

LocuVentas

1. INTRODUCCIÓN.....	2
1.1. Presentación del proyecto.....	2
1.2. Objetivos del proyecto.....	2
1.3. Justificación del proyecto.....	2
2. ANÁLISIS DE REQUERIMIENTOS.....	2
2.1. Identificación de necesidades y requerimientos.....	2
2.2. Identificación de público.....	3
2.3. Estudio de mercado y competencia.....	3
3. DISEÑO Y PLANIFICACIÓN.....	4
3.1. Definición de la arquitectura del proyecto.....	4
3.2. Diseño de la interfaz de usuario.....	5
3.3. Planificación de las tareas y los recursos necesarios.....	6
4. IMPLEMENTACIÓN Y PRUEBAS.....	7
4.1. Desarrollo de las funcionalidades del proyecto.....	7
4.1.1. Registro y gestión de usuario.....	7
4.1.2. Creación y gestión de productos.....	7
4.1.3. Gestión y realización de una venta.....	8
4.2. Pruebas unitarias y de integración.....	9
4.3. Corrección de errores y optimización del rendimiento.....	13
5. DOCUMENTACIÓN.....	14
5.1. Documentación técnica.....	15
5.2. Documentación del usuario.....	15
5.3. Administrador.....	15
5.4. Vendedor.....	15
5.5. Manual de instalación y configuración.....	15
6. MANTENIMIENTO Y EVOLUCIÓN.....	15
6.1. Plan de mantenimiento y soporte.....	15
6.2. Identificación de posibles mejoras y evolución del proyecto.....	15
6.3. Actualizaciones y mejoras futuras.....	15
7. CONCLUSIONES.....	15
7.1. Evaluación del proyecto.....	15
7.2. Cumplimiento de objetivos y requisitos.....	15
7.3. Lecciones aprendidas y recomendaciones para futuros proyectos.....	15

8. BIBLIOGRAFÍA Y REFERENCIAS.....	17
8.1. Fuentes utilizadas en el proyecto.....	17
8.2. Referencias y enlaces de interés.....	17

● INTRODUCCIÓN

○ **Presentación del proyecto**

Nuestro proyecto se centra en el desarrollo de una aplicación web modesta diseñada para optimizar la gestión de ventas en los negocios conocidos como "Locutorios". La elección del formato web proporciona una versatilidad excepcional, permitiendo a los usuarios acceder a la aplicación sin necesidad de equipos de alta gama.

○ **Objetivos del proyecto**

El objetivo principal de nuestro proyecto es simplificar las tareas diarias de los vendedores en los locutorios, ofreciendo una herramienta que promueva la rapidez y eficiencia en sus operaciones.

○ **Justificación del proyecto**

La motivación detrás de este proyecto surge de la necesidad de mejorar la experiencia de los vendedores en los locutorios. Además de facilitar la realización y registro de ventas de manera más ágil y eficiente, buscamos aportar valores añadidos que proporcionen soluciones innovadoras y previamente no consideradas.

● ANÁLISIS DE REQUERIMIENTOS

○ **Identificación de necesidades y requerimientos**

Con el objetivo de identificar adecuadamente los requisitos de este proyecto, primero tenemos que definir dos conceptos claves: “Requisitos funcionales” y “Requisitos no funcionales”.

Los requisitos funcionales son aquellos que van a definir nuestras reglas de negocio, es decir, es decir, qué acciones se van a poder llevar a cabo dentro de la aplicación y como se maneja la información que los usuarios nos va a confiar.

Por otra parte los requisitos no funcionales hacen referencia a las virtudes que podemos ofrecer, por poner algunos ejemplos: usabilidad, confiabilidad, rendimiento, seguridad, diseño, accesibilidad, etc.

Tras la breve introducción anterior, procedemos a listar nuestros requerimientos:

→ **Requerimientos funcionales**

- ◆ Sistema de login donde habrá tres roles, “ADMIN”, “USER” y “VENDEDOR”.
- ◆ Cualquier usuario se puede registrar, teniendo solo el rol “USER”, sin embargo, para que el usuario pueda realizar las operaciones de vendedor, debe ser autorizado previamente por un administrador, el cual le otorga al usuario el rol de “VENDEDOR”.
- ◆ Un administrador puede dar de alta nuevas categorías(No implementado), productos, asignar el rol de vendedor a nuevos usuarios o por el contrario, descartarlos.
- ◆ Un vendedor puede realizar ventas.
- ◆ El sistema contará con una categorización por país de los productos, aparte de poder asignar otras categorías personalizadas por los administradores(No implementado).
- ◆ Cuando se lleve a cabo una venta, se mostrará el detalle de esta y la opción de descargar el ticket en formato pdf.
- ◆ A la hora de registrarse como nuevo usuario, será obligatorio proporcionar una foto, con el objetivo de que el administrador visualice de forma más diferenciada a sus vendedores.

→ **Requerimientos no funcionales**

- ◆ En el apartado de usabilidad nuestro objetivo es que la aplicación sea muy fácil de usar.
- ◆ Alto rendimiento, buscando que los procesos se sientan con una alta fluidez.
- ◆ A nivel de seguridad se busca que no haya vulnerabilidades.
- ◆ Ofrecer métodos de ordenamiento adecuados de los datos(parcialmente implementado).
- ◆ Aplicación usable para dispositivos móviles.

○ **Identificación de público**

El público objetivo de LocuVentas son aquellos pequeños comercios que busquen una manera sencilla de gestionar sus ventas diarias, donde no haya infinidad de funcionalidades que no estén relacionados al negocio, logrando satisfacer sus necesidades más básicas y prioritarias.

- **Estudio de mercado y competencia**

Se trata de un mercado no tan explorado, debido a que muchos locales están acostumbrados a apuntar sus ventas a papel, aunque existen aplicaciones que permiten gestionar ventas, no existe uno tan enfocado al nicho.

● DISEÑO Y PLANIFICACIÓN

- **Definición de la arquitectura del proyecto**

LocuVentas se ha desarrollado siguiendo una arquitectura cliente-servidor, con una separación clara entre el frontend y el backend. Esta estructura modular permite una mejor organización del código, facilita el mantenimiento del sistema y lo hace escalable ante futuras ampliaciones o integraciones.

Frontend

- Tecnologías utilizadas:
 - HTML, CSS y JavaScript, para la estructura, estilo y funcionalidad básica de la interfaz.
 - React.js, como librería principal para construir interfaces dinámicas, reactivas y centradas en la experiencia del usuario.
- Responsabilidades:
 - Interacción directa con el usuario final.
 - Consumo de la API REST expuesta por el backend.
 - Validación básica de datos antes de enviarlos al servidor.
 - Navegación entre vistas (registro, login, productos, ventas, etc.).

Backend

- Lenguaje de programación:
 - Java, por su robustez, soporte comunitario y escalabilidad.
- Framework:
 - Spring Boot, que facilita el desarrollo de aplicaciones RESTful con configuración mínima y gran capacidad de extensión.
- Base de datos:
 - MySQL, utilizada para almacenar de forma persistente los datos principales del sistema: usuarios, productos, ventas, relaciones y pagos.
- Responsabilidades:
 - Autenticación y autorización de usuarios mediante roles (administrador y vendedor).

- Gestión de la lógica de negocio, incluyendo reglas como restricciones de eliminación o visibilidad según rol.
- Exposición de una API RESTful para interactuar con las funcionalidades del sistema desde el frontend.
- Gestión de operaciones CRUD sobre usuarios, productos y ventas, respetando las reglas de negocio definidas para cada entidad. Por ejemplo:
 - Solo se permite eliminar productos si no están asociados a una venta.
 - Las ventas no se eliminan, sino que pueden ser canceladas por el administrador.
 - Los usuarios no pueden eliminarse libremente; su aprobación o denegación depende de un administrador.

Gestión de imágenes:

Los usuarios deben subir una imagen de perfil al registrarse. Esta imagen se guarda en el directorio resources/uploads dentro del backend.

Si bien en un entorno profesional lo recomendable sería almacenar estos archivos en un sistema externo o en un contenedor de archivos estático, esta solución local fue elegida por su simplicidad y adecuación a los requerimientos y alcance de este proyecto académico.

La imagen se asocia al usuario y se recupera al iniciar sesión, mostrándose en el frontend para una experiencia más personalizada.

Comunicación Cliente-Servidor

La comunicación entre el frontend y el backend se basa en una API RESTful, utilizando:

- Protocolo: HTTP/HTTPS
- Formato de datos: JSON, tanto en las peticiones como en las respuestas.

Esta arquitectura facilita la independencia entre cliente y servidor, permitiendo evolucionar cada parte por separado y habilitando una posible reutilización del backend por otros clientes en el futuro (como aplicaciones móviles).

Los detalles específicos de los endpoints utilizados se documentan en la sección [5.1. Documentación técnica.](#)

○ Diseño de la interfaz de usuario

El diseño de la interfaz de usuario se enfocó en ofrecer una experiencia clara, funcional y visualmente coherente. Se construyó utilizando React.js junto con Tailwind CSS, lo que permitió desarrollar una interfaz moderna, responsiva y mantenible. Se buscó siempre un equilibrio entre estética, usabilidad y velocidad de desarrollo.

Paleta de colores

Se definió una paleta visual con colores contrastantes y actuales, aplicados de forma consistente en botones, encabezados, fondos y elementos interactivos. Esta paleta no solo

aporta identidad visual al sistema, sino que también mejora la accesibilidad y la diferenciación entre secciones.

Color	Hex	Clase Tailwind
Azul eléctrico	#007BFF	bg-blue-500 o bg-blue-600
Violeta	#9B51E0	bg-purple-500 o bg-purple-600
Naranja brillante	#FF7F50	bg-orange-400 o bg-orange-500
Negro/Gris oscuro	#222222	bg-gray-900 o bg-gray-800
Blanco	#FFFFFF	text-white o bg-white

Diseño responsivo

Se diseñó una interfaz responsiva, adaptada principalmente a dos tipos de dispositivos: móviles y escritorio. En formato móvil se priorizó el uso de cards verticales para facilitar la lectura y la interacción táctil, mientras que en escritorio se utilizaron tablas para mostrar mayor cantidad de datos en pantalla. Aunque no se desarrolló una capa específica para tablets, la interfaz mantiene una funcionalidad aceptable, aunque puede presentar pequeños desajustes visuales en dicho formato.

Elementos interactivos y experiencia de usuario

Con el objetivo de mejorar la experiencia general del usuario, se incorporaron diversos elementos dinámicos y visuales:

- Modales: Se implementaron modales de confirmación e información. En ciertas secciones se utilizaron modales transparentes y efectos visuales con bg-opacity, backdrop-blur y otras utilidades de Tailwind, logrando una interfaz más moderna y estética.
- Animaciones con Tailwind CSS: Se añadieron animaciones suaves para transiciones de elementos, como la aparición/desaparición del header mediante clases como transition-all, ease-out y transform. Este tipo de animaciones ayudan a mejorar la fluidez visual de la interfaz sin necesidad de librerías externas.
- Notificaciones: Se utilizó react-toastify para mostrar toasts informando al usuario sobre acciones realizadas (creaciones, errores, eliminaciones, etc.).
- Validaciones por campo: Se implementaron mensajes de error específicos para cada campo de formulario, permitiendo identificar errores de forma rápida y clara durante la interacción.
- Formato relativo de fechas: Se incorporó una librería para mostrar tiempos relativos tipo “hace 7 minutos”, generando una experiencia familiar similar a plataformas como Instagram.

Librerías y reutilización de componentes

- Iconos: Se usó lucide-react para añadir iconos modernos y minimalistas (ej. `import { X } from "lucide-react"` para botones de cierre u otros íconos de acción).
- Componentes reutilizables: A lo largo del proyecto se creó una base de componentes reutilizables para formularios, modales, inputs y botones. Aunque en algunas secciones se pudo haber aplicado con más rigurosidad, se estableció una base reutilizable sólida para futuras mejoras.

