ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO
FACULTAD: INFORMÁTICA Y ELECTRÓNICA
ESCUELA DE INGENIERÍA EN SISTEMAS
CARRERA: SOFTWARE

GUÍA DE LABORATORIO DE APLICACIONES INFORMÁTICAS II

PRÁCTICA No. 2

1. DATOS GENERALES:

NOMBRE DEL ESTUDIANTE	CODIGO DEL ESTUDIANTE
Scarlet Cayapa	7166

FECHA DE REALIZACIÓN:	FECHA DE ENTREGA:
2026/01/30	2026/02/03

2. OBJETIVO(S):

2.1. GENERAL

Documentar un plan de mantenimiento integral para el sistema de gestión de inventario mediante el análisis exhaustivo de su arquitectura, la identificación sistemática de oportunidades de mejora y la propuesta de acciones clasificadas por tipo de mantenimiento y nivel de prioridad.

2.2. ESPECÍFÍCOS

- Analizar la arquitectura del sistema y su modelo de datos para identificar componentes críticos sujetos a mantenimiento.
- Clasificar las oportunidades de mejora detectadas según los cuatro tipos de mantenimiento; correctivo, adaptativo, perfectivo y preventivo.
- Proponer un plan de intervención priorizado que detalle módulos a modificar, acciones específicas y criterios de aceptación para cada mejora.
- Establecer riesgos potenciales del proceso de mantenimiento y definir planes de mitigación para garantizar la estabilidad del sistema.

3. METODOLOGÍA

El presente trabajo aplica una metodología iterativa fundamentada en el estándar internacional ISO/IEC 14764:2022 para mantenimiento de software, complementada con prácticas ágiles de gestión de cambios. La metodología se estructura en cinco fases secuenciales con retroalimentación continua:

Fase 1: Análisis del sistema existente

Se realizó una revisión exhaustiva de la arquitectura de tres capas implementada con el patrón MVC + Repository, incluyendo el análisis del código fuente de 8 controladores, 7 modelos, 8 repositorios y 5 middlewares. Se examinó el modelo de datos relacional con sus 8 tablas principales y 12 relaciones, evaluando normalización, índices y restricciones de integridad. Esta fase identificó componentes críticos, deudas técnicas acumuladas, vulnerabilidades de seguridad y oportunidades de optimización mediante revisión de código estática y análisis de dependencias.

Fase 2: Clasificación de oportunidades de mejora

Los hallazgos del análisis se categorizaron según los cuatro tipos de mantenimiento definidos en ISO/IEC 14764:2022. Se aplicó un enfoque sistemático de clasificación evaluando: la naturaleza del cambio (correctiva, adaptativa, perfectiva o preventiva), el origen de la solicitud (error detectado, cambio de requisitos, mejora proactiva), el impacto en la funcionalidad visible y la urgencia de implementación. Cada oportunidad fue documentada con su justificación técnica y evidencias de soporte.

Fase 3: Priorización por impacto y criticidad

Se evaluó cada mejora mediante una matriz de riesgo tridimensional considerando:
(1) impacto en seguridad (vulnerabilidades OWASP)
(2) estabilidad del sistema (disponibilidad, integridad de datos)
(3) experiencia del usuario (usabilidad, rendimiento).

Se asignaron niveles de prioridad (P1-Alta, P2-Media, P3-Baja, P4-Diferida) según criterios cuantificables: tiempo de implementación estimado, recursos técnicos requeridos, dependencias entre mejoras y riesgos asociados.

Fase 4: Planificación de intervenciones

Se diseñaron acciones específicas por módulo incluyendo: archivos a modificar con líneas de código afectadas, criterios de aceptación verificables mediante pruebas automatizadas, herramientas y dependencias requeridas (librerías, frameworks, servicios externos), y estrategias de mitigación de riesgos con planes de rollback. Cada intervención fue documentada con pseudocódigo, diagramas de flujo y ejemplos de implementación para facilitar la ejecución por parte del equipo de mantenimiento.

Fase 5: Documentación del plan de mantenimiento

La elaboración de este informe técnico como entregable final consolidó todos los hallazgos, clasificaciones, priorizaciones y planificaciones en un documento estructurado que sirve como guía operativa para futuros desarrolladores mantenedores.

Se incluyeron diagramas de arquitectura, modelos de datos, tablas de decisión y anexos técnicos que complementan la documentación del código fuente.

4. EQUIPOS Y MATERIALES:

- Computador
- Entorno integrado de desarrollo (IDE)
- Aula virtual
- Acceso a internet
- Bibliografía

5. MARCO TEORICO:

El mantenimiento de software es un proceso técnico continuo que comprende la modificación de un producto software después de su entrega, con el propósito de corregir defectos, adaptarse a cambios en el entorno, mejorar funcionalidades o prevenir futuros problemas (ISO/IEC/IEEE, 2022). Según estudios empíricos, entre el 60% y 80% del esfuerzo total en el ciclo de vida de un sistema se destina a actividades de mantenimiento, superando con creces el esfuerzo de desarrollo inicial (Sommerville, 2016).

