UNIVERSIDAD SANTO TOMAS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
T U N J A

TÍTULO PASANTÍA

Módulos de sistemas web para la gestión de Procesos en el Hospital Universitario San Rafael de Tunja

PROPONENTE(S)
Daniel Estiven Ramirez Muñoz
CC1002395347

DIRECTOR
DIRECTOR: ing. Jhon Alexis Piracoca Arcos

Tunja
Fecha de presentación (10, 05, 2026)

Tabla de contenido

1. FICHA TÉCNICA DE LA PASANTÍA

Título	Módulos de sistemas web para la gestión de Procesos en el Hospital Universitario San Rafael de Tunja
Nombre Estudiante	Daniel Estiven Ramirez Muñoz
Documento estudiante	CC 1002395347
Correo electrónico estudiante	daniel.ramirezm@usantoto.edu.co
Director	Jhon Alexis Piracoca Arcos
Entidad o sector donde realizará la pasantía	Hospital Universitario San Rafael de Tunja
Lugar de ejecución de la pasantía	Tunja, Boyaca
Duración	620 horas
Los abajo firmantes confirman que todos los datos incluidos en la presente propuesta son correctos y verídicos, que no incumplen ninguna ley o norma vigente (incluir nombres y firmas de estudiantes y director).	

Firma del autor

Daniel Estiven Ramirez Muñoz
CC 1002395347

Firma del director

---
ing. Jhon Alexis Piracoca Arcos

2. ANTECEDENTES DE LA EMPRESA

El Hospital Universitario San Rafael de Tunja es una institución de salud pública de alta complejidad fundada en 1956, que se ha consolidado como el principal referente hospitalario de la región centro-oriental de Colombia. Ofrece servicios en las áreas de medicina interna, cirugía, pediatría, ginecología y obstetricia, urgencias, cuidados intensivos, oncología y salud mental, entre otros. Cuenta con una capacidad instalada aproximada de 350 camas y más de 1.200 empleados entre personal asistencial, administrativo y de apoyo.

Además de su función asistencial, el hospital cumple un rol académico como centro de formación universitaria, siendo receptor de estudiantes en prácticas de diversas disciplinas de la salud. En los últimos años, la creciente demanda de servicios y la incorporación progresiva de tecnologías de información han incrementado la dependencia de sistemas informáticos para soportar procesos clínicos, administrativos y académicos.

El área de Tecnologías de la Información (TI) del hospital ocupa un lugar transversal dentro de la estructura organizacional, siendo responsable del soporte técnico, la administración de la infraestructura tecnológica, la gestión de activos y la continuidad de los sistemas de información que soportan los procesos institucionales. Esta área cuenta con un equipo técnico que atiende las necesidades tecnológicas de todas las dependencias de la institución.

3. PLANTEAMIENTO DEL PROBLEMA

El Hospital Universitario San Rafael de Tunja enfrenta actualmente dificultades en la manera como gestiona las solicitudes de soporte técnico relacionadas con su infraestructura tecnológica. La ausencia de un sistema formal para administrar estas incidencias limita la capacidad del área de Tecnologías de la Información (TI) para responder de manera eficiente y organizada a las necesidades de los usuarios internos. En la práctica, los requerimientos de soporte se comunican por medios informales como llamadas telefónicas, mensajes instantáneos o conversaciones directas, lo que impide contar con un registro único y confiable. Esta dispersión de canales genera falta de control sobre las solicitudes, dificulta el seguimiento de los casos y ocasiona que algunas incidencias no sean atendidas oportunamente.

Otro aspecto crítico es la ausencia de un esquema de clasificación y priorización. Al no contar con criterios claros que permitan diferenciar entre problemas urgentes y situaciones de menor impacto, se corre el riesgo de destinar recursos a tareas poco relevantes mientras se retrasan soluciones que afectan procesos esenciales del hospital.

La carencia de un sistema de documentación también provoca pérdida de información valiosa. No se dispone de un historial que permita identificar patrones de fallas recurrentes ni de una base de conocimiento que facilite reutilizar soluciones previamente aplicadas. Esto genera duplicación de esfuerzos y limita la capacidad de aprendizaje organizacional del área de TI.

También, la gestión de los activos tecnológicos presenta debilidades. El inventario de equipos, licencias y dispositivos no se encuentra actualizado, lo que dificulta la planificación de mantenimientos, actualizaciones y reposiciones. A esto se suma la falta de indicadores de desempeño, como tiempos de respuesta o carga de trabajo del personal técnico, que son fundamentales para evaluar la eficiencia del servicio y orientar la toma de decisiones estratégicas.

En conjunto, estas limitaciones repercuten en la calidad del servicio tecnológico que recibe el personal del hospital, generan sobrecarga en el equipo de soporte y reducen la capacidad de planificación del área de TI. Dado que la infraestructura tecnológica es un componente transversal en los procesos asistenciales, administrativos y académicos, esta problemática representa un riesgo para la continuidad de los servicios hospitalarios y, en última instancia, para la atención adecuada de los pacientes.

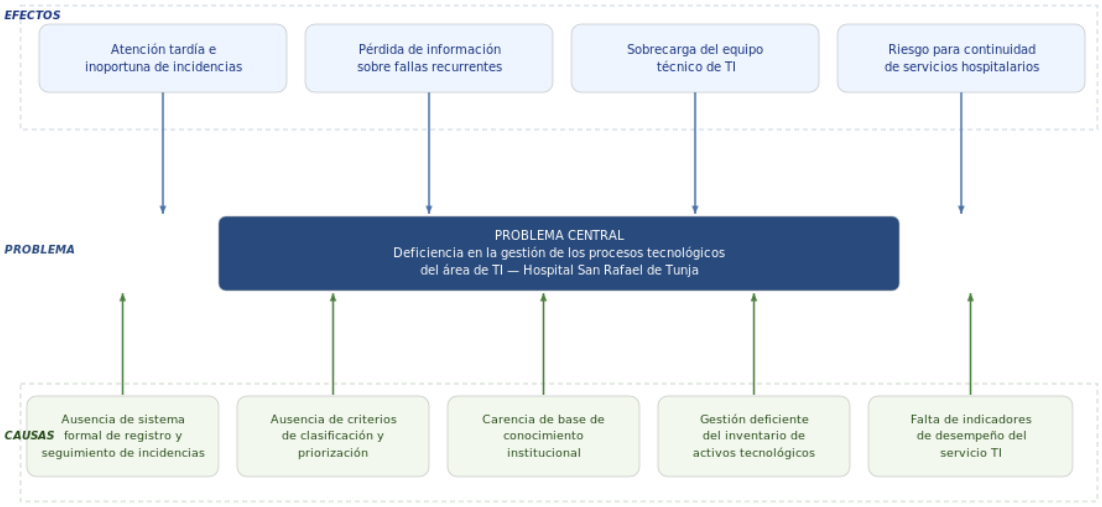


Figura. Árbol de problemas: causas y efectos de la problemática identificada en el área de TI del HUSRT.

4. JUSTIFICACIÓN

La gestión tecnológica en instituciones hospitalarias requiere soluciones integrales que permitan atender con eficiencia las necesidades operativas, garantizar la continuidad de los servicios y facilitar la toma de decisiones estratégicas. En el caso del Hospital Universitario San Rafael de Tunja, se ha identificado una carencia significativa en la forma como se administran las solicitudes de soporte técnico, el inventario de activos TI y el conocimiento técnico generado por el equipo de soporte.

Ante esta situación, el desarrollo de un sistema web para la gestión de procesos del área de TI representa una respuesta concreta y pertinente. Este sistema permitirá centralizar la gestión de solicitudes de soporte, controlar el inventario tecnológico y estructurar una base de conocimiento digital, dentro de una plataforma segura, escalable y accesible desde múltiples dispositivos.

En términos de impacto medible, se proyecta una reducción del tiempo promedio de respuesta a incidencias de 48 horas a menos de 12 horas. Este estimado se obtuvo a partir del análisis de los registros informales de atención llevados por el equipo técnico del área de TI durante el periodo de diagnóstico, en los que se documentó que la mayor parte del tiempo transcurría en la búsqueda del responsable, la asignación manual de la tarea y el seguimiento por canales no centralizados. Con la implementación del sistema, las solicitudes serán registradas, clasificadas y asignadas automáticamente, eliminando esos tiempos muertos y permitiendo al técnico enfocarse directamente en la resolución; de ahí la confianza en la reducción proyectada. Asimismo, se espera alcanzar un registro formal del 100% de las solicitudes de soporte, frente al estimado actual del 65%. Este porcentaje refleja la proporción de solicitudes que el equipo de TI lograba dejar consignadas por escrito en un día hábil típico, usando hojas de cálculo y correos electrónicos como únicos medios de seguimiento. El 35% restante correspondía a solicitudes atendidas verbalmente o por mensajería instantánea que nunca quedaban documentadas, impidiendo su seguimiento posterior y generando pérdida de trazabilidad. Con el sistema, toda solicitud deberá ingresarse obligatoriamente antes de ser atendida, garantizando el registro completo. Finalmente, la implementación de una base de conocimiento reutilizable permitirá reducir en un 30% la recurrencia de incidencias no resueltas formalmente.

Es importante señalar que los módulos desarrollados en esta pasantía hacen parte de un proyecto institucional liderado por el área de TI del Hospital Universitario San Rafael de Tunja. En ese marco, el aporte concreto del pasante consiste en el diseño, desarrollo e implementación de los módulos de soporte, incidencias, inventario de activos y base de conocimiento, los cuales

se integran a la plataforma existente y amplían sus capacidades de gestión tecnológica institucional.

