INSTITUT
PUIG CASTELLAR
SANTA COLOMA DE GRAMENET

closely
marketplace

Plataforma de compra y venta local

Projecto de Desarrollo
CFGS Desarrollo de Aplicaciones Web

Tutor: David Delgado Santamaria

Fecha: 16/05/2026

Matias Ezequiel Obando Grajeda
Japhet Esau Lazo Garcia
Daw 2 Grupo A

Licencia

Autores:

Matías Ezequiel Obando Grajeda

Japhet Esau Lazo García

Resumen del projecte

La plataforma Closely Marketplace facilita la compraventa local de artículos de segunda mano mediante una interfaz web responsiva que conecta vendedores y compradores. El objetivo principal es simplificar la publicación y gestión de anuncios, mejorar la búsqueda por categorías y filtros, y ofrecer mensajería directa segura entre usuarios para concretar ventas. El desarrollo siguió un enfoque incremental: diseño de interfaz y experiencia, implementación de autenticación y CRUD de anuncios, motor de búsqueda y filtros, y sistema de mensajería. El frontend está desarrollado con Nuxt y el backend con Spring Boot (API REST) integrando una base de datos tipo PostgreSQL/Supabase para persistencia. Se realizaron pruebas funcionales y de usabilidad en dispositivos móviles y escritorio. Como resultado se obtuvo un prototipo funcional y operativo que permite publicar artículos, editar y gestionar inventario, buscar por criterios y chatear entre usuarios; cumple los requisitos funcionales y mejora la experiencia de compra local. Trabajo futuro: integrar pasarela de pago, valoraciones y panel de gestión más avanzado.

Palabras clave

- Comercio local
- Marketplace de segunda mano
- Publicaciones y gestión
- Mensajería entre usuarios
- Búsqueda y filtros
- Nuxt
- Spring Boot
- Supabase / PostgreSQL

Abstract

Closely Marketplace is a responsive web platform for local second-hand trading that connects sellers and buyers. Its main goal is to streamline listing management, enhance search with filters and categories, and provide secure direct messaging for transactions. Development followed an incremental approach: UI/UX design, authentication and CRUD, search functionality, and messaging. The frontend uses Nuxt and the backend Spring Boot with a PostgreSQL/Supabase database. Functional and usability tests were performed on mobile and desktop. The resulting functional prototype allows creating and managing listings, searching by criteria, and user-to-user chat, meeting the project goals. Future work includes payment integration, ratings, and an advanced admin panel.

Keywords

- Local commerce
- Second-hand marketplace
- Listings management
- User messaging
- Search & filters
- Nuxt
- Spring Boot
- Supabase / PostgreSQL

Índice

1. Presentación del proyecto

1.1. Introducción

El presente proyecto, Closely Marketplace, consiste en el desarrollo de una plataforma web destinada a facilitar la compraventa local de artículos de segunda mano. Surge de la necesidad de mejorar la experiencia de publicación y búsqueda de anuncios en entornos cercanos, reducir la fricción en la comunicación entre vendedor y comprador, y ofrecer una alternativa más segura y ordenada al uso de canales dispersos. El objetivo principal es proporcionar un prototipo funcional que permita crear, editar y gestionar anuncios, buscar por categorías y filtros, y comunicarse mediante mensajería privada para cerrar transacciones.

El trabajo se ha abordado de forma incremental: análisis de requisitos y diseño de la interfaz, implementación del sistema de autenticación y CRUD de anuncios, incorporación de búsqueda y filtros, y añadido del sistema de mensajería. El frontend se ha implementado con Nuxt y el backend con Spring Boot, integrando una base de datos en Supabase/PostgreSQL. Se han realizado pruebas funcionales y de usabilidad en dispositivos móviles y escritorio. El documento describe el planteamiento del proyecto, el análisis del problema, las decisiones de diseño, la arquitectura técnica, las principales funcionalidades implementadas y las pruebas realizadas. Finalmente se presentan conclusiones y propuestas de mejora futura, entre las que destacan pasarela de pago, sistema de valoraciones y panel administrativo avanzado.

1.2. Contexto

El sector del comercio local de segunda mano está creciendo impulsado por la búsqueda de consumo sostenible, la economía colaborativa y la facilidad de uso de dispositivos móviles. Actualmente, muchas transacciones entre particulares se realizan en grupos de redes sociales, aplicaciones de mensajería o plataformas generalistas que no están optimizadas para la gestión de anuncios locales: la información queda dispersa, faltan herramientas para filtrar por proximidad o estado del artículo y la comunicación entre comprador y vendedor suele ser poco estructurada. Además, algunas soluciones comerciales están orientadas a grandes volúmenes o requieren suscripciones, lo que puede no resultar rentable para usuarios particulares o pequeños comercios.

En este contexto surge Closely Marketplace: una propuesta para ofrecer una plataforma web accesible y ligera que centralice la publicación de anuncios, facilite la búsqueda por categorías y filtros locales, y proporcione mensajería integrada para acordar transacciones. La solución se orienta a usuarios que valoran la proximidad y la simplicidad (particulares y micro comercios), y busca cubrir la falta de herramientas específicas para mercados locales sin imponer costes elevados.

1.3. Justificación

El proyecto Closely Marketplace se justifica por la ausencia de herramientas accesibles y específicas para la compraventa local que faciliten publicar, buscar y negociar artículos de segunda mano de forma ordenada y segura. Hoy la mayor parte de esas transacciones se gestionan en redes sociales o chats dispersos, lo que provoca pérdida de información, dificultad para filtrar por ubicación y estado, y comunicación poco estructurada entre las partes. Las soluciones comerciales existentes suelen estar orientadas a grandes volúmenes o implican costes que no resultan adecuados para particulares o pequeños comercios.

Desarrollar Closely Marketplace aporta valor práctico: centraliza anuncios, mejora la búsqueda por criterios locales, facilita la gestión personal de publicaciones y ofrece mensajería integrada para acordar ventas con trazabilidad. Además, desde el punto de vista formativo permite aplicar y consolidar competencias técnicas en desarrollo web (frontend con Nuxt, backend con Spring Boot, persistencia en Supabase/PostgreSQL) y en diseño de interfaces orientadas a la usabilidad móvil. En resumen, el proyecto resuelve una necesidad real con una solución asequible y pedagógicamente relevante.

1.4. Objetivos

Objetivo general: Desarrollar un prototipo funcional de Closely Marketplace que permita publicar, buscar y gestionar anuncios de compraventa local y facilitar la comunicación segura entre compradores y vendedores.

Objetivos específicos:

- Implementar un sistema de registro, autenticación y gestión de perfiles de usuario.
- Permitir la creación, edición y eliminación de anuncios con fotos y metadatos (categoría, estado, precio).
- Desarrollar funcionalidades de búsqueda y filtros por categoría, precio, estado.
- Implementar un sistema de mensajería privada entre usuarios para coordinar transacciones.
- Garantizar la usabilidad y accesibilidad en dispositivos móviles.
- Persistir datos en Supabase/PostgreSQL y ofrecer una API REST en Spring Boot consumida por el frontend Nuxt.
- Realizar pruebas funcionales y pruebas de usabilidad para validar el flujo principal.

2. Estrategia y planificación del proyecto

2.1. Estrategia de desarrollo y viabilidad

El proyecto se plantea como el desarrollo de una plataforma web desde cero Closely Marketplace destinada a la compraventa local de artículos de segunda mano. El objetivo es construir un prototipo funcional que permita publicar, buscar y gestionar anuncios y comunicar vendedores y compradores mediante mensajería integrada.

Punto de partida
La aplicación se desarrollará completamente desde cero, sin reutilizar un producto comercial existente. Se parte de las especificaciones iniciales y prototipos creados por el equipo; el repositorio actual contendrá el nuevo código del frontend (Nuxt) y del backend (Spring Boot) que se implementarán durante el proyecto. La base de datos prevista será Supabase/PostgreSQL y se planifica el uso de Supabase Storage para imágenes.

Estrategia de desarrollo

Se adopta un desarrollo progresivo y modular, implementando funcionalidades por fases y validando cada bloque antes de avanzar:

1. Autenticación y gestión de perfiles (registro/login, validaciones).
2.Gestión de anuncios (crear/editar/borrar, subida de imágenes, validaciones).
3.Búsqueda y filtros (texto, categoría, rango de precio).
4.Mensajería privada asociada a anuncios.
5.Ajustes de UI/UX y accesibilidad; pruebas de usabilidad.
5.Despliegue y documentación.

Cada fase incluirá un prototipo funcional, tests básicos y una checklist de aceptación; el trabajo se organizará en sprints cortos (1–2 semanas) y el progreso se gestionará en Trello con PRs vinculadas a tarjetas.

Viabilidad del proyecto

El desarrollo desde cero es viable si se respeta la priorización del prototipo funcional(puntos 1–4) y se evita incorporar funcionalidades avanzadas en fases tempranas. Las tecnologías elegidas (Nuxt, Spring Boot, Supabase) son conocidas y permiten despliegues rápidos en plataformas gestionadas. El enfoque incremental posibilita entregar versiones funcionales tempranas para pruebas reales y feedback.

Riesgos previstos y medidas de mitigación

Las posibles dificultades principales son: problemas al subir o almacenar imágenes, retrasos en la integración entre frontend y backend, fallos en el sistema de acceso de usuarios y falta de tiempo para hacer pruebas reales. Para mitigarlos se usará un servicio de almacenamiento fiable, se trabajará con contratos y pruebas sencillas para evitar bloqueos entre partes, se emplearán soluciones de autenticación ya probadas y se reservará un sprint para pruebas de usabilidad con al menos 5 usuarios.

2.2. Metodología de trabajo

Para el desarrollo de Closely Marketplace se seguirá una metodología práctica y sencilla, centrada en entregar un prototipo funcional cuanto antes y mejorar por iteraciones.

Enfoque de trabajo

El proyecto se desarrollará de forma progresiva por fases: se implementan primero las funcionalidades esenciales y, una vez validadas, se añaden mejoras. Se prioriza siempre disponer de una versión funcional en cada etapa para poder probarla y recibir feedback.

Organización del trabajo

El trabajo se divide en bloques funcionales: usuarios, anuncios, búsqueda, mensajería y ajustes UI/UX. Cada bloque se descompone en tareas pequeñas que se registran en **Trello** y que contienen una descripción breve y criterios de aceptación. Antes de integrar cualquier bloque se verifica su correcto funcionamiento mediante pruebas básicas y una demo corta.

Herramientas de seguimiento

El tablero principal será **Trello** con listas: Pendiente, En proceso, Revisado y Hecho. Se usarán etiquetas para prioridad y componente (frontend/backend/UI UX). El código se almacenará en **GitHub** y las PRs se enlazará a las tarjetas correspondientes.

2.3. Planificación

El proyecto se organizará en seis fases que permiten avanzar de forma progresiva desde la definición inicial hasta una versión funcional completa.

Fase 1 · Análisis y diseño inicial

En esta fase se define el alcance del proyecto y se concreta qué funcionalidades tendrá el prototipo funcional. También se realiza una propuesta básica de la interfaz y se organiza la estructura general del sistema. El objetivo de esta fase es dejar claro qué se va a construir y cómo se va a presentar.

Fase 2 · Preparación del entorno de trabajo

Se prepara el repositorio, se organiza la estructura del proyecto y se configura el entorno necesario para empezar a desarrollar con orden. El objetivo es disponer de una base de trabajo lista para ir incorporando las distintas partes del sistema.

Fase 3 · Desarrollo de las funcionalidades principales

Se implementan las funciones más importantes del proyecto, como el acceso de usuarios, la publicación de artículos y la gestión básica de anuncios. El objetivo de esta fase es conseguir una primera versión funcional del sistema.

Fase 4 · Desarrollo de funcionalidades secundarias

Una vez cubiertas las funciones principales, se añaden el resto de apartados, como la búsqueda avanzada, los filtros, la mensajería y otras mejoras que completan la experiencia de uso.

Fase 5 · Pruebas y corrección de errores

En esta fase se revisa que todo funcione correctamente, se corrigen los fallos detectados y se comprueba que la aplicación responde bien tanto en ordenador como en móvil.

Fase 6 · Mejoras finales y preparación de la entrega

Por último, se realizan los ajustes visuales finales, se revisa la documentación y se prepara la entrega del proyecto con una versión limpia y ordenada.

Distribución temporal

Fase	Duración estimada
Fase 1 · Análisis y diseño inicial	2 semanas
Fase 2 · Preparación del entorno de trabajo	1 semanas
Fase 3 · Desarrollo de funcionalidades principales	6 semanas
Fase 4 · Desarrollo de funcionalidades secundarias	4 semanas
Fase 5 · Pruebas y corrección de errores	2 semanas
Fase 6 · Mejoras finales y preparación de la entrega	1 semanas

*Esta planificación es orientativa y puede ajustarse durante el desarrollo en función del avance real y de las dificultades que vayan apareciendo.

3. Planificación

3.1. Casos de uso

El sistema tiene como actores principales al usuario no registrado, que puede consultar artículos y ver información general, al usuario registrado, que puede publicar y gestionar sus anuncios, y al administrador, que supervisa el contenido y el funcionamiento general de la plataforma. A continuación se documentan los casos de uso más representativos del sistema.

Formato narrativo

CU-01 · Publicar un anuncio

Actor: Usuario registrado

Descripción: El usuario introduce la información de un artículo que quiere vender y lo publica en la plataforma.

Flujo principal:

1.El usuario accede a la opción de publicar.

2.Rellena el título, la descripción, la categoría, el precio y las imágenes del artículo.

3.El sistema comprueba que los datos son correctos.

4.El usuario confirma la publicación.

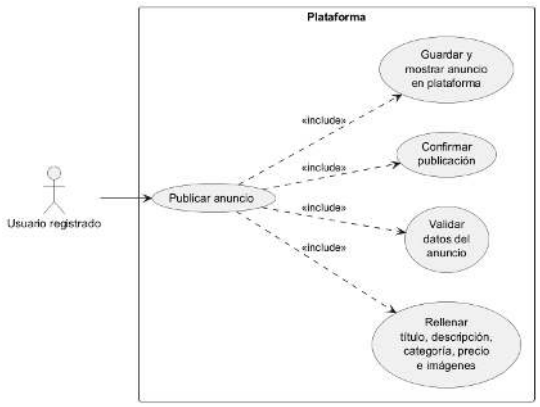
5.El sistema guarda el anuncio y lo muestra en la plataforma.

Flujo alternativo:

-Si algún campo obligatorio está vacío, el sistema avisa al usuario y no permite continuar.

-Si el usuario decide cancelar, vuelve a la pantalla anterior sin guardar cambios.

Resultado: El anuncio queda publicado y visible para otros usuarios.



CU-02 · Contactar con un vendedor

Actor: Usuario registrado

Descripción: El usuario consulta un anuncio y envía un mensaje al vendedor para pedir información o cerrar la compra.

Flujo principal:

1. El usuario entra en la ficha de un artículo.

2.Selecciona la opción de contactar.

3.Escribe su mensaje y lo envía.

4.El sistema guarda la conversación.

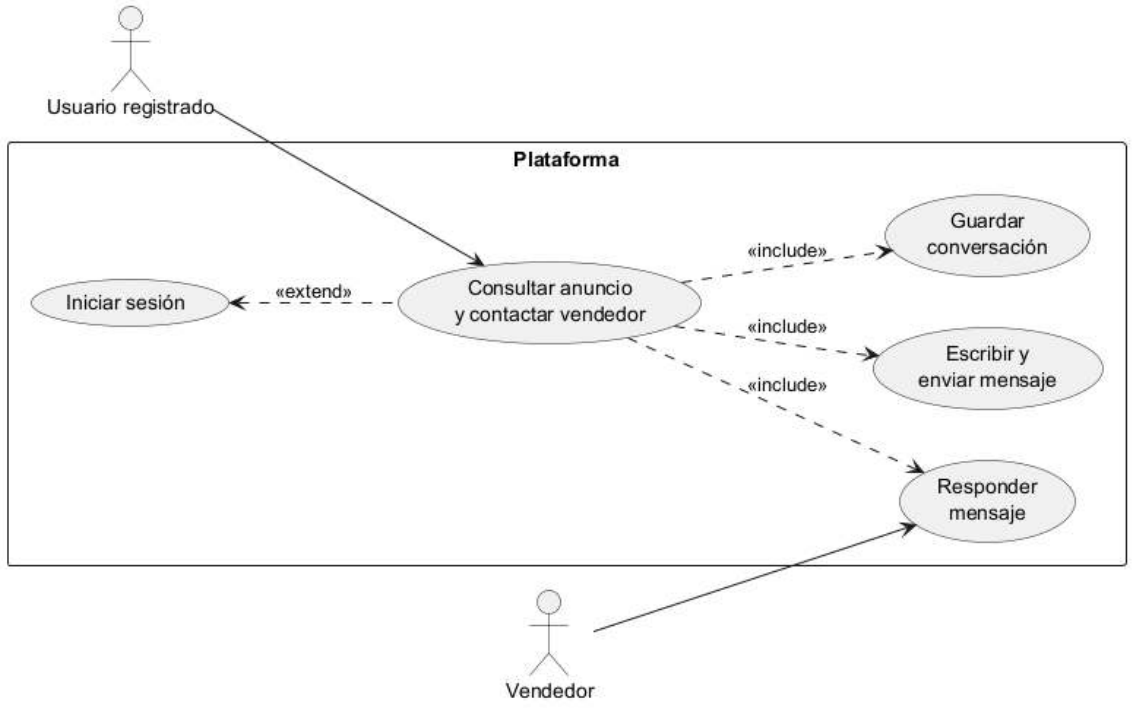
5.El vendedor recibe el mensaje y puede responder.

Flujo alternativo:

-Si el usuario no ha iniciado sesión, el sistema le pide que acceda antes de continuar.

-Si el vendedor no responde de inmediato, el mensaje queda guardado en el historial.

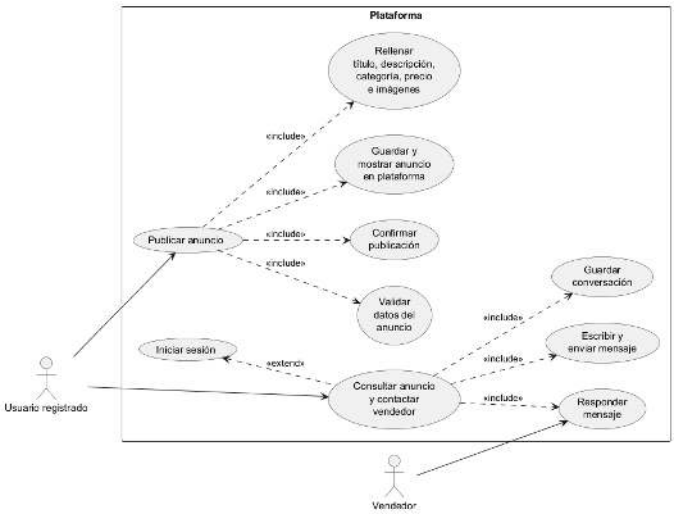
Resultado: Se crea una conversación entre comprador y vendedor.



Formato tabla estructurada

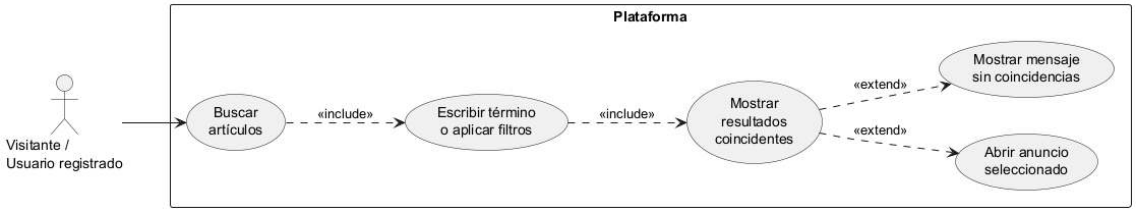
CU-03 · Gestionar mis anuncios

Campo	Descripción
Identificador	CU-03
Nombre	Gestionar mis anuncios
Actor	Usuario registrado
Precondiciones	El usuario ha iniciado sesión y tiene al menos un anuncio publicado.
Flujo principal	1. El usuario accede a “Mis publicaciones”. 2. Selecciona el anuncio que quiere modificar. 3. El sistema muestra los datos actuales. 4. El usuario edita la información o elimina el anuncio. 5. El sistema guarda los cambios y actualiza la lista.
Flujo alternativo	Si el usuario no quiere guardar los cambios, vuelve a la pantalla anterior sin modificar nada.
Postcondiciones	El anuncio queda actualizado o eliminado según la acción realizada.



CU-04 · Buscar artículos

Campo	Descripción
Identificador	CU-04
Nombre	Buscar artículos
Actor	Visitante o usuario registrado
Precondiciones	El sistema tiene anuncios publicados.
Flujo principal	1. El usuario accede al buscador. 2. Escribe un término o aplica filtros. 3. El sistema muestra los resultados que coinciden. 4. El usuario abre el anuncio que le interesa.
Flujo alternativo	Si no hay resultados, el sistema muestra un mensaje indicando que no se han encontrado coincidencias.
Postcondiciones	El usuario obtiene una lista de artículos filtrada según sus criterios.



3.2. Requisitos funcionales

La aplicación debe ofrecer las funcionalidades que permitan a usuarios publicar, buscar y negociar la compraventa local de artículos. A continuación se presentan los requisitos funcionales principales organizados por prioridad.

ID	Requisito	Prioridad
RF01	El sistema permitirá a los usuarios registrarse y autenticarse con correo y contraseña	Alta
RF02	El usuario podrá editar su perfil (nombre, foto, descripción, datos de contacto)	Media
RF03	El usuario registrado podrá crear un anuncio con título, descripción, categoría, precio, ubicación aproximada y al menos una imagen	Alta
RF04	El usuario podrá editar y eliminar sus anuncios	Alta
RF05	El sistema permitirá listar anuncios por página y mostrar la ficha detallada de cada anuncio	Alta
RF06	El sistema ofrecerá búsqueda por texto y filtros por categoría, rango de precio, estado y proximidad	Alta
RF07	El usuario podrá marcar anuncios como favoritos y acceder a su lista de favoritos	Media
RF08	El sistema incluirá un sistema de mensajería privada entre comprador y vendedor asociado a cada anuncio	Alta
RF09	El sistema enviará notificaciones (internas o por correo) ante mensajes nuevos y cambios relevantes en anuncios propios	Media
RF10	El usuario podrá marcar un anuncio como reservado o vendido (estado) y el sistema actualizará la visibilidad	Alta
RF11	El sistema permitirá subir imágenes y gestionarlas (miniatura, borrado), con límites de tamaño	Alta
RF12	El administrador podrá moderar y eliminar anuncios que incumplan normas y gestionar usuarios (bloqueo)	Media
RF13	El sistema guardará un historial básico de conversaciones y cambios en anuncios (registro mínimo)	Media

*Los requisitos con prioridad "Alta" constituyen el núcleo funcional necesario para entregar el prototipo funcional; los de prioridad "Media" completan y mejoran la experiencia y pueden implementarse tras asegurar las funcionalidades core.

3.3. Requisitos no funcionales

Además de las funcionalidades descritas, la aplicación debe cumplir una serie de condiciones de calidad que afectan a su comportamiento general. Estos requisitos no definen qué hace la aplicación, sino cómo debe hacerlo, y su cumplimiento es tan importante como el de los requisitos funcionales para considerar el proyecto correcto y usable.

Usabilidad

La interfaz debe ser intuitiva y permitir completar las tareas principales (publicar un anuncio, buscar artículos y contactar) en pocos pasos y sin formación previa.

Rendimiento

La experiencia debe ser ágil: las pantallas de listado, ficha y búsqueda deben responder con rapidez tanto en móvil como en escritorio

Seguridad y autenticación

Las credenciales se almacenarán de forma segura y las operaciones sensibles se protegerán mediante conexiones seguras. El sistema debe evitar accesos no autorizados y permitir la gestión básica de permisos.

Privacidad y protección de datos

Se recogerán únicamente los datos necesarios y se pedirá consentimiento al registrarse. La política de privacidad estará accesible desde la plataforma y se indicará claramente el uso de los datos recogidos.

Almacenamiento de multimedia

La carga y entrega de imágenes debe ser eficiente y fiable: se establecerán límites de tamaño, subida controlada y feedback al usuario durante el proceso.

Accesibilidad y compatibilidad

La interfaz seguirá pautas básicas de accesibilidad (contraste legible, navegación esencial con teclado) y será compatible con los navegadores actuales (Chrome, Firefox, Edge, Safari) en sus versiones recientes, tanto en desktop como en móvil.

Disponibilidad y recuperación

El servicio debe estar accesible durante las horas habituales de uso y disponer de un procedimiento básico de copia y restauración de datos.

Mantenibilidad y escalabilidad

El código deberá estar organizado y documentado lo suficiente para permitir mantenimiento y futuras ampliaciones. La arquitectura se seleccionará para facilitar un crecimiento razonable.

Estos requisitos no condicionan qué hace la aplicación, sino cómo debe hacerlo. Su cumplimiento es tan importante como el de los requisitos funcionales para que el proyecto sea considerado completo y funcional.

3.4. Análisis de alternativas tecnológicas

Las tecnologías utilizadas en este proyecto no se han elegido de forma arbitraria, sino valorando varias opciones para cada decisión relevante. A continuación se describe el proceso de análisis seguido en cada caso y la opción escogida para Closely Marketplace.

Lenguaje y framework de backend

Se consideraron Java con Spring Boot, JavaScript/TypeScript con Node.js (Express/NestJS) y Python con Flask. Spring Boot se eligió por su estabilidad, soporte para APIs REST bien estructuradas y la experiencia del equipo con Java, lo que facilita implementar seguridad, pruebas e integración en el tiempo disponible. Node.js ofrecía un ecosistema amplio y rapidez de prototipado, pero habría exigido definir más detalles arquitectónicos; Flask es ligero y rápido para prototipos, pero aporta menos estructura para un servicio con requisitos de seguridad y pruebas.

Framework de frontend

Se valoraron Nuxt (Vue), Next.js (React) y Svelte. Nuxt se seleccionó por permitir una entrega rápida de interfaces responsivas, su productividad para construir vistas y la experiencia previa del equipo con Vue. Next.js es potente y maduro, pero el equipo prefirió Nuxt por velocidad de desarrollo; Svelte es interesante por rendimiento, pero el ecosistema es más limitado para integraciones habituales.

Base de datos y autenticación

Se compararon PostgreSQL, MySQL y soluciones BaaS como Supabase o Firebase. Se eligió Supabase (PostgreSQL gestionado) por ofrecer persistencia robusta junto con servicios gestionados de autenticación y storage, lo que reduce la carga operativa y acelera el prototipado. MySQL es una alternativa válida para producción, y Firebase habría sido útil por su BaaS, pero Supabase aporta SQL completo y mayor control del esquema.

Almacenamiento de imágenes

Se analizaron Supabase Storage, Cloudinary y Amazon S3. Para el alcance actual se optó por Supabase Storage por su integración nativa con la base de datos y el auth del proyecto y por su simplicidad de uso en desarrollo. Cloudinary o S3 ofrecen ventajas en CDN y transformaciones masivas; se contemplan como migración futura si crece la demanda.

Sistema de mensajería (SMTP)

Se propuso la idea de crear un sistema de restauración de contraseña mediante un enlace que se enviará por correo electrónico, se plantearon varias herramientas pero estas necesitaban un pago para poder ser utilizadas, hasta que durante la investigación se encontró con un sitio web llamado Brevo que este no necesitaba un método de pago para poder ser utilizado además de disponer de una versión gratuita de la cual se permitía enviar un máximo de 300 correos al mes.

Despliegue e infraestructura

Se compararon plataformas gestionadas (Vercel/Netlify para frontend y servicios gestionados tipo Render/Heroku para backend) frente a VPS autogestionados. Se prefieren plataformas gestionadas por su rapidez de despliegue y menor coste operativo en la fase de prototipo; la migración a infraestructuras más controladas puede considerarse en fases posteriores.

Herramientas de desarrollo y colaboración

Se utilizará GitHub para control de versiones y PRs, Trello para la gestión ágil de tareas y GitHub Actions para integraciones básicas. Para prototipos UI se usarán maquetas en Figma. Estas elecciones priorizan simplicidad y trazabilidad durante el desarrollo.

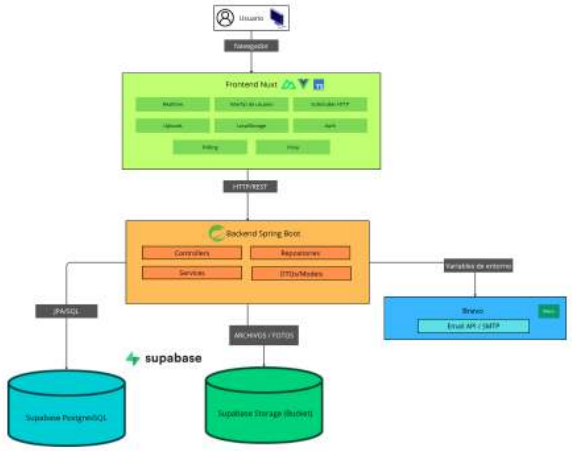
Conclusión

Las tecnologías elegidas (Spring Boot, Nuxt y Supabase) forman un stack coherente que equilibra productividad, seguridad y facilidad de despliegue, adecuado al tiempo y objetivos del proyecto. Las alternativas descartadas quedan justificadas para ampliaciones futuras si cambian necesidades de escala o prestaciones.

4. Diseño

4.1. Arquitectura del sistema

La aplicación sigue una arquitectura cliente-servidor clásica, dividida en tres capas bien diferenciadas que separan la presentación, la lógica de negocio y el almacenamiento de datos.



Punto de partida

El sistema se ha desarrollado desde cero y está pensado como una aplicación web donde el cliente (navegador) consume servicios proporcionados por un servidor. El objetivo es obtener una versión funcional que permita publicar, buscar y contactar entre usuarios.

Capa de presentación (frontend)

La capa de presentación está desarrollada con Nuxt (Vue). Es la parte que ve el usuario: páginas de listado, ficha de anuncio, formularios de publicación y el panel de usuario. Su responsabilidad es mostrar información, realizar

validaciones básicas en cliente y comunicarse con la API del servidor. Está diseñada para ser responsiva y usable en móvil y escritorio.

Capa de lógica de negocio (backend / API)

La capa de servidor está implementada con Spring Boot y expone una API REST. Recibe las peticiones del frontend, aplica las reglas del sistema (gestión de usuarios, anuncios y mensajería), valida datos y orquesta llamadas a la capa de datos y al almacenamiento de ficheros. Esta separación permite centralizar la seguridad, las comprobaciones y la lógica aplicable a todo el sistema.

Capa de datos y servicios gestionados

La persistencia se basa en Supabase (PostgreSQL gestionado) y Supabase Storage para las imágenes. Supabase también se utiliza para la autenticación, reduciendo la complejidad operativa. La capa de datos almacena usuarios, anuncios, conversaciones y metadatos necesarios para el funcionamiento de la plataforma.

Servicios auxiliares

Se contemplan servicios auxiliares para notificaciones/correos, backups y logging. En despliegues posteriores se puede añadir CDN para optimizar la entrega de imágenes.

Ventajas de esta organización

Separar las responsabilidades en capas facilita el desarrollo paralelo (frontend y backend), mejora la mantenibilidad y permite validar el prototipo funcional rápidamente. Además, el uso de servicios gestionados (Supabase) reduce la carga operativa en la fase inicial.

Esta arquitectura en capas es adecuada para el alcance del proyecto: permite entregar pronto una versión funcional, facilita pruebas y mantiene la puerta abierta a futuras mejoras (CDN, escalado del backend, panel administrativo).

4.2. Modelo de datos

La base de datos de Closely Marketplace está formada por varias tablas principales que almacenan la información necesaria para gestionar usuarios, anuncios y las interacciones entre ellos. A continuación se describen las tablas más relevantes y sus relaciones, siguiendo el estilo del ejemplo.

Tabla Usuario(users)

La tabla usuario almacena los datos que proporciona cada usuario que se registra a la web:

Campos: id(PK), username, email, avalar_url, bio, location, phone, role.

Relaciones de la tabla users con el resto de las tablas:

- users(id) → items.seller_id (un usuario puede tener muchos anuncios/items) (User 1:N Items).
- users(id) → conversations.buyer_id y conversations.seller_id.(User 1:N Conversations como buyer/seller).
- users (id) → messages.sender_id (User 1:N Messages).
- users (id) → orders.buyer_id y orders.seller_id (User 1:N Orders).
- users (id) → payments.user_id, shippings.user_id, password_reset_tokens.user_id (User 1:N Pagos/Envíos/Tokens).

Tabla Anuncios / Items (items)

La tabla Anuncio contiene cada oferta publicada.

Campos: id (PK), seller_id (FK → users), titulo, descripcion, precio_eur, categoria, marca, talla, estado, imagen y available.

Relaciones:

- items (id) → item_images.item_id (Item 1:N ItemImages).
- items (id) → conversations.item_id (Item 1:N Conversations).

Tabla Imágenes del anuncio(item_images)

La tabla Imagen guarda las referencias a las imágenes de un anuncio:

Campos: item_id (FK → items), image_url, PK (item_id, image_url).

Relaciones:

- Item 1:N ItemImages
- Cada imagen está “atrapada” a su item_id por la FK.

Tabla Órdenes / Compras (orders)

Aquí es donde se almacena las órdenes que se generen cuando un usuario compre un producto:

Campos: id (PK), item_id (columna del item), buyer_id, seller_id, item_title, item_image, amount_eur, datos de envío/pickup, payment_*, status y created_at.

Relaciones:

- buyer_id referencia lógicamente a users.id → User 1:N Orders
- seller_id referencia lógicamente a users.id → User 1:N Orders
- item_id referencia lógicamente a items.id → Item 1:N Orders*_

Tabla de Conversación(conversations)

Almacena las conversaciones que tiene el usuario con otros usuarios:

Campos: id (PK), item_id (FK → items), buyer_id (FK → users), seller_id (FK → users), created_at, updated_at.

Relaciones:

- Item 1:N Conversations
- Conversación 1:N Messages (vía messages.conversation_id)

Tabla de Mensaje (messages)

Almacena los mensajes de la conversación:

Campos: id (PK), conversation_id (FK → conversations), sender_id (FK → users), content, is_read, created_at.

Relaciones:

- Conversación 1:N Mensajes
- User 1:N Mensajes como remitente

Puente Order ↔ Conversación (order_conversations)

Asocia una orden con su conversación de chat correspondiente.

Campos: id (PK), order_id (UNIQUE), conversation_id (FK → conversations), created_at.

Relaciones:

- conversation_id → conversations.id: Conversation 1:N OrderConversations
- order_id es UNIQUE, lo que sugiere que cada orden solo mapea a una conversación → Order 1:1 con Conversation

Tabla de Pagos(payments)

En esta tabla se almacenan los pagos realizados por la compra de un producto.

Campos: id (PK), user_id (FK → users), method, amount, status, created_at.

Relación:

- User 1:N Payments

Tabla Envíos(shippings)

En esta tabla se almacena el registro de envío de los productos.

Campos: id (PK), user_id (FK → users), address, status, created_at.

Relación:

- User 1:N Shippings

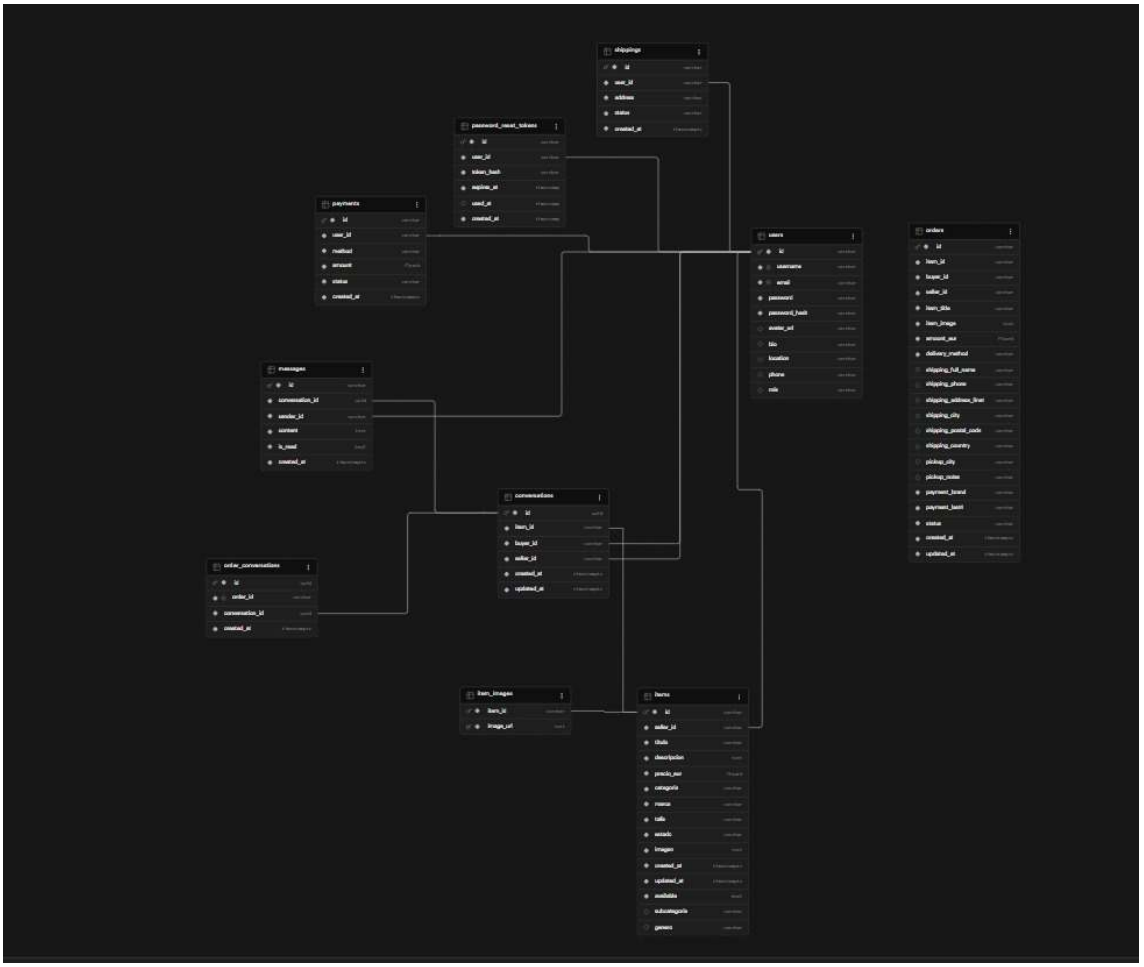
Tabla de Tokens de restablecimiento (password_reset_tokens)

Campos: id (PK), user_id (FK → users), token_hash, expires_at, used_at, created_at.

Relación:

- User 1:N PasswordResetTokens

Esta estructura normalizada evita redundancias (imágenes y favoritos en tablas separadas), facilita consultas habituales (listar anuncios por categoría o usuario, obtener mensajes de una conversación) y se adapta bien a Supabase/PostgreSQL.



4.3. Diseño de interfaz

La interfaz de la aplicación está diseñada para ser sencilla y funcional, priorizando que el usuario pueda publicar un anuncio, buscar artículos y contactar con el vendedor con el menor número de pasos posible. Se ha adoptado un enfoque mobile-first para garantizar una buena experiencia desde dispositivos móviles sin necesidad de un diseño específico por plataforma.

Las pantallas principales de la aplicación son las siguientes:

Pantalla de inicio: Muestra un buscador prominente, accesos a categorías y un listado de tarjetas con imagen, título, precio y distancia aproximada. Desde aquí el usuario puede filtrar resultados y acceder a la ficha del artículo.

Pantalla de registro: Formulario para crear una nueva cuenta con nombre, correo y contraseña; incluye campo opcional de perfil y enlace a centro de asistencia al usuario .

Pantalla de inicio de sesión: Permite al usuario acceder a su cuenta mediante correo y contraseña; incluye enlace para recuperar la contraseña y acceso al registro.

Pantalla principal / Listado: Muestra los resultados de búsqueda o exploración en formato tarjeta ; permite aplicar filtros (categoría, precio, estado) y ordenar resultados.

Ficha de artículo: Presenta galería de imágenes, título, precio, descripción, datos del vendedor y acciones (contactar, favorito).

Formulario de publicación: Flujo guiado en pasos para crear un anuncio (datos básicos → fotos → confirmación). Incluye validaciones que impiden publicar sin campos obligatorios y subida de imágenes asíncrona con barra de progreso y previsualización.

Pantalla de mis publicaciones: Lista las publicaciones del usuario con opciones para editar, eliminar; muestra métricas básicas (vistas, mensajes).

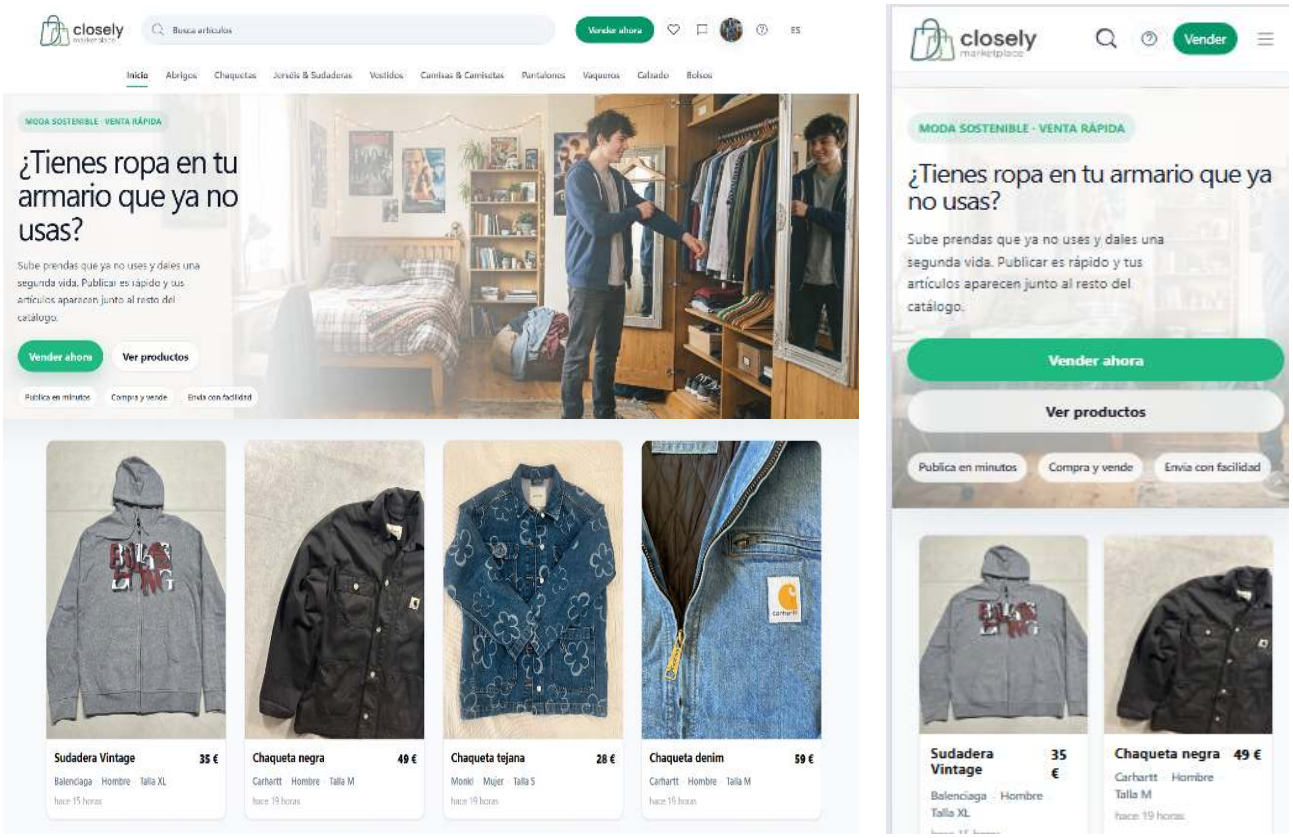
Mensajería / Conversaciones: Lista de conversaciones por anuncio y vista de chat con historial, envío de texto y indicador de leído.

Perfil público de vendedor: Página con información pública del vendedor y su listado de anuncios.

Favoritos: Página con los anuncios guardados por el usuario.

Panel de administración: Accesible solo para usuarios con rol de administrador; permite moderar anuncios, gestionar usuarios y revisar reportes.

La navegación entre pantallas sigue un flujo lineal y predecible: el usuario explora, accede a la ficha, puede contactar o favoritar, y gestiona sus publicaciones desde su panel. El panel de administración es independiente y solo aparece para el rol correspondiente.



5. Desarrollo

5.1. Estructura del proyecto

El proyecto sigue una organización separada en dos repositorios la lógica de backend (API y datos), el frontend (interfaz de usuario), los scripts/recursos de base de datos y la documentación.

Árbol de directorios :

```
Closely/
├─app
│   └─ pages
│            └─ auth
│                   └─ index.vue
│        └─vender.vue
├─backend/
├─ cp.txt
├─ Dockerfile
├─ mvnw
├─ mvnw.cmd
├─ pom.xml
├─ railway.json
├─ setup-supabase.bat
├─ db/
│   ├─ chat_phase1.sql
│   ├─ items_phase3_available.sql
│   └─ orders_phase1.sql
├─ src/
│   ├─ main/
│   │   ├─ java/
│   │   │   └─ com/
│   │   │      └─ raremarket/
│   │   │         └─ backend/
│   │   │            ├─ config/
│   │   │            ├─ controller/
│   │   │            ├─ dto/
```

```
│   │   │       ├── model/
│   │   │       │   ├── User.java
│   │   │       │   ├── Item.java
│   │   │       │   ├── Order.java
│   │   │       │   ├── Conversation.java
│   │   │       │   └── Message.java
│   │   │       ├── repository/
│   │   │       ├── security/
│   │   │       └── service/
│   │   └── resources/
│   │       ├── application.properties
│   │       └── schema.sql
│   └── test/
└── target/
├── ├── classes/
├── ├── generated-sources/
├──  └── backend-0.0.1-SNAPSHOT.jar.original
├──
├──frontend/
├── .nuxt/
├── .output/
└── app/
    ├── componentes/
    │   ├── admin/
    │   ├── estructura/
    │   │   ├── CabeceraAdmin.vue
    │   │   ├── CabeceraPrincipal.vue
    │   │   ├── NavegacionCategorias.vue
    │   │   └── PiePaginaPrincipal.vue
    │   └── explorar/
    │       ├── BarraLateralFiltrosExplorar.vue
    │       ├── BarraBusqueda.vue
    │       ├── CentroMensajes.vue
    │       ├── EditarPerfil.vue
    │       ├── TarjetaArticulo.vue
```

```
│   └── TarjetaVistaPreviaVenta.vue
├── constantes/
├── paginas/
│   ├── admin/
│   │   └── dashboard.vue
│   ├── articulos/
│   │   └── [id].vue
│   ├── autenticacion/
│   │   ├── index.vue
│   │   ├── recuperar.vue
│   │   └── restablecer.vue
│   └── legal/
│       └── aviso-legal.vue
├── plugins/
├── recursos/
├── tiendas/
└── utilidades/
```

Explicación:

backend: Contiene la aplicación Spring Boot gestionada por Maven. Aquí están los controladores REST, la lógica de negocio, los repositorios de acceso a datos, el sistema de autenticación, la configuración de la aplicación y los test unitarios para comprobar el correcto funcionamiento de backend. Las migraciones/esquema y propiedades se encuentran en src/main/resources.

frontend: Aplicación Nuxt (Vue) con la UI; componentes reutilizables en componentes/, vistas en paginas/ y utilidades para Supabase y stores en tiendas/ y utilidades/. nuxt.config.ts centraliza la configuración.

db/: Scripts SQL para poblar datos de ejemplo y pruebas; útiles para reproducción de entornos locales o CI.

docs: Guías de arranque, decisiones arquitectónicas y documentación de integraciones externas — punto de referencia para nuevos desarrolladores.

app: Contiene ejemplos de páginas o piezas de adaptación rápida (p. ej. vistas usadas como referencia).

setup-supabase.bat: Script de soporte para configurar Supabase en desarrollo.

target/: Salida de la compilación Maven; no se versiona.

Esta organización permite a cualquier desarrollador localizar rápidamente la API, el frontend, los scripts de datos y la documentación necesaria para arrancar o modificar el proyecto.

5.2. Implementación de funcionalidades

5.2.1. Autenticación y gestión de sesión

Qué hace: Permite registro, inicio/cierre de sesión y gestión de la sesión del usuario (roles básicos de vendedor/cliente).

Implementación conceptual: Autenticación gestionada por Supabase (autenticación y usuarios) integrada desde el frontend mediante utilidades (useClienteSupabase) y endpoints del backend cuando es necesario para lógica adicional.

Decisiones técnicas: Se optó por delegar la autenticación a Supabase para acelerar desarrollo y aprovechar tokens gestionados; el backend válida tokens en rutas protegidas cuando procede. Configuración y guías en backend-auth.md.

Estado: Implementada (frontend + integración con Supabase) con comprobaciones en backend para rutas que requieren autorización.

5.2.2. Publicación y edición de artículos (flujo de venta)

Qué hace: Permite a un usuario publicar nuevos artículos, editar su ficha y eliminar/publicar nuevamente; incluye subida de imágenes y campos de descripción, precio y categoría.

Implementación conceptual: Interfaz de creación/edición en vender.vue y componentes reutilizables (TarjetaArticulo.vue, TarjetaVistaPreviaVenta.vue). Datos persistidos mediante llamadas a la API del backend o directamente a Supabase según la operación.

Decisiones técnicas: Se usa una arquitectura de componentes en Nuxt para separar vista y lógica; la subida de archivos se gestiona con el almacenamiento de Supabase para simplificar integraciones. Los stores (useArticulosStore.ts) mantienen el estado local y la caché.

Estado: Implementada; revisar validaciones y control de tamaños de imagen para robustez.

5.2.3. Exploración, búsqueda y filtros

Qué hace: Página de exploración con filtros por categoría/tienda, barra de búsqueda y listas paginadas de artículos.

Implementación conceptual: Componente de búsqueda BarraBusqueda.vue, filtros en BarraLateralFiltrosExplorar.vue y listados en TarjetaArticulo.vue. El frontend consulta endpoints o directamente Supabase para obtener resultados filtrados.

Decisiones técnicas: Filtrado en frontend combinado con consultas optimizadas al backend/Supabase para reducir payloads; se prioriza UX reactiva (debounce en búsqueda). Categorías centralizadas en categorias.ts.

Estado: Implementada; posible mejora: añadir indexing full-text o cache en backend para resultados más rápidos en producción.

5.2.4. Chat y mensajería entre usuarios

Qué hace: Permite conversaciones entre usuarios interesados en artículos (mensajes en tiempo real o near-real-time).

Implementación conceptual: UI de chat en chat.vue y componente CentroMensajes.vue; backend contiene lógica de persistencia/entrega de mensajes y/o se utiliza Supabase Realtime según la configuración. Hay utilidades para conteo de mensajes no leídos (useConteoChatsNoLeidos.ts).

Decisiones técnicas: Se documentó la estrategia en backend-chat.md. Se eligió una combinación de persistencia en la base de datos y notificaciones por WebSocket/Realtime para fiabilidad y reentregas.

Estado: Implementada en modo básico; mejoras posibles: escalado de realtime y tests de carga.

5.2.5. Órdenes y proceso de compra/venta

Qué hace: Gestiona la creación de órdenes, estado de compra/venta y su persistencia.

Implementación conceptual: Endpoints en backend para crear y actualizar órdenes; scripts y esquema de base de datos para ordenes en db/orders_phase1.sql y lógica en backend/src/main/java/....

Decisiones técnicas: Las validaciones críticas (consistencia de stock/ disponibilidad) se realizan en el servidor dentro de transacciones para evitar condiciones de carrera. Se mantiene el histórico de estados para auditoría.

Estado: Implementada la base; posible ampliación: integración de pasarelas de pago y gestión de notificaciones al vendedor.

5.2.6. Perfiles de usuario y panel de vendedor

Qué hace: Gestión de perfil, visor de publicaciones propias, estadísticas básicas y edición de datos del usuario.

Implementación conceptual: Páginas editar-perfil.vue, mis-publicaciones.vue y componentes de perfil en componentes. Backend expone endpoints para actualizar datos de usuario y recuperar métricas.

Decisiones técnicas: Separación clara entre datos públicos y privados; comprobaciones de autorización en backend para acceder y modificar recursos del vendedor.

Estado: Implementada; añadir métricas más avanzadas está planificado.

5.2.7. Integración con Supabase y scripts de base de datos

Qué hace: Proporciona autenticación, almacenamiento de archivos, base de datos y funcionalidades realtime donde procede. Scripts SQL en db/ permiten poblar el entorno local.

Implementación conceptual: Uso de clientes y hooks (useClienteSupabase.ts) para centralizar llamadas a Supabase desde el frontend; backups y cargas iniciales en db/. Documentación de arranque en SUPABASE_SETUP.md y CONEXION_SUPABASE_RESUMEN.md.

Decisiones técnicas: Delegar en Supabase reduce esfuerzo de autenticación y almacenamiento; se mantiene control en backend para estrictas reglas de negocio y validaciones de integridad.

Estado: Integración estable para entornos de desarrollo; revisar políticas de seguridad/rules en Supabase antes de producción.

5.3. Pruebas

Las pruebas se centraron en verificar el correcto funcionamiento de las funcionalidades principales mediante pruebas funcionales e integración ligera, prestando especial atención a casos límite relevantes para el flujo de venta.

Se han realizado manuales y exploratorias:

- Registro, inicio/cierre de sesión y acceso a rutas protegidas (integración con Supabase).
- Publicación, edición y eliminación de artículos desde la UI (comprobación de persistencia y subida de imágenes).
- Búsqueda y filtros en la vista de exploración.
- Envío/recepción de mensajes en el chat y conteo de no-leídos.
- Creación de órdenes y verificación de estados y disponibilidad.

Casos relevantes detectados

- Concurrencia en creación de órdenes: se comprobó que las validaciones críticas se ejecutan en el servidor; se recomienda añadir pruebas concurrentes para garantizar la transaccionalidad.
- Validación de subida de imágenes: falta manejo sólido de imágenes muy grandes y mensajes de error claros.

Conclusión

Pruebas manuales funcionales realizadas; no se han detectado fallos críticos. Además, el equipo ya añadió pruebas unitarias en BackendApplicationTests.java y un panel de administración para gestionar publicaciones en mis-publicaciones.vue. Para aumentar la fiabilidad antes de producción es necesario ampliar cobertura automatizada (unitarios + E2E) y ejecutar pruebas de concurrencia/carga.

6. Conclusiones

6.1. Conclusiones generales

El resultado del proyecto es una aplicación web funcional que cubre las necesidades básicas de un marketplace: registro y gestión de usuarios, publicación y edición de artículos, búsqueda y filtros, mensajería y gestión de órdenes. La solución ofrece una mejora real frente a la gestión manual inicial y permite navegar y operar los flujos principales de compra/venta de forma coherente.

A nivel de aprendizaje, el desarrollo ha permitido consolidar conocimientos prácticos en diseño de interfaces componentizadas, gestión de estado en el frontend y construcción de APIs y validaciones en el backend. Integrar servicios gestionados ha agilizado el desarrollo y obligó a afrontar retos reales (p. ej. integraciones realtime y consistencia en órdenes) que han resultado formativos y enriquecedores.

El aspecto menos logrado es la cobertura de pruebas automatizadas y la robustez frente a cargas/concurrencia; el panel de administración y algunas validaciones (subida de imágenes, límites de tamaño) quedan más básicos de lo deseado por falta de tiempo. Recomendación: priorizar tests E2E y de concurrencia, configurar CI y endurecer validaciones antes de un despliegue en producción.

6.2. Consecución de objetivos

El objetivo general del proyecto desarrollar una plataforma web que permita publicar y gestionar artículos, facilitar la comunicación entre compradores y vendedores y soportar el ciclo básico de órdenes se ha alcanzado satisfactoriamente. La aplicación es funcional y permite realizar los principales flujos de publicación, búsqueda, mensajería y compra/venta desde la interfaz de usuario.

Respecto a los objetivos específicos, la mayoría se han cumplido de forma completa: registro e inicio de sesión, consulta y filtrado de artículos, creación y edición de fichas de producto, y mensajería básica funcionan según lo previsto. La creación de órdenes está implementada y valida la disponibilidad en el servidor, garantizando el flujo esencial de compra/venta.

Hay objetivos cumplidos parcialmente: la gestión avanzada de órdenes en escenarios concurrentes requiere pruebas adicionales y trabajo para endurecer la transaccionalidad; el panel de administración funciona de forma básica pero se quedó corto en filtros, estadísticas y ergonomía por limitaciones de tiempo.

También faltan pruebas E2E y una cobertura automatizada completa que asegure los flujos integrados antes de producción.

Un único objetivo no prioritario quedó fuera por alcance: la integración de pipelines de despliegue/CI que ejecuten automáticamente builds, tests y seeds reproducibles.

6.3. Valoración de la metodología y planificación

La planificación inicial se siguió en términos generales, aunque con algunas desviaciones en fases intermedias. La fase de desarrollo de las funcionalidades principales requirió más tiempo del estimado, principalmente por la resolución del problema de concurrencia en el flujo de órdenes, un aspecto que no se había anticipado con la profundidad necesaria en la planificación original.

Como consecuencia, la fase dedicada a funcionalidades secundarias se redujo, lo que explica que el panel de administración haya quedado más básico de lo previsto. En una planificación futura sería recomendable reservar más tiempo para imprevistos técnicos en las fases de desarrollo más complejas y priorizar tareas que mitiguen riesgos de integración.

La metodología de trabajo por bloques funcionales ha resultado adecuada: desarrollar y validar cada funcionalidad de forma independiente antes de

avanzar permitió detectar errores con mayor facilidad y mantener una versión funcional del sistema en todo momento. La dependencia de servicios externos agilizó el desarrollo, pero exigió iteraciones adicionales para ajustar reglas de seguridad e integración.

Recomendaciones prácticas: reservar tiempo específico para pruebas E2E y de concurrencia desde etapas tempranas; prever margen para ajustes de integración con servicios externos; dedicar una iteración a robustecer el panel administrativo y configurar CI que ejecute builds y pruebas antes del despliegue.

6.4. Visión de futuro

La línea de mejora más inmediata sería completar y endurecer las piezas pendientes de la plataforma que añaden valor directo a usuarios y operaciones: mejorar las validaciones y experiencia de subida de imágenes, añadir notificaciones (por ejemplo, correo y push) para eventos clave y ampliar el panel de administración con estadísticas y filtros avanzados para facilitar la moderación y el soporte.

A medio plazo, interesa consolidar la calidad y la resiliencia: añadir pruebas E2E que cubran los flujos críticos (publicar, comprar, mensajear), ejecutar pruebas de concurrencia sobre la creación de órdenes y configurar una pipeline de CI/CD que automatice builds, tests y despliegues seguros. Esto reducirá riesgos en producción y facilitará entregas frecuentes.

A largo plazo, la plataforma puede evolucionar hacia un marketplace más completo: soporte multi-vendedor y roles administrativos más refinados, integración con pasarelas de pago, internacionalización y versión PWA/móvil para mejorar alcance y retención. También sería recomendable incorporar analíticas de uso y telemetría para priorizar mejoras basadas en datos.

7. Glosario

A continuación se definen los principales términos técnicos y acrónimos utilizados en esta memoria.

Autenticación: Proceso que verifica la identidad de un usuario para permitir el acceso a la aplicación.

Supabase: Plataforma BaaS utilizada aquí para autenticación, base de datos, storage y realtime.

RLS (Row-Level Security): Política que controla el acceso a filas de la base de datos según reglas por usuario.

Realtime: Mecanismo para sincronizar eventos en tiempo real (mensajería, actualizaciones) entre cliente y servidor.

Nuxt.js: Framework sobre Vue.js que organiza la aplicación frontend y optimiza renderizado y rutas.

Vue.js: Librería de JavaScript usada para construir componentes e interfaces reactivas.

PWA (Progressive Web App): Web app con capacidades nativas (offline, instalación) para mejorar la experiencia móvil.

REST API: Interfaz basada en HTTP para comunicar frontend y backend mediante recursos.

CRUD: Operaciones básicas sobre datos: Create, Read, Update, Delete.

ORM (Object-Relational Mapping): Capa que mapea objetos del lenguaje a consultas SQL en la base de datos.

Maven: Herramienta de gestión y construcción del backend Java (dependencias, compilación, tests).

CI/CD: Prácticas y herramientas que automatizan integración, pruebas y despliegues.

Pruebas unitarias: Tests que verifican unidades pequeñas de código; presentes en el backend del proyecto.

E2E (end-to-end): Pruebas que simulan flujos completos desde la UI hasta el backend; recomendadas para flujos críticos.

Seed (datos de inicialización): Scripts o archivos con datos de ejemplo para poblar la base en pruebas/despliegues.

JWT (JSON Web Token): Token compacto para transmitir información de identidad de forma segura entre partes.

UUID: Identificador único universal usado para identificar recursos de manera no colisionante.

Storage: Servicio para guardar ficheros (imágenes, documentos) asociado al proyecto.

8. Bibliografía

Supabase. Supabase Documentation. https://supabase.com/docs

Next Team. Nuxt 3 Documentation. https://nuxt.com/docs

Vue.js Core Team. Vue.js Guide. https://vuejs.org/guide/

Tailwind Labs. Tailwind CSS Documentation. https://tailwindcss.com/docs

Pinia Team. Pinia Documentation. https://pinia.vuejs.org/

Spring Team. Spring Boot Reference Documentation. https://docs.spring.io/spring-boot/docs/current/reference/html/

Spring Security Project. Spring Security Reference. https://docs.spring.io/spring-security/reference/

PostgreSQL Global Development Group. PostgreSQL Documentation. https://www.postgresql.org/docs/

Apache Maven Project. Maven: The Apache Software Foundation. https://maven.apache.org/

GitHub Docs. GitHub Actions Documentation. https://docs.github.com/en/actions.

Playwright Team. Playwright Documentation. https://playwright.dev/docs/intro

OWASP Foundation. OWASP Top Ten & Secure Development Guidance. https://owasp.org/

MDN Web Docs. HTTP: Hypertext Transfer Protocol. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

JWT.io. Introduction to JSON Web Tokens. https://jwt.io/introduction

GitHub. GitHub Security best practices / Dependabot / Guides. https://docs.github.com/

DockerFile reference.

https://docs.docker.com/reference/dockerfile

9. Anexos

Anexo A - Documento funcional

https://docs.google.com/document/d/1pcESm3o2wjYojaOjW0Vcs97iRcBbzidQDJdudlBgt_0/edit?usp=sharing

Versión: V1

fecha del documento:16/02/2026

Anexo B - Repositorio GitHub:
https://github.com/Matias-Obando/tienda-raremarket.git

Nota sobre el uso de inteligencia artificial

La elaboración de esta memoria ha contado con el apoyo de herramientas de inteligencia artificial como recurso de asistencia en tareas concretas de redacción, corrección y orientación técnica. Su uso se ha limitado a facilitar la mejora de la expresión, la organización de contenidos y la resolución de dudas puntuales durante el desarrollo del documento y del proyecto.

En particular, la inteligencia artificial se ha utilizado para:

1. Realizar correcciones y ajustes de redacción en la memoria.

2. Obtener apoyo en la creación y revisión de pruebas unitarias para verificar la correcta creación de las pruebas y también de comprobar que cada prueba funcione correctamente.

3. Orientar parte del proceso de despliegue en Vercel ante la falta de documentación o tutoriales claros en algunos casos. Cambien el proceso de despliegue de railway(que es donde se despliega el backend) por qué su documentación estaba anticuada y su interfaz era poco intuitiva a parte de que se tuvo que tomar como referencia otro proyecto que se desplegaba de manera similar, solo se necesito la ayuda de IA para asesoramiento para este proceso realizando diferentes tipos de despliegue.

El contenido final ha sido revisado, validado y adaptado por el autor, que asume la responsabilidad completa sobre la precisión, coherencia y adecuación del trabajo presentado. La inteligencia artificial ha actuado únicamente como herramienta de apoyo dentro de un proceso supervisado por el autor.

El propósito de esta nota es fomentar la transparencia en el uso de herramientas de inteligencia artificial y promover un uso responsable, crítico y ético de las mismas en cualquier ámbito académico y profesional.