○ **Planificación de las tareas y los recursos necesarios**

Inicialmente, la planificación se pensó realizar con herramientas como Trello, dividiendo las tareas por módulos funcionales (registro, login, gestión de productos, ventas, etc.). Sin embargo, en la práctica, el desarrollo del proyecto adoptó un enfoque más iterativo y flexible, donde muchas decisiones de implementación y diseño se fueron tomando a medida que avanzaba con el código.

Este enfoque, más cercano a una metodología ágil, permitió adaptarse rápidamente a cambios de requisitos, mejoras en la interfaz o ajustes en la lógica de negocio a medida que se detectaban nuevas necesidades o problemas durante la construcción de la aplicación.

Aunque no se siguió un esquema rígido de planificación anticipada, se procuró mantener una organización básica mediante:

- Listas de tareas temporales y personales en Trello para tener visibilidad de lo pendiente y lo ya realizado.
- Priorización de tareas más críticas o estructurales en etapas iniciales (registro, login, base de datos).
- Reajuste continuo de objetivos semanales en función del progreso y los desafíos técnicos encontrados.

En resumen, si bien no se siguió una planificación formal y estática, el proyecto se desarrolló con una estrategia adaptativa, lo cual fue clave para lograr avances constantes, descubrir mejoras sobre la marcha y enfocarse en entregar una versión funcional del sistema.

● **IMPLEMENTACIÓN Y PRUEBAS**

○ **Desarrollo de las funcionalidades del proyecto**

●.○.1. Registro y gestión de usuario.

- Registro de Usuario
 - Los usuarios pueden registrarse proporcionando los siguientes datos obligatorios:
 - Foto de perfil
 - Correo electrónico
 - Nombre de usuario
 - Contraseña
 - Tras el registro, la cuenta queda en estado pendiente de aprobación.
- Aprobación de Usuario por Administrador
 - Un administrador revisa las solicitudes de registro.
 - El administrador puede:
 - Aprobar la cuenta → El usuario obtiene acceso a la plataforma de ventas.
 - Denegar la cuenta → La cuenta se elimina de forma inmediata.
 - Si la cuenta está pendiente, el usuario no puede iniciar sesión en la plataforma.
- Edición de Perfil
 - Los usuarios aprobados pueden actualizar su información personal:
 - Foto de perfil
 - Correo electrónico
 - Nombre de usuario
 - Contraseña
- Acceso a la Plataforma
 - Solo los usuarios aprobados pueden acceder a las funcionalidades de venta.
 - Los usuarios en estado pendiente o denegado no tienen acceso a la plataforma.

●.○.2. Creación y gestión de productos.

La gestión de productos está disponible exclusivamente para los usuarios con rol de administrador. A través de una interfaz accesible desde el panel de administración, el administrador puede realizar las siguientes acciones:

- Crear productos: Se permite registrar nuevos productos, incluyendo su nombre, precio, país de origen, IVA, foto y las categorías a las que pertenece. Todos estos campos son validados antes de su almacenamiento.
- Editar productos: El administrador puede modificar tanto la información como la imagen asociada a un producto previamente creado.

- Eliminar productos: La eliminación de un producto solo es posible si este no ha sido vinculado a ninguna venta, garantizando la integridad de los datos históricos del sistema.

El acceso a esta funcionalidad está protegido y restringido al perfil adecuado, garantizando que solo personal autorizado pueda gestionar el catálogo de productos.

●.○.3. Gestión y realización de una venta

Una de las funcionalidades principales del sistema es la realización y gestión de ventas. Solo los usuarios aprobados por un administrador (es decir, administradores y vendedores activos) pueden acceder al panel principal y registrar nuevas ventas.

Creación de una venta

Los administradores y vendedores pueden realizar ventas desde la plataforma. Al crear una venta, se pueden registrar una o varias líneas de producto, indicando cantidad y tipo. Cada venta puede registrarse en uno de los siguientes estados de pago:

- Pagado: El total ha sido abonado en el momento de la venta.
- Parcial: Se ha pagado solo una parte del importe total.
- Pendiente: No se ha realizado ningún pago en el momento de la venta.

Estos tres estados permiten llevar un seguimiento más flexible de las transacciones.

Listado de ventas

El sistema permite consultar las ventas a través de dos vistas principales:

- Todas las ventas: Visualiza todas las ventas realizadas. Está disponible para administradores y vendedores (aunque los vendedores solo pueden ver las que han realizado ellos mismos).
- Ventas pendientes: Muestra únicamente aquellas ventas con estado de pago "pendiente" o "parcial". Esta vista también respeta las restricciones de visibilidad por rol.

Acceso a detalle de venta

Tras completar una venta, el usuario es redirigido automáticamente al detalle de dicha operación. Desde esta vista se puede consultar:

- Los productos vendidos y sus cantidades.
- Los subtotales con y sin IVA.
- El estado de pago y el saldo pendiente.
- El nombre del vendedor responsable.
- La fecha de creación de la venta.

Además, el detalle de cada venta puede consultarse posteriormente desde cualquiera de las vistas de listado mencionadas.

Gestión de pagos

En las ventas con estado "pendiente" o "parcial", se pueden registrar pagos posteriores. El sistema actualiza automáticamente el estado de la venta en función del monto abonado, cambiando a "pagado" cuando se completa el total.

Cancelación de ventas

Solo los administradores tienen permisos para cancelar ventas. Una vez cancelada, una venta ya no puede ser modificada ni recibir pagos.

Generación de tickets

El sistema permite generar y descargar un ticket en formato PDF para cada venta, el cual incluye todos los detalles de la transacción. Este ticket puede obtenerse inmediatamente después de finalizar la venta o desde el detalle de la misma en cualquier momento posterior.

○ Pruebas unitarias y de integración

●.○.1. Pruebas unitarias:

ID	Módulo	Caso de Prueba	Resultado esperado	Estado
1	Registro de usuario	Registrar un usuario con datos válidos.	El usuario debe ser registrado correctamente y quedar en estado pendiente de aprobación.	OK
2	Inicio de sesión	Iniciar sesión con un usuario pendiente de aprobación.	El sistema debe impedir el acceso e indicar que la cuenta requiere aprobación de un administrador.	OK
3	Inicio de sesión	Iniciar sesión con credenciales inválidas	El sistema debe mostrar un mensaje de error indicando que los datos son incorrectos.	OK
4	Inicio de sesión	Iniciar sesión con usuario aprobado y credenciales válidas.	El usuario debe acceder correctamente al panel principal (Dashboard).	OK
5	Registro de usuario	Registrar un usuario con datos inválidos.	El sistema debe mostrar mensajes de error para los campos incorrectos.	OK

6	Gestión de productos	Crear un producto con datos válidos.	El producto debe crearse correctamente y aparecer en la tabla de productos.	OK
7	Gestión de productos	Intentar crear un producto con datos inválidos.	El sistema debe mostrar mensajes de error para los campos incorrectos.	No comprobado
8	Perfil de usuario	Editar los datos del perfil del usuario.	Los cambios realizados deben guardarse correctamente y reflejarse en la interfaz	OK
9	Gestión de usuarios	El administrador elimina un usuario pendiente.	Si el administrador pulsa “denegar”, la cuenta del usuario pendiente debe eliminarse	OK
10	Ventas	Visualizar productos disponibles para venta.	El usuario autenticado (ADMIN o VENDEDOR) debe poder visualizar los productos en el panel de ventas.	OK
11	Ventas	Registrar una venta con distintas formas de pago.	La venta debe guardarse correctamente y reflejarse según la opción elegida (“Finalizar y cobrar” o “Guardar sin cobrar”).	OK
12	Ventas	Visualizar todas las ventas (vendedor).	El vendedor debe poder ver únicamente sus propias ventas.	OK
13	Ventas	Visualizar todas las ventas (administrador).	El administrador debe poder ver todas las ventas registradas en el sistema.	OK
14	Ventas	Visualizar ventas pendientes (administrador).	El administrador debe poder ver todas las ventas con estado “Pendiente” o “Parcial”.	OK
15	Ventas	Visualizar ventas pendientes (vendedor).	El vendedor debe ver únicamente sus ventas con estado	OK

			"Pendiente" o "Parcial".	
16	Gestión de productos	Crear producto y mostrar listas de países y categorías.	Al crear un producto, se deben mostrar las opciones disponibles de países y categorías.	OK
17	Gestión de productos	Editar un producto y mostrar listas de países y categorías	Al editar un producto, se deben mostrar correctamente los países y categorías disponibles	OK
18	Gestión de productos	Intentar eliminar un producto asociado a una venta.	El sistema debe impedir la eliminación y mostrar un mensaje indicando que el producto ya ha sido vendido.	OK
19	Registro de usuario	Subir una imagen inválida como foto de perfil.	El sistema debe rechazar la imagen y mostrar un mensaje de error.	OK
20	Gestión de productos	Subir una imagen inválida al crear un producto.	El sistema debe rechazar la imagen y no permitir crear el producto.	OK
21	Ventas	Registrar un pago con monto mayor al saldo pendiente.	El sistema debe rechazar el pago y mostrar un error de validación.	NO COMPROBADO
22	Ventas	Cancelar una venta ya cancelada.	El sistema debe mostrar un mensaje de error indicando que la venta ya está cancelada.	NO COMPROBADO
23	Registro de usuario	Registrar múltiples usuarios con el mismo correo.	El sistema debe evitar registros duplicados y mostrar un mensaje de error.	OK
24	Gestión de ventas	Intentar registrar una venta sin productos seleccionados.	El sistema debe impedir la creación de la venta y mostrar un mensaje de advertencia	OK

Algunos casos de prueba no pudieron ser verificados por falta de tiempo o por no estar completamente implementadas ciertas funcionalidades. Sin embargo, se han documentado para facilitar su validación en futuras versiones del proyecto.

●.○.2. Pruebas de integración:

Las pruebas de integración tienen como objetivo verificar que los diferentes módulos del sistema funcionan correctamente en conjunto, no solo de forma aislada como en las pruebas unitarias. Estas pruebas permiten detectar errores en la interacción entre componentes del frontend y backend, así como con servicios externos o bases de datos.

Debido a restricciones de tiempo y enfoque en la entrega funcional del sistema, las pruebas de integración aún no han sido implementadas de forma completa. No obstante, se contempla su desarrollo en etapas posteriores del proyecto o en una futura versión de mantenimiento.

Enfoque propuesto (si se implementaran)

En caso de implementar pruebas de integración, la estrategia sería la siguiente:

- En el backend (Spring Boot):
 - Utilizar Spring Boot Test, junto con anotaciones como `@SpringBootTest`, `@MockBean` y `@TestConfiguration` para levantar el contexto completo de la aplicación y simular llamadas HTTP o acceso a la base de datos.
 - Emplear herramientas como `TestRestTemplate` o `MockMvc` para simular peticiones REST y comprobar el correcto funcionamiento de los endpoints.
 - Probar flujos como:
 - Registro + aprobación + login
 - Creación de producto + listado en tabla
 - Venta registrada + visualización en panel de ventas
- En el frontend (React js):
 - Usar bibliotecas como `React Testing Library` y `Jest` para simular componentes y su comportamiento ante datos reales de la API (mockeados o con una API local).
 - Validar que al cargar datos reales desde el backend, los componentes de React renderizan correctamente la información y gestionan bien los estados de carga, error y éxito.
 - Verificar flujos como:
 - Autenticación completa
 - Visualización de ventas tras realizar una operación
 - Navegación entre vistas con estado compartido

Consideración futura

A pesar de no estar implementadas aún, se recomienda la incorporación de estas pruebas en futuras iteraciones para:

- Asegurar robustez en la comunicación entre capas.
- Reducir errores de integración en producción.
- Aumentar la confianza al realizar cambios o refactorizaciones.

- **Corrección de errores y optimización del rendimiento**

- Corrección de errores.

Los resultados obtenidos de las pruebas unitarias e integración se deben tener en cuenta para corregir errores y a ser posible, las pruebas deben ser continuas, ya que lo más común es que hayan errores no descubiertos aún.

