Universidad de la Habana
Facultad de Matemática y Computación
Carrera: Ciencia de la Computación
Año: 3ero
Asignaturas: Ingeniería de Software y Sistemas de Bases de Datos II

(a) Logo

(b) Miniatura

Figura 1: Arte de la aplicación

Sistema de Gestión de Misiones de Jujutsu Kaisen

Informe Técnico

Desarrolladores:

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

El presente documento constituye el informe técnico del Sistema de Gestión de Misiones de Jujutsu Kaisen, una aplicación web diseñada para la administración operativa de misiones, hechiceros y maldiciones. Este informe está orientado a desarrolladores y personal de mantenimiento del sistema, proporcionando documentación detallada sobre la arquitectura de software, el modelo de datos, los patrones de diseño implementados y las consideraciones necesarias para el mantenimiento y extensión del sistema.

Índice

1 Introducción

El Sistema de Gestión de Misiones de Jujutsu Kaisen surge como respuesta a la necesidad de centralizar y optimizar la administración de operaciones de exorcismo dentro del universo narrativo de la serie. El sistema permite gestionar el ciclo completo de las misiones desde la recepción de solicitudes hasta la generación de reportes estadísticos.

Desde el punto de vista arquitectónico, el sistema implementa una arquitectura de cinco capas que separa claramente las responsabilidades: núcleo de dominio, infraestructura de persistencia, lógica de aplicación, capa web de API y cliente frontend. Esta separación permite modificar o reemplazar componentes individuales sin afectar al resto del sistema.

El stack tecnológico seleccionado comprende ASP.NET Core 9.0 con Entity Framework Core para el backend, React 19 con TypeScript para el frontend y SQL Server como sistema gestor de base de datos. La comunicación entre capas se realiza mediante inyección de dependencias y el patrón repositorio, donde cada capa depende de abstracciones (interfaces) en lugar de implementaciones concretas, siguiendo así los principios de inversión de dependencias para garantizar bajo acoplamiento y alta cohesión.

2 Diccionario de Datos

Esta sección describe las entidades del modelo de dominio, sus atributos, tipos de datos y restricciones. El modelo implementa el paradigma Code-First de Entity Framework Core, donde las clases del dominio determinan el esquema de la base de datos.

2.1 Entidades Principales

2.1.1. Hechicero

Representa a los hechiceros del sistema, quienes ejecutan las misiones de exorcismo.

Atributo	**Tipo**	**Descripción**
id	int (PK)	Identificador único generado automáticamente
Nombre	string	Nombre del hechicero (requerido)
Grado	EGrados (enum)	Grado del hechicero: estudiante, aprendiz, medio, alto, especial
Experiencia	int	Años de experiencia del hechicero
Estado	EEstado (enum)	Estado actual: activo, lesionado, recuperandose, baja, inactivo
TecnicaPrincipalId	int? (FK)	Referencia a la técnica principal dominada

El atributo Grado utiliza la enumeración EGrados que define la jerarquía de poder: especial representa el nivel más alto de maestría, seguido de alto, medio, aprendiz y estudiante en orden descendente de competencia.

2.1.2. Maldicion

Representa las entidades malditas que deben ser exorcizadas mediante misiones.

Atributo	Tipo	Descripción
id	int (PK)	Identificador único
Nombre	string	Nombre identificativo de la maldición
FechaYHoraDeAparicion	DateTime	Momento de detección de la maldición
Grado	EGrado (enum)	Peligrosidad: grado_1, grado_2, grado_3, semi_especial, especial
Tipo	ETipo (enum)	Clasificación: maligna, semi_maldicion, residual, desconocida
EstadoActual	EEstadoActual (enum)	Estado: activa, en_proceso_de_exorcismo, exorcisada
NivelPeligro	ENivelPeligro (enum)	Nivel: bajo, moderado, alto
UbicacionDeAparicionId	int (FK)	Referencia a la ubicación donde apareció

El sistema clasifica las maldiciones según su naturaleza mediante ETipo: maligna representa entidades inherentemente hostiles, semi_maldicion corresponde a seres con maldiciones incompletas, residual a remanentes de energía maldita sin conciencia propia, y desconocida para clasificaciones pendientes de análisis.

2.1.3. Mision

Entidad central que representa las operaciones de exorcismo asignadas a los hechiceros.

Atributo	Tipo	Descripción
id	int (PK)	Identificador único
FechaYHoraDeInicio	DateTime	Inicio programado de la misión
FechaYHoraDeFinalizacion	DateTime?	Finalización de la misión (nullable)
UbicacionId	int (FK)	Referencia a la ubicación de la misión
Estado	EEstadoMision (enum)	Estado: EnProgreso, Completada, Fallida
EventosOcurridos	string?	Registro de eventos durante la misión
DannosColaterales	string?	Descripción de daños ocasionados
NivelUrgencia	ENivelUrgencia (enum)	Urgencia: Bajo, Medio, Alto, Critico

La entidad implementa validación de integridad temporal mediante una propiedad con lógica: FechaYHoraDeFin debe ser posterior a FechaYHoraDeInicio y lanza una excepción si se intenta asignar una fecha anterior. Además, el Estado por defecto es Pendiente.

