Universidad de la Habana
Facultad de Matemática y Computación
Carrera: Ciencia de la Computación
Año: 3ero
Asignatura: Ingeniería de Software

(a) Logo

(b) Miniatura

Figura 1: Arte de la aplicación

Sistema de Gestión de Misiones de Jujutsu Kaisen

Informe del Seminario Integrador

Integrantes:

Nombre y Apellidos	**Grupo**	**Usuario Telegram**
Ronald Alfonso Pérez	C-311	(@RAlfonsoP),
Juan Carlos Carmenate Díaz	C-311	(@Juank404),
Sebastian González Alfonso	C-312	(@sebagonz106),
Abraham Rey Sánchez Amador	C-312	(@AbrahamR_Sanchez).

Curso: 2025-2026

Resumen

Este documento presenta el diseño integral del sistema de Gestión de Misiones desarrollado en el contexto académico de la asignatura de Ingeniería de Software y Bases de Datos. El sistema permite administrar el ciclo completo de operaciones de exorcismo dentro del universo de Jujutsu Kaisen, abarcando desde la detección de maldiciones hasta la generación de reportes analíticos. Mediante un proceso de análisis exhaustivo, se definieron los requerimientos funcionales del problema y se diseñó un modelo conceptual relacional-extendido que garantiza la integridad y consistencia de los datos. La arquitectura implementa un patrón en capas con cinco niveles: dominio, repositorio, servicios, controladores y frontend, utilizando ASP.NET con Entity Framework para el backend y React con TypeScript para el frontend. El modelo de datos, normalizado hasta la tercera forma normal, asegura la eliminación de redundancias y la preservación de dependencias. Los resultados demuestran que el diseño constituye una base sólida para la implementación, cumpliendo con todos los objetivos establecidos y ofreciendo flexibilidad para extensiones futuras.

Índice

1 Introducción

El presente informe documenta el proceso de análisis, diseño e implementación parcial del sistema de Gestión de Misiones, desarrollado en el marco de las asignaturas de Ingeniería de Software y Bases de Datos impartidas en el primer Semestre de Tercer Año de la Facultad de Matemática y Computación de la Universidad de la Habana. Este proyecto tiene como objetivo principal crear una solución tecnológica robusta y escalable para administrar el ciclo completo de operaciones relacionadas con el exorcismo de maldiciones en el universo de Jujutsu Kaisen, integrando principios modernos de ingeniería de software con un diseño riguroso de bases de datos relacionales. Este informe corresponde a una entrega intermedia del proyecto, documentando tanto las funcionalidades core completadas como las proyecciones de trabajo para consultas analíticas avanzadas.

El sistema responde a la necesidad de centralizar y optimizar la gestión de entidades críticas del dominio: maldiciones, hechiceros, misiones, recursos, técnicas malditas, traslados y ubicaciones. Proporciona una plataforma integral que permite desde el registro inicial de amenazas hasta la generación de reportes analíticos avanzados sobre efectividad operativa. La complejidad del dominio se caracteriza por relaciones jerárquicas entre hechiceros, múltiples estados de entidades, restricciones de asignación basadas en compatibilidad de grados y seguimiento detallado de técnicas malditas aplicadas durante las misiones. Esta complejidad inherente requiere un diseño meticuloso que garantice consistencia referencial, escalabilidad horizontal y rendimiento óptimo bajo cargas concurrentes.

Desde la perspectiva tecnológica, el sistema implementa una arquitectura en capas de cinco niveles claramente diferenciados: **Core** (modelos de dominio con lógica de negocio), **Infraestructura** (repositorios con acceso a datos), **Aplicación** (servicios con orquestación de lógica), **Web** (controladores REST con validación) y **Frontend** (interfaz de intercambio con el usuario). El backend se construye con **ASP.NET Core 9.0** utilizando **Entity Framework Core 9.0.9** como ORM sobre **SQL Server**, garantizando transacciones ACID y migraciones versionadas. El frontend utiliza **React 19.1** con **TypeScript 5.6+** para seguridad de tipos, **React Hook Form 7.53 + Zod 3.23** para validación de formularios, y **Tailwind CSS 4.1** con PostCSS para estilización modular. La autenticación se implementa mediante **JWT (JSON Web Tokens)** con BCrypt para hashing de contraseñas, y CORS configurado para permitir comunicación segura entre frontend (puerto 5173) y backend (puerto 5189).

El alcance de este informe abarca la especificación detallada de requerimientos específicos de la aplicación, una descripción completa de la arquitectura del sistema en 5 capas con justificación de patrones aplicados, un análisis del modelo de datos y la exposición del enfoque metodológico basado en desarrollo incremental con integración continua y organización en subequipos especializados.

2 Alcance del Producto

El sistema de Gestión de Misiones constituye una plataforma web integral para administrar de manera centralizada y eficiente el ciclo completo de operaciones de exorcismo de maldiciones. El alcance funcional del producto abarca desde la detección y registro inicial de amenazas (maldiciones) hasta el análisis retrospectivo de efectividad operativa mediante

reportes analíticos avanzados.

2.1 Capacidades Principales del Sistema

El sistema permite ejecutar las siguientes operaciones críticas del negocio:

- **Gestión de Maldiciones**: Registro completo de maldiciones con atributos como nombre, grado, tipo, ubicación, estado, fecha de aparición y descripción detallada. Permite seguimiento de evolución de amenazas y establecimiento de prioridades.
- **Gestión de Hechiceros**: Administración del personal operativo con información de nombre, grado, nivel de experiencia, estado operativo, técnica principal dominada y registro de técnicas malditas adicionales. Incluye evaluación de disponibilidad para asignaciones.
- **Planificación y Asignación de Misiones**: Creación de misiones vinculadas a maldiciones activas con definición de ubicación objetivo, nivel de urgencia, asignación inteligente de hechiceros considerando compatibilidad de grados, disponibilidad y experiencia, y designación de hechiceros encargados de grado alto o especial.
- **Seguimiento de Ejecución**: Monitoreo en tiempo real del estado de misiones con transiciones entre estados (pendiente, en curso, completada con éxito, completada con fracaso, cancelada), registro de eventos ocurridos durante la ejecución, documentación de daños colaterales, y captura de técnicas malditas aplicadas por cada hechicero participante.
- **Gestión de Recursos**: Control de inventario de recursos materiales, registro de uso de recursos en misiones específicas con cantidades y fechas, y gestión de personal de apoyo con sus roles y estado de disponibilidad.
- **Control de Traslados**: Registro detallado de traslados de hechiceros entre ubicaciones con origen, destino, fecha/hora, método de transporte, motivo del traslado, y estado. Permite optimización de tiempos de respuesta ante emergencias.
- **Catálogo de Técnicas Malditas**: Gestión de técnicas con nombre, tipo, nivel de dominio requerido, condiciones específicas de uso, y seguimiento de cuáles hechiceros dominan cada técnica con su nivel de maestría.
- **Generación de Reportes Analíticos**: Sistema de consultas avanzadas que incluye:
  - Historial completo de misiones por hechicero con tasas de éxito/fracaso
  - Cálculo de efectividad promedio de técnicas clasificada como Alta ($> 75\,\%$), Media ($50 - 75\,\%$), o Baja ($< 50\,\%$)
  - Ranking de top 3 hechiceros por cada grado según porcentaje de misiones exitosas
  - Análisis comparativo de desempeño entre grados en misiones críticas
  - Estadísticas de frecuencia de aparición de maldiciones por ubicación y período temporal
  - Evaluación de utilización de recursos y correlación con resultados de misiones

- **Sistema de Autenticación y Autorización**: Control de acceso basado en roles (soporte, hechicero, administrador) con permisos diferenciados:
  - **Personal de Soporte**: Gestión de recursos, traslados y logística
  - **Hechiceros**: Consulta de misiones asignadas, actualización de estado propio y registro de resultados
  - **Administradores**: Acceso completo incluyendo creación de misiones, asignación de hechiceros, gestión de usuarios y generación de reportes ejecutivos

2.2 Descripción General del Producto

El sistema es una aplicación web centralizada de tipo cliente-servidor que opera bajo el paradigma REST para comunicación frontend-backend. Actúa como el núcleo operativo para todas las actividades relacionadas con gestión de misiones, reemplazando métodos manuales o dispersos de coordinación y proporcionando una fuente única de verdad (single source of truth) para datos operativos críticos.

Está diseñado para atender cuatro perfiles diferenciados:

- **Supervisores/Administradores**: Personal responsable de mantener la consistencia global del sistema y de gestionar la asignación y la evolución profesional de los hechiceros. Disponen de vistas ejecutivas consolidadas, capacidades de análisis multidimensional y privilegios administrativos para garantizar la gobernanza y la integridad del sistema.
- **Hechiceros Operativos**: Usuarios de campo cuya función principal es ejecutar misiones, registrar resultados en terreno y consultar información operativa. Además, incorporan un conjunto de capacidades de gestión relacionadas con asignación y evolución de personal: pueden solicitar o proponer reasignaciones, actualizar su perfil de competencia y, según su nivel, ejecutar acciones de asignación o evaluación sobre personal de menor grado.
- **Personal de Apoyo**: Especialistas en logística, comunicaciones o áreas médicas encargados de gestionar traslados, administrar inventarios de recursos y coordinar la infraestructura de soporte. Sus flujos de trabajo están orientados a la eficiencia operativa y requieren interfaces y vistas especializadas para sus responsabilidades, así como trazabilidad y notificaciones integradas.