También es muy importante tras realizar correcciones de errores, volver a realizar las pruebas que tuvieron éxito en un pasado, ya que al realizar cambios, hay probabilidades de romper algo que ya funcionaba.

- Optimización de rendimiento.

Antes meternos de lleno en las formas que existen de optimizar el rendimiento, hay que mencionar un punto clave en el desarrollo software, “la refactorización”, que hace referencia a un conjunto de prácticas cuya finalidad es reestructurar el código sin cambiar su comportamiento, con el objetivo de mejorar la calidad del código, lo que a su vez lo hará más legible y por ende más mantenible, tanto por la persona que lo desarrolló, como por otros desarrolladores. Algunas de las estrategias para refactorizar el código son:

- Dividir un método gigante en partes más pequeñas y encontrar una buena manera de orquestar esas partes.
- En partes donde las operaciones sean iguales, aislar ese método y utilizarlo donde corresponda.
- Usar constantes siempre que los valores esperados sean los mismos.
- Cambiar los nombres de los métodos, siempre y cuando esto mejore la comprensión.
- Mover métodos a las clases apropiadas.

Dicho esto existen diversas herramientas para ayudarnos a optimizar nuestra aplicación, en nuestro caso, como estamos trabajando con “Spring boot” y “React js”, para cada uno de ellos, disponemos de :

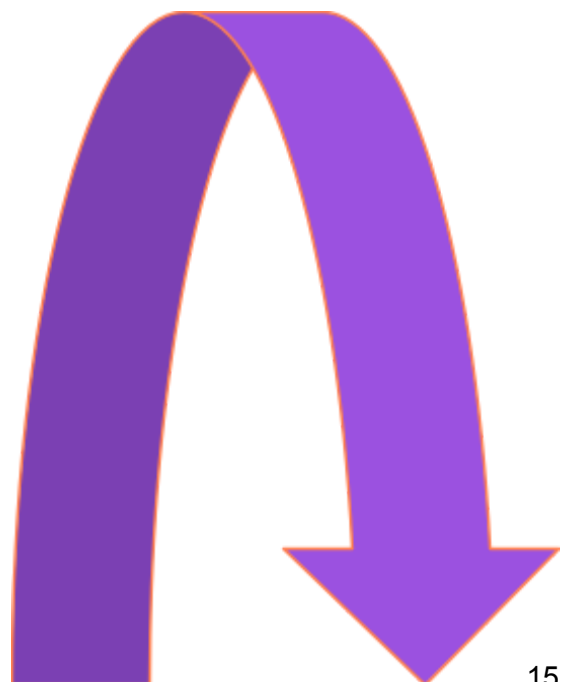
- Spring Boot

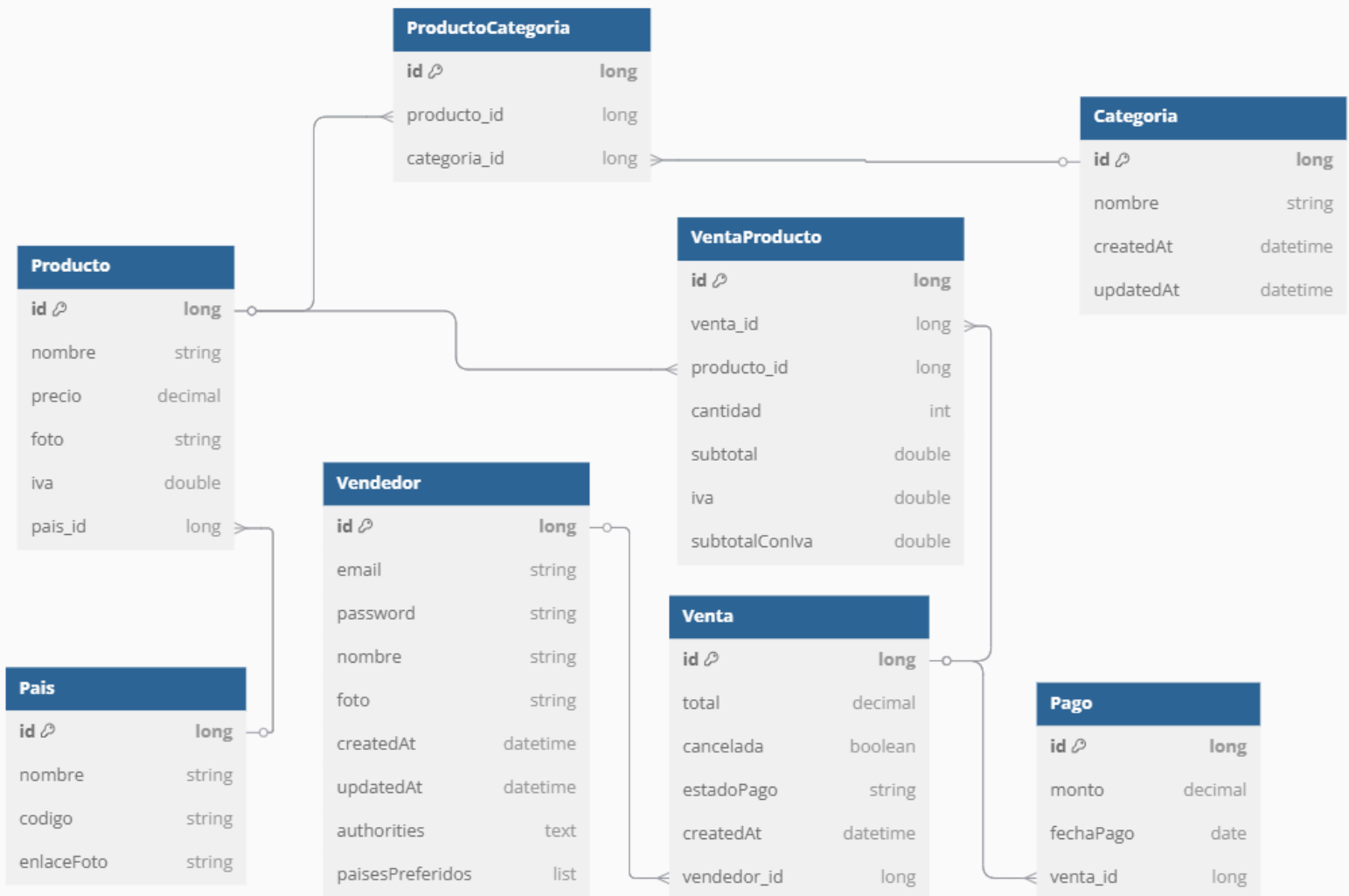
- Spring Boot Actuator: Permite monitorizar el estado de la aplicación, métricas de rendimiento, uso de memoria, hilos activos, entre otros. Es útil para detectar cuellos de botella y supervisar en tiempo real.
- Spring Cache: Permite la implementación de caché en métodos costosos o de consulta frecuente, como búsquedas en base de datos, para reducir el tiempo de respuesta y el uso de recursos.
- JPA y consultas optimizadas: Asegurarse de usar fetch joins en lugar de lazy loading cuando sea necesario, evitar consultas N+1, e indexar correctamente las tablas en la base de datos.
- Profiling y herramientas externas: Herramientas como VisualVM, JProfiler, o incluso Micrometer con Prometheus/Grafana pueden integrarse para una visión profunda del rendimiento.
- Compilación y empaquetado eficiente: Uso de herramientas como Spring Native (cuando sea viable) para convertir la aplicación a binario nativo, reduciendo considerablemente el tiempo de arranque y el uso de memoria.

- React js
 - React Developer Tools: Extensión de navegador que permite inspeccionar el árbol de componentes, identificar renders innecesarios y comprender la estructura de la aplicación.
 - React.memo y useMemo/useCallback: Permiten evitar renders innecesarios en componentes funcionales, mejorando la eficiencia en componentes con gran cantidad de datos o interacciones.
 - Lazy loading y code splitting: Usando React.lazy() y Suspense, se pueden cargar los componentes de manera diferida, dividiendo el bundle principal y mejorando los tiempos de carga inicial.
 - Herramientas de análisis de bundle: Como Webpack Bundle Analyzer o Source Map Explorer, que ayudan a identificar archivos grandes o dependencias innecesarias.
 - Optimización de imágenes y assets estáticos: Utilizar formatos modernos como WebP, compresión automática en tiempo de build y estrategias de caching del lado del cliente.

● DOCUMENTACIÓN

- **Documentación técnica**
 - .○.1. Backend
 - A. Modelo de datos (nivel base de datos)
 - a. Diagrama ER generado a partir de las entidades





b. Descripción de entidades:

- i. **Producto:** representa los artículos disponibles para la venta. Cada producto incluye campos como nombre, precio, IVA, imagen de referencia (foto) y una relación con el país de origen.
- ii. **Categoria:** define los distintos tipos de productos o servicios. Un mismo producto puede estar asociado a varias categorías, y viceversa, mediante la entidad intermedia ProductoCategoria.

- iii. **Pais:** almacena información geográfica, como el nombre, el código ISO y un enlace visual a su bandera, asociada a los productos.
- iv. **Vendedor:** representa tanto a usuarios con permisos de administrador como a vendedores estándar. Incluye datos de acceso (correo y contraseña), nombre, foto de perfil, roles asignados (authorities) y una lista de países preferidos.
- v. **Venta:** recoge los registros de ventas realizados. Cada venta contiene el total acumulado, su estado de pago (pendiente, parcial o pagado), una marca de cancelación (cancelada) y la fecha de creación.
- vi. **VentaProducto:** tabla intermedia que almacena las líneas de detalle de cada venta, vinculando un producto a una venta junto con la cantidad vendida, el subtotal, el IVA aplicado y el total con impuestos.
- vii. **Pago:** se asocia a una venta concreta y representa un abono realizado por el cliente. Registra el importe pagado y la fecha correspondiente.
- viii. **ProductoCategoría:** entidad de relación muchos a muchos entre productos y categorías.

Nota aclaratoria del modelo

- Las entidades **ProductoCategoría** y **VentaProducto** son tablas intermedias para gestionar relaciones complejas (muchos a muchos).
- En el caso de **VentaProducto**, se utiliza una clave primaria compuesta formada por los campos `venta_id` y `producto_id`.
- Algunos objetos, como **Vendedor**, **Venta** y **Categoría**, incorporan campos de auditoría como `createdAt` y `updatedAt`.
- El campo `estadoPago` permite clasificar el estado financiero de la venta.
- El campo `authorities` en **Vendedor** representa los roles asignados a cada usuario, como ADMIN, VENDEDOR o USER.

B. Persistencia con JPA y Spring Data

El modelo anterior se implementa mediante Java Persistence API (JPA) utilizando Spring Data JPA como capa de abstracción para el acceso a la base de datos. Esta arquitectura permite mantener una fuerte separación entre la lógica de negocio y la lógica de persistencia.

Mapeo de entidades

Cada entidad del sistema se implementa como una clase Java anotada con `@Entity`, que representa una tabla en la base de datos. Las relaciones entre entidades se definen mediante anotaciones estándar:

- `@Id`: identifica el campo clave primaria.
- `@ManyToOne`, `@OneToMany`, `@ManyToMany`: definen las asociaciones entre entidades.
- `@JoinColumn`: especifica claves foráneas cuando es necesario.
- `@EmbeddedId` o `@IdClass`: utilizadas para modelar claves compuestas (como en `VentaProducto`).

Repositorios

Para cada entidad, existe una interfaz que extiende `JpaRepository`, permitiendo operaciones CRUD sin necesidad de escribir SQL explícito. Además, es posible definir métodos personalizados por convención de nombres, como `findByEmail` o `findByNombreContainingIgnoreCase`.

Ventajas del uso de JPA

- Desacoplamiento entre lógica de negocio y persistencia.
- Transaccionalidad automática a nivel de servicio.
- Mayor mantenibilidad y escalabilidad, al evitar código repetitivo de acceso a datos.
- Facilidad para realizar pruebas unitarias y simular el comportamiento del sistema sin conexión directa a la base de datos.