5.1 Clasificación de tipos de mantenimiento

El estándar internacional ISO/IEC/IEEE 14764:2022 establece cuatro categorías fundamentales de mantenimiento, clasificadas según su motivación y naturaleza (ISO/IEC/IEEE, 2022):

- Mantenimiento correctivo: Modificación reactiva del software para corregir defectos, errores o fallos detectados durante su operación. Representa aproximadamente el 20% del esfuerzo total de mantenimiento en sistemas bien diseñados (Sommerville, 2016).

- Mantenimiento adaptativo: Modificación proactiva para adaptar el software a cambios en su entorno operativo, como actualizaciones de sistemas operativos, bases de datos, regulaciones legales o integración con nuevos sistemas (ISO/IEC/IEEE, 2022).

- Mantenimiento perfectivo: Mejora de funcionalidades existentes o incorporación de nuevas características para satisfacer necesidades cambiantes de los usuarios, optimizar rendimiento o mejorar la usabilidad (Sommerville, 2016).

- Mantenimiento preventivo: Modificación anticipada para mejorar la mantenibilidad futura del software, mediante refactorización, actualización de dependencias obsoletas o mejora de la documentación, sin alterar su funcionalidad visible (ISO/IEC/IEEE, 2022).

Esta clasificación evolucionó a partir del trabajo pionero de Lientz y Swanson (1980), quienes identificaron inicialmente tres categorías (correctivo, adaptativo y perfectivo), posteriormente ampliadas con el mantenimiento preventivo para abordar la deuda técnica acumulada (Lientz & Swanson, 1980).

5.2 Proceso de mantenimiento según estándares

El proceso de mantenimiento se estructura en cinco actividades principales definidas por ISO/IEC/IEEE 14764:2022: (1) análisis del problema y modificación, (2) implementación de cambios, (3) verificación y validación, (4) migración/despliegue, y (5) revisión del proceso de mantenimiento. Cada actividad incluye tareas específicas como gestión de configuración, control de versiones y documentación de cambios, garantizando trazabilidad y calidad en las intervenciones (ISO/IEC/IEEE, 2022).

6. PROCEDIMIENTO:

1. Introducción

1.1 Propósito del documento

El presente documento constituye una guía técnica integral y estructurada de mantenimiento para el sistema H-M-C Inventory Management System, dirigida específicamente a desarrolladores mantenedores actuales y futuros del software. Su propósito fundamental es proporcionar un marco de referencia sistemático que oriente las intervenciones técnicas de mantenimiento mediante criterios objetivos de clasificación por tipo de mantenimiento según ISO/IEC 14764:2022, niveles de prioridad fundamentados en matrices de impacto-riesgo, y criterios de aceptación verificables que permitan evaluar la correcta implementación de cada mejora propuesta.

1.2 Alcance del mantenimiento

El alcance de este plan de mantenimiento abarca exclusivamente la versión actual del sistema H-M-C Inventory Management System alojada en el repositorio oficial de GitHub del proyecto. Las versiones y componentes cubiertos incluyen:

Capa de Presentación (Frontend):

- Framework: React 18.x con Vite como bundler y servidor de desarrollo.
- Gestión de estado: Redux Toolkit para estado global, TanStack Query (React Query) para estado del servidor.
- Routing: React Router v6 para navegación SPA.
- UI/Estilos: CSS Modules, librería de componentes custom.

Capa de Lógica de Negocio (Backend):

- Runtime: Node.js versión LTS con Express.js como framework web.
- API REST: 8 endpoints principales distribuidos en 5 rutas (auth, users, products, sales, inventory).
- Controladores: 8 controladores que implementan lógica de negocio.
- Modelos y Repositorios: 7 modelos de dominio con sus respectivos repositorios para acceso a datos.

Capa de Datos (Base de Datos):

- SGBD: PostgreSQL versión 12 o superior.
- Esquema: 8 tablas principales con modelo relacional normalizado en 3FN.
- Relaciones: 12 relaciones de clave foránea con restricciones de integridad referencial.
- Índices: Índices primarios automáticos y 5 índices secundarios para optimización de consultas frecuentes.

Middleware de Seguridad y Autorización:

- Autenticación: Passport.js con estrategia local y gestión de sesiones server-side.
- Autorización: Sistema RBAC (Role-Based Access Control) con tres roles jerárquicos: superadmin, admin y staff.
- Permisos: Matriz de permisos granulares por módulo y acción (create, read, update, delete).

Servicios Auxiliares y Configuración:

- Servicio de email: Nodemailer con soporte SMTP para notificaciones transaccionales.
- Configuración CORS: Política de mismo origen con orígenes permitidos configurables.
- Gestión de sesiones: Express-session con almacenamiento en base de datos PostgreSQL.
- Rate limiting: Express-rate-limit para prevención de ataques de fuerza bruta.