Pueden identificarse como restricciones generales:

- **Integridad de Datos**: Debe garantizar consistencia absoluta de datos críticos mediante restricciones de integridad referencial, validaciones de negocio en capa de aplicación, y transacciones ACID. Ninguna misión puede quedar en estado inconsistente ante fallos.
- **Disponibilidad Operativa**: Debe mantener disponibilidad 24/7 para soportar operaciones de emergencia con tolerancia a fallos mediante retry logic en acceso a datos, manejo robusto de excepciones, y capacidad de recuperación ante desconexiones temporales.

- **Usabilidad**: La interfaz debe ser intuitiva para usuarios con diferentes niveles de alfabetización tecnológica, con tiempo de capacitación no superior a 2 horas para operaciones básicas y 4 horas para funcionalidades administrativas avanzadas.
- **Escalabilidad**: La arquitectura debe soportar crecimiento en número de usuarios concurrentes (proyección de 100+ usuarios simultáneos), volumen de datos (miles de misiones históricas), y extensión funcional sin requerir rediseño arquitectónico fundamental.
- **Seguridad**: Protección de información sensible mediante autenticación robusta (JWT con expiración), autorización basada en roles con principio de mínimo privilegio, y comunicación segura (HTTPS en producción) entre componentes.
- **Portabilidad**: El frontend debe ser accesible desde navegadores modernos (Chrome 90+, Firefox 88+, Safari 14+, Edge 90+) y dispositivos con diferentes factores de forma (desktop, tablets, smartphones) mediante diseño responsivo.

3 Requerimientos Específicos

3.1 Requerimientos Funcionales

Los requerimientos funcionales se prsentan clasificados en dos categorías según su estado de implementación: **Implementados** (funcionalidades CRUD core completadas) y **En Desarrollo** (consultas analíticas y reportes avanzados).

3.1.1. Requerimientos Funcionales Implementados

ID	Nombre	Descripción
RF-01	Gestión de Maldiciones	Registrar, actualizar, consultar y eliminar maldiciones con información completa.
RF-02	Gestión de Hechiceros	Mantener registro completo de hechiceros con información completa
RF-03	Gestión de Técnicas Malditas	Catálogo de técnicas malditas con su respectiva informaccion. Registro de dominio de técnicas por hechicero.
RF-04	Creación y Asignación de Misiones	Generar misiones asociadas a maldiciones activas especificando ubicación y nivel de urgencia, asignando hechiceros participantes y supervisores.
RF-05	Seguimiento de Estado de Misiones	Actualizar en tiempo real el estado de misiones mediante transiciones: pendiente → en progreso → completada con éxito/fracaso o cancelada. Registro de eventos ocurridos, daños colaterales y técnicas aplicadas durante ejecución.

ID	Nombre	Descripción
RF-06	Gestión de Ubicaciones	Administrar ubicaciones geoespaciales donde ocurren misiones y aparecen maldiciones.
RF-07	Registro de Traslados	Documentar traslados de hechiceros entre ubicacione.
RF-08	Gestión de Recursos	Administrar inventario de recursos materiales y registrar uso de recursos en misiones específicas con cantidades y fechas.
RF-09	Gestión de Personal de Apoyo	Mantener registro de personal de soporte con roles y estado de disponibilidad.
RF-10	Autenticación y Autorización	Sistema de login/registro con roles diferenciados, control de acceso basado en roles a endpoints específicos, y gestión de tokens JWT con expiración.
RF-11	Sistema de Auditoría	Registro automático de operaciones críticas con timestamp, usuario responsable, tipo de entidad y datos modificados para trazabilidad completa.

3.1.2. Requerimientos Funcionales en Desarrollo

ID	Nombre	Descripción
RF-12	Consulta de Maldiciones por Estado	Permitir consultar todas las maldiciones filtradas por estado específico. Incluir ordenamiento por columnas y exportación a PDF.
RF-13	Historial de Misiones por Hechicero	Obtener listado completo de todas las misiones en las que ha participado un hechicero específico.
RF-14	Reporte de Misiones Exitosas por Período	Consultar todas las misiones completadas con éxito dentro de un rango de fechas específico.
RF-15	Cálculo de Efectividad de Técnicas	Generar reporte para cada hechicero calculando el promedio de efectividad de las técnicas utilizadas en combate, clasificando el resultado como Alta ($> 75\,\%$), Media ($50 - 75\,\%$) o Baja ($< 50\,\%$) según tasa de éxito en misiones donde aplicó dichas técnicas.
RF-16	Ranking de Hechiceros por Nivel de Misión	Listar los tres hechiceros con mayor porcentaje de éxito para cada nivel de urgencia de misión. Mostrar número total de misiones realizadas, misiones exitosas y porcentaje de éxito.
RF-17	Jerarquía Hechicero-Discípulos	Establecer relación jerárquica entre cada hechicero y sus discípulos o miembros de equipo habituales.

ID	Nombre	Descripción
RF-18	Análisis Comparativo de Efectividad por Grado	Determinar el porcentaje de efectividad de hechiceros de grado medio en misiones de emergencia crítica que involucraron maldiciones de grado especial, y comparar estadísticamente este rendimiento con el de hechiceros de grado alto en situaciones equivalentes.
RF-19	Exportación de Reportes a PDF	Funcionalidad transversal que permita exportar cualquier consulta o reporte generado a formato PDF con formato estructurado, incluyendo gráficos cuando corresponda, encabezados con fecha de generación y usuario responsable.

3.2 Requerimientos No Funcionales

3.2.1. Usabilidad

- **RNF-U01 - Interfaz Intuitiva**: La interfaz debe seguir principios de diseño centrado en el usuario con navegación consistente entre módulos, feedback visual inmediato ante acciones del usuario, y mensajes de error descriptivos en español. Todos los formularios deben incluir validación en tiempo real con indicadores claros de campos obligatorios y errores específicos.
- **RNF-U02 - Curva de Aprendizaje**: El tiempo de capacitación no debe exceder 2 horas para usuarios operativos en funciones básicas, y 4 horas para administradores en funcionalidades avanzadas de generación de reportes y gestión de usuarios.
- **RNF-U03 - Flujos de Trabajo Diferenciados**: Cada rol debe tener un dashboard personalizado que priorice las funcionalidades más relevantes para su perfil, minimizando clics necesarios para operaciones frecuentes (máximo 3 clics para acciones principales).
- **RNF-U04 - Responsive Design**: La interfaz debe adaptarse fluidamente a diferentes resoluciones (mínimo 320px de ancho) y factores de forma (desktop, tablet, smartphone) sin pérdida de funcionalidad crítica, aunque la entrada de datos complejos se optimiza para desktop.
- **RNF-U05 - Accesibilidad**: Cumplimiento con estándares WCAG 2.1 nivel AA incluyendo navegación por teclado, contraste de colores adecuado, y soporte para lectores de pantalla en vistas principales.
- **RNF-U06 - Internacionalización**: La arquitectura debe facilitar traducción futura mediante archivos de recursos externalizados (i18n) sin requerir modificaciones de código.

3.2.2. Seguridad

- **RNF-S01 - Confidencialidad**: Autenticación obligatoria mediante JWT (JSON Web Tokens) con expiración configurable (por defecto 24 horas), contraseñas hasheadas con

BCrypt utilizando factor de trabajo mínimo de 10 rounds y comunicación HTTPS obligatoria en ambiente de producción con certificados TLS 1.2+

- **RNF-S02 - Integridad**: Transacciones ACID garantizadas por Entity Framework y SQL Server para operaciones críticas; y validación de datos en múltiples capas: cliente, controladores, y capa de aplicación.
- **RNF-S03 - Autorización Basada en Roles**: Control de acceso granular mediante decoradores `[Authorize]` con roles específicos en controladores bajo el principio de mínimo privilegio: cada rol tiene acceso únicamente a operaciones estrictamente necesarias:
  - **Soporte**: GET + POST/PUT/DELETE en recursos, traslados, personal de apoyo
  - **Hechicero**: GET global + PUT en misiones propias y hechicero propio
  - **Admin**: Acceso completo incluyendo creación de usuarios, asignación de misiones, y eliminación de registros.
- **RNF-S04 - Protección contra Amenazas Comunes**: Prevención de SQL Injection mediante uso exclusivo de queries parametrizadas de Entity Framework, rate limiting en endpoints de autenticación (máximo 5 intentos de login fallidos en 15 minutos por IP) y CORS configurado restrictivamente para permitir solo orígenes autorizados (frontend en localhost:5173-5175 en desarrollo, dominio específico en producción).

3.2.3. Rendimiento

- **RNF-P01 - Tiempos de Respuesta**: Tiempos esperados con percentil 95:
  - Operaciones GET simples (por ID): $< 200ms$.
  - Listados paginados (misiones, hechiceros, maldiciones): $< 500ms$.
  - Operaciones POST/PUT/DELETE: $< 1s$.
  - Generación de reportes analíticos complejos: $< 5s$.
- **RNF-P02 - Escalabilidad**: Soporte para mínimo 100 usuarios concurrentes sin degradación perceptible, con base de datos escalable hasta 100,000+ misiones históricas manteniendo tiempos de respuesta aceptables mediante índices adecuados.
- **RNF-P03 - Optimización de Recursos**: Paginación obligatoria en endpoints que retornan colecciones grandes (límite default 20 items, máximo 100). Caché del lado del cliente mediante TanStack Query con estrategia stale-while-revalidate (datos considerados frescos por 30 segundos).

