Manual Técnico

Sistema de Gestión de Bajas Técnicas

Equipo 5:
Jocdan L. López Mantecón
Lianny Reveé Valdivieso
Kevin A. Torres Perera
Cristhian Delgado García

Índice

1. Introducción

Este manual técnico tiene como objetivo documentar de manera estructurada y precisa los aspectos fundamentales del producto desarrollado, abarcando desde su alcance funcional hasta los criterios técnicos que sustentan su implementación. Se inicia con una descripción general que contextualiza el sistema dentro de su entorno operativo, identifica las características de los usuarios finales y establece las restricciones generales que condicionan su diseño. Esta sección proporciona una visión clara del propósito del producto y delimita el marco en el que se desarrollará.

A continuación, se detallan los requerimientos específicos que guían el desarrollo. Los requerimientos funcionales enumeran las acciones que el sistema debe realizar según las necesidades expresadas por el cliente, mientras que los requerimientos no funcionales establecen condiciones adicionales como usabilidad, seguridad, decisiones de diseño e implementación, y restricciones de entorno. Estos elementos aseguran que el producto no solo cumpla con su propósito funcional, sino que también sea seguro, accesible, mantenible y compatible con el entorno tecnológico del usuario.

Finalmente, el documento aborda las funcionalidades del sistema mediante diagramas de casos de uso, justifica el enfoque metodológico adoptado, presenta la arquitectura propuesta, y argumenta la elección de patrones de diseño, visualización y datos. Se concluye con el modelo de datos que define la estructura de la base de datos.

2. Descripción del Producto

El alcance del producto, denominado **InfraGestión**, abarca la creación de un sistema integral para la gestión del ciclo de vida completo de los equipos técnicos de una organización. El sistema cubrirá desde el registro y la validación inicial de un equipo, pasando por su asignación, gestión de mantenimientos , y traslados entre diferentes ubicaciones o departamentos. Además, incluirá módulos para la gestión de personal técnico, permitiendo registrar valoraciones de rendimiento y bonificaciones, y un robusto sistema de reportes para la toma de decisiones estratégicas por parte de la dirección. Finalmente, el sistema gestionará la baja de equipos, asegurando una trazabilidad completa desde la adquisición hasta su destino final.

Desde la perspectiva del producto, InfraGestión es una aplicación web interna diseñada para centralizar y automatizar la administración de activos técnicos, reemplazando posibles procesos manuales o descentralizados. Sus usuarios son empleados de la organización con roles bien definidos: Directores que necesitan visibilidad global y reportes para la toma de decisiones; Administradores que gestionan usuarios y la estructura organizativa; Responsables de Sección que supervisan el inventario y el personal de su área; Técnicos que ejecutan las tareas de mantenimiento y validación; y Personal de Logística que gestiona la recepción y el proceso de baja de equipos. Cada rol tendrá acceso únicamente a las funcionalidades pertinentes a sus responsabilidades, garantizando la seguridad y la integridad de los datos. Finalmente, la usabilidad es clave, requiriendo un diseño responsivo compatible con los principales navegadores modernos y una curva de aprendizaje rápida para los nuevos usuarios.

3. Análisis de Requerimientos

Se utiliza la priorización MoSCoW:

- **M (Must Have)**: Requisito esencial para el lanzamiento. Sin él, el sistema no es viable.
- **S (Should Have)**: Requisito importante, pero no vital. Podría posponerse si es necesario.
- **C (Could Have)**: Requisito deseable pero no necesario. Aporta valor, pero puede omitirse.
- **W (Won't Have)**: Requisito que no se incluirá en esta versión del proyecto.

3.1. Requerimientos Funcionales

RF-AUTH: Módulo de Autenticación de Usuarios

ID	Descripción	Prioridad
RF-AUTH-001	El sistema deberá permitir a los usuarios iniciar sesión utilizando un identificador único y una contraseña.	M
RF-AUTH-002	El sistema deberá cerrar la sesión del usuario de forma segura.	M
RF-AUTH-003	El sistema deberá implementar un control de acceso basado en roles (RBAC) con los siguientes perfiles definidos: Director, Administrador, Responsable de Sección, Técnico y Personal de Logística (Receptor de equipo). Cada rol solo podrá ver y ejecutar las funciones asignadas.	M
RF-AUTH-004	El sistema deberá permitir a los usuarios con rol "Administrador" crear, modificar y desactivar cuentas de usuario. Al crear o modificar un usuario, se le deberá asignar su rol (según RF-AUTH-003) y su Sección/Departamento de pertenencia. No se permitirá la eliminación de usuarios para mantener la integridad histórica.	M
RF-AUTH-006	El sistema deberá contar con cuentas de usuario demo para las demostraciones.	S

RF-INV: Módulo de Gestión de Inventario de Equipos Técnicos

ID	Descripción	Prioridad
RF-INV-001	El sistema deberá permitir al "Administrador" iniciar el registro de nuevos equipos con los siguientes campos obligatorios: identificador único, nombre, tipo y fecha de adquisición. El equipo ingresará con el estado 'Pendiente de Revisión'.	M
RF-INV-002	El sistema deberá mostrar una vista de lista de todo el inventario de equipos. Los usuarios solo podrán ver los equipos a los que su rol les da permiso.	M
RF-INV-003	La vista de lista del inventario deberá ser filtrable por, al menos, los siguientes campos: tipo de equipo, estado y ubicación actual.	M
RF-INV-004	La vista de lista del inventario deberá permitir la búsqueda por identificador único y por nombre del equipo.	M
RF-INV-005	El sistema deberá permitir hacer clic en un equipo para ver una "Ficha de Equipo" detallada, que incluirá toda su información y el historial de mantenimientos y traslados. La información incluye: Nombre, Estado del equipo, Tipo, Fecha de Adquisición, Departamento, Sección, Nombre del responsable de sección, Registro de mantenimientos (Fecha, Tipo, Técnico, Notas, Costo), Registro de Traslados (Fecha, Origen, Destino, Responsable, Receptor), Información sobre la defectación inicial (Fecha de emisión, solicitante, técnico, estado, fecha de respuesta).	M
RF-INV-006	El sistema deberá permitir al "Gestor de Inventario" modificar los datos de un equipo existente.	M
RF-INV-007	El rol de "Responsable de Sección" debe permitir la vista de la lista de todo el inventario de su área y de los equipos en desuso de otras áreas para solicitar traslado.	M
RF-INV-008	El rol de "Director" debe permitir la vista de todo el inventario de la empresa.	M
RF-INV-009	Tras el registro (RF-INV-001), el sistema deberá permitir al "Administrador" asignar el equipo 'Pendiente de Revisión' a un "Técnico" específico para su validación.	M
RF-INV-010	El sistema deberá permitir al "Técnico" asignado aprobar o rechazar la revisión del nuevo equipo. Si se aprueba, el técnico asignará la ubicación (Sección y Departamento) y el estado cambiará a 'Operativo'. Si se rechaza, el técnico deberá registrar una justificación y el equipo quedará en estado 'Rechazado'.	M