Procesos implementados:

Módulo	Proceso
Autenticación	Registro de usuarios
	Login con sesiones
	Logout
	Verificación de email
	Recuperación de contraseña
Productos	CRUD completo
	Búsqueda y filtrado
	Variantes de producto
Inventario	Ajuste manual de stock
	Logs de movimientos
	Alertas de bajo stock
Ventas	Registro de ventas
	Cálculo automático de ganancia
	Reducción automática de stock
Analytics	Estadísticas diarias/mensuales
	Valor de inventario
	Productos más vendidos
	Ganancias por período
UI/UX	Tema oscuro/claro
	Diseño responsivo
	Animaciones

Procesos Implementados Parcialmente:

Proceso	Faltante
Gestión de usuarios	Solo visualización y cambio de rol, no hay edición de perfil ni eliminación.
Cancelación de ventas	No se puede anular una venta.
Historial de precios	No se registra cuando cambian los precios.
Multipleventas en carrito	Solo una variante por venta.

Procesos No Implementados:

Proceso	Faltante
Devoluciones/Reembolsos	No se pueden procesar devoluciones.
Múltiples ítems por venta	Cada venta es de un solo producto.
Reportes exportables	No hay exportación a PDF/Excel.
Gestión de proveedores	No hay registro de proveedores.
Órdenes de compra	No hay gestión de compras a proveedores.
Notificaciones en tiempo real	No hay WebSocket/push notifications
Auditoría de acciones de usuario	Solo logs de inventario, no de todas las acciones.
Backup automático	No hay sistema de respaldos.
Multi-tienda/Multi-sucursal	Sistema single-store.
Código de barras	No hay escaneo de códigos.

1.3 Definiciones, acrónimos y abreviaturas

Términos Técnicos Generales:

Término	Definición
API	Application Programming Interface - Interfaz de programación que permite la comunicación estandarizada entre el frontend y el backend mediante protocolo HTTP.
MVC	Model-View-Controller - Patrón arquitectónico que separa la lógica de negocio (Model), la presentación (View) y el control del flujo de datos (Controller).
SPA	Single Page Application - Aplicación web de una sola página que carga dinámicamente el contenido mediante JavaScript sin recargar completamente el navegador.
RBAC	Role-Based Access Control - Control de acceso basado en roles jerárquicos de usuario: superadmin (acceso total), admin (gestión operativa), staff (operaciones básicas).
CRUD	Create, Read, Update, Delete - Operaciones básicas de manipulación de datos en sistemas de información.

Términos de Seguridad:

Término	Definición
CORS	Cross-Origin Resource Sharing - Mecanismo de seguridad que controla las solicitudes HTTP entre diferentes orígenes (dominios, protocolos o puertos).
MITM	Man-In-The-Middle - Tipo de ataque donde un tercero intercepta y potencialmente modifica la comunicación entre dos partes.
SSL/TLS	Secure Sockets Layer / Transport Layer Security - Protocolos criptográficos que proporcionan comunicación segura cifrada sobre redes.
CSRF	Cross-Site Request Forgery - Ataque que fuerza a usuarios autenticados a ejecutar acciones no deseadas en aplicaciones web.
XSS	Cross-Site Scripting - Vulnerabilidad que permite inyectar scripts maliciosos en páginas web vistas por otros usuarios.

Términos de Base de Datos:

Término	Definición
PITR	Point-In-Time Recovery - Técnica de recuperación de base de datos que permite restaurar datos a un momento específico en el tiempo.
N+1 Query	Problema de rendimiento donde se ejecutan N consultas adicionales a la base de datos por cada registro obtenido en una consulta inicial, generando carga innecesaria.
ORM	Object-Relational Mapping - Técnica de programación que convierte datos entre sistemas incompatibles usando programación orientada a objetos.
3FN	Third Normal Form - Nivel de normalización de bases de datos que elimina dependencias transitivas para reducir redundancia.

1.4 Referencias:

Estándares Internacionales:

- ISO/IEC 14764:2022 - Software Engineering - Software Life Cycle Processes - Maintenance: Marco normativo para procesos de mantenimiento de software.
- ISO/IEC 25010:2011 - Systems and Software Quality Requirements and Evaluation (SQuaRE): Modelo de calidad de software con características y subcaracterísticas.

Estándares de Seguridad:

- OWASP Top 10 2021: Lista de los diez riesgos de seguridad más críticos en aplicaciones web, utilizada para priorización de vulnerabilidades.
- OWASP API Security Top 10: Guía especializada para seguridad en interfaces de programación de aplicaciones.

Estándares de Código:

- ESLint - JavaScript Linting: Estándar de facto para análisis estático de código JavaScript, detección de problemas y enforcement de convenciones.
- Airbnb JavaScript Style Guide: Guía de estilo ampliamente adoptada en la comunidad JavaScript.

Estándares de API:

- RESTful API Design: Principios arquitectónicos para diseño de APIs web escalables y mantenibles.
- HTTP Status Codes (RFC 7231): Especificación de códigos de respuesta HTTP semánticamente correctos.
- JSON API Specification: Especificación para construcción de APIs con formato JSON consistente.

2. Descripción General del Sistema

2.1 Resumen del sistema

H-M-C Inventory Management es un sistema web integral de gestión de inventario desarrollado específicamente para el sector de comercio de productos electrónicos, con énfasis en dispositivos móviles (smartphones, tablets) y accesorios tecnológicos. El

sistema implementa una arquitectura de tres capas con separación clara de responsabilidades y comunicación mediante API REST.

Funcionalidades Principales del Sistema:

1. **Gestión Avanzada de Productos con Variantes:** El sistema permite registrar productos con múltiples variantes (color, capacidad de almacenamiento) como entidades independientes con SKUs únicos, precios diferenciados y control de inventario individual. Soporta categorización jerárquica, asociación de imágenes múltiples por producto, y definición de umbrales de stock mínimo con alertas automáticas.

2. **Control de Inventario Inteligente:** Implementa seguimiento en tiempo real de entradas y salidas de mercancía con trazabilidad completa. Genera alertas proactivas cuando los niveles de stock caen por debajo del umbral mínimo configurado, permitiendo reabastecimiento anticipado. Mantiene historial detallado de todos los movimientos de inventario con timestamps y responsables.

3. **Registro de Ventas con Análisis de Rentabilidad:** Permite registrar transacciones de venta con cálculo automático de ganancias mediante comparación entre precio de costo y precio de venta. Actualiza automáticamente el inventario al procesar ventas. Proporciona estadísticas básicas de ventas y ganancias por período.

4. **Sistema de Control de Acceso Basado en Roles (RBAC):** Implementa tres niveles jerárquicos de usuarios con permisos granulares: Superadmin (acceso total incluyendo gestión de usuarios y configuración del sistema), Admin (operaciones de gestión de productos, inventario y ventas), y Staff (operaciones básicas de ventas e inventario limitadas). Cada rol tiene una matriz de permisos específica por módulo y acción.

5. **Autenticación Robusta con Verificación de Email:** El proceso de registro incluye envío de email de verificación con token temporal para confirmar la autenticidad del correo electrónico. Implementa recuperación de contraseña mediante token de un solo uso enviado por email. Las sesiones se gestionan del lado del servidor con expiración configurable.

Objetivos Estratégicos del Sistema:

- **Centralización de Información:** Consolidar en una única plataforma todos los datos relacionados con productos, movimientos de inventario, ventas y usuarios, eliminando sistemas dispersos y hojas de cálculo manuales.

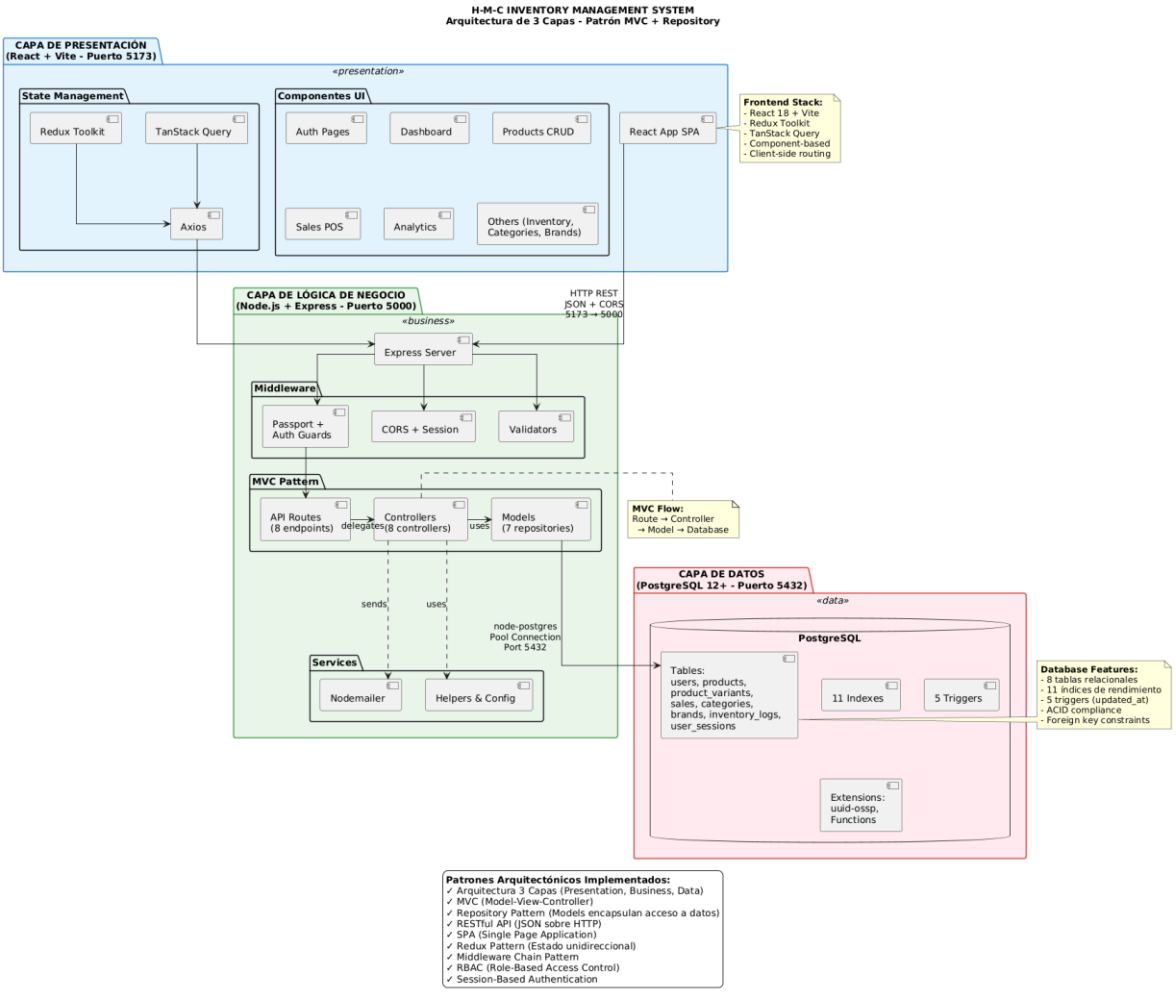
- **Automatización de Procesos Críticos:** Reducir errores humanos mediante cálculos automáticos de ganancias, actualización automática de inventario post-venta, y generación de alertas proactivas de reabastecimiento.

- **Control de Acceso Granular:** Garantizar la seguridad de la información mediante permisos específicos por rol, impidiendo accesos no autorizados a funcionalidades críticas como cambio de precios o gestión de usuarios.

- **Trazabilidad Completa:** Mantener auditoría detallada de todos los movimientos de inventario y transacciones con responsables identificados, facilitando análisis forense en caso de discrepancias.

2.2 Arquitectura del sistema

Diagrama de arquitectura



Componentes principales

Componente	Tecnología	Función
Frontend	React + Vite	Interfaz de usuario
Backend	Node.js + Express	API REST
Base de Datos	PostgreSQL	Almacenamiento
Autenticación	Passport.js	Sesiones y login
Email	Nodemailer	Notificaciones

Dependencias internas y externas

Paquete	Propósito
express	Framework web
pg	Driver PostgreSQL
passport	Autenticación
bcryptjs	Hash de contraseñas
nodemailer	Envío de emails
express-rate-limit	Limitación de peticiones

2.3 Entornos

Entorno	Características
Desarrollo	SSL desactivado, cookies inseguras, rutas test habilitadas

Pruebas	Base de Datos separada, email mock, rate limiting ampliado
Producción	SSL activo, cookies seguras, rutas test deshabilitadas

3. Tipos de Mantenimiento

3.1 Mantenimiento correctivo

Corrección de errores detectados:

Problema	Archivo	Solución
Token de verificación expuesto en respuesta	authController.js	Eliminado token de la respuesta JSON
Rutas /api/test en producción | server.js	server.js	Condicionado a NODE_ENV !== 'production'
CORS con origen único	server.js	Implementada validación dinámica múltiple

3.2 Mantenimiento adaptativo

Cambios para adaptarse a diferentes entornos:

Cambio	Archivo	Descripción
SSL configurable	database.js	Variable DB_SSL_REJECT_UNAUTHORIZED
CORS múltiples orígenes	server.js	Variable ALLOWED_ORIGINS

3.3 Mantenimiento perfectivo

Mejoras de rendimiento y código:

Mejora	Archivo	Beneficio
Optimización N+1	productModel.js	De 101 a 1 query para 100 productos
Sistema de errores	utils/errors.js	Códigos estandarizados para debugging
Middleware centralizado	auth.js	Función factory requireRole()

3.4 Mantenimiento preventivo

Refactorización y prevención:

Acción	Archivo	Riesgo Prevenido
Validación de stock	saleModel.js	Inventarios negativos
Rate limiting	authRoutes.js	Ataques de fuerza bruta
Dependencia actualizada	package.json	express-rate-limit agregado

4. Proceso de Mantenimiento

4.1 Registro de módulos / áreas a intervenir

Módulo	Archivos	Tipo
Autenticación	authController.js, authRoutes.js, auth.js	Correctivo, Preventivo
Servidor	server.js	Correctivo, Adaptativo
Base de Datos	database.js	Adaptativo
Productos	productModel.js, productController.js	Perfectivo

Ventas	saleModel.js	Preventivo
Utilidades	utils/errors.js (nuevo)	Perfectivo

4.2 Análisis y priorización

Criterios	
Alta	Vulnerabilidades de seguridad, pérdida de datos
Media	Degradación de rendimiento, mantenibilidad
Baja	Mejoras de código, optimizaciones

