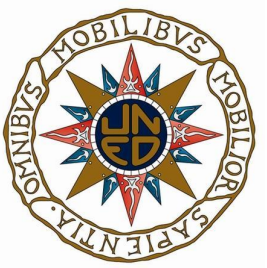


UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

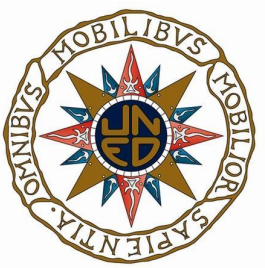
Proyecto de Fin de Grado en Ingeniería Informática

Desarrollo de una aplicación para la gestión de inventario Online

Eduardo García Romera

Dirigido por: Julio Antonio Gonzalo Arroyo

Curso: 2024-2025



Desarrollo de una aplicación para la gestión de inventario Online

Proyecto de Fin de Grado en Ingeniería Informática

de modalidad general

Realizado por: Eduardo García Romera

Dirigido por: Julio Antonio Gonzalo Arroyo

Fecha de lectura y defensa:.....

Desarrollo de una aplicación para la gestión de inventario Online

Resumen del proyecto

Este Trabajo de Fin de Grado ha consistido en desarrollar una aplicación web para gestionar inventarios online, inspirada en el funcionamiento de una empresa tipo Amazon. La idea era construir un sistema completo y funcional que reflejara de forma realista cómo se gestionan productos y almacenes en un entorno empresarial. Además, he querido aplicar todo lo aprendido durante la carrera, intentando seguir buenas prácticas de desarrollo en cada parte del proyecto.

La aplicación permite gestionar productos, ventas y movimientos de inventario entre diferentes almacenes, incluyendo control de stock, alertas automáticas y varios paneles para visualizar datos. También implementa un sistema de acceso por roles (como administrador, encargado, reponedor o marketing), con seguridad basada en JWT y documentación automática gracias a Swagger.

Para el desarrollo he usado Angular 19 con PrimeNG en el frontend, Java 17 con Spring Boot 3 en el backend y PostgreSQL como base de datos. Todo el proyecto lo he gestionado con Git y GitHub, siguiendo un enfoque modular e incremental.

La base de datos está bien normalizada y recoge entidades clave como productos, almacenes, usuarios, roles, transacciones, alertas, etc. Además, he definido algunos tipos personalizados y triggers para mantener la integridad de los datos y controlar aspectos temporales del sistema.

El backend sigue una arquitectura por capas (controladores, servicios, repositorios) y he utilizado DTOs y mapeadores para evitar exponer directamente las entidades. En el frontend, he organizado la aplicación en módulos por funcionalidad (auth, admin, inventario, estadísticas...) y he protegido las rutas según el rol de cada usuario mediante guards.

Para probar la aplicación, he generado datos realistas combinando algunos datasets públicos de Kaggle con lógica de seeding en Java. Esto me ha permitido hacer pruebas funcionales, de rendimiento y de usabilidad, con buenos resultados: la mayoría de las operaciones responden en menos de 300ms y el uso de PrimeNG aporta una experiencia bastante fluida e intuitiva.

Aunque se trata de un TFG y el alcance es limitado, creo que he logrado construir un sistema sólido, seguro y listo para seguir creciendo. Algunas ideas que me gustaría añadir en el futuro son: notificaciones por correo, permisos más detallados, exportación de informes, análisis avanzado de datos y soporte para varios idiomas.

Más allá de ser un trabajo académico, este proyecto me ha servido para consolidar muchos conocimientos y poner en práctica habilidades que hasta ahora solo había visto en teoría. He aprendido bastante sobre arquitectura de software, integración de tecnologías modernas y cómo pensar en el usuario final a la hora de diseñar. Ha sido una experiencia muy completa y, sin duda, un paso importante en mi camino como ingeniero de software.

Lista de palabras clave

Gestión de inventario

Aplicación web

Angular 19

PrimeNG

Spring Boot

Java 17

PostgreSQL

JWT

API REST

Swagger

E-commerce

Roles de usuario

Visualización de datos

Dashboard

Control de stock

Seguridad en aplicaciones web

Arquitectura en capas

DTO

Seeder de datos

Auditoría de acciones

Alertas automáticas

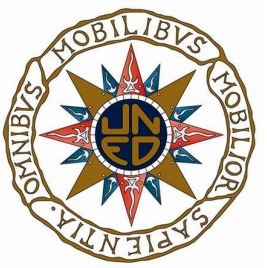
SPA (Single Page Application)

Desarrollo full stack

Ingeniería del software

Pruebas funcionales y de rendimiento

Desarrollo de una aplicación para la gestión de inventario Online



Development of an Application for Online Inventory Management

Bachelor's Thesis in Computer Engineering

General Modality

Authored by: Eduardo García Romera

Supervised by: Julio Antonio Gonzalo Arroyo

Date of Presentation and Defense:.....

Project Summary

This Final Degree Project consisted of developing a web application to manage online inventories, inspired by the operations of an Amazon-style company. The goal was to build a complete and functional system that realistically reflects how products and warehouses are managed in a business environment. Additionally, I aimed to apply everything I've learned during my studies, trying to follow good development practices in every part of the project.

The application allows management of products, sales, and inventory movements between different warehouses, including stock control, automatic alerts, and various dashboards for data visualization. It also implements a role-based access system (such as administrator, manager, stocker, or marketing), with JWT-based security and automatic documentation using Swagger.

For development, I used Angular 19 with PrimeNG on the frontend, Java 17 with Spring Boot 3 on the backend, and PostgreSQL as the database. I managed the entire project with Git and GitHub, following a modular and incremental approach.

The database is well normalized and includes key entities such as products, warehouses, users, roles, transactions, alerts, and more. I also defined some custom types and triggers to maintain data integrity and handle time-sensitive aspects of the system.

The backend follows a layered architecture (controllers, services, repositories), and I used DTOs and mappers to avoid exposing entities directly. On the frontend, I organized the application into modules by functionality (auth, admin, inventory, statistics...) and protected routes based on the user's role using guards.

To test the application, I generated realistic data by combining some public Kaggle datasets with custom Java seeding logic. This allowed me to perform functional, performance, and usability testing, with good results: most operations respond in under 300ms, and the use of PrimeNG provides a smooth and intuitive experience.

Although it is a Final Degree Project with limited scope, I believe I managed to build a solid, secure system ready to continue growing. Some features I would like to add in the future include: email notifications, more granular permissions, report exports, advanced data analysis, and multilingual support.

Beyond being an academic project, this work has helped me consolidate a lot of knowledge and put into practice skills that I had only seen in theory. I learned a lot about software architecture, integrating modern technologies, and thinking about the end user when designing. It has been a very complete experience and, without a doubt, an important step in my journey as a software engineer.

Desarrollo de una aplicación para la gestión de inventario Online

List of keywords

Inventory Management

Web Application

Angular 19

PrimeNG

Spring Boot

Java 17

PostgreSQL

JWT

REST API

Swagger

E-commerce

User Roles

Data Visualization

Dashboard

Stock Control

Web Application Security

Layered Architecture

DTO

Data Seeder

Action Auditing

Automatic Alerts

SPA (Single Page Application)

Fullstack Development

Software Engineering

Functional and Performance Testing

Lista de figuras y tablas

1. Introducción

1.1 Presentación Personal

Me llamo **Eduardo García Romera** y soy estudiante del Grado en **Ingeniería Informática** en la **UNED**. Desde que era niño me ha gustado desmontar cosas para ver cómo funcionaban, sobre todo si tenían botones o pantallas. Con el tiempo, esa manía de "ver qué hay detrás" se convirtió en interés real por los ordenadores y la programación. Gracias al grado y a mi experiencia trabajando en Baleària y Minsait, he aprendido tanto de teoría como de la práctica.

Este TFG no lo veo solo como un trámite para titularme, sino como una oportunidad para juntar todo lo aprendido y construir algo que funcione de verdad, como lo haría en un entorno profesional.

1.2 Contexto del Proyecto

Este Trabajo de Fin de Grado gira en torno al desarrollo de software para la gestión de inventarios, algo que hoy en día es clave en muchas empresas, sobre todo en las que venden online.

El proyecto parte de un caso ficticio pero muy realista: una empresa parecida a Amazon que necesita llevar un buen control de su inventario, registrar ventas, gestionar distintos tipos de usuarios y recibir alertas cuando algo va mal, como por ejemplo si se acaba el stock.

Aunque no está pensado para una empresa real, sí que he intentado que el sistema responda a problemas habituales que cualquier e-commerce podría tener. Al final, la idea es que sirva como base para una solución realista, que se pueda adaptar fácilmente a un entorno profesional.

1.3 Objetivos Generales y Específicos

Objetivo general:

Diseñar y desarrollar una aplicación web que permita gestionar inventarios de forma eficiente en un entorno empresarial, con control sobre las ventas, el stock y los distintos perfiles de usuario. Además, se busca incorporar sistemas de alertas y visualizaciones que faciliten la detección de incidencias y el análisis de la evolución del negocio.

Objetivos específicos:

1. Definir una estructura de base de datos relacional que garantice la integridad y consistencia de la información.
2. Implementar un sistema robusto de gestión de usuarios con diferentes roles y niveles de acceso.
3. Crear una interfaz de usuario intuitiva, clara y adaptable a diferentes dispositivos.
4. Incorporar un sistema de autenticación y autorización que proteja el acceso a los recursos de la aplicación.
5. Facilitar el análisis de datos mediante paneles que muestren estadísticas, alertas y comparativas.
6. Asegurar que el sistema sea escalable, mantenible y capaz de evolucionar con nuevas funcionalidades.
7. Validar el comportamiento de la aplicación tanto en condiciones normales como en escenarios límite o poco comunes.
8. Documentar adecuadamente todas las funcionalidades y flujos del sistema para facilitar su comprensión y mantenimiento.

1.4 Justificación del Proyecto

He elegido este proyecto porque representa una oportunidad excelente para integrar muchos de los conocimientos adquiridos a lo largo del grado, como el diseño de bases de datos, el desarrollo web, la arquitectura de software, la seguridad o el diseño de interfaces. Más allá de aplicar conceptos teóricos, mi intención desde el inicio fue desarrollar una solución con sentido práctico, alejada de ejemplos simplificados o puramente académicos.

En lugar de limitarme a construir un sistema CRUD básico, he querido plantear un proyecto que afronte retos reales, como la gestión de roles y permisos, la escalabilidad del software, la organización modular del código o la protección de los datos. Crear una solución desde cero obliga a reflexionar sobre cada una de estas cuestiones y a buscar respuestas concretas, como se haría en un entorno profesional.

Además, este tipo de aplicaciones son muy habituales en el mundo laboral, especialmente en el sector del comercio electrónico, lo que me ha permitido trabajar sobre un caso realista que guarda relación directa con mi experiencia profesional. Desde el primer momento tuve claro que esta era la mejor opción de TFG genérico para mí, ya que me brinda la posibilidad de aplicar conocimientos de prácticamente todas las asignaturas fundamentales del grado, en un contexto lo suficientemente exigente como para suponer un verdadero reto.

2. Alcance del Proyecto

2.1 Funcionalidades Esenciales

El sistema que voy a desarrollar está pensado para cubrir las funciones básicas que necesitaría una empresa que vende por internet, cuenta con un almacenamiento distribuído y quiere controlar su inventario de forma ordenada. A continuación, resumo las funciones más importantes que he incluido:

- **Gestión de productos**: se pueden crear, editar y borrar productos. Cada uno debe tener nombre, precio, SKU, descripción, categoría y un límite mínimo de stock para lanzar alertas.

- **Inventario por almacenes**: un mismo producto puede estar en varios almacenes, y el sistema ha de llevar la cuenta de cuánto hay disponible en cada uno. Si el stock cae por debajo de cierto límite establecido, se genera una alerta automática.

- **Registro de ventas y movimientos**: todo lo que se mueve en el inventario queda registrado: entradas (add), salidas (remove) y ventas (sale). Así se puede ver el historial completo de lo que ha pasado con cada producto.

- **Usuarios y roles**: dispondremos de 4 tipos de usuarios, cada uno con permisos distintos según su función en la empresa (administrador, encargado, marketing y reponedor).

- **Alertas automáticas**: el sistema ha de avisar cuando cierto producto cae por debajo de su mínimo establecido tras una salida o una venta, y automáticamente eliminar la alerta al reponer.

- **Dashboard con visualizaciones**: crearemos una sección de estadísticas donde se puedan ver gráficos y tablas sobre ventas, almacenes y productos.

- **Seguridad**: todo el sistema estará protegido por tokens JWT. Cada usuario solo podrá acceder a las partes que le corresponden según su rol.

2.2 Casos de Uso del Sistema

La aplicación contempla **cuatro tipos de usuarios**, cada uno con permisos y funcionalidades adaptadas a su rol dentro del flujo de trabajo de una empresa de tipo e-commerce. Esta segmentación permite una experiencia personalizada, mejora la seguridad y evita errores operativos.

2.2.1 Personal de Marketing

Los usuarios de marketing tienen **acceso exclusivo a la visualización de datos y estadísticas**. Su objetivo principal es analizar tendencias, identificar productos con bajo rendimiento y apoyar la toma de decisiones estratégicas.