3.2.4. Diseño e Implementación

- **RNF-D01 - Stack Tecnológico Frontend**:
  - **Framework UI**: React 19.1.1
  - **Lenguaje**: TypeScript 5.6+ con modo estricto habilitado para seguridad de tipos completa

- **Ruteo**: React Router 7.9.5 con rutas protegidas y guardias de roles
- **Estado Servidor**: TanStack Query 5.56.2 para fetching, caching, sincronización y actualización de estado servidor
- **Formularios**: React Hook Form 7.53.0 con validación mediante esquemas Zod 3.23.8
- **Estilización**: Tailwind CSS 4.1.17 con PostCSS
- **HTTP Client**: Axios 1.7.9 con interceptors para manejo de autenticación y errores globales
- **Notificaciones**: Sonner 1.5.0 para toasts consistentes
- **Build Tool**: Vite 5.0+

- **RNF-D02 - Stack Tecnológico Backend**:
  - **Framework**: ASP.NET Core 9.0 con Minimal APIs deshabilitado (uso de controladores tradicionales)
  - **ORM**: Entity Framework Core 9.0.9 con migraciones code-first
  - **Base de Datos**: SQL Server con soporte para transacciones distribuidas
  - **Autenticación**: Microsoft.AspNetCore.Authentication.JwtBearer 9.0.4
  - **Hashing**: BCrypt.Net-Next 4.0.3 para passwords
  - **Documentación API**: Swagger/OpenAPI (Swashbuckle.AspNetCore 9.0.4) con esquemas generados automáticamente
- **RNF-D03 - Arquitectura de Software**:
  - Patrón en capas con 5 niveles (Core, Infraestructura, Aplicación, Web, Frontend) con separación estricta de responsabilidades
  - Inyección de dependencias nativa de ASP.NET Core con registro de servicios como Scoped
  - Patrón Repository para abstracción de acceso a datos con interfaces `I*Repository`
  - Patrón Service Layer para orquestación de lógica de negocio con interfaces `I*Service`
- **RNF-D04 - Control de Versiones**:
  - Git + GitHub para control de versiones con estrategia de branching (main para producción, dev para integración, feature branches para desarrollo)
  - Commits semánticos con prefijos (feat:, fix:, docs:, refactor:, test:)
  - Husky 9.1.7 + lint-staged 16.2.6 para pre-commit hooks con linting automático
  - ESLint 9.39.1 + TypeScript ESLint para análisis estático de código frontend
- **RNF-D05 - Testing**:
  - Infraestructura de testing frontend configurada con Vitest + React Testing Library + jsdom 27.1.0
  - Mock Service Worker (MSW 2.6.7) configurado para burlar llamadas API durante desarrollo

3.3 Requerimientos de Entorno

3.3.1. Hardware del Servidor en desarrollo

- **CPU**: Mínimo 2 núcleos (4 recomendados) x86-64 a 2.5GHz+
- **RAM**: Mínimo 8GB (16GB recomendado para SQL Server + aplicación)
- **Almacenamiento**: 50GB SSD para sistema operativo y aplicaciones, 100GB adicionales para base de datos con proyección de crecimiento
- **Red**: Conexión ethernet 100Mbps+ con latencia $< 50ms$ hacia clientes en red local

3.3.2. Hardware del Servidor en producción

- **CPU**: 4+ núcleos dedicados (8 recomendados para alta concurrencia)
- **RAM**: 16GB mínimo (32GB recomendado) para caché de SQL Server y aplicación
- **Almacenamiento**: 500GB SSD NVMe en RAID 1 para redundancia, con backup automático a storage secundario
- **Red**: Conexión redundante 1Gbps+ con uptime 99.9 %

3.3.3. Software del Servidor

- **Sistema Operativo**: Windows Server 2019/2022 o Linux (Ubuntu 22.04 LTS, RHEL 8+)
- **Runtime**: .NET 9.0 Runtime (ASP.NET Core)
- **Base de Datos**: SQL Server 2019+ (Express para desarrollo, Standard/Enterprise para producción) o compatibles (Azure SQL Database)
- **Reverse Proxy**: IIS 10+ (Windows) o Nginx/Apache (Linux) para SSL termination y load balancing
- **Monitoreo**: Application Insights o similar para telemetría

3.3.4. Cliente

- **Sistema Operativo**: Probado en Windows 10/11. Diseñado para ser multiplataforma (accesible desde cualquier SO con navegador moderno)
- **Navegadores Soportados**:
  - Chrome/Edge (Chromium) 90+ (recomendado y probado)
  - Firefox 88+
  - Safari 14+

    - Opera 75+
- **Resolución Mínima**: 320px ancho (mobile), 1366x768 recomendado para desktop
- **Conexión**: Mínimo 3G (5Mbps), recomendado 4G/WiFi (25Mbps+) para experiencia óptima
- **Dispositivos Móviles**: Diseño responsive compatible con Android 8+ y iOS 13+, optimizado principalmente para desktop

3.3.5. Entorno de Desarrollo

- **IDE Backend**: Visual Studio 2022 17.8+ o JetBrains Rider 2023.3+
- **IDE Frontend**: Visual Studio Code 1.85+ con extensiones (ESLint, Prettier, TypeScript)
- **Node.js**: Versión 18.18+ o 20.9+ LTS para tooling frontend (npm)
- **Git**: 2.40+ para control de versiones
- **SQL Server Management Studio**: 19+ para administración de BD en desarrollo

4 Funcionalidades del Producto

4.1 Diagrama de Casos de Uso

4.2 Descripción de Actores

- **Supervisor**: Asigna misiones, genera reportes, gestiona hechiceros
- **Hechicero**: Consulta misiones asignadas, registra resultados, actualiza estado
- **Personal de Apoyo**: Gestiona traslados, recursos y logística
- **Sistema**: Procesos automáticos (notificaciones, generación de solicitudes)

4.3 Prototipos de Interfaz

5 Enfoque Metodológico

El desarrollo del sistema de Gestión de Misiones adoptó un enfoque metodológico híbrido que combina principios de metodologías ágiles con prácticas de ingeniería de software robustas, adaptándose a las características específicas del contexto académico y los objetivos del proyecto.

5.1 Metodología de Desarrollo

5.1.1. Marco Ágil Adaptado

Se implementó una metodología ágil iterativa-incremental inspirada en Scrum y eXtreme Programming (XP), con las siguientes características:

- **Iteraciones de Desarrollo (Sprints)**: El proyecto se organizó en ciclos de desarrollo de 2 semanas de duración, cada uno enfocado en entregar incrementos funcionales del sistema. Cada iteración incluye una sesión inicial donde se priorizaban aspectos según su valor de negocio y dependencias técnicas, definiendo objetivos claros y alcanzables para la iteración y una demostración de funcionalidades implementadas al final de cada sprint con validación contra criterios de aceptación predefinidos.
- **Desarrollo Incremental**: El sistema se construyó mediante entregas incrementales siguiendo una secuencia lógica:
    1. **Sprint 1-2**: Configuración de infraestructura (repositorio Git con GitHub Projects para planificación, estructura de proyecto en 5 capas, configuración de base de datos SQL Server, migraciones iniciales EF Core) y modelo de dominio core (entidades principales: Mision, Hechicero, Maldicion con propiedades y validaciones básicas).
    2. **Sprint 3-4**: Capa de repositorios con operaciones CRUD básicas implementadas por subequipo backend, implementación de servicios con lógica de negocio, y controladores REST con endpoints fundamentales (GET, POST, PUT, DELETE para entidades principales).
    3. **Sprint 5-6**: Sistema de autenticación JWT con roles persistentes en BD, control de acceso basado en roles, frontend básico con React desarrollado por subequipo frontend (páginas de login/registro, routing protegido, componentes de listado con paginación).
    4. **Sprint 7-8**: Relaciones complejas entre entidades (HechiceroEnMision, TecnicaMalditaDominada, UsoDeRecurso), gestión de traslados, formularios de creación/edición con validación Zod en frontend, e integración frontend-backend mediante cliente Axios.
    5. **Sprint 9-10**: Sistema de auditoría para trazabilidad de operaciones, paginación cursor-based en endpoints principales, refinamiento de UX con feedback visual y manejo de errores, y preparación de infraestructura para reportes analíticos.
    6. **Sprint 11-12**: Configuración de infraestructura de testing (Vitest, MSW, React Testing Library), corrección de bugs identificados durante pruebas manuales, documentación técnica, y preparación de presentación final del proyecto.
- **Historias de Usuario**: Los requerimientos se expresaron como historias de usuario con formato estándar:

    Como [tipo de usuario], quiero [funcionalidad] para [beneficio/objetivo]

    Cada historia incluía criterios de aceptación específicos y estimaciones de esfuerzo relativo. Por ejemplo:

Como administrador, quiero asignar múltiples hechiceros a una misión especificando roles (participante/encargado) para formar equipos balanceados según grado de amenaza.

Criterios de aceptación: (1) Interfaz permite seleccionar hechiceros disponibles, (2) Se valida que al menos un encargado tenga grado alto o especial, (3) Sistema previene asignar hechiceros con misiones activas solapadas, (4) Cambios se persisten correctamente en BD.