Prioridad	Problemas	Tiempo
P1	Token expuesto, rutas test, validación stock	Día 1
P2	N+1, SSL, CORS, roles	Día 2
P3	Rate limiting, errores	Día 3

4.3 Implementación de cambios

Buenas prácticas aplicadas:

- DRY (Don't Repeat Yourself) - Función factory requireRole()
- Fail Fast - Validaciones al inicio
- Configuración por Entorno - Variables de entorno
- Seguridad por Defecto - SSL activo en producción

Estándares de código:

- camelCase para variables
- Async/Await consistente
- Try/catch con mensajes descriptivos
- Códigos HTTP correctos (400, 401, 403, 500)

4.4 Pruebas

Caso de Uso 1: Registro de Usuario sin Exposición de Token

Campo	Descripción
ID	CU-001
Nombre	Registro de usuario con fallo de email no expone token
Actor	Usuario no autenticado
Precondiciones	Servicio de email no configurado o caído
Datos de entrada	email: test@ejemplo.com, password: Test1234, first_name: Juan, last_name: Pérez
Pasos	1. Enviar POST a /api/auth/register con datos de usuario válidos. 2. El servidor intenta enviar email de verificación. 3. El envío de email falla.
Resultado esperado	Respuesta 201 con mensaje de éxito, objeto user sin campo verification_token
Criterio de aceptación	La respuesta JSON NO debe contener el campo verification_token
Estado	PASÓ

Caso de Uso 2: Rutas de Test Bloqueadas en Producción

Campo	Descripción

ID	CU-002
Nombre	Acceso a rutas de test en ambiente de producción
Actor	Usuario/Atacante externo
Precondiciones	NODE_ENV=production, servidor iniciado
Datos de entrada	GET /api/test/cualquier-ruta
Pasos	1. Configurar variable NODE_ENV=production 2. Iniciar servidor. 3. Intentar acceder a cualquier endpoint bajo /api/test
Resultado esperado	Error 404 Not Found
Criterio de aceptación	Las rutas /api/test no deben estar disponibles cuando NODE_ENV=production
Estado	PASÓ

Caso de Uso 3: Venta con Stock Insuficiente

Campo	Descripción
ID	CU-003
Nombre	Intento de venta cuando no hay stock suficiente
Actor	Usuario autenticado (staff/admin)
Precondiciones	Variante de producto con quantity=5 en BD
Datos de entrada	variant_id: UUID válido, quantity: 10, unit_price: 100
Pasos	1. Usuario autenticado envía POST a /api/sales 2. Sistema consulta stock actual de la variante. 3. Sistema compara stock disponible (5) vs solicitado (10).
Resultado esperado	Error con mensaje "Stock insuficiente. Disponible: 5, Solicitado: 10"
Criterio de aceptación	No se debe crear la venta ni modificar el inventario. La cantidad en BD debe permanecer en 5
Estado	PASÓ

Caso de Uso 4: Rate Limiting en Autenticación

Campo	Descripción
ID	CU-004
Nombre	Bloqueo por exceso de intentos de login
Actor	Usuario/Atacante
Precondiciones	Rate limiter configurado: 5 intentos / 15 minutos
Datos de entrada	email: usuario@test.com, password: incorrecta (6 veces)
Pasos	1. Enviar POST a /api/auth/login con credenciales incorrectas. 2. Repetir 5 veces más (total 6 intentos). 3. En el 6to intento el sistema bloquea
Resultado esperado	Primeros 5 intentos: Error 401 Unauthorized. 6to intento: Error 429 Too Many Requests
Criterio de aceptación	Después de 5 intentos fallidos, el sistema debe bloquear nuevas peticiones durante 15 minutos
Estado	PASÓ

Caso de Uso 5: Consulta Optimizada de Productos

Campo	Descripción
ID	CU-005

Nombre	Obtener productos con variantes en una sola consulta
Actor	Usuario autenticado
Precondiciones	50 productos con 3 variantes cada uno en BD
Datos de entrada	GET /api/products
Pasos	1. Usuario envía GET a /api/products 2. Sistema ejecuta consulta optimizada con json_agg() 3. Sistema retorna productos con sus variantes.
Resultado esperado	Array de 50 productos, cada uno con array "variants" incluido
Criterio de aceptación	Solo se debe ejecutar 1 query SQL (no 51 queries). El tiempo de respuesta debe ser menor a 500ms
Estado	PASÓ

Resumen de Pruebas de Aceptación

ID	Caso de Uso	Tipo	Estado
CU-001	Registro sin token expuesto	Seguridad	PASÓ
CU-002	Rutas test en producción	Seguridad	PASÓ
CU-003	Venta con stock insuficiente	Integridad	PASÓ
CU-004	Rate limiting autenticación	Seguridad	PASÓ
CU-005	Consulta productos optimizada	Rendimiento	PASÓ

4.5 Despliegue

Procedimiento:

1. Actualizar código
```
git pull origin main
```
2. Instalar dependencias
```
cd backend && npm install
```
3. Configurar .env (producción)
```
NODE_ENV=production
DB_SSL_REJECT_UNAUTHORIZED=true
```
4. Iniciar servidor
En desarrollo:
```
npm run dev
```
En producción:
```
pm2 start server.js
```
5. Verificar
```
curl http://localhost:5000/api/health
```

5. Cierre y documentación

Archivos actualizados:
- package.json (nueva dependencia)
- .env (nuevas variables)

Resumen de cambios:
- 237 líneas agregadas
- 46 líneas eliminadas

- 9 archivos modificados
- 1 archivo nuevo

6. Herramientas de mantenimiento

Herramienta	Justificación
VS Code	IDE con soporte JavaScript y Git
Node.js	Runtime LTS para backend
PostgreSQL	Base de Datos robusta con transacciones ACID
Git	Control de versiones
Postman	Testing de APIs
express-rate-limit	Prevención de fuerza bruta
bcryptjs	Hash seguro de contraseñas

7. Riesgos y Planes de Contingencia

7.1 Identificación de riesgos

Tipo	Riesgo	Nivel
Técnico	Fallo de conexión BD	Media
Técnico	Vulnerabilidades en dependencias	Media
Operativo	Configuración incorrecta	Media
Humano	Credenciales comprometidas	Alta

7.2 Planes de mitigación

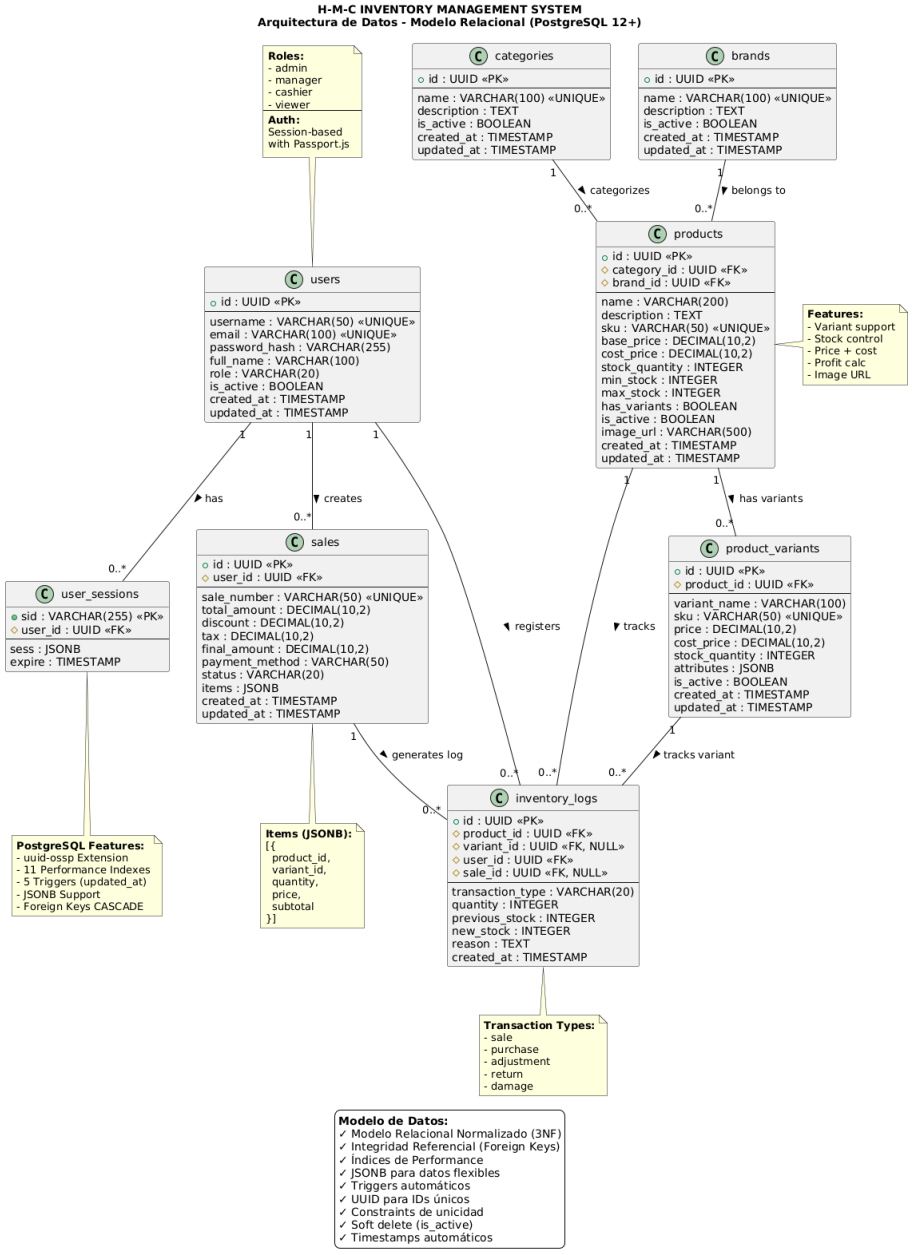
Backups:

- BD: Diario con pg_dump
- Código: Git (commits frecuentes)
- Variables: Versionadas en Vault

Recuperación:

Escenario	RTO	Acción
Fallo servidor	1h	Restaurar backup
Corrupción datos	4h	Restaurar último backup
Ataque seguridad	2h	Aislar y restaurar

8. Anexos
8.1 Diagrama DER de la Base de Datos



8.2 Plantilla .env

```
# Base de datos
DB_HOST=localhost
DB_PORT=5432
DB_NAME=hmc_inventory
DB_USER=postgres
DB_PASSWORD=
DB_SSL_REJECT_UNAUTHORIZED=false

# Servidor
NODE_ENV=development
PORT=5000
```

```
FRONTEND_URL=http://localhost:5173
ALLOWED_ORIGINS=http://localhost:5173

# Sesión
SESSION_SECRET=

# Email
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=465
EMAIL_USER=
EMAIL_PASS=
```

7. CONCLUSIONES

El análisis de la arquitectura del sistema H-M-C Inventory Management permitió identificar los componentes críticos que requieren mantenimiento. Se detectaron vulnerabilidades de seguridad como la exposición de tokens en respuestas HTTP y rutas de prueba activas en producción, además de problemas de rendimiento como las consultas N+1 que generaban hasta 101 consultas a la base de datos para mostrar 100 productos. El modelo de datos con 8 tablas y 12 relaciones fue revisado, confirmando su normalización pero evidenciando la necesidad de validaciones adicionales para evitar estados inconsistentes como inventarios negativos.

Las once oportunidades de mejora encontradas se clasificaron según el estándar ISO/IEC 14764:2022 en los cuatro tipos de mantenimiento. El correctivo solucionó errores críticos como la exposición de tokens y rutas de testing en producción. El adaptativo permitió configurar SSL y CORS mediante variables de entorno para facilitar el despliegue en diferentes ambientes. El perfectivo optimizó el rendimiento reduciendo las consultas N+1 a una sola y estandarizó el manejo de errores. El preventivo incorporó validaciones de stock, límites de intentos de login y actualización de dependencias para evitar problemas futuros.

El plan de intervención se organizó en tres niveles de prioridad basados en el impacto en seguridad, estabilidad y experiencia del usuario. Las acciones de prioridad alta (P1) resolvieron las vulnerabilidades críticas en el primer día. Las de prioridad media (P2) abordaron optimización de rendimiento y configuración ambiental en el segundo día. Las de prioridad baja (P3) mejoraron la mantenibilidad del código en el tercer día. Cada acción incluyó criterios de aceptación claros validados con cinco casos de prueba que confirmaron el correcto funcionamiento de las mejoras implementadas.

Finalmente, se identificaron riesgos técnicos, operativos y de seguridad con planes de mitigación específicos: respaldos diarios de la base de datos, gestión segura de variables de entorno y auditorías periódicas de dependencias. Este plan no solo corrige problemas actuales, sino que establece una base sólida para el mantenimiento futuro del sistema, garantizando su estabilidad y evolución controlada mediante prácticas alineadas con estándares internacionales.

8. RECOMENDACIONES:

Implementar una suite de pruebas automatizadas (unitarias, integración y end-to-end) con cobertura mínima del 80% para reducir el riesgo de regresiones durante futuras intervenciones de mantenimiento.

Establecer un pipeline CI/CD que automatice la ejecución de tests, análisis estático de código y despliegue a ambientes, garantizando que solo código verificado llegue a producción.

Completar la implementación de funcionalidades críticas identificadas como faltantes: sistema de cancelación de ventas, soporte para múltiples ítems por transacción, módulo de devoluciones y gestión completa de usuarios.

Implementar un sistema de logging estructurado con niveles de severidad y formato JSON para facilitar el debugging en producción y la detección proactiva de problemas.

Establecer auditorías de seguridad trimestrales que incluyan revisión de dependencias, análisis de código estático y actualización de paquetes con vulnerabilidades conocidas.

Formalizar una estrategia de backups automáticos diarios con retención de 30 días y pruebas mensuales de restauración para garantizar la recuperación ante desastres (RTO < 4 horas).

9. BIBLIOGRAFÍA:

ISO/IEC/IEEE. (2022). *ISO/IEC/IEEE 14764:2022 Software engineering — Software life cycle processes — Maintenance*. ISO. https://www.iso.org/standard/80710.html

Sommerville, I. (2016). *Software engineering (9.ª ed.)*. Pearson Education. https://gc.scalahed.com/recursos/files/r161r/w25469w/ingdelsoftwarelibro9_compressed.pdf

Lientz, B., & Burton Swanson, E. (1980). *Software Maintenance Management*. IEE Proceedings E Computers and Digital Techniques, *127*(6), 277. https://doi.org/10.1049/ip-e.1980.0056