C. Lógica de negocio (Servicios)

La aplicación sigue el patrón de separación de responsabilidades, dividiendo claramente la capa de presentación (Controladores) y la capa de lógica (Servicios). Los servicios (`@Service`) encapsulan las reglas de negocio y el procesamiento necesario para responder a las peticiones del usuario.

Estructura general

- Los controladores (`@RestController`) se encargan de:
 - Recibir y responder a las solicitudes HTTP.
 - Validar los datos de entrada más básicos.
 - Delegar en los servicios la lógica del dominio.
- Los servicios (`@Service`) implementan:
 - Validación específica de negocio.
 - Procesos de transformación o cálculo.
 - Acceso a la base de datos a través de los repositorios (si es necesario).
 - Flujo de operaciones encadenadas o atomizadas.

Ventajas de esta estructura

Escalabilidad: facilita la extensión de funcionalidades sin romper controladores.

Mantenibilidad: mejora la legibilidad del código y facilita los tests unitarios.

Reutilización: los mismos servicios pueden ser reutilizados por distintos controladores o tareas automatizadas (ej. generación programada de informes).

Desacoplamiento: los controladores no conocen detalles del modelo ni del acceso a datos.

D. Controladores y API REST

- a. Explicación del uso de `@RestController`, mapeos `@GetMapping`, `@PostMapping`, etc.
- b. Cómo se expone la lógica a través de endpoints RESTful.
- c. Ejemplo general del patrón DTO → Servicio → Entidad.

E. Seguridad

El sistema implementa una estrategia de seguridad basada en JWT (JSON Web Tokens) y control de acceso por roles, lo que permite una autenticación robusta y un control preciso sobre las operaciones que puede realizar cada usuario. Al funcionar en modo stateless, se ha deshabilitado la protección CSRF, ya que no se utilizan sesiones tradicionales sino tokens enviados en cada petición.

1. Enfoque general

- Autenticación basada en JWT.
- Control de acceso mediante anotaciones como `@PreAuthorize`.
- Arquitectura sin estado (stateless), lo que mejora la escalabilidad.
- CSRF deshabilitado de forma justificada.
- Contraseñas cifradas con `BCryptPasswordEncoder`.

2. Flujo de autenticación con JWT

Cada petición HTTP pasa a través del filtro `JwtFilter`, que intercepta el token JWT incluido en la cabecera `Authorization`. Este token es validado utilizando el componente `JwtTokenProvider`. Si el token es válido, se recupera el nombre de usuario y se crea un objeto `Authentication`, que se establece en el contexto de seguridad (`SecurityContextHolder`), permitiendo que la aplicación reconozca al usuario autenticado en los controladores protegidos.

3. Carga de usuario desde la base de datos

El servicio `UserDetailsServiceImpl` implementa la interfaz `UserDetailsService` y se encarga de recuperar el usuario desde la base de datos a través de su email. Esto es utilizado internamente por el proceso de autenticación para validar credenciales y cargar los roles del usuario.

4. Configuración del sistema de seguridad

La clase `SecurityConfig` define el comportamiento general de seguridad de la aplicación:

- Desactiva CSRF (`csrf().disable()`).
- Habilita CORS para permitir el acceso desde distintos orígenes.
- Define los endpoints públicos (como `/auth/**`, `/imagenes/**`, o `/swagger-ui/**`).
- Establece que cualquier otra ruta requiere autenticación.
- Configura la política de sesión como `STATELESS`.
- Añade el `JwtFilter` antes del filtro por defecto de autenticación de Spring.

Además, se configura un `AuthenticationManager` con un `DaoAuthenticationProvider` que utiliza `UserDetailsServiceImpl` y `BCryptPasswordEncoder`.

5. Generación y validación de tokens JWT

La clase `JwtTokenProvider` se encarga de generar tokens JWT tras una autenticación exitosa. Estos tokens incluyen información como el ID del usuario, su email, nombre de usuario y una fecha de expiración. La firma se realiza mediante HMAC usando una clave secreta.

También ofrece métodos para:

- Validar tokens (`isValidToken`).
- Extraer el nombre de usuario (`getUsernameFromToken`).
- Obtener el ID del usuario (`getIdFromToken`).

6. Gestión de contraseñas

Para garantizar la seguridad de las credenciales, las contraseñas se almacenan cifradas mediante `BCryptPasswordEncoder`, un algoritmo de hash adaptativo que dificulta los ataques por fuerza bruta.

F. Gestión de archivos e imágenes

La aplicación implementa un sistema interno de gestión de imágenes diseñado para manejar de forma segura y eficiente los archivos subidos por los usuarios, especialmente fotografías de productos y vendedores.

1. Servicio FotoService

El servicio `FotoService` encapsula toda la lógica relacionada con el tratamiento de imágenes en el backend. Sus principales responsabilidades incluyen:

- Validación del tipo MIME: Solo se aceptan imágenes con tipos seguros como `image/jpeg`, `image/png`, `image/gif`, `image/webp`, entre otros. Esta medida previene la subida de archivos maliciosos.
- Control de tamaño: Se establece un límite de 10 MB por archivo para evitar abusos en el almacenamiento local y preservar el rendimiento del sistema.
- Renombrado con UUID: Para evitar conflictos y garantizar unicidad, cada archivo es renombrado usando un identificador único (UUID) al momento de guardarlo.
- Redimensionamiento automático: Antes de ser almacenada, cada imagen es procesada con la biblioteca `Thumbnailator`, reduciendo su tamaño a un ancho máximo de 1000 píxeles, manteniendo la proporción original. Esto permite optimizar el consumo de ancho de banda y mejorar el rendimiento en el frontend.
- Gestión por subdirectorios: Las imágenes se almacenan en subcarpetas dentro del directorio raíz de subidas (por ejemplo, `uploads/productos/` o `uploads/vendedores/`), facilitando su organización y mantenimiento.
- Eliminación segura: Se incluyen métodos para eliminar imágenes asociadas a entidades cuando ya no son necesarias, con verificación previa de su existencia para evitar errores.

2. Almacenamiento local simplificado

Como solución práctica para el alcance del proyecto, las imágenes se almacenan en el sistema de archivos del servidor, específicamente en la ruta relativa `resources/uploads/` del backend. Aunque este enfoque no es el más escalable, resulta adecuado para un entorno de desarrollo o despliegue limitado.

En caso de evolución futura del sistema, se recomienda desacoplar este almacenamiento utilizando servicios en la nube como AWS S3, Google Cloud Storage o Cloudinary, lo cual mejoraría la escalabilidad, disponibilidad y seguridad de los archivos.

3. Procesamiento de imágenes

La clase auxiliar `ImageService` se encarga exclusivamente del redimensionamiento. Esto garantiza una separación de responsabilidades y facilita el mantenimiento del código. La imagen se convierte en `BufferedImage`, se ajusta su tamaño, y se vuelve a convertir en un arreglo de bytes listo para ser guardado.

G. Inicialización de datos

- a. Clase `DataInitializer`: admins, vendedor, categorías, países.
- b. Descarga dinámica desde API externa o fallback local.

H. Generación de tickets de venta

Descripción general

El sistema incluye una funcionalidad para generar tickets de venta en formato PDF a través del servicio `VentaTicketService`. Esta característica permite emitir comprobantes simples y legibles para el cliente, reforzando tanto la profesionalidad del servicio como la trazabilidad de las transacciones.

Tecnología utilizada

- Librería: `iText` para generación de documentos PDF.
- Formato de salida: Tamaño A6 (ideal para impresión térmica de tickets).
- Codificación: Retorno del ticket en un `byte[]`, facilitando su descarga o envío por correo.

Estructura del ticket

El contenido del ticket incluye:

- Encabezado corporativo:
 - Nombre de la empresa.
 - Dirección física (en este caso, fija como "Vinland 13600").
- Datos de la venta:
 - Estado del pago (pendiente, parcial o pagado).
 - Número total de productos.
 - Fecha y hora de creación.
 - Nombre del vendedor que registró la venta.
- Detalle por producto:
 - Nombre del producto.
 - Cantidad vendida.

- Subtotal sin IVA.
 - Porcentaje y valor del IVA aplicado.
 - Total con IVA por línea.
- Resumen de IVA:
 - Muestra los distintos tipos de IVA aplicados y su acumulado.
- Totales generales:
 - Total de la venta.
 - Monto pagado hasta el momento.
 - Saldo pendiente en caso de pago parcial.
- Cierre del ticket:
 - Mensaje de agradecimiento al cliente.

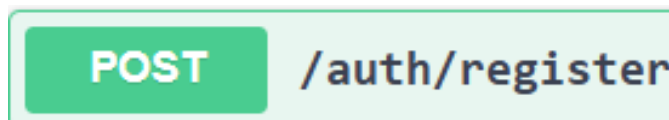
Consideraciones técnicas

- Se utiliza VentaProductoRepository para obtener todas las líneas de venta asociadas a una transacción.
- El formato de fecha se maneja con DateTimeFormatter, asegurando un formato legible (día/mes/año y hora:minuto).
- Los cálculos de totales y subtotales se realizan con BigDecimal y RoundingMode.HALF_UP para precisión en los decimales.
- Los nombres de los archivos generados siguen la convención: ticket_locuventas_dd-MM-yyyy_HH-mm.pdf.

I. Endpoints principales

Nota: Si despliegas el proyecto en local, tienes la posibilidad de hacer pruebas en:

<http://localhost:8080/swagger-ui/index.html>



Content-Type: multipart/form-data

Autenticación: No requiere token

Descripción : Registra un nuevo usuario en el sistema LocuVentas. Este endpoint espera datos JSON del usuario junto con una imagen de perfil enviada mediante un formulario multipart.

Parámetros del cuerpo (form-data)

Campo	Tipo	Requerido	Descripción
user	JSON en texto	Si	Datos de registro: nombre, email, password
foto	Archivo	Si*	Imagen de perfil en formatos permitidos

Aunque en el código está marcado como required = false, la lógica de negocio requiere que se suba una foto.

Ver: AuthController.java:50.

Esquema del campo user (UserRegisterDTO)

```
{
  "nombre": "string (entre 3 y 50 caracteres, solo letras y espacios)",
  "email": "string (formato email válido, máx. 100 caracteres)",
  "password": "string (mínimo 8 caracteres, al menos 1 mayúscula, 1 minúscula, 1 número y 1 carácter especial)"
}
```

Validaciones de la imagen

- Formatos permitidos: JPEG, PNG, GIF, WEBP, AVIF
 - (FotoService.java:17)
- Tamaño máximo: 10 MB
 - (FotoService.java:18)
- Procesamiento: La imagen se redimensiona automáticamente a un ancho máximo de 1000px.

En el frontend se gestiona desde: FormVendedorRegister.jsx:52-55

Respuesta exitosa (201 Created)

```
{
  "email": "usuario@ejemplo.com"
}
```

Posibles errores (400 Bad Request)

```
{
  "email": "Email ya utilizado",
  "foto": "Debes seleccionar una foto.",
  "nombre": "El nombre requiere mínimo tres caracteres y máximo 50"
}
```

POST

/auth/login

Autenticación: No requiere token (endpoint público)

Descripción

Autentica a un usuario en el sistema LocuVentas y devuelve un token JWT junto con la información del usuario.

Solo los usuarios con rol VENDEDOR o ADMIN pueden iniciar sesión correctamente.
Los usuarios que solo tengan el rol USER recibirán una respuesta 403 Forbidden.