Funciones disponibles:

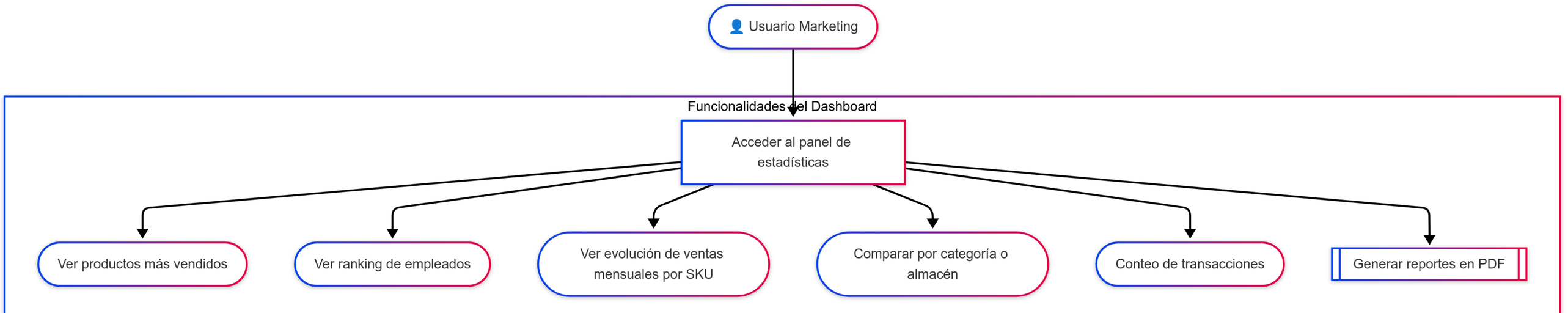
- Acceder al **dashboard de estadísticas**.
- Visualizar informes sobre:
    - Productos más vendidos
    - Variaciones de ventas por mes
    - Productos con ventas en descenso
- Filtrar por fechas, categorías o almacenes.

Ejemplo de uso:

Un usuario con rol de marketing accede a la sección de estadísticas y selecciona el período correspondiente al último trimestre. A través de los filtros por categoría y almacén, identifica una disminución significativa en las ventas de una familia de productos. A partir de esta información, puede generar un informe y trasladar recomendaciones al equipo correspondiente.

En la figura 1 se observa el caso de uso para el usuario marketing:

Diagrama Casos de Uso Marketing



2.2.2 Reponedores

Este perfil está orientado a los operarios encargados de gestionar físicamente el inventario en almacenes. Pueden consultar el stock, registrar entradas/salidas y responder ante alertas.

Funciones disponibles:

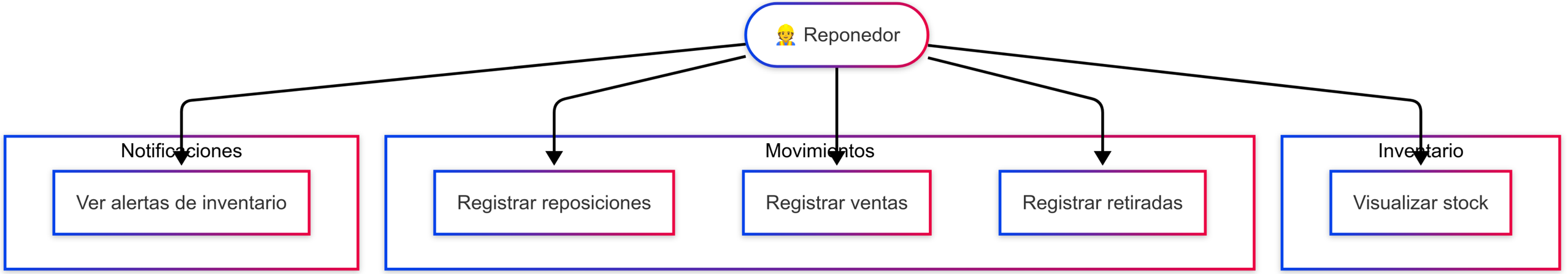
- Acceder a la sección de **gestión de inventario**.
- Visualizar productos con:
    - Stock bajo o agotado
    - Exceso de stock
    - Sin movimiento durante largo tiempo
- Registrar movimientos de tipo add o remove.

Ejemplo de uso:

Tras iniciar sesión, el reponedor accede al panel de inventario y visualiza una lista de alertas activas. Uno de los productos muestra un nivel de stock por debajo del umbral definido. El reponedor actualiza manualmente la cantidad disponible tras reponer físicamente el producto, lo que provoca la desactivación automática de la alerta.

En la figura 2 se observa el caso de uso para el usuario reponedor:

Diagrama Casos de Uso Reponedor



2.2.3 Encargado

El rol de **encargado** está pensado para supervisores o responsables de un almacén concreto. Este perfil tiene **acceso total pero limitado únicamente a su almacén**, lo que le permite gestionar tanto inventario como usuarios, movimientos y alertas asociadas exclusivamente a su centro de trabajo.

Este cambio respecto al perfil original de "administración" mencionado en el documento del TFG responde a una **mejora de precisión funcional y control por almacén**. En las especificaciones se señala que:

> "...los reponedores pueden ver alertas de stock en su almacén, y los administradores tienen acceso total."

Sin embargo, **no se menciona una figura intermedia**, lo que abre la puerta a refinar los permisos según necesidades reales de un sistema escalable. El perfil de **encargado** cubre este hueco, y además, respeta la separación de competencias, manteniendo la seguridad y la trazabilidad.

Funciones disponibles:

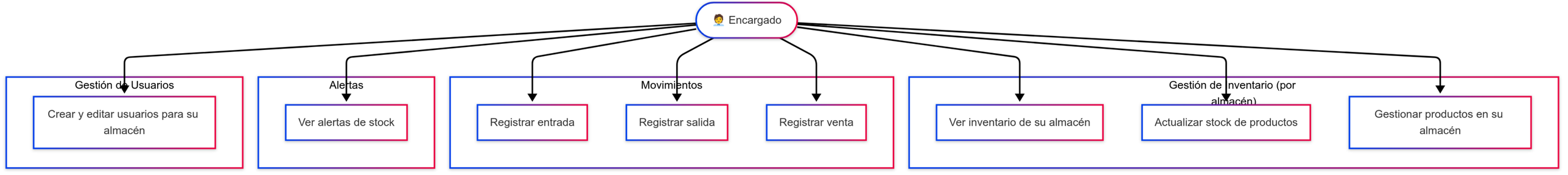
- Consultar y gestionar productos e inventario de **su almacén**.
- Registrar movimientos (add, remove, sale) exclusivamente para su almacén.
- Visualizar estadísticas e informes relacionados únicamente con su almacén.
- Ver y resolver alertas de stock (bajo, exceso, sin movimiento…).
- Crear y gestionar usuarios reponedores dentro de su almacén.

Ejemplo de uso:

Un encargado accede al sistema y visualiza el estado del inventario del almacén al que está asignado. Tras revisar los niveles de stock y las alertas existentes, decide redistribuir un producto con exceso de existencias hacia otro almacén. Además, registra un nuevo reponedor en su turno, asignándole permisos limitados dentro del mismo almacén.

En la figura 3 se observa el caso de uso para el usuario encargado:

Diagrama Casos de Uso Encargargado



2.2.4 Administradores

Los administradores tienen **acceso completo** a todas las funcionalidades. Su rol combina las capacidades de todos los perfiles anteriores.

Funciones disponibles:

- Todo lo que pueden hacer marketing, reponedores y administración.
- Supervisión completa del sistema.
- Gestión avanzada de alertas, roles y auditoría.

Ejemplo de uso:

Un usuario con rol de administrador accede a la plataforma y consulta los reportes de actividad del sistema. Desde la sección de auditoría, revisa una serie de transacciones relacionadas con productos sin movimiento durante los últimos tres meses. A partir de esta información, procede a desactivar varios productos obsoletos y reasigna otros a distintos almacenes.

En la figura 4 se observa el caso de uso para el usuario administrador:

Diagrama Casos de Uso Administrador

2.3 Limitaciones Previstas

A pesar de intentar desarrollar un sistema lo más completo posible, hay ciertas limitaciones que se aceptan como parte del alcance razonable de un TFG:

- No se incluye pasarela de pagos ni gestión económica detallada para las posibles transacciones con el cliente.
- La generación de informes se limita a filtros básicos por fecha, almacén y categoría, no incluye BI (Business Intelligence) avanzado ni análisis predictivo.
- Aunque se aplican buenas prácticas de seguridad, no se integran herramientas de escaneo automático de vulnerabilidades ni pentesting profesional.
- Sistema no internacionalizado: el sistema no contempla internacionalización para múltiples monedas/idiomas.
- No se da soporte integrado a lector de código de barras SKU para dispositivos del tipo PDA.

Estas limitaciones no afectan a la funcionalidad esencial del sistema, pero se podrían tener en cuenta en futuras ampliaciones o versiones comerciales.

2.4 Público Objetivo

El sistema está pensado para empresas pequeñas o medianas de e-commerce que necesitan tener un control centralizado de su inventario distribuido en varios almacenes, con distintos perfiles de trabajadores implicados en su gestión. Es ideal para negocios de venta online que manejan un catálogo de productos de tamaño medio y necesitan herramientas de análisis sin pagar licencias de software comerciales.

3. Tecnologías Elegidas y Justificación

La elección del conjunto de tecnologías ha sido uno de los primeros pasos importantes del proyecto. He optado por herramientas óptimas para el desarrollo de una aplicación web, con un gran reconocimiento en la industria y que ya conocía previamente, ya fuera por mi experiencia laboral o por proyectos anteriores, pero he optado por utilizar versiones más actualizadas con el objetivo de profundizar en su uso y ampliar mis conocimientos. A continuación, detallo cada una de las tecnologías empleadas, junto con una breve justificación de su elección y posibles alternativas.

3.1 Angular 19 con PrimeNG – Frontend

Alternativas consideradas:

React: Muy popular, con una gran comunidad, flexible y ligero. Requiere configurar muchas dependencias externas para lograr una solución completa.
Vue.js: Más simple que Angular, muy intuitivo para proyectos más pequeños, pero menos usado en entornos corporativos grandes.

Justificación:
He elegido Angular 19 por ser un framework completo y robusto que ya conocía por mi experiencia profesional (aunque en versiones anteriores). Esta versión más reciente ofrece mejoras de rendimiento, mejor tipado, lazy loading más eficiente y simplificación de la inyección de dependencias.
Complementarlo con **PrimeNG** fue una decisión estratégica para agilizar el desarrollo del frontend sin sacrificar calidad visual. Otras librerías como Angular Material eran opciones válidas, pero PrimeNG ofrecía más componentes listos para usar y un diseño más adaptable a una aplicación empresarial.

3.2 Java + Spring Boot – Backend

Alternativas consideradas:

Node.js + Express: Solución más ligera y rápida de arrancar, ideal para microservicios pero menos adecuada para estructuras complejas y seguridad avanzada.
Python + Django: Muy ágil para desarrollo rápido y con buen ORM, pero menos adoptado en entornos empresariales Java.

Justificación:
Java 17 y Spring Boot 3 forman un stack maduro y ampliamente usado en la industria para sistemas empresariales. Opté por esta combinación para aprovechar el ecosistema Spring (seguridad, documentación, acceso a datos, validación, etc.), su clara separación en capas y la integración fácil con herramientas como Swagger o JWT.
Además, las nuevas versiones incorporan mejoras como soporte para Jakarta EE y una configuración más moderna y declarativa, lo cual me ha permitido modernizar mis conocimientos previos.

3.3 PostgreSQL – Sistema de Bases de Datos

Alternativas consideradas:

MySQL/MariaDB: Muy extendidas, especialmente en hosting compartido, pero con menos soporte para tipos avanzados.

MongoDB: Base de datos NoSQL ideal para datos semi-estructurados, pero no adecuada para la lógica transaccional del sistema.

Justificación:
PostgreSQL se ajusta perfectamente al modelo relacional requerido por el sistema, incluyendo integridad referencial, tipos ENUM, consultas complejas y funciones personalizadas. Su compatibilidad con Spring Boot es excelente y su rendimiento está a la altura incluso para soluciones en producción. Elegí PostgreSQL también por su capacidad de crecimiento y simplicidad.

3.4 JWT con Bearer Token – Seguridad

Alternativas consideradas:

Session-based Authentication: Clásico sistema de sesiones almacenadas en el servidor, más simple pero menos escalable para SPAs.
OAuth2 con OpenID Connect: Muy robusto, recomendado para integración con terceros (Google, Facebook), pero excesivo para un entorno académico.

Justificación:
JWT ofrece una solución moderna, sin estado y segura para autenticar usuarios en aplicaciones de tipo SPA. El token generado tras el login se incluye en cada petición como Bearer y es validado por el backend, permitiendo una gestión eficaz del acceso por roles. Esta técnica mejora la escalabilidad y la seguridad del sistema al evitar almacenamiento de sesiones.