5.1.2. Prácticas de Ingeniería

Complementando el marco ágil, se aplicaron prácticas técnicas específicas:

- **Integración Continua (CI)**: Uso de Git con GitHub como plataforma central para versionado de código. Se estableció un flujo de trabajo basado en ramas:
  - `master`: Rama principal con código estable y desplegable
  - `dev`: Rama de integración donde convergen features completados
  - `dev/*`: Ramas efímeras para desarrollo de funcionalidades específicas (ej: `dev/auth-jwt`, `dev/mission-assignment`)

  Pre-commit hooks con Husky + lint-staged aseguran que todo código pusheado cumple estándares de linting.
- **Test-Driven Development (TDD) parcial**: Aunque no se aplicó TDD puro en todo el proyecto, se implementó en componentes críticos:
  - Tests unitarios de servicios backend
  - Tests de componentes React para formularios complejos
  - Mock Service Worker (MSW) para tests de integración frontend-API sin dependencias de backend real
- **Refactoring Continuo**: A medida que el sistema evolucionaba, se destinaba tiempo explícito (aproximadamente 20 % de cada sprint) para mejorar código existente mediante la extracción de lógica duplicada en servicios reutilizables, consolidación de validaciones repetitivas, optimización de queries Entity Framework y mejora de nombres de variables/métodos para reflejar mejor su propósito.

5.2 Proceso de Análisis y Diseño

El análisis y diseño siguieron un enfoque iterativo coordinado con los sprints de desarrollo:

- **Fase de Análisis Inicial (Semana 1-2)**:
  - Talleres de elicitación de requerimientos donde se identificaron actores principales (supervisores, hechiceros, personal de apoyo) y casos de uso críticos
  - Análisis del dominio del problema mediante diagramas conceptuales

- Priorización de requerimientos funcionales según valor de negocio
- Definición de requerimientos no funcionales críticos

- **Diseño de Base de Datos (Semana 3-4)**:
  - Modelado conceptual mediante MERX (Modelo Entidad-Relación eXtendido) identificando entidades principales, cardinalidades, y atributos
  - Normalización hasta 3FN (Tercera Forma Normal) para eliminar redundancias y dependencias transitivas
  - Definición de restricciones de integridad referencial (foreign keys) y restricciones de negocio
  - Implementación mediante migraciones EF Code-First para trazabilidad de cambios en esquema

- **Diseño Arquitectónico (Semana 4-5)**:
  - Selección de arquitectura en capas de 5 niveles justificada por separación de responsabilidades y facilidad de testing
  - Elección de stack tecnológico: ASP.NET Core 9.0 + EF Core por ecosistema maduro y soporte de migraciones; React + TypeScript por seguridad de tipos y ecosistema robusto; SQL Server por capacidades transaccionales
  - Definición de convenciones: versionado API (`/api/v1`), nomenclatura de controladores en español con mapeo a rutas en inglés en frontend, estructura de respuestas REST (objetos para GET by ID, arrays para listados, envelope {`items, nextCursor, hasMore`} para paginación)

- **Diseño Incremental (Durante Sprints)**:
  - Diseño de interfaces de servicios/repositorios antes de implementar cada feature
  - Creación de wireframes de UI antes de implementar componentes React complejos
  - Diseño de esquemas Zod para validación de formularios durante desarrollo de vistas de creación/edición

5.3 Gestión de Configuración y Herramientas

- **Control de Versiones**: Git + GitHub con estrategia de branching (`master` para producción, `dev` para integración, `dev/*` para desarrollo de funcionalidades). Commits semánticos con prefijos `feat:`, `fix:`, `refactor:`, `docs:` para trazabilidad.

- **Planificación de Proyecto**: GitHub Projects como herramienta CASE para gestión de tareas, sprints y tableros Kanban. Issues de GitHub para tracking de features, bugs y deuda técnica.

- **Versionado de Base de Datos**: Migraciones EF Core con nomenclatura timestamp-based generadas automáticamente mediante `dotnet ef migrations add`, aplicadas con `dotnet ef database update` en desarrollo. Permite rollback y forward de esquema con trazabilidad completa.

- **Gestión de Dependencias**:
  - Backend: NuGet packages versionados en `GestionDeMisiones.csproj` con restore automático
  - Frontend: npm con `package.json` y `package-lock.json` para reproducibilidad de builds

5.4 Organización del Equipo

El equipo de 4 integrantes se organizó en dos subequipos especializados trabajando en paralelo con sincronización regular:

- **Subequipo Backend (2 integrantes)**: Responsable de la implementación de capas Core, Infraestructura y Aplicación. Desarrollaron el modelo de dominio con Entity Framework, repositorios con operaciones CRUD, servicios con lógica de negocio, migraciones de base de datos, y sistema de autenticación JWT. Configuraron SQL Server y definieron estructura de API REST.
- **Subequipo Frontend e Integración (2 integrantes)**: Contribuidor a la capa de aplicación y responsable de la capa de presentación con React + TypeScript. Implementaron los componentes UI, routing con protección por roles, formularios con validación Zod, integración con backend mediante Axios, y configuración de MSW para desarrollo con mocks. Coordinaron la integración frontend-backend.

5.5 Justificación del Enfoque

La metodología híbrida adoptada combina prácticas ágiles con planificación iterativa para adaptarse al contexto académico y al tamaño reducido del equipo. Los sprints de dos semanas se alinearon con la disponibilidad del equipo y el calendario de evaluaciones y el intercambio realizado en cada conclusión fue suficiente para garantizar la coordinación. La división en subequipos (backend/frontend), sustentada por contratos REST bien definidos, permitió la paralelización efectiva del trabajo y maximizó los resultados del equipo.

La gestión de riesgos se realizó mediante desarrollo incremental y validación temprana de decisiones arquitectónicas —por ejemplo, la implementación de autenticación JWT en Sprints 5–6 y la paginación en Sprints 9–10— lo que redujo trabajos repetidos costosos. La metodología facilitó además la incorporación de aprendizajes durante el desarrollo y promovió la calidad incremental a través de revisiones de código y actualizaciones continuas para mantener la deuda técnica bajo control. En conjunto, esta combinación resultó adecuada para un proyecto académico como el presente que busca equilibrar aprendizaje, trabajo en equipo distribuido y entrega de un producto funcional con arquitectura sólida.

6 Arquitectura del Sistema

6.1 Visión General de la Arquitectura

El sistema de Gestión de Misiones implementa una arquitectura en capas (N-Layered Architecture) con cinco niveles claramente diferenciados, siguiendo principios de separación de responsabilidades (Separation of Concerns) y diseño orientado al dominio (Domain-Driven Design simplificado). Esta arquitectura se fundamenta en la organización de componentes en capas horizontales donde cada capa tiene una responsabilidad específica y depende únicamente de capas inferiores, evitando acoplamiento circular.

6.1.1. Capas del Sistema

1. **Capa Core (Dominio/Modelos)**: Contiene las entidades de dominio que representan conceptos del negocio con sus propiedades, relaciones y lógica de negocio intrínseca. No tiene dependencias hacia otras capas del sistema, garantizando portabilidad del modelo de dominio. Sus principales características son:
   - Anotaciones de validación con Data Annotations (`[Required]`, `[Range]`, `[DataType]`)
   - Propiedades con lógica de validación interna (ej: validación de `FechaYHoraDeFin` posterior a `FechaYHoraDeInicio` en `Mision`)
   - Enumeraciones para estados y tipos (ej: `EEstadoMision`, `EGrados`, `ETipoMaldicion`)
   - Navegación de relaciones mediante propiedades de colección (`ICollection<>`) y referencias (`Ubicacion? Ubicacion`)
   - Atributo `[JsonIgnore]` para prevenir serialización circular en navegaciones bidireccionales
2. **Capa de Infraestructura (Repositorios)**: Abstrae el acceso a datos y persistencia, implementando el patrón Repository para desacoplar lógica de negocio de detalles de infraestructura. Depende de la capa Core (modelos), Entity Framework Core y SQL Server Provider. Sus principales características incluyen:
   - Uso de operaciones asíncronas (`async/await`) para no bloquear threads
   - Queries LINQ traducidas a SQL por EF Core con prevención automática de SQL Injection
   - Soporte de paginación cursor-based en entidades principales (Misiones, Hechiceros, Maldiciones)
   - Configuración de retry logic con `EnableRetryOnFailure()` para tolerancia a fallos transitorios
   - Migraciones code-first para versionado de esquema de BD
3. **Capa de Aplicación (Servicios)**: Orquesta la lógica de negocio compleja que involucra múltiples entidades o repositorios, implementa reglas de negocio transversales y coordina transacciones. Depende de las capas Core e infraestructura (interfaces de repositorios) y de librerías como BCrypt para hashing y JWT para tokens. Se caracteriza por:

- Métodos de servicio que coordinan múltiples operaciones de repositorio en transacciones implícitas de EF
- Validación de reglas de negocio complejas (ej: verificar que hechicero no tenga misiones solapadas antes de asignar)
- Transformaciones de datos entre modelos de dominio y DTOs cuando las estructuras difieren
- Manejo de excepciones de negocio con mensajes descriptivos
- Inyección de dependencias de repositorios mediante inyeccion de constructores

4. **Capa Web (Controladores REST)**: Expone las funcionalidades del sistema mediante API REST, maneja requests HTTP, valida entradas, delega procesamiento a servicios y formatea respuestas. Depende de la capa de Aplicación (interfaces de servicios) y de ASP.NET Core MVC. Incluye el CORS configurado para orígenes específicos (`localhost:5173-5175`) con credenciales, serialización JSON en camelCase para compatibilidad con el frontend, configuración de autenticación JWT con validación de issuer, audience y signing key, y Swagger/OpenAPI para documentación automática de la API. Sus características principales son:

   - Anotaciones de routing: `[Route('api/[controller]')]` con convención global que añade prefijo `/v1` (`RoutePrefixConvention`)
   - Verbos HTTP explícitos: `[HttpGet]`, `[HttpPost]`, `[HttpPut]`, `[HttpDelete]` con rutas parametrizadas (`[HttpGet("{id}")]`)
   - Validación automática de `ModelState` con retorno de `BadRequest` en caso de violación de Data Annotations
   - Autorización mediante `[Authorize(Roles = 'admin' , 'sorcerer')]` para endpoints protegidos
   - Respuestas REST semánticas: `Ok(data)` para 200, `CreatedAtAction` para 201, `NotFound` para 404, `BadRequest` para 400
   - Soporte de paginación condicional: si el request incluye `limit/cursor`, devuelve envelope {`items, nextCursor, hasMore`}; si no, devuelve el array completo

5. **Capa Frontend (Interfaz de Usuario)**: Proporciona una interfaz de usuario reactiva y responsiva, gestiona el estado del cliente, realiza validaciones de entradas, y se comunica con el backend mediante API REST. Depende de la capa Web y de las teconologías enumeradas en secciones anteriores. Tiene una Component-Based Architecture con separación de responsabilidades mediante contenedores/presentadores. El flujo de datos comienza con la interacción del usuario con un componente, tras lo que React Hook Form valida los datos con esquema Zod y de ser validados, invoca la función de la API correspondiente; el cliente Axios envía el request al backend, que lo procesa, revalida, persiste en la BD, y devuelve una respuesta que es mostrada por la UI mediante un toast de éxito o error. Se caracteiza por:

   - SPA (Single Page Application) con client-side routing sin recargas completas de página
   - Actualizaciones optiistas de la UI ante acciones del usuario mientras el request está siendo procesado.

- Limita errores para prevenir fallos de la aplicación completa ante errores de componentes individuales
- Cargado perezoso de rutas con React.lazy + Suspense para reducir el tamaño de carga inicial

6.2 Diagrama de la Arquitectura

6.3 Justificación de la selección

La arquitectura en capas de cinco niveles fue seleccionada por las siguientes razones estratégicas:

- **Separación de Responsabilidades (SoC)**: Cada capa tiene un propósito único bien definido, lo que facilita la comprensión del sistema, la incorporación de nuevos desarrolladores y la localización de bugs. Los cambios en UI no afectan la lógica de negocio, los cambios en infraestructura de BD no impactan controladores, etc.
- **Testabilidad**: Interfaces explícitas entre capas permiten pruebas aisladas mediante mocks/stubs. Los servicios pueden testearse independientemente de la BD usando repositorios mock y los controladores sin servicios reales. TanStack Query y MSW facilitan el testing del frontend sin el backend.
- **Mantenibilidad**: Las modificaciones localizadas en una capa no requieren cambios en cascada. Por ejemplo, una migración de SQL Server a PostgreSQL solo afecta los repositorios; un cambio de React a Vue solo afecta el frontend sin tocar el backend.
- **Seguridad por Capas (Defense in Depth)**:
  - Frontend: La validación Zod previene el envío de datos inválidos
  - Web: Las Data Annotations validan el formato de los requests
  - Aplicación: Los servicios validan reglas de negocio complejas
  - Infraestructura: EF impide SQL Injection y los constraints de la BD garantizan integridad
- **Independencia de Infraestructura**: El patrón Repository abstrae detalles de persistencia: Cambiar de EF a Dapper o de SQL Server a Cosmos DB solo requiere reimplementar repositorios sin afectar los servicios o controladores.
- **Evolución Independiente**: Varios equipos pueden trabajar en capas diferentes sin conflictos frecuentes. Por ejemplo, el frontend puede adoptar nuevas librerías UI sin coordinar con el backend, mientras que el backend puede optimizar queries sin notificar al frontend siempre que los contratos REST no cambien.

6.4 Patrones de Diseño Aplicados

7 Patrones de Diseño, Visualización y Datos

7.1 Patrones Arquitectónicos

- **Repository Pattern**: Abstracción de acceso a datos con interfaces genéricas (`IRepository<T>`) e implementaciones concretas que encapsulan queries EF. Facilita el testing con repositorios in-memory y permite cambiar tecnología de persistencia sin afectar consumidores.
- **Service Layer Pattern**: Capa intermedia que orquesta operaciones complejas coordinando múltiples repositorios y aplicando reglas de negocio transversales. Evita que los controladores contengan lógica de negocio (thin controllers).
- **Dependency Injection (DI)**: Inversión de control mediante contenedor DI nativo de ASP.NET Core. Los servicios y repositorios se registran como interfaces con implementaciones concretas inyectadas en constructores. Facilita testing y desacoplamiento.
- **MVC (Modelo-Vista-Controlador)**: Separación clara de responsabilidades.

7.2 Patrones de Diseño

- **Factory Method**: Generación de tokens JWT en `AuthService` mediante un método centralizado que encapsula la creación de `JwtSecurityToken` con claims específicos.
- **Singleton**: DbContext se registra como Scoped (instancia única por request HTTP) para garantizar consistencia transaccional dentro de cada operación.
- **Template Method (en testing)**: Los controladores MSW definen un template de respuestas mock con estructura común pero datos específicos por entidad.

7.3 Patrones de Visualización

- **Container/Presenter (Smart/Dumb Components)**: Separación entre componentesn y contenedores que gestionan el estado y la lógica (ej: `MissionsPage` con `useQuery`) y componentes de presentación puros que solo renderizan (ej: `MissionList` que recibe array de misiones y callbacks).
- **Higher-Order Components (HOC)**: `ProtectedRoute` y `RoleGuard` envuelven componentes añadiendo lógica de autorización sin modificar componentes internos.
- **Hooks Pattern**: Hooks personalizados encapsulan lógica reutilizable (ej: `useAuth()` para acceder a contexto de autenticación, `useMissions()` para queries de misiones con TanStack Query).
- **Responsive Design**: Adaptación a diferentes dispositivos según sus características.

7.4 Patrones de Datos

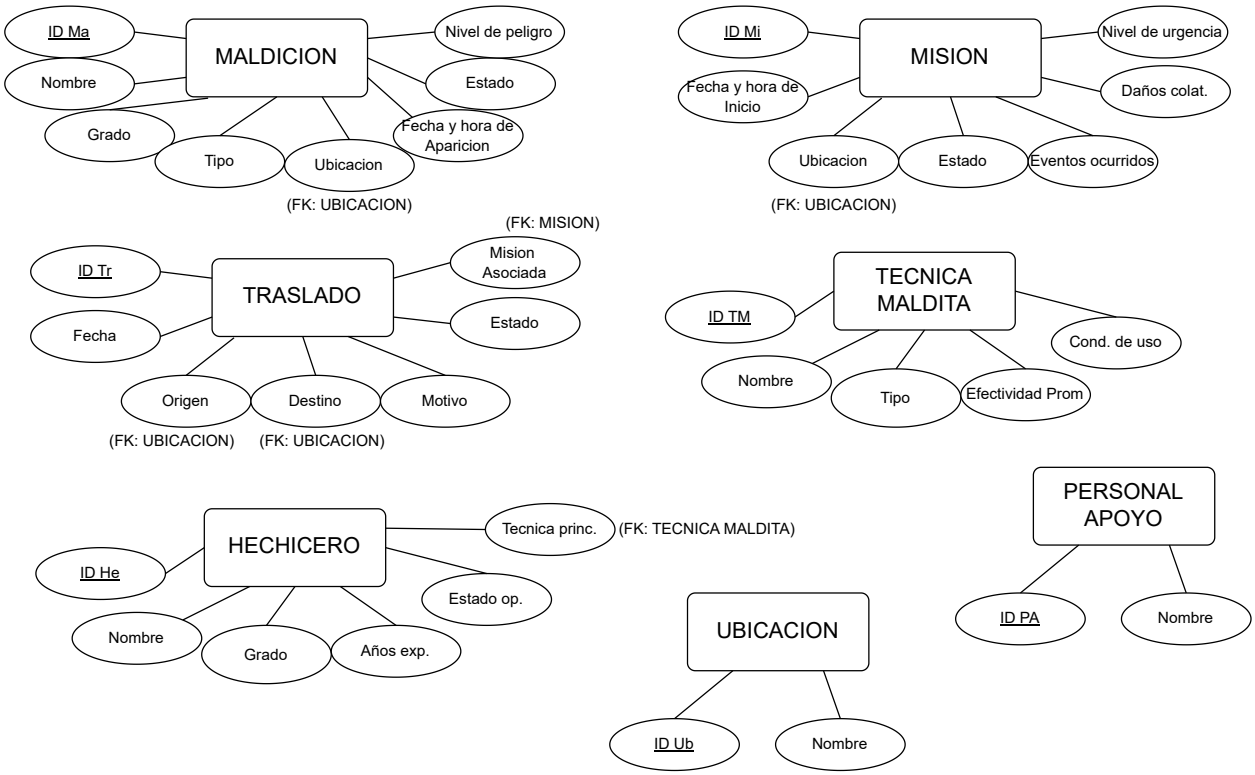
- **Active Record**: Mapeo objeto-relacional simple
- **Data Mapper**: Separación entre modelo de dominio y persistencia
- **Unit of Work**: Gestión de transacciones complejas

