PROYECTO TERCER CORTE ARQUITECTURA DE SOFTWARE

Julián Eduardo Bocanegra Sánchez

Laura Daniela López Díaz

Juan José Mora Carrascal

Facultad de Ciencias Naturales e Ingeniería

Universidad de Bogotá Jorge Tadeo Lozano

Arquitectura de Software

Felipe Esteban Baquero Hernández

1 de noviembre de 2025

Tabla de contenido

INTRODUCCIÓN

Este proyecto implementa un sistema monolítico cliente-servidor para la gestión de ninjas, aldeas y misiones inspirado en el universo de Naruto.

El sistema fue desarrollado en Java Spring Boot, aplicando los principios de Programación Orientada a Objetos (POO) y una arquitectura en capas (Controller, Service, Repository, Domain y DTO).

El objetivo principal es permitir al Hokage administrar la información de los ninjas de la aldea, las misiones disponibles, los rangos y las asignaciones, asegurando además que los datos sean persistentes y se puedan exportar mediante reportes en formato

ALCANCE DEL PROYECTO

El sistema cubre la administración completa de entidades clave del mundo ninja:

- Ninjas**:** creación, consulta y listado de todos los ninjas registrados, incluyendo su aldea, estadísticas y jutsus.
- Misiones**:** registro y listado de misiones con su rango, recompensa y rango requerido.
- Asignaciones**:** asignación de misiones a ninjas, validando que el ninja cumpla con el rango necesario.
- Reportes**:** generación automática de un informe de texto (reporte_ninjas_misiones.txt) que consolida toda la información de ninjas, misiones y equipos asignados.

El proyecto garantiza un entorno centralizado y modular, en el que las operaciones principales se ejecutan desde el servidor, y el cliente (por consola o interfaz web mínima) puede interactuar mediante endpoints REST.

MODELADO DEL DOMINIO

El dominio está compuesto por las siguientes entidades principales:

- Ninja**:** representa a los guerreros de la aldea. Contiene su nombre, rango (GENIN, CHUNIN, JONIN), estadísticas (Stats), aldea (Aldea) y lista de jutsus (List<Jutsu>).
- Misión**:** describe las tareas asignables a los ninjas. Contiene título, descripción, rango (RangoMision), recompensa y el rango mínimo requerido (Rango).
- AsignacionMision**:** establece la relación entre ninjas y misiones, incluyendo si fue completada y las fechas de asignación y finalización.
- Aldea**:** almacena el nombre de cada aldea ninja registrada.
- Stats**:** agrupa las estadísticas del ninja (ataque, defensa, chakra).
- ChakraProfile**:** representa las afinidades elementales del ninja (yin, yang, fuego, agua, etc.).
- Jutsu**:** define las técnicas o habilidades especiales de cada ninja, junto con su tipo (JutsuTipo).
- Enums**:**

- Rango: GENIN, CHUNIN, JONIN.
- RangoMision: D, C, B, A, S.
- Elemento: FUEGO, AGUA, TIERRA, VIENTO, RAYO.
- JutsuTipo: NINJUTSU, GENJUTSU, TAIJUTSU, etc.

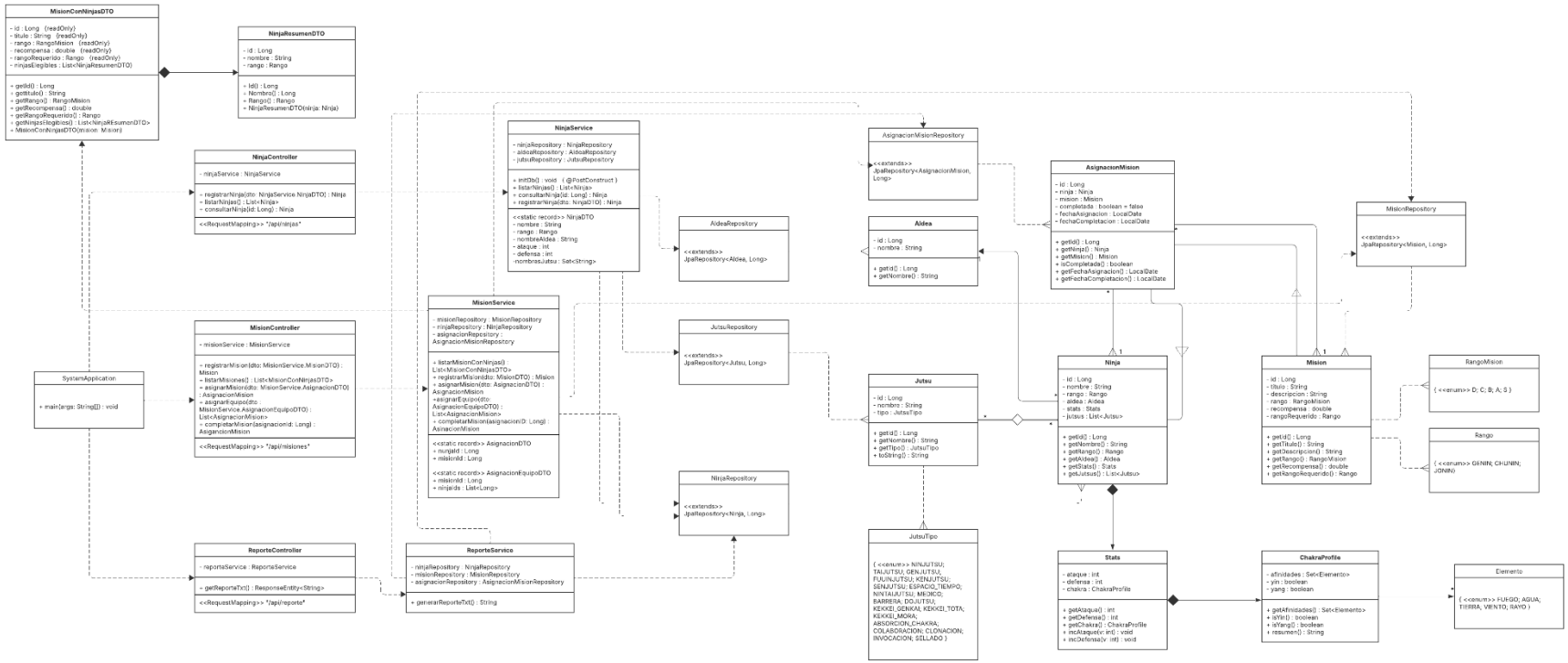
REQUERIMIENTOS FUNCIONALES

1. Gestión de Ninjas
   - Registrar, consultar y listar ninjas.
   - Asociar a cada ninja un rango, aldea, estadísticas y jutsus aprendidos.
2. Gestión de Misiones
   - Registrar y listar misiones disponibles.
   - Establecer la recompensa y rango requerido para cada misión.
3. Asignación de Misiones
   - Asignar ninjas a misiones según su rango.
   - Validar que el ninja cumpla con el rango mínimo requerido.
   - Registrar el estado de la misión (en progreso o completada).
4. Exportación de Información
   - Generar un reporte consolidado en formato .txt que incluya todos los ninjas, misiones y asignaciones.