2.1.4. TecnicaMaldita

Define las técnicas de energía maldita que pueden dominar los hechiceros.

Atributo	**Tipo**	**Descripción**
id	int (PK)	Identificador único
Nombre	string	Nombre de la técnica (requerido)
Tipo	ETipoTecnica (enum)	Clasificación: amplificacion, dominio, restriccion, soporte
EfectividadProm	int	Efectividad promedio calculada (0-100), valor por defecto 0
CondicionesDeUso	string?	Requisitos para activar la técnica

ETipoTecnica clasifica las técnicas según su naturaleza: amplificacion (potenciación de energía), dominio (control territorial), restriccion (limitación de poderes) y soporte (asistencia a aliados).

2.1.5. Ubicacion

Representa los lugares geográficos donde ocurren las apariciones y misiones.

Atributo	**Tipo**	**Descripción**
id	int (PK)	Identificador único
Nombre	string	Nombre descriptivo del lugar

2.1.6. Traslado

Registra los movimientos de personal y recursos entre ubicaciones.

Atributo	**Tipo**	**Descripción**
id	int (PK)	Identificador único
Fecha	DateTime	Fecha del traslado
Estado	EEstadoTraslado (enum)	Estado: programado, en_curso, finalizado
Motivo	string?	Razón del traslado
Origen	int (FK)	Ubicación de origen
Destino	int (FK)	Ubicación de destino
MisionId	int? (FK)	Misión asociada (opcional)

2.1.7. Recurso

Define los recursos materiales disponibles para las misiones.

Atributo	Tipo	Descripción
id	int (PK)	Identificador único
Nombre	string	Nombre del recurso
TipoRecurso	ETipoRecurso (enum)	Tipo: EquipamientoDeCombate, Herramienta, Transporte, Suministros
Descripcion	string?	Descripción detallada
CantidadDisponible	int	Unidades disponibles en inventario

2.1.8. PersonalDeApoyo

Representa al personal auxiliar que asiste en las misiones.

Atributo	Tipo	Descripción
id	int (PK)	Identificador único
Nombre	string	Nombre del personal

2.1.9. Solicitud

Representa las peticiones de intervención ante la aparición de maldiciones.

Atributo	Tipo	Descripción
id	int (PK)	Identificador único
MaldicionId	int (FK)	Referencia a la maldición reportada
Estado	EEstadoSolicitud (enum)	Estado: pendiente, atendiendose, atendida

2.1.10. Usuario

Entidad de autenticación y autorización del sistema.

Atributo	Tipo	Descripción
id	int (PK)	Identificador único
Nombre	string	Nombre del usuario (requerido)
Email	string	Correo electrónico único
PasswordHash	string	Hash BCrypt de la contraseña
Rol	string	Rol del usuario (Admin, User)
Rango	string	Rango operativo del usuario
CreadoEn	DateTime	Fecha de creación de la cuenta

Las contraseñas se almacenan utilizando el algoritmo BCrypt con factor de trabajo configurable, nunca en texto plano.

2.2 Entidades de Relación

Las siguientes entidades implementan relaciones muchos a muchos con atributos adicionales.

2.2.1. HechiceroEnMision

Asocia hechiceros con las misiones en las que participan.

Atributo	Tipo	Descripción
id	int (PK)	Identificador único
HechiceroId	int (FK)	Referencia al hechicero
MisionId	int (FK)	Referencia a la misión

2.2.2. HechiceroEncargado

Registra al hechicero responsable de una solicitud y su misión resultante.

Atributo	Tipo	Descripción
id	int (PK)	Identificador único
HechiceroId	int (FK)	Referencia al hechicero encargado
SolicitudId	int (FK)	Referencia a la solicitud
MisionId	int? (FK)	Referencia a la misión generada

2.2.3. TecnicaMalditaDominada

Relaciona hechiceros con las técnicas que dominan y su nivel de maestría.

Atributo	Tipo	Descripción
id	int (PK)	Identificador único
HechiceroId	int (FK)	Referencia al hechicero
TecnicaMalditaId	int (FK)	Referencia a la técnica
NivelDeDominio	int	Porcentaje de dominio (0-100)

2.2.4. TecnicaMalditaAplicada

Registra el uso de técnicas en misiones específicas con su efectividad.

Atributo	Tipo	Descripción
id	int (PK)	Identificador único
TecnicaMalditaId	int (FK)	Referencia a la técnica
MisionId	int (FK)	Referencia a la misión
Efectividad	int	Efectividad registrada (0-100)

2.2.5. UsoDeRecurso

Documenta la utilización de recursos durante las misiones.

Atributo	Tipo	Descripción
id	int (PK)	Identificador único
MisionId	int (FK)	Referencia a la misión
RecursoId	int (FK)	Referencia al recurso
FechaInicio	DateTime	Inicio del uso del recurso
FechaFin	DateTime?	Fin del uso (nullable)
Cantidad	int	Unidades utilizadas
Observaciones	string?	Notas sobre el uso

2.2.6. Subordinacion

Modela la relación maestro-discípulo entre hechiceros.

Atributo	Tipo	Descripción
id	int (PK)	Identificador único
MaestroId	int (FK)	Referencia al hechicero maestro
DiscipuloId	int (FK)	Referencia al hechicero discípulo
FechaInicio	DateTime	Inicio de la relación
FechaFin	DateTime?	Fin de la relación (nullable)
TipoRelacion	ETipoRelacion (enum)	Tipo: Tutoría, Supervisión, LiderazgoEquipo, Entrenamiento
Activa	bool	Indica si la relación está vigente

2.2.7. AuditEntry

Registra las operaciones realizadas en el sistema para auditoría.

Atributo	Tipo	Descripción
Id	int (PK)	Identificador único
Timestamp	DateTime	Momento de la operación
Entity	string	Nombre de la entidad afectada
Action	string	Tipo de acción (Create, Update, Delete)
EntityId	int	Identificador de la entidad afectada
ActorRole	string	Rol del usuario que ejecutó la acción
ActorRango	string	Rango del usuario que ejecutó la acción

3 Arquitectura del Sistema

3.1 Arquitectura de Cinco Capas

El sistema implementa una arquitectura de cinco (seis si se considera la capa de datos separada de la de infraestructura) capas que garantiza la separación de responsabilidades

y facilita el mantenimiento independiente de cada componente.

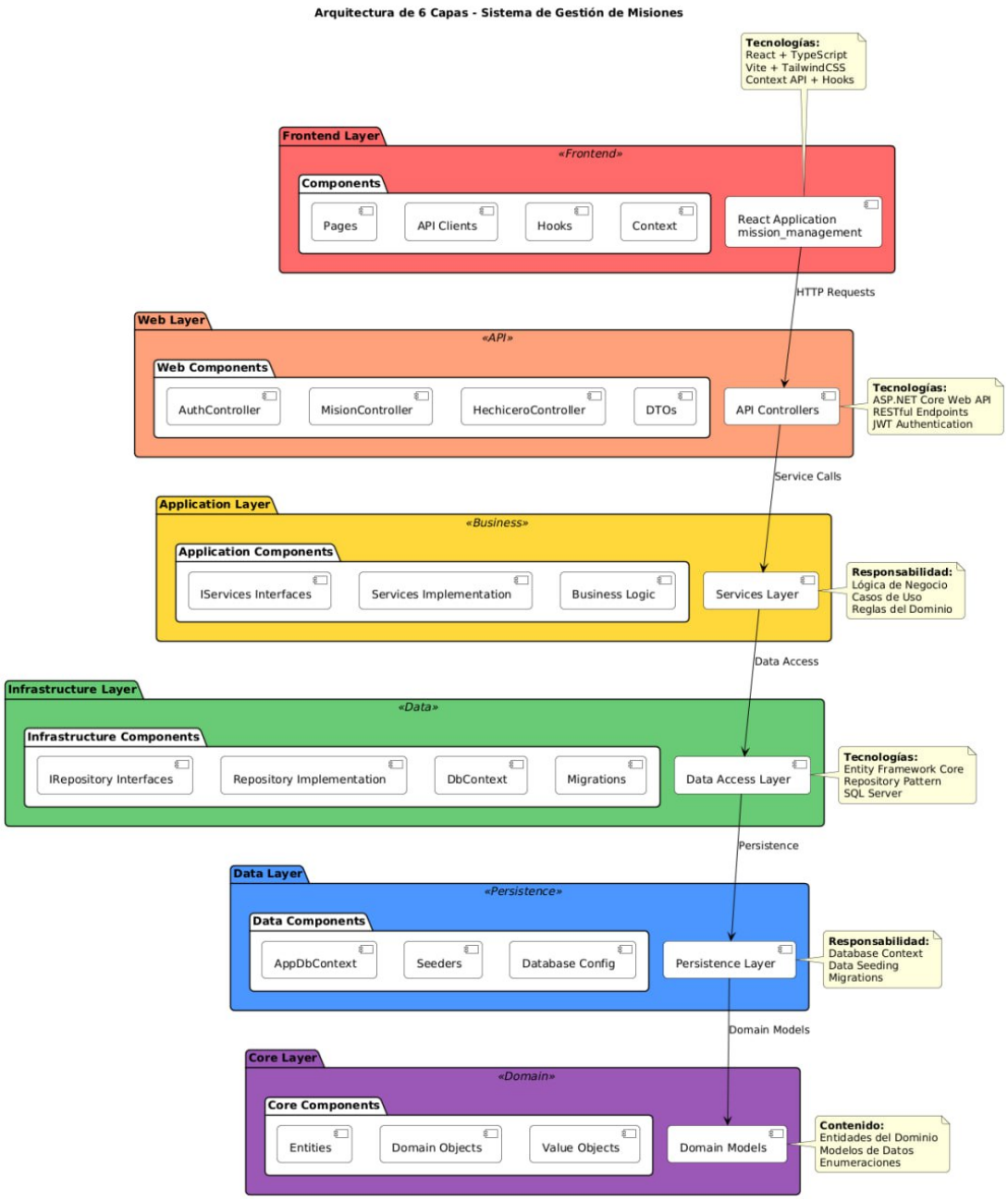


Figura 2: Diagrama de arquitectura del sistema

La capa de Núcleo (Core) contiene las entidades del dominio, enumeraciones y reglas de negocio fundamentales. Esta capa no tiene dependencias externas y define los contratos que el resto de capas deben cumplir.

La capa de Infraestructura implementa el acceso a datos mediante el patrón Repository. Contiene el DbContext de Entity Framework Core, las configuraciones de mapeo objeto-relacional y las implementaciones concretas de los repositorios. La clase AppDbContext centraliza la configuración de las entidades y sus relaciones.

La capa de Aplicación orquesta los casos de uso del sistema. Los servicios de esta capa implementan la lógica de negocio combinando operaciones de múltiples repositorios y

aplicando validaciones transversales. Cada servicio expone una interfaz (IService) que define su contrato público.

La capa Web expone la funcionalidad mediante controladores REST. Los controladores reciben peticiones HTTP, las validan, invocan los servicios correspondientes y transforman las respuestas utilizando DTOs (Data Transfer Objects) para evitar exponer las entidades del dominio directamente.

La capa de Frontend constituye la interfaz de usuario, implementada como una Single Page Application en React. Esta capa consume la API REST y gestiona el estado de la aplicación mediante TanStack Query para el caché de servidor.

3.2 Patrones de Diseño

El sistema aplica varios patrones de diseño empresarial documentados por Fowler [3].

El patrón Repository abstrae el acceso a datos proporcionando una interfaz de colección sobre las entidades del dominio. Cada repositorio genérico implementa operaciones CRUD básicas, mientras que repositorios especializados añaden consultas específicas del dominio.

El patrón Service Layer encapsula la lógica de aplicación en servicios transaccionales. Los servicios coordinan las operaciones entre múltiples repositorios y aplican reglas de negocio que trascienden una única entidad.

El patrón Dependency Injection se utiliza extensivamente mediante el contenedor de servicios de ASP.NET Core. Las dependencias se inyectan por constructor, facilitando las pruebas unitarias y el reemplazo de implementaciones.

El patrón DTO separa la representación interna de las entidades de su representación en la API. Los DTOs de entrada validan los datos recibidos mientras que los DTOs de salida controlan qué información se expone al cliente.

3.3 Diagrama de Clases

El siguiente diagrama presenta las relaciones entre las principales entidades del dominio a través de los casos de uso del sistema.

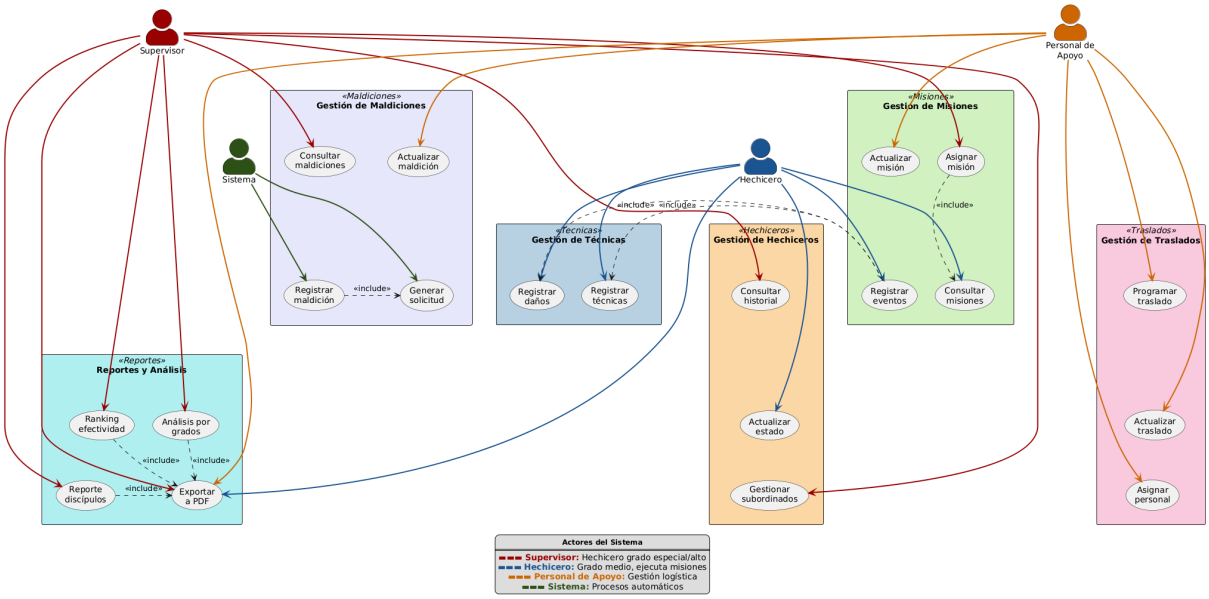
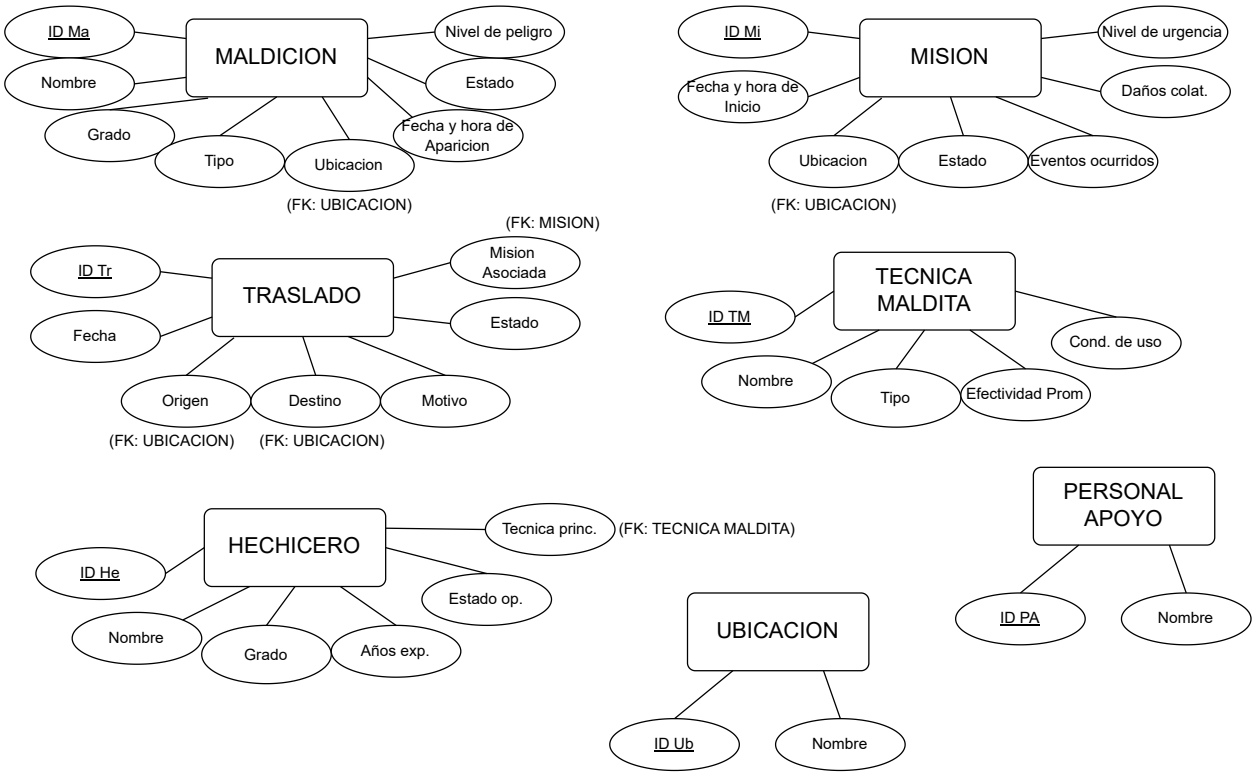


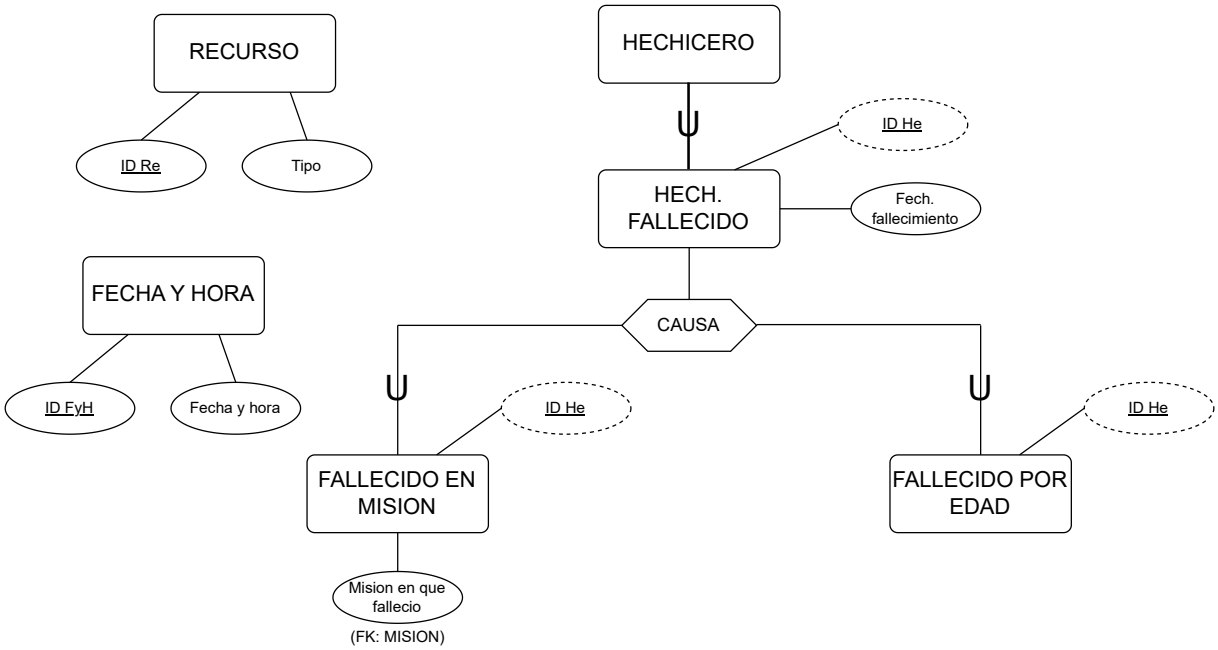
Figura 3: Diagrama de casos de uso

4 Modelo Entidad-Relación Extendido (MERX)

El Modelo Entidad-Relación Extendido (MERX) constituye el núcleo del diseño conceptual de la base de datos. Es una notación formal que permite modelar intuitivamente las interrelaciones entre conjuntos de entidades de interés en el dominio específico, basada en Teoría de Conjuntos y Programación Orientada a Objetos. El MERX describe el contenido de la información a almacenar sin comprometerse con las estructuras de almacenamiento específicas, permitiendo que el modelo transmita la máxima información sobre la realidad del dominio para guiar la posterior implementación.

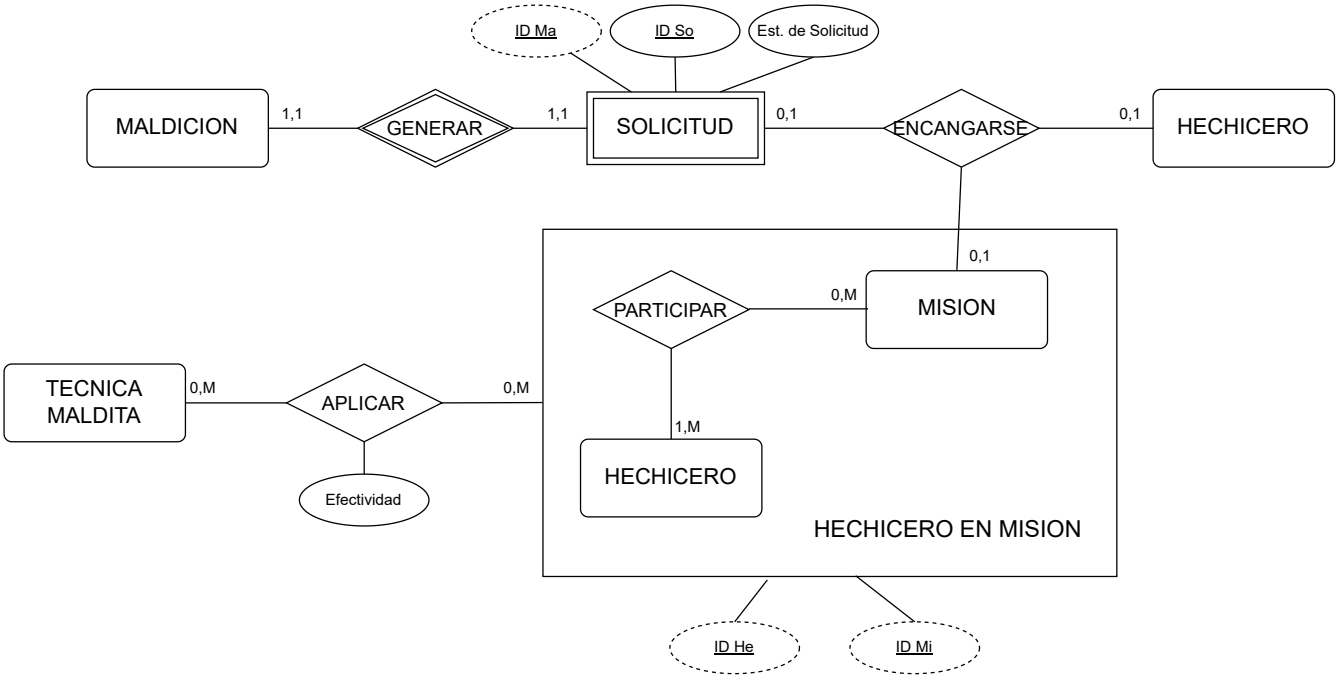
Entidades

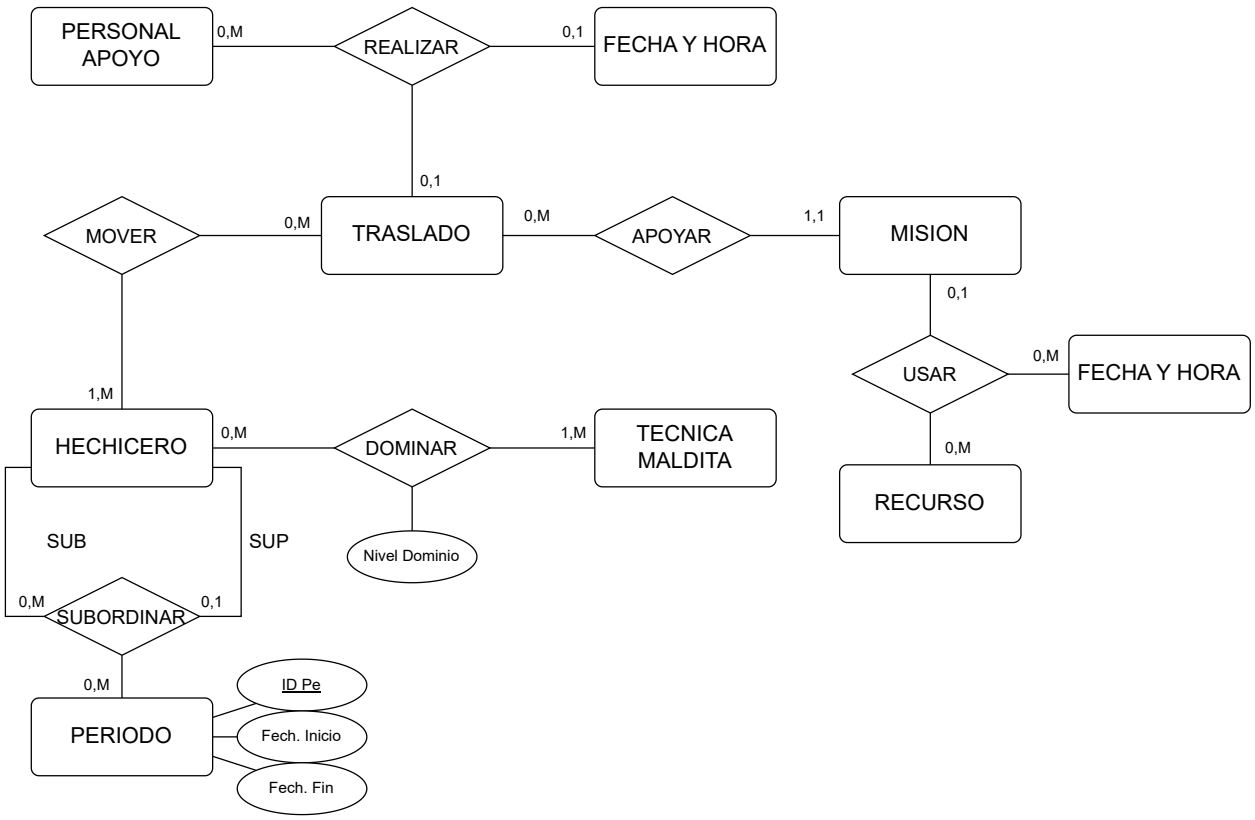

ID Ma
Nombre
Grado
Tipo
Ubicacion
Nivel de peligro
Estado
Fecha y hora de Aparicion
MALDICION
(FK: UBICACION)
ID Mi
Fecha y hora de Inicio
Ubicacion
Estado
Eventos ocurridos
Nivel de urgencia
Daños colat.
MISION
(FK: UBICACION)
ID Tr
Fecha
Origen
Destino
Motivo
Mision Asociada
Estado
TRASLADO
(FK: MISION)
(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

5 Flujo de Trabajo Metodológico

La implementación del sistema siguió una metodología de desarrollo iterativo organizado en fases que garantizan trazabilidad entre requisitos y componentes implementados.

La fase de Planificación estableció el alcance del proyecto, identificó los actores del sistema y definió los requisitos funcionales de alto nivel. Durante esta fase se elaboró el documento de visión y se priorizaron las funcionalidades según su valor de negocio.

La fase de Elaboración profundizó en el análisis de requisitos, produciendo los casos de uso detallados y el modelo conceptual del dominio mediante MERX. Se identificaron los requisitos no funcionales relacionados con seguridad, rendimiento y usabilidad.

La fase de Desarrollo implementó los requisitos siguiendo la arquitectura de cinco capas definida. El desarrollo se organizó por iteraciones, donde cada iteración entregaba funcionalidad completa y probada. Se aplicaron principios SOLID y se mantuvieron pruebas automatizadas.

La fase de Prueba verificó que la implementación cumple los requisitos especificados. Se realizaron pruebas unitarias de los servicios, pruebas de integración de la API y pruebas de aceptación de la interfaz de usuario.

La fase de Implantación prepara el sistema para su despliegue en producción. Se documentaron los procedimientos de instalación, configuración y mantenimiento.

6 Solución Propuesta

6.1 Módulos del Sistema

El sistema se organiza en módulos funcionales que corresponden a las áreas de gestión identificadas durante el análisis de requisitos.

El módulo de Gestión de Hechiceros permite registrar, consultar, modificar y dar de baja hechiceros. Incluye la asignación de técnicas malditas dominadas con su nivel de maestría y el establecimiento de relaciones de subordinación entre hechiceros.

El módulo de Gestión de Maldiciones administra el catálogo de maldiciones detectadas. Permite clasificarlas por grado, tipo y nivel de peligro, y mantiene el registro de su ubicación de aparición y estado actual.

El módulo de Gestión de Misiones constituye el núcleo operativo del sistema. Permite crear misiones a partir de solicitudes aprobadas, asignar hechiceros y recursos, registrar el progreso y documentar los resultados incluyendo eventos ocurridos y daños colaterales.

El módulo de Gestión de Solicitudes procesa las peticiones de intervención. Los usuarios con rol apropiado pueden aprobar o rechazar solicitudes, generando automáticamente las misiones correspondientes.

El módulo de Gestión de Recursos controla el inventario de recursos materiales disponibles. Permite registrar el uso de recursos en misiones y mantiene actualizada la cantidad disponible.

El módulo de Reportes genera informes estadísticos y exportaciones en formato PDF. Los reportes incluyen listados de hechiceros por misión, maldiciones por ubicación, efectividad de técnicas y relaciones maestro-discípulo.

El módulo de Administración gestiona los usuarios del sistema, sus roles y permisos. Permite crear usuarios, asignar roles y consultar el registro de auditoría.

6.2 Tecnologías Implementadas

El backend utiliza ASP.NET Core 9.0 como framework web, aprovechando su sistema de inyección de dependencias nativo y el middleware pipeline para procesamiento de peticiones. Entity Framework Core 9.0 proporciona el ORM con soporte para migraciones Code-First y consultas LINQ traducidas a SQL.

La autenticación se implementa mediante tokens JWT (JSON Web Tokens) firmados con clave simétrica. Los tokens incluyen claims de identidad, rol y rango del usuario, permitiendo autorización basada en políticas.

La generación de documentos PDF utiliza la librería QuestPDF, que permite definir documentos mediante una API fluent en C#. Los reportes se generan dinámicamente a partir de consultas a la base de datos.

El frontend emplea React 19 con TypeScript para tipado estático. TanStack Query v5 gestiona el estado del servidor con caché automático, invalidación y sincronización en segundo plano. Los formularios utilizan react-hook-form con validación mediante esquemas Zod.

La comunicación cliente-servidor se realiza mediante Axios, con interceptores configurados para adjuntar tokens de autenticación y manejar errores de forma centralizada.

El sistema de internacionalización soporta múltiples idiomas mediante archivos de traducción JSON procesados en tiempo de ejecución.

7 Consideraciones de Mantenimiento

7.1 Estructura del Código

El código fuente se organiza siguiendo las convenciones del stack tecnológico. El backend reside en la carpeta `backend/` con subcarpetas para cada capa: `Core/Modelos` contiene las entidades, `Infrastructure/Repository` los repositorios, `Application/Services` los servicios y `Web/Controllers` los controladores.

El frontend se ubica en `mission_management/src/` con organización por características: `pages/` contiene los componentes de página, `components/` los componentes reutilizables, `api/` los clientes de API, `hooks/` los hooks personalizados y `types/` las definiciones TypeScript.

7.2 Migraciones de Base de Datos

Las modificaciones al esquema de base de datos se gestionan mediante migraciones de Entity Framework Core. Para crear una nueva migración tras modificar las entidades:

```
dotnet ef migrations add NombreMigracion
dotnet ef database update
```

Las migraciones se almacenan en `backend/Migrations/` y deben versionarse junto con el código fuente.

7.3 Configuración

Los parámetros de configuración se definen en `appsettings.json` para el backend y variables de entorno para el frontend. La cadena de conexión a la base de datos, la clave de firma JWT y los orígenes CORS permitidos son configurables sin recompilación.

7.4 Extensibilidad

Para añadir una nueva entidad al sistema:

1. Crear la clase del modelo en `Core/Modelos/`
2. Añadir el DbSet correspondiente en `AppDbContext`
3. Crear la interfaz del repositorio en `Infrastructure/IRepository/`
4. Implementar el repositorio en `Infrastructure/Repository/`
5. Crear la interfaz del servicio en `Application/IServices/`
6. Implementar el servicio en `Application/Services/`
7. Crear los DTOs en `Web/DTOs/`
8. Implementar el controlador en `Web/Controllers/`
9. Registrar las dependencias en `Program.cs`
10. Generar y aplicar la migración de base de datos

7.5 Registro de Auditoría

El sistema mantiene un registro automático de operaciones mediante la entidad AuditEntry. Cada creación, modificación o eliminación de entidades genera una entrada con la marca temporal, la entidad afectada, la acción realizada y el usuario responsable.

Para consultar el historial de auditoría, el módulo de administración proporciona filtros por fecha, entidad y tipo de acción.

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.

[4] Microsoft. (2024). *ASP.NET Core Documentation*. `https://docs.microsoft.com/aspnet/core`

[5] Meta. (2024). *React Documentation*. `https://react.dev`