Gestión de un almacén

Raúl Fernández Delgado

INDICE

[**1. Introducción** 2](#_Toc201052543)

[**2. Propuestas para mejorar la gestión de un almacén** 3](#_Toc201052544)

[3. REQUISITOS FUNCIONALES 4](#_Toc201052545)

[**4. Diseño del Sistema** 8](#_Toc201052546)

[**5. Implementación** 9](#_Toc201052547)

# **1. Introducción**

El presente documento describe el diseño y desarrollo de un sistema de gestión de almacén para una empresa. Este proyecto tiene como objetivo proporcionar una herramienta eficiente para administrar productos, pedidos y usuarios, optimizando las operaciones del almacén y mejorando la experiencia de los clientes.  
El sistema está diseñado para ser seguro, escalable y fácil de mantener, con funcionalidades específicas para cada tipo de usuario: administradores, trabajadores y clientes. Se busca cubrir las necesidades de control de inventario, gestión de pedidos y notificaciones en tiempo real.

# **2. Propuestas para mejorar la gestión de un almacén**

Actualmente, muchas empresas necesitan un sistema integral y centralizado que les permita gestionar de manera eficaz todas las operaciones del almacén. Esto abarca el control exhaustivo de los productos, la supervisión detallada de los pedidos y la asignación precisa de roles a diferentes tipos de usuarios, garantizando siempre la seguridad y la eficiencia en los procesos.  
Principales áreas de mejora identificadas:

• Implementar un control completo y eficiente del sistema por parte del administrador.  
• Optimizar la gestión de productos y el control de stock en tiempo real.  
• Mejorar la gestión de pedidos con actualizaciones automáticas de estado.  
• Definir y aplicar roles de usuario diferenciados: administrador, trabajador y cliente.  
• Incorporar medidas de seguridad robustas para proteger la información de usuarios y las transacciones.

# 3. REQUISITOS FUNCIONALES

**3.1. Requisitos funcionales**  
🔹 Gestión de Productos  
• Permitir la creación, lectura, actualización y eliminación (CRUD) de productos.  
• Registrar el stock actual y notificar al administrador cuando esté bajo.  
• Permitir la asociación de imágenes de producto.  
• Soportar la clasificación por categorías o tipos de producto.

🔹 Gestión de Stock  
• Controlar la disminución automática del stock cuando se realiza un pedido.  
• Bloquear pedidos de productos sin stock o con stock insuficiente.  
• Permitir la consulta del stock en tiempo real para clientes y trabajadores.

🔹 Gestión de Pedidos  
• Crear pedidos por parte de los clientes.  
• Permitir a los trabajadores y administradores actualizar el estado de los pedidos (Pendiente, Aceptado, Preparación, Enviado, Entregado).  
• Cancelar pedidos:  
o Por parte del cliente (solo si el pedido está en estado Pendiente).  
o Por parte del trabajador o administrador en casos justificados (problemas de stock, errores de pedido).  
• Generar y descargar la factura en PDF para el cliente.

🔹 Gestión de Roles de Usuario  
• Registro de nuevos usuarios con rol asignado.  
• Modificación y asignación de roles (administrador, trabajador, cliente).  
• Control de acceso a funcionalidades según el rol:  
o Administrador: acceso completo.  
o Trabajador: acceso a productos y pedidos.  
o Cliente: acceso a catálogo, pedidos propios y valoraciones.

🔹 Gestión de Valoraciones  
• Permitir a los clientes valorar productos con puntuaciones y comentarios.  
• Asociar cada valoración a un producto y a un usuario.

🔹 Gestión de Usuarios  
• Registro y autenticación de usuarios.  
• Recuperación de contraseña.  
• Actualización de datos personales (correo electrónico, contraseña, etc.).  
• Baja lógica del usuario (isDeleted).

**3.2. Requisitos no funcionales**  
🔹 Seguridad  
• Uso de JWT (JSON Web Token) para la autenticación y autorización de usuarios.  
• Protección de datos sensibles mediante cifrado y encriptación de contraseñas.  
• Control de accesos según el rol del usuario.  
• Implementación de logs de acceso y acciones críticas.  
🔹 Disponibilidad y Rendimiento  
• Alta disponibilidad del sistema para evitar caídas de servicio.  
• Respuesta rápida y eficiente de la API ante múltiples solicitudes concurrentes.  
• Optimización de consultas a base de datos y uso eficiente de recursos.  
🔹 Escalabilidad  
• Capacidad de crecer en número de usuarios, pedidos y productos sin afectar el rendimiento.  
• Posibilidad de integrarse con otros sistemas externos (ERP, CRM).  
🔹 Trazabilidad  
• Registro de logs detallados para auditoría de acciones del sistema:  
o Cambios en pedidos.  
o Modificaciones de stock.  
o Acciones de administración.  
🔹 Usabilidad  
• Interfaz de usuario intuitiva y accesible para todos los perfiles (administrador, trabajador, cliente).  
• Experiencia de usuario optimizada para dispositivos móviles y de escritorio.

**3.3. Requisitos de información**  
🔹 Usuarios  
• ID único.  
• Nombre completo.  
• Correo electrónico.  
• Contraseña (encriptada).  
• Rol asignado (administrador, trabajador, cliente).  
• Estado: activo o dado de baja.  
• Fechas de creación y actualización.  
• Campo isDeleted para borrado lógico.

🔹 Clientes  
• ID único.  
• ID de usuario asociado.  
• DNI.  
• Fotografía del DNI.  
• Dirección de envío.  
• Lista de pedidos asociados.  
• Fechas de creación y actualización.  
• Campo isDeleted para borrado lógico.

🔹 Trabajadores  
• ID único.  
• ID de usuario asociado.  
• Número de seguridad social.  
• Rol asignado.  
• Fechas de creación y actualización.  
• Campo isDeleted.

🔹 Administradores  
• ID único.  
• ID de usuario asociado.  
• Rol asignado.  
• Fechas de creación y actualización.  
• Campo isDeleted.

🔹 Productos  
• ID único.  
• Nombre del producto.  
• Tipo o categoría.  
• Imagen asociada.  
• Descripción detallada.  
• Precio.  
• Stock actual.  
• ID de pedidos asociados.  
• Fechas de creación y actualización.  
• Campo isDeleted.

🔹 Pedidos  
• ID único.  
• Fecha del pedido.  
• ID de usuario que realizó el pedido.  
• Estado actual (Pendiente, Aceptado, Preparación, Enviado, Entregado).  
• Historial de cambios de estado.  
• Fechas de creación y actualización.  
• Campo isDeleted.

🔹 Líneas de venta  
• ID único.  
• ID de pedido asociado.  
• ID de producto asociado.  
• Precio del producto al momento de la compra.  
• Cantidad de productos solicitados.

🔹 Valoraciones de productos  
• ID único.  
• ID de producto asociado.  
• ID de usuario asociado.  
• Puntuación.  
• Comentario o reseña.

# **4. Diseño del Sistema**

**4.1. Modelado de Datos**  
Entidades principales:  
• Usuario: ID, nombre, correo electrónico, contraseña, rol (admin, trabajador, cliente), created, updated, isDeleted.  
• Cliente: ID, idUsuario, DNI, foto DNI, dirección de envío, listaDePedidos, created, updated, isDeleted.  
• Trabajador: ID, idUsuario, número de seguridad social, created, updated, isDeleted.  
• Admin: ID, idUsuario, created, updated, isDeleted.  
• Producto: ID, nombre, tipo, imagen, descripción, precio, stock, idPedido, created, updated, isDeleted.  
• Pedido: ID, fecha, idUsuario, estado (Pendiente, Aceptado, Preparación, Enviado, Entregado), created, updated, isDeleted.  
• Línea de Venta: ID, idPedido, idProducto, precio, cantidad.  
• Historial de cambios de estado de pedidos: ID, ID pedido, estado anterior, estado nuevo, fecha de cambio.  
• Notificación: ID, tipo (pedido, stock), mensaje, usuario destinatario, fecha de creación, estado de lectura.

**4.2. Relaciones entre Entidades**  
• Un cliente puede tener múltiples pedidos.  
• Un pedido puede incluir varios productos y viceversa.  
• Un trabajador es un usuario y puede gestionar múltiples productos y pedidos.  
• Un admin es un usuario con acceso total al sistema y puede gestionar usuarios, productos y pedidos.  
• Los productos tienen valoraciones realizadas por los usuarios.  
• Los productos generan notificaciones de stock al admin.

**4.3. Arquitectura del Sistema**  
El sistema sigue una arquitectura modular basada en Spring Boot como framework principal. Se implementan las siguientes capas:  
• Controladores (REST API): Exponen los endpoints para las operaciones de usuario, pedidos y productos.  
• Servicios: Contienen la lógica de negocio y validaciones.  
• Repositorios: Interactúan con la base de datos PostgreSQL.  
• Seguridad: Implementación de JWT para autenticación y autorización.  
• Notificaciones: Uso de Spring Email para el envío de notificaciones.

**4.4. Tecnologías Utilizadas**  
• Backend: Java con Spring Boot.  
• Base de Datos: PostgreSQL.  
• Seguridad: JWT (Json Web Tokens).  
• Notificaciones: Spring Email.  
• Documentación: Word (.  
• Documentación técnica: Swagger/OpenAPI.  
• Testing: JUnit, Mockito para pruebas unitarias y de integración.  
• Auditoría: Uso de logs con Logback.

# **5. Implementación**

La implementación del sistema se realizó utilizando Spring Boot, con una estructura en capas (controladores, servicios, repositorios) que facilita el mantenimiento y la escalabilidad.

**5.1. Seguridad**  
• Implementación de JWT para el manejo de autenticación y autorización.  
• Cada rol tiene acceso restringido a sus propias funcionalidades.  
• Protección de endpoints mediante filtros de seguridad.

**5.2. Gestión de Productos**  
• CRUD completo para productos (alta, baja, modificación y consulta).  
• Control de stock con notificación automática al administrador cuando el stock es cero o bajo.

**5.3. Gestión de Pedidos**  
• Cambios de estado con notificaciones al usuario y al trabajador (Pendiente → Aceptado → Preparación → Enviado → Entregado).  
• Cancelación de pedidos según el estado y el rol del usuario.  
• Generación de factura en PDF para el cliente a través de correo electrónico.

**5.4. Roles y Permisos**  
• Implementación de control de acceso basado en roles: administrador, trabajador y cliente.  
• Asignación de roles mediante endpoints seguros para el administrador.

**5.5. Notificaciones**  
• Spring Email para el envío de correos automáticos a los usuarios en diferentes eventos:  
o Pedido aceptado.  
o Pedido enviado.  
o Pedido entregado.  
o Notificación de stock bajo para el administrador.

**5.6. Base de Datos**  
• PostgreSQL como motor de base de datos.  
• Relaciones definidas según el modelo relacional descrito.  
• Control de integridad y auditoría de cambios (created, updated, isDeleted).

**5.7. Arquitectura de la API REST**  
• Se implementa una API RESTful utilizando Spring MVC.  
• Endpoints organizados de forma lógica y agrupados por recursos: /productos, /pedidos, /usuarios, etc.  
• Uso de DTOs (Data Transfer Objects) para separar la lógica de la entidad y la capa de presentación, asegurando flexibilidad y seguridad.  
• Inclusión de validaciones mediante anotaciones (por ejemplo, @Valid, @NotNull, @Size).  
• Respuesta uniforme en formato JSON.

**5.8. Documentación de la API**  
• Uso de Swagger/OpenAPI con SpringDoc para generar documentación interactiva.  
• Acceso a la documentación de la API a través del endpoint /swagger-ui.html.  
• Descripción de cada endpoint: método HTTP, parámetros de entrada, respuesta esperada y códigos de error.

**5.9. Registro de Logs y Auditoría**  
• Uso de SLF4J y Logback para el registro de eventos:  
o Errores y excepciones.  
o Acciones de administración y cambios de estado de pedidos.  
• Campos created, updated e isDeleted en cada entidad para auditoría de cambios.  
• Posibilidad de filtrar logs para análisis posterior.

**5.10. Pruebas y Calidad del Código**  
• Uso de JUnit y Mockito para pruebas unitarias en los servicios y controladores.  
• Cobertura de pruebas para casos de éxito y de error.  
• Pruebas de integración con base de datos en memoria (H2) para validar la persistencia.  
• Uso de SonarQube (opcional) para medir la calidad del código y detectar vulnerabilidades.