3.5 Swagger – Documentación de APIs

Alternativas consideradas:

Postman (colecciones): Muy útil en pruebas, pero no genera documentación automática ni se integra bien en el flujo de desarrollo.
Spring REST Docs: Permite documentar APIs a partir de pruebas, pero es más complejo de configurar y menos visual.

Justificación:
Swagger ha sido la opción ideal para este proyecto al generar automáticamente una interfaz gráfica que documenta y permite probar los endpoints desde el navegador. Esto ha facilitado el desarrollo, depuración y validación de las funcionalidades REST. Además, mejora la comunicación con posibles colaboradores técnicos.

Para acceder a swagger, una vez desplegada la aplicación en local, puede abrirse el siguiente enlace para visualizar la documentación generada para la API:
http://localhost:8080/swagger-ui/index.html#/

3.6 Git y GitHub – Control de Versiones

Alternativas consideradas:

Git + GitLab/Bitbucket: Plataformas similares a GitHub, con características similares, aunque algo menos extendidas en entornos personales.

Justificación:
El uso de Git es prácticamente un estándar en cualquier entorno de desarrollo actual, otras opciones como SVN (subversion) han quedado algo anticuadas y son menos comunes. Elegí GitHub como plataforma por su integración con herramientas CI/CD, su entorno de colaboración visual y su facilidad de uso. Posteriormente ha sido muy útil para el despliegue con Railway. Ha sido clave para mantener el control del avance del proyecto, trabajar con ramas por funcionalidad y documentar cada paso mediante commits.

Enlaces a los repositorios GitHub:

Mono repositorio: https://github.com/EduGar130/TFG-LSI

SubTree Front: https://github.com/EduGar130/TFG-LSI-frontend

SubTree Back: https://github.com/EduGar130/TFG-LSI-backend

3.7 Railway – Despliegue del Backend

Alternativas consideradas:

Heroku: Plataforma muy popular para pequeños proyectos y pruebas, aunque con limitaciones recientes en sus planes gratuitos.
Render: Alternativa moderna similar a Heroku, con despliegues automáticos y buena integración con Git.
AWS/GCP/Azure: Plataformas más profesionales y escalables, pero con mayor complejidad de configuración y grandes costes asociados.

Justificación:
Para el despliegue del backend opté por **Railway**, una plataforma que permite desplegar aplicaciones backend y bases de datos de forma rápida, sencilla y automatizada. Su integración con GitHub facilita el despliegue continuo: cada vez que se realiza un push a la rama principal, Railway reconstruye y despliega la aplicación de forma automática.

Railway ha resultado especialmente útil durante el desarrollo, ya que permite contar con una versión funcional del backend accesible desde cualquier lugar, lo que facilita pruebas del frontend y validaciones sin necesidad de montar el backend localmente. También ofrece métricas básicas, logs en tiempo real y una interfaz limpia para gestionar entornos y variables, lo que ha sido ideal para un proyecto académico con orientación profesional.

4. Análisis y Diseño del Sistema

La aplicación se ha diseñado siguiendo una arquitectura modular y desacoplada que facilita tanto su mantenimiento como su escalabilidad futura. A lo largo del desarrollo se han aplicado principios de diseño orientado a objetos, separación de responsabilidades y seguridad multicapa. A continuación, se presenta el análisis detallado de los distintos bloques y su interacción.

4.1 Arquitectura General del Sistema

La aplicación sigue una arquitectura **cliente-servidor** basada en una API RESTful desarrollada con Spring Boot y consumida desde un frontend SPA desarrollado en Angular. La comunicación entre ambos se realiza mediante HTTP y está protegida con JWT.

La estructura se puede descomponer en los siguientes módulos:

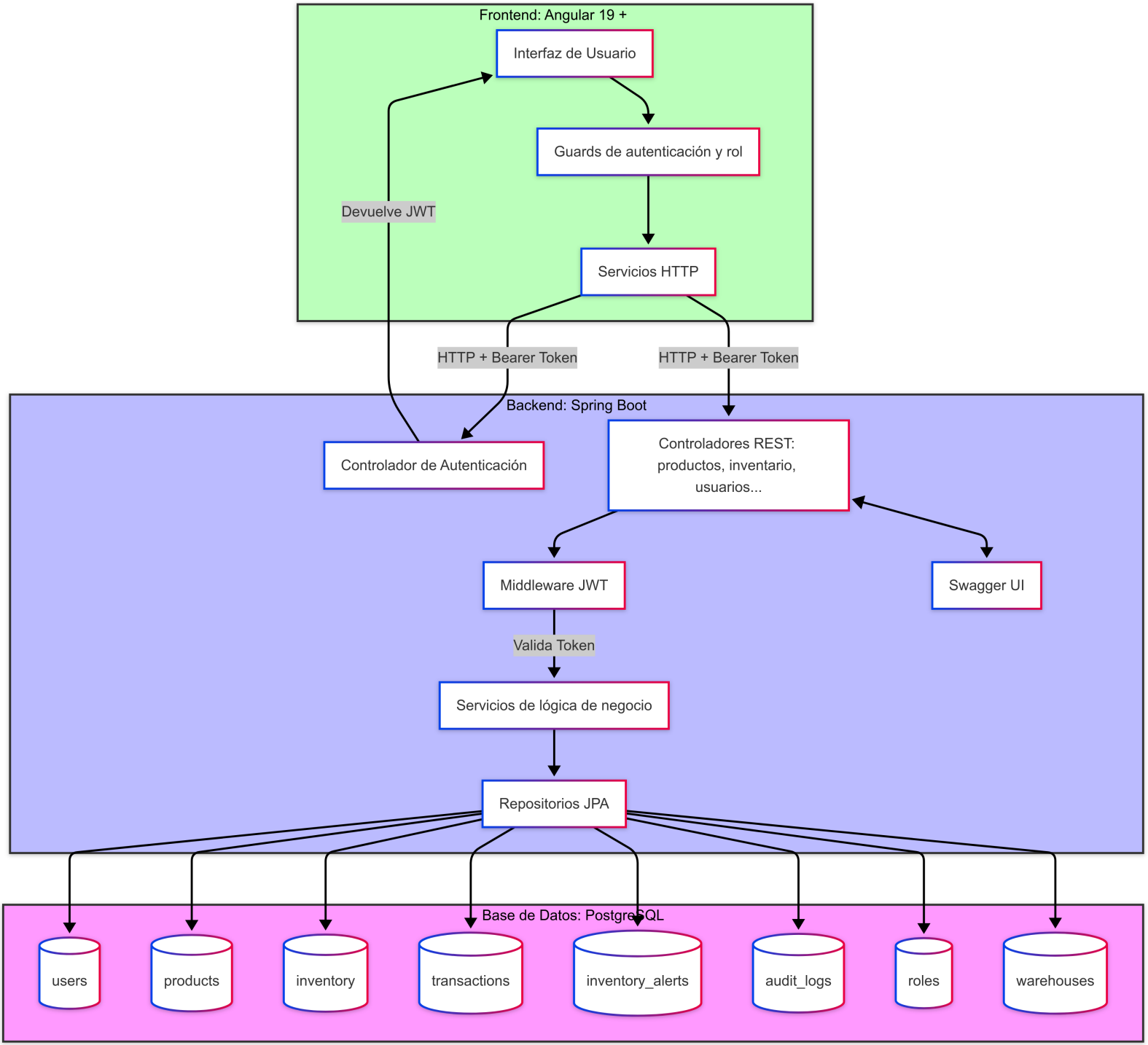
Frontend Angular (cliente): interfaz SPA basada en rutas, componentes y módulos perezosos (lazy loading), con servicios que interactúan con la API.

Backend Spring Boot (servidor): aplicación que expone endpoints REST organizados por funcionalidades, usando DTOs, mapeadores, entidades y controladores.

Base de datos PostgreSQL: almacenamiento persistente, relacional y optimizado mediante relaciones y tipos personalizados.

En la figura 5 se puede observar el Diagrama de Arquitectura General (cliente-servidor, base de datos, JWT, rutas principales) de la aplicación:

Diagrama de Arquitectura



4.2 Organización del Backend

El backend se encuentra dividido en **paquetes lógicos bien estructurados**, siguiendo el patrón clásico de Spring con capas diferenciadas:

- entity: contiene las clases mapeadas a la base de datos.
- dto: define los objetos de transferencia de datos entre frontend y backend.
- mapper: conversores entre entidades y DTOs (se evita exponer la entidad directamente).
- repository: interfaces que extienden JpaRepository para acceder a los datos.
- service: contiene la lógica de negocio.
- controller: endpoints REST accesibles desde el frontend.
- exception: control global de errores con excepciones personalizadas.
- security: configuración de JWT, filtros de autenticación, modelo de usuario autenticado.

Además, hay una clase principal InventarioBackApplication.java como punto de entrada, y un SwaggerConfig.java que documenta automáticamente los controladores expuestos.

Se ha seguido una arquitectura en **capas separadas** (Controller → Service → Repository → DB), lo que permite mayor mantenibilidad, reutilización y testabilidad.

4.3 Organización del Frontend

El frontend en Angular 19 está estructurado en **módulos funcionales independientes** según el dominio de la aplicación:

- **admin**: gestión de usuarios y permisos
- **auth**: login y guards de autenticación
- **common**: contiene el listado de constantes de la app
- **inventario**: gestión de productos y almacenes

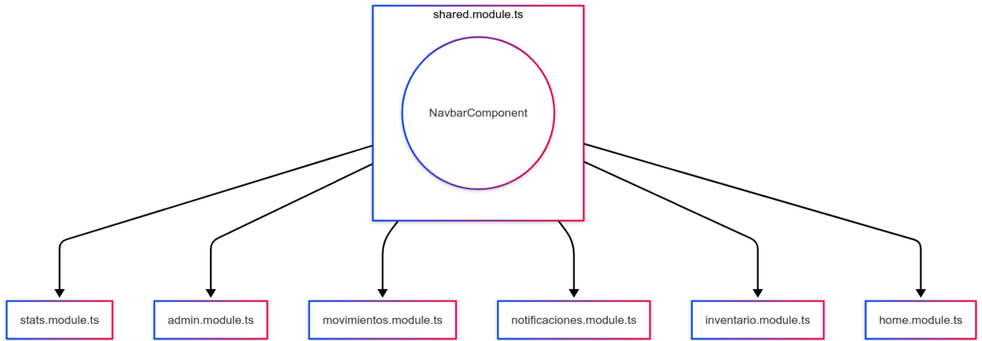
- **movimientos**: visualización de transacciones (ventas, entradas, salidas)
- **notificaciones**: alertas de stock
- **stats**: visualización de estadísticas en dashboards
- **shared**: componentes comunes como el navbar y servicios reutilizables
- **home:** incluye la página de bienvenida

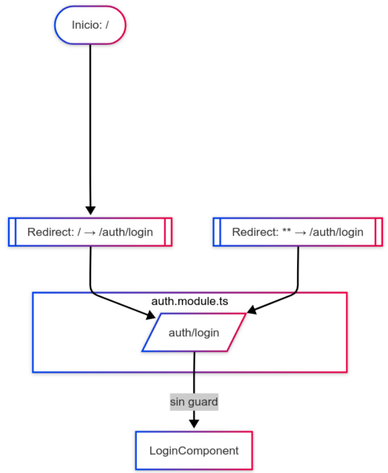
Cada módulo contiene sus propios modelos, servicios, rutas y componentes. Se emplean guards para controlar el acceso según el rol del usuario (por ejemplo, admin.guard.ts).

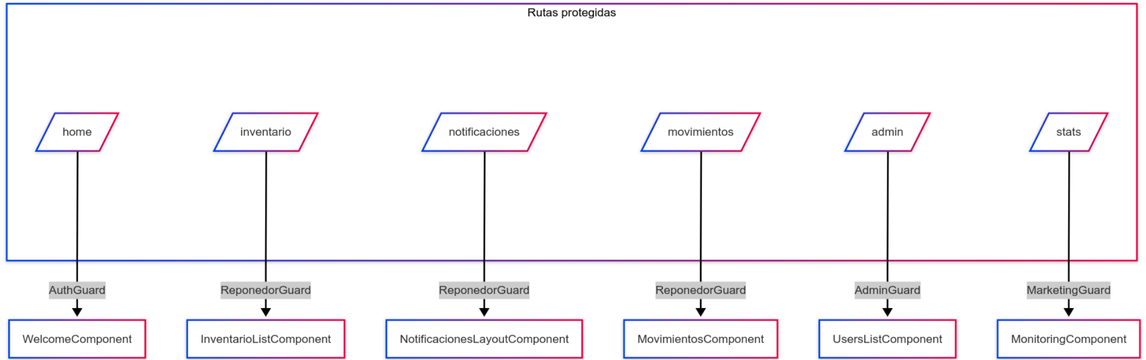
Se hace uso extensivo de **PrimeNG** para la interfaz y se mantiene una **estructura limpia y separada por responsabilidad**, lo cual es una buena práctica para proyectos de mediana y gran escala.

A continuación, en la figura 6 se muestra el Diagrama de Módulos Angular (estructura de rutas y relaciones entre módulos):

Diagrama de Módulos Angular







4.4 Seguridad y Control de Acceso

El sistema se ha diseñado con seguridad desde el inicio:

- **Autenticación con JWT**: los usuarios reciben un token al hacer login, que se envía en el header Authorization como Bearer <token> en cada petición.

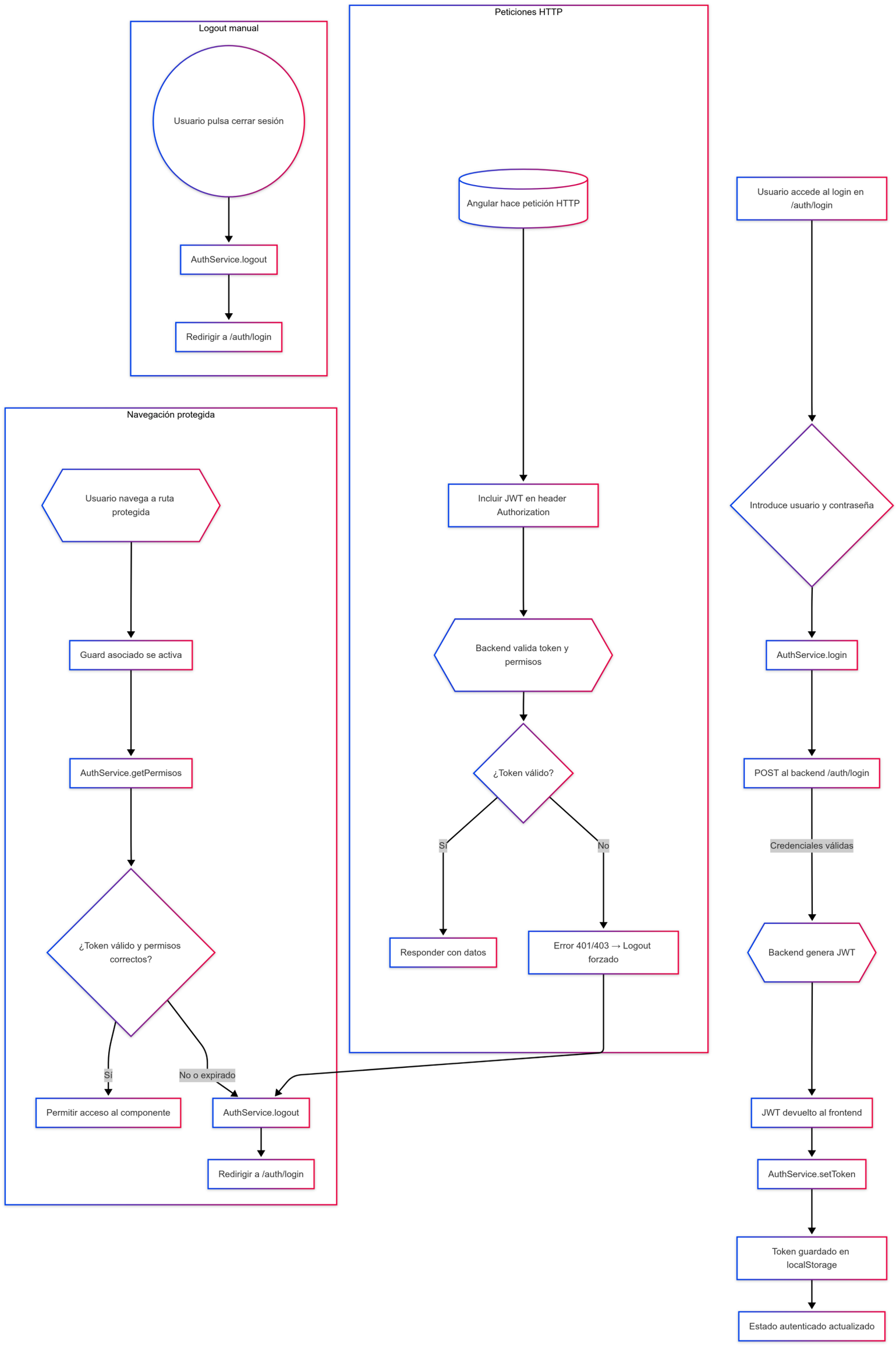
- **Control de acceso por roles**: se valida tanto en frontend (guards) como en backend (anotaciones o validaciones en servicios).

- **Protección de endpoints sensibles**: solo accesibles por usuarios con permisos adecuados.

- **Manejo global de errores**: mediante un handler global GlobalExceptionHandler.java, que centraliza las respuestas ante errores comunes.

A continuación se muestra el Diagrama de Flujo para la Autenticación y Autorización, representado en la Figura 7

Diagrama de Flujo Autorización



4.5 Flujo de Comunicación Frontend - Backend

El flujo de datos sigue esta lógica:

1. El usuario inicia sesión → se obtiene un token JWT.
2. Las peticiones desde Angular se envían con ese token en el header.
3. El backend valida el token y extrae el rol del usuario.
4. Se accede a los controladores, donde los datos viajan entre DTOs y entidades gracias a los mappers.
5. Los servicios acceden a la base de datos mediante los repositorios.
6. La respuesta se devuelve al frontend ya formateada.

Este flujo asegura consistencia, seguridad y separación entre capas.

A continuación en la figura 8 se muestra el Diagrama de Secuencia Angular ↔ Spring Boot ↔ PostgreSQL

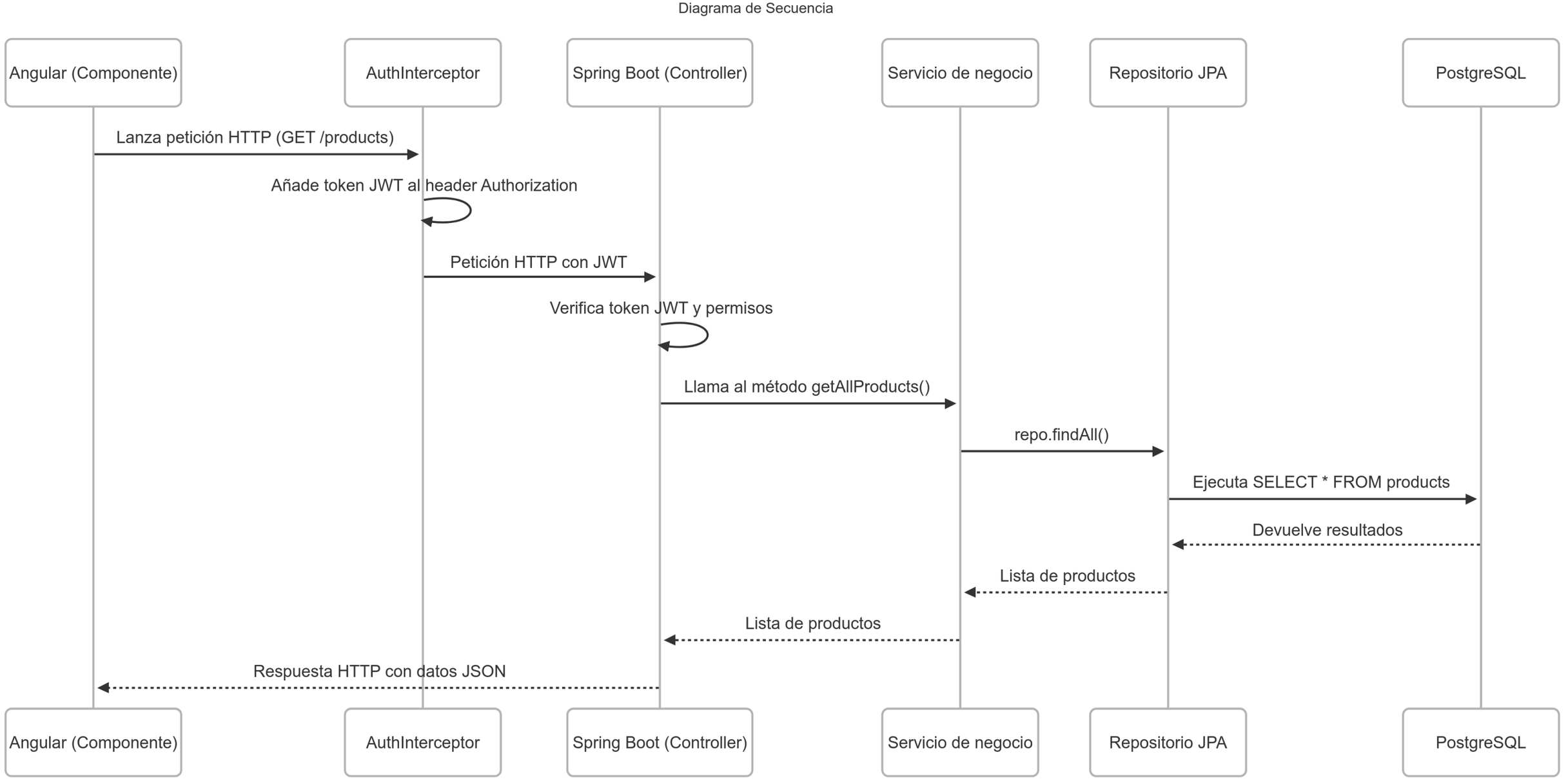


Diagrama de Secuencia
Angular (Componente)
AuthInterceptor
Spring Boot (Controller)
Servicio de negocio
Repositorio JPA
PostgreSQL
Lanza petición HTTP (GET /products)
Añade token JWT al header Authorization
Petición HTTP con JWT
Verifica token JWT y permisos
Llama al método getAllProducts()
repo.findAll()
Ejecuta SELECT * FROM products
Devuelve resultados
Lista de productos
Lista de productos
Respuesta HTTP con datos JSON
Angular (Componente)
AuthInterceptor
Spring Boot (Controller)
Servicio de negocio
Repositorio JPA
PostgreSQL

5. Diseño y Modelo de la Base de Datos

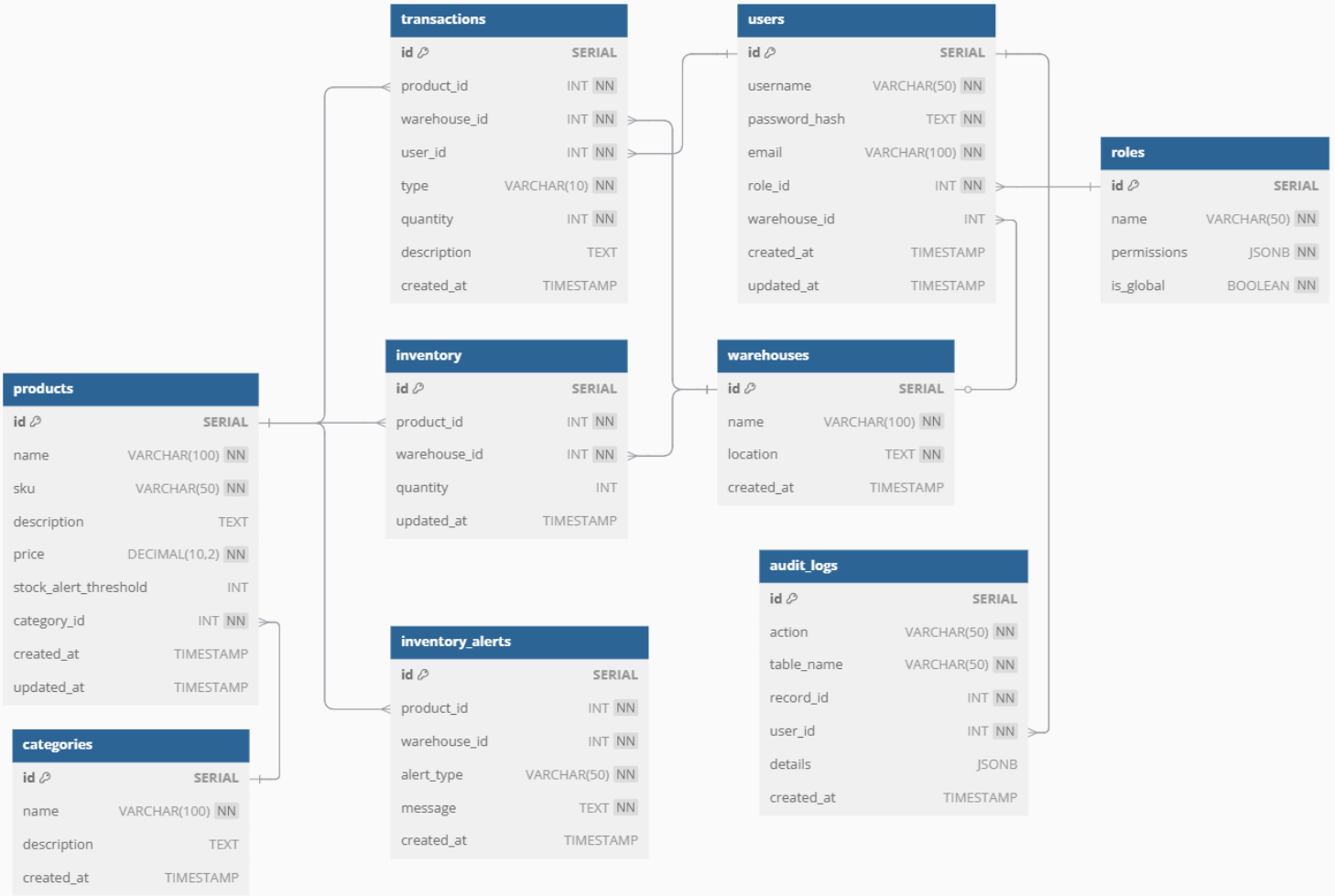
Para este proyecto se ha optado por una base de datos **relacional en PostgreSQL**, estructurada de forma que garantiza la integridad referencial, la escalabilidad y la capacidad de registrar eventos clave en el flujo del negocio. A continuación, se describen las tablas, relaciones, restricciones y elementos especiales implementados.

5.1 Entidades Principales y Relaciones

La base de datos está compuesta por las siguientes entidades clave:

- **roles**: define los distintos perfiles de usuario con sus permisos.
- **users**: almacena la información de los usuarios del sistema, incluyendo su rol y el almacén al que están asociados, si aplica.
- **categories**: agrupa productos por temática.
- **products**: almacena información detallada del producto (precio, SKU, descripción, categoría…).
- **warehouses**: almacenes físicos o virtuales donde se guarda inventario.
- **inventory**: relación entre productos y almacenes, incluyendo la cantidad disponible.
- **transactions**: registra cada movimiento de stock, ya sea venta, entrada o salida.
- **audit_logs**: registro de acciones importantes, para control y trazabilidad.
- **inventory_alerts**: notificaciones generadas ante situaciones críticas (stock bajo, exceso, etc.).

A continuación, en la figura 9 se incluye el diagrama entidad relación para la base de datos:


transactions
id SERIAL
product_id INT NN
warehouse_id INT NN
user_id INT NN
type VARCHAR(10) NN
quantity INT NN
description TEXT
created_at TIMESTAMP
users
id SERIAL
username VARCHAR(50) NN
password_hash TEXT NN
email VARCHAR(100) NN
role_id INT NN
warehouse_id INT
created_at TIMESTAMP
updated_at TIMESTAMP
roles
id SERIAL
name VARCHAR(50) NN
permissions JSONB NN
is_global BOOLEAN NN
products
id SERIAL
name VARCHAR(100) NN
sku VARCHAR(50) NN
description TEXT
price DECIMAL(10,2) NN
stock_alert_threshold INT
category_id INT NN
created_at TIMESTAMP
updated_at TIMESTAMP
inventory
id SERIAL
product_id INT NN
warehouse_id INT NN
quantity INT
updated_at TIMESTAMP
warehouses
id SERIAL
name VARCHAR(100) NN
location TEXT NN
created_at TIMESTAMP
audit_logs
id SERIAL
action VARCHAR(50) NN
table_name VARCHAR(50) NN
record_id INT NN
user_id INT NN
details JSONB
created_at TIMESTAMP
inventory_alerts
id SERIAL
product_id INT NN
warehouse_id INT NN
alert_type VARCHAR(50) NN
message TEXT NN
created_at TIMESTAMP
categories
id SERIAL
name VARCHAR(100) NN
description TEXT
created_at TIMESTAMP

5.2 Enumeraciones y Tipos Personalizados

Para mayor claridad y consistencia, se ha creado un tipo ENUM en PostgreSQL llamado transaction_type, que puede tomar los valores:

- add: entrada de stock
- remove: salida por deterioro, movimiento u otros
- sale: venta de un producto

Esto permite limitar el campo type de la tabla transactions a esos valores, mejorando la validación y evitando errores en el manejo de datos.

5.3 Funciones y Triggers

Se han definido **triggers** para mantener automáticamente la integridad de los datos temporales. Por ejemplo:

- **updated_at automático**: cada vez que se actualiza un registro de users, products o inventory, se actualiza su campo updated_at a la fecha y hora actual.

Esto se implementa con la función update_updated_at_column() en PostgreSQL, y se asocia mediante BEFORE UPDATE a las tablas indicadas.

> Esta práctica es muy útil para auditoría interna, sincronización y diagnóstico de problemas.

5.4 Seguridad e Integridad Referencial

Se han aplicado claves foráneas y restricciones NOT NULL a todos los campos que lo requieren para garantizar que:

- No haya productos sin categoría
- Cada transacción esté asociada a un producto, almacén y usuario válidos

- No se puedan eliminar registros clave si otros dependen de ellos

Además, los emails y nombres de usuario en la tabla users son únicos, evitando duplicidades de acceso.

5.5 Resumen del Script de BBDD

El script inicial realiza las siguientes acciones:

- Define un ENUM para los tipos de transacción.
- Crea todas las tablas principales con sus relaciones, claves primarias y foráneas.
- Añade columnas automáticas como created_at y updated_at.
- Define y aplica triggers con lógica personalizada.

6. Generación y Tratamiento de Datos

Durante el desarrollo de este sistema de gestión de inventario, uno de los aspectos más infravalorados al inicio fue la necesidad de contar con una base de datos poblada con datos realistas desde el primer momento. Aunque podría parecer un detalle menor, la falta de información verosímil afecta directamente a la experiencia de usuario, dificulta la validación de funcionalidades, y resta credibilidad a cualquier demostración o entrega. Pese a que en un principio intenté resolver este problema utilizando bibliotecas tradicionales como Faker, en español, los resultados no fueron satisfactorios, especialmente en lo referente a la generación de productos.

Faker resultó ser útil para generar datos personales como nombres, correos electrónicos y direcciones localizadas en España. Sin embargo, cuando se trataba de productos, el contenido generado era incoherente, sin relación alguna con el tipo de inventario esperado, y en inglés. Debido a su estructura clásica y aleatoria adjetivo + material + producto, resultaban comunes descripciones absurdas o productos sin contexto, como "fantastic rubber shoes" o "ergonomic cotton table", que no aportaban ningún valor al sistema y, de hecho, podrían llegar a distraer el foco del usuario durante una demostración.

Esta carencia motivó una solución más avanzada: integrar inteligencia artificial directamente en el backend. La decisión fue implementar la API de OpenAI para delegar en ella la generación de productos realistas, coherentes y redactados en español, adaptados a la temática del inventario configurada dinámicamente en las variables de entorno.

6.1 Lógica del seeder: Faker, OpenAI y condiciones de ejecución

La lógica de generación de datos se encuentra completamente centralizada en la clase DataSeeder, diseñada para ejecutarse automáticamente al iniciar la aplicación, gracias a la anotación @PostConstruct de Spring Boot. Este componente revisa una por una las principales tablas del sistema (roles, almacenes, categorías, productos, usuarios, inventario y transacciones) y solo lanza el proceso de seeding si detecta que esa tabla está vacía. De esta manera se evita duplicar información o sobreescribir datos existentes, garantizando una inicialización exitosa.

El sistema comienza creando los roles predefinidos si no existen, y a continuación genera entre cuatro y cinco almacenes ficticios con nombres de ciudades españolas. Luego se activa la lógica de generación de categorías y productos. Si la base de datos no contiene ningún registro de estas entidades, el sistema lanza un conjunto de peticiones (una por categoría, cuyo número se puede especificar en las variables de entorno) a la API de OpenAI. Cada una de estas peticiones incluye un prompt en el que se solicita una categoría temática (la temática puede especificarse en las variables de entorno) con diez productos asociados (tras varias pruebas se ha detectado que 10 es

el *sweet spot* para la generación automática), detallando nombre, descripción, SKU y precio. La IA responde con un JSON estructurado, que es interpretado mediante ObjectMapper y convertido en entidades persistentes dentro de la base de datos.

Durante este proceso se comprueba que los nombres de categoría no se repitan y que los SKUs de los productos generados sean únicos. En caso de colisión, el sistema genera nuevos códigos hasta encontrar uno disponible. Si por cualquier motivo la IA falla —ya sea por formato, red o límite de uso—, se activa un modo alternativo basado completamente en Faker. En este modo se generan nombres aleatorios y se distribuyen productos entre las categorías existentes, lo cual garantiza que el sistema siempre contará con un conjunto mínimo de datos funcionales aunque la calidad de estos empeore.

Tras la generación de productos, el seeder crea usuarios con diferentes perfiles: un administrador global, un usuario de marketing, un manager genérico y un reponedor genérico, además de varios managers y reponedores específicos para cada almacén. Posteriormente se genera el inventario, asignando cantidades aleatorias de cada producto a cada almacén. Finalmente, se simulan 10.000 transacciones históricas (entradas, salidas y ventas) utilizando fechas aleatorias del último año, lo cual permite que los paneles de estadísticas del sistema reflejen dinámicas creíbles desde el primer momento. Como cierre, el sistema evalúa si algún producto está por debajo de su umbral de alerta y genera automáticamente las notificaciones correspondientes si es necesario.

6.2 Reinicio controlado y pruebas iterativas

Para facilitar las pruebas y permitir un entorno controlado durante el desarrollo, implementé un script de reinicio de base de datos, disponible en el anexo E. Este script está diseñado para eliminar todos los registros respetando el orden de dependencias entre tablas, lo que evita errores de integridad referencial. Además, reinicia las secuencias de IDs, permitiendo que los identificadores comiencen de nuevo desde uno, lo cual resulta especialmente útil para depurar errores o validar comportamientos en un entorno reproducible.

Este mecanismo de reinicio, combinado con la lógica del DataSeeder, permite reiniciar el sistema desde cero en cualquier momento, obteniendo un estado completamente funcional en apenas unos segundos y con una temática a elección cada vez.

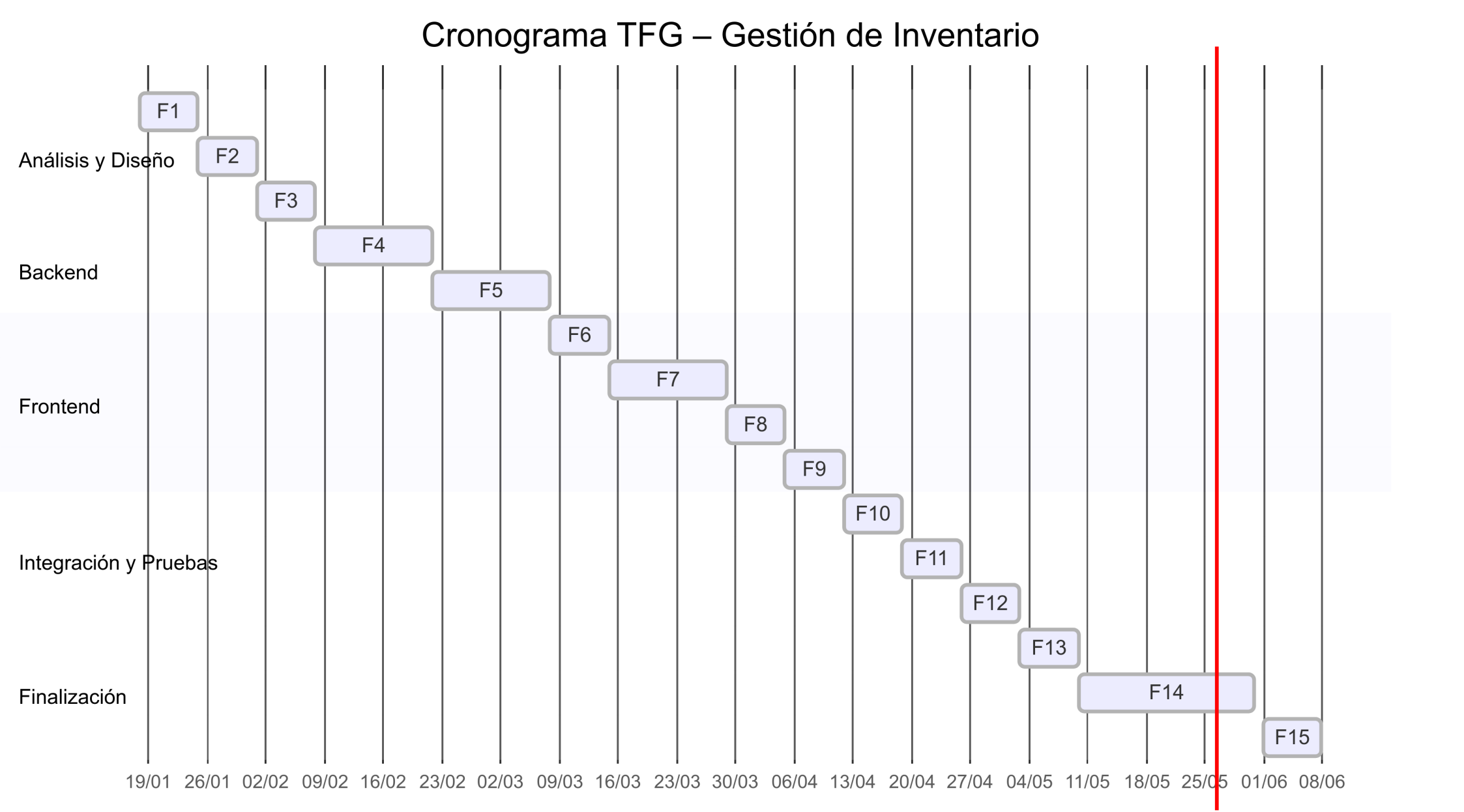
En conjunto, la estrategia de generación de datos basada en inteligencia artificial y respaldo con Faker ha sido clave para el desarrollo del sistema. No solo ha facilitado la validación funcional y el desarrollo de interfaces visuales, sino que también ha permitido presentar la aplicación como un sistema profesional y realista desde su primera ejecución. Aunque esta lógica sólo tiene sentido en entornos de desarrollo, su impacto ha sido fundamental para construir un entorno sólido, completo y listo para ser evaluado.

7. Planificación y Desarrollo

Para asegurar el cumplimiento de los plazos y mantener la motivación y el control sobre el progreso del proyecto, se ha diseñado un plan de trabajo ajustado a un calendario realista, dividido en **fases iterativas**, desde el análisis inicial hasta la entrega final.

7.1 Cronograma de Tareas

El desarrollo ha sido planificado desde el **18 de enero de 2025** hasta mediados de junio de 2025, distribuyéndose en bloques funcionales que permiten iterar y validar cada parte del sistema por separado. A continuación la figura 10, representa el Diagrama de Gantt para los hitos propuestos


Cronograma TFG – Gestión de Inventario
Análisis y Diseño
Backend
Frontend
Integración y Pruebas
Finalización
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
19/01
26/01
02/02
09/02
16/02
23/02
02/03
09/03
16/03
23/03
30/03
06/04
13/04
20/04
27/04
04/05
11/05
18/05
25/05
01/06
08/06

A continuación se encuentra la Tabla 1 en la que se muestran los hitos de desarrollo:

Fase	Actividad	Duración estimada	Fecha de inicio	Fecha fin
F1	Análisis y definición de requisitos	1 semana	18 enero	24 enero
F2	Diseño de base de datos y diagrama ER	1 semana	25 enero	31 enero
F3	Configuración del entorno y estructura de proyecto	1 semana	1 febrero	7 febrero
F4	Desarrollo backend – módulos core (entidades, repositorios, seguridad)	2 semanas	8 febrero	21 febrero
F5	Desarrollo backend – servicios y controladores	2 semanas	22 febrero	7 marzo
F6	Desarrollo frontend – estructura general y login	1 semana	8 marzo	14 marzo
F7	Desarrollo frontend – gestión de usuarios e inventario	2 semanas	15 marzo	28 marzo
F8	Dashboard y visualización de datos	1 semana	29 marzo	4 abril
F9	Sistema de alertas y transacciones	1 semana	5 abril	11 abril
F10	Integración completa frontend-backend	1 semana	12 abril	18 abril
F11	Generación de datos y pruebas funcionales	1 semana	19 abril	25 abril
F12	Pruebas de rendimiento y documentación Swagger	1 semana	26 abril	2 mayo
F13	Revisión, correcciones y validaciones finales	1 semana	3 mayo	9 mayo
F14	Redacción memoria, anexos y manuales	3 semanas	10 mayo	31 mayo
F15	Preparación defensa pública	1 semana	1 junio	7 junio

Nota: Durante todo el proceso se ha hecho uso de Git y GitHub para mantener controladas las versiones y el progreso del proyecto.

7.2 Metodología de Trabajo

Se ha seguido una **metodología ágil adaptada al desarrollo individual**, con enfoque incremental. Cada semana ha supuesto una entrega parcial que ha permitido verificar:

- Que los objetivos parciales se iban cumpliendo
- Que los módulos funcionaban de forma independiente antes de integrarse
- Que el sistema evolucionaba de forma controlada

Esto permite también **recibir feedback temprano** (por parte de compañeros, tutor o pruebas propias) y aplicar correcciones sobre la marcha sin comprometer el conjunto del proyecto.

7.3 Control de Versiones y Gestión del Proyecto

Todo el código ha sido gestionado mediante **Git**.

El repositorio en GitHub, en formato mono repositorio, ha sido el núcleo de gestión del proyecto, facilitando control de cambios, documentación de commits y posibilidad de revisión continua. Además de disponer de una guía rápida de instalación para cada módulo.

8. Pruebas y Validación

Una parte fundamental en cualquier proyecto de software es la **verificación de que el sistema cumple los requisitos definidos**, se comporta correctamente ante diferentes situaciones y es robusto frente a posibles fallos. En este TFG se han aplicado pruebas en tres niveles: funcionales, de rendimiento y de experiencia de usuario.

8.1 Pruebas Funcionales y Unitarias

Las pruebas funcionales se han centrado en comprobar que **cada funcionalidad del sistema responde correctamente** bajo condiciones normales de uso. Estas pruebas se han desarrollado tanto en el backend como en el frontend.

En el backend (Spring Boot):

- Se han implementado tests unitarios sobre servicios clave como ProductService, InventoryService o UserService. Utilizando JUnit y Mokito

- Se ha verificado que los controladores REST devuelven las respuestas esperadas.

- Se han probado distintos casos límite (productos sin stock, usuarios inexistentes, roles sin permisos, etc.).

En el frontend (Angular + PrimeNG):

- Se han realizado pruebas sobre los componentes visuales con los datos mockeados (productos.data.ts).

- Se han validado formularios, validaciones y navegación entre módulos.

- Se han probado los guards para asegurar que usuarios sin permisos no accedan a zonas restringidas.

8.2 Pruebas de Rendimiento

Aunque se trata de un proyecto académico, se ha querido validar que el sistema responde de forma eficiente en condiciones de uso normal.

- Se ha medido el tiempo de respuesta de los endpoints más utilizados (GET /products, GET /inventory, POST /transactions) con una base de datos poblada con varios cientos de registros.

- Se ha monitorizado el comportamiento de la app con herramientas como el propio navegador (devtools) y logs de Spring.

Resultados observados:

- El sistema mantiene respuestas por debajo de los 300ms en la mayoría de operaciones CRUD.

- No se detectan cuellos de botella graves en consultas gracias al uso de relaciones bien diseñadas y PostgreSQL.

8.3 Pruebas de Usabilidad

Con el objetivo de validar la experiencia real de uso de la aplicación desde el punto de vista de distintos tipos de usuarios, he llevado a cabo pruebas de usabilidad con un grupo de diez personas con perfiles variados, tanto técnicos como no técnicos.

Para facilitar el acceso a la plataforma, se aprovechó el despliegue público en **Railway**, permitiendo a todos los participantes probar la aplicación desde cualquier navegador moderno mediante el siguiente enlace:

https://tfg-lsi-frontend-production.up.railway.app/

Además, se les proporcionó un **manual de usabilidad** detallado (disponible en el **Anexo B**) y se les dieron unas instrucciones claras para realizar la prueba de forma autónoma pero guiada. El proceso sugerido fue el siguiente:

Pasos para probar la app con éxito:

1. Leer el manual de usabilidad.
2. Acceder a la web desde su ordenador.
3. Abrir el formulario proporcionado.
4. Seguir el paso a paso del formulario utilizando los distintos usuarios por rol.

Para registrar sus observaciones, todos los participantes recibieron acceso a un formulario de Google Forms diseñado específicamente para esta prueba:

Formulario: https://forms.gle/hJeWR6kB7xvjvvUm8

Este formulario incluía una **guía paso a paso tipo "logros"** que ayudaba a comprobar todas las funcionalidades clave de la aplicación en cada rol. Al finalizar, se les pedía valorar la experiencia general y proponer posibles mejoras.

Por último, para que pudieran probar el sistema en profundidad, se facilitó a todos los usuarios **un conjunto completo de credenciales**, permitiéndoles acceder como administrador, encargado, reponedor o personal de marketing según el caso.

Gracias a este proceso estructurado, ha sido posible recopilar sugerencias muy valiosas y detectar tanto errores funcionales como mejoras potenciales de la experiencia de usuario. Estas observaciones se detallan a continuación.

8.3.1 Perfil A – Estudiante de último año de Ingeniería de Software (yo mismo)

Como desarrollador del sistema, he podido observar de primera mano los obstáculos que enfrentan otros usuarios. Esta visión me ha permitido anticiparme a ciertos problemas de usabilidad y detectar áreas críticas durante el desarrollo, como la accesibilidad en dispositivos móviles o la velocidad de carga de dashboards complejos.

8.3.2 Perfil B – Desarrollador Web con experiencia técnica

Este usuario cuenta con tres años de experiencia en desarrollo web. Sus sugerencias se centraron en aspectos visuales y de comportamiento de componentes: mejoras en los estilos de los

modales, resolución de superposiciones en el navbar y mayor coherencia visual en formularios y botones.

8.3.3 Perfil C – Ingeniero en Telecomunicaciones con experiencia en desarrollo

Con experiencia en generación de informes y aplicaciones empresariales, este perfil propuso incorporar spinners de carga para mejorar la percepción de fluidez, mejorar la jerarquía de títulos y etiquetas, así como ampliar el filtro de búsqueda de productos para permitir también búsqueda por nombre, no solo por SKU.

8.3.4 Perfil D – Técnico en telecomunicaciones con experiencia en monitorización

Este usuario propuso mejoras significativas en el módulo de estadísticas, especialmente la posibilidad de comparar productos en el gráfico de ventas mensuales. También sugirió permitir silenciar alertas ya leídas y mejorar el formulario de evaluación de usabilidad para facilitar su cumplimentación.

8.3.5 Perfil E – Usuario sin experiencia previa

Fue el primer sujeto en enfrentarse a la prueba de manera completamente externa al entorno técnico. Su uso ayudó a identificar errores iniciales como la necesidad de pulsar "Enter" para iniciar sesión de forma rápida, y pequeñas incongruencias en el flujo de navegación.

8.3.6 Perfil F – Usuario sin experiencia previa

Este usuario permitió detectar errores funcionales en la sección de movimientos, proponiendo además una mejora en las validaciones de los formularios para evitar errores de introducción de datos que podrían pasar desapercibidos.

8.3.7 Perfil G – Usuario experto en entornos de inventario

Gracias a su experiencia previa con PDA y sistemas de gestión logística, propuso una mejora de gran calado: adaptar la interfaz a dispositivos móviles industriales (tipo PDA con lector de códigos de barras), permitiendo así operar sin necesidad de transportar un ordenador portátil.

8.3.8 Perfil H – Diseñador Web con experiencia en UX/UI

Este perfil proporcionó observaciones muy valiosas sobre la jerarquía visual de la aplicación, sugiriendo una mayor diferenciación entre acciones primarias y secundarias, uso de color más significativo en las alertas, y recomendaciones para mantener consistencia tipográfica y de espaciado en formularios y gráficos.

8.3.9 Perfil I – Reponedor con experiencia en entornos logísticos reales

Detectó que algunas etiquetas no eran intuitivas (“Agregar movimiento” frente a “Registrar entrada/salida”). También sugirió que las alertas fueran más visibles desde el dashboard principal.

8.3.10 Perfil J – Estudiante de informática en segundo curso

Pese a no tener experiencia profesional, este usuario demostró sensibilidad hacia los detalles técnicos. Reportó inconsistencias menores en los mensajes de error y propuso una funcionalidad de ayuda contextual en formularios (tooltips o botones de información).

8.3.11 Conclusiones de la Evaluación de Usabilidad

Las pruebas han revelado una variedad de sugerencias útiles y mejoras viables. Las más destacadas han sido:

- **Accesibilidad móvil y adaptabilidad a PDA.**
- **Claridad visual y jerarquía de información.**
- **Feedback inmediato al usuario (spinners, validaciones, alertas claras).**
- **Coherencia visual y experiencia fluida entre formularios y gráficos.**
- **Propuestas realistas desde usuarios con experiencia logística directa.**

Muchas de estas mejoras ya se han incorporado antes del cierre del proyecto. Otras de mayor envergadura (como soporte específico para PDA) se incluyen en el apartado de posibles ampliaciones futuras.

9. Requisitos Mínimos del Sistema

9.1 Requisitos para el entorno de desarrollo:

9.1.1 Frontend:

- Navegador moderno (Chrome, Firefox)
- Node.js v20+
- Angular CLI

9.1.2 Backend:

- Java 17+
- Spring Boot 3
- Maven

9.1.3 Base de datos:

- PostgreSQL 15 o superior

9.1.4 Otros:

- Docker (opcional para despliegue)
- Git (control de versiones)
- IDEs recomendados: Visual Studio Code (frontend), IntelliJ IDEA (backend)

9.2 Requisitos para el usuario final:

- Acceso a un navegador web
- Para la versión de escritorio (recomendada): resolución mínima 1280x720
- Para la versión móvil (vertical): 360 x 640 píxeles (ancho x alto)
- Conexión a internet

10. Conclusiones

10.1 Valoración de Resultados

El desarrollo de esta aplicación para la gestión de inventario online ha supuesto un **reto técnico y organizativo** que ha requerido la aplicación real de los conocimientos adquiridos durante el grado. Los resultados obtenidos han sido muy satisfactorios:

- Se ha construido una aplicación completa, funcional y bien estructurada tanto a nivel de backend como frontend.
- La base de datos es sólida, bien normalizada y preparada para crecer sin comprometer la integridad de los datos.
- Se han implementado funcionalidades avanzadas como alertas automatizadas, control de accesos por roles y dashboard interactivo.
- La arquitectura modular y la separación de capas hacen que el sistema sea fácilmente extensible y mantenible.

En definitiva, el objetivo general del proyecto —desarrollar una aplicación profesional de gestión de inventario con tecnologías modernas y buenas prácticas— se ha cumplido con éxito.