Cuerpo de la petición (LoginRequestDTO.java:7-10)

Campo	Tipo	String	Descripción
email	String	Si	Correo electrónico del usuario
password	String	Si	Contraseña del usuario

Ejemplo de solicitud:

```
{
  "email": "usuario@ejemplo.com",
  "password": "Password123!"
}
```

Proceso de autenticación

- Verifica las credenciales con AuthenticationManager de Spring Security
- Extrae los roles del usuario autenticado
- Comprueba que tenga el rol VENDEDOR o ADMIN
- Si es autorizado, genera un token JWT
- Devuelve los datos del usuario + token
 - Código relevante: AuthController.java:101-152

Respuesta exitosa (200 OK) — LoginResponseDTO.java:10-16

```
{
  "email": "usuario@ejemplo.com",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "nombre": "Juan Pérez",
  "foto": "imagen_perfil.jpg",
  "roles": ["ROLE_VENDEDOR"]
}
```

Sin permisos de rol (403 Forbidden)

```
{
  "message": "Su cuenta aún no ha sido habilitada como vendedor. Espere a que un administrador le otorgue permisos.",
  "path": "/auth/login",
  "timestamp": "2024-01-15T10:30:00.000Z"
}
```



```
}
```

Credenciales incorrectas (401 Unauthorized)

```
{  
  "message": "Credenciales erróneas",  
  "path": "/auth/login",  
  "timestamp": "2024-01-15T10:30:00.000Z"  
}
```

Frontend: Este endpoint se maneja en FormVendedorLogin.jsx:21-58

Configuración de seguridad: Es público en SecurityConfig.java:53

Notas

- A diferencia del endpoint de registro (/auth/register), aquí no se suben archivos, por lo tanto se utiliza application/json.
- El sistema aplica control de acceso por roles: usuarios sin el rol adecuado no podrán avanzar.
- El token JWT devuelto debe incluirse en los headers Authorization en las peticiones futuras (Bearer <token>).



Content-Type: No aplica (no se envía cuerpo)

Autenticación: Requiere token JWT con rol ADMIN

Descripción

Elimina un usuario del sistema LocuVentas, incluyendo su foto de perfil asociada.

Esta operación es permanente y no se puede deshacer.

Está pensada exclusivamente para administración.

Parámetro	Tipo	Obligatorio	Descripción
id	Long	Si	ID único del usuario a eliminar

Ejemplo de solicitud

```
DELETE /usuarios/123  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Detalles de implementación (AuthController.java:326-347)

- Verifica que el usuario exista en la base de datos
- Intenta eliminar la foto del sistema de archivos (uploads/)
- Elimina el registro del usuario en la base de datos
- Devuelve confirmación de éxito o error

Respuesta exitosa (200 OK)

```
{
  "message": "Usuario y foto eliminados correctamente."
}
```

Usuario no encontrado (404 Not Found)

```
{
  "error": "Usuario no encontrado."
}
```

Requiere autenticación/rol (401 o 403)

```
{
  "error": "Debes iniciar sesión como admin."
}
```

Uso en el frontend

Este endpoint es utilizado desde el panel de administración para gestionar usuarios pendientes.

Implementaciones visibles en:

- VendedoresPendientes.jsx:67-78
- PendientesList.jsx:50-60

Configuración de seguridad

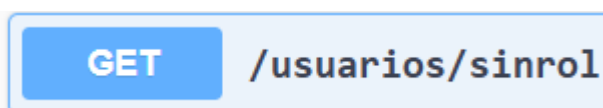
Restringido a usuarios con rol ADMIN, gracias a la anotación:

```
@PreAuthorize("hasRole('ADMIN')")
```

Notas adicionales

Si ocurre un error al eliminar la foto, la eliminación del usuario continúa, para no bloquear la operación.

Se usa principalmente en el flujo donde un administrador rechaza registros de vendedores pendientes.



Content-Type: No aplica

Autenticación: Requiere token JWT con rol ADMIN

Descripción

Este endpoint devuelve una lista de usuarios que aún no han sido aprobados como vendedores ni administradores, es decir, aquellos que solo tienen el rol básico ROLE_USER.

Está diseñado específicamente para que los administradores gestionen usuarios pendientes de aprobación.

Implementación técnica (AuthController.java:307-325)

- Usa `@PreAuthorize("hasRole('ADMIN')")` para restringir el acceso solo a administradores
- Consulta la base de datos y filtra usuarios que no tienen ROLE_ADMIN ni ROLE_VENDEDOR
- Devuelve una lista de entidades Vendedor con solo ROLE_USER