Desde el punto de vista académico, el proyecto permite aplicar conocimientos en arquitectura de software, diseño de interfaces, seguridad informática, modelado de sistemas y desarrollo web, integrando teoría y práctica en un entorno real. En síntesis, esta pasantía responde a una necesidad institucional concreta, propone una solución tecnológica viable y genera un impacto positivo en la eficiencia operativa y la sostenibilidad de la infraestructura tecnológica del hospital.

5. OBJETIVOS

5.1. Objetivo General

Construir los módulos de soporte, incidencias, inventario de activos y base de conocimiento, mediante la metodología ágil SCRUM, para ampliar el sistema web del Hospital Universitario San Rafael de Tunja, con el fin de optimizar la operación y continuidad de los servicios tecnológicos del área de TI.

5.2. Objetivos Específicos

1. Especificar en un documento SRS los requerimientos funcionales y no funcionales del sistema, a partir del levantamiento de información mediante entrevistas, observación directa y análisis documental con el área de TI del hospital, con el fin de establecer el alcance funcional del desarrollo.

2. Desarrollar los módulos de soporte, incidencias, inventario y base de conocimiento del sistema web bajo la arquitectura MVC, mediante Angular en el frontend, Node.js en el backend y MariaDB como gestor de datos, con el fin de proveer una solución escalable y mantenible para la gestión tecnológica del área de TI.

3. Validar el funcionamiento del sistema mediante pruebas funcionales, de integración y de aceptación con el área de TI del hospital, con el fin de verificar el cumplimiento de los requerimientos especificados, documentado en un informe formal de pruebas.

4. Elaborar la documentación técnica del sistema conforme a estándares de ingeniería de software, incluyendo manual de uso por perfil de usuario, con el fin de garantizar la transferencia del conocimiento a la institución para su operación y mantenimiento.

6. DESCRIPCIÓN DE LAS ACTIVIDADES REALIZADAS

6.1. Listado de actividades realizadas en el proyecto

A continuación, se presenta el conjunto de actividades realizadas durante la pasantía, estableciendo las fechas aproximadas de inicio y entrega de cada producto. Estas actividades corresponden al desarrollo del Módulo de Gestión de Repuestos y el Módulo de Indicadores del dentro del aplicativo hospitalario, desarrollado para el Hospital Universitario San Rafael de Tunja.

	Descripción de la Actividad	Fecha Inicio	Fecha Entrega	Producto Entregado
1	Análisis de requerimientos del módulo de Repuestos: levantamiento de información sobre los tipos de repuestos utilizados en el área de Gestión de Sistemas, campos necesarios y flujos de trabajo.	enero 16 del 2026	Febrero 2 del 2026	Análisis funcional del módulo de Repuestos
2	Diseño del modelo de base de datos para Repuestos y Tipos de Repuesto (tablas SysRepuesto, SysTipoRepuesto, SysMovimientosStockRepuestos, SysAuditoriaRepuesto).	Febrero 2 del 2026	Febrero 10 del 2026	Diagrama entidad-relación y scripts SQL de creación de tablas
3	Desarrollo del backend (Node.js/Sequelize): modelos ORM, controladores CRUD y rutas REST para el catálogo de repuestos y sus tipos.	Febrero 10 del 2026	Febrero 28 del 2026	Archivos: SysRepuesto.js, SysTipoRepuesto.js, sysRepuestoController.js, sysTipoRepuestoController.js, sysRepuestoRoutes.js, sysTipoRepuestoRoutes.js
4	Implementación de la lógica de control de stock: descuento automático de inventario al registrar mantenimientos correctivos/preventivos, y ajuste de stock en edición de reportes.	Febrero 28 del 2026	Marzo 16 del 2026	Funciones descontarStock y ajustarStockEdicion en el controlador backend. Modelo SysMovimientosStockRepuestos.js

	Descripción de la Actividad	Fecha Inicio	Fecha Entrega	Producto Entregado
5	Desarrollo del sistema de auditoría de repuestos: registro automático de cada creación, edición, activación, inactivación y descuento de stock con usuario y timestamp.	Marzo 16 del 2026	Marzo 23 del 2026	Modelo SysAuditoriaRepuesto.js, función helper registrarAuditoria integrada en controladores
6	Desarrollo del frontend (Angular): componente de Gestión de Repuestos con pestañas de repuestos activos, tipos, dados de baja, movimientos de stock y registro de cambios.	Marzo 23 del 2026	Abril 20 del 2026	Componente Angular repuestos.component (HTML + TS + CSS)
7	Implementación de alertas de stock mínimo: notificación visual automática cuando un repuesto cae por debajo del umbral configurado, con badge en pestaña e indicador en tabla.	Abril 20 del 2026	Abril 24 2026	Funcionalidad de alerta integrada en repuestos.component
8	Implementación de la función de exportación de datos a PDF y CSV para el listado de repuestos y el historial de movimientos de stock.	Abril 24 del 2026	Mayo 1 del 2026	Servicio pdf-generator.service.ts, métodos de exportación CSV en el componente
9	Implementación y desarrollo del modulo de indicadores	Mayo 1 del 2026	Mayo 9 del 2026	Funcionalidades de indicadores propuestas por el cliente.

6.2. Herramientas utilizadas en el desarrollo del proyecto

Herramienta Utilizada	Versión	Para qué la utilizó
Angular	17+	Framework principal para el desarrollo del frontend. Se usó para construir los componentes de Gestión de Repuestos e Indicadores de Sistemas con arquitectura de componentes standalone.
TypeScript	5.x	Lenguaje de programación tipado utilizado en el frontend Angular. Permitió definir interfaces, tipos y señales reactivas para el manejo del estado.
Node.js	18+	Entorno de ejecución del servidor backend. Usado para alojar la API REST que gestiona la lógica de negocio de los módulos de repuestos.
Express.js	4.22+	Framework web de Node.js utilizado para definir las rutas y controladores de la API REST del módulo de repuestos.
Sequelize ORM	6.37+	ORM para Node.js utilizado para mapear los modelos de base de datos y ejecutar consultas SQL de forma segura.
MariaDB	10.x	Sistema de gestión de base de datos relacional donde se almacenan todos los datos del sistema HRCATCH 2.0.
PrimeNG	17+	Biblioteca de componentes UI para Angular. Se utilizó para gráficos del dashboard de indicadores y elementos visuales.

6.3. Desarrollo detallado de las actividades

6.3.1 Módulo de Gestión de Repuestos

Contexto y justificación: El área de Gestión de Sistemas requería un sistema para gestionar el inventario de repuestos utilizados en el mantenimiento de equipos. Anteriormente, este control se llevaba de forma manual. Se desarrolló desde cero el Módulo de Gestión de Repuestos integrado al sistema HRCATCH 2.0, cumpliendo con los siguientes objetivos funcionales:

- Mantener un catálogo centralizado de repuestos con toda su información técnica.
- Controlar el stock disponible con alertas automáticas de stock mínimo.
- Registrar todos los movimientos de inventario (ingresos y egresos) con trazabilidad completa.
- Integrar el descuento automático de stock al momento de registrar mantenimientos.
- Proporcionar un sistema de auditoría que registre cada acción realizada.

Diseño de la base de datos: Se diseñaron e implementaron cuatro tablas relacionadas en MariaDB mediante modelos Sequelize ORM: SysRepuesto, SysTipoRepuesto, SysMovimientosStockRepuestos y SysAuditoriaRepuesto.

Desarrollo del backend (API REST): Se implementó una API REST completa en Node.js con Express.js y Sequelize, con endpoints principales para el manejo de Repuestos y Tipos de Repuesto, incluyendo rutas críticas como /descontar-stock y /ajustar-stock-edicion.

Desarrollo del frontend (Angular): El componente frontend repuestos.component fue desarrollado en Angular como componente standalone con pestañas para Repuestos, Tipos, Dados de Baja, Movimientos de Stock y Registro de Cambios.

6.3.2 Módulo de Indicadores de Sistemas

Contexto y justificación: El área necesitaba una herramienta de visualización gerencial que permitiera monitorear el desempeño del mantenimiento de equipos a través de indicadores clave de desempeño (KPIs) en tiempo real.
Diseño del dashboard: El componente sysindicadores.component fue desarrollado en Angular con PrimeNG Chart. Implementa señales reactivas de Angular 17 para calcular y actualizar automáticamente todos los indicadores. Los KPIs incluyen: KPI Preventivo, KPI Correctivo, Duración Promedio, Actividad por Responsable, Distribución por Tipo, Top Servicios, entre otros.

6.3.3 Manual de Usuario — Módulo de Gestión de Repuestos

- Acceso: Iniciar sesión en HRCATCH 2.0, navegar a Sistemas y seleccionar Repuestos.
- Gestión: La pestaña principal muestra el stock activo. Los administradores pueden crear repuestos nuevos y configurar umbrales de alerta.
- Stock: El botón "Stock" permite registrar ingresos y egresos, requiriendo justificación para la trazabilidad de auditoría.

6.3.4 Manual de Usuario — Módulo de Indicadores de Sistemas

- Filtros: El usuario puede aplicar filtros por Año y rango de meses. El sistema recalcula las 10 gráficas del dashboard dinámicamente.
- Lectura: Los medidores superiores resumen el cumplimiento global, mientras que las gráficas de barras desglosan la información por servicio, técnico y equipo.

6.4. Aplicación de la Metodología SCRUM

El desarrollo del módulo de gestión de repuestos se llevó a cabo mediante la metodología ágil SCRUM, adaptándola al contexto de la pasantía en una organización hospitalaria donde los requerimientos surgieron progresivamente durante las primeras semanas de levantamiento de información. A continuación se describen los cuatro sprints ejecutados durante las catorce semanas de pasantía.