10.2 Dificultades Encontradas

Durante el desarrollo del proyecto se han presentado algunos retos que han requerido adaptación y búsqueda de soluciones:

- **Coordinación de múltiples capas**: integrar seguridad JWT, control de acceso, y el paso correcto de datos entre DTOs y entidades supuso un esfuerzo adicional.
- **Diseño modular en Angular**: estructurar correctamente los módulos lazy y la gestión de rutas y guards requirió revisar documentación y ejemplos.
- **Generación de datos realistas**: encontrar datasets adecuados y adaptarlos al modelo fue más costoso de lo previsto, y requirió crear lógica propia para poblar datos con sentido.

Estas dificultades, lejos de suponer un obstáculo, han sido una oportunidad para aprender herramientas nuevas, mejorar habilidades de debugging y reforzar el dominio de las tecnologías empleadas.

10.3 Posibles Mejoras y Ampliaciones Futuras

Aunque el sistema desarrollado cumple con los requisitos iniciales y ofrece una solución completa para la gestión de inventario en una empresa de comercio electrónico, durante el desarrollo han surgido varias ideas que podrían llevarse a cabo en futuras iteraciones para mejorar su funcionalidad, escalabilidad y adaptabilidad a distintos entornos.

Una de las mejoras más evidentes sería incorporar **notificaciones automáticas más flexibles**, no solo en forma de alertas internas visibles desde el sistema, sino también mediante correos electrónicos o mensajes instantáneos. Esto permitiría a los usuarios recibir avisos urgentes sin necesidad de estar conectados a la plataforma en todo momento.

Otra posible evolución sería **ampliar el sistema de permisos**, pasando de un control por roles a un control más granular, en el que se puedan definir acciones específicas por usuario o grupo. Esta funcionalidad resulta especialmente útil en empresas grandes, donde los niveles de responsabilidad no siempre coinciden exactamente con los perfiles predefinidos.

En cuanto al análisis de datos, el sistema actualmente presenta gráficos y estadísticas básicas que permiten interpretar el estado del inventario y las ventas. Una posible mejora sería añadir **herramientas de análisis más avanzadas**, como comparativas por periodos o detección de productos inactivos a través de reglas personalizables, lo que ayudaría en la toma de decisiones sin depender de funcionalidades externas.

También se ha detectado que sería interesante disponer de una opción para **exportar informes en formatos estándar**, como PDF o CSV. Esto facilitaría la compartición de datos con otros departamentos o su integración en otros sistemas.

Por último, para facilitar su uso en entornos más amplios, se podría contemplar la **internacionalización del sistema**, permitiendo adaptar tanto la interfaz como las unidades de medida o monedas a diferentes regiones.

Estas mejoras no se han implementado en esta primera versión por motivos de alcance y tiempo, pero se han tenido en cuenta a nivel de diseño para permitir su incorporación futura sin necesidad de grandes reestructuraciones.

Con este proyecto no solo he desarrollado una solución funcional, sino que he profundizado en buenas prácticas, organización del trabajo, control de versiones, pruebas y despliegue de un sistema real. Ha sido, en muchos sentidos, **una simulación realista del entorno laboral**, y un paso importante en mi camino profesional como desarrollador.

11. Glosario

11.1 Tecnologías y Herramientas

Angular 19

Framework de desarrollo frontend basado en TypeScript, mantenido por Google. La versión 19 incorpora mejoras de rendimiento, simplificaciones en la inyección de dependencias y mayor compatibilidad con TypeScript moderno.
Documentación oficial

PrimeNG

Conjunto de componentes UI para Angular que proporciona elementos listos para usar como tablas, gráficos, formularios y menús, con estilos modernos y altamente configurables. Ideal para aplicaciones empresariales.
Documentación oficial

Java 17

Versión LTS (Long Term Support) del lenguaje Java. Introduce mejoras en el rendimiento, nuevas características como sealed classes y pattern matching, y se considera una base estable para desarrollo profesional.
JDK 17 - Oracle

Spring Boot 3

Framework para crear microservicios y APIs REST con Java de manera rápida. Spring Boot 3 introduce compatibilidad completa con Jakarta EE 10 y requiere Java 17 como mínimo.
Spring Boot Docs

PostgreSQL

Sistema de gestión de bases de datos relacional avanzado y de código abierto. Destaca por su cumplimiento de estándares, soporte para tipos complejos y fiabilidad en entornos de producción.
Documentación oficial

JWT (JSON Web Token)

Estándar abierto para la autenticación segura sin estado. Permite transmitir información entre partes como un objeto JSON firmado, utilizado comúnmente en APIs y SPAs.
JWT.io

Bearer Token

Método de autenticación en el que el token se pasa en la cabecera Authorization como

Bearer <token>. Se usa para validar peticiones en sistemas que implementan JWT.

Swagger

Herramienta para documentar APIs REST de forma automática y visual. Permite probar los endpoints desde el navegador y genera especificaciones OpenAPI.
Swagger UI

Git

Sistema de control de versiones distribuido ampliamente utilizado en desarrollo de software. Permite registrar cambios, gestionar ramas y colaborar de forma eficiente.
Git - Sitio oficial

GitHub

Plataforma en la nube basada en Git que permite alojar repositorios, colaborar en proyectos, revisar código y automatizar despliegues mediante CI/CD.
GitHub Docs

SPA (Single Page Application)

Arquitectura de aplicaciones web donde la navegación no recarga la página completa, sino que se actualiza dinámicamente desde el frontend. Mejora la experiencia del usuario.
MDN - SPA

RESTful API

Interfaz de programación que sigue los principios REST. Utiliza HTTP para exponer recursos mediante operaciones estándar como GET, POST, PUT y DELETE.
REST - Wikipedia

DTO (Data Transfer Object)

Objeto utilizado para transportar datos entre procesos, como entre frontend y backend. Ayuda a encapsular y limitar la exposición de entidades de dominio.
DTO Pattern - Martin Fowler

Mapper

Componente encargado de convertir datos entre dos representaciones, como una entidad y un DTO. Facilita el desacoplamiento y evita lógica redundante.
MapStruct (ejemplo de librería)

Docker

Plataforma para contenerizar aplicaciones. Permite empaquetar el software y todas sus

dependencias en contenedores portables y consistentes en distintos entornos.
Docker Docs

Maven

Herramienta de construcción y gestión de proyectos Java. Automatiza compilación, pruebas, empaquetado y manejo de dependencias mediante archivos pom.xml.
Maven Documentation

Jakarta EE

Conjunto de especificaciones para desarrollo de aplicaciones empresariales en Java. Es la evolución de Java EE bajo la fundación Eclipse, y se integra con frameworks como Spring.
Jakarta EE Overview

11.2 Arquitectura y Diseño

Arquitectura por capas

Modelo de diseño en el que la aplicación se divide en capas independientes, cada una con una responsabilidad concreta: presentación (frontend), lógica de negocio (servicio), acceso a datos (repositorio) y persistencia (base de datos). Facilita el mantenimiento, testeo y escalabilidad.
Layered Architecture - Microsoft

Controladores

Clases que reciben y procesan las peticiones HTTP del frontend. En Spring Boot suelen estar anotadas con @RestController y exponen los endpoints de la API. Delegan la lógica en los servicios.
Spring Controllers

Servicios

Capa intermedia que contiene la lógica de negocio de la aplicación. Reciben datos de los controladores, los procesan, y acceden a los repositorios para realizar operaciones con la base de datos.

Repositorios

Interfaces que encapsulan el acceso a los datos. En Spring Boot se implementan con JpaRepository, permitiendo consultar, guardar o eliminar entidades sin necesidad de SQL explícito.
Spring Data JPA - Repositories

Módulos (auth, admin, inventario, estadísticas, etc.)

Divisiones lógicas del frontend Angular que agrupan componentes, servicios y rutas relacionadas. Facilitan la organización del código y permiten aplicar técnicas como lazy loading.

Guards

Mecanismos de Angular que controlan el acceso a rutas. Permiten definir condiciones para que un usuario entre o no a un módulo (por ejemplo, si está autenticado o tiene un rol concreto).

Seguridad multicapa

Enfoque que protege el sistema desde diferentes niveles: autenticación (login), autorización (roles), validación de datos, control de acceso a endpoints, y control en el frontend.

Modularidad

Propiedad de un sistema bien diseñado donde sus componentes (módulos) pueden ser desarrollados, mantenidos y probados de manera independiente. Mejora la reutilización y la legibilidad del código.
Modular Design - Wikipedia

Escalabilidad

Capacidad del sistema para manejar un crecimiento en el número de usuarios, volumen de datos o complejidad sin degradar su rendimiento ni estructura. Se puede escalar horizontal (más instancias) o verticalmente (mejor hardware).

Lazy loading

Técnica para cargar módulos o recursos solo cuando se necesitan, en lugar de al inicio. En Angular mejora la velocidad de carga inicial al dividir el código en chunks.

Separación de responsabilidades

Principio de diseño que indica que cada módulo, clase o función debe encargarse de una única responsabilidad. Permite tener un código más limpio, fácil de mantener y testear.
Single Responsibility Principle - SOLID

11.3 Base de Datos

Entidad

Representación de un objeto del mundo real dentro de la base de datos. Cada entidad se modela como una tabla y cada fila como una instancia (registro). Ejemplo: Producto, Usuario, Almacén.
Entity - Wikipedia

Relación

Conexión lógica entre dos o más entidades. Se expresa mediante claves foráneas. Por ejemplo, un producto puede estar relacionado con una categoría.

Normalización

Proceso de diseño de la base de datos que elimina redundancias y dependencias innecesarias. Mejora la integridad y facilita el mantenimiento.

Trigger

Procedimiento que se ejecuta automáticamente al producirse un evento (INSERT, UPDATE, DELETE) en una tabla. Se usa para mantener consistencia, auditar o validar.
PostgreSQL Triggers

Tipo personalizado

Tipo de dato definido por el usuario para adaptar el modelo a necesidades específicas. En PostgreSQL, pueden ser enumeraciones (ENUM), rangos, registros, etc.
PostgreSQL User-Defined Types

ENUM

Tipo de dato que limita los valores posibles de una columna a un conjunto predefinido. Por ejemplo: transaction_type con valores add, remove, sale.
ENUM Type - PostgreSQL

Integridad referencial

Restricción que asegura que una clave foránea siempre apunte a un registro existente. Previene errores como referencias huérfanas.

Claves foráneas

Campos que referencian claves primarias en otras tablas. Establecen relaciones entre entidades.
FOREIGN KEY - PostgreSQL

Claves primarias

Campos que identifican de forma única cada fila de una tabla. No pueden ser nulos ni repetirse.
PRIMARY KEY - PostgreSQL

JSONB

Tipo de dato de PostgreSQL que almacena datos en formato JSON binario. Permite

consultas, indexación y validaciones sobre estructuras semi-estructuradas.
JSONB - PostgreSQL

updated_at

Columna que almacena la fecha y hora de la última actualización de un registro. Suele actualizarse automáticamente mediante triggers.

created_at

Columna que indica cuándo se creó un registro. Generalmente se establece por defecto al insertar la fila.

Script de BBDD

Archivo que contiene las instrucciones SQL necesarias para crear la estructura inicial de la base de datos (tablas, tipos, restricciones, inserts iniciales, etc.).

11.4 Funcionalidades del Sistema

Gestión de productos

Funcionalidad que permite crear, editar, eliminar y consultar información de los productos almacenados en el sistema, como su nombre, precio, descripción, SKU y categoría.

Gestión de inventario

Permite supervisar y actualizar las cantidades de productos disponibles en cada almacén. Incluye operaciones como registrar entradas, salidas y transferencias entre almacenes.

Gestión de usuarios

Herramienta para administrar las cuentas de los usuarios del sistema: crear nuevos perfiles, modificar datos, asignar roles y controlar su acceso a las distintas funcionalidades.

Gestión de ventas

Registro y seguimiento de las ventas realizadas. Cada venta queda asociada a un producto, una cantidad, un usuario y una fecha, permitiendo analizar el rendimiento comercial.

Control de stock

Sistema que monitoriza los niveles de inventario. Lanza alertas cuando el stock de un producto está por debajo (o por encima) de ciertos umbrales definidos.

Alertas automáticas

Notificaciones generadas por el sistema cuando se detectan situaciones críticas, como bajo nivel de stock, productos sin movimiento o exceso de inventario.

Auditoría

Registro de acciones relevantes realizadas en el sistema (como movimientos de inventario o cambios de configuración) para garantizar la trazabilidad y seguridad.

Visualización de datos

Representación gráfica o tabular de la información del sistema, como ventas, stock o transacciones, de forma clara y comprensible para el usuario.

Dashboard

Vista centralizada que agrupa distintos indicadores clave del sistema (ventas, alertas, inventario, etc.) en forma de gráficos, tablas o contadores.

Informes

Documentos o vistas que agrupan datos históricos del sistema, organizados por criterios como fecha, categoría o almacén, para facilitar el análisis y la toma de decisiones.

Roles de usuario

Clasificaciones que determinan el nivel de acceso y las acciones permitidas a cada usuario del sistema. Ejemplos comunes: administrador, encargado, marketing, reponedor.

Acceso por roles

Mecanismo de seguridad que restringe o permite el acceso a distintas partes del sistema en función del rol asignado al usuario.

Notificaciones

Mensajes que alertan al usuario sobre eventos importantes del sistema, como la creación de alertas, transacciones inusuales o cambios en el estado del inventario.

Seeding

Proceso de generación automática de datos iniciales en la base de datos, útil para pruebas, desarrollo o demostraciones. Puede incluir productos, usuarios, inventario, etc.

Logs de auditoría

Entradas que registran eventos críticos o acciones sensibles (como movimientos de inventario o cambios de permisos), incluyendo información sobre qué usuario las realizó y cuándo.

Transacciones

Acciones registradas que afectan al inventario, como ventas, entradas y salidas. Incluyen detalles como fecha, usuario implicado, almacén y cantidad.

11.5 Pruebas y Validaciones

Pruebas funcionales

Validan que cada funcionalidad del sistema haga lo que se espera de ella. Simulan el uso real para asegurar que el sistema responde correctamente bajo condiciones normales.

Pruebas unitarias

Evalúan unidades individuales de código (como métodos o funciones) de forma aislada para comprobar que producen los resultados esperados. Son la base del testing automatizado.

Pruebas de rendimiento

Miden el tiempo de respuesta, la eficiencia y la estabilidad del sistema cuando se somete a una carga de trabajo representativa o elevada. Permiten identificar cuellos de botella.

Pruebas de usabilidad

Analizan cómo interactúan los usuarios reales con la aplicación. Evalúan aspectos como claridad, facilidad de uso, accesibilidad y navegación para mejorar la experiencia del usuario.

Mocking

Técnica usada en pruebas unitarias para simular dependencias externas o componentes que no se desea ejecutar (como una base de datos o una API externa), permitiendo pruebas más rápidas y controladas.

Testing

Proceso general de validación de un sistema mediante distintos tipos de pruebas. Incluye pruebas manuales y automáticas para verificar funcionalidad, rendimiento y experiencia de usuario.

@SpringBootTest

Anotación de Spring Boot que permite ejecutar pruebas de integración con todo el contexto de la aplicación cargado. Se usa para validar el comportamiento de componentes reales en conjunto.

Jest

Framework de testing en JavaScript utilizado comúnmente en aplicaciones Angular o

React. Permite realizar pruebas unitarias y de integración de forma rápida y con buen soporte para mocks.

Karma

Herramienta de ejecución de pruebas para proyectos Angular. Se integra con frameworks como Jasmine o Jest y permite correr pruebas en navegadores reales o en headless mode.

11.6 Otros conceptos importantes

Inventario distribuido

Sistema en el que los productos se almacenan en múltiples ubicaciones físicas o virtuales, y su control se gestiona desde un único software. Aumenta la eficiencia logística.

SKU (Stock Keeping Unit)

Código único asignado a cada producto para su identificación dentro del inventario. Sirve para diferenciar variantes de un mismo artículo (color, talla, modelo, etc.).
SKU - Wikipedia

Categorías

Clasificación temática que agrupa productos similares. Facilita la organización y búsqueda dentro del catálogo.

Almacenes

Ubicaciones físicas o lógicas donde se guarda el inventario. Pueden tener stock independiente y generar alertas específicas.

Movimiento de stock

Registro de cambios en la cantidad de productos en el inventario: entradas, salidas y transferencias.

E-commerce

Comercio electrónico. Consiste en la compra y venta de productos o servicios a través de internet.
E-commerce - Wikipedia

Análisis de datos

Proceso de recopilación, limpieza e interpretación de datos para extraer conclusiones útiles, detectar patrones y tomar decisiones.

Control de versiones

Técnica para gestionar los cambios en el código fuente a lo largo del tiempo. Permite trabajar en equipo, revertir errores y mantener un historial.
Version Control - Atlassian

Visualización gráfica

Representación visual de datos mediante gráficos, tablas o diagramas para facilitar su comprensión.

Repositorio

Espacio (local o remoto) donde se almacena el código fuente de un proyecto. En Git, puede estar en plataformas como GitHub o GitLab.
GitHub Repositories

Proyecto incremental

Modelo de desarrollo que divide el proyecto en pequeñas partes funcionales (incrementos), cada una con entregables y validaciones propias.

Roles: administrador, encargado, reponedor, marketing

Perfiles con distintos niveles de acceso y funciones dentro del sistema. El administrador tiene control total; los demás, permisos específicos.

Dataset

Conjunto estructurado de datos. En este proyecto, se refiere a colecciones de datos de productos o ventas usadas para pruebas o seeding.
Dataset - Wikipedia

Kaggle

Plataforma que ofrece datasets públicos, competiciones de ciencia de datos y notebooks colaborativos.
Kaggle

UNED

Universidad Nacional de Educación a Distancia, institución española de enseñanza superior a distancia.
UNED

Ingeniería Informática

Carrera universitaria centrada en el estudio de software, hardware, redes y sistemas. Forma profesionales del desarrollo tecnológico.

Baleària

Empresa española de transporte marítimo. En este contexto, se menciona como experiencia profesional del autor.
Baleària

Minsait

Compañía del grupo Indra especializada en transformación digital y soluciones tecnológicas.
Minsait

Entorno profesional

Contexto real de trabajo en empresas, donde se aplican conocimientos técnicos con objetivos productivos.

Caso ficticio

Escenario simulado que representa una situación real para fines académicos o de prueba, sin implicar una empresa real.

Aplicación web

Programa accesible desde un navegador, sin necesidad de instalación local. Puede tener backend, frontend y base de datos.
Web Application - Wikipedia

Incidencias

Eventos no deseados o excepcionales que requieren atención del usuario o del sistema. Ejemplo: producto sin stock.

Base de datos relacional

Sistema de almacenamiento de datos estructurados mediante tablas con relaciones entre ellas. Ejemplo: PostgreSQL.
Relational Database - Oracle

Consistencia

Propiedad que garantiza que los datos cumplen con las reglas del sistema y mantienen coherencia tras cualquier operación.

Niveles de acceso

Restricciones aplicadas a los usuarios según su rol para limitar o permitir ciertas acciones dentro del sistema.

Autenticación

Proceso mediante el cual un usuario demuestra su identidad, por ejemplo, a través de usuario y contraseña.

Autorización

Proceso posterior a la autenticación que determina qué puede hacer o ver un usuario dentro del sistema.

Paneles de estadísticas

Secciones de la aplicación que muestran métricas clave mediante gráficos y contadores. Facilitan el análisis rápido del estado del sistema.

Comparativas

Visualizaciones o informes que contrastan datos entre periodos, categorías o almacenes para detectar patrones o cambios.

Escenarios límite

Situaciones extremas o poco comunes que se prueban para asegurar que el sistema responde correctamente. Ejemplo: stock negativo, usuarios inexistentes.

Documentación funcional

Conjunto de documentos que describen el comportamiento, las entradas/salidas y el propósito de cada funcionalidad del sistema.

Flujos del sistema

Secuencia lógica de pasos o interacciones que sigue un usuario o un proceso dentro de la aplicación para lograr un objetivo.

Diseño de interfaces

Proceso de definir la estructura, comportamiento y aspecto visual de las pantallas y componentes con los que interactúa el usuario.

CRUD

Acrónimo de Create, Read, Update, Delete. Son las operaciones básicas que puede realizar un sistema sobre sus datos.

Organización modular

Estrategia de dividir la aplicación en secciones o módulos independientes, con funciones y dependencias bien delimitadas.

Protección de datos

Conjunto de medidas para garantizar la privacidad, seguridad y buen uso de la información personal o sensible de los usuarios.
GDPR - Reglamento General de Protección de Datos

Comercio electrónico

Modalidad de negocio basada en la venta y compra de productos o servicios a través de medios digitales.

Caso realista

Situación representada en el sistema que, aunque no sea real, refleja con precisión las condiciones y retos del mundo laboral.

Asignaturas fundamentales

Materias clave del grado en Ingeniería Informática que proporcionan las bases necesarias para el desarrollo de software profesional.

12. Referencias y bibliografía

A continuación se presenta una recopilación de las herramientas, recursos y tecnologías empleadas o consultadas durante el desarrollo del presente Trabajo de Fin de Grado. Se incluyen entornos de desarrollo, servicios web, librerías, documentación técnica y plataformas de asistencia utilizadas en distintas fases del proyecto.

Tecnologías y librerías

- **Spring Boot 3**: Framework para la creación de APIs REST en Java. Documentación oficial: https://spring.io/projects/spring-boot
- **Angular 19:** Framework frontend basado en TypeScript. https://angular.io
- **PrimeNG:** Conjunto de componentes UI para Angular. https://primeng.org
- **PostgreSQL:** Sistema de gestión de bases de datos relacional. https://www.postgresql.org
- **Swagger / OpenAPI:** Herramienta de documentación interactiva para APIs. https://swagger.io
- **JWT (JSON Web Token):** Sistema de autenticación por token. https://jwt.io
- **Mermaid.js:** Lenguaje de marcado para diagramas UML integrables en Markdown. https://mermaid.js.org
- **dbdiagram.io:** Generador de diagramas entidad-relación desde SQL o DSL. https://dbdiagram.io

Entornos de desarrollo y herramientas

- **IntelliJ IDEA:** Entorno de desarrollo integrado (IDE) para Java. https://www.jetbrains.com/idea
- **Visual Studio Code**: Editor de código ligero utilizado para el frontend. https://code.visualstudio.com
- **DBeaver**: Herramienta de administración y exploración de bases de datos SQL. https://dbeaver.io
- **Railway.app**: Plataforma para despliegue de backend y base de datos en la nube. https://railway.app

- **Google Forms**: Utilizado para obtener feedback de usuarios durante las pruebas de usabilidad. https://forms.google.com
- **Google Docs:** Usado para redactar y organizar la memoria del proyecto.
- **Awesome Screenshot** – Extensión de Google Chrome utilizada para capturar pantallas del sistema. https://chromewebstore.google.com/detail/awesome-screenshot
- **ColorZilla** – Extensión de Chrome para inspeccionar y copiar colores CSS. https://www.colorzilla.com

Control de versiones y colaboración

- **Git y GitHub**: Control de versiones y alojamiento del código fuente. https://git-scm.com | https://github.com

Asistencia y generación de contenido

- **ChatGPT (OpenAI)**: Asistente de redacción técnica, depuración de errores. https://chat.openai.com
- **OpenAI API**: Empleada en el seeder de datos para generar nombres y descripciones realistas de productos. https://platform.openai.com
- **Stack Overflow**: Consultas puntuales sobre errores de configuración, sintaxis y buenas prácticas. https://stackoverflow.com
- **Codex (OpenAI)**: Se ha utilizado en una breve prueba del nuevo agente de inteligencia artificial, se interactuó con este modelo para la revisión de una tarea concreta: localizar y corregir un error de validación en todo el repositorio. Detectó un Null Check faltante en TransactionService. La solución fue implementada respetando las convenciones del sistema y validada correctamente.
- **GitHub Copilot** – Ha servido como asistente de codificación en tiempo real, sugiriendo fragmentos de código, estructuras de control y firmas de funciones tanto en Java como en TypeScript. Aunque su uso no ha sido exclusivo, ha complementado eficazmente la escritura de código repetitivo y ha facilitado la exploración de soluciones rápidas en momentos de bloqueo.

Documentación y fuentes complementarias

- Documentación oficial de Angular, Spring Boot, PostgreSQL, PrimeNG y Swagger UI.

13. Anexos

Todos los anexos que acompañan a esta memoria están disponibles en la carpeta docs del repositorio de GitHub del proyecto. Se incluyen en formato PDF o .sql/.txt según el tipo de contenido, y sirven como documentación complementaria para la instalación, uso, configuración y pruebas del sistema.

13.1. Anexo A Manual de instalación

Contiene una guía paso a paso para desplegar el sistema en un entorno local de desarrollo. Incluye requisitos, configuración del entorno, instrucciones para ejecutar el backend, frontend y base de datos. Se puede encontrar en los respectivos README.md en github, y concretamentede los directorios code/front y code/back

13.2. Anexo B Manual de usabilidad

Manual destinado a los usuarios finales, donde se explican las funcionalidades disponibles según el rol asignado. Incluye capturas de pantalla, ejemplos de flujo de trabajo y recomendaciones de uso para una experiencia óptima.

13.3. Anexo C Swagger UI de documentación del backend

Captura de pantalla y explicación del panel Swagger UI generado automáticamente con Springdoc OpenAPI. Permite explorar y probar los endpoints REST del backend desde una interfaz gráfica accesible vía navegador.

13.4. Anexo D Script completo de inicio para la BBDD

Archivo SQL (SCRIPT BBDD.txt) que define toda la estructura inicial de la base de datos PostgreSQL: tablas, claves foráneas, triggers, funciones, constraints y roles iniciales. Es fundamental para poner en marcha el sistema desde cero.

13.5. Anexo E Script de reinicio para la BBDD

Script adicional que elimina todos los datos de la base de datos de forma ordenada respetando las dependencias, reinicia las secuencias de IDs y deja el sistema listo para un nuevo ciclo de pruebas sin alterar la estructura.