COLEGIO litterator

# StockAgile

Autor: Mayra Barrantes Pi

Tutor: Héctor Ángeles Borrás

Desarrollo de Aplicaciones Multiplataforma

Junio - 2025

COLEGIO litterator

# StockAgile

Autor: Mayra Barrantes Pi

Tutor: Héctor Ángeles Borrás

Desarrollo de Aplicaciones Multiplataforma

Junio - 2025

# Resumen

El presente proyecto consiste en el desarrollo de una aplicación de escritorio orientada a la gestión de inventario en pequeños comercios. Surge como respuesta a la necesidad real de mejorar el control de stock, las entradas y salidas de productos, así como la gestión de fechas de caducidad, especialmente en negocios que aún emplean métodos manuales o herramientas básicas. La solución propuesta se ha diseñado para ser sencilla, funcional y adaptada a entornos con recursos limitados, aplicando los conocimientos adquiridos en el ciclo formativo de Desarrollo de Aplicaciones Multiplataforma y los principios de la metodología ágil Scrum.

# Palabras clave

Inventario, productos, ventas, proveedores, caducidad, gestión, stock, API, React.js, Flask, MongoDB, Python, fronted, backend, Scrum

# Abstract

This project consists of the development of a desktop application for inventory management in small businesses. It arises as a response to the real need to improve stock control, product inflows and outflows, and expiration date tracking, especially in businesses that still rely on manual methods or basic tools. The proposed solution is designed to be simple, functional, and adapted to environments with limited resources. It applies the knowledge acquired during the Multiplatform Application Development program and follows the principles of the agile Scrum methodology.

# Keywords

Inventory, products, sales, suppliers, expiration, stock, management, stock, API, React.js, Flask, MongoDB, Python, frontend, backend, Scrum.

# Lista de Acrónimos

| | |
|---|---|
| **ERS** | Especificación de Requisitos de Software |
| **SRS** | Software Requirements Specification (en inglés) |
| **UI** | User Interface (Interfaz de Usuario) |
| **API** | Application Programming Interface |
| **CRUD** | Create, Read, Update, Delete |
| **DB** | Database (Base de Datos) |
| **ID** | Identificador Único |
| **UX** | User Experience (Experiencia de Usuario) |
| **CSV** | Comma-Separated Values (formato de archivo) |
| **JSON** | JavaScript Object Notation (formato de datos) |
| **HTTP** | Hypertext Transfer Protocol |

# Agradecimientos

Quisiera expresar mi más profundo agradecimiento a todas las personas que han formado parte de este camino.

A mi marido, por creer en mí incondicionalmente y apoyarme en esta etapa de reinvención profesional, tras más de veinte años dedicada a un sector completamente distinto. Su confianza ha sido un pilar fundamental para alcanzar esta meta.

A mi hijo, por su paciencia y por el tiempo que no pude dedicarle durante este proceso. Cada minuto invertido en este proyecto ha tenido también el propósito de construir un futuro mejor para ambos.

A mi familia, por su apoyo constante, su comprensión y por estar siempre presente, incluso en los momentos más exigentes.

A mis tutores, Héctor Ángeles Borrás y Jonhatan Carrero, por su orientación, disponibilidad y compromiso. Sus consejos y acompañamiento han sido clave para el desarrollo de este trabajo.

Al resto del profesorado del ciclo de Desarrollo de Aplicaciones Multiplataforma, por compartir sus conocimientos con pasión y por motivarnos a superar cada reto con determinación.

A mis compañeros del ciclo de Desarrollo de Aplicaciones Multiplataforma, por compartir este camino, por su compañerismo y por el aprendizaje mutuo que hemos construido juntos.

# Índice de Contenido

# Índice de Tablas

# Índice de Figuras

# 1 Introducción

Durante mi formación en el ciclo de Desarrollo de Aplicaciones Multiplataforma, se ha comprendido cómo la tecnología puede facilitar significativamente la gestión de procesos en distintos sectores, especialmente en el ámbito comercial. A partir de esta reflexión y de la experiencia personal en pequeños negocios, se decidió desarrollar una aplicación de escritorio orientada a la gestión de inventario, con el objetivo de ofrecer una herramienta útil, clara y eficiente para comercios que enfrentan dificultades en el control de su mercancía.

Este proyecto nace de una necesidad concreta: numerosos pequeños comercios carecen de sistemas adecuados para gestionar su inventario, lo que conlleva errores frecuentes, pérdidas económicas y desorganización. En la mayoría de los casos, se utilizan métodos manuales o soluciones muy básicas que no permiten un seguimiento preciso del stock, ni alertas sobre productos caducados o en bajo nivel de existencias.

La solución propuesta es una aplicación que permite registrar productos, registrar ventas, controlar entradas y salidas, gestionar fechas de caducidad, generar alertas de stock bajo y organizar proveedores, hacer inventario de stock para detectar diferencias. Todo ello se ha desarrollado aplicando los conocimientos adquiridos durante el ciclo formativo, así como los principios de la metodología ágil Scrum, adaptada a un entorno de desarrollo individual.

En este sentido, se propone el desarrollo de una aplicación de escritorio con diseño responsive, que permita gestionar de forma clara y eficaz el stock e inventario de productos, incluyendo funcionalidades específicas como el control de caducidad y las alertas por bajo stock, especialmente relevantes en productos perecederos.

Como informa la página Realidad Económica [1], una gestión de inventarios eficiente en pequeñas empresas no solo permite reducir costes y evitar pérdidas, sino que también mejora la capacidad de respuesta ante la demanda y fortalece la toma de decisiones estratégicas. Esta fuente destaca la importancia de adoptar herramientas tecnológicas adaptadas a las necesidades reales de las pymes, lo cual refuerza la motivación y el enfoque del presente proyecto.
Como se verá en las siguientes secciones, se detallan los motivos que llevaron a la elección de esta idea, los objetivos perseguidos, la metodología de trabajo aplicada y las tecnologías seleccionadas para su implementación, todo ello con el objetivo de construir una herramienta útil, escalable y adaptada a las necesidades reales de pequeños comercios.

# 2 Justificación del tema seleccionado

La idea de desarrollar esta herramienta nace de mi experiencia previa en el sector del comercio. Durante 18 años trabajé en una cadena de supermercados, donde la gestión del stock, el inventario y el control de caducidades formaban parte de las tareas esenciales del día a día. Al tratarse de un comercio de tamaño medio, muchas de estas tareas se realizaban de forma manual o con herramientas muy básicas, lo que provocaba errores frecuentes, pérdidas de productos y una escasa capacidad de anticiparse a las necesidades reales del negocio.

Asimismo, he podido observar en el entorno familiar y social que numerosos pequeños comercios como tiendas de barrio, fruterías o autoservicios, continúan enfrentándose a dificultades similares, Esta situación se debe, en gran medida, a la carencia de herramientas tecnológicas adaptadas a su realidad. Muchas de las soluciones disponibles en el mercado están orientadas a grandes empresas, lo que las convierte en herramientas excesivamente complejas, costosas o poco intuitivas para los pequeños comerciantes, quienes requieren soluciones simples y directas.
En este contexto, consideré que desarrollar esta aplicación no solo sería un excelente proyecto de fin de ciclo, sino también una herramienta con un uso real y práctico. El objetivo principal es ofrecer una solución accesible, sencilla y funcional, diseñada específicamente para ayudar a quienes más lo necesitan en el día a día de la gestión de su negocio.

## 2.1 Razones para elegir la idea

La elección de desarrollar una aplicación de gestión de stock e inventario para pequeños comercios se fundamenta en una necesidad real: ofrecer soluciones tecnológicas eficaces a negocios que no disponen de los recursos para implantar herramientas complejas o costosas. Este tipo de comercios necesita soluciones sencillas pero funcionales, que les permitan organizar mejor su inventario, evitar pérdidas por productos vencidos y optimizar la reposición de mercancía. La aplicación propuesta busca cubrir este vacío, siendo una alternativa intuitiva, efectiva y accesible.

## 2.2 Puntos fuertes del proyecto

- **Utilidad práctica:** La aplicación está orientada a resolver problemas reales en el entorno comercial, especialmente en pequeños negocios que carecen de herramientas tecnológicas adecuadas..

- **Simplicidad y funcionalidad:** Se ha priorizado la implementación de funcionalidades esenciales, evitando sobrecargar la aplicación con funciones innecesarias, lo que la hace más ágil y fácil de usar.
- **Adaptabilidad:** Con pequeñas modificaciones, puede ajustarse a distintos tipos de comercios (alimentación, droguería, bazar, entre otros.).
- **Escalabilidad:** Está diseñada pensando en su crecimiento, permitiendo incluir nuevas funcionalidades en el futuro, como informes de ventas o estadísticas personalizadas.
- **Tecnologías modernas:** Al utilizar tecnologías como React, Tailwind CSS y MongoDB, se garantiza un desarrollo actual, eficiente y mantenible.
- **Responsive design:** La interfaz ha sido diseñada para adaptarse a distintos dispositivos incluyendo ordenadores, tablets o teléfonos móviles, lo que amplía su accesibilidad y versatilidad.
- **Control de caducidad:** Se han incorporado alertas automáticas para productos perecederos, lo que permitirá minimizar pérdidas y mejorar la planificación de ventas.
- **Control Stock bajo:** Se han incorporado alertas para productos que se encuentren por debajo del umbral definido, lo que permite anticipar la reposición y evitar quiebres de stock.

Como puede observarse, estos elementos no solo refuerzan la funcionalidad del sistema, sino que también lo posicionan como una herramienta eficaz y adaptable a las necesidades reales del pequeño comercio

## 2.3 Problemas detectados y soluciones propuestas

- **Gestión ineficiente de inventario:** Muchos pequeños negocios no disponen de una herramienta específica para llevar un control detallado de los productos. La aplicación automatiza y centraliza la gestión, facilitando el seguimiento y la toma de decisiones.
- **Pérdida de productos caducados:** La ausencia de un sistema de alertas o control de fechas de vencimiento genera pérdidas económicas innecesarias. Esta herramienta incorporará una función de control de caducidades que notificará con antelación qué productos están próximos a la fecha de vencimiento.

- **Dificultad de acceso a soluciones tecnológicas:** Las herramientas existentes en el mercado suelen estar orientadas a grandes empresas, lo que las hace inaccesibles para pequeños comercios debido a su complejidad, coste o requerimientos técnicos. En este sentido, la solución propuesta ha sido diseñada para ser fácilmente implementable, sin necesidad de conocimientos técnicos avanzados ni inversiones significativas.

Como puede observarse, cada uno de estos problemas ha sido abordado mediante funcionalidades específicas que buscan mejorar la eficiencia operativa, reducir pérdidas y facilitar el acceso a la tecnología en entornos con recursos limitados.

# 3 Objetivos

## 3.1 Objetivo general

El objetivo principal de este proyecto consiste en diseñar y desarrollar una aplicación de escritorio que permita gestionar de forma sencilla, eficiente y estructurada el inventario de productos en pequeños comercios. Esta herramienta debe facilitar el control de stock, las entradas y salidas de productos, así como la gestión de fechas de caducidad, especialmente en productos perecederos. La finalidad es mejorar la organización interna del negocio, reducir pérdidas económicas derivadas de una mala gestión del inventario y ofrecer una solución tecnológica accesible para pequeños empresarios y trabajadores autónomos que no disponen de herramientas avanzadas.

## 3.2 Objetivos específicos

Para alcanzar el objetivo general, se han definido los siguientes objetivos específicos:

- **Diseñar una interfaz clara, intuitiva y atractiva :** utilizando tecnologías modernas como React y Tailwind CSS, se busca crear una experiencia de usuario amigable, que permita a personas sin conocimientos técnicos interactuar con la aplicación de forma natural y eficiente.

- **Implementar un sistema de registro y seguimiento de productos:** que permita controlar tanto las entradas como las salidas del inventario, manteniendo actualizado el estado del stock en todo momento.

- **Desarrollar funcionalidades específicas para el control de caducidades**: incorporando alertas automáticas que notifiquen al usuario cuando un producto esté próximo a vencer, lo que facilitará una mejor rotación de mercancía y evitará pérdidas innecesarias.

- **Establecer una conexión eficiente con la base de datos MongoDB:** garantizando el almacenamiento seguro y la gestión eficaz de la información relacionada con productos,proveedores,ventas, fechas de caducidad y movimientos de inventario.
- **Aplicar herramientas de control de versiones mediante GitHub:** con el fin de llevar un seguimiento del avance del desarrollo, mantener un historial de cambios y garantizar la estabilidad del código.
- **Organizar el proyecto siguiendo una adaptación de la metodología ágil Scrum:** dividiendo el desarrollo en sprints que permitan trabajar de forma ordenada, planificando tareas, priorizar funcionalidades y evaluar avances al final de cada ciclo.
- **Documentar exhaustivamente todo el proceso de desarrollo:** Desde la fase inicial de análisis y planificación hasta la implementación final, incluyendo pruebas, validaciones, posibles mejoras y conclusiones finales del proyecto.

# 4 Metodología

Aunque el presente proyecto ha sido desarrollado de forma individual, se ha optado por aplicar una adaptación de la metodología ágil Scrum como marco de trabajo. Esta decisión se fundamenta en las ventajas que ofrece dicho enfoque para estructurar el desarrollo en fases cortas, establecer prioridades claras y mantener una visión global y ordenada del progreso, tal como se expone en la bibliografía consultada [2].

Los conceptos fundamentales de Scrum han sido adaptados a un entorno de trabajo unipersonal, organizando el proyecto en varios sprints, cada uno con objetivos concretos y entregables definidos. Para la planificación y seguimiento de tareas, se ha utilizado un tablero Kanban, lo que ha permitido visualizar el estado de cada actividad, identificar bloqueos y gestionar el flujo de trabajo de manera eficiente.

Al finalizar cada sprint, se ha llevado a cabo una breve retrospectiva personal, en la que se ha evaluado lo que ha funcionado correctamente, los desafíos encontrados y las posibles mejoras a implementar en el siguiente ciclo. A pesar de no contar con un equipo de desarrollo, este enfoque ha contribuido significativamente a mantener la motivación, la organización y el enfoque en los objetivos establecidos.
Como puede observarse, la metodología seleccionada ha permitido estructurar el desarrollo del proyecto de forma flexible y eficiente, favoreciendo la entrega continua de valor y la mejora progresiva del producto.

# 5 Análisis de requistos

En esta sección se definen los requisitos que debe cumplir el sistema para garantizar su correcto funcionamiento y alineación con los objetivos del proyecto. Estos requisitos se dividen en dos categorías principales: requisitos funcionales y requisitos no funcionales.

## 5.1 Descripción general del sistema

El sistema desarrollado corresponde a una aplicación de escritorio destinada a la gestión de inventario en pequeños comercios. Su propósito es facilitar el control de productos, entradas y salidas, fechas de caducidad, devoluciones, ventas y proveedores, todo ello desde una interfaz moderna, clara y accesible.

La aplicación permite registrar, editar, eliminar y consultar productos; gestionar proveedores y sus productos asociados; realizar ventas con generación automática de tickets; registrar devoluciones; y emitir alertas visuales ante situaciones críticas como bajo stock o productos caducados. Además, se ha integrado un lector de código de barras para agilizar la introducción de productos y mejorar la eficiencia operativa.

El sistema ha sido diseñado con una arquitectura modular y escalable, utilizando tecnologías modernas como React, Flask y MongoDB, lo que permite su evolución futura sin comprometer la estabilidad ni la mantenibilidad del código.

## 5.2 Requisitos funcionales

Los requisitos funcionales representan las capacidades específicas que debe ofrecer el sistema para cumplir con los objetivos establecidos. En este contexto, se definen las acciones que el sistema debe ser capaz de realizar desde la perspectiva del usuario final. Entre estas funcionalidades se incluyen el registro de productos, la gestión de ventas, el control de proveedores y la supervisión de fechas de caducidad. Estos requisitos resultan esenciales para asegurar que el sistema cumpla adecuadamente con su propósito y responda a las necesidades operativas del negocio; a continuación, se detallan los requisitos funcionales de la aplicación.

### 5.2.1 Gestión de productos

- El sistema debe permitir añadir nuevos productos al inventario, incluyendo código de barras, nombre, unidades, precio, umbral de stock bajo, si es perecedero, fecha de caducidad y proveedor.
- El sistema debe validar que no se repitan productos con el mismo código de barras.

- El sistema debe permitir editar los datos de un producto existente.
- El sistema debe permitir grabar devoluciones de productos en mal estado y descontarlo del inventario.
- El sistema debe permitir reponer productos, actualizando su cantidad y, opcionalmente, su fecha de caducidad.
- El sistema debe permitir busqueda y filtrado de productos.
- Se integrará un lector de código de barras para agilizar la introducción de productos en el inventario y durante las ventas.

### 5.2.2 Gestión de fechas de caducidad

- El sistema debe permitir añadir fecha de caducidad a los productos perecederos.
- El sistema debe permitir modificar la fecha de caducidad de productos perecederos.
- El sistema debe identificar productos perecederos y mostrar alertas para aquellos que estén próximos a caducar o ya caducados.

### 5.2.3 Gestión de proveedores

- El sistema debe permitir registrar, editar y eliminar proveedores.
- El sistema debe permitir asociar productos con proveedores existentes
- El sistema debe obtener y mostrar la lista de proveedores disponibles
- El sistema debe permitir visualizar los productos de un proveedor específico.
- El sistema debe perimitir generar pedidos a proveedores desde la interfaz.

### 5.2.4 Ventas

- El sistema debe permitir scanear productos para añadirlos a un carrito de compra.
- Se debe calcular el total de la venta y permitir seleccionar el método de pago (efectivo o tarjeta).
- Al finalizar la venta, el sistema debe generar un ticket con el resumen de la compra y actualizar el inventario automáticamente.
- El sistema debe permitir poder devolver un producto ya comprado y actualizar el inventario automáticamente.

### 5.2.5 Interfaz de usuario

- El sistema debe contar con una barra lateral de navegación que permita acceder a las distintas secciones: inicio, productos, devoluciones, alertas, caducados, proveedores, inventario y ventas.
- La interfaz debe ser intuitiva y responsiva, adaptándose a distintos tamaños de pantalla.
- Debe incluir formularios para la gestión de productos, proveedores, devoluciones y ventas.
- Debe mostrar alertas visuales para productos con bajo stock o caducados.

### 5.2.6 Conteo de inventario

- El sistema debe permitir escanear o introducir manualmente el código de barras de un producto para registrar su cantidad contada durante un inventario físico.
- El sistema debe validar que el producto exista y que la cantidad ingresada sea válida (número no negativo).
- El sistema debe permitir comparar el stock registrado con el stock contado y mostrar un informe de diferencias.
- El sistema debe permitir validar el inventario, actualizando automáticamente las cantidades en la base de datos.
- El sistema debe mostrar mensajes de error o confirmación según el resultado de las acciones.
- El sistema debe permitir limpiar el formulario y reiniciar el conteo en cualquier momento.
- La interfaz debe mostrar una tabla con los productos contados y sus diferencias respecto al inventario registrado

## 5.3 Requisitos no funcionales

Por su parte, los requisitos no funcionales establecen las condiciones bajo las cuales el sistema debe operar. No se centran en funciones concretas, sino en atributos de calidad como el rendimiento, la usabilidad, la seguridad, la escalabilidad o la compatibilidad. Estos requisitos garantizan que el sistema no solo funcione correctamente, sino que también lo haga de manera eficiente, segura, accesible y mantenible a lo largo del tiempo.

### 5.3.1 Usabilidad

- La interfaz de usuario debe ser intuitiva, clara y fácil de usar para personas sin conocimientos técnicos.
- El sistema debe proporcionar mensajes de error y confirmación comprensibles para guiar al usuario durante su uso.

### 5.3.2 Rendimiento

- Las operaciones comunes (como añadir, editar,devolver o reponer productos) deben ejecutarse en menos de 2 segundos bajo condiciones normales de uso.
- La carga inicial de la aplicación no debe superar los 5 segundos en conexiones estándar.

### 5.3.3 Escalabilidad

- El sistema debe estar diseñado de forma modular para permitir la incorporación de nuevas funcionalidades (por ejemplo, gestión de usuarios, informes, etc.) sin necesidad de reestructurar el código base.

### 5.3.4 Mantenibilidad

- El código debe estar estructurado y comentado adecuadamente para facilitar su mantenimiento y evolución por parte de otros desarrolladores.
- Se deben seguir buenas prácticas de desarrollo como separación de componentes, reutilización de funciones y uso de librerías estándar.

### 5.3.5 Compatibilidad

- La aplicación debe ser compatible con los navegadores modernos (Chrome, Firefox, Edge).
- Debe adaptarse correctamente a distintos tamaños de pantalla (responsive design), permitiendo su uso en dispositivos móviles y tablets.

### 5.3.6 Seguridad

- El sistema debe validar los datos introducidos por el usuario para evitar errores y posibles vulnerabilidades (como inyecciones de código).
- Las operaciones sensibles (como la modificación de productos,etc) deben estar protegidas mediante autenticación (si se implementa en producción).

### 5.3.7 Disponibilidad

- El sistema debe estar disponible para su uso en cualquier momento, siempre que el servidor backend se encuentre en funcionamiento.

# 6 Diseño del sistema

## 6.1 Arquitectura del sistema

El sistema desarrollado se basa en una arquitectura cliente-servidor, estructurada en tres capas principales, siguiendo los principios de diseño de aplicaciones web modernas. Esta arquitectura ha sido seleccionada por su capacidad para separar responsabilidades, facilitar el mantenimiento y permitir la escalabilidad de cada componente de forma independiente. Las capas que la conforman son:

- **Frontend:** Desarrollado con React y Tailwind CSS, proporciona una interfaz de usuario interactiva, modular y responsiva, adaptada a distintos dispositivos.
- **Backend:** Implementado con Flask, expone una API RESTful que gestiona la lógica de negocio y la comunicación con la base de datos.
- **Base de datos:** MongoDB, una base de datos NoSQL que almacena la información de productos, proveedores y ventas.

En la Figura 1 se muestra un esquema general de esta arquitectura, donde se visualiza la interacción entre las capas. El frontend se comunica con el backend a través de peticiones HTTP utilizando una API RESTful, mientras que el backend realiza operaciones sobre la base de datos para almacenar y recuperar la información necesaria. Esta estructura modular permite una evolución progresiva del sistema sin comprometer su estabilidad.

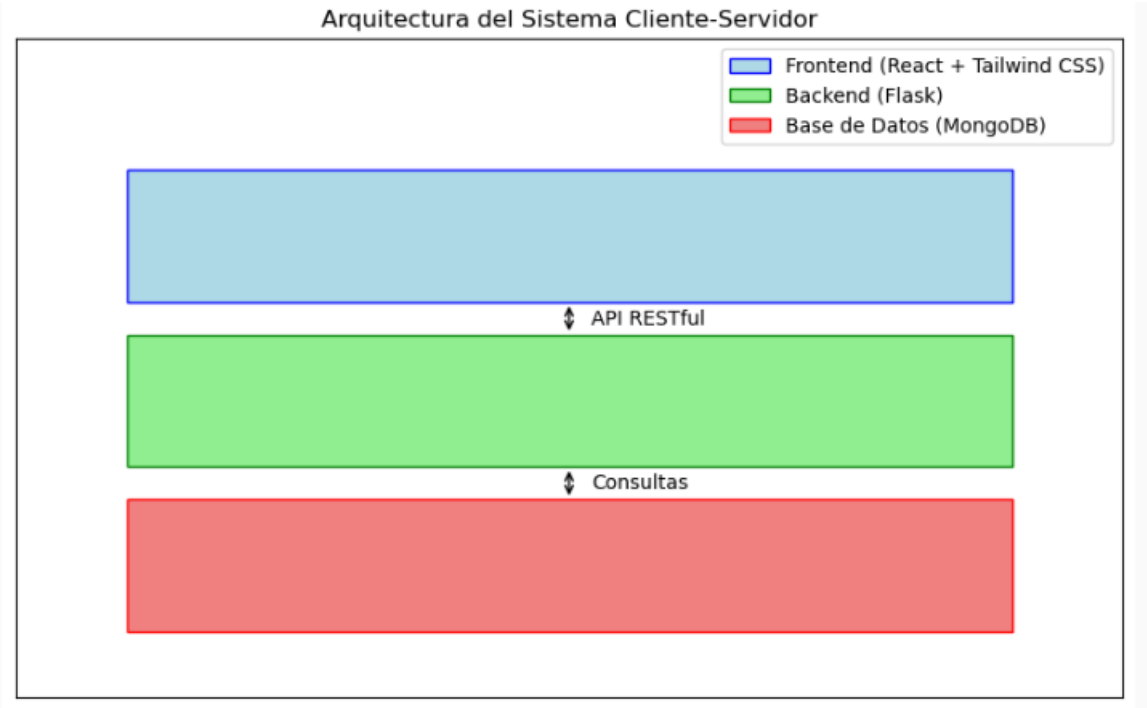


Figure 1: Diagrama de arquitectura del sistema.

### 6.1.1 Resumen Técnico del Sistema

El sistema desarrollado es una aplicación web de gestión de inventario que permite controlar productos, proveedores, ventas, devoluciones y alertas de stock. Se ha diseñado con un enfoque modular, intuitivo y adaptable a distintos dispositivos.

- **Tecnologías clave utilizadas:**
  - **Frontend:** React, JavaScript , Tailwind CSS, React Router
  - **Backend:** Python, Flask, Flask-CORS, PyMongo
  - **Base de datos:** MongoDB
  - **Otras herramientas:** Logging, UUID, smtplib (simulación de pedidos por correo)
  - **Control de versiones:** GitHub
  - **Entorno de desarrollo:**Visual Studio Code y Jupyter Notebook
  - **Gestión de dependencias:** requirements.txt para el backend
- **Funcionalidades principales:**
  - Gestión de productos
  - Control de stock y alertas por bajo inventario
  - Gestión de fechas de caducidad

  - Registro de devoluciones
  - Realización de ventas con ticket
  - Gestión de proveedores y pedidos simulados
  - Conteo de inventario

### 6.1.2 Diseño del Backend

El backend ha sido desarrollado utilizando Flask y expone una API RESTful que permite al frontend interactuar con la base de datos. Se encarga de procesar las solicitudes, validar los datos y realizar operaciones sobre las colecciones de MongoDB.

- **Endpoints principales:**
  - **Productos:**
    * GET /products
    * POST /products
    * PUT /products/id
    * DELETE /products/id
    * POST /fix-products-id
  - **Proveedores:**
    * GET /providers
    * POST /providers
    * PUT /providers/id
    * DELETE /providers/id
    * GET /providers/id/products
- **Características adicionales:**
  - Registro de eventos con logging.
  - Validación de datos y manejo de errores.
  - Comunicación segura con el frontend mediante CORS.

### 6.1.3 Diseño del Frontend

El frontend ha sido construido con React.js, utilizando componentes funcionales para representar cada sección del sistema. La navegación se gestiona con React Router, y el diseño visual se ha implementado con Tailwind CSS, lo que permite una interfaz moderna y responsiva.

- **Componentes principales:**

  - **Home:** Página de inicio con accesos rápidos.
  - **Productos:** Gestión completa de productos (añadir, editar, eliminar, buscar, reponer).
  - **Alertas:** Visualización de productos con stock bajo.
  - **Caducados:** Control de productos próximos a caducar o ya caducados.
  - **Devoluciones:** Registro de devoluciones de productos caducados.
  - **Ventas:** Carrito de compra, selección de método de pago y generación de ticket.
  - **Proveedores:** Gestión de proveedores y simulación de pedidos
  - **Inventario:** Conteo físico de productos y compararlo con el stock registrado en el sistema.

- **Interacción con el backend:**
  - Se utilizan peticiones HTTP (mediante fetch o axios) para consumir la API.
  - Los datos se actualizan dinámicamente en la interfaz tras cada operación.

## 6.2 Diagramas de flujo de la aplicación

En esta sección se presentan los diagramas de flujo correspondientes a los distintos componentes funcionales del sistema, acompañados de sus respectivas especificaciones técnicas. Estos diagramas han sido elaborados con el objetivo de representar de forma visual y estructurada la lógica de funcionamiento de cada módulo de la aplicación. Cada componente se describe mediante una tabla que detalla los siguientes aspectos:

- **Función:** Describe la acción principal que realiza el componente.
- **Prioridad:** Indica la relevancia del componente dentro del sistema.
- **Descripción:** Explica el propósito y comportamiento general del componente.
- **Entradas y salidas:** Define los datos requeridos y los resultados esperados.
- **Origen y destino:** Identifica el punto de inicio de la acción y su impacto en el sistema.
- **Precondiciones y postcondiciones:** Establecen los requisitos previos y los efectos tras la ejecución.
- **Efectos laterales:** Detallan las acciones adicionales que se producen como consecuencia de la operación.

A continuación de cada tabla, se incluye un diagrama de flujo que ilustra gráficamente el proceso descrito, facilitando así la comprensión del comportamiento del sistema y sirviendo como guía para su implementación.

Estos diagramas han sido fundamentales durante la fase de diseño, ya que han permitido validar la coherencia de los procesos, detectar posibles mejoras y asegurar una implementación alineada con los objetivos funcionales del proyecto.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Añadir un nuevo producto al sistema mediante el escaneo de un código de barras y el llenado de un formulario. |
| Prioridad | Alta. Es una funcionalidad esencial para la gestión de inventario. |
| Descripción | Permite registrar productos nuevos en la base de datos si no existen previamente, validando su unicidad por código de barras. |
| Entrada | Código de barras, nombre del producto, unidades, precio, umbral de stock bajo, si es perecedero, fecha de caducidad (si aplica), proveedor. |
| Salida | Producto registrado en la base de datos o mensaje de error si el producto ya existe. |
| Origen | Usuario que interactúa con el formulario en la interfaz de usuario. |
| Destino | Base de datos del sistema de inventario. |
| Precondición | El producto no debe existir previamente en la base de datos (según su código de barras). |
| Postcondición | El producto queda registrado correctamente en la base de datos y se actualiza la interfaz. |
| Efectos laterales | Se muestra un mensaje de confirmación o error. Se limpia el formulario tras el registro exitoso. Se puede cerrar el formulario automáticamente. |

Table 1: Especificación funcional del componente de registro de productos

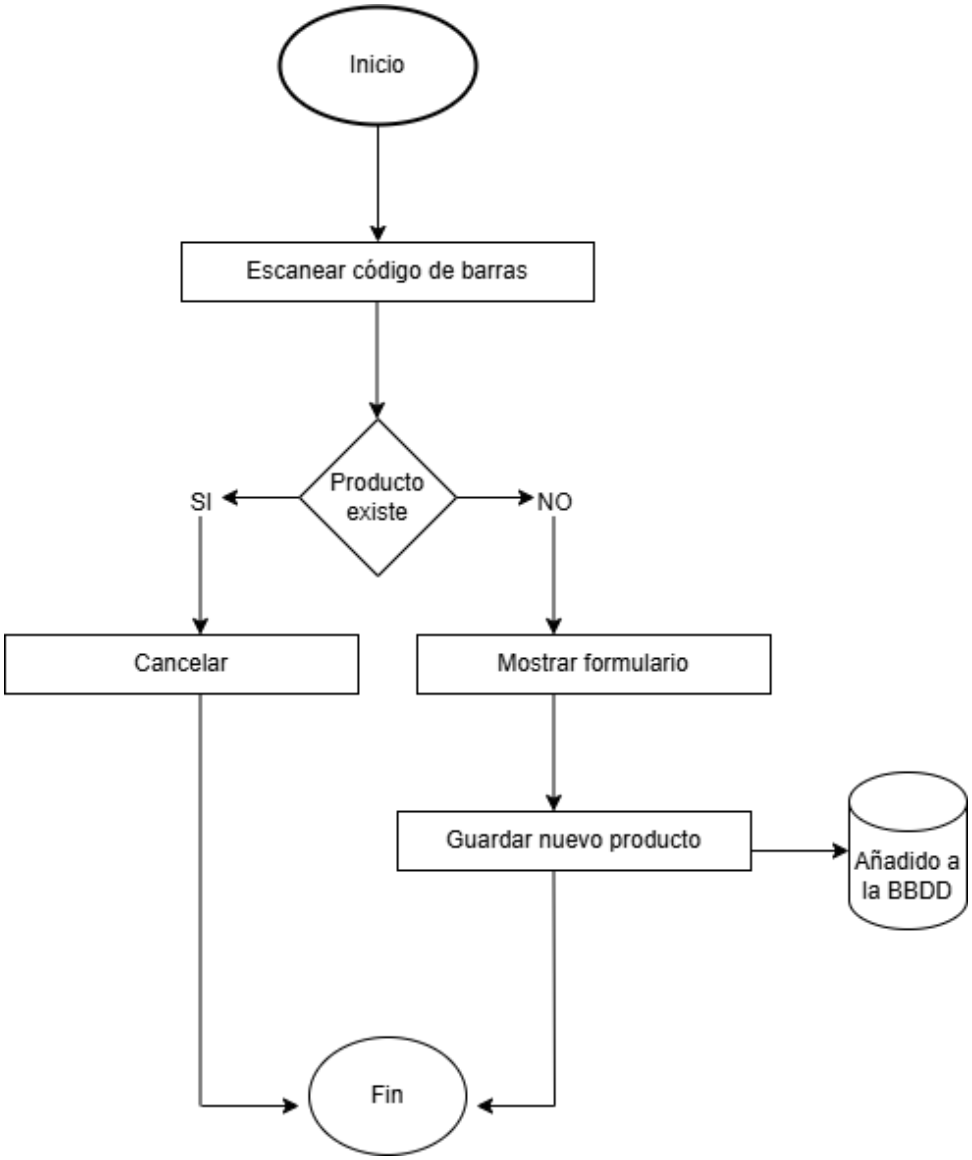


Figure 2: Diagrama de flujo de Registro Productos.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Permitir el conteo físico de productos en inventario, compararlo con el stock registrado y actualizar la base de datos si hay diferencias. |
| Prioridad | Alta. Es fundamental para mantener la precisión del inventario. |
| Descripción | El usuario escanea productos y registra la cantidad contada. El sistema compara con el stock registrado y muestra diferencias. Se puede validar el inventario para actualizar el stock. |
| Entrada | Código de barras del producto, unidades contadas. |
| Salida | Informe de diferencias de inventario, actualización del stock en la base de datos, mensajes de confirmación o error. |
| Origen | Usuario que realiza el conteo físico a través de la interfaz. |
| Destino | Base de datos del sistema de inventario. |
| Precondición | Los productos deben estar previamente registrados en el sistema. |
| Postcondición | El inventario se valida y actualiza si el usuario lo confirma. Se muestra un informe de diferencias. |
| Efectos laterales | Se muestran mensajes de error o confirmación. Se limpia el formulario tras cada acción. Se puede reiniciar el conteo. |

Table 2: Especificación funcional del componente de conteo de inventario

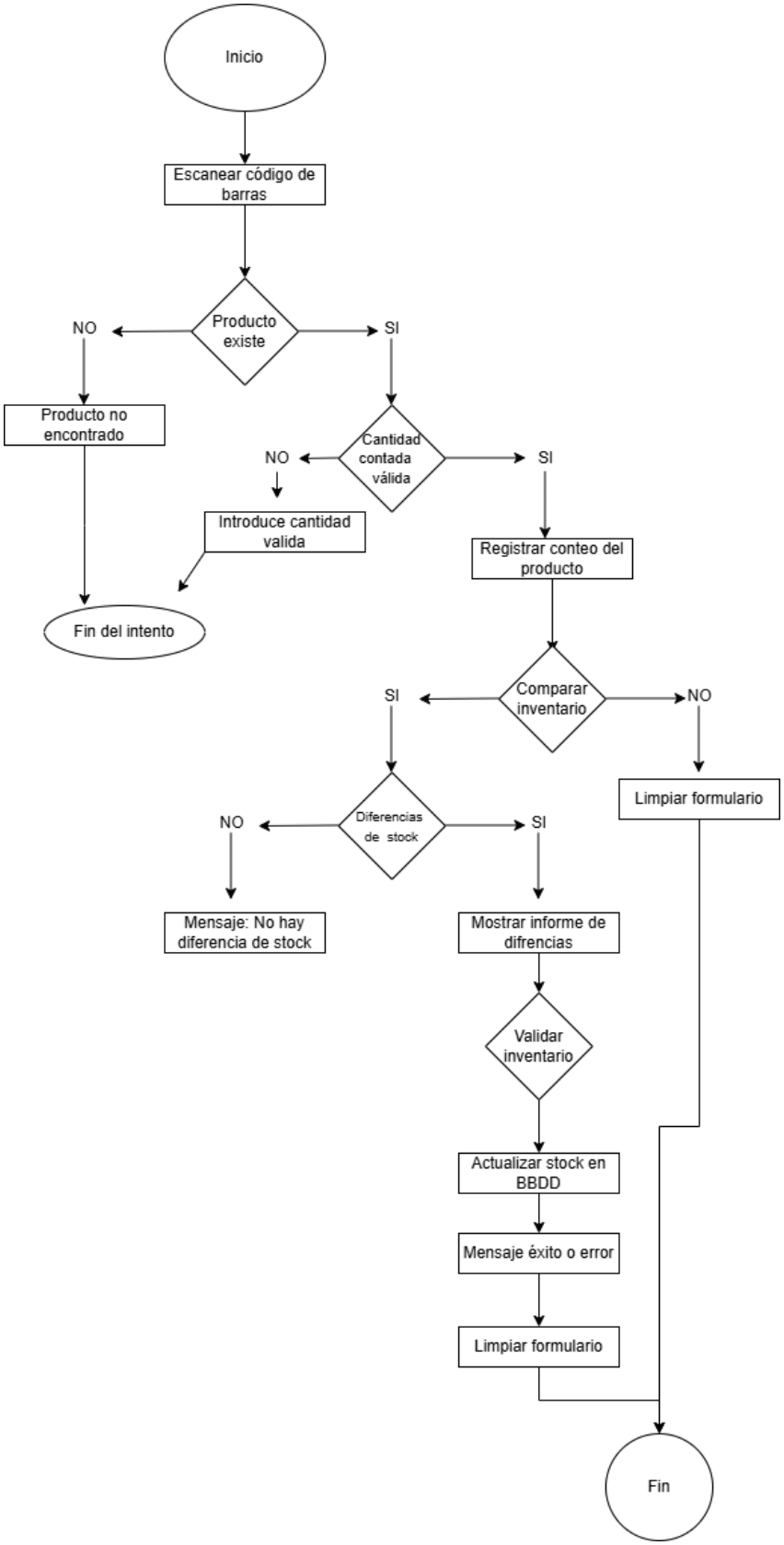


Figure 3: Diagrama de flujo de componente inventario.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Mostrar un modal con los detalles completos de un producto seleccionado, incluyendo alertas visuales si el stock es bajo. |
| Prioridad | Media-Alta. Mejora la experiencia del usuario al proporcionar información detallada de productos. |
| Descripción | Al seleccionar un producto, se despliega un modal con información como nombre, cantidad, precio, proveedor, si es perecedero, fecha de caducidad y umbral de stock bajo. |
| Entrada | Objeto 'producto' con sus propiedades, función 'onClose', y nombre del proveedor ('proveedorNombre'). |
| Salida | Modal visual con los datos del producto y alertas si el stock es bajo. |
| Origen | Usuario que selecciona un producto desde la interfaz. |
| Destino | Interfaz de usuario (modal emergente). |
| Precondición | El producto debe estar previamente cargado y seleccionado. |
| Postcondición | El usuario visualiza los detalles del producto y puede cerrar el modal. |
| Efectos laterales | Se muestra una alerta visual si el stock es bajo. El modal se cierra al pulsar el botón de cierre. |

Table 3: Especificación funcional del componente de detalles del producto

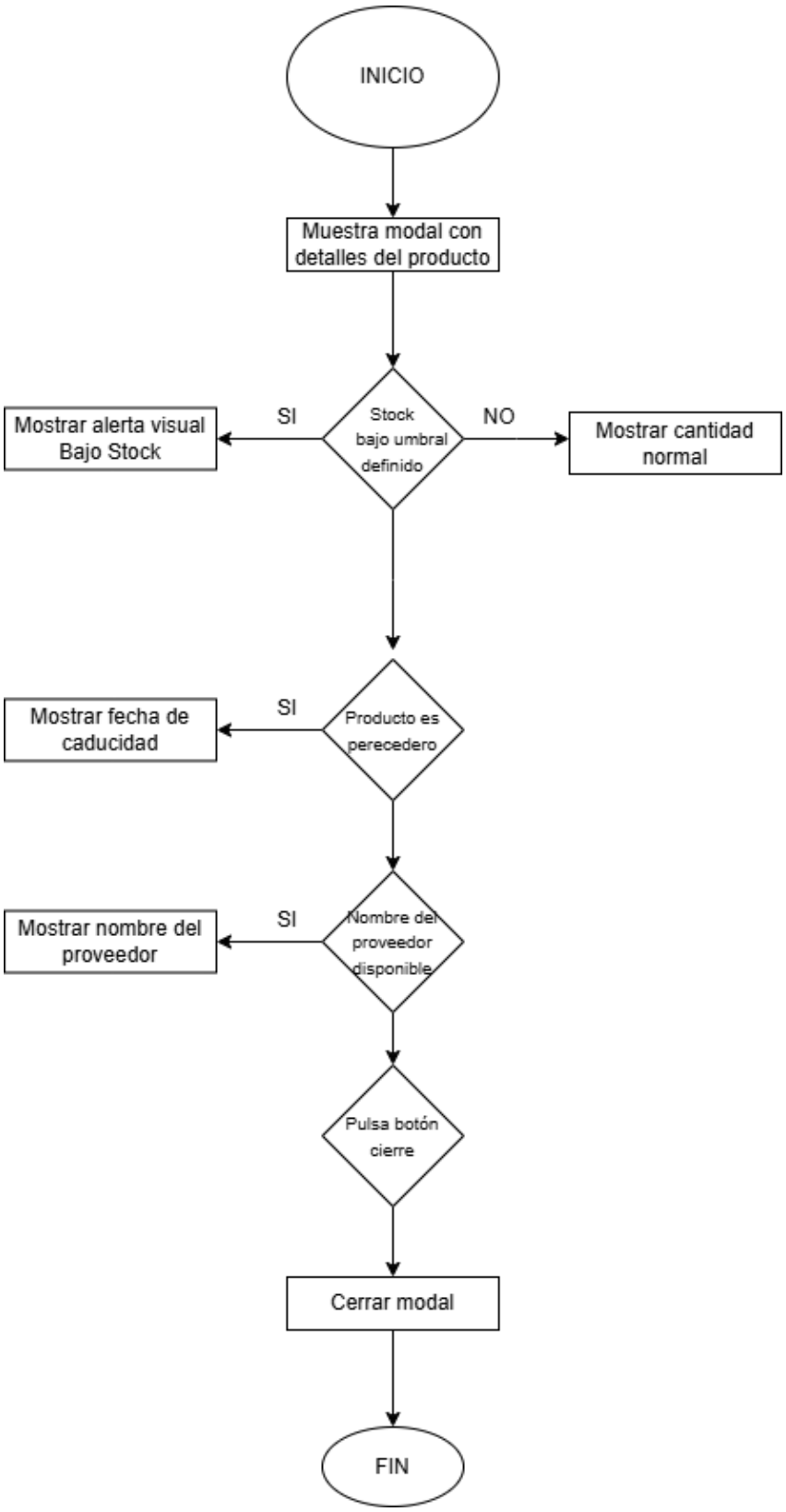


Figure 4: Diagrama de flujo de componente modal ver detalle del producto.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Editar un producto existente en el inventario mediante un formulario. |
| Prioridad | Alta. Es esencial para mantener actualizada la información de los productos. |
| Descripción | Permite al usuario modificar los datos de un producto, incluyendo nombre, unidades, precio, umbral de stock bajo, si es perecedero, fecha de caducidad y proveedor. |
| Entrada | Nombre del producto, unidades, precio, umbral de stock bajo, si es perecedero, fecha de caducidad, proveedor. |
| Salida | Producto actualizado en la base de datos o mensaje de error si ocurre un fallo. |
| Origen | Usuario que interactúa con el formulario de edición de producto. |
| Destino | Base de datos del sistema de inventario. |
| Precondición | El producto debe existir previamente en la base de datos. |
| Postcondición | El producto queda actualizado correctamente en la base de datos. |
| Efectos laterales | Se muestra un mensaje de confirmación o error. El formulario se cierra tras completar la operación. |

Table 4: Especificación funcional del componente de edición de productos

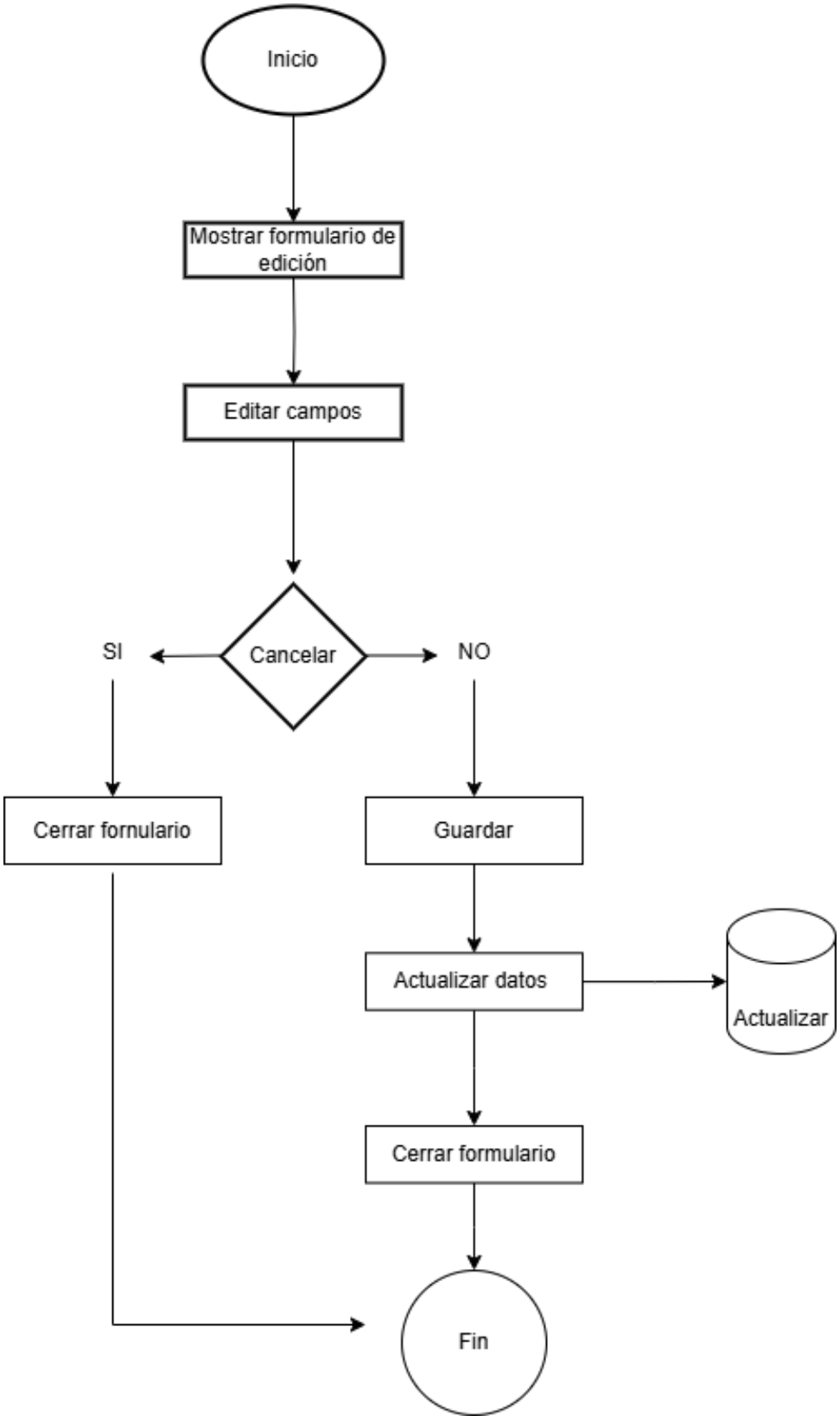


Figure 5: Diagrama de flujo del componente de edición de productos.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Registrar la devolución de productos escaneando su código de barras y actualizando el inventario. |
| Prioridad | Media. Es importante para mantener la precisión del inventario. |
| Descripción | Permite buscar un producto por código de barras, ingresar la cantidad devuelta y actualizar el stock en la base de datos. |
| Entrada | Código de barras del producto, cantidad de unidades devueltas. |
| Salida | Producto actualizado en la base de datos con nuevas unidades. |
| Origen | Usuario que realiza la devolución desde la interfaz. |
| Destino | Base de datos del sistema (actualización del producto). |
| Precondición | El producto debe existir en la base de datos. |
| Postcondición | El stock del producto se actualiza correctamente y se notifica al usuario. |
| Efectos laterales | Se limpia el formulario, se muestra una alerta de éxito o error, y se actualiza la vista del inventario. |

Table 5: Especificación funcional del componente de registro de devoluciones

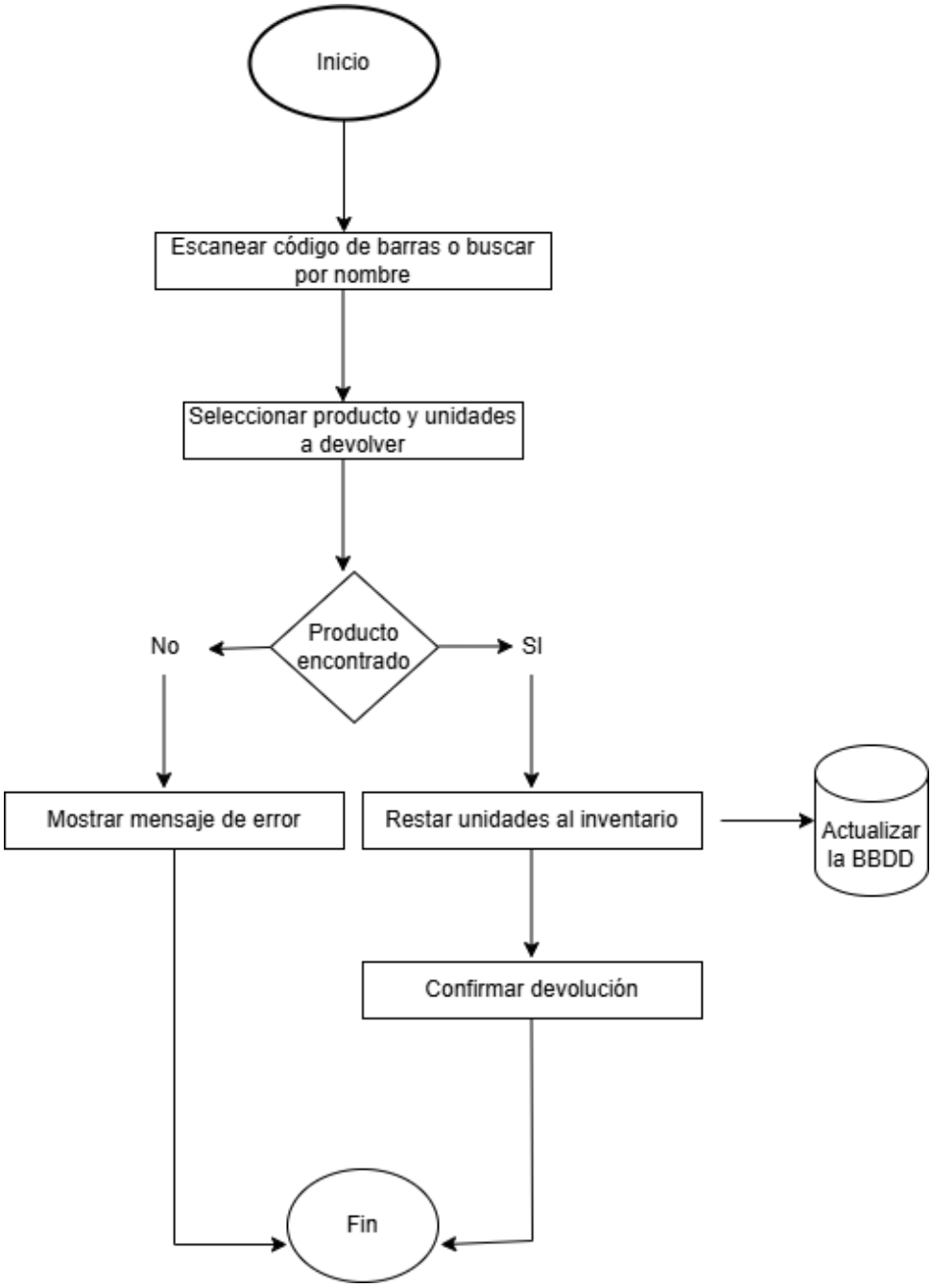


Figure 6: Diagrama de flujo de registro de devoluciones.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Reponer el stock de un producto existente mediante la introducción de su código de barras, cantidad y fecha de caducidad opcional. |
| Prioridad | Alta. Es esencial para mantener actualizado el inventario. |
| Descripción | Permite al usuario buscar un producto por su código de barras, añadir unidades al stock actual y actualizar la fecha de caducidad si se proporciona. |
| Entrada | Código de barras, cantidad a reponer, fecha de caducidad (opcional). |
| Salida | Producto actualizado en la base de datos o mensaje de error si ocurre un fallo. |
| Origen | Usuario que interactúa con el formulario de reposición. |
| Destino | Base de datos del sistema de inventario. |
| Precondición | El producto debe existir previamente en la base de datos. |
| Postcondición | El producto tiene más unidades en stock y puede tener una nueva fecha de caducidad. |
| Efectos laterales | Se muestra un mensaje de confirmación o error. Se puede cerrar el formulario tras la operación. |

Table 6: Especificación funcional del componente de reposición de productos

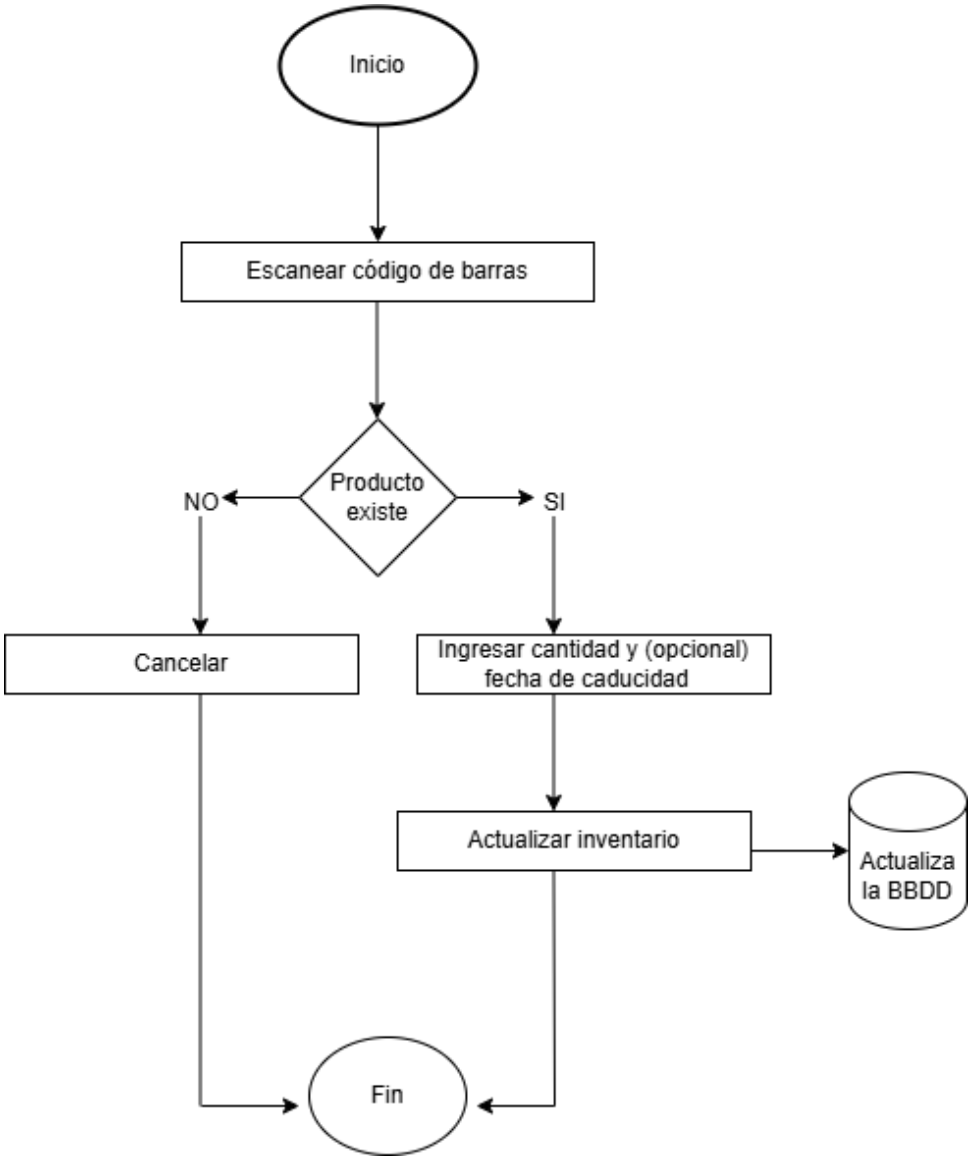


Figure 7: Diagrama de flujo del componente de reposición de productos.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Registrar un nuevo proveedor o modificar los datos de uno existente mediante un formulario. |
| Prioridad | Media-Alta. Es importante para mantener actualizada la información de los proveedores. |
| Descripción | Permite al usuario introducir o editar los datos de un proveedor, incluyendo nombre, dirección, teléfono y correo electrónico. |
| Entrada | Nombre, dirección, teléfono, correo electrónico del proveedor. |
| Salida | Proveedor registrado o actualizado en la base de datos. |
| Origen | Usuario que interactúa con el formulario de registro o edición de proveedor. |
| Destino | Base de datos del sistema de gestión de proveedores. |
| Precondición | Si se trata de una edición, el proveedor debe existir previamente. |
| Postcondición | El proveedor queda registrado o actualizado correctamente en la base de datos. |
| Efectos laterales | Se muestra un mensaje de confirmación. El formulario se cierra tras completar la operación. |

Table 7: Especificación funcional del componente de registro/modificación de proveedores

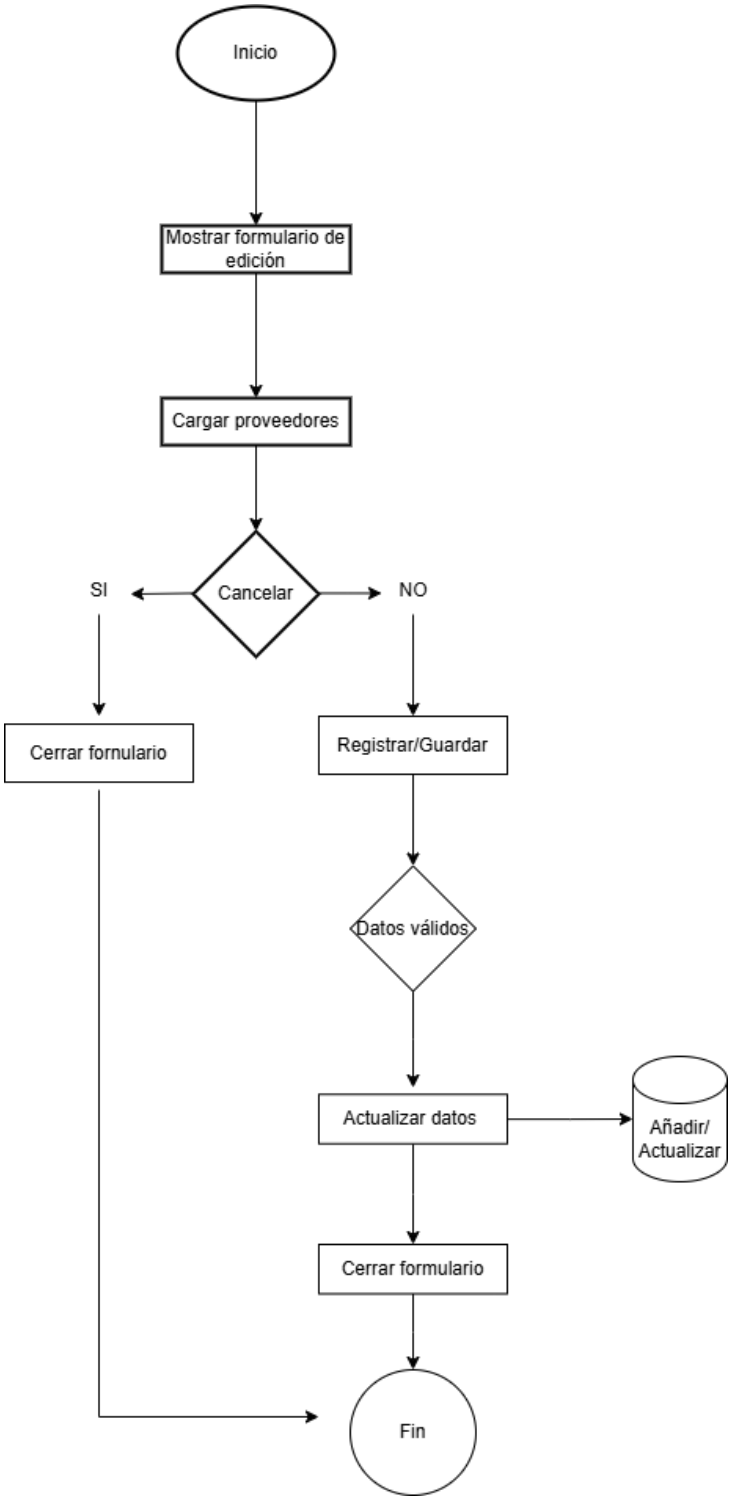


Figure 8: Diagrama de flujo del componente de registro de un proveedor.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Mostrar un menú lateral de navegación con enlaces a diferentes secciones del sistema de gestión de inventario. |
| Prioridad | Media. Mejora la usabilidad y navegación del sistema. |
| Descripción | Permite al usuario acceder rápidamente a distintas páginas como productos, devoluciones, alertas, caducados, proveedores, etc. |
| Entrada | Interacción del usuario con el botón de menú y los enlaces de navegación. |
| Salida | Redirección a la ruta seleccionada o cierre del menú en dispositivos móviles. |
| Origen | Usuario que interactúa con el botón de menú o los enlaces. |
| Destino | Interfaz de usuario y sistema de rutas del frontend. |
| Precondición | El componente debe estar montado y las rutas deben estar definidas en el sistema. |
| Postcondición | El usuario es redirigido a la sección correspondiente o el menú se oculta. |
| Efectos laterales | Transición visual del menú, cambio de vista según la ruta seleccionada. |

Table 8: Especificación funcional del componente Sidebar

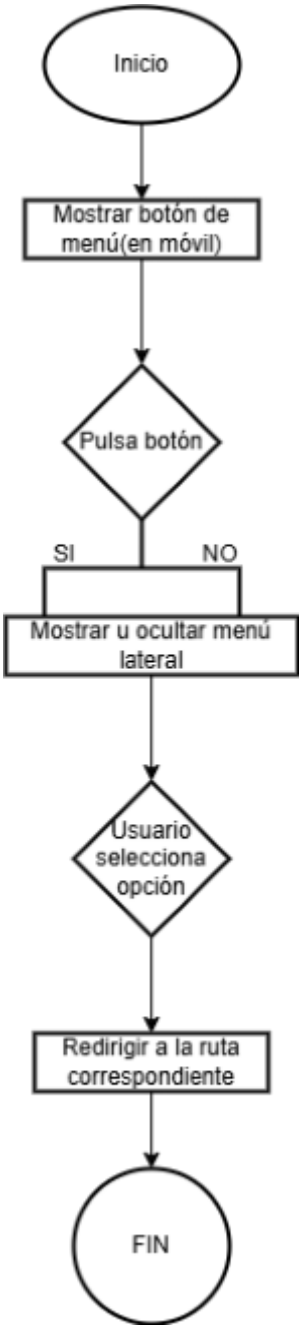


Figure 9: Diagrama de flujo del componente sidebar.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Actuar como pantalla de inicio con navegación a otras secciones del sistema. |
| Prioridad | Alta. Es la vista principal para la navegación del sistema. |
| Descripción | Permite al usuario seleccionar entre opciones de gestión de stock y punto de venta, y navegar a las respectivas secciones. |
| Entrada | Interacción del usuario con los botones de navegación. |
| Salida | Redirección a la ruta seleccionada. |
| Origen | Usuario que interactúa con la pantalla de inicio. |
| Destino | Interfaz de usuario y sistema de rutas del frontend. |
| Precondición | El componente debe estar montado y las rutas deben estar definidas en el sistema. |
| Postcondición | El usuario es redirigido a la sección correspondiente. |
| Efectos laterales | Transición visual de la pantalla, cambio de vista según la ruta seleccionada. |

Table 9: Especificación funcional de la página Home

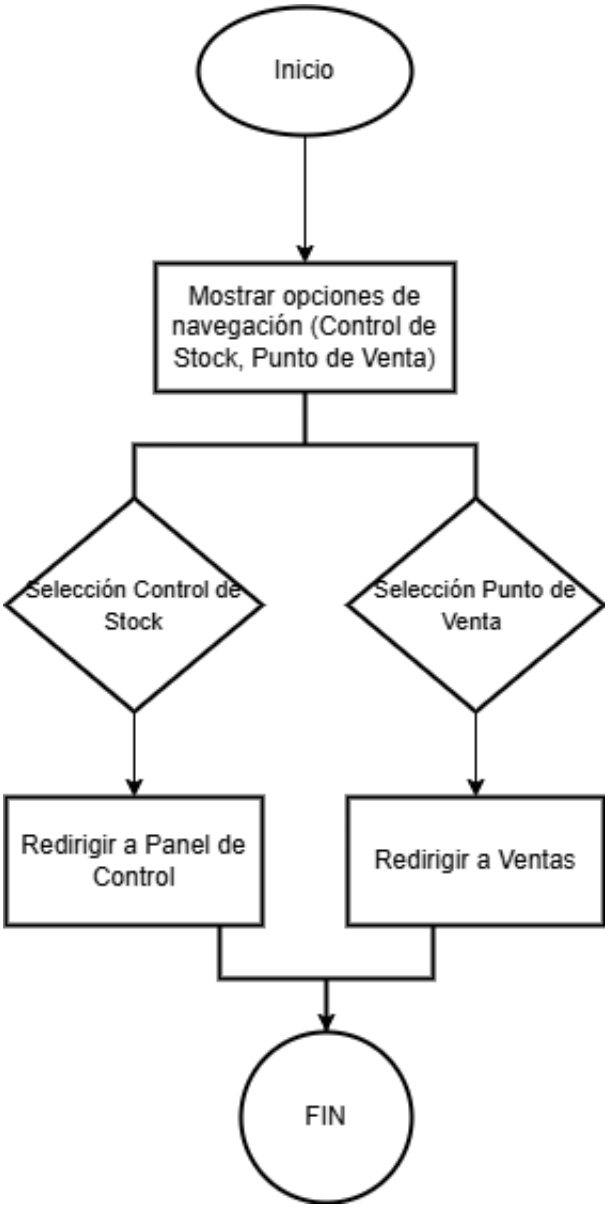


Figure 10: Diagrama de flujo de la página home.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Mostrar un panel de control con métricas clave del inventario y accesos rápidos a funcionalidades del sistema. |
| Prioridad | Alta. Es la vista principal para la supervisión del estado del inventario. |
| Descripción | Recupera los productos desde la API y calcula métricas como total de productos, alertas de stock bajo, productos caducados, próximos a caducar y valor total del inventario. También ofrece accesos rápidos a otras secciones. |
| Entrada | Lista de productos obtenida desde la API. |
| Salida | Panel visual con métricas y botones de navegación. |
| Origen | Usuario que accede a la vista principal del sistema. |
| Destino | Interfaz de usuario y sistema de rutas del frontend. |
| Precondición | El sistema debe tener productos registrados y la API debe estar disponible. |
| Postcondición | Se muestran las métricas actualizadas y el usuario puede navegar a otras secciones. |
| Efectos laterales | Se actualiza el estado del componente con los datos obtenidos. Se puede redirigir a otras rutas mediante los botones de acción rápida. |

Table 10: Especificación funcional de la página Panel de Control

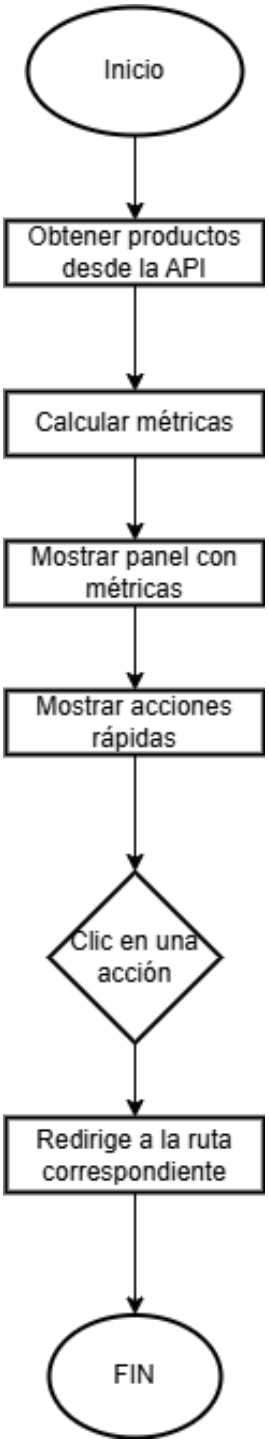


Figure 11: Diagrama de flujo de la página del panel de control.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Gestionar productos en el inventario, incluyendo añadir, reponer, editar, eliminar y buscar productos. |
| Prioridad | Alta. Es esencial para mantener actualizado el inventario. |
| Descripción | Permite al usuario añadir nuevos productos, reponer stock, editar información de productos existentes, eliminar productos y buscar productos en el inventario. |
| Entrada | Datos del producto (nombre, unidades, precio, umbral de stock bajo, si es perecedero, fecha de caducidad, proveedor), término de búsqueda. |
| Salida | Producto añadido, repuesto, editado o eliminado en la base de datos, lista de productos filtrada según el término de búsqueda. |
| Origen | Usuario que interactúa con el formulario de gestión de productos. |
| Destino | Base de datos del sistema de inventario. |
| Precondición | El sistema debe tener productos registrados y la API debe estar disponible. |
| Postcondición | Se actualiza la información del producto en la base de datos y se refleja en la interfaz de usuario. |
| Efectos laterales | Se muestra un mensaje de confirmación o error. Se actualiza la lista de productos en la interfaz. |

Table 11: Especificación funcional de la página de gestión de productos

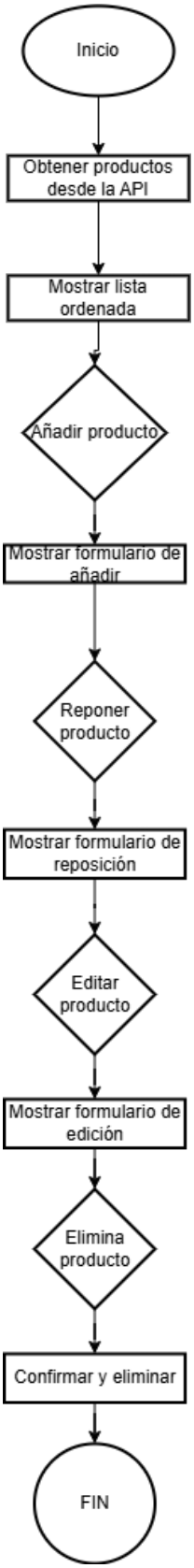


Figure 12: Diagrama de flujo de la página de gestión de productos.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Permitir registrar devoluciones de productos y actualizar el inventario. |
| Prioridad | Alta. Es esencial para mantener la precisión del inventario. |
| Descripción | Permite al usuario registrar una nueva devolución de producto, actualizando el inventario en consecuencia. |
| Entrada | Datos del producto devuelto (nombre, cantidad, motivo de la devolución). |
| Salida | Inventario actualizado con la devolución registrada. |
| Origen | Usuario que interactúa con el formulario de devolución. |
| Destino | Base de datos del sistema de inventario. |
| Precondición | El sistema debe tener productos registrados y la API debe estar disponible. |
| Postcondición | El inventario se actualiza correctamente y se refleja en la interfaz de usuario. |
| Efectos laterales | Se muestra un mensaje de confirmación o error. Se actualiza la lista de productos en la interfaz. |

Table 12: Especificación funcional de la página de Devolucion

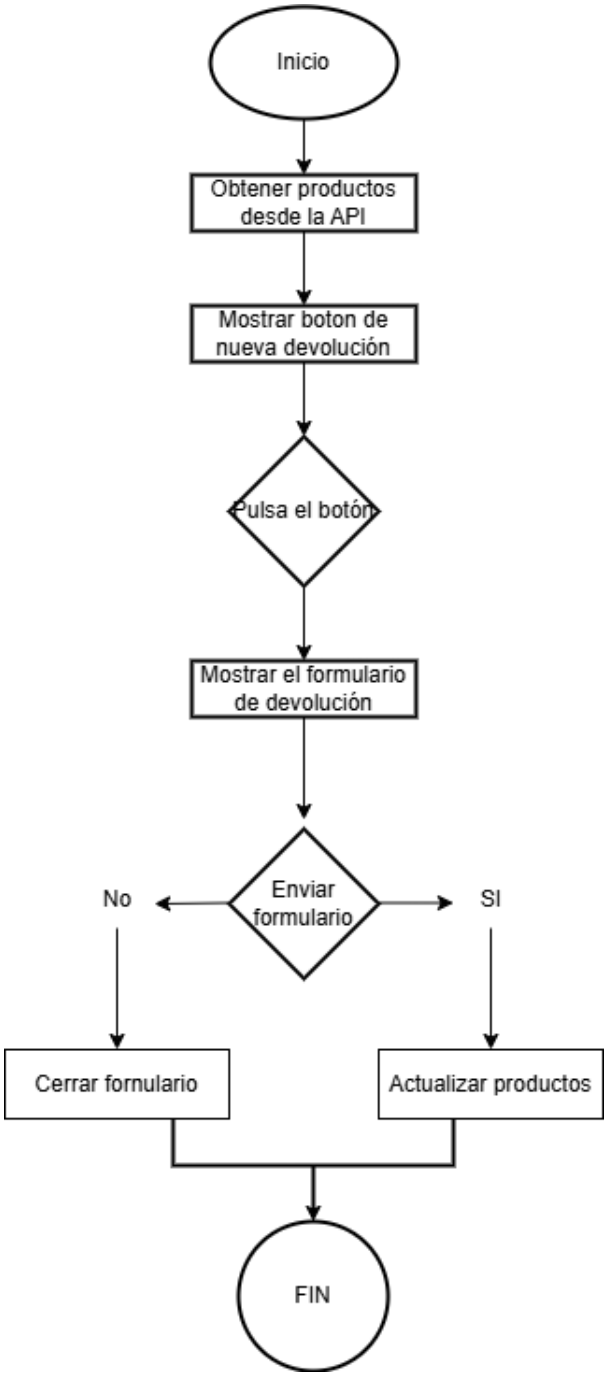


Figure 13: Diagrama de flujo de la página de devolución.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Gestionar el proceso de venta de productos mediante escaneo de código de barras, selección manual, y generación de ticket. |
| Prioridad | Alta. Es una funcionalidad central del sistema de punto de venta. |
| Descripción | Permite buscar productos por código de barras, agregarlos a un carrito, seleccionar método de pago, finalizar la venta, actualizar el stock y generar un ticket. |
| Entrada | Código de barras, selección de productos, cantidad, método de pago (efectivo, tarjeta, transferencia). |
| Salida | Venta registrada, stock actualizado, ticket de compra generado. |
| Origen | Usuario que interactúa con la interfaz del punto de venta. |
| Destino | Base de datos del sistema (registro de ventas y actualización de productos). |
| Precondición | El producto debe existir en la base de datos y tener stock suficiente. |
| Postcondición | La venta queda registrada, el stock actualizado y se muestra el ticket al usuario. |
| Efectos laterales | Se actualiza el inventario, se muestra un ticket emergente, se limpia el carrito tras la venta. |

Table 13: Especificación funcional de la página de ventas

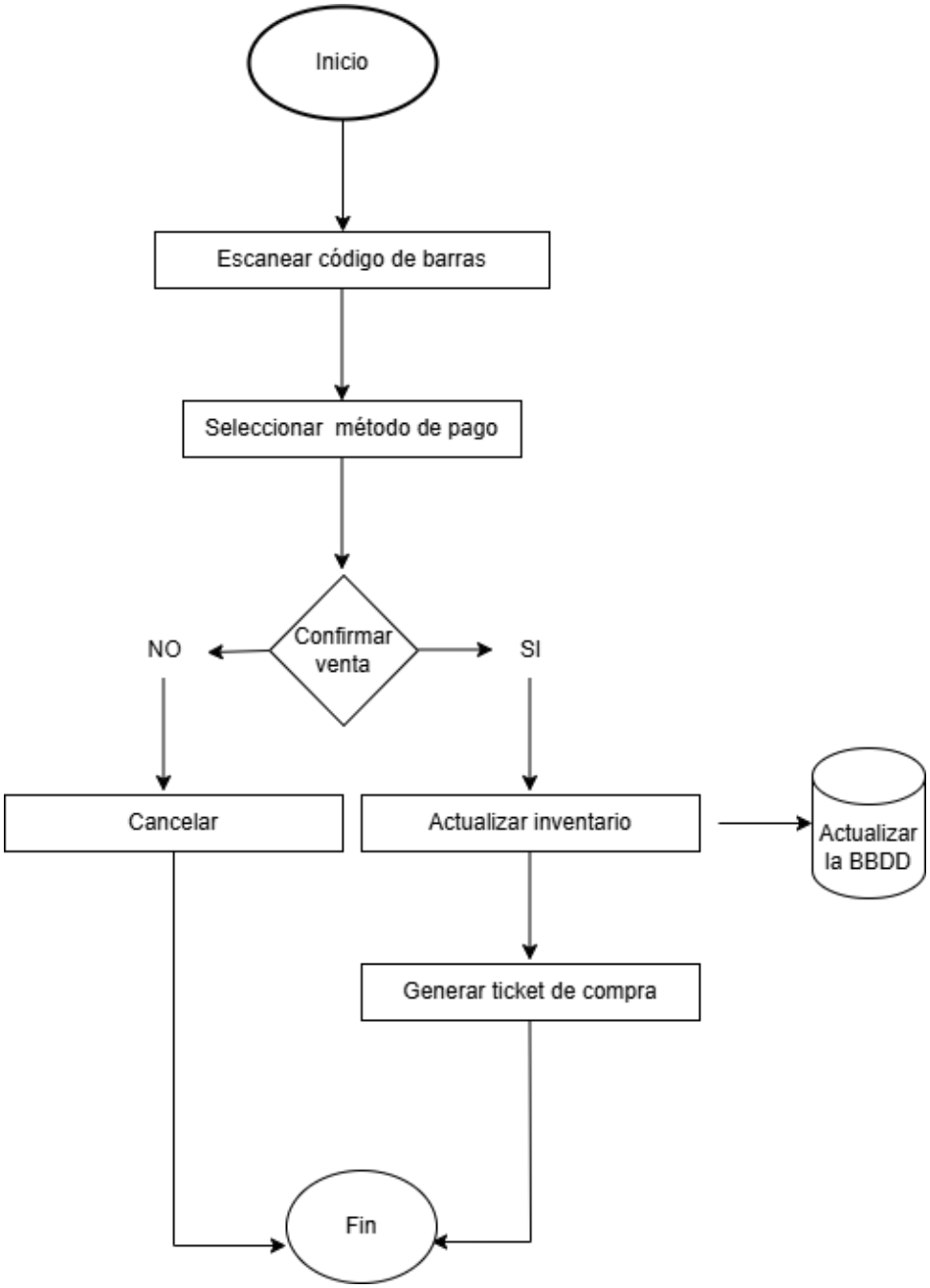


Figure 14: Diagrama de flujo de proceso de la página de ventas.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Mostrar alertas visuales de productos cuyo stock está por debajo del umbral mínimo definido. |
| Prioridad | Media-Alta. Ayuda a mantener el inventario en niveles adecuados. |
| Descripción | Filtra productos con unidades por debajo del umbral de stock bajo y los presenta en tarjetas con detalles y opción de ver más información. |
| Entrada | Lista de productos obtenida desde la base de datos, incluyendo unidades disponibles y umbral de stock bajo. |
| Salida | Lista visual de productos con bajo stock y detalles del producto seleccionado. |
| Origen | Usuario que accede a la sección de alertas en la interfaz. |
| Destino | Interfaz de usuario (pantalla de alertas). |
| Precondición | Debe existir al menos un producto con unidades por debajo del umbral definido. |
| Postcondición | Se muestran los productos en alerta y se puede visualizar información detallada de cada uno. |
| Efectos laterales | Se abre un modal con detalles del producto seleccionado. No se modifica la base de datos. |

Table 14: Especificación funcional de la página de alertas de stock


Figure 15: Diagrama de flujo de la página de alertas de stock.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Gestionar proveedores: registrar, editar, eliminar, ver productos asociados y generar pedidos. |
| Prioridad | Alta. Es fundamental para la gestión de inventario y abastecimiento. |
| Descripción | Permite administrar proveedores, visualizar sus productos, y generar pedidos con cantidades específicas. |
| Entrada | Datos del proveedor (nombre, dirección, teléfono, correo), selección de proveedor, productos asociados, cantidades para pedido. |
| Salida | Proveedor registrado o actualizado, proveedor eliminado, productos mostrados, pedido generado. |
| Origen | Usuario que interactúa con la interfaz de gestión de proveedores. |
| Destino | Base de datos del sistema (proveedores y pedidos). |
| Precondición | El usuario debe tener acceso a la sección de proveedores. |
| Postcondición | Se actualiza la lista de proveedores, se muestra el pedido generado o se actualiza la información del proveedor. |
| Efectos laterales | Se abren modales para formularios y pedidos, se actualiza la interfaz tras cada acción, se muestra confirmación al usuario. |

Table 15: Especificación funcional de la página de gestión de proveedores

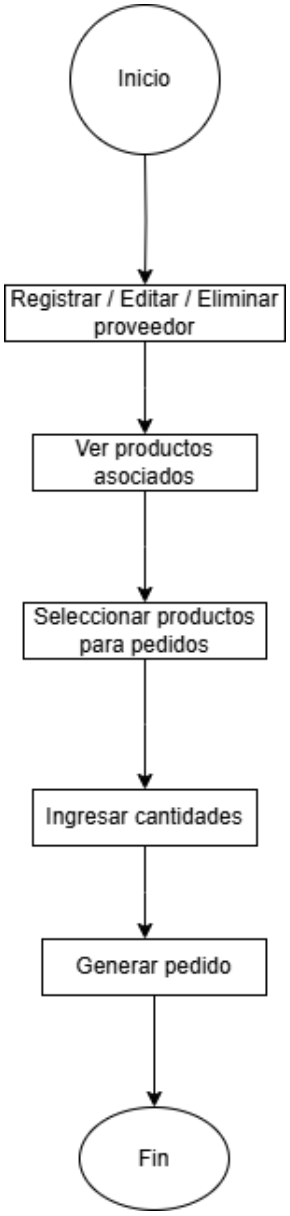


Figure 16: Diagrama de flujo del componente de la página de gestión de proveedores.

| **Parámetro** | **Descripción** |
|---|---|
| Función | Visualizar productos perecederos próximos a caducar o ya caducados, y permitir acciones como modificar la fecha de caducidad o registrar una devolución. |
| Prioridad | Alta. Es fundamental para la gestión de inventario y control de productos perecederos. |
| Descripción | Filtra productos según su fecha de caducidad y permite al usuario tomar acciones correctivas como actualizar la fecha o registrar una devolución. |
| Entrada | Lista de productos con atributos como nombre, unidades, si es perecedero, y fecha de caducidad. |
| Salida | Interfaz con productos clasificados como "Próximos a caducar" o "Caducados", con opciones de modificación o devolución. |
| Origen | Usuario que interactúa con la interfaz de control de caducidad. |
| Destino | Base de datos del sistema de inventario (a través de acciones como actualización de fecha o registro de devolución). |
| Precondición | El sistema debe tener productos registrados con atributos de perecibilidad y fecha de caducidad. |
| Postcondición | Se actualiza la información del producto (fecha o estado de devolución) y se refleja en la interfaz. |
| Efectos laterales | Se muestra un mensaje de confirmación, se actualiza la vista del inventario, y se puede cerrar el formulario de devolución automáticamente. |

Table 16: Especificación funcional de la página de control de productos caducados

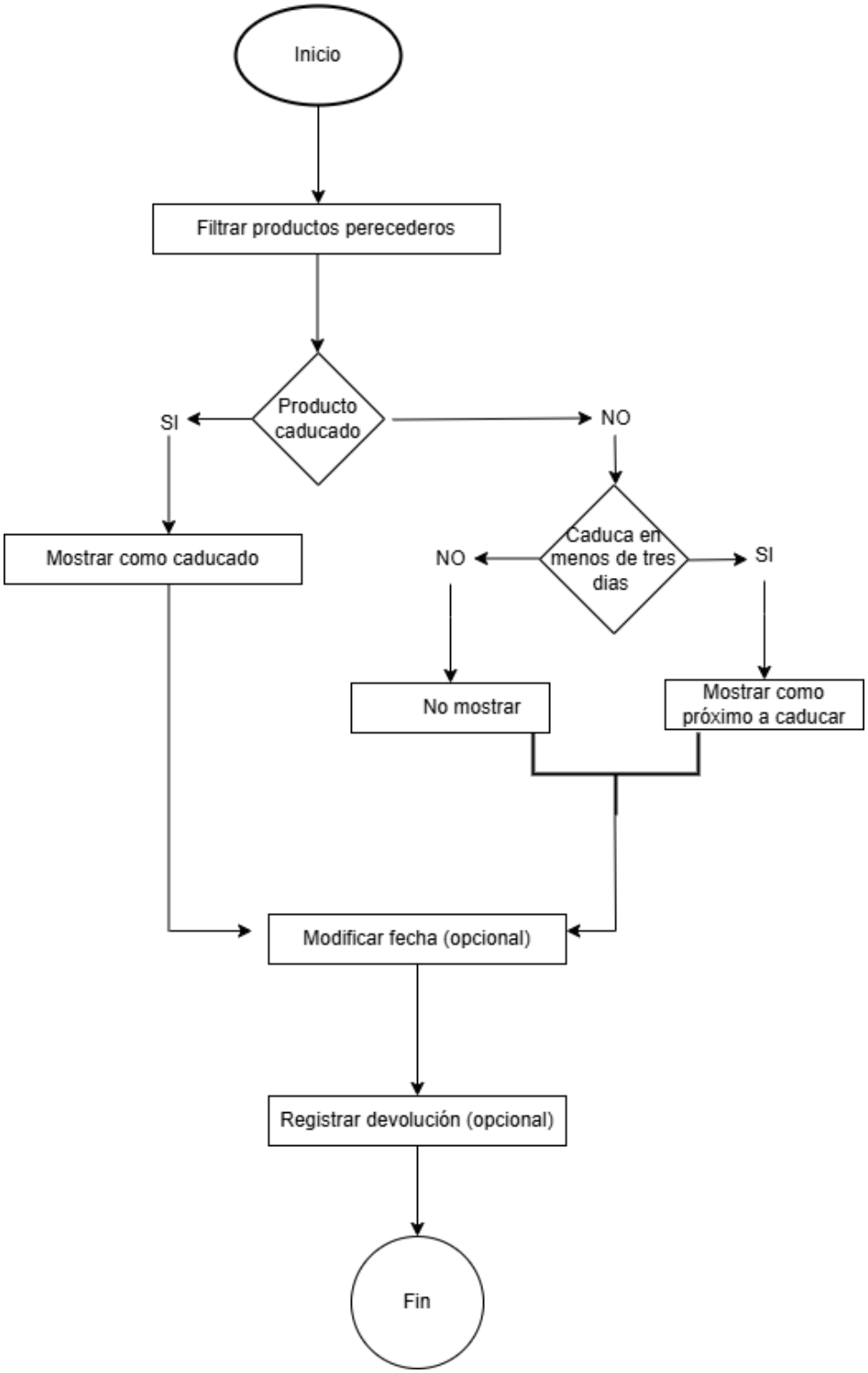


Figure 17: Diagrama de flujo de la página de gestión de productos caducados.

## 6.3 Diagramas de casos de uso

# 7 Diagramas de casos de uso

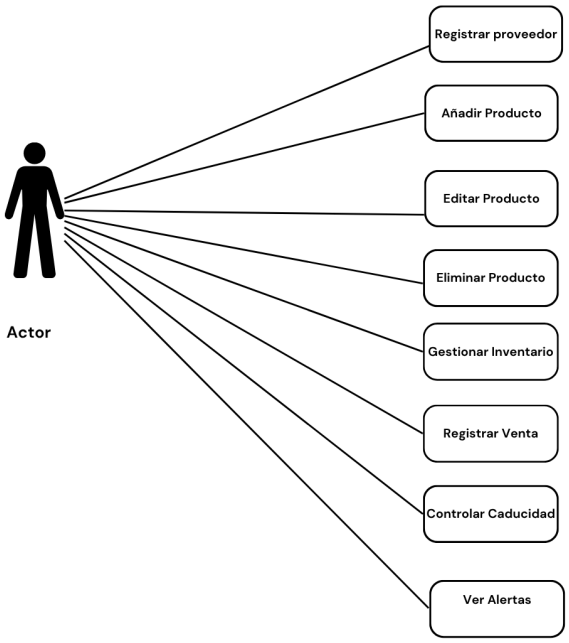


Figure 18: Diagrama de casos de uso general del sistema de gestión de inventario

El diagrama de casos de uso general del sistema de gestión de inventario se presenta en la Figura 18. Este diagrama ilustra las principales interacciones entre el usuario y el sistema, destacando las funcionalidades clave que ofrece la aplicación. A continuación, se describen los casos de uso más relevantes:

- **Registrar productos:** Permite al usuario añadir nuevos productos al inventario, especificando detalles como el nombre, unidades, precio, proveedor, etc.
- **Editar productos:** Facilita la modificación de la información de productos existentes en el inventario.
- **Eliminar productos:** Permite la eliminación de productos del inventario.
- **Reponer productos:** Actualiza la cantidad de productos en el inventario, incluyendo la opción de modificar la fecha de caducidad.
- **Registrar ventas:** Gestiona el proceso de venta de productos, incluyendo el escaneo de códigos de barras, selección de método de pago y generación de tickets.
- **Registrar devoluciones:** Permite registrar devoluciones de productos, actualizando el inventario en consecuencia.

- **Consultar productos caducados:** Muestra productos que están próximos a caducar o ya caducados, generando alertas visuales.
- **Gestionar proveedores:** Facilita el registro, edición y eliminación de proveedores, así como la asociación de productos con proveedores y la generación de pedidos.

# 8 Implementación

## 8.1 Tecnologías Utilizadas

Para el desarrollo de este proyecto se han seleccionado tecnologías modernas y ampliamente utilizadas en el ámbito de desarrollo actual de aplicaciones web, que se adaptan a las necesidades técnicas del proyecto. Cada elección ha sido realizada con el objetivo de construir una herramienta funcional, escalable y con potencial de evolución a futuro, tanto a nivel técnico como operativo. A continuación, se detalla cada una y se explican los motivos de su elección:

### 8.1.1 Frontend: React y Tailwind CSS

Se ha optado por React como biblioteca principal para la construcción de la interfaz de usuario, debido a su capacidad para desarrollar aplicaciones dinámicas y modulares de forma eficiente. Su sistema de componentes facilita la organización del código y la reutilización de elementos, lo cual resulta clave en un proyecto que puede ampliarse en el futuro.
Junto con React, se ha utilizado Tailwind CSS, un framework de estilos que permite diseñar interfaces modernas y responsivas de forma ágil. Gracias a su sistema de clases utilitarias, se han podido aplicar estilos directamente desde el código y personalizar la experiencia visual sin complicaciones.

### 8.1.2 Backend y lógica de negocio: Flask

Para la parte lógica del sistema y cualquier procesamiento que requiera el backend, se ha empleado Flask, un microframework de Python que destaca por su versatilidad y ligereza. Como describe la documentación de Python en [3] Flask permite crear aplicaciones web de manera sencilla y eficiente, con una sintaxis clara y múltiples extensiones útiles para aplicaciones de este tipo. Es una tecnología que se ha ido conociendo durante la formación y que me permite estructurar el proyecto de forma limpia y escalable.

### 8.1.3 Base de datos: MongoDB

Como sistema de gestión de base de datos, se ha utilizado MongoDB, por ser una base de datos NoSQL que ofrece gran flexibilidad, ideal para trabajar con colecciones de datos sin

necesidad de estructuras tan rígidas como en bases de datos relacionales. Esta elección me permite almacenar y consultar fácilmente los productos, movimientos de stock y fechas de caducidad, almacenar proveedores y sus productos y almacenar los tickets de venta, facilitando la evolución del proyecto a medida que crecen los datos.

### 8.1.4 Entorno de desarrollo y apoyo: Visual Studio Code y Jupyter Notebook

El desarrollo del código se ha llevado a cabo principalmente en Visual Studio Code, un editor moderno y extensible, con múltiples extensiones que facilitan la escritura y depuración del código. Adicionalmente, se ha utilizado Jupyter Notebook como herramienta de apoyo para pruebas rápidas, validaciones de lógica o análisis de datos puntuales durante el proceso de desarrollo.

### 8.1.5 Control de versiones: GitHub

Para el control de versiones, se ha utilizado GitHub, lo que ha permitido mantener un historial claro de los avances, realizar copias de seguridad y tener un entorno de trabajo profesional que se asemeja al que se utiliza en entornos reales de desarrollo.

## 8.2 Detalles de implementación

### 8.2.1 Gestión de productos

La gestión de productos es una de las funcionalidades centrales del sistema. Se ha implementado un módulo completo que permite:

- **Alta de productos:** mediante un formulario, el usuario puede registrar nuevos productos. Cada producto incluye campos como:
  - Nombre
  - Unidades disponibles
  - Cantidad (en kilos, si aplica)
  - Precio de venta
  - Umbral de stock bajo
  - Indicador de si es perecedero
  - Fecha de caducidad (si aplica)
  - Proveedor asociado
- **Reposicion de un producto:** El formulario de reposición permite al usuario seleccionar un producto existente y especificar la cantidad adicional que se desea

agregar al inventario. Esta operación actualiza automáticamente el número de unidades disponibles del producto en la base de datos.

- **Edición de productos:** Los productos existentes pueden ser modificados desde la interfaz. Al seleccionar un producto, se abre un formulario modal con los datos precargados.
- **Eliminación de productos:** Se permite eliminar productos del inventario, previa confirmación del usuario.
- **Visualización y búsqueda:** Los productos se muestran en una tabla con paginación y un campo de búsqueda que permite filtrar por nombre. Esto facilita la navegación en inventarios grandes.
- **Validaciones:** Se implementaron validaciones para evitar duplicados y asegurar que los campos numéricos sean positivos.
- **Persistencia:** Los datos se almacenan en MongoDB y se gestionan desde el backend con Flask, utilizando rutas RESTful para operaciones CRUD.

### 8.2.2 Gestión de Proveedores

El sistema permite una gestión integral de proveedores, lo cual es clave para mantener un control eficiente del abastecimiento. Las funcionalidades incluyen:

- **Registro de proveedores:** se pueden añadir nuevos proveedores con datos como nombre, dirección, teléfono y correo electrónico.
- **Edición y eliminación:** los proveedores pueden ser modificados o eliminados desde la interfaz. Se utiliza un formulario modal reutilizable para ambas acciones.
- **Asociación con productos:** cada producto puede estar vinculado a un proveedor mediante su ID. Esto permite filtrar productos por proveedor y generar pedidos específicos.
- **Visualización de productos por proveedor:** al seleccionar un proveedor, se muestra una lista de los productos que le están asociados.
- **Generación de pedidos:** se puede simular un pedido seleccionando productos y cantidades, y generando un resumen que incluye el correo del proveedor y los productos solicitados.

### 8.2.3 Control de Caducidad y Devoluciones

Se ha desarrollado un módulo específico para el control de productos perecederos, que permite:

- **Detección automática:** se identifican productos con fecha de caducidad próxima (por defecto, dentro de 3 días) y productos ya caducados.
- **Alertas visuales:** los productos próximos a caducar se muestran con etiquetas naranjas, y los caducados con etiquetas rojas. Esto facilita la toma de decisiones rápidas.
- **Modificación de fecha de caducidad:** se puede actualizar la fecha de caducidad desde la interfaz mediante un componente específico.
- **Gestión de devoluciones:** los productos caducados o defectuosos pueden ser devueltos en la app para su posterior destrucción. El sistema permite seleccionar el producto, indicar la cantidad a devolver y actualiza automáticamente el inventario.
- **Registro de devoluciones:** cada devolución se registra y se refleja en la cantidad de unidades disponibles, permitiendo mantener un inventario actualizado y confiable.

### 8.2.4 Ventas

El módulo de ventas ha sido diseñado para ser rápido, intuitivo y funcional. Este componente central del sistema permite gestionar el proceso de compra de manera integral, desde la selección de productos hasta la generación del ticket de venta. A continuación, se detallan sus principales características:

- Carrito de compras: los productos seleccionados se agregan a un carrito, donde se puede ver el nombre del producto,la cantidad y el subtotal.
- **Cálculo automático del total:** el sistema calcula el total de la venta en tiempo real.
- **Selección de método de pago:** el usuario puede elegir entre diferentes métodos de pago (efectivo o tarjeta).
- **Finalización de la venta:** al confirmar la venta, se actualiza el inventario restando las unidades vendidas.
- **Generación de ticket:** Una vez finalizada la venta, se muestra un ticket digital con todos los detalles de la transacción: productos adquiridos, cantidades, precios individuales, total de la compra y método de pago utilizado. Este ticket puede ser impreso o simplemente mostrado al cliente como comprobante.

- **Devolución de productos del carrito:** En caso de que un cliente desee devolver un producto después de haber completado la compra, el sistema incluye un botón específico para gestionar devoluciones. Esta funcionalidad permite reembolsar el importe correspondiente y actualizar nuevamente el inventario.
- **Persistencia y consistencia:** todas las operaciones se sincronizan con la base de datos para mantener la integridad del inventario.

### 8.2.5 Conteo de Inventario

El módulo de conteo de inventario permite realizar auditorías físicas del stock y compararlas con los datos registrados en el sistema. Esta funcionalidad es clave para mantener la precisión del inventario y detectar posibles errores o pérdidas. Las principales características son:

- **Ingreso de conteo:** el usuario puede escanear o introducir manualmente el código de barras de un producto, junto con la cantidad contada físicamente.
- **Visualización del conteo:** los productos contados se muestran en una tabla que incluye el nombre del producto, el stock registrado y la cantidad contada.
- **Comparación de inventario:** al presionar el botón correspondiente, el sistema genera un informe de diferencias entre el conteo físico y el stock registrado.
- **Informe de diferencias:** se listan los productos con discrepancias, indicando si hay exceso o faltante. Las diferencias positivas se muestran en verde y las negativas en rojo.
- **Validación del inventario:** permite actualizar el stock registrado con las cantidades contadas, sincronizando así el sistema con el inventario real.
- **Limpieza del conteo:** el usuario puede reiniciar el proceso de conteo en cualquier momento para comenzar una nueva auditoría.
- **Persistencia:** los datos se obtienen y actualizan mediante peticiones HTTP a la API, y se almacenan en MongoDB a través del backend en Flask.

### 8.2.6 Integración del lector de código de barras

Para agilizar la introducción de productos y ventas, se ha integrado un lector de código de barras. Este dispositivo actúa como un teclado que introduce automáticamente el código del producto en el campo de búsqueda o formulario. El sistema ha sido adaptado para:

- Detectar automáticamente el producto al escanear su código.

- Rellenar los campos del formulario de forma automática si el producto ya existe.
- Agregar el producto al carrito de ventas sin necesidad de búsqueda manual.

La finalidad principal de esta integración es reducir el tiempo de registro manual, minimizar errores humanos y facilitar la gestión de inventario en entornos comerciales como tiendas, almacenes o supermercados.

# 9 Despliegue y pruebas

En esta sección se describe el proceso mediante el cual la aplicación fue preparada para su ejecución y validación. El entorno de despliegue ha sido configurado localmente, utilizando herramientas accesibles como Visual Studio Code y servidores de desarrollo integrados en React y Flask. El backend fue ejecutado mediante el comando flask run, mientras que el frontend fue iniciado con npm start, permitiendo así la interacción entre ambos componentes a través de peticiones HTTP habilitadas por CORS.

Las pruebas fueron realizadas de forma manual durante cada sprint, siguiendo un enfoque iterativo. Se llevaron a cabo pruebas funcionales para verificar que las operaciones CRUD sobre productos, proveedores y ventas se ejecutaran correctamente. Asimismo, se realizaron pruebas de interfaz con el objetivo de asegurar una experiencia de usuario fluida y adaptada a distintos dispositivos.

También se efectuaron pruebas de integración, mediante las cuales se validó la correcta comunicación entre el frontend y el backend, así como la persistencia de datos en MongoDB. Por ejemplo, al registrar una venta, se comprobó que el stock se actualizara automáticamente y que el ticket correspondiente fuera generado sin errores.

En las pruebas de rendimiento el sistema respondió de manera satisfactoria en condiciones normales de uso, manteniendo tiempos de respuesta inferiores a los 2 segundos en operaciones comunes, y una carga inicial inferior a los 5 segundos, cumpliendo así con los requisitos no funcionales establecidos.

## 9.1 Análisis de riesgos

Durante el desarrollo del sistema, se han identificado diversos riesgos potenciales que podrían afectar tanto al funcionamiento de la aplicación como a su adopción por parte de los usuarios finales. A continuación, se presenta un análisis de los principales riesgos detectados, junto con las medidas propuestas para su mitigación.

### 9.1.1 Riesgos técnicos

- **Pérdida de datos**: Existe el riesgo de pérdida de información en caso de fallo del sistema o cierre inesperado de la aplicación. Para mitigar este riesgo, se ha implementado un sistema de persistencia basado en MongoDB, que garantiza la integridad de los datos mediante operaciones atómicas y almacenamiento redundante.
- **Errores en la lógica de negocio**: La incorrecta implementación de reglas de negocio podría provocar inconsistencias en el inventario. Para reducir esta posibilidad, se han realizado pruebas funcionales exhaustivas y validaciones en el backend.
- **Dependencia de tecnologías externas**: El uso de bibliotecas y frameworks de terceros (React, Flask, MongoDB) implica una dependencia de su mantenimiento y compatibilidad futura. Este riesgo se ha abordado seleccionando tecnologías ampliamente adoptadas y con comunidades activas.

### 9.1.2 Riesgos de usabilidad

- **Curva de aprendizaje**: Algunos usuarios podrían experimentar dificultades iniciales al interactuar con la interfaz. Para minimizar este riesgo, se ha diseñado una interfaz intuitiva y se ha incluido un manual de usuario detallado.
- **Errores de introducción de datos**: La introducción manual de información puede dar lugar a errores humanos. Para reducir su impacto, se ha integrado un lector de código de barras que automatiza el ingreso de productos.

### 9.1.3 Riesgos operativos

- **Falta de adopción por parte del usuario final**: Existe la posibilidad de que los usuarios no adopten la herramienta por resistencia al cambio o falta de confianza en soluciones digitales. Este riesgo se ha considerado mediante el diseño de una interfaz sencilla, adaptada a usuarios sin experiencia técnica.
- **Limitaciones del entorno de ejecución**: El rendimiento de la aplicación podría verse afectado por las características del equipo donde se ejecute. Para ello, se ha optimizado el rendimiento y se ha verificado su funcionamiento en entornos de recursos limitados.

## 9.2 Plan de mantenimiento y escalabilidad

Con el fin de garantizar la continuidad operativa del sistema y su capacidad de adaptación a futuras necesidades, se ha definido un plan de mantenimiento y escalabilidad que contempla tanto acciones preventivas como estrategias de evolución tecnológica.

### 9.2.1 Mantenimiento

El mantenimiento del sistema se ha estructurado en tres niveles:

- **Mantenimiento correctivo**: Se contempla la revisión y corrección de errores detectados durante el uso de la aplicación. Para ello, se ha documentado el código de forma clara y se han seguido buenas prácticas de desarrollo que facilitan la localización y resolución de incidencias.
- **Mantenimiento adaptativo**: Se prevé la actualización del sistema ante posibles cambios en el entorno tecnológico, como nuevas versiones de bibliotecas o cambios en los navegadores. La modularidad del código y el uso de tecnologías ampliamente soportadas permiten una rápida adaptación.
- **Mantenimiento evolutivo**: Se ha considerado la posibilidad de incorporar nuevas funcionalidades en el futuro, como la autenticación de usuarios, generación de informes o integración con servicios externos. La arquitectura del sistema ha sido diseñada para facilitar estas ampliaciones sin comprometer la estabilidad del núcleo funcional.

### 9.2.2 Escalabilidad

El sistema ha sido concebido con una arquitectura escalable, lo que permite su crecimiento tanto en funcionalidad como en capacidad de procesamiento. Las siguientes estrategias han sido consideradas:

- **Escalabilidad funcional**: Gracias al uso de componentes desacoplados en el frontend y endpoints RESTful en el backend, pueden añadirse nuevas secciones o módulos sin afectar al funcionamiento existente.
- **Escalabilidad de datos**: La base de datos MongoDB permite gestionar grandes volúmenes de información y escalar horizontalmente, lo que garantiza un rendimiento óptimo incluso con un crecimiento significativo del inventario.

## 9.3 Indicadores de rendimiento

En esta sección se presentan los indicadores de rendimiento del sistema, los cuales han sido medidos durante las pruebas de validación. Estos indicadores permiten evaluar la eficiencia y rapidez de las operaciones clave del sistema, asegurando que se cumplan los requisitos no funcionales establecidos.

| Operación | Tiempo promedio (s) | Observación |
|---|---|---|
| Alta de producto | 1.2 | Incluye validación de datos e inserción en la base de datos. |
| Edición de producto | 1.5 | Incluye búsqueda y actualización de datos. |
| Búsqueda de producto | 0.8 | Mediante código de barras o nombre. |
| Venta | 1.8 | Escaneo, cálculo del total y actualización del inventario. |
| Generación de ticket | 0.5 | Generación automática al finalizar la venta. |

Table 17: Indicadores de rendimiento para operaciones clave del sistema

## 9.4 Monetización de la aplicación

Con el objetivo de garantizar la sostenibilidad del proyecto y ofrecer un servicio de calidad a largo plazo, se ha definido un modelo de monetización que contempla tanto un pago inicial como una suscripción mensual. Este modelo está diseñado para ser accesible a pequeños comercios, al mismo tiempo que permite cubrir los costes de hardware, mantenimiento y futuras mejoras.

### 9.4.1 Pago inicial

Se establece un pago único de 413.22€ + IVA (21%), lo que equivale a un total de 500.00€ IVA incluido. Este importe incluye:

- Una tablet con la aplicación preinstalada y configurada.
- Dos lectores de código de barras compatibles con el sistema.
- Asistencia inicial para la puesta en marcha del sistema.

### 9.4.2 Suscripción mensual

Además del pago inicial, se propone una suscripción mensual de 16.53€ + IVA (21%), lo que equivale a un total de 20.00€ IVA incluido. Esta cuota cubre:

- Mantenimiento técnico del sistema.
- Actualizaciones periódicas con mejoras funcionales y de seguridad.
- Soporte básico para resolución de incidencias.

Esta cuota permite asegurar la continuidad del servicio, así como la evolución constante de la herramienta en función de las necesidades de los usuarios.

### 9.4.3 Futuras mejoras y suscripción premium

En fases posteriores del proyecto, se contempla la implementación de una suscripción premium, con un coste ligeramente superior, que incluirá funcionalidades avanzadas como:

- Adjuntar fotografías de productos para una mejor identificación visual.
- Login multiusuario, permitiendo el acceso a la aplicación por parte de varios empleados con distintos roles.
- Generación de informes personalizados sobre ventas, productos más vendidos, stock crítico, entre otros.
- Exportación de datos en formatos como CSV o PDF.
- Notificaciones automáticas por correo electrónico.

Estas funcionalidades estarán orientadas a comercios que deseen un mayor control y análisis de su inventario, ofreciendo así una solución escalable que se adapta al crecimiento del negocio.

# 10 Estado del Arte

La gestión de inventario ha sido reconocida como un elemento esencial para la eficiencia operativa de las organizaciones, especialmente en el ámbito de las pequeñas y medianas empresas (pymes). Tal como se expone en la página [4], se han desarrollado soluciones tecnológicas avanzadas como *SAP*, *Oracle NetSuite* o *Microsoft Dynamics*, y muchas otras [5] las cuales han sido diseñadas para grandes corporaciones. Sin embargo, estas plataformas han sido consideradas poco accesibles para pequeños comercios debido a su complejidad, coste elevado y requerimientos técnicos especializados. En el ámbito de las pequeñas y medianas empresas (pymes), han surgido herramientas más ligeras como *Odoo*, *Zoho Inventory*, *inFlow Inventory* o *Stockpile*, que ofrecen funcionalidades básicas de control de stock, ventas y proveedores. No obstante, muchas de estas soluciones están orientadas a entornos web con conexión permanente a internet, no siempre permiten una personalización profunda según las necesidades específicas del negocio.
En este sentido, se ha desarrollado el presente proyecto como una solución full-stack personalizada, utilizando tecnologías modernas como React, Flask y MongoDB, con el objetivo de:

- Ejecutarse localmente sin depender de servicios en la nube.
- Adaptarse a entornos con recursos limitados.
- Ser fácilmente escalable y mantenible.

- Incorporar funcionalidades específicas para comercios con productos perecederos.

Asimismo, el diseño centrado en la usabilidad hace que esta aplicación sea especialmente adecuada para usuarios sin experiencia técnica, lo que representa una ventaja competitiva frente a otras soluciones del mercado.

# 11 Conclusiones

Como se ha podido observar a lo largo de este documento, el desarrollo de esta aplicación ha permitido dar respuesta a una necesidad concreta en el ámbito de los pequeños comercios: la gestión eficiente del inventario. A través de la implementación de tecnologías modernas, se ha logrado construir una herramienta funcional, intuitiva y escalable.

Entre los principales logros alcanzados, cabe destacar la creación de una interfaz clara y responsiva, la integración de un lector de código de barras para agilizar procesos, y la implementación de alertas para productos caducados o con bajo stock. Estos elementos han sido diseñados con el objetivo de facilitar la toma de decisiones y reducir errores humanos.

Puedo afirmar que este proyecto no solo ha representado un ejercicio técnico, sino también una experiencia formativa integral, en la que se han puesto en práctica habilidades de análisis, diseño, programación y gestión del tiempo. Además, se ha demostrado la viabilidad de aplicar soluciones tecnológicas accesibles a contextos reales con recursos limitados.

# 12 Trabajo futuro

Aunque la aplicación cumple con los objetivos planteados, se han identificado diversas líneas de mejora que podrían abordarse en futuras versiones, algunas líneas de trabajo futuro incluyen:

- **Autenticación de usuarios:** Incorporar un sistema de login para proteger el acceso a la aplicación y permitir distintos roles (administrador, empleado).
- **Informes y estadísticas:** Generar reportes automáticos sobre ventas, productos más vendidos, stock crítico, etc.
- **Exportación de datos:** Permitir la exportación de inventario y ventas en formatos como CSV o PDF.
- **Notificaciones automáticas:** Enviar alertas por correo electrónico sobre productos caducados o stock bajo.

- **Soporte multilenguaje:** Adaptar la interfaz para distintos idiomas, ampliando su alcance a más usuarios.

# 13 Bibliografía

# References


[1] R. Económica, "Estrategias para optimizar la gestión de inventarios en pequeñas empresas," 2023. [Online]. Available: https://www.realidadeconomica.es/estrategias-para-optimizar-la-gestion-de-inventarios-en-pequenas-empresas/41307/.

[2] J. Carrero, "Tema 1: Marco scrum," 2024.

[3] L. J. Caballero G, *Aprende frameworks de desarrollo web en python*, es, https://entrenamiento-frameworks-web-python.readthedocs.io/es/latest/leccion6/index.html, Accessed: 2025-5-5.

[4] T. de Negocios, *Los 10 mejores programas de gestión de stock para retail*, Accessed: 2023-10-05, 2023. [Online]. Available: https://tiempodenegocios.com/los-10-mejores-programas-de-gestion-de-stock-para-retail/.

[5] PortalERP, *Tendencias en gestión de inventarios en las empresas*, Accedido: 2023-10-05, 2023. [Online]. Available: https://portalerp.es/tendencias-en-gestion-de-inventarios-en-las-empresas.

# 14 Anexos

## 14.1 Código Fuente

Debido a la extensión del código, se ha optado por incluir únicamente fragmentos representativos de los módulos principales. El código completo puede consultarse en el siguiente repositorio: `https://github.com/mayrabpi/TFG-gestor-inventario`
Los fragmentos incluidos a continuación corresponden a: Los fragmentos incluidos a continuación corresponden a:

- Backend: archivo `app.py`
- Frontend: componentes principales en React
- Configuración de base de datos y rutas

### 14.1.1 Configuración Inicial

Listing 1: Configuración inicial y conexión a MongoDB

```
from flask import Flask, request, jsonify
from flask_cors import CORS
from pymongo import MongoClient
from bson.objectid import ObjectId
import logging
import smtplib
from email.mime.text import MIMEText

app = Flask(__name__)
CORS(app, resources={r"/*": {"origins": "*"}},
   supports_credentials=True)

# Configuraci n de MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["inventory_db"]
products_collection = db["products"]
providers_collection = db["providers"]
ventas_collection = db["ventas"]

# Configurar el registro
logging.basicConfig(level=logging.DEBUG)
```

### 14.1.2 Rutas para Productos

Listing 2: Rutas para obtener, añadir, actualizar y eliminar productos

```
# Ruta para obtener todos los productos
@app.route("/products", methods=["GET"])
def get_products():
    products = list(products_collection.find({}, {"_id": 0, "id":
        1, "name": 1, "units": 1, "quantity": 1, "price": 1, "
       lowStockThreshold": 1, "perishable": 1, "expirationDate":
       1, "providerId": 1}))
    logging.debug("Productos enviados al frontend: %s", products)
    return jsonify(products)

# Ruta para a adir un producto
@app.route("/products", methods=["POST"])
def add_product():
    data = request.json
    products_collection.insert_one(data)
    return jsonify({"message": "Producto agregado exitosamente"})
       , 201

# Ruta para actualizar un producto
@app.route("/products/<string:product_id>", methods=["PUT"])
def update_product(product_id):
    data = request.json
    logging.debug(f"Recibido product_id: {product_id}")
    logging.debug(f"Datos recibidos para actualizar: {data}")

    result = products_collection.update_one({"id": product_id}, {
       "$set": data})

    if result.matched_count == 0:
        logging.error(f"Producto con id {product_id} no
           encontrado.")
        return jsonify({"error": "Product not found"}), 404

    logging.info(f"Producto con id {product_id} actualizado
       correctamente.")
    return jsonify({"message": "Product updated successfully"})

# Ruta para eliminar un producto
```

```
@app.route("/products/<string:product_id>", methods=["DELETE"])
def delete_product(product_id):
    products_collection.delete_one({"id": product_id})
    return jsonify({"message": "Product deleted successfully"})
```

### 14.1.3 Rutas para Proveedores

Listing 3: Rutas para obtener, añadir, actualizar y eliminar proveedores

```
# Ruta para obtener todos los proveedores
@app.route("/providers", methods=["GET"])
def get_providers():
    providers = list(providers_collection.find({}, {"_id": 0, "id
       ": 1, "name": 1, "address": 1, "phone": 1, "email": 1}))
    logging.debug("Proveedores enviados al frontend: %s",
       providers)
    return jsonify(providers)

# Ruta para a adir un proveedor
@app.route("/providers", methods=["POST"])
def add_provider():
    data = request.json
    data["id"] = str(ObjectId())  # Generar un ID  nico  para el
       proveedor
    providers_collection.insert_one(data)
    logging.info(f"Proveedor agregado: {data}")
    return jsonify({"message": "Proveedor agregado exitosamente"
       }), 201

# Ruta para actualizar un proveedor
@app.route("/providers/<string:provider_id>", methods=["PUT"])
def update_provider(provider_id):
    data = request.json
    logging.debug(f"Recibido provider_id: {provider_id}")
    logging.debug(f"Datos recibidos para actualizar: {data}")

    result = providers_collection.update_one({"id": provider_id},
        {"$set": data})

    if result.matched_count == 0:
        logging.error(f"Proveedor con id {provider_id} no
           encontrado.")
```

```
        return jsonify({"error": "Provider not found"}), 404

    logging.info(f"Proveedor con id {provider_id} actualizado
       correctamente.")
    return jsonify({"message": "Provider updated successfully"})

# Ruta para eliminar un proveedor
@app.route("/providers/<string:provider_id>", methods=["DELETE"])
def delete_provider(provider_id):
    result = providers_collection.delete_one({"id": provider_id})

    if result.deleted_count == 0:
        logging.error(f"Proveedor con id {provider_id} no
           encontrado.")
        return jsonify({"error": "Provider not found"}), 404

    logging.info(f"Proveedor con id {provider_id} eliminado
       correctamente.")
    return jsonify({"message": "Provider deleted successfully"})

# Ruta para obtener productos por proveedor
@app.route("/providers/<string:provider_id>/products", methods=["
   GET"])
def get_products_by_provider(provider_id):
    products = list(products_collection.find({"providerId":
       provider_id}, {"_id": 0, "id": 1, "name": 1, "units": 1, "
       quantity": 1, "price": 1, "lowStockThreshold": 1, "
       perishable": 1, "expirationDate": 1}))
    logging.debug(f"Productos enviados para el proveedor {
       provider_id}: {products}")
    return jsonify(products)
```

### 14.1.4 Rutas para Pedidos

Listing 4: Ruta para enviar pedidos a proveedores

```
@app.route("/api/send-order", methods=["POST"])
def send_order():
    try:
        data = request.json
        provider_email = data.get("providerEmail")
        products = data.get("products")
```

```
        # Validar datos
        if not provider_email or not products:
            logging.error("Datos incompletos: providerEmail o
                products faltan")
            return jsonify({"error": "Faltan datos para procesar
                el pedido"}), 400

        # Crear el contenido del pedido
        product_list = [{"name": p["name"], "quantity": p["
            quantity"]} for p in products]
        pedido = {
            "providerEmail": provider_email,
            "products": product_list,
        }

        logging.info("Pedido generado correctamente")
        return jsonify({"message": "Pedido generado correctamente
            ", "pedido": pedido}), 200

    except Exception as e:
        logging.error(f"Error inesperado: {e}")
        return jsonify({"error": "Error inesperado en el servidor
            "}), 500
```

### 14.1.5 Rutas para Ventas

Listing 5: Ruta para registrar ventas

```
# Ruta para registrar una venta
@app.route("/ventas", methods=["POST"])
def guardar_venta():
    data = request.json
    data["fecha"]= data.get("fecha") or None
    ventas_collection.insert_one(data)
    return jsonify({"message": "Venta registrada exitosamente"}),
        201
```

### 14.1.6 Rutas Adicionales

Listing 6: Ruta para corregir IDs de productos

```
# Ruta para corregir IDs de productos
@app.route("/fix-products-ids", methods=["POST"])
def fix_products_ids():
    products = products_collection.find()
    updates = []

    for product in products:
        if "id" not in product:
            new_id = str(ObjectId())  # Generar un ID  nico
            products_collection.update_one({"_id": product["_id"
               ]}, {"$set": {"id": new_id}})
            updates.append({"_id": str(product["_id"]), "new_id":
                new_id})

    return jsonify({"message": "IDs actualizados para productos
       sin ID", "updates": updates})
```

## 14.2 Código Fuente: App.js

### 14.2.1 Importaciones y Configuración Inicial

Listing 7: Importaciones y Configuración Inicial

```
import React, { useEffect, useState } from "react";
import { BrowserRouter as Router, Routes, Route } from "react-
   router-dom";
import Sidebar from "./componentes/Sidebar";
import Home from "./pages/Home"; // Importa la p gina Home
import Inicio from "./pages/Inicio";
import Productos from "./pages/Productos";
import Devolucion from "./pages/Devolucion";
import Alertas from "./pages/Alertas";
import Caducados from "./pages/Caducados";
import Proveedores from "./pages/Proveedores";
import Ventas from "./pages/Ventas"; // Importa la p gina de
   ventas
import { getProducts } from "./api";
import Inventario from "./pages/Inventario";
```

### 14.2.2 Componente Principal: App.jsx

Listing 8: Componente Principal: App

```
const App = () => {
  const [productos, setProductos] = useState([]);

  useEffect(() => {
    getProducts()
      .then((response) => setProductos(response.data))
      .catch((err) => console.error("Error al obtener los
         productos:", err));
  }, []);

  return (
    <Router>
      <div className="flex">
        <Sidebar />
        <div className="flex-1 p-4">
          <Routes>
            <Route path="/" element={<Home />} /> {/* Ruta para
               la p gina Home */}
            <Route path="/inicio" element={<Inicio />} /> {/*
               Ruta para el control de stock */}
            <Route path="/productos" element={<Productos />} />
            <Route path="/devolucion" element={<Devolucion />} />
            <Route path="/alertas" element={<Alertas />} />
            <Route path="/caducados" element={<Caducados
               productos={productos} />} />
            <Route path="/proveedores" element={<Proveedores />}
               />
            <Route path="/ventas" element={<Ventas />} /> {/*
               Ruta para la p gina de ventas */}
            <Route path="/inventario" element={<Inventario />} />
          </Routes>
        </div>
      </div>
    </Router>
  );
};

export default App;
```

## 14.3 Manual de usuario

- **Añadir Productos**

  1. Navega a la sección 'Productos'.
  2. Haz clic en el botón 'Añadir Producto'.
  3. Completa el formulario con los detalles del producto.
  4. Haz clic en 'Guardar' para añadir el producto al inventario.

- **Realizar Ventas**

  1. Navega a la sección 'Ventas'.
  2. Escanea o selecciona los productos que el cliente desea comprar.
  3. Añade los productos al carrito
  4. Selecciona el método de pago.
  5. Haz clic en 'Finalizar Venta' para completar la transacción.

- **Registrar Devoluciones**

  1. Navega a la sección 'Devoluciones'.
  2. Busca el producto que se va a devolver.
  3. Selecciona el producto y la cantidad a devolver.
  4. Haz clic en 'Registrar Devolución' para completar el proceso.

- **Reponer Productos**

  1. Navega a la sección 'Productos'.
  2. Busca el producto que deseas reponer.
  3. Haz clic en 'Reponer Producto'.
  4. Ingresa la cantidad y la fecha de caducidad (si aplica).
  5. Haz clic en 'Guardar' para actualizar el inventario.

- **Gestionar Caducidad**

  1. Navega a la sección 'Caducidad'.
  2. Filtra los productos por fecha de caducidad
  3. Modifica la fecha de caducidad si es necesario.
  4. Registra devoluciones de productos caducados.

- **Gestionar Proveedores**

1. Navega a la sección 'Proveedores'.
2. Añade, edita o elimina proveedores según sea necesario.
3. Genera pedidos a proveedores.

- **Proceso de Venta**

1. Dirígete a la sección de ventas.
2. Escanea o selecciona los productos que el cliente desea adquirir.
3. Añade los productos al carrito y selecciona el método de pago.
4. Finaliza la venta para registrar la transacción y generar el ticket.

- **Realizar Conteo de Inventario**

1. Navega a la sección de Inventario.
2. Escanea o introduce manualmente el código de barras del producto.
3. Introduce la cantidad contada físicamente.
4. Haz clic en "Agregar" para registrar el conteo.
5. Una vez registrados los conteos, haz clic en "Comparar inventario".
6. El sistema generará un informe de diferencias entre el conteo físico y el stock registrado.
7. Revisa las diferencias mostradas en el informe.
8. Si estás conforme con las diferencias, haz clic en "Validar inventario".
9. El sistema actualizará el stock registrado con las cantidades contadas.
10. Si deseas reiniciar el proceso de conteo, haz clic en "Limpiar conteo".
11. El formulario se limpiará y podrás comenzar un nuevo conteo.