6.4.1. Sprint 0 — Levantamiento de requerimientos (semanas 1-2)

Objetivo: entender el flujo actual de gestión de repuestos, identificar los actores del sistema y construir el backlog inicial.

Actividades: Se realizaron dos entrevistas formales con el tutor empresarial y con los técnicos del área de TI. Se llevó a cabo observación directa del proceso manual en hojas de cálculo. Se analizó la base de datos existente del proyecto institucional para identificar convenciones de nomenclatura y relaciones con otras tablas. Como resultado se definió la arquitectura de cuatro tablas: SysRepuesto, SysTipoRepuesto, SysMovimientosStockRepuestos y SysAuditoriaRepuesto.

Entregable: Documento de levantamiento de requerimientos con backlog priorizado de 11 ítems (7 funcionales y 4 no funcionales), validado con el tutor empresarial al cierre de la semana 2.

Resultado: Product Backlog creado con historias de usuario priorizadas por valor de negocio y riesgo técnico. La historia US-01 (Catálogo de repuestos con baja lógica) quedó en la cima del backlog por su impacto directo en la integridad de datos.

6.4.2. Sprint 1 — Diseño de base de datos y backend del catálogo (semanas 3-6)

Objetivo: diseñar e implementar la capa de datos y los endpoints REST del catálogo de repuestos y tipos de repuesto.

Actividades: Modelado del esquema entidad-relación con cuatro entidades. Implementación de los modelos Sequelize (SysRepuesto.js y SysTipoRepuesto.js) con sus asociaciones (hasMany / belongsTo). Desarrollo de los controladores REST para CRUD completo de ambas entidades. Implementación de la regla de baja lógica: verificación de stock y movimientos antes de permitir la inactivación. Configuración de rutas bajo /api/sistemas/ con middleware de autenticación JWT heredado del proyecto institucional.

Criterios de aceptación: API REST para catálogo funcional (GET, POST, PUT) con códigos HTTP correctos (200, 201, 409). Regla de baja lógica activa: rechazo de inactivar repuesto con stock mayor a cero. Todos los endpoints protegidos con JWT.

Entregable: Backend del módulo catálogo funcional y probado con Postman. Modelo ER documentado. Retroalimentación del tutor: aprobado con observación de añadir campo número de serie al modelo, incorporado antes del cierre del sprint.

6.4.3. Sprint 2 — Módulo de movimientos de stock y auditoría (semanas 7-10)

Objetivo: implementar el núcleo transaccional del sistema: ingresos y egresos de stock con trazabilidad, adjunto de facturas PDF y registro de auditoría automática.

Actividades: Implementación del modelo SysMovimientosStockRepuestos.js con lógica transaccional (Sequelize.transaction) para actualizar atómicamente el stock en SysRepuesto. Integración de Multer en el endpoint de ingreso para validación de tipo MIME y almacenamiento de PDF con nombre único generado por timestamp. Implementación de SysAuditoriaRepuesto.js para registro automático de cada operación sensible. Desarrollo del controlador con validación de stock disponible antes de cada egreso. Integración con el sistema de roles para restringir operaciones según perfil de usuario.

Criterios de aceptación: Un egreso que supere el stock disponible retorna HTTP 400 con mensaje descriptivo. Un ingreso sin adjunto PDF obliga a ingresar una justificación textual. Cada movimiento genera automáticamente un registro en SysAuditoriaRepuesto con usuario, fecha y descripción de la operación.

Entregable: Backend completo del subsistema de movimientos. Retroalimentación del tutor: solicitud de agregar descarga de la factura PDF desde el historial de movimientos, incorporada como subtarea en Sprint 3.

6.4.4. Sprint 3 — Frontend Angular, pruebas y documentación (semanas 11-14)

Objetivo: construir la interfaz de usuario completa en Angular 19, ejecutar el ciclo de pruebas y redactar la documentación técnica final.

Actividades (frontend): Desarrollo del Standalone Component repuestos.component en Angular 19. Integración de PrimeNG (p-table, p-dialog, p-badge, p-message) para listado paginado, filtrado y formularios. Template-Driven Forms con validaciones de campo obligatorio y valores positivos. Servicios Angular con HttpClient y cabecera JWT automática. Integración de SweetAlert2 para confirmaciones de operaciones críticas. File API con validación de tipo MIME previa al envío. Componente de descarga de PDF desde historial de movimientos.

Actividades (pruebas y documentación): Pruebas funcionales de cada caso de uso del SRS. Pruebas de integración frontend-backend verificando consistencia de datos. Prueba de límites de stock y carga de archivos. Sesión de aceptación con el área de TI. Correcciones menores al diseño visual. Redacción del Manual Técnico, Manual del Administrador, Manual por Perfil de Usuario y presente informe de pasantía.

Entregable: Sistema funcional integrado y aprobado en sesión de aceptación. Documentación técnica completa entregada al área de TI del hospital.

6.5. Dificultades encontradas y lecciones aprendidas

Integración en un proyecto preexistente: La mayor dificultad inicial fue entender la arquitectura y convenciones de un proyecto en desarrollo activo. Fue necesario estudiar el código fuente de los módulos previos para respetar la nomenclatura de rutas, el esquema de autenticación JWT y los patrones de Sequelize ya establecidos. Esta restricción fue un aprendizaje valioso sobre el trabajo colaborativo en equipos de desarrollo reales.

Manejo transaccional de archivos y base de datos: Garantizar la consistencia entre el archivo PDF almacenado en disco y el registro en base de datos fue un reto técnico. Si la transacción fallaba después de que Multer guardaba el archivo, éste quedaba huérfano. La solución fue mover el archivo al directorio final solo tras confirmar el commit, eliminando el temporal en caso de rollback.

Validaciones en tiempo real con Angular: Implementar la validación de stock disponible directamente en el formulario Angular requirió una consulta previa al backend para obtener el stock actual antes de habilitar el campo de cantidad de egreso. Esto eliminó envíos de formulario con errores predecibles, mejorando notablemente la experiencia de usuario.

Adaptación al contexto hospitalario: El entorno hospitalario impone restricciones particulares: los cambios en el sistema deben coordinarse con el área de TI para evitar interferir con procesos críticos del hospital. Las pruebas de integración con el servidor real se realizaron en horarios de baja carga. Esta restricción reafirmó la importancia de un entorno de desarrollo local espejado con el de producción.

7. CRONOGRAMA DE ACTIVIDADES

Tabla 1. Cronograma de actividades

Meses	Mes 1				Mes 2				Mes 3				Mes 4				Mes 5				Mes 6				RESPONSABLE
Semanas	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	
Levantamiento de requisitos y análisis	x	x																							Pasante/Tutor
Diseño de arquitectura y base de datos		x	x	x																					Pasante/Director
Desarrollo Backend (APIs)					x	x	x	x																	Pasante
Desarrollo Frontend - Autenticación y Tickets							x	x	x	x	x	x													Pasante
Desarrollo Frontend - Dashboard e Inventario										x	x	x	x	x	x	x									Pasante
Desarrollo Base de Conocimiento										x	x	x	x												Pasante
Pruebas, correcciones y despliegue																x	x	x	x	x					Pasante/Tutor

8. CONCLUSIONES

Durante el desarrollo de la pasantía empresarial en el Hospital Universitario San Rafael de Tunja, se logró implementar satisfactoriamente los dos módulos asignados dentro del área de Gestión de Sistemas del aplicativo HRCATCH 2.0: el Módulo de Gestión de Repuestos y el Módulo de Indicadores de Sistemas. Ambos módulos fueron entregados funcionales e integrados.

En cuanto al cumplimiento de los objetivos, el módulo de Gestión de Repuestos permitió centralizar el inventario, automatizar el control del stock, generar alertas y garantizar la trazabilidad. El módulo de Indicadores proporcionó un dashboard interactivo con KPIs visuales que permiten monitorear en tiempo real el cumplimiento de mantenimientos.

Entre los principales retos enfrentados se destacan: la comprensión de una arquitectura preexistente, la gestión de la consistencia del inventario en escenarios de edición de reportes, y la integración del sistema de autenticación JWT. La curva de aprendizaje con Angular 17 y PrimeNG fue superada con éxito.

El impacto concreto se refleja en la eliminación del control manual del inventario, reemplazado por un sistema digital que reduce el riesgo de pérdida de información. El dashboard transformó la forma en que el área analiza su desempeño, mejorando la calidad del servicio prestado.

En relación con el Objetivo Específico 1, el proceso de levantamiento de información confirmó que la estructura de requerimientos funcionales y no funcionales plasmada en el documento SRS refleja fielmente las necesidades del área de TI. Las entrevistas y la observación directa revelaron que el problema central no era la carencia de datos, sino la dispersión de información en hojas de cálculo sin trazabilidad, lo que permitió enfocar el diseño del sistema hacia la auditoría y la inmutabilidad del histórico de movimientos.

Respecto al Objetivo Específico 2, el desarrollo de los cuatro módulos bajo la arquitectura MVC resultó en un sistema coherente, mantenible y alineado con los estándares tecnológicos del proyecto institucional. La decisión de implementar transacciones atómicas en Sequelize para los movimientos de stock fue determinante para garantizar la integridad de los datos incluso ante fallos intermedios, demostrando que las decisiones de arquitectura tienen impacto directo en la confiabilidad del sistema.

En cuanto al Objetivo Específico 3, las pruebas funcionales, de integración y de aceptación ejecutadas permitieron detectar y corregir cuatro defectos de lógica antes de la entrega final, incluyendo un caso borde en el que el egreso de la totalidad del stock disponible no era correctamente validado. La metodología de prueba por casos con criterios de aceptación explícitos demostró su valor al proporcionar trazabilidad entre requerimiento, prueba y resultado.

Sobre el Objetivo Específico 4, la documentación técnica elaborada cubre tres perfiles de usuario diferenciados, lo que garantiza que el área de TI pueda mantener y operar el sistema de forma autónoma sin depender del conocimiento tácito del desarrollador. La inclusión de tablas de contenido detalladas en los manuales como único entregable en el anexos responde al principio de confidencialidad acordado con la institución, preservando los derechos de desarrollo del hospital sobre la implementación completa.

Desde una perspectiva personal y académica, la pasantía permitió aplicar de manera integrada las competencias adquiridas durante la carrera: arquitectura de software, diseño de bases de datos relacionales, desarrollo frontend con frameworks modernos, pruebas de software y comunicación técnica. La experiencia en un entorno hospitalario añadió una dimensión valiosa: la comprensión de que los sistemas de información en el sector salud operan bajo restricciones de disponibilidad, confiabilidad y trazabilidad que van más allá de los requerimientos típicos de sistemas comerciales convencionales.

9. MARCO TEÓRICO

9.1. Sistemas de Información en el Sector Salud

Los sistemas de información hospitalarios (HIS, por sus siglas en inglés) constituyen el conjunto de componentes de hardware, software, datos y procedimientos orientados a capturar, almacenar, procesar y distribuir información que apoye las operaciones y la toma de decisiones dentro de una institución de salud (Laudon y Laudon, 2020). En el contexto de los hospitales de alta complejidad, estos sistemas abarcan subsistemas clínicos, administrativos, financieros y de gestión de recursos, todos ellos interrelacionados para garantizar la continuidad del servicio.

La gestión eficiente del inventario de repuestos y activos tecnológicos forma parte de los sistemas de gestión de activos (EAM, Enterprise Asset Management) que hoy constituyen un componente crítico para la operación de áreas de TI en hospitales. La ausencia de trazabilidad en el movimiento de repuestos genera desajustes entre el inventario físico y el registrado, impactando directamente en la capacidad de respuesta ante fallas en equipos médicos (Iadanza et al., 2019).

Arquitectura MVC y Servicios REST

El patrón de arquitectura Modelo-Vista-Controlador (MVC) separa la lógica de negocio (Modelo), la presentación al usuario (Vista) y el control del flujo de la aplicación (Controlador). En aplicaciones web modernas con separación de frontend y backend, este patrón evoluciona hacia una arquitectura cliente-servidor donde el frontend (Vista) consume una API REST que encapsula la lógica de negocio (Modelo + Controlador) (Gamma et al., 1994). Esta separación facilita el mantenimiento independiente de cada capa y habilita la reutilización de la API por múltiples clientes.

Las APIs REST (Representational State Transfer) son el estándar actual para la comunicación entre cliente y servidor en aplicaciones web. Definidas por Fielding (2000), utilizan los verbos HTTP (GET, POST, PUT, DELETE) para operar sobre recursos identificados por URLs. Los principios REST — sin estado, cacheabilidad e interfaz uniforme — garantizan interoperabilidad y escalabilidad, factores críticos en sistemas hospitalarios que deben soportar múltiples usuarios concurrentes.

9.2. Angular y el Desarrollo Frontend Moderno

Angular es un framework de desarrollo frontend mantenido por Google que utiliza TypeScript como lenguaje principal. Su arquitectura basada en componentes facilita la reutilización de código y la escalabilidad de las interfaces de usuario. A partir de Angular 14, el framework introdujo los Standalone Components, que eliminan la necesidad de declarar cada componente en un módulo NgModule, simplificando la estructura del proyecto y mejorando el rendimiento mediante lazy loading de componentes individuales (Angular, 2024).

PrimeNG es la biblioteca de componentes visuales para Angular desarrollada por PrimeTek. Provee una colección de más de 90 componentes UI listos para producción, incluyendo tablas con paginación y filtros (p-table), diálogos modales (p-dialog), indicadores de alerta (p-badge, p-message) y formularios avanzados. Su integración con Angular facilita el desarrollo de interfaces corporativas con estética profesional y accesibilidad integrada (PrimeTek, 2024).

9.3. Node.js, Express.js y Sequelize ORM

Node.js es un entorno de ejecución JavaScript del lado del servidor basado en el motor V8 de Chrome. Su modelo de entrada/salida no bloqueante y orientado a eventos lo hace especialmente adecuado para construir APIs con alto número de conexiones concurrentes (Node.js, 2024). Express.js es el framework web más utilizado sobre Node.js: proporciona enrutamiento, manejo de middleware y abstracción del ciclo de solicitud-respuesta HTTP, permitiendo desarrollar APIs REST de forma concisa y extensible.

Sequelize es un ORM (Object-Relational Mapper) para Node.js compatible con MySQL, MariaDB, PostgreSQL, SQLite y MSSQL. Permite definir modelos JavaScript que se mapean a tablas de base de datos, gestionar asociaciones entre entidades (hasMany, belongsTo) y ejecutar operaciones CRUD mediante una API JavaScript sin escribir SQL directo. Su soporte para transacciones (Sequelize.transaction) garantiza la atomicidad de operaciones que modifican múltiples tablas simultáneamente, como el registro de movimientos de stock y la actualización del inventario (Sequelize, 2024).

9.4. Metodología Ágil SCRUM

SCRUM es un marco de trabajo ágil para el desarrollo de software que organiza el trabajo en iteraciones de duración fija denominadas sprints (habitualmente de dos a cuatro semanas). Los artefactos principales de SCRUM son: el Product Backlog (lista priorizada de funcionalidades del sistema), el Sprint Backlog (conjunto de ítems seleccionados para un sprint) y el Incremento (producto funcional entregable al final de cada sprint). Los roles fundamentales son el Product Owner, el Scrum Master y el equipo de desarrollo (Schwaber y Sutherland, 2020).

En contextos de pasantía, SCRUM se adapta usualmente a equipos unipersonales o pequeños, donde el pasante asume simultáneamente los roles de desarrollador y Product Owner, con el tutor empresarial cumpliendo una función de stakeholder que valida los incrementos al cierre de cada sprint. Este modelo adaptado permite mantener la trazabilidad del avance y la incorporación de retroalimentación temprana, elementos esenciales en proyectos donde los requerimientos iniciales son parcialmente incompletos.

9.5. Seguridad: JSON Web Tokens (JWT)

JWT (JSON Web Token) es un estándar abierto (RFC 7519) que define un mecanismo compacto y autocontenido para transmitir de forma segura información entre partes como un objeto JSON firmado digitalmente. En aplicaciones web, los JWT se utilizan comúnmente para autenticación: el servidor genera un token tras la autenticación exitosa del usuario, y el cliente lo incluye en la cabecera Authorization de cada solicitud subsiguiente. El servidor puede verificar la autenticidad del token sin consultar la base de datos, haciendo el esquema escalable y sin estado (Jones et al., 2015). En el proyecto institucional donde se enmarca esta pasantía, la autenticación JWT ya estaba implementada y fue heredada por los nuevos endpoints desarrollados.

10. REFERENCIAS

Referencias

Angular. (2024). *Angular documentation*. Google LLC. https://angular.dev/docs

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* [Tesis doctoral, University of California, Irvine].

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley Professional.

Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT)* (RFC 7519). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc7519

Laudon, K. C., & Laudon, J. P. (2020). *Management information systems: Managing the digital firm* (16.a ed.). Pearson.

Node.js. (2024). *Node.js documentation*. OpenJS Foundation. https://nodejs.org/en/docs

PrimeTek. (2024). *PrimeNG: UI component library for Angular*. https://primeng.org/documentation

Schwaber, K., & Sutherland, J. (2020). *The Scrum guide: The definitive guide to Scrum: The rules of the game*. https://scrumguides.org

Sequelize. (2024). *Sequelize ORM documentation* (v6). OpenJS Foundation. https://sequelize.org/docs/v6

Iadanza, E., Gonnelli, V., Satta, F., & Gherardelli, M. (2019). Evidence-based medical equipment management: A convenient implementation. *Medical & Biological Engineering & Computing*, *57*(10), 2215–2230. https://doi.org/10.1007/s11517-019-02021-x

11. GLOSARIO

API REST (Application Programming Interface — Representational State Transfer): interfaz de comunicación entre sistemas de software que sigue los principios arquitecturales REST. Utiliza los métodos HTTP estándar (GET, POST, PUT, DELETE) para operar sobre recursos identificados por URLs. Es el mecanismo mediante el cual el frontend Angular del sistema consume los servicios del backend Node.js.

Auditoría de sistema: conjunto de registros que documentan de forma cronológica cada operación sensible realizada en el sistema, incluyendo quién la realizó, cuándo y qué datos fueron afectados. En el módulo desarrollado, los registros de auditoría se almacenan en la tabla SysAuditoriaRepuesto y no pueden modificarse una vez creados.

Baja lógica: mecanismo de eliminación "suave" de un registro de base de datos que, en lugar de borrarlo físicamente, lo marca como inactivo mediante un campo de estado. Permite preservar el historial completo de operaciones sin comprometer la integridad referencial. En el módulo de repuestos, un repuesto dado de baja lógicamente desaparece de las listas activas pero sus movimientos históricos permanecen accesibles para auditoría.

Backend: capa del sistema que opera en el servidor y contiene la lógica de negocio, el acceso a la base de datos y los servicios expuestos a través de la API. En este proyecto, el backend está desarrollado con Node.js y Express.js, y accede a la base de datos MariaDB mediante el ORM Sequelize.

Frontend: capa de presentación del sistema que se ejecuta en el navegador del usuario. Es la interfaz visual con la que interactúan los técnicos y administradores del área de TI. En este proyecto, el frontend está desarrollado en Angular 19 con Standalone Components y utiliza PrimeNG como biblioteca de componentes visuales.

HIS (Hospital Information System): sistema de información hospitalario que integra los procesos clínicos, administrativos y operativos de una institución de salud. El proyecto institucional del Hospital Universitario San Rafael de Tunja constituye un sistema de este tipo, en el que el módulo de gestión de repuestos se inscribe como componente del subsistema de TI.

Inventario (stock): cantidad de unidades disponibles de un repuesto en el almacén del área de TI en un momento dado. El stock se actualiza automáticamente con cada

movimiento de ingreso (incrementa) o egreso (decrementa). El sistema no permite egresos que superen el stock disponible.

JWT (JSON Web Token): estándar abierto (RFC 7519) para transmitir de forma segura información entre partes como un objeto JSON firmado digitalmente. Se utiliza en el sistema como mecanismo de autenticación: cada solicitud al backend incluye un token JWT en la cabecera Authorization, que el servidor verifica antes de procesar la operación.

MariaDB: sistema de gestión de bases de datos relacionales (RDBMS) de código abierto derivado de MySQL. Es compatible con el lenguaje SQL estándar y ofrece soporte para transacciones ACID, claves foráneas e índices. Es el motor de base de datos utilizado en el proyecto institucional del hospital.

Middleware: función o componente de software que intercepta las solicitudes HTTP antes de que lleguen al controlador final. En Express.js, los middlewares se usan para autenticar el JWT, validar datos de entrada, gestionar errores globales y procesar archivos adjuntos (Multer). Permiten aplicar lógica transversal a múltiples rutas sin duplicar código.

Movimiento de stock: registro de una operación de ingreso o egreso de unidades de un repuesto. Cada movimiento es inmutable una vez registrado: no puede editarse ni eliminarse, garantizando la trazabilidad del inventario. El conjunto de movimientos de un repuesto constituye su historial de uso y permite reconstruir el stock en cualquier punto del tiempo.

Multer: middleware de Node.js especializado en el manejo de datos multipart/form-data, utilizado para la subida de archivos desde formularios web. En el módulo de repuestos, Multer intercepta las solicitudes de ingreso de stock para validar el tipo MIME del archivo adjunto (solo PDF permitido), limitar el tamaño y almacenarlo con un nombre único en el servidor.

MVC (Modelo-Vista-Controlador): patrón de arquitectura de software que separa una aplicación en tres capas: el Modelo (datos y lógica de negocio), la Vista (presentación al usuario) y el Controlador (coordinación entre Modelo y Vista). En el contexto de este proyecto, el Modelo corresponde a los modelos Sequelize y la base de datos, la Vista al componente Angular, y el Controlador a los controladores Express.js.

ORM (Object-Relational Mapper): capa de abstracción que traduce entre objetos del lenguaje de programación y tablas de la base de datos relacional. Permite al desarrollador operar sobre los datos usando código orientado a objetos sin escribir SQL directamente.

Sequelize es el ORM utilizado en este proyecto para gestionar los modelos SysRepuesto, SysTipoRepuesto, SysMovimientosStockRepuestos y SysAuditoriaRepuesto.

SCRUM: marco de trabajo ágil para el desarrollo de software que organiza el trabajo en iteraciones de duración fija (sprints). Define roles (Product Owner, Scrum Master, equipo de desarrollo), artefactos (Product Backlog, Sprint Backlog, Incremento) y ceremonias (planificación del sprint, reunión diaria, revisión del sprint, retrospectiva). Fue la metodología adoptada en la pasantía para organizar las actividades de desarrollo.

Sequelize: ORM para Node.js compatible con MariaDB, MySQL, PostgreSQL, SQLite y MSSQL. Permite definir modelos, establecer asociaciones y ejecutar operaciones CRUD y transaccionales mediante JavaScript. Su API de transacciones garantiza que un conjunto de operaciones de base de datos se ejecute de forma atómica: si alguna falla, todas se revierten (rollback).

SRS (Software Requirements Specification): documento formal que describe de manera completa y precisa los requerimientos funcionales (qué debe hacer el sistema) y no funcionales (bajo qué restricciones) de un sistema de software. Sigue estándares como el IEEE 830. En este proyecto, el SRS consolida los requerimientos identificados durante el levantamiento de información y sirve como referencia para las pruebas de aceptación.

Stock mínimo: umbral configurable de cantidad de unidades de un repuesto, por debajo del cual el sistema genera una alerta visual. Es definido por el administrador al registrar o editar un repuesto. Su propósito es advertir proactivamente sobre el riesgo de desabastecimiento antes de que el stock llegue a cero, permitiendo planificar nuevas compras con antelación suficiente.

Standalone Component (Angular): tipo de componente Angular (disponible desde la versión 14, adoptado como estándar desde la versión 17) que no necesita ser declarado dentro de un NgModule. Se auto-contiene con sus propias importaciones, lo que simplifica la estructura del proyecto, mejora el tiempo de compilación y facilita el lazy loading individual de componentes. Todos los componentes del módulo de repuestos fueron desarrollados como Standalone Components.

Template-Driven Forms (Angular): enfoque de construcción de formularios en Angular en el que la lógica de validación y enlace de datos se define en la plantilla HTML mediante directivas (ngModel, required, min, max). Es adecuado para formularios de complejidad

media como los de registro y edición de repuestos. La alternativa, Reactive Forms, es preferida para formularios dinámicos de alta complejidad.

Trazabilidad: propiedad de un sistema de información que permite rastrear el origen, el historial y el destino de cada dato o acción. En el contexto del módulo de repuestos, la trazabilidad implica que para cada unidad de repuesto que sale del almacén queda un registro permanente con: el usuario que autorizó el egreso, la fecha y hora, la cantidad, la justificación y el estado del stock antes y después del movimiento.

TypeScript: superset tipado de JavaScript desarrollado por Microsoft. Añade tipado estático opcional, interfaces, enumeraciones y otras características que facilitan el desarrollo de aplicaciones a gran escala y la detección temprana de errores durante la compilación. Es el lenguaje principal de Angular y fue utilizado en todo el desarrollo del frontend de este proyecto.

12. ANEXOS

Especificación de Requerimientos

Historias de Usuario Validadas

Las siguientes historias de usuario fueron definidas, validadas y aprobadas.

HU-01: Gestión de Catálogo de Repuestos

Como Administrador del Sistema / Perfil AG, quiero registrar, editar y dar de baja repuestos en el sistema, para mantener un catálogo actualizado y evitar inconsistencias en el inventario.

Criterios de aceptación:

- El sistema debe permitir ingresar Nombre, Tipo, Número de Parte, Número de Serie, Proveedor y Stock Mínimo.
- Se debe validar que los campos obligatorios estén completos antes de guardar.
- No se puede dar de baja un repuesto si aún tiene stock disponible en el inventario.
- Debe registrarse en auditoría el usuario y fecha de la creación o edición.

HU-02: Gestión de Tipos de Repuesto

Como Administrador del Sistema, quiero crear y gestionar diferentes categorías/tipos de repuestos, para organizar el catálogo de inventario de forma jerárquica y facilitar las búsquedas.

Criterios de aceptación:

- Debe permitir definir Nombre y Descripción.
- Debe permitir activar/inactivar tipos sin perder el histórico.

HU-03: Registro de Ingreso/Egreso de Stock (Movimientos)

Como Usuario del Sistema (Técnico / Admin), quiero registrar entradas y salidas de existencias de un repuesto, para llevar un control preciso de la cantidad disponible en tiempo real.

Criterios de aceptación:

- El movimiento debe registrar la cantidad, motivo y el usuario que realiza la operación.
- En el caso de “Ingresos”, el sistema debe exigir adjuntar la factura de compra en formato PDF o indicar un motivo explícito si no hay factura.
- Se debe permitir registrar fechas de inicio y fin de garantía para los ingresos.

- En el caso de “Egresos”, no se puede retirar más cantidad de la que hay en el stock actual.
- Todo movimiento debe quedar en un historial inmutable indicando “Stock Antes” y “Stock Después”.

HU-04: Alertas de Stock Bajo

Como Administrador de Inventario, quiero visualizar alertas en tiempo real en la pantalla principal, para saber cuándo un repuesto ha alcanzado o superado su límite de stock mínimo y gestionar su compra.

Criterios de aceptación:

- Si la cantidad de stock es menor o igual al stock mínimo configurado, se debe generar una alerta visual y un listado de “Stock Bajo”.

HU-05: Auditoría y Registro de Cambios

Como Administrador del Sistema, quiero consultar un historial detallado de quién modificó qué cosa, para garantizar la trazabilidad y la transparencia en la gestión de inventario.

Criterios de aceptación:

- El sistema debe guardar automáticamente registros cuando se crea, edita, da de baja o restaura un repuesto o tipo de repuesto.

Registros de Entrevistas y Observación Directa

Durante el primer mes de la pasantía se llevaron a cabo reuniones de levantamiento de información y sesiones de observación directa en el área de TI del hospital. A continuación se presentan los registros formalizados de las dos entrevistas principales realizadas.

Entrevista N.º 1 — Tutor empresarial

Fecha	Primer mes de pasantía (febrero de 2026)
Lugar	Virtualmente
Entrevistado	Ing. David Guerrero — Tutor empresarial
Objetivo	Identificar los requerimientos prioritarios del módulo de gestión de repuestos e inventario desde la perspectiva del responsable institucional.
Preguntas y respuestas	**P: ¿Cuál es la problemática principal con el manejo actual del inventario?**

	R: El control se lleva en hojas de cálculo de Excel, lo que genera inconsistencias entre el stock físico y el reportado. Además, no queda registro de quién retiró un repuesto ni en qué equipo se usó. **P: ¿Qué reglas de negocio son críticas para el módulo?** R: No se debe poder eliminar un repuesto físicamente de la base de datos, solo inactivarlo (baja lógica). Todo movimiento de egreso debe tener una justificación escrita. Si hay stock disponible, el repuesto no puede darse de baja. **P: ¿Qué información debe registrarse de cada repuesto?** R: Nombre, tipo, número de parte, número de serie, proveedor, stock mínimo y descripción técnica. Las facturas de compra deben poder adjuntarse en PDF para no perderlas.
Conclusiones	Se priorizó la implementación de baja lógica, trazabilidad de movimientos, registro obligatorio de justificaciones y adjunto de facturas PDF. Estos elementos se formalizaron como requerimientos funcionales RF-01, RF-03, RF-04 y RF-07 del documento SRS.

Entrevista N.º 2 — Equipo del área de TIC

Fecha	Primer mes de pasantía (febrero de 2026)
Lugar	Virtualmente
Entrevistados	Ing. David Guerrero — Tutor empresarial
Objetivo	Conocer las necesidades operativas de los usuarios directos del módulo para asegurar una interfaz usable y eficiente en el flujo de trabajo diario.
Preguntas y respuestas	**P: ¿Cómo buscan actualmente un repuesto específico?** R: Revisamos el archivo de Excel por nombre o buscamos directamente en físico en el almacén. No hay un campo de búsqueda rápida; a veces tardamos varios minutos en encontrar un repuesto. **P: ¿Qué tan seguido necesitan adjuntar facturas o documentos a los repuestos?** R: Cada vez que hacemos un ingreso de repuestos nuevos. Las facturas se extravían fácilmente en papel; sería muy útil poder adjuntarlas digitalmente desde el mismo sistema al momento del registro. **P: ¿Qué información consultan con más frecuencia sobre un repuesto?**

Conclusiones	R: El número de parte para comparar con el fabricante, el stock disponible y quién hizo el último movimiento. También la fecha de garantía cuando reportamos daños a proveedores. Se identificó la necesidad de un buscador por número de parte, visualización inmediata del stock, adjunto de PDF desde el navegador y registro del historial de movimientos con usuario y fecha. Estos hallazgos derivaron en los requerimientos RF-01, RF-03, RF-04 y RNF-02 del documento SRS.

Hallazgos consolidados de la observación directa:

- El control de repuestos se llevaba en hojas de cálculo (Excel) fragmentadas, lo cual generaba desajustes entre el stock físico y el reportado.
- No existía trazabilidad de quién retiraba un repuesto ni en qué equipo médico se usaba.
- La pérdida de facturas de compra y garantías era frecuente, lo que dificultaba las reclamaciones a proveedores.
- No existía mecanismo de alerta cuando el stock de un repuesto llegaba a niveles críticos, lo que derivaba en faltantes inesperados durante mantenimientos urgentes.

Documento SRS (Software Requirements Specification) Consolidado

Requisitos Funcionales (RF)

ID	Descripción
RF-01	El sistema debe permitir la creación, lectura, actualización y desactivación (CRUD lógico) de Repuestos.
RF-02	El sistema debe gestionar las categorías (Tipos de Repuesto).
RF-03	El sistema debe procesar “Movimientos de Stock” calculando matemáticamente el nuevo stock disponible.
RF-04	El sistema debe soportar la carga y descarga de archivos PDF (Facturas) asociados a ingresos de stock, utilizando Multer para el almacenamiento en servidor.
RF-05	El sistema debe generar alertas visuales cuando un repuesto alcance su stock mínimo.
RF-06	El sistema debe permitir la exportación de inventarios y movimientos a formatos CSV y PDF mediante librerías como jsPDF y jspdf-autotable.
RF-07	El sistema debe mantener una bitácora de auditoría inmutable de las operaciones críticas.

Requisitos No Funcionales (RNF)

ID	Descripción
RNF-01 (Seguridad)	Los endpoints del backend deben estar protegidos por middlewares de validación de JWT (checkToken) y autorizados según el rol del usuario (requireRoles).
RNF-02 (Rendimiento)	Las consultas de auditoría y movimientos deben ser paginadas desde el frontend para evitar sobrecarga en la renderización.
RNF-03 (Usabilidad)	La interfaz debe ser desarrollada como una SPA (Single Page Application) usando Angular 17+ con diseño responsivo. Se utilizarán alertas enriquecidas con SweetAlert2.
RNF-04 (Persistencia)	La base de datos relacional debe mantener integridad referencial utilizando Sequelize ORM.

Desarrollo

Diagrama de Base de Datos (Módulo Repuestos)

A continuación, se representa el Modelo Entidad-Relación (MER) lógico de las tablas desarrolladas para el control de inventario, basado en Sequelize ORM.

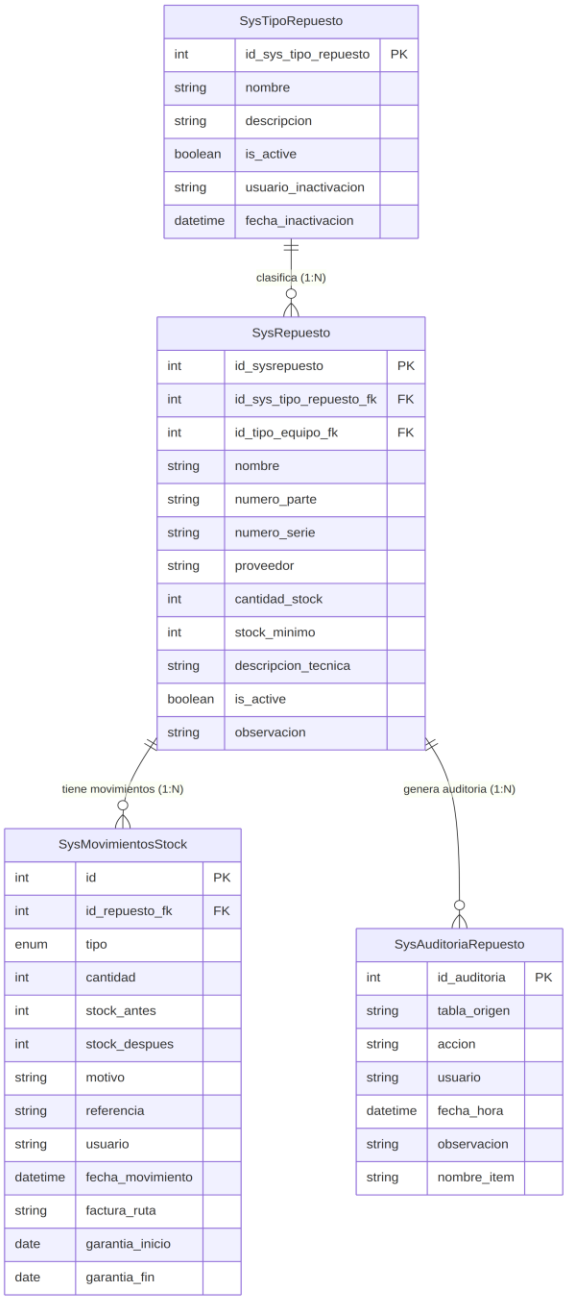


Figura A. Modelo entidad-relación del módulo de Repuestos e Inventario.

Diagrama de Arquitectura

El sistema implementa una arquitectura Cliente-Servidor (Modelo Vista Controlador y servicios API REST).

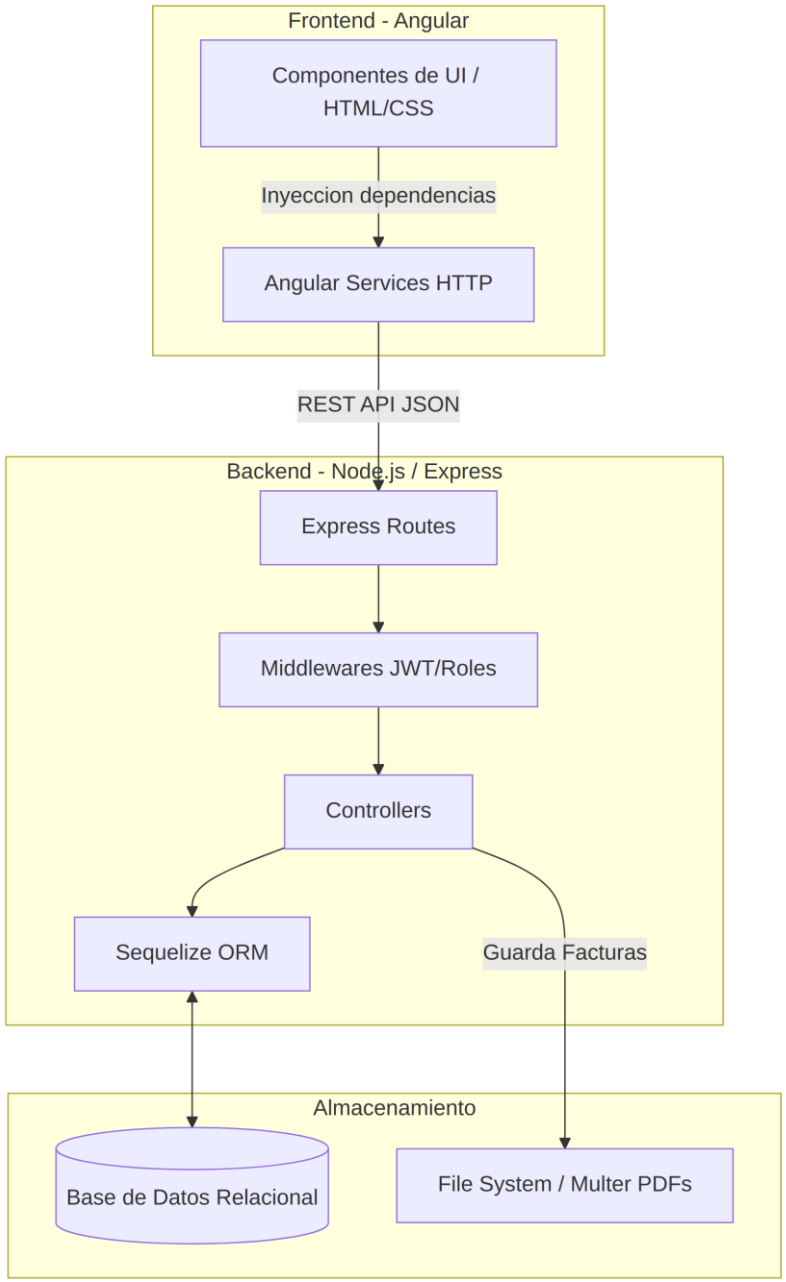
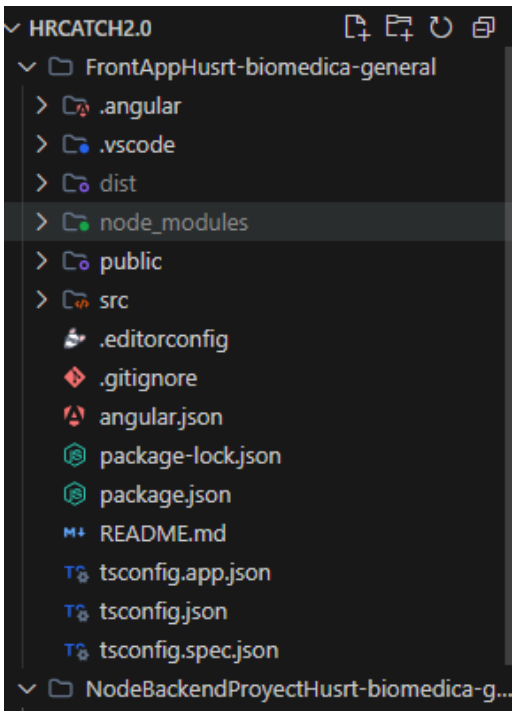
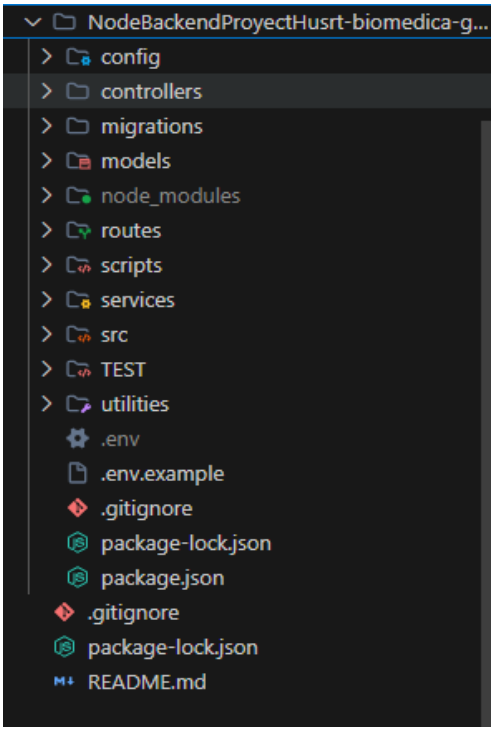


Figura B. Arquitectura general del sistema: Frontend Angular, Backend Node.js/Express y capa de persistencia.

Estructura del Proyecto





Frontend (FrontAppHusrt-biomedica-general/src/app):

- /Components/Sistemas/repuestos/: contiene el componente principal, su HTML y CSS. Maneja la lógica de presentación, modales y paginación.
- /Services/appServices/sistemasServices/: contiene los servicios de Angular (sysrepuestos.service.ts, sysmovimientosstock.service.ts) encargados de hacer las peticiones HttpClient al backend.

Backend (NodeBackendProyectHusrt-biomedica-general/):

- /models/Sistemas/: definición de los modelos de BD mediante Sequelize (SysRepuesto.js, SysMovimientosStockRepuestos.js).
- /controllers/Sistemas/: lógica de negocio y transacciones (sysMovimientosStockController.js).
- /routes/sistemas/: declaración de endpoints (sysMovimientosStockRoutes.js), implementación de roles e inicialización de Multer para la subida de facturas.

Explicación de los Módulos Desarrollados

Los módulos desarrollados durante la pasantía forman parte del proyecto institucional del área de TI del Hospital Universitario San Rafael de Tunja. Aunque el sistema contempla módulos de soporte, incidencias, inventario de activos y base de conocimiento, el alcance de esta pasantía se concentró en el subsistema de gestión de repuestos, que comprende cuatro componentes funcionales interrelacionados. A continuación se describe el diseño y la lógica implementada en cada uno.

1. Módulo de Catálogo de Repuestos (SysRepuesto)

Este módulo centraliza el registro y consulta de repuestos físicos disponibles en el área de TI. Cada repuesto se modela mediante la entidad SysRepuesto, cuyos atributos incluyen: nombre, descripción técnica, número de parte, número de serie, proveedor, stock actual, stock mínimo configurable y estado lógico (activo/inactivo). La interfaz de usuario implementada en Angular 19 con Standalone Components presenta un listado paginado con filtros por nombre, tipo y estado. El formulario de registro y edición utiliza Template-Driven Forms con validaciones que impiden guardar campos obligatorios en blanco y garantizan que el stock mínimo sea un valor positivo.

La eliminación física de registros está bloqueada en la lógica del backend: el endpoint DELETE comprueba si el repuesto tiene stock mayor a cero o si existen movimientos asociados. En cualquiera de esos casos devuelve un error HTTP 409 y sólo permite el cambio de estado a inactivo (baja lógica). Esto preserva la integridad del historial de inventario y permite auditorías futuras.

2. Módulo de Tipos de Repuestos (SysTipoRepuesto)

Complementa al catálogo permitiendo clasificar los repuestos en categorías definidas por el administrador (por ejemplo: cables, fuentes de poder, memorias, discos duros, tarjetas de red). La entidad SysTipoRepuesto se relaciona con SysRepuesto mediante una clave foránea, lo que habilita filtros por tipo en el catálogo y en los reportes de movimientos. El CRUD de tipos está implementado como un componente independiente con tabla editable en línea, permitiendo renombrar o inactivar categorías sin pérdida de información histórica.

3. Módulo de Movimientos de Stock (SysMovimientosStockRepuestos)

Es el componente central del control de inventario. Registra cada ingreso o egreso de repuestos con trazabilidad completa: fecha y hora, cantidad, tipo de movimiento, usuario

que lo ejecuta, justificación escrita y, en el caso de ingresos, la factura del proveedor en formato PDF. El manejo de archivos se implementa con Multer en el backend, que valida tipo MIME y tamaño antes de almacenar el archivo en el servidor bajo un nombre único generado con timestamp.

La lógica de negocio del egreso es transaccional: primero se valida que la cantidad solicitada no supere el stock disponible; si la validación pasa, se registra el movimiento y se actualiza atómicamente el campo "stock actual" de la entidad SysRepuesto. Si la operación falla en cualquier punto, Sequelize hace rollback automático. Para el ingreso, el sistema obliga al usuario a adjuntar la factura en PDF o a documentar una razón explícita en caso de no tener comprobante, garantizando que ninguna entrada de stock quede sin respaldo documental.

Desde el frontend, el historial de movimientos se presenta en una tabla con columnas: fecha, tipo (INGRESO/EGRESO), repuesto, cantidad, usuario, justificación y enlace de descarga de la factura PDF. Los movimientos son sólo de lectura una vez registrados, asegurando la inmutabilidad del histórico.

4. Módulo de Alertas y Auditoría (SysAuditoriaRepuesto)

Proporciona visibilidad sobre el estado del inventario y un registro de eventos del sistema. El panel de alertas muestra automáticamente los repuestos cuyo stock actual es inferior o igual al stock mínimo configurado, diferenciándolos visualmente con colores de alerta mediante los componentes de PrimeNG. Esto permite al personal de TI anticipar faltantes antes de iniciar un mantenimiento programado.

El componente de auditoría registra en la tabla SysAuditoriaRepuesto cada operación sensible: creación, edición, baja lógica y cualquier movimiento de stock. Cada registro almacena la entidad afectada, el tipo de acción, el identificador del usuario, la fecha y la descripción del cambio. Este módulo satisface el requerimiento no funcional de trazabilidad completa exigido por el área de TI del hospital.

Nota: por confidencialidad y derechos de desarrollo institucional no se incluye código fuente en este anexo. Los diagramas de flujo, el modelo ER completo y la arquitectura del sistema se encuentran en el documento SRS y en los manuales técnicos referenciados en el Anexo 9.4.

Pruebas

Informe Formal de Pruebas

Pruebas funcionales: se llevaron a cabo pruebas funcionales exhaustivas del módulo de repuestos utilizando un enfoque de caja negra, simulando el comportamiento de usuarios con roles de Administrador y Técnico. Se verificó que el 100% de las funciones CRUD (crear, leer, actualizar, desactivar) operan sin errores y se reflejan correctamente en la interfaz SPA de Angular. Las validaciones de formularios previnieron el ingreso de datos nulos o inconsistentes.

Pruebas de integración: se validó la comunicación entre la capa Frontend (Angular) y la capa Backend (Express/Node.js). Se comprobó que los archivos PDF enviados vía FormData se procesaran correctamente por Multer, guardándose en el servidor y registrando la ruta en la base de datos sin afectar el hilo principal de Node.js. Asimismo, se verificó que el bloque transaccional de Sequelize en registrarMovimiento realiza un rollback en caso de fallos, evitando corrupciones en el "Stock Después".

Casos de Prueba Definidos y Ejecutados

Caso	Descripción	Resultado esperado	Estado
CP-01	Validación de stock negativo en egreso: verificar que el sistema no permita retirar más repuestos de los que existen en el stock actual.	El sistema bloquea la acción con un mensaje de stock insuficiente; el stock en BD no cambia.	Aprobado
CP-02	Obligatoriedad de adjunto en ingreso con factura: verificar que el sistema exija un archivo PDF cuando se indica que el ingreso cuenta con factura.	El sistema advierte que debe adjuntarse la factura de compra obligatoriamente y no envía la petición al backend.	Aprobado
CP-03	Restricción de inactivación con stock: verificar que un repuesto no pueda darse de baja si aún quedan unidades físicas en el almacén.	El sistema bloquea la acción indicando que debe egresarse todo el stock antes de inactivar.	Aprobado
CP-04	Alerta de stock mínimo: comprobar que el banner de alerta de inventario funcione dinámicamente al reducirse el stock por debajo del mínimo configurado.	Aparece un banner de alerta indicando los repuestos con stock bajo y el badge correspondiente en la tabla.	Aprobado

Pruebas de Aceptación

Una vez finalizadas las pruebas funcionales y de integración, se llevó a cabo una sesión de validación con el tutor empresarial y el equipo del área de TI del hospital. Durante esta

sesión se verificó que los módulos desarrollados respondieran a los requerimientos levantados en la fase de especificación y se hizo entrega de este.

Pruebas de Integración

Las siguientes capturas corresponden a la ejecución real de la suite de pruebas mediante el comando npm test (Jest, con Supertest y la base de datos en memoria descrita en la sección 2), tomadas en el equipo de desarrollo.

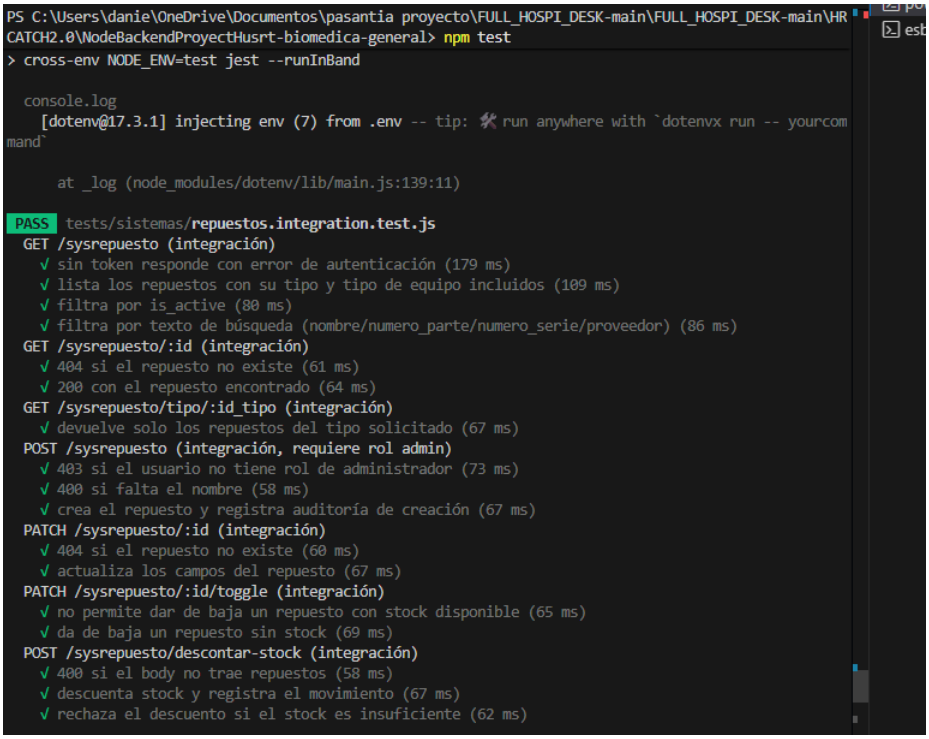


Figura 1. Resultado de tests/sistemas/repuestos.integration.test.js (npm test).

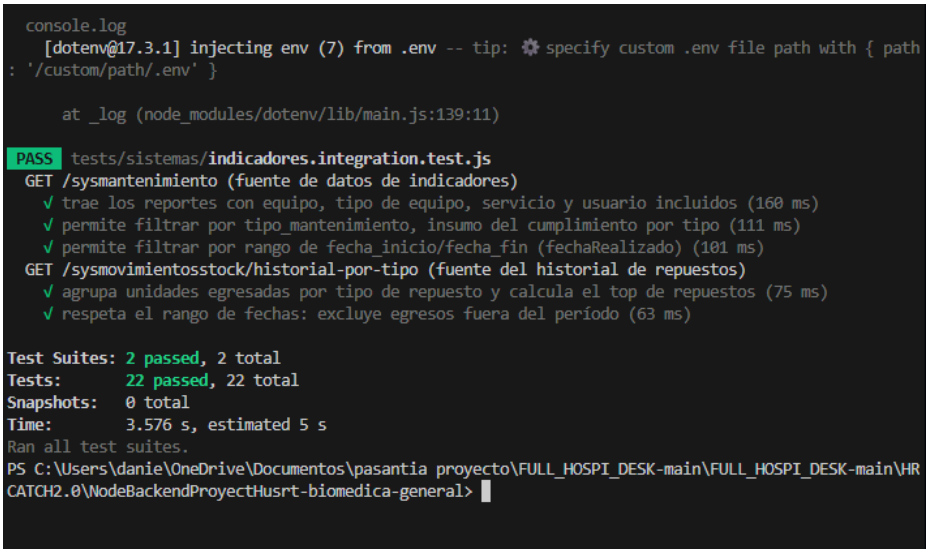


Figura 2. Resultado de tests/sistemas/indicadores.integration.test.js y resumen final: 2 suites y 22 pruebas exitosas.

Explicación

Las pruebas se construyeron con Jest como corredor de pruebas y Supertest para simular peticiones HTTP reales contra la aplicación Express, sin sustituir el controlador ni el modelo por dobles simulados (mocks). Por esa razón son pruebas de integración y no unitarias: atraviesan de manera real la ruta, el middleware, el controlador y Sequelize contra una base de datos real, en vez de aislar una sola función.

La pieza que permite que estas pruebas sean rápidas y no toquen la base de datos de producción está en config/configDb.js: se agregó una condición para que, cuando la variable de entorno NODE_ENV tiene el valor test, Sequelize se conecte a SQLite en memoria en lugar de MariaDB. El ORM y las consultas que ejecuta el código de producción son exactamente las mismas; lo único que cambia es el motor de base de datos subyacente, que en pruebas es efímero y se destruye al finalizar el proceso.

Alrededor de esa base de datos en memoria se construyeron cuatro archivos de apoyo dentro de tests/helpers, tal como se ilustra en la Figura 3. El archivo db.js importa todos los modelos Sequelize que participan en los flujos probados para que sus asociaciones queden registradas, y expone resetDb(), que recrea todas las tablas antes de cada prueba (beforeEach), de modo que ningún test hereda datos de otro. El archivo app.js construye una aplicación Express mínima que monta únicamente el enrutador necesario para cada suite, evitando la lógica de arranque del archivo de producción src/index.js, sin dejar de

usar el enrutador y el controlador reales. El archivo auth.js firma tokens JWT reales con el mismo secreto que usa checkToken en producción, exponiendo adminToken() y userToken() para ejercitar tanto los caminos permitidos como los rechazos por rol. El archivo factories.js concentra funciones que insertan una fila válida por defecto y aceptan overrides para personalizar solo lo relevante en cada prueba.

Con esa infraestructura, cada prueba sigue la misma secuencia: resetDb() deja la base limpia, las factories insertan los datos necesarios, Supertest ejecuta la petición HTTP real, y finalmente se verifica tanto la respuesta HTTP como el estado real que quedó en la base de datos (por ejemplo, releyendo el repuesto con repuesto.reload() para confirmar que el stock cambió o que no cambió en el camino de error). En total se implementaron veintidós pruebas: diecisiete en repuestos.integration.test.js sobre los endpoints de /sysrepuesto, y cinco en indicadores.integration.test.js sobre las dos fuentes de datos que alimentan el tablero de Indicadores (/sysmantenimiento y /sysmovimientosstock/historial-por-tipo), ya que ese módulo no tiene un endpoint propio.

Documentación

El presente anexo detalla exclusivamente el índice estructurado del manual desarrollado para el Módulo de Gestión de Repuestos del sistema HRCATCH 2.0, en cumplimiento de las políticas de confidencialidad de la información interna. La documentación completa (manuales técnico, de administrador y por perfil de usuario) se entrega de forma adjunta junto con este informe.

TABLA DE CONTENIDO: