-multi-tenant-saas-applications-with-django-a-comprehensive-guide-2a3206bde31f (Fecha de consulta: 30 de junio de 2025)

- PostgreSQL Global Development Group. (s.f.). PostgreSQL documentation. Recuperado de https://www.postgresql.org/docs
  (Fecha de consulta: 5 de julio de 2025)
- Pressman, R. S., & Maxim, B. R. (2020). Ingeniería del software: Un enfoque práctico (8.ª ed.). McGraw-Hill.
- Vue.js. (s.f.). Vue.js documentation. Recuperado de https://vuejs.org/
  (Fecha de consulta: 27 de abril de 2025)

18. Glosario

- **API REST**: estilo de arquitectura para la comunicación entre sistemas mediante el protocolo HTTP, siguiendo principios como recursos identificables por URI y operaciones mediante métodos estándar (GET, POST, PUT, DELETE).
- **Arquitectura Multi-tenant**: modelo de arquitectura en el cual una única instancia lógica del sistema puede ser utilizada por múltiples clientes (tenants), con separación lógica de datos y configuraciones. Es ideal para soluciones SaaS por su eficiencia operativa.API REST
- **Ciclo de vida del desarrollo de software**: conjunto de etapas estructuradas que guían el proceso de creación de un sistema, desde el análisis de requerimientos hasta su mantenimiento. Puede implementarse de forma secuencial (modelo en cascada) o iterativa e incremental (como en metodologías ágiles).
- **CI/CD (Integración Continua / Despliegue Continuo)**: estrategia de automatización en el desarrollo de software que permite ejecutar pruebas, compilar código y desplegar versiones del sistema de forma continua y segura.
- **Desacoplamiento**: principio de diseño de software que busca reducir la dependencia entre componentes, permitiendo su desarrollo, mantenimiento y escalado de forma independiente. Mejora la modularidad y la flexibilidad del sistema.
- **Django**: framework web de alto nivel basado en Python que promueve el desarrollo rápido, limpio y seguro de aplicaciones web, integrando ORM, autenticación, panel de administración y soporte para APIs.
- **Docker**: plataforma que permite crear contenedores para ejecutar aplicaciones junto con sus dependencias de forma aislada, portable y replicable, facilitando el despliegue en distintos entornos.
- **Endpoint**: punto de acceso a una API que representa una operación específica. Es la URL donde el cliente (por ejemplo, el frontend) interactúa con el backend para enviar o recuperar datos.
- **Firebase Cloud Messaging (FCM)**: servicio de Google que permite enviar notificaciones push en tiempo real a navegadores o dispositivos móviles, facilitando la comunicación directa con el usuario.
- **Instancia**: ejecución específica de una aplicación o sistema. En el contexto del desarrollo web, puede referirse a una copia activa del software desplegada para uno o varios clientes.

- **JWT (JSON Web Token)**: formato estándar utilizado para transmitir información de forma segura entre partes como token de autenticación. En sistemas modernos, permite validar identidades sin mantener sesiones en el servidor.
- **MVP (Minimum Viable Product)**: versión funcional mínima de un producto que contiene las características esenciales para ser validada por usuarios reales, permitiendo recoger retroalimentación temprana.
- **ORM (Object-Relational Mapping)**: técnica que permite mapear objetos del lenguaje de programación con tablas de una base de datos relacional. Facilita las operaciones CRUD sin escribir consultas SQL manuales.
- **Scrumban**: metodología ágil híbrida que combina elementos de Scrum (como la planificación por funcionalidades) y Kanban (visualización del flujo de trabajo continuo). Ideal para equipos reducidos y proyectos en entornos variables.
- **Tenant**: unidad lógica que representa a un cliente dentro de un sistema multi-tenant. Cada tenant opera con datos, configuraciones y usuarios propios, aunque comparta la misma infraestructura general.
- **Token**: cadena de caracteres que representa la autenticación de un usuario en una sesión. Puede ser almacenada y enviada para verificar la identidad del usuario en cada solicitud al sistema.
- **Vue.js**: framework progresivo de JavaScript para construir interfaces de usuario dinámicas y reactivas. Utiliza una arquitectura basada en componentes, ideal para aplicaciones de una sola página (SPA).