```
[
  {
    "id": 27,
    "nombre": "Carlos López",
    "email": "carlos@example.com",
    "foto": "f77d9210.png",
    "roles": ["ROLE_USER"]
  },
  {
    "id": 31,
    "nombre": "María Gómez",
    "email": "maria@example.com",
    "foto": "a9e810b2.jpg",
    "roles": ["ROLE_USER"]
  }
]
```

Contexto del sistema de roles (Roles.java:3-6)

Rol	Descripción
ROLE_ADMIN	Acceso total al sistema
ROLE_VENDEDOR	Acceso a operaciones de venta
ROLE_USER	Rol asignado por defecto al registrarse

PUT

/usuarios/{id}/asignar-rol

Content-Type: No aplica (no necesita cuerpo en la petición)

Autenticación: Requiere token JWT con rol ADMIN

Descripción

Este endpoint permite que un administrador asigne el rol de VENDEDOR a un usuario previamente registrado (que solo tenía el rol básico USER).

Es parte del flujo de aprobación de nuevos usuarios en el sistema LocuVentas.

Parámetros

Parámetro	Tipo	Obligatorio	Descripción
id	Long	<input checked="" type="checkbox"/> Sí	ID del usuario a quien asignar el rol

Implementación técnica (AuthController.java:179-237)

El endpoint realiza las siguientes validaciones:

- Verifica que el usuario esté autenticado como ADMIN (@PreAuthorize)
- Previene que un admin se asigne el rol a sí mismo (.getId().equals(id))
- Comprueba si el usuario ya tiene el rol VENDEDOR
- Asigna el rol VENDEDOR y actualiza en base de datos

Uso en el frontend

Este endpoint se utiliza en varios componentes de administración:

- VendedoresPendientes.jsx:61: Función handleAprobar
- PendientesList.jsx:43: Confirmación y llamada al backend

Flujo de asignación de rol

- El administrador accede a /usuarios/sinrol para ver los usuarios sin rol
- Visualiza un botón "Aprobar" junto a cada usuario pendiente
- Aparece un modal de confirmación
- Se envía la solicitud PUT /usuarios/{id}/asignar-rol
- El backend valida y asigna el nuevo rol
- El frontend actualiza la lista de usuarios pendientes
- Se muestra una notificación de éxito

Validaciones de seguridad implementadas

- Autenticación y autorización mediante JWT
- Prevención de autoasignación de rol (admin \neq usuario)
- Prevención de asignaciones duplicadas (ya tenía VENDEDOR)

Uso en el frontend

Este endpoint es utilizado en la pantalla de gestión de usuarios pendientes:

- VendedoresPendientes.jsx:37-41
- Se muestra una lista de usuarios sin rol
- Desde esa vista, el administrador puede:
 - Aprobar al usuario (/usuarios/{id}/asignar-rol)
 - Eliminar al usuario (/usuarios/{id})

Flujo de aprobación

- Un usuario se registra → se le asigna solo ROLE_USER
- Un administrador consulta /usuarios/sinrol
- Puede aprobar (asignar rol VENDEDOR) o eliminar al usuario

Notas adicionales

Este endpoint no devuelve usuarios con roles elevados (solo filtra los de tipo ROLE_USER)
Forma parte del flujo de moderación de nuevos registros

PUT

/usuarios/editar-perfil

Editar perfil del usuario autenticado

Permite editar nombre, email, contraseña y subir una nueva foto de perfil.

Request body required

multipart/form-data

user	Datos del usuario en formato JSON (nombre, email, password)
string(\$binary)	
foto	Archivo de la nueva foto de perfil
string(\$binary)	

GET

/países

Obtener listado de países

Devuelve todos los países disponibles en el sistema. Solo accesible para usuarios con rol ADMIN o VENDEDOR.

Se utiliza para dar de alta los productos, ya que estos tienen asociaciones con países.

GET

/imagenes/{tipo}/{filename}

Obtener una imagen por tipo y nombre

Devuelve una imagen almacenada en el servidor. Solo permite acceder a carpetas seguras: **vendedores**, **productos**, **productosprecargados**.

Útil para que un vendedor visualice sus fotografías, también para que pueda ver las fotos de los usuarios a los que puede aprobar o denegar, y claro está para los productos también

GET**/categorias** Obtener todas las categorías

Devuelve una lista de todas las categorías disponibles. Requiere rol VENDEDOR o ADMIN.

Por el momento sirve para que el admin pueda dar de alta los productos.

Los tres siguientes endpoints solo son usables por un administrador:

PUT**/productos/{id}** Editar un producto existente

Permite actualizar datos de un producto y cambiar su imagen si se desea.

DELETE**/productos/{id}** Eliminar un producto

Elimina un producto por ID, siempre que no haya sido vendido previamente.

POST**/productos** Crear un nuevo producto

Permite crear un producto nuevo incluyendo datos JSON y una imagen.

GET	/productos	Listar productos
Devuelve una lista paginada de productos.		

Se usa para pintar los productos dentro del punto de venta y para listarlo en la tabla de gestión.

GET	/ventas	Listar todas las ventas
Devuelve una lista paginada de ventas del usuario autenticado. Si el usuario es administrador, devuelve todas las ventas.		

POST	/ventas	Crear una nueva venta
Crea una venta con una o más líneas. Requiere autenticación. El vendedor se asigna automáticamente desde el token.		

POST	/ventas/{id}/pago	Registrar pago a una venta
Registra un pago parcial o total a una venta existente, identificada por su ID.		

PATCH	/ventas/{id}/cancelar	Cancelar una venta
Permite cancelar una venta existente. Solo accesible por administradores.		

GET	/ventas/{id}	Obtener detalles de una venta
Devuelve toda la información detallada de una venta específica por su ID, incluyendo sus productos, pagos y estado.		

GET**/ventas/{id}/ticket-pdf** Descargar ticket en PDF

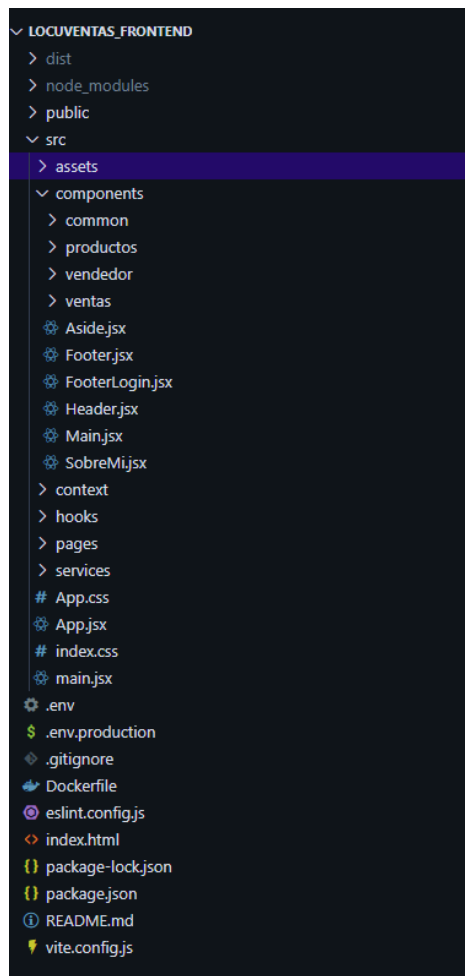
Devuelve un archivo PDF con el ticket de la venta especificada.

GET**/ventas/pendientes** Listar ventas pendientes o parcialmente pagadas

Devuelve una lista paginada de ventas cuyo estado de pago es PENDIENTE o PARCIAL. Solo accesible para ADMIN o VENDEDORES. El ADMIN ve todas, el vendedor solo las suyas.

1.1. Frontend

Estructura general:



Para facilitar la organización del código y evitar rutas relativas largas y propensas a errores (`.././pages/...`), se configuraron alias de importación en el proyecto mediante el

archivo vite.config.js. Esto permite importar componentes, páginas o servicios de forma más limpia y mantenible.

Dentro de services, destaca el archivo [“api.js”](#):

Centraliza toda la lógica necesaria para interactuar con la API REST del backend. Esta capa actúa como intermediario entre los componentes del frontend y los endpoints expuestos por el servidor.

Responsabilidades principales:

- Leer la URL base desde las variables de entorno (VITE_API_URL).
- Incluir automáticamente el token JWT en las cabeceras de las peticiones, salvo para login y registro.
- Adaptarse al tipo de payload (application/json o FormData) según la configuración.
- Procesar respuestas con o sin cuerpo (incluye fallback para respuestas vacías).
- Gestionar errores propagando la respuesta parseada del backend.

Sistema de autenticación y control de acceso

La aplicación implementa un sistema de autenticación basado en tokens JWT, que permite verificar la identidad del usuario y controlar el acceso a las distintas secciones del sistema. Los datos del usuario autenticado (token, nombre, roles, etc.) se almacenan en localStorage y se gestionan a través de un contexto global (AuthContext), creado con la API de Context de React.

Este contexto se encarga de:

- Inicializar el estado de autenticación al arrancar la aplicación.
- Persistirlo ante cambios.
- Detectar automáticamente tokens expirados para cerrar sesión de forma segura.

El acceso al contexto se realiza mediante un hook personalizado (useAuth), lo que permite que cualquier componente pueda consultar el estado del usuario o realizar acciones como logout.

Para garantizar que solo los usuarios autorizados accedan a determinadas rutas, se ha desarrollado un componente PrivateRoute. Este componente actúa como una capa de protección que:

- Redirige a usuarios no autenticados al login.
- Bloquea el acceso si el usuario no tiene el rol adecuado (ROLE_VENDEDOR o ROLE_ADMIN).
- Muestra un mensaje claro en caso de que la cuenta aún esté pendiente de aprobación.

La navegación de la app se gestiona con react-router-dom, diferenciando claramente entre rutas públicas (como login o “Sobre mí”) y rutas privadas (dashboard, gestión de ventas, productos, etc.). Cada ruta sensible está protegida mediante el uso de PrivateRoute.

Desde el punto de entrada de la aplicación (main.jsx), toda la estructura está envuelta dentro del componente AuthProvider, asegurando que el contexto de autenticación esté disponible desde el primer renderizado. Además, se utiliza StrictMode para seguir las buenas prácticas de desarrollo y detectar posibles errores durante la fase de desarrollo.

Este enfoque proporciona un sistema de control de acceso sólido, escalable y alineado con los estándares modernos de desarrollo en React.

Componentes clave del frontend(Sistema de ventas)

El sistema está compuesto por múltiples componentes reutilizables y modulares. A continuación se destacan los más relevantes por su papel en la experiencia de usuario y la lógica funcional de la aplicación.

Dashboard.jsx

Actúa como el núcleo operativo de la aplicación para usuarios vendedores y administradores. Orquesta la interacción entre los principales módulos: Header, Main, Aside y Footer. Gestiona el flujo de una venta (agregar productos, quitar productos, guardar o finalizar y cobrar) y maneja el estado de las modales de pago y detalle de venta.

Main.jsx

Se encarga de mostrar los productos disponibles para la venta. Obtiene los datos del backend con paginación y renderiza cada producto mediante el componente ProductoSimpleCard. Adapta la disposición del grid al tamaño de pantalla y permite añadir productos a la carga activa.

ProductoSimpleCard.jsx

Representa visualmente un producto individual. Muestra nombre, imagen, precio, país y categorías. Es interactivo y está optimizado para proporcionar feedback visual al usuario al seleccionarlo. Si el producto está ya en la carga, muestra un contador superpuesto.

Aside.jsx

Muestra el resumen de la venta en curso, incluyendo los productos añadidos, base imponible, IVA y total. También ofrece botones para guardar o finalizar y cobrar. Refleja en tiempo real el número de productos, la hora y el usuario autenticado.

Footer.jsx

Proporciona navegación rápida entre las secciones principales: nueva venta, todas las ventas y pendientes de pago. Detecta la ruta activa y adapta los botones según el contexto. Es fijo en la parte inferior para mantener la accesibilidad.

Utilidades y estilos comunes

Se emplean componentes como Boton, BotonClaro, ModalPago, Paginacion o ModalDetalleVenta, así como servicios como apiRequest para la comunicación con el backend. El diseño está pensado para ser accesible y adaptable a distintos tamaños de pantalla, aplicando estilos de TailwindCSS con clases dinámicas.

Componente Header(Barra de navegación)

El componente Header actúa como barra de navegación principal de la aplicación y está diseñado para ofrecer una experiencia responsiva y personalizada según el rol del usuario autenticado (administrador o vendedor).

Funciones principales:

- Identificación del usuario: muestra el nombre, email y avatar del usuario activo. También incluye un acceso directo a la edición de perfil y al cierre de sesión.
- Gestión de navegación condicional: muestra distintas opciones según el rol:
 - Los administradores acceden a menús desplegables con opciones de gestión (vendedores, productos, categorías).
 - Los vendedores visualizan únicamente las secciones relacionadas con su rol.
- Diseño adaptable: se adapta a dispositivos móviles mostrando un menú colapsable con animaciones suaves, manteniendo todas las opciones clave accesibles.
- Gestión de estado y modales: utiliza múltiples useState para controlar el estado de visibilidad de menús, submenús y modales (confirmación de cierre de sesión, edición de perfil, gestión de usuarios).

Destacados técnicos:

- Uso de lucide-react para íconos dinámicos (hamburguesa/cerrar).
- Aplicación de efectos visuales como transiciones, backdrop-blur, y colores degradados para mejorar la estética.
- Manejadores de eventos globales (mousedown) para cerrar menús al hacer clic fuera.
- Control centralizado de modales reutilizables, pasando la configuración a través de props (mensaje, onConfirmar, etc.).
- Totalmente integrado con el sistema de autenticación (useAuth) y navegación (react-router-dom).

Este componente no solo centraliza la navegación y acceso del usuario, sino que contribuye activamente a la experiencia general de uso, manteniendo la interfaz coherente, accesible y adaptada al contexto de cada perfil.

Componentes auxiliares y de interfaz

A continuación se describen los componentes visuales y funcionales más destacados que, si bien no forman parte directa del flujo de venta o autenticación, aportan valor clave a la usabilidad, estética y gestión de usuarios en la aplicación.

Gestión de usuarios y navegación

AvatarUsuario

Este componente se muestra en la cabecera para usuarios autenticados. Presenta la foto, nombre y correo del usuario, y permite acceder a acciones personales como editar el perfil, cerrar sesión o visitar el perfil del desarrollador. Incluye un menú desplegable y un modal de confirmación para el logout.

GestionDropdown + VendedoresDropdown

Permiten a los administradores acceder a secciones específicas como la gestión de productos, usuarios y categorías. Ambos están diseñados como menús contextuales anidados, compatibles con escritorio y móviles, e integran confirmaciones para acciones sensibles.

PendientesList

Muestra una lista dinámica de usuarios registrados que aún no han sido aprobados. Cada tarjeta de usuario contiene nombre, email, fecha de alta relativa y botones para aprobar o eliminar. Utiliza un panel flotante, cargado asíncronamente desde el backend, con confirmación previa a cada acción.

Formularios y subida de archivos

UploadAvatar

Componentes reutilizables para carga de imágenes, tanto cuadradas (productos) como circulares (avatar de usuario). Muestran una imagen previa y aplican un pequeño efecto visual al cambiar. En caso de error al cargar, se usa una imagen por defecto. La interacción es intuitiva: basta con hacer clic en la imagen para seleccionar un archivo.

Detalles técnicos comunes


- Todos los componentes usan TailwindCSS con clases dinámicas para lograr una experiencia visual reactiva y moderna.
- Se emplea useRef para manejar entradas ocultas (inputs de tipo file), y useState para gestionar la previsualización y transiciones.
- Se integran con react-toastify para mostrar notificaciones no intrusivas tras acciones administrativas o cambios de estado.
- Las rutas, acciones y roles están sincronizados con el sistema de autenticación (useAuth) y el backend, a través de apiRequest.

○ Documentación del usuario(Escritorio)

Login



Registro



Registro

Recuerda que en Locuventas es obligatorio proporcionar una foto al registrarse!

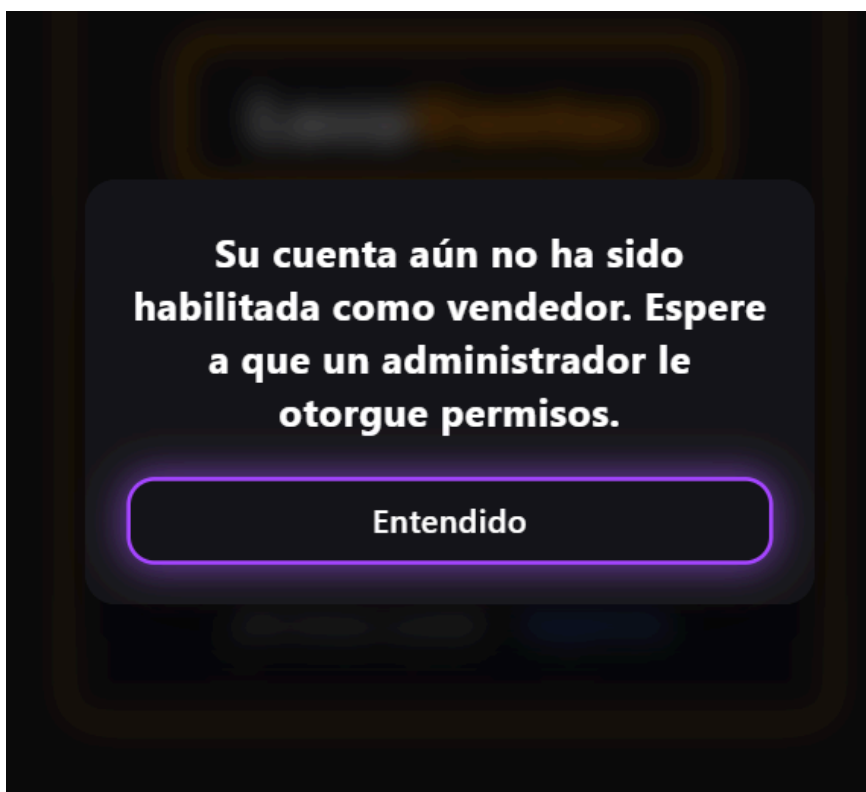
Dinos cómo te llamas

Correo electrónico

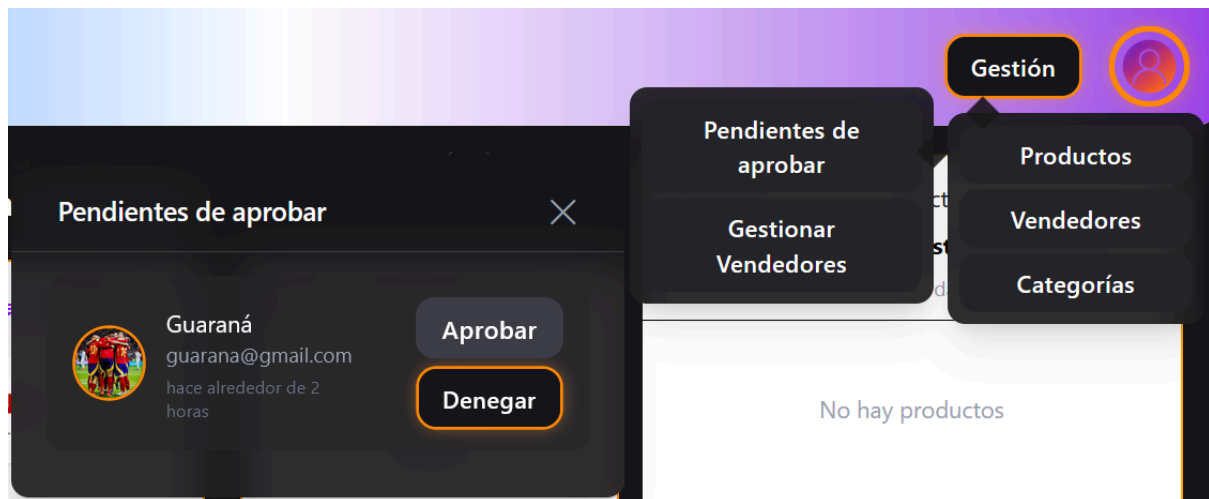
Contraseña

Registrarse

Login tras registro exitoso

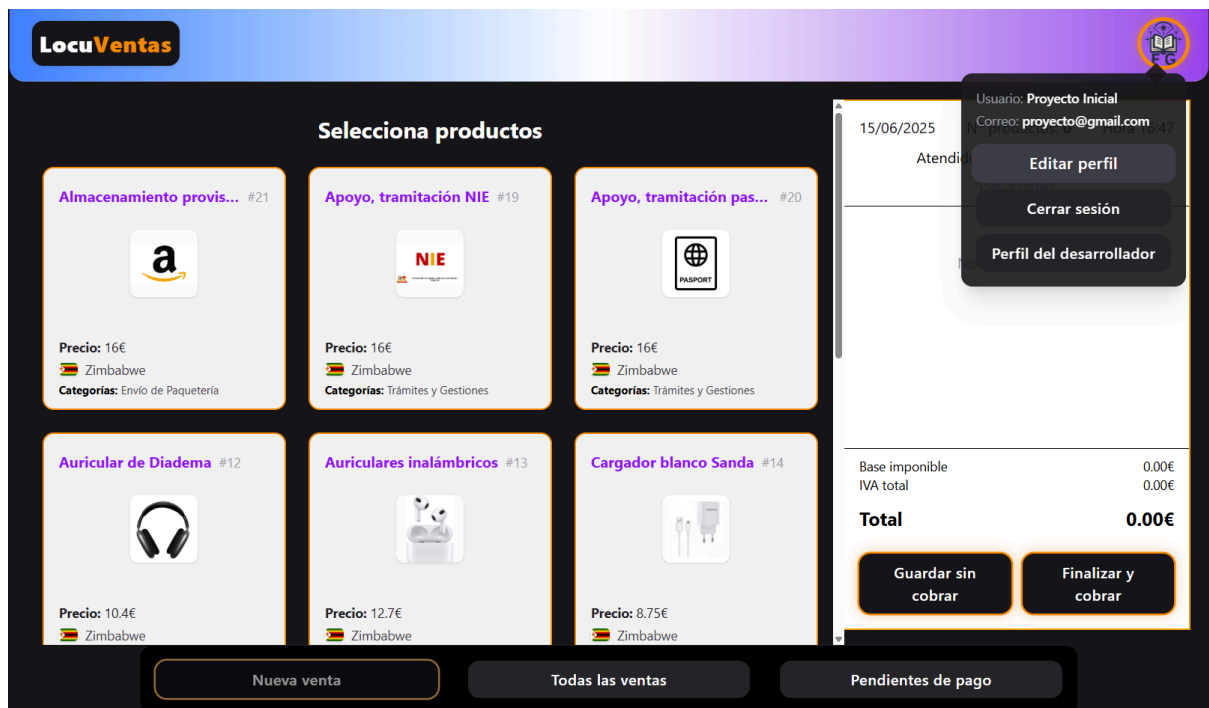


Vista del administrador, tras el registro de un nuevo usuario



En caso de pulsar “Denegar”, el usuario se elimina de la base de datos y la foto que se subió a “uploads/vendedores” también.

En caso de ser aprobado, puede iniciar sesión satisfactoriamente:



Como podemos observar, este usuario al no ser administrador, no se muestra el botón de gestión.

El usuario tiene la posibilidad de editar sus datos:

Como se observa en la imagen, a la derecha aparecen los productos, con el botón “-”, por si lo queremos quitar del carrito. Si pulsamos “Guardar sin cobrar”, la venta se registra con estado “PENDIENTE” y salta la ventana del detalle de la venta:

Detalle de Venta #1

Total: 49.83 €

Pagado: 0.00 €

Saldo pendiente: 49.83 €

Estado: PENDIENTE

Vendedor: Francisco Javier

Fecha: 15/6/2025, 15:05:25

Productos:

Impresiones Premium × 2

Subtotal sin IVA: 4.18 €

IVA (10 %): 0.42 €

Total línea: 4.60 €

CLARO × 2

Subtotal sin IVA: 21.98 €

IVA (21 %): 4.62 €

Total línea: 26.60 €

Internet con Boost × 1

Subtotal sin IVA: 15.40 €

IVA (21 %): 3.23 €

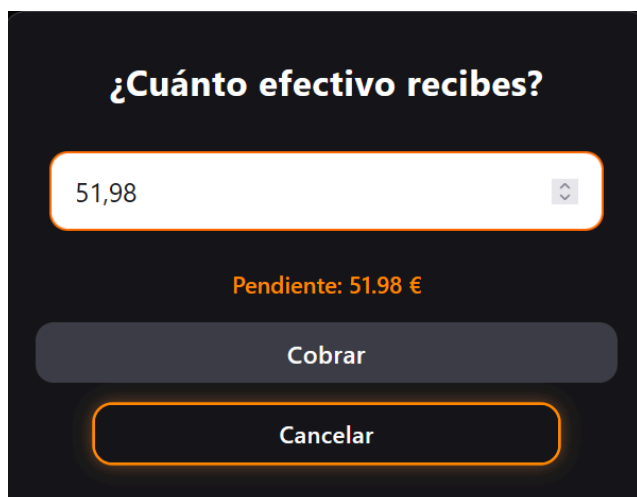
Total línea: 18.63 €

Descargar ticket

Cerrar

Si pulsas sobre “Descargar ticket”, obtienes un pdf detallado de la venta.

Si realiza una venta con la opción de “Finalizar y cobrar”, salta una ventana para confirmar el monto:



¿Cuánto efectivo recibes?

51,98

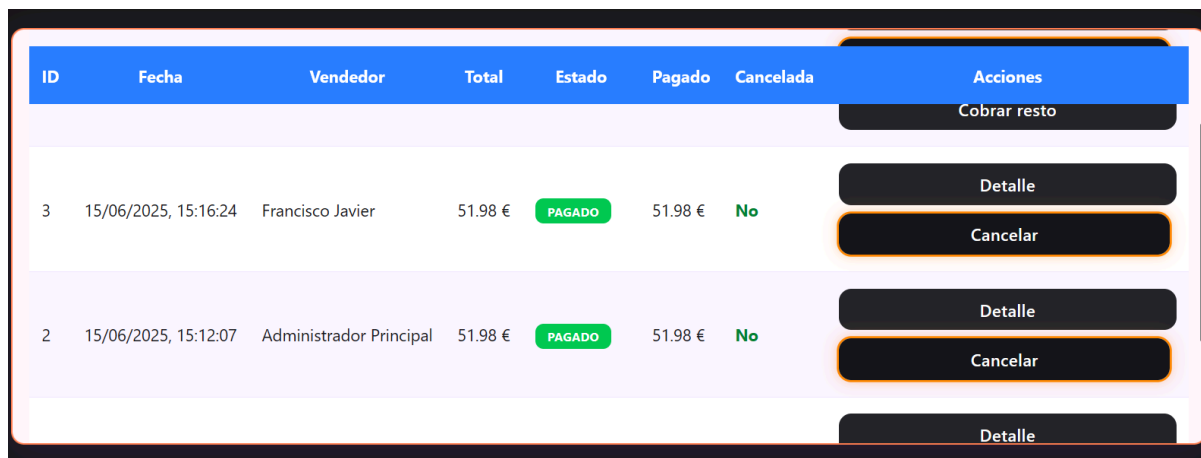
Pendiente: 51.98 €

Cobrar

Cancelar

Si se cobra el monto total de la venta, el estado de esa venta es “PAGADO”, si es un monto inferior “PARCIAL”, en ambos casos, volvemos a tener la ventana de detalle, con las mismas opciones del apartado anterior.

En “Todas la ventas”, tenemos todas nuestras ventas, en caso de ser “ADMIN”, podremos ver todas las ventas, de todos los usuarios.













ID	Fecha	Vendedor	Total	Estado	Pagado	Cancelada	Acciones
							Cobrar resto
3	15/06/2025, 15:16:24	Francisco Javier	51.98 €	PAGADO	51.98 €	No	Detalle Cancelar
2	15/06/2025, 15:12:07	Administrador Principal	51.98 €	PAGADO	51.98 €	No	Detalle Cancelar
							Detalle

En “Pendientes de pago”, tenemos las ventas con estado “PENDIENTE” Y “PARCIAL”, solo los administradores pueden ver las ventas todos.

En las dos tablas anteriores tenemos la opción de cobrar el resto y la opción de cancelar una venta solo está disponible para administradores.

Gestión de productos(Solo para administradores)

+ Agregar producto

ID	Foto	Nombre	Precio	País	Categorías	Acciones	
21		Almacenamiento provisional de Amazon	16€	 Zimbabwe	Envío de Paquetería	Editar	Eliminar
19		Apoyo, tramitación NIE	16€	 Zimbabwe	Trámites y Gestiones	Editar	Eliminar
20		Apoyo, tramitación pasaporte	16€	 Zimbabwe	Trámites y Gestiones	Editar	Eliminar
12		Auricular de Diadema	10.4€	 Zimbabwe	Venta de Accesorios	Editar	Eliminar
13		Auriculares inalámbricos	12.7€	 Zimbabwe	Venta de Accesorios	Editar	Eliminar


<

Página 1 de 3

>

Desde aquí el administrador puede modificar los productos ya existentes, agregar nuevos o borrarlos.

Agregar Producto



Nombre

Jamón cerrano

Precio

101

IVA

4

Selecciona un país

Spain

Selecciona categorías

Guardar

Cancelar

31		Jamón cerrano	101€	 Spain	Alimentos y Bebidas	Editar	Eliminar
----	---	---------------	------	---	---------------------	--------	----------

Si el administrador quiere volver al punto de venta, solo tiene que pulsar sobre el logo de la aplicación para ello.

○ Manual de instalación y configuración

Requisitos previos

Antes de empezar, asegúrate de tener instalado:

- Docker
- Docker Compose

En algunas versiones de Docker, el comando es docker-compose (con guion), en otras es docker compose (sin guión). Ambos funcionan si Docker está correctamente instalado.

Pasos

- Crea un archivo: **docker-compose.yml**
Cuyo contenido es(Respetar el formato y sus tabulaciones):

```
version: '3.8'

services:
  db:
    image: mysql:8.0
    container_name: locuventas_db
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
    ports:
      - "${MYSQL_PORT}:3306"
    volumes:
      - mysql_data:/var/lib/mysql
    networks:
      - locuventas_net
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      timeout: 20s
      retries: 10

  backend:
    image: ${DOCKER_HUB_USER}/locuventas-backend:1.0
    container_name: locuventas_backend
    environment:
      SPRING_PROFILES_ACTIVE: ${SPRING_PROFILES_ACTIVE}
      DB_HOST: ${DB_HOST}
      DB_PORT: ${DB_PORT}
      DB_NAME: ${DB_NAME}
      DB_USER: ${DB_USER}
      DB_PASS: ${DB_PASS}
      APP_ADMIN_EMAIL: ${APP_ADMIN_EMAIL}
      APP_ADMIN_PASSWORD: ${APP_ADMIN_PASSWORD}
      APP_ADMIN_NOMBRE: ${APP_ADMIN_NOMBRE}
    ports:
      - "${BACKEND_PORT}:8080"
    depends_on:
      db:
```

```

        condition: service_healthy
    volumes:
        - ./uploads/productos:/app/uploads/productos
        - ./uploads/vendedores:/app/uploads/vendedores
        -
./uploads/productosprecargados:/app/uploads/productosprecargados
    networks:
        - locuventas_net

frontend:
    image: ${DOCKER_HUB_USER}/locuventas-frontend:1.0
    container_name: locuventas_frontend
    ports:
        - "${FRONTEND_PORT}:80"
    depends_on:
        - backend
    networks:
        - locuventas_net

volumes:
    mysql_data:

networks:
    locuventas_net:
        driver: bridge

```

- Crea un archivo: **.env**

Solo tienes que modificar los tres parámetros de #Admin inicial, esta cuenta es importante ya que la app funciona por roles.

```

# =====
# Docker Hub
# =====
DOCKER_HUB_USER=dawcarlosp

# =====
# 🛠 Perfil de Spring
# =====
SPRING_PROFILES_ACTIVE=prod

# =====
# MySQL
# =====
MYSQL_ROOT_PASSWORD=your_root_password

```

```

MYSQL_DATABASE=locuventas
MYSQL_USER=locuventas_user
MYSQL_PASSWORD=secure_user_password
MYSQL_PORT=3306

# =====
# Backend DB Access
# =====

DB_HOST=db
DB_PORT=3306
DB_NAME=locuventas
DB_USER=locuventas_user
DB_PASS=secure_user_password

# =====
# Admin inicial
# =====

APP_ADMIN_EMAIL=admin@empresa.com
APP_ADMIN_PASSWORD=AdminSeguro1234
APP_ADMIN_NOMBRE=Administrador Principal

# =====
# Puertos expuestos
# =====

BACKEND_PORT=8080
FRONTEND_PORT=3000

```

Una vez tengas los dos archivos: **.env** y **docker-compose.yml** en el mismo directorio y asegurarte que los puertos requeridos no estén ocupados por otros servicios. Ejecuta:

- Con Docker Compose V2 (Docker moderno)
docker compose --env-file .env up -d
- O con Docker Compose clásico
docker-compose --env-file .env up -d

Después de esperar unos cuantos minutos puedes comprobar que todo ha marchado bien:

- Frontend: http://localhost:3000
- Revisa el backend (si lo necesitas): http://localhost:8080

Detener y limpiar

- Para los servicios:

docker compose down

- Para eliminar también volúmenes (base de datos incluida):

docker compose down -v

● MANTENIMIENTO Y EVOLUCIÓN

○ Plan de mantenimiento y soporte

Aunque LocuVentas es un proyecto académico, se ha diseñado siguiendo principios que favorecen su mantenimiento y evolución a largo plazo. Se contemplan las siguientes acciones:
Corrección de errores detectados durante las pruebas o el uso real del sistema.

- Supervisión de logs y mensajes del servidor para identificar fallos de forma proactiva.
- Gestión de archivos optimizada, con validación de tipo MIME, control de tamaño, redimensionamiento y eliminación segura de imágenes, lo que garantiza un manejo eficiente de los recursos locales.
- Autenticación sin sesiones mediante tokens JWT, lo que simplifica la seguridad y reduce los riesgos comunes de aplicaciones basadas en sesiones.
- Organización modular del código, tanto en frontend como en backend, para facilitar la localización de errores y el desarrollo de nuevas funcionalidades.

○ Identificación de posibles mejoras y evolución del proyecto

Durante el desarrollo del sistema se han detectado funcionalidades útiles que no han podido implementarse por limitaciones de tiempo o que podrían optimizarse en el futuro. Estas mejoras aumentarían la calidad, escalabilidad y experiencia de uso de LocuVentas:

- Finalizar la gestión de categorías y vendedores en el panel de administración.
- Incorporar filtros de búsqueda por nombre, categoría o país en el catálogo de productos.
- Añadir autenticación mediante cuentas de Google o Microsoft (OAuth 2.0).
- Permitir el registro de clientes al realizar ventas y el envío opcional del ticket por correo electrónico.
- Integrar KPIs en el panel de administrador (ventas diarias, ingresos, pendientes, más vendidos...).
- Habilitar impresión directa de tickets en puntos de venta físicos.
- Mejorar la validación en frontend y experiencia de usuario en formularios.
- Añadir notificaciones internas para eventos críticos (ventas sin cobrar, nuevos usuarios...).
- Implementar atajos de teclado para tareas frecuentes.

- Optimizar la interfaz para pantallas táctiles y dispositivos móviles.
- Desacoplar el almacenamiento de imágenes a servicios externos.
- Preparar la protección CSRF si se cambia a una autenticación basada en cookies.

- **Actualizaciones y mejoras futuras**

Para llevar a cabo estas mejoras y profesionalizar el sistema, se plantean las siguientes líneas estratégicas de evolución:

- Despliegue real del sistema en un entorno productivo con dominio personalizado, base de datos remota y medidas de seguridad adaptadas.
- Pruebas automatizadas de integración y regresión que aseguren la estabilidad del sistema tras futuras modificaciones.
- Dashboard administrativo avanzado, con representación visual de KPIs y datos de negocio.
- Rediseño responsive adaptativo y compatibilidad con dispositivos móviles, tablets y pantallas táctiles.
- Implementación de accesibilidad avanzada, con navegación por teclado y atajos para flujos comunes (ventas, búsqueda, cobros...).
- Integración con servicios externos (como AWS S3 para imágenes o proveedores OAuth para autenticación).
- Sistema de notificaciones internas y por email, conectadas a eventos clave del negocio.
- Internacionalización (i18n) para ofrecer soporte multilingüe en caso de expansión geográfica.

● CONCLUSIONES

- **Evaluación del proyecto**

El desarrollo de LocuVentas ha sido una experiencia enriquecedora tanto a nivel técnico como personal. A lo largo del proceso se logró construir una aplicación funcional, escalable y bien estructurada, que cumple con los requisitos definidos inicialmente. Se ha implementado una arquitectura cliente-servidor moderna con tecnologías actuales como React.js para el frontend y Spring Boot para el backend, además de una base de datos MySQL para el almacenamiento persistente de los datos.

Se destaca la integración de buenas prácticas como la autenticación mediante tokens JWT, validación de archivos subidos, control de roles, modularización de servicios y generación de tickets en formato PDF. El sistema ha demostrado ser robusto, con una base sólida para futuras extensiones.

A pesar de algunas funcionalidades no implementadas por falta de tiempo (como filtros avanzados, informes completos o login social), el proyecto entrega una versión totalmente operativa que permite la gestión de usuarios, productos y ventas con distintas reglas de negocio según rol.

○ **Cumplimiento de objetivos y requisitos**

Los principales objetivos y requisitos definidos al inicio del proyecto han sido satisfechos. Entre los más importantes se destacan:

- Creación y gestión de usuarios con sistema de aprobación por parte del administrador.
- Autenticación segura mediante JWT, con acceso diferenciado según rol (administrador o vendedor).
- Creación, edición y eliminación de productos, con validación de datos e imágenes.
- Registro y seguimiento de ventas con distintos estados de pago (pendiente, parcial, pagado).
- Interfaz de usuario clara y funcional, desarrollada con React.js y orientada a la experiencia de uso.
- Generación de tickets PDF y visualización detallada de cada venta.

Aunque algunos aspectos quedaron pendientes —como filtros de búsqueda, informes más potentes o autenticación OAuth— el sistema ha sido diseñado de forma modular, permitiendo añadir estas mejoras en futuras versiones sin comprometer la estabilidad de la arquitectura.

○ **Lecciones aprendidas y recomendaciones para futuros proyectos**

Una de las principales lecciones aprendidas durante el desarrollo de este proyecto ha sido la importancia de contar con una visión clara desde el inicio, tanto en términos funcionales como organizativos. La falta de una planificación detallada y de una distribución adecuada de tareas provocó una acumulación significativa de trabajo hacia las fases finales del proyecto. Esto derivó en jornadas muy intensas, con pocas horas de descanso, especialmente al acercarse la fecha de entrega, debido a errores imprevistos y tareas críticas que no se habían resuelto a tiempo.

A lo largo del desarrollo también hubo momentos en los que, por falta de motivación puntual o por desconexión, procrastiné y di prioridad a otras facetas de mi vida personal, como hacer deporte, compartir tiempo con amigos o simplemente descansar. Si bien estas decisiones fueron necesarias para mantener el equilibrio emocional y físico, también contribuyeron a reducir el tiempo efectivo de trabajo, lo que terminó impactando en la presión acumulada en la etapa final.

En paralelo, existieron momentos de duda respecto a la arquitectura elegida. Consideré seriamente cambiar la estructura cliente-servidor con React.js y Spring Boot por una arquitectura monolítica con Thymeleaf, con la que tenía más experiencia reciente.

React es una tecnología relativamente nueva para mí, y su integración con el backend supuso una curva de aprendizaje considerable. Aun así, decidí mantener la arquitectura original para respetar los objetivos planteados inicialmente y retarme a nivel técnico.

Durante la recta final del desarrollo, también surgieron complicaciones importantes al intentar desplegar el sistema en Google Cloud utilizando contenedores Docker. Aunque el entorno local funcionaba correctamente gracias a docker-compose, la transición a un entorno en la nube conllevó múltiples dificultades: diferencias en la gestión de redes, persistencia de datos, variables de entorno y configuración de servicios como Cloud SQL. Estas barreras técnicas generaron frustración y consumieron más tiempo del previsto, evidenciando que una solución que funciona en local no siempre es trivial de adaptar al entorno de producción. Aun así, estas dificultades fueron una fuente valiosa de aprendizaje en términos de despliegue, arquitectura y diagnóstico de errores en entornos reales.

El uso de herramientas de inteligencia artificial fue otro aspecto decisivo para avanzar de forma más ágil. Estas herramientas me permitieron resolver dudas, validar soluciones, generar ejemplos de código y superar bloqueos técnicos con mayor rapidez, lo que fue clave para recuperar tiempo perdido y mantener el proyecto encaminado.

En este sentido, quiero destacar especialmente la ayuda de mi tutor, quien no solo me orientó técnicamente, sino que también me facilitó el uso de la herramienta Deepwiki, la cual resultó fundamental para documentar y estructurar conceptos clave del proyecto, así como para explorar conexiones entre distintas ideas de forma más eficiente. Esta plataforma se convirtió en un recurso valioso para profundizar en el contenido académico y enriquecer la base conceptual del trabajo.

Asimismo, agradezco el apoyo de los profesores auxiliares que estuvieron disponibles durante las distintas fases del proyecto. Sus aportaciones, consejos y puntos de vista complementaron la orientación principal del tutor y me ayudaron a resolver dudas específicas, además de motivarme a mantener una mirada crítica y estructurada sobre el desarrollo del sistema.

No obstante, lo que realmente impulsó la finalización del proyecto fue la pasión por la programación y la satisfacción de ver los resultados tomar forma. A pesar del cansancio y los momentos de duda, la motivación personal por aprender, mejorar y construir algo funcional fue el motor principal que me permitió seguir adelante.

Recomendaciones para futuros proyectos:

- Establecer desde el inicio una planificación realista con hitos y fechas clave.
- No subestimar la importancia de la organización, aunque se adopte un enfoque flexible.
- Identificar y limitar los momentos de procrastinación, buscando un equilibrio saludable entre lo personal y lo académico/profesional.
- Apostar por tecnologías conocidas cuando el tiempo es limitado, o planificar una fase previa de aprendizaje si se desea incorporar herramientas nuevas.
- Utilizar herramientas de apoyo como la inteligencia artificial o plataformas como Deepwiki para optimizar el tiempo y resolver problemas con mayor agilidad.

En definitiva, este proyecto no solo fue una experiencia técnica, sino también una lección de gestión del tiempo, toma de decisiones, autoconocimiento y colaboración. El resultado final es fruto de una combinación de esfuerzo personal, uso estratégico de herramientas modernas, orientación docente y una motivación constante por aprender y mejorar como desarrollador.

● BIBLIOGRAFÍA Y REFERENCIAS

○ Fuentes utilizadas en el proyecto

- [Xebrío](#)
- [PMOinformatica.com](#)
- [OpenWebinars](#)
- <https://owasp.org/Top10/es/>
- <https://chatgpt.com/>
- <https://dbdiagram.io/d>

○ Referencias y enlaces de interés

- <https://prismic.io/blog/react-component-libraries>
- <https://es.legacy.reactjs.org/docs/testing.html>