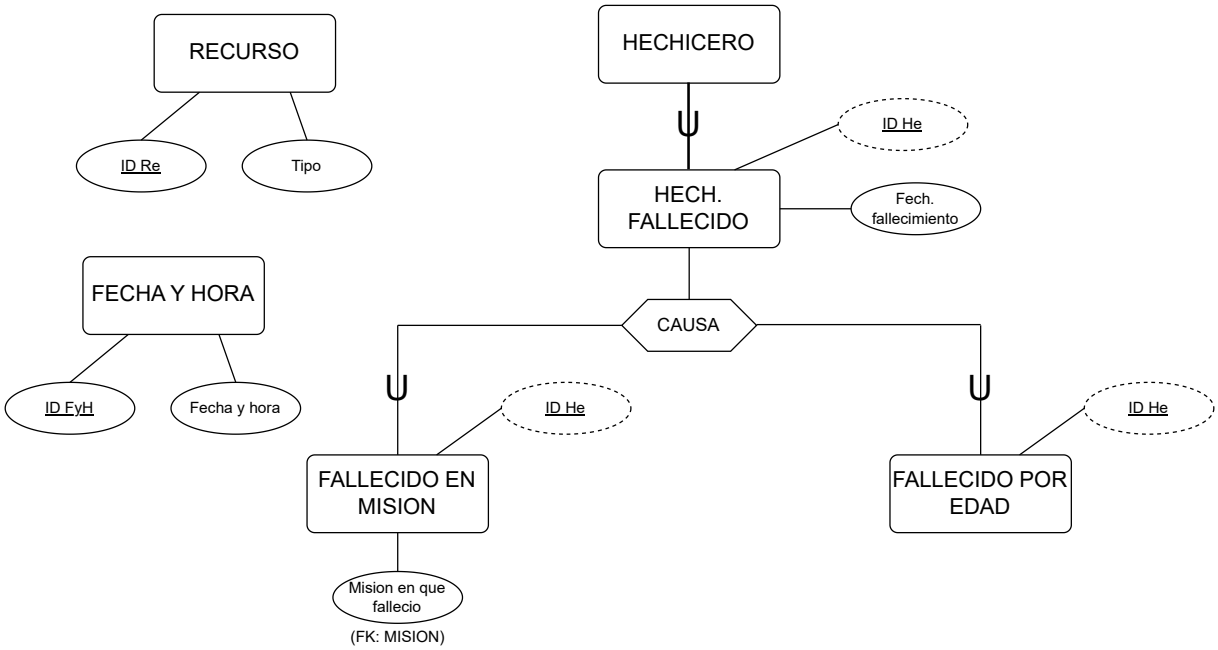
8 Modelo de Datos

8.1 Modelo Conceptual Relacional-Extendido (MERX)

El Modelo Entidad-Relacionalidad Extendido (MERX) consituye una forma intuitiva de modelar conceptualmente las interrelaciones entre conjuntos de interés en un dominio específico del conocimiento basada en Teoría de Conjuntos y Programación Orientada a Objetos. Se desarrolla como parte del proceso de diseño conceptual de la base de datos para describir el contenido de la información a almacenar, más que las estructuras de almacenamiento necesarias [**?**]. El modelo debe ser capaz de transmitir la mayor cantidad de información acerca de la realidad que se presenta para guiar el posterior proceso de implementación y desarrollo.

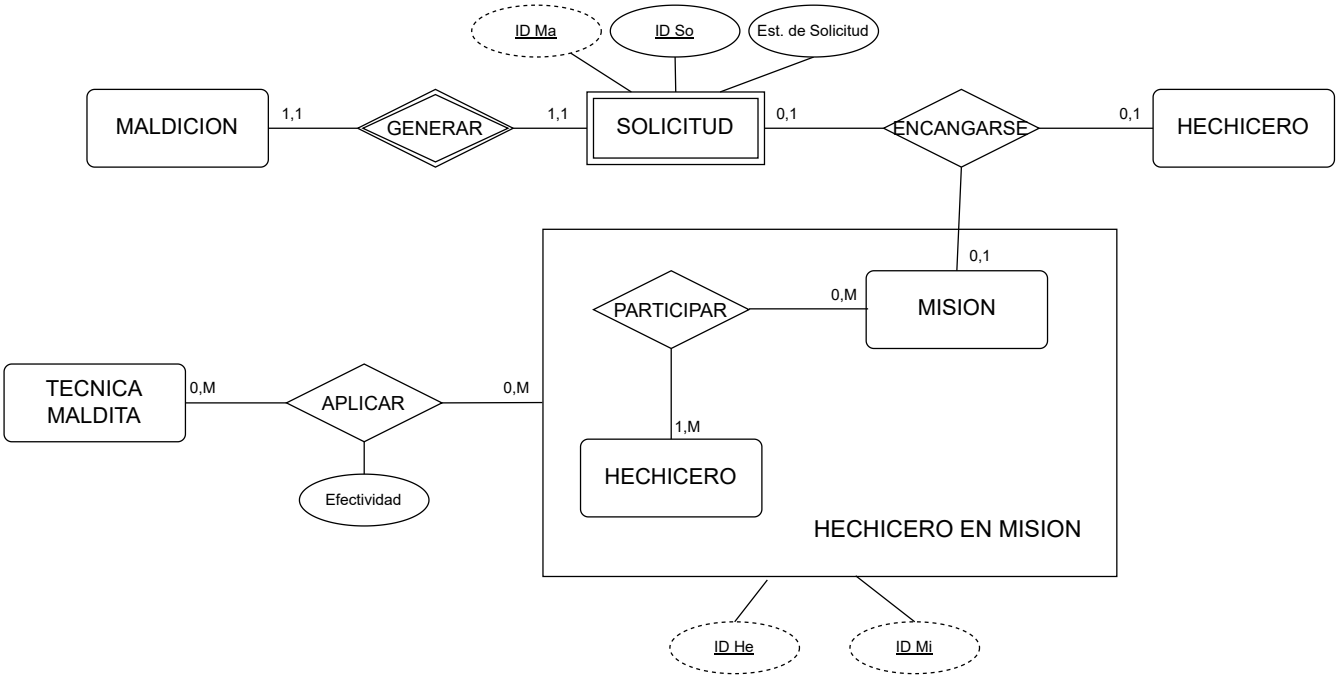
Entidades

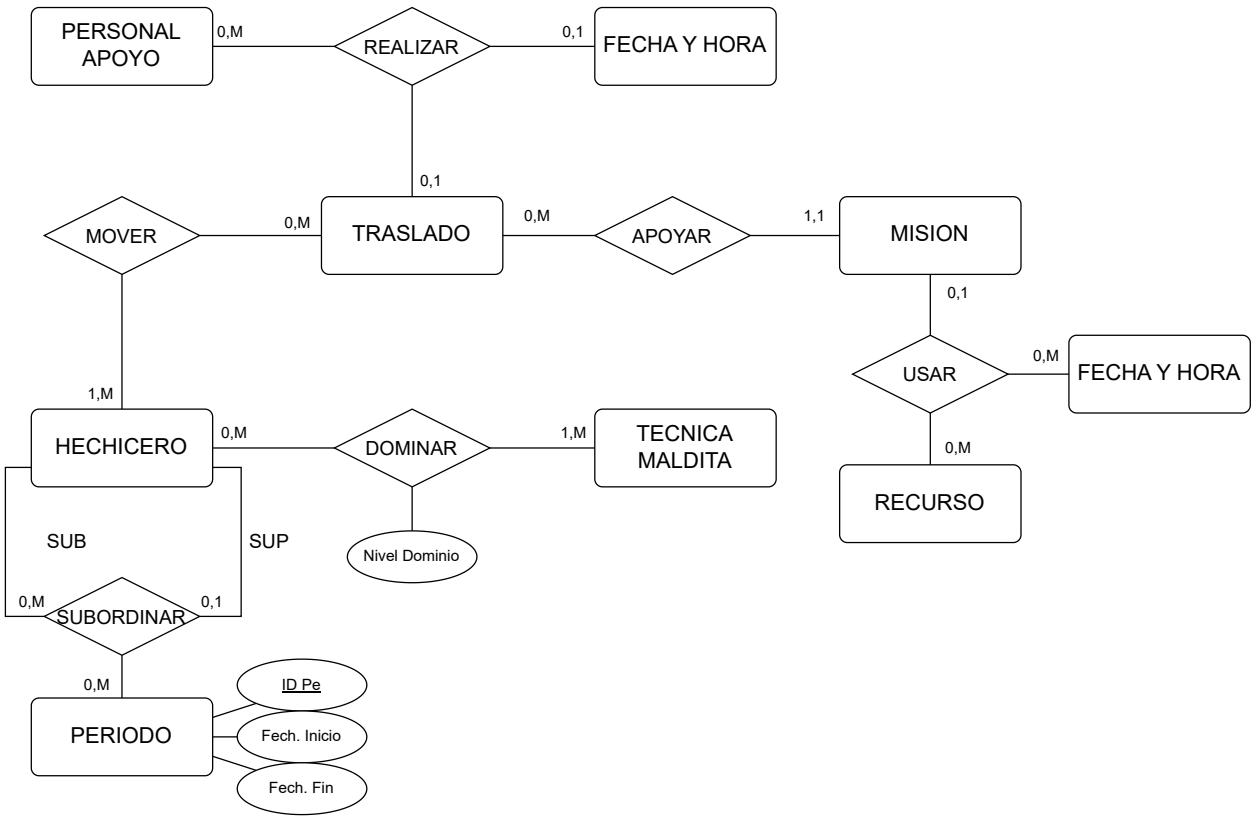

ID Ma
Nombre
Grado
Tipo
Ubicacion
Nivel de peligro
Estado
Fecha y hora de Aparicion
MALDICION
(FK: UBICACION)
ID Mi
Fecha y hora de Inicio
Ubicacion
Estado
Eventos ocurridos
Nivel de urgencia
Daños colat.
MISION
(FK: UBICACION)
ID Tr
Fecha
Origen
Destino
Motivo
Mision Asociada
Estado
TRASLADO
(FK: MISION)
(FK: UBICACION)
(FK: UBICACION)
ID TM
Nombre
Tipo
Efectividad Prom
Cond. de uso
TECNICA MALDITA
ID He
Nombre
Grado
Años exp.
Tecnica princ.
Estado op.
HECHICERO
(FK: TECNICA MALDITA)
ID Ub
Nombre
UBICACION
ID PA
Nombre
PERSONAL APOYO