RF-MAIN: Módulo de Gestión de Mantenimientos

ID	Descripción	Prioridad
RF-MAIN-001	El sistema deberá permitir a los "Técnicos" registrar una intervención de mantenimiento asociada a un equipo específico.	M
RF-MAIN-002	El registro de mantenimiento deberá incluir: fecha, tipo de mantenimiento (preventivo, correctivo, diagnóstico), costo asociado (si aplica) y una descripción de los trabajos realizados. El técnico responsable se asignará automáticamente al usuario que registra la intervención.	M
RF-MAIN-003	Cada intervención registrada deberá añadirse automáticamente al historial de la "Ficha de Equipo" correspondiente.	M

RF-BAJA: Módulo de Gestión de Bajas Técnicas

ID	Descripción	Prioridad
RF-BAJA-001	El sistema deberá permitir a un "Técnico" iniciar una "Propuesta de Baja" para un equipo, cambiando su estado a 'Pendiente de Baja'.	M
RF-BAJA-002	La propuesta deberá registrar la causa de la baja (seleccionada de una lista predefinida), la fecha y notas adicionales.	M
RF-BAJA-003	El sistema deberá notificar al rol "Personal de Logística" sobre las nuevas propuestas de baja pendientes de revisión.	M
RF-BAJA-004	El "Personal de Logística" deberá poder aprobar o rechazar una propuesta de baja. Si se rechaza, el estado del equipo volverá a su estado anterior y se deberá añadir una nota de justificación.	M
RF-BAJA-005	Al aprobar una baja, el sistema deberá permitir al "Personal de Logística" registrar el destino final del equipo y los datos de la persona o entidad receptora, incluyendo campos obligatorios para: identificador del receptor, nombre del receptor, y departamento al que pertenece. El estado del equipo cambiará permanentemente a 'Dado de Baja'.	M
RF-BAJA-006	El sistema deberá permitir al "Personal de Logística", durante el proceso de recepción de equipos (para baja o mantenimiento), definir y asignar el "Departamento responsable" que trabajará internamente con el dispositivo.	M

RF-TRAS: Módulo de Gestión de Traslados

ID	Descripción	Prioridad
RF-TRAS-001	El sistema deberá permitir a los "Responsables de Sección" iniciar un traslado de un equipo, especificando origen, destino y la persona responsable se asignará automáticamente al usuario que realiza la intervención.	M
RF-TRAS-002	Al iniciar un traslado, el estado del equipo deberá cambiar a 'En Tránsito'.	M
RF-TRAS-003	El sistema deberá permitir al receptor en el destino confirmar la recepción del equipo.	M
RF-TRAS-004	La confirmación de recepción podrá realizarse mediante el ingreso del ID del equipo a través de un dispositivo móvil o tablet, que llevará directamente a la pantalla de confirmación.	S
RF-TRAS-005	Una vez confirmada la recepción, la ubicación del equipo se actualizará en el sistema y su estado volverá a 'Operativo' (o el que tuviera antes del traslado).	M
RF-TRAS-006	Si desde la vista del inventario del "Responsable de Sección" se hace click en un equipo en desuso en otra sección, debe mostrarse la opción de iniciar traslado.	S

RF-PERS: Módulo de Gestión de Personal Técnico

ID	Descripción	Prioridad
RF-PERS-001	El sistema deberá mantener un registro de cada técnico con su nombre, número de identificación, años de experiencia y especialidad.	M
RF-PERS-002	El sistema deberá permitir a los "Responsables de Sección" registrar valoraciones de rendimiento (ej. de 1 a 5 estrellas, con comentarios) asociadas al historial de intervenciones de los técnicos de su equipo.	M
RF-PERS-003	El sistema deberá proveer una función para el "Director" que permita registrar bonificaciones o penalizaciones (monetarias o descriptivas) asociadas al historial de rendimiento de un técnico.	

RF-REP: Módulo de Reportes

ID	Descripción	Prioridad
RF-REP-001	El sistema deberá permitir al "Director" y "Responsables de Sección" generar un reporte del estado del inventario, filtrable por estado, tipo y ubicación.	M
RF-REP-002	El sistema deberá permitir al "Director" generar un reporte de bajas técnicas en un rango de fechas, que pueda ser agrupado por causa, tipo de equipo o departamento de origen.	M
RF-REP-003	El sistema deberá permitir al "Director" generar un reporte de efectividad del personal técnico, basado en el número de intervenciones y las valoraciones de sus superiores.	M
RF-REP-004	Cualquier vista de lista o reporte generado por el sistema deberá poder exportarse a formato PDF, accesible para todos los roles de usuario.	M
RF-REP-005	El sistema deberá incluir una funcionalidad para ordenar ascendentemente o descendentemente los datos mostrados en cualquier tabla (listas de inventario, reportes) haciendo clic en el encabezado de la columna.	M
RF-REP-006	El sistema deberá generar un reporte de equipos que requieren reemplazo, identificando aquellos que han recibido tres o más mantenimientos (preventivos o correctivos) en los últimos 12 meses.	M
RF-REP-007	El sistema deberá generar un reporte de equipos enviados a un departamento específico, mostrando fecha, origen, destino, nombre de quien envía y nombre de quien recibe.	M
RF-REP-008	El sistema deberá generar un reporte de correlación de rendimiento técnico (Análisis de Fallos). El reporte debe mostrar los 5 técnicos con la peor correlación (mayor costo total de mantenimiento y menor longevidad) en equipos que fueron dados de baja por "fallo técnico irreparable", incluyendo su valoración promedio.	M
RF-REP-009	El sistema deberá permitir al "Director" generar un reporte para determinar bonificaciones (RF-PERS-003), basado en la cantidad de intervenciones y las valoraciones promedio (RF-PERS-002) de los técnicos en un rango de fechas.	M

RF-ORG: Módulo de Gestión Organizacional

ID	Descripción	Prioridad
RF-ORG-001	El sistema deberá permitir al "Administrador" crear, modificar y desactivar Secciones.	M
RF-ORG-002	El sistema deberá permitir al "Administrador" crear, modificar y desactivar Departamentos, asignando cada departamento a una Sección existente.	M

ID	Descripción	Prioridad
RF-ORG-003	El sistema deberá permitir al "Administrador" asignar un usuario con rol "Responsable de Sección" como el encargado principal de una Sección específica.	M

3.2. Requisitos No Funcionales (RNF)

RNF-PERF: Rendimiento

ID	Descripción	Prioridad
RNF-PERF-001	El tiempo de carga para cualquier página de la aplicación no deberá exceder los 3 segundos en una conexión de banda ancha estándar (10 Mbps).	M
RNF-PERF-002	Las consultas y búsquedas en la vista de inventario con hasta 50,000 registros deberán devolver resultados en menos de 2 segundos.	M
RNF-PERF-003	La generación de reportes complejos no deberá exceder los 10 segundos.	M
RNF-PERF-004	El sistema deberá soportar al menos 50 usuarios concurrentes realizando operaciones de lectura/escritura sin una degradación del rendimiento superior al 20 % respecto a la línea base.	M

RNF-USA: Usabilidad y Accesibilidad

ID	Descripción	Prioridad
RNF-USA-001	La interfaz deberá ser responsiva, garantizando una experiencia de usuario completa y funcional en navegadores de escritorio (resolución mínima 1280x720) y tablets (resolución mínima 768x1024).	M
RNF-USA-002	Un nuevo usuario con el rol de "Técnico" deberá ser capaz de registrar una intervención de mantenimiento y proponer una baja sin requerir más de 60 minutos de formación.	M
RNF-USA-003	Los elementos de la interfaz (botones, menús, terminología) deberán ser consistentes a lo largo de toda la aplicación.	M

RNF-SEC: Seguridad

ID	Descripción	Prioridad
RNF-SEC-001	Las contraseñas de los usuarios deberán almacenarse en la base de datos utilizando un algoritmo de hash.	S

RNF-REL: Fiabilidad y Disponibilidad

ID	Descripción	Prioridad
RNF-REL-001	El sistema deberá tener una disponibilidad (uptime) del 99.5 % durante el horario laboral (Lunes a Viernes, 8:00-18:00).	M

RNF-PORT: Portabilidad

ID	Descripción	Prioridad
RNF-PORT-001	La aplicación web deberá ser compatible con las dos últimas versiones estables de los siguientes navegadores: Google Chrome, Mozilla Firefox, Microsoft Edge y Safari.	M

RNF-IMP: Restricciones de Implementación

ID	Descripción	Prioridad
RNF-IMP-001	El backend deberá desarrollarse utilizando un framework moderno y mantenido.	M
RNF-IMP-002	El frontend deberá desarrollarse utilizando un framework de moderno.	M
RNF-IMP-003	La base de datos deberá ser un sistema de gestión de bases de datos relacional.	M
RNF-IMP-004	El código fuente deberá estar gestionado en un sistema de control de versiones (Git) y alojado en un repositorio central (GitHub).	M

3.3. Requisitos de Entorno

3.4. Requerimientos de Hardware

Servidor Backend (API)

Recomendados (Entorno de Producción)

- **Procesador:** Intel Core i5 (10ª generación+) o AMD Ryzen 5 equivalente
- **Núcleos:** 4 cores físicos / 8 threads
- **Velocidad:** 3.0 GHz o superior
- **Memoria RAM:** 8-16 GB
- **Almacenamiento:** 100 GB de espacio libre en disco SSD NVMe
- **Tipo de almacenamiento:** SSD NVMe para mejor rendimiento de I/O

Servidor de Base de Datos

Recomendados (PostgreSQL - Producción)

- **Procesador:** Intel Core i5/i7 o AMD Ryzen 5/7
- **Núcleos:** 4+ cores físicos
- **Memoria RAM:** 8-16 GB dedicados
- **Almacenamiento:** 100-500 GB SSD NVMe (dependiendo del volumen de datos)
- **IOPS:** Mínimo 3000 IOPS para operaciones de lectura/escritura
- **Backup Storage:** Capacidad adicional equivalente a 2-3x el tamaño de la BD principal

Cálculo Estimado de Almacenamiento

Componente	Tamaño Estimado
50,000 equipos en inventario	2-5 GB
Historial de mantenimientos (5 años)	10-20 GB
Historial de traslados y bajas	5-10 GB
Logs y auditoría	5 GB/año
Total estimado	**30-50 GB**
Recomendado con margen	**100 GB**

Cuadro 15: Estimación de almacenamiento de base de datos

Servidor Web/Frontend

Mínimos

- **Procesador:** Intel Core i3 o AMD Ryzen 3

- **Memoria RAM:** 2 GB
- **Almacenamiento:** 5 GB SSD

Estaciones de Trabajo (Usuarios Finales)

Mínimos

- **Procesador:** 2 vCPUs
- **RAM:** 2 GB
- **Disco Duro:** 5 GB de espacio libre
- **Ancho de Banda:** 10 Mbps

Dispositivos Móviles/Tablets

Mínimos (para confirmación de traslados - RF-TRAS-004)

- **Sistema Operativo:** iOS 13+ o Android 8.0+
- **Memoria RAM:** 2 GB
- **Resolución:** 768x1024 (mínimo para tablets)
- **Conectividad:** Wi-Fi 802.11n o 4G LTE

3.5. Requerimientos de Software

Sistemas Operativos Compatibles

Desarrollo

- Windows 10/11 (64-bit)
- macOS 10.15 (Catalina) o superior
- Linux (Ubuntu 20.04+, Debian 10+, Fedora 33+)

Producción (Servidor)

- Windows Server 2016 R2 o superior
- Linux (Ubuntu Server 20.04+, CentOS 8+, RHEL 8+)
- Contenedores Docker (Linux/Windows)

Backend (.NET API)

Runtime y SDK

- **.NET SDK:** Versión 9.0 (LTS)
  - Versión completa: 9.0.x (última revisión estable)
  - Descarga: `https://dotnet.microsoft.com/download/dotnet/9.0`
- **ASP.NET Core Runtime:** 9.0.x
- **C# Language Version:** 11.0

Paquetes NuGet Principales

Paquete	Versión	Propósito
Microsoft.EntityFrameworkCore	9.0.10	ORM para acceso a datos
Microsoft.EntityFrameworkCore.SQLite	9.0.10	Proveedor SQLite (Desarrollo)
Microsoft.AspNetCore.Authentication.JwtBearer	8.0.11	Autenticación JWT
AutoMapper	12.0.1	Mapeo objeto-a-objeto
FluentValidation	11.9.0	Validación de datos
BCrypt.Net-Next	4.0.3	Hashing de contraseñas
System.IdentityModel.Tokens.Jwt	8.14.0	Generación de tokens JWT
Swashbuckle.AspNetCore	6.4.0	Documentación Swagger/OpenAPI

Base de Datos

Producción (Recomendado)

- **PostgreSQL:** Versión 15.x o superior (Recomendado: 16.x)
  - Extensiones requeridas: `pgcrypto`, `uuid-ossp`
  - Configuración de Character Set: UTF-8
  - Timezone: UTC o zona horaria local de la empresa
  - Max Connections: Mínimo 100

Cadena de Conexión PostgreSQL (Ejemplo):

```
Host=localhost;Port=5432;Database=infragestion;
Username=infra_user;Password=secure_password;
SSL Mode=Require;
```

Frontend

Para Desarrollo

Componente	Versión	Obligatorio
.NET SDK	8.0 o superior (proyecto usa .NET 9.0)	Sí
Visual Studio Code	Última versión	Sí
Extensión C# para VS Code	Última versión	Sí
Git	2.x o superior	Sí
Node.js	16.x o superior	Opcional

Dependencias Principales

- **Blazor WebAssembly:** Framework frontend
- **ASP.NET Core Web API:** Servicios backend
- **Blazored.LocalStorage:** Almacenamiento local del navegador
- **Entity Framework Core:** ORM para acceso a datos

Para Ejecución (Runtime)

Componente	Requisito
Navegador Web	Chrome 90+, Firefox 88+, Edge 90+, Safari 14+
Backend API	Configurado en `http://localhost:5147`
ASP.NET Core Runtime	.NET 9.0
Entity Framework Core	Incluido en el proyecto

4. Funcionalidades del Sistema

4.1. Actores del Sistema

Actores del Sistema

En el sistema existen los siguientes actores, cada uno con responsabilidades específicas:

- **Director:** es un usuario con acceso total al sistema. Tiene la capacidad de generar reportes y autorizar bonificaciones.
- **Administrador:** también posee acceso completo. Gestiona las cuentas de usuario, configurar secciones y departamentos, así como iniciar equipos. Tiene control administrativo total del sistema.
- **Responsable de Sección:** supervisa los equipos y técnicos dentro de su área. Además, puede solicitar traslados y registrar evaluaciones de los técnicos, asegurando el correcto desempeño de su sección.
- **Técnico:** se encarga de registrar mantenimientos, proponer bajas de equipos y validar la incorporación de nuevos dispositivos. Su rol está directamente ligado a la operación técnica y al estado de los equipos.
- **Personal de Logística(Receptor):** procesa las bajas y traslados de equipos, además de registrar las recepciones correspondientes.
- **Sistema:** Representa notificaciones, eventos y tareas que el program realiza de forma automática.

4.2. Casos de Uso

MÓDULO 1: AUTENTICACIÓN (RF-AUTH)

ID	Nombre	Descripción	Actor Principal	Actor Sec.
UC-AUTH-001	Iniciar Sesión	Usuario se autentica con ID y contraseña	Todos los usuarios	Sistema
UC-AUTH-002	Cerrar Sesión	Usuario cierra sesión de forma segura	Todos los usuarios	Sistema
UC-AUTH-003	Validar Permisos	Sistema valida permisos basados en rol	Sistema	Todos los usuarios
UC-AUTH-004	Gestionar Cuentas de Usuario	Admin crea/modifica/desactiva usuarios	Administrador	Sistema

Cuadro 19

MÓDULO 2: INVENTARIO (RF-INV)

ID	Nombre	Descripción	Actor Principal	Actor Sec.
UC-INV-001	Registrar Nuevo Equipo	Administrador crea registro de equipo	Administrador	Sistema
UC-INV-002	Ver Lista de Inventario	Visualizar todos los equipos disponibles	Director, Admin, Resp., Técnico, Log.	Sistema
UC-INV-003	Filtrar Inventario	Filtrar por estado, tipo, ubicación	Todos	Sistema
UC-INV-004	Buscar Equipo	Buscar por ID o nombre específico	Todos	Sistema
UC-INV-005	Ver Ficha Detallada	Visualizar detalles completos + historial	Todos	Sistema
UC-INV-006	Modificar Datos Equipo	Gestor actualiza información de equipo	Administrador	Sistema
UC-INV-009	Asignar Equipo a Técnico	Administrador asigna para validación	Administrador	Técnico, Sistema

ID	Nombre	Descripción	Actor Principal	Actor Sec.
UC-INV-010	Validar/Rechazar Equipo	Técnico aprueba o rechaza con justificación	Técnico	Sistema

Cuadro 20

MÓDULO 3: MANTENIMIENTOS (RF-MAIN)

ID	Nombre	Descripción	Actor Principal	Actor Sec.
UC-MAIN-001	Registrar Intervención	Técnico documenta trabajo realizado	Técnico	Sistema
UC-MAIN-003	Actualizar Historial Auto.	Sistema añade intervención a ficha	Sistema	Equipo

Cuadro 21

MÓDULO 4: BAJAS TÉCNICAS (RF-BAJA)

ID	Nombre	Descripción	Actor Principal	Actor Sec.
UC-BAJA-001	Proponer Baja de Equipo	Técnico inicia proceso de baja	Técnico	Sistema, Log.
UC-BAJA-002	Registrar Detalles Baja	Técnico especifica causa y notas	Técnico	Sistema
UC-BAJA-003	Notificar Baja Pendiente	Sistema notifica a Logística	Sistema	Pers. Log.
UC-BAJA-004	Aprobar/Rechazar Baja	Logística aprueba o rechaza propuesta	Pers. Log.	Sistema, Técnico
UC-BAJA-005	Registrar Receptor	Logística documenta quién recibe	Pers. Log.	Sistema
UC-BAJA-006	Asignar Depto. Resp.	Logística asigna depto para trabajo	Pers. Log.	Sistema

Cuadro 22

MÓDULO 5: TRASLADOS (RF-TRAS)

ID	Nombre	Descripción	Actor Principal	Actor Sec.
UC-TRAS-001	Iniciar Traslado	Responsable solicita movimiento	Resp. Sección	Sistema, Log.
UC-TRAS-002	Cambiar a En Tránsito	Sistema actualiza estado durante transporte	Sistema	Equipo
UC-TRAS-003	Confirmar Recepción	Logística confirma llegada	Pers. Log.	Sistema
UC-TRAS-004	Confirmar por QR/ID	Escaneo de código para confirmación rápida	Pers. Log.	Sistema
UC-TRAS-005	Actualizar Ubicación Auto.	Sistema cambia ubicación e historial	Sistema	Equipo

Cuadro 23

MÓDULO 6: PERSONAL TÉCNICO (RF-PERS)

ID	Nombre	Descripción	Actor Principal	Actor Sec.
UC-PERS-001	Mantener Registro Técnico	Admin crea/actualiza datos técnico	Administrador	Sistema
UC-PERS-002	Registrar Valoración	Responsable valora técnico	Resp. Sección	Sistema
UC-PERS-003	Registrar Bonificación/Penalización	Director registra incentivos económicos	Director	Sistema

Cuadro 24

MÓDULO 7: REPORTES (RF-REP)

ID	Nombre	Descripción	Actor Principal	Actor Sec.
UC-REP-001	Reporte Estado Inventario	Director/Resp. ve estado equipos	Director, Resp.	Sistema
UC-REP-002	Reporte Bajas Técnicas	Director ve bajas en rango de fechas	Director	Sistema
UC-REP-003	Reporte Efectividad	Director analiza rendimiento técnicos	Director	Sistema
UC-REP-004	Exportar a PDF	Se exporta el reporte en PDF	Director	Sistema
UC-REP-006	Reporte Equipos a Reemplazar	Director ve equipos con 3+ mant. en 12m	Director	Sistema
UC-REP-007	Reporte Traslados	Director ve equipos enviados a depto	Director	Sistema

Cuadro 25

MÓDULO 8: ORGANIZACIÓN (RF-ORG)

ID	Nombre	Descripción	Actor Principal	Actor Sec.
UC-ORG-001	Gestionar Secciones	Administrador CRUD de secciones	Administrador	Sistema
UC-ORG-002	Gestionar Departamentos	Administrador CRUD de departamentos	Administrador	Sistema
UC-ORG-003	Asignar Responsable	Administrador designa responsable	Administrador	Sistema

Cuadro 26

4.3. Historias de Usuario

Gestión de Equipos

- **US1.1**: Como Administrador, quiero registrar un nuevo equipo para mantener el inventario actualizado
- **US1.2**: Como Técnico, quiero ver los detalles completos de un equipo para realizar mantenimiento informado
- **US1.3**: Como Responsable, quiero filtrar equipos por estado para identificar problemas
- **US1.4**: Como Técnico, quiero validar un equipo nuevo para asegurar su operatividad

Gestión de Bajas

- **US2.1**: Como Técnico, quiero proponer la baja de un equipo para removerlo del inventario
- **US2.2**: Como Personal Logística, quiero revisar bajas pendientes para autorizar su disposición
- **US2.3**: Como Personal Logística, quiero registrar el destino final para auditar el proceso
- **US2.4**: Como Técnico, quiero recibir notificación del resultado de mi propuesta

Gestión de Traslados

- **US3.1**: Como Responsable Sección, quiero solicitar un traslado para reorganizar recursos
- **US3.2**: Como Personal Logística, quiero confirmar la recepción para completar traslado
- **US3.3**: Como Personal Logística, quiero usar QR para confirmar rápidamente

Reportes y Análisis

- **US4.1**: Como Director, quiero ver reportes del estado del inventario para tomar decisiones
- **US4.2**: Como Director, quiero analizar el rendimiento técnico para optimizar recursos
- **US4.3**: Como Director, quiero exportar reportes para compartir información

Gestión de Usuarios

- **US5.1**: Como Administrador, quiero crear usuarios con roles específicos para controlar acceso
- **US5.2**: Como Administrador, quiero desactivar cuentas para mantener seguridad

5. Enfoque Metodológico

Para el desarrollo del sistema **InfraGestion**, se adoptó un enfoque metodológico híbrido, combinando la robustez de una fase inicial de planificación y diseño, con la flexibilidad y adaptabilidad de un marco de trabajo ágil para las fases de construcción, pruebas y entrega.

En una primera etapa se llevó a cabo una pofunda fase de Análisis y Planificación donde se presentaron al cliente los resultados de este análisis: Informe de Análisis y Diseño, que documenta el trabajo del equipo de modelar y justificar la estructura de datos para el proyecto la descripción inicial del problema y una formulación de los requerimientos funcionales y no funcionales que sirven de base al modelo, además presenta el Modelo Entidad Relacional Extendido (MERX) diseñado para representar de forma íntegra las entidades, atributos y relaciones del dominio, acompañado de la especificación de las restricciones de integridad detectadas durante el diseño. También se presentó el Informe de Arquitectura, donde se presenta un análisis profundo y una propuesta para la Arquitectura y Diseño de la Aplicación.

A partir de entonces se adoptó una metodología ágil donde se planificaron una serie de entregas cortamente espaciadas en el tiempo, iterativas y progresivas, basada en Hitos que presentan al cliente funcionalidades completas (Vertical Slices).

5.1. Justificación del Enfoque

El problema está muy bien definido, como lo demuestra orientación detallada del proyecto, lo que facilita una fuerte planificación inicial. Existen múltiples módulos interconectados (Inventario, Mantenimiento, Usuarios, Reportes, etc.) y requisitos no funcionales estrictos (rendimiento, seguridad). Una fase inicial de planificación y arquitectura es fundamental para diseñar una base sólida que soporte esta complejidad, defina las relaciones clave entre las entidades de la base de datos y establezca una arquitectura escalable.

Aunque los requisitos están bien definidos, es crucial que el cliente pueda ver y probar el software en funcionamiento para proporcionar retroalimentación valiosa. El enfoque ágil, con sus entregas incrementales al final de cada sprint (cada 1-2 semanas), permite al cliente interactuar con el producto de manera regular, validar que las funcionalidades cumplen con sus expectativas y solicitar ajustes de forma temprana, evitando desviaciones costosas al final del proyecto.

Dividir el desarrollo en hitos manejables permite al equipo concentrarse en un conjunto limitado de funcionalidades a la vez. Esto reduce la carga cognitiva, mejora la calidad del código y facilita la gestión de un proyecto complejo a largo plazo.

Un enfoque híbrido aprovecha lo mejor de ambos mundos: la planificación y el diseño riguroso para establecer una base arquitectónica sólida y un alcance claro, combinado con la flexibilidad, la entrega de valor temprana y la capacidad de adaptación Ágil para construir el producto de manera eficiente y colaborativa.

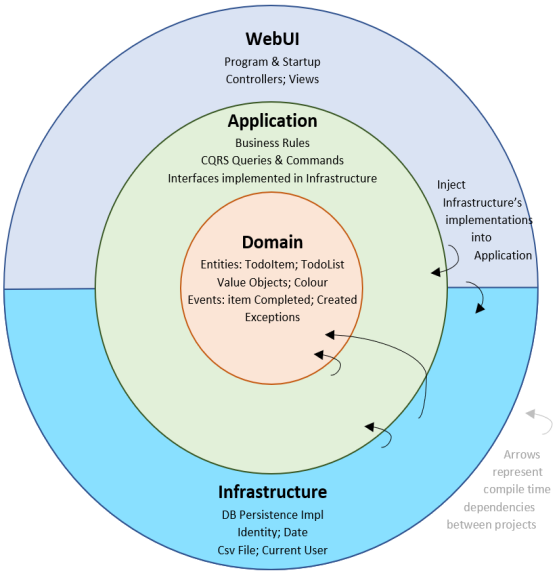
6. Arquitectura

De forma general, se sigue una arquitectura **Cliente-Servidor** donde el backend(servidor) contiene el modelo y la lógica del negocio, realiza el procesamiento y la persistencia de datos. El frontend(cliente) es una **Single Page Application (SPA)** que proporciona una interfaz cómoda e intuitiva al usuario para interactuar con el programa desde el navegador web a través de una API REST.

6.1. Backend

Para el backend se adoptó **Clean Architecture**. La Clean Architecture organiza el código en capas concéntricas, donde las dependencias apuntan siempre hacia el centro (el dominio). Esta arquitectura prioriza la independencia de frameworks, UI, bases de datos y cualquier agente externo, colocando la lógica de negocio en el núcleo del sistema. La misma: Proporciona desacoplamiento total entre capas, garantiza alta testabilidad en todos los niveles, facilitar extensibilidad y mantenibilidad a largo plazo y cumple con todos los requisitos técnicos del proyecto.

Figura 1: Arquitectura del Backend




```
InfraGestion/
├── InfraGestion.sln ..... Archivo de solución que agrupa todos los proyectos
├── README.md .....Información general del proyecto
├── LICENSE .....Licencia del proyecto
├── docs/ ..... Directorio para la documentación
│   ├── Requirements.md .....Requisitos del sistema
│   ├── adr/ ..... Decisiones de arquitectura (Architecture Decision Records)
│   └── architecture/ .....Otros documentos de arquitectura
├── src/ .....CÓDIGO FUENTE DE LA APLICACIÓN
│   ├── Domain/ .....CAPA DE DOMINIO (Entidades, Reglas de Negocio)
│   │   ├── Aggregations/ ..... Agregados del dominio
│   │   ├── Entities/ ..... Entidades de negocio (ej. User, Device)
│   │   ├── Enums/ .....Tipos enumerados
│   │   ├── Exceptions/ .....Excepciones de negocio personalizadas
│   │   ├── Interfaces/ ..... Abstracciones de repositorios y servicios del dominio
│   │   └── Domain.csproj .....Archivo del proyecto de la capa Domain
│   ├── Application/ ..... CAPA DE APLICACIÓN (Casos de Uso, Lógica de App)
│   │   ├── DTOs/ ..... Data Transfer Objects para la comunicación con la API
│   │   ├── Mappings/ ..... Perfiles de mapeo (ej. AutoMapper)
│   │   ├── Services/ .....Implementación de la lógica de los casos de uso
│   │   ├── Validators/ ..... Lógica de validación para los DTOs
│   │   └── Application.csproj .....Archivo del proyecto de la capa Application
│   ├── Infrastructure/ CAPA DE INFRAESTRUCTURA (Base de datos, APIs externas)
│   │   ├── Data/ .....DbContext de Entity Framework Core
│   │   ├── Migrations/ ..... Migraciones de la base de datos
│   │   ├── Repositories/ .....Implementación concreta de los repositorios
│   │   ├── Services/ .....Servicios externos (ej. envío de email)
│   │   └── Infrastructure.csproj ..... Archivo del proyecto de la capa Infrastructure
│   └── Web.API/ .....CAPA DE PRESENTACIÓN (API REST)
│       ├── Controllers/ ..... Controladores que exponen los endpoints de la API
│       ├── Middleware/ .....Middleware personalizado (ej. para manejo de errores)
│       ├── Program.cs .....Punto de entrada y configuración de la aplicación web
│       ├── appsettings.json ..... Configuración de la aplicación
│       └── Web.API.csproj .....Archivo del proyecto de la capa Web.API
└── tests/ ..... PROYECTOS DE PRUEBAS
    ├── Application.UnitTests/ ..... Pruebas unitarias para la capa de Aplicación
    └── Web.API.IntegrationTests/ .....Pruebas de integración para la API
```

6.2. Frontend

El proyecto está desarrollado como una **Single Page Application (SPA)** utilizando Blazor WebAssembly, lo que implica que todo el código C# se ejecuta directamente en el navegador del cliente gracias a WebAssembly. De esta manera, no es necesario contar con un servidor web para manejar la lógica de negocio del frontend, lo que permite ofrecer una experiencia de usuario fluida y continua sin recargas de página. Además, la aplicación mantiene la comunicación con un backend RESTful a través de solicitudes HTTP, garantizando la integración de datos y funcionalidades de manera eficiente.

Feature-Based Architecture (Arquitectura Basada en Características)

El código está organizado por **features/módulos funcionales** en lugar de por capas técnicas.

```
InfraGestion.Web/
└── Features/
    ├── Auth/ ..... Módulo de Autenticación
    │   ├── Components/
    │   ├── Pages/
    │   ├── Services/
    │   └── Models/
    ├── Users/ ..... Módulo de Usuarios
    │   ├── Components/
    │   ├── Pages/
    │   ├── Services/
    │   └── Models/
    ├── Inventory/ ..... Módulo de Inventario
    │   ├── Components/
    │   ├── Pages/
    │   ├── Services/
    │   ├── Models/
    │   ├── DTOs/
    │   └── Helpers/
    └── Departments/ ..... Módulo de Departamentos
        └── Services/
```

Figura 2: Estructura Feature-Based de InfraGestion.Web

Ventajas de este enfoque:

- **Alta cohesión**: Todo lo relacionado con una característica está junto
- **Bajo acoplamiento**: Los módulos son independientes entre sí
- **Escalabilidad**: Fácil agregar nuevos módulos sin afectar los existentes
- **Mantenibilidad**: Los desarrolladores encuentran rápidamente el código relevante
- **Reutilización**: Componentes específicos del dominio bien encapsulados

7. Patrones de Diseño

7.1. Patrones Creacionales

7.1.1. Dependency Injection (DI)

Propósito: Inversión de control para gestionar dependencias.

Implementación en el Proyecto:

```
private static void InjectInfrastructure(WebApplicationBuilder builder)
{
  ...
  // Repository Pattern - Todos con Scoped Lifetime
  builder.Services.AddScoped<ISectionRepository, SectionRepository>();
  builder.Services.AddScoped<IUserRepository, UserRepository>();
  builder.Services.AddScoped<IMaintenanceRecordRepository, MaintenanceRepository
      >();
  builder.Services.AddScoped<IDeviceRepository, DeviceRepository>();
  builder.Services.AddScoped<IDepartmentRepository, DepartmentRepository>();
  builder.Services.AddScoped<IDecommissioningRequestRepository,
  DecommissioningRequestRepository>();
  builder.Services.AddScoped<ITransferRepository, TransferRepository>();
  builder.Services.AddScoped<IRejectionRepository, RejectionRepository>();
  builder.Services.AddScoped<IPerformanceRatingRepository,
  PerformanceRatingRepository>();
  builder.Services.AddScoped<IDecommissioningRepository,
      DecommissioningRepository>();
  builder.Services.AddScoped<ITechnicianRepository, TechnicianRepository>();

  // Infrastructure Services
  builder.Services.AddScoped<IPasswordHasher, PasswordHasher>();
  builder.Services.AddScoped<IJwtTokenGenerator, JwtTokenGenerator>();
}
```

Ventajas :

- **Testabilidad:** Fácil mockear servicios en tests.
- **Bajo Acoplamiento:** Las clases no instancian sus dependencias.
- **Configuración Centralizada:** Todo en `Program.cs`.

7.2. Patrones Estructurales

7.2.1. Repository Pattern

Propósito: Abstracción de acceso a datos.

Implementación Genérica:

```
public interface IRepository<TEntity> where TEntity : class
{

  Task<TEntity?> GetByIdAsync(int id, CancellationToken cancellationToken =
      default);
  Task<IEnumerable<TEntity>> GetAllAsync(CancellationToken cancellationToken =
      default);
  Task<IEnumerable<TEntity>> FindAsync(
  Expression<Func<TEntity, bool>> predicate,
  CancellationToken cancellationToken = default);

  // Command operations (WRITE)
```

```
    Task<TEntity> AddAsync(TEntity entity, CancellationToken cancellationToken =
        default);
    Task AddRangeAsync(IEnumerable<TEntity> entities, CancellationToken
        cancellationToken = default);
    Task UpdateAsync(TEntity entity, CancellationToken cancellationToken = default)
        ;
    Task DeleteAsync(TEntity entity, CancellationToken cancellationToken = default)
        ;
    Task DeleteRangeAsync(IEnumerable<TEntity> entities, CancellationToken
        cancellationToken = default);
}
```

Implementación Base:

```
public class Repository<TEntity> : IRepository<TEntity> where TEntity : class
{
  protected readonly ApplicationDbContext _context;
  protected readonly DbSet<TEntity> _dbSet;

  public Repository(ApplicationDbContext context)
  {
    _context = context ?? throw new ArgumentNullException(nameof(context));
    _dbSet = _context.Set<TEntity>();
  }

  public virtual async Task<TEntity?> GetByIdAsync(
  int id,
  CancellationToken cancellationToken = default)
  {
    return await _dbSet.FindAsync(new object[] { id }, cancellationToken);
  }

  public virtual async Task<IEnumerable<TEntity>> GetAllAsync(
  CancellationToken cancellationToken = default)
  {
    return await _dbSet.ToListAsync(cancellationToken);
  }

  public virtual async Task<IEnumerable<TEntity>> FindAsync(
  Expression<Func<TEntity, bool>> predicate,
  CancellationToken cancellationToken = default)
  {
    return await _dbSet.Where(predicate).ToListAsync(cancellationToken);
  }

  public virtual async Task<TEntity> AddAsync(
  TEntity entity,
  CancellationToken cancellationToken = default)
  {
    if (entity == null)
    throw new ArgumentNullException(nameof(entity));

    await _dbSet.AddAsync(entity, cancellationToken);
    return entity;
  }


}
```

Ventajas:

- **Abstracción de Persistencia:** Cambiar de SQLite a PostgreSQL sin cambiar lógica.

- **Testabilidad:** Fácil mockear repositorios.
- **Centralización:** Toda la lógica de acceso a datos en un lugar.
- **Expresiones LINQ:** Queries fuertemente tipadas.

7.2.2. Unit of Work Pattern

Propósito: Mantener transaccionalidad y consistencia en operaciones.

Implementación:

```
public interface IUnitOfWork
{
  Task<int> SaveChangesAsync(CancellationToken cancellationToken = default);
}
```

```
//Ubicacion: src/Infrastructure/Data/ApplicationDbContext.cs

public class ApplicationDbContext : DbContext, IUnitOfWork
{
  public async Task<int> SaveChangesAsync(CancellationToken cancellationToken =
      default)
  {
    return await base.SaveChangesAsync(cancellationToken);
  }
}
```

Ventajas:

- **Transaccionalidad:** Todas las operaciones se guardan atómicamente.
- **Consistencia:** Evita estados intermedios inconsistentes.
- **Performance:** Múltiples operaciones en una sola transacción DB.
- **Control Explícito:** El servicio decide cuándo persistir.

7.2.3. Data Transfer Object (DTO) Pattern

Propósito: Transferir datos entre capas sin exponer entidades de dominio.

Implementación:

```
//Ubicacion: src/Application/DTOs/Auth/

// DTO para Login
public record LoginRequestDto(string UserId, string Password);

public record LoginResponseDto
{
  public required string UserId { get; init; }
  public required string FullName { get; init; }
  public required string Role { get; init; }
  public required string DepartmentName { get; init; }
  public required string AccessToken { get; init; }
  public string? RefreshToken { get; init; }
  public DateTime ExpiresAt { get; init; }
}

// DTO para Crear Usuario
public record CreateUserRequestDto
{
  public required string UserId { get; init; }
  public required string FullName { get; init; }
  public required string Email { get; init; }
  public required string Password { get; init; }
  public required string Role { get; init; }
```

```
    public required int DepartmentId { get; init; }
}

// DTO para Usuario (salida)
public record UserDto
{
    public required string UserId { get; init; }
    public required string FullName { get; init; }
    public required string Email { get; init; }
    public required string Role { get; init; }
    public required string DepartmentName { get; init; }
    public required bool IsActive { get; init; }
    public DateTime CreatedAt { get; init; }
}
```

```
//Ubicacion: src/Application/DTOs/Inventory/

// DTO para Dispositivo
public record DeviceDto
{
    public int DeviceId { get; init; }
    public required string Name { get; init; }
    public required string Type { get; init; }
    public required string Status { get; init; }
    public required string CurrentLocation { get; init; }
    public DateTime AcquisitionDate { get; init; }
}
```

Ventajas:

- **Encapsulación:** No expone modelo de dominio al cliente.
- **Seguridad:** Evita *over-posting attacks*.
- **Versionado:** Fácil agregar nuevas versiones de API.
- **Optimización:** Solo transfiere datos necesarios.
- **Inmutabilidad:** Uso de `record` garantiza inmutabilidad.

7.2.4. Mapper Pattern (AutoMapper)

Propósito: Conversión automática entre entidades y DTOs.

Implementación:

```
//Ubicacion: src/Application/Mappings/AuthMappingProfile.cs

using AutoMapper;

public class AuthMappingProfile : Profile
{
    public AuthMappingProfile()
    {
        // User to UserDto
        CreateMap<User, UserDto>()
        .ForMember(dest => dest.UserId, opt => opt.MapFrom(src => src.UserId))
        .ForMember(dest => dest.Role, opt => opt.MapFrom(src => src.Role.Name))
        .ForMember(dest => dest.DepartmentName,
        opt => opt.MapFrom(src => src.Department.Name));

        // User to LoginResponseDto
        CreateMap<User, LoginResponseDto>()
        .ForMember(dest => dest.UserId, opt => opt.MapFrom(src => src.UserId))
        .ForMember(dest => dest.Role, opt => opt.MapFrom(src => src.Role.Name))
        .ForMember(dest => dest.DepartmentName,
        opt => opt.MapFrom(src => src.Department.Name))
```

```
            .ForMember(dest => dest.AccessToken, opt => opt.Ignore())
            .ForMember(dest => dest.RefreshToken, opt => opt.Ignore())
            .ForMember(dest => dest.ExpiresAt, opt => opt.Ignore());

            // CreateUserRequestDto to User
            CreateMap<CreateUserRequestDto, User>()
            .ForMember(dest => dest.UserId, opt => opt.Ignore())
            .ForMember(dest => dest.PasswordHash, opt => opt.Ignore())
            .ForMember(dest => dest.RoleId, opt => opt.Ignore())
            .ForMember(dest => dest.Role, opt => opt.Ignore())
            .ForMember(dest => dest.Department, opt => opt.Ignore())
            .ForMember(dest => dest.CreatedAt, opt => opt.Ignore())
            .ForMember(dest => dest.IsActive, opt => opt.Ignore());
        }
    }
```

Ventajas:

- **Reducción de Código Boilerplate:** Mapeo automático.
- **Configuración Centralizada:** Todos los mapeos en un lugar.
- **Type-Safe:** Errores de compilación si el mapeo falla.
- **Flatten Properties:** Mapeo de propiedades anidadas.

7.3. Patrones Comportamentales

7.3.1. Service Layer Pattern

Propósito: Encapsular lógica de negocio en servicios.

Implementación:

```
    //Ubicacion: src/Application/Services/Interfaces/IInventoryService.cs

    public interface IInventoryService
    {
        Task<IEnumerable<DeviceDto>> GetInventoryAsync(
        DeviceFilterDto filter,
        int userId);
        Task<DeviceDetailDto> GetDeviceDetailAsync(int deviceId);
        Task<IEnumerable<DeviceDto>> SearchDevicesAsync(string searchTerm);
        Task<DeviceDto> CreateDeviceAsync(CreateDeviceRequestDto request);
        Task<DeviceDto> UpdateDeviceAsync(int deviceId, UpdateDeviceRequestDto
            request);
    }
```

```
    //Ubicacion: src/Application/Services/Implementations/InventoryService.cs

    public class InventoryService : IInventoryService
    {
        // Implementacion del Servicio
    }
```

Ventajas:

- **Separación de Responsabilidades:** Lógica de negocio separada de controladores.
- **Reusabilidad:** Servicios usables desde múltiples controladores.
- **Testabilidad:** Fácil probar lógica de negocio.
- **Transaccionalidad:** Control de `UnitOfWork` en servicios.

8. Modelo de Datos

A continuación se presenta un enlace al Diagrama del Modelo de Datos que determina la estructura de la base de datos del proyecto. Se empleó el **Modelo Entidad-Relacional Extendido (MERX)**:

./Resources/Merx.png