Nota: En algunos atributos se usa (FK: X), esto representa una interrelacion de cardinalidad uno a muchos. Es decir la entidad X la pueden tener como atributo 0, 1 o muchas entidades asociadas; la entidad que tiene la FK tiene exactamente una entidad X asociada por la interrelacion (tiene exactamente una X como atributo)

Relaciones


ID Ma
ID So
Est. de Solicitud
MALDICION
1,1
GENERAR
1,1
SOLICITUD
0,1
ENCANGARSE
0,1
HECHICERO
0,1
PARTICIPAR
0,M
MISION
TECNICA MALDITA
0,M
APLICAR
0,M
1,M
HECHICERO
Efectividad
HECHICERO EN MISION
ID He
ID Mi


PERSONAL APOYO
0,M
REALIZAR
0,1
FECHA Y HORA
0,1
MOVER
0,M
TRASLADO
0,M
APOYAR
1,1
MISION
0,1
USAR
0,M
FECHA Y HORA
1,M
HECHICERO
0,M
DOMINAR
1,M
TECNICA MALDITA
0,M
RECURSO
SUB
SUP
Nivel Dominio
0,M
SUBORDINAR
0,1
0,M
PERIODO
ID Pe
Fech. Inicio
Fech. Fin

8.2 Correctitud del diseño de la base de datos

Para realizar un diseño correcto de la base de datos (figura 2), primeramente se extraen las dependencias funcionales del MERX (cubrimiento minimal), luego se aplica el algoritmo para obtener una descomposición en 3FN, que garantiza el cumplimiento de la Propiedad de Preservación de Dependencias Funcionales (PPDF), y finalmente se añade una llave del esquema previo a la descomposición para garantizar el cumplimiento de la Propiedad del Join sin Pérdida de Información (PLJ) a través del Lema de Ullman [**?**].

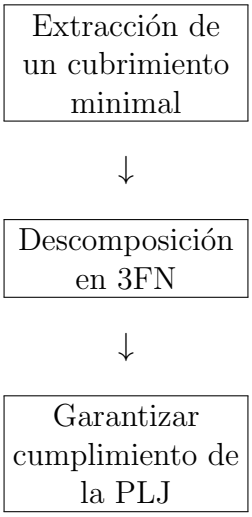


Figura 2: Flujo de obtención de un diseño correcto de la base de datos

8.3 Extracción de un cubrimiento minimal

Para la extracción de un cubrimiento minimal del MERX se sigue el siguiente algoritmo:

1. Por cada conjunto de entidades con un conjunto de atributos $X \subseteq U$, se añade la dependencia funcional $K \to X$ donde K es la llave del conjunto de entidades.
2. Por cada conjunto de interrelaciones se toma su llave K y se añade la dependencia funcional $K \to K$. Además, por cada conjunto de entidades en un extremo de cardinalidad máxima 1 en la interrelación, se añade la dependencia funcional $K - KE \to KE$ donde KE es la llave del conjunto de entidades.
3. Por cada agregación con un conjunto de atributos $X \subseteq U$ se añade la dependencia funcional $K \to X$ donde K es la llave del conjunto de interrelaciones que encierra la agregación.
4. Añadir aquellas dependencias funcionales asociadas a otras restricciones del negocio especificadas en los requerimientos.

8.3.1. Cubrimiento minimal

$idMa \to nMa,\ grMa,\ tMa,\ fMa,\ esMa,\ nP,\ idUb$

$idUb \rightarrow nUb$

$idMi \rightarrow nU,\ dC,\ evO,\ esMi,\ fMi,\ idUb$

$idPA \rightarrow nPA$

$idTr \rightarrow fTr,\ mot,\ esTr,\ idUbO,\ idUbD,\ idPA,\ idMi$

$idTM \rightarrow nTM,\ tTM,\ efProm,\ cUso$

$idHe \rightarrow nHe,\ grHe,\ exp,\ esHe,\ idTM,\ fall,\ fFall,\ idMiF$

$idPe \rightarrow fIn,\ fFin$

$idSo \rightarrow idMa,\ esSol$

$idRec \rightarrow Tipo$

$idFH \rightarrow fH$

$idSo,\ idHe,\ idMi \rightarrow idSo,\ idHe,\ idMi$

$idHe,\ idMi \rightarrow idHe,\ idMi$

$idHe,\ idMi,\ idTM \rightarrow idHe,\ idMi,\ idTM,\ ef$

$idHe,\ idTr \rightarrow idHe,\ idTr$

$idHe,\ idTM \rightarrow idHe,\ idTM,\ nDom$

$idHe1,\ idHe2,\ idPe \rightarrow idHe1,\ idHe2,\ idPe$

$idTr,\ idPA \rightarrow idTr,\ idFH,\ idPA$

$idRec,\ idMi,\ idFH \rightarrow idRec,\ idMi,\ idFH$

8.4 Descomposición en 3FN

Para obtener la descomposición ρ en 3FN del esquema relacional $R(U, F)$, donde F es el cubrimiento minimal obtenido, se usa el siguiente algoritmo:

1. Eliminar cada $X \rightarrow Y$ en F y añadir $X \rightarrow A_i$ para todo $A_i \in Y$.
2. Por cada dependencia funcional $X \rightarrow Ai$ en F crear el esquema relacional $R_i(U_i, F_i)$ tal que $U_i = X \cup \{A_i\}$ y $F_i = \Pi_{R_i}(F)$. Si en F se tiene $X \rightarrow A_1$, $X \rightarrow A_2$,..., $X \rightarrow A_k$ se puede utilizar un esquema relacional de la forma $R_j(U_j, F_j)$ con $U_j = X \cup \{A_1, A_2, ..., A_k\}$ y $F_j = \Pi_{R_j}(F)$.
3. Si en U existe algún atributo que no está contenido en ninguna dependencia funcional de F, este atributo puede formar un esquema relacional por sí mismo.
4. Luego, $\rho = (R_i)_{1 \leq i \leq n}$, tal que los esquemas relacionales $R_i(U_i, F_i)$ están en 3FN con respecto a $\Pi_{R_i}(F)$, $\forall i, 1 \leq i \leq n$.

8.4.1. Descomposición

$\rho = (R_i)_{1 \leq i \leq 15}$

- $R_1 = (U_1, F_1)$
  $U_1 = \{idMa,\ nMa,\ grMa,\ tMa,\ fMa,\ esMa,\ nP,\ idUb\}$
  $F_1 = \{idMa \rightarrow nMa,\ grMa,\ tMa,\ fMa,\ esMa,\ nP,\ idUb\}$
- $R_2 = (U_2, F_2)$
  $U_2 = \{idUb,\ nUb\}$
  $F_2 = \{idUb \rightarrow nUb\}$
- $R_3 = (U_3, F_3)$
  $U_3 = \{idMi,\ nU,\ dC,\ evO,\ esMi,\ fMi,\ idUb\}$
  $F_3 = \{idMi \rightarrow nU,\ dC,\ evO,\ esMi,\ fMi,\ idUb\}$
- $R_4 = (U_4, F_4)$
  $U_4 = \{idPA,\ nPA\}$
  $F_4 = \{idPA \rightarrow nPA\}$
- $R_5 = (U_5, F_5)$
  $U_5 = \{idTr,\ fTr,\ mot,\ esTr,\ idUbO,\ idUbD,\ idPA,\ idMi\}$
  $F_5 = \{idTr \rightarrow fTr,\ mot,\ esTr,\ idUbO,\ idUbD,\ idPA,\ idMi\}$
- $R_6 = (U_6, F_6)$
  $U_6 = \{idTM,\ nTM,\ tTM,\ efProm,\ cUso\}$
  $F_6 = \{idTM \rightarrow nTM,\ tTM,\ efProm,\ cUso\}$
- $R_7 = (U_7, F_7)$
  $U_7 = \{idHe,\ nHe,\ grHe,\ exp,\ esHe,\ idTM,\ fall,\ fFall,\ idMiF\}$
  $F_7 = \{idHe \rightarrow nHe,\ grHe,\ exp,\ esHe,\ idTM,\ fall,\ fFall,\ idMiF\}$
- $R_8 = (U_8, F_8)$
  $U_8 = \{idPe,\ fIn,\ fFin\}$
  $F_8 = \{idPe \rightarrow fIn,\ fFin\}$
- $R_9 = (U_9, F_9)$
  $U_9 = \{idSo,\ idMa,\ esSol\}$
  $F_9 = \{idSo \rightarrow idMa,\ esSol\}$
- $R_{10} = (U_{10}, F_{10})$
  $U_{10} = \{idRec,\ Tipo\ \}$
  $F_{10} = \{\ idRec \rightarrow Tipo\}$
- $R_{11} = (U_{11}, F_{11})$
  $U_{11} = \{idFH,\ fH\ \}$
  $F_{11} = \{\ idFH \rightarrow fH\}$

- $R_{12} = (U_{12}, F_{12})$
  $U_{12} = \{idSo,\ idHe,\ idMi\ \}$
  $F_{12} = \{\ idSo,\ idHe,\ idMi \rightarrow idSo,\ idHe,\ idMi\}$
- $R_{13} = (U_{13}, F_{13})$
  $U_{13} = \{\ idHe,\ idMi\ \}$
  $F_{13} = \{\ idHe,\ idMi \rightarrow idHe,\ idMi\ \}$
- $R_{14} = (U_{14}, F_{14})$
  $U_{14} = \{\ idHe,\ idMi,\ idTM,\ ef\ \}$
  $F_{14} = \{\ idHe,\ idMi,\ idTM \rightarrow idHe,\ idMi,\ idTM,\ ef\ \}$
- $R_{15} = (U_{15}, F_{15})$
  $U_{15} = \{\ idHe,\ idTr\ \}$
  $F_{15} = \{\ idHe,\ idTr \rightarrow idHe,\ idTr\ \}$
- $R_{16} = (U_{16}, F_{16})$
  $U_{16} = \{\ idHe,\ idTM,\ nDom\ \}$
  $F_{16} = \{\ idHe,\ idTM \rightarrow idHe,\ idTM,\ nDom\ \}$
- $R_{17} = (U_{17}, F_{17})$
  $U_{17} = \{\ idHe1,\ idHe2,\ idPe\ \}$
  $F_{17} = \{\ idHe1,\ idHe2,\ idPe \rightarrow idHe1,\ idHe2,\ idPe\ \}$
- $R_{18} = (U_{18}, F_{18})$
  $U_{18} = \{\ idTr,\ idFH,\ idPA\ \}$
  $F_{18} = \{\ idTr,\ idFH,\ idPA \rightarrow idTr,\ idFH,\ idPA\ \ \}$
- $R_{19} = (U_{19}, F_{19})$
  $U_{19} = \{\ idRec,\ idMi,\ idFH\ \}$
  $F_{19} = \{\ idRec,\ idMi,\ idFH \rightarrow idRec,\ idMi,\ idFH\ \}$

8.5 Garantía de cumplimiento de la PLJ

Se añade un conjunto conteniendo la llave del esquema relacional inicial, según lo planteado en el Lema de Ullman (cuadro 3).

Lema de Ullman
Sea ρ una descomposición en 3FN para $R(U, F)$ construida utilizando el algoritmo para obtener una descomposición en 3FN que cumple la PPDF, y sea X una llave del esquema $R(U, F)$. Entonces, $\omega = \rho \cup X$ es una descomposición de $R(U, F)$ con todos sus esquemas relacionales en 3FN que cumple la PPDF, pero que además cumple con la PLJ.

En nuestro diseño, la llave a añadir sería:
$X = \{$ *idSo, idHe, idMi, idTr, idTM, idPA, idUb, idPe, IdRec, IdFH* $\}$,
lo cual completaría el esquema relacional obtenido y garantizaría la correctitud del diseño.

9 Discusión del diseño

9.1 Correctitud

El modelo refleja de manera fiel el dominio del problema, representando las entidades fundamentales: Maldición, Misión, Hechicero, Técnica Maldita, Traslado, Personal de Apoyo, Ubicación y Solicitud de Misión. Asimismo, se contemplan extensiones importantes como la subentidad de Hechicero Fallecido (con sus variantes por misión o por edad), lo que permite dar seguimiento a condiciones específicas descritas en el enunciado. La inclusión de relaciones como Encargarse (para la asignación de misiones), Aplicar (para el uso de técnicas en misiones) y Subordinar (para la jerarquía y análisis de subordinados en caso de fallecimiento) evidencia una correcta interpretación de las reglas del negocio.

9.2 Completitud respecto a los requerimientos

El modelo cubre de forma integral los requerimientos informacionales:

- Se permite el registro detallado de maldiciones con grado, tipo, nivel de peligro, ubicación y estado.
- Se gestionan misiones con urgencia, participantes, supervisor, eventos ocurridos, daños colaterales y relación con la maldición objetivo.
- Se incluye el control de técnicas malditas y su dominio por cada hechicero, así como el registro de su efectividad en combates.
- Se contempla la logística de traslados y la participación del personal de apoyo.
- Se modelan adecuadamente los casos de muerte de hechiceros, tanto en misión como por edad, con los enlaces necesarios a subordinados.

De esta manera, el modelo ofrece la base necesaria para implementar todas las consultas y reportes solicitados (efectividad de técnicas, rankings de éxito, análisis por regiones, etc.).

9.3 Normalización y consistencia

El modelo evita redundancias innecesarias mediante una clara separación de entidades y relaciones. La agregación Hechicero en Misión asegura que los atributos propios de la participación no se mezclen en la entidad principal Hechicero. La entidad débil Solicitud de Misión dependiente de Maldición mantiene coherencia con el flujo real del sistema (detección $\rightarrow$ solicitud $\rightarrow$ asignación $\rightarrow$ misión). Además, los atributos se encuentran correctamente ubicados en cada entidad (p. ej. nivel de peligro en Maldición y nivel de urgencia en Misión).

9.4 Flexibilidad y extensibilidad

El diseño facilita la extensibilidad del sistema. La modelación de Subordinar con periodos de inicio y fin permite mantener históricos de relaciones jerárquicas, lo cual asegura consultas temporales. La existencia de relaciones como Aplicar permite añadir nuevos atributos (p. ej. intensidad de uso, condiciones del combate) sin alterar la estructura central. El enfoque modular garantiza que puedan agregarse nuevos tipos de estados, grados o roles sin romper la integridad del modelo.

10 Conclusiones

El desarrollo del sistema de Gestión de Misiones ha permitido consolidar una visión integral y estructurada que combina principios de ingeniería de software con diseño robusto de bases de datos. A través del proceso de análisis, diseño e implementación parcial realizado, se han alcanzado los siguientes logros fundamentales:

En cuanto a los requerimientos, se logró una especificación completa y no ambigua de 19 requerimientos funcionales que cubren todas las operaciones del sistema. Los primeros 11 requerimientos (RF-01 a RF-11) comprenden las funcionalidades CRUD core del sistema y han sido completamente implementados, incluyendo gestión de entidades principales, sistema de autenticación con control de acceso basado en roles y auditoría de operaciones críticas. Los 8 requerimientos restantes (RF-12 a RF-19) corresponden a consultas analíticas avanzadas y reportes especializados actualmente en fase de desarrollo. Los requerimientos no funcionales definidos aseguran que el sistema cumple con estándares de usabilidad, seguridad y rendimiento adecuados para el contexto operativo, con infraestructura de testing preparada para implementación futura.

En el diseño de bases de datos, el modelo conceptual relacional-extendido desarrollado representa fielmente el dominio del problema, capturando las entidades principales con sus relaciones, cardinalidades y atributos. La normalización hasta la tercera forma normal garantiza la eliminación de redundancias y anomalías, mientras que las restricciones de integridad referencial y de negocio aseguran la consistencia y validez de los datos en todo momento. El diseño implementado mediante migraciones EF Core proporciona trazabilidad completa de evolución del esquema con capacidad de rollback.

En la arquitectura del sistema, la implementación del patrón en cinco capas —Core (dominio), Infraestructura (repositorios), Aplicación (servicios), Web (controladores) y Frontend (React)— junto con la selección tecnológica de ASP.NET Core 9.0 con Entity Framework Core 9.0.9 para el backend y React 19.1 con TypeScript 5.6+ para el frontend, proporciona una base sólida para el desarrollo y mantenimiento del sistema. Esta separación clara de responsabilidades facilita la escalabilidad, el testing unitario y la evolución independiente de cada componente. La organización del equipo en dos subequipos especializados (backend e integración/frontend) permitió paralelización efectiva del trabajo una vez establecidos los contratos REST.

La metodología de desarrollo ágil con GitHub Projects como herramienta CASE permitió una iteración constante sobre el diseño en sprints de 2 semanas, incorporando retroalimentación temprana y asegurando que la solución se alineara con las necesidades reales del dominio problemático. La división en subequipos con sincronización constante facilitó el

conocimiento compartido del código y la calidad incremental.

El estado actual del sistema demuestra ser correcto, completo en su núcleo CRUD y consistente con los requerimientos establecidos, constituyendo una base sólida para las fases de desarrollo restantes. Las funcionalidades core implementadas (RF-01 a RF-11) permiten operación básica del sistema con gestión de todas las entidades principales, autenticación segura con roles diferenciados y auditoría de operaciones. No obstante, se reconoce como trabajo pendiente la implementación de las 7 consultas analíticas especializadas especificadas en la orientación del proyecto (RF-12 a RF-18), la funcionalidad de exportación a PDF (RF-19), y la suite completa de pruebas unitarias automatizadas (infraestructura configurada).

Referencias

[1] García H, L. & Montes de Oca R, M. (2005). *Sistemas de Bases de Datos: Modelación y Diseño - Capítulos para estudiantes*. Versión digital.

[2] Pressman S. R. (2010). *Ingeniería del Software. Un enfoque práctico* (Séptima edición). McGraw-Hill Interamericana Editores, México, D. F.

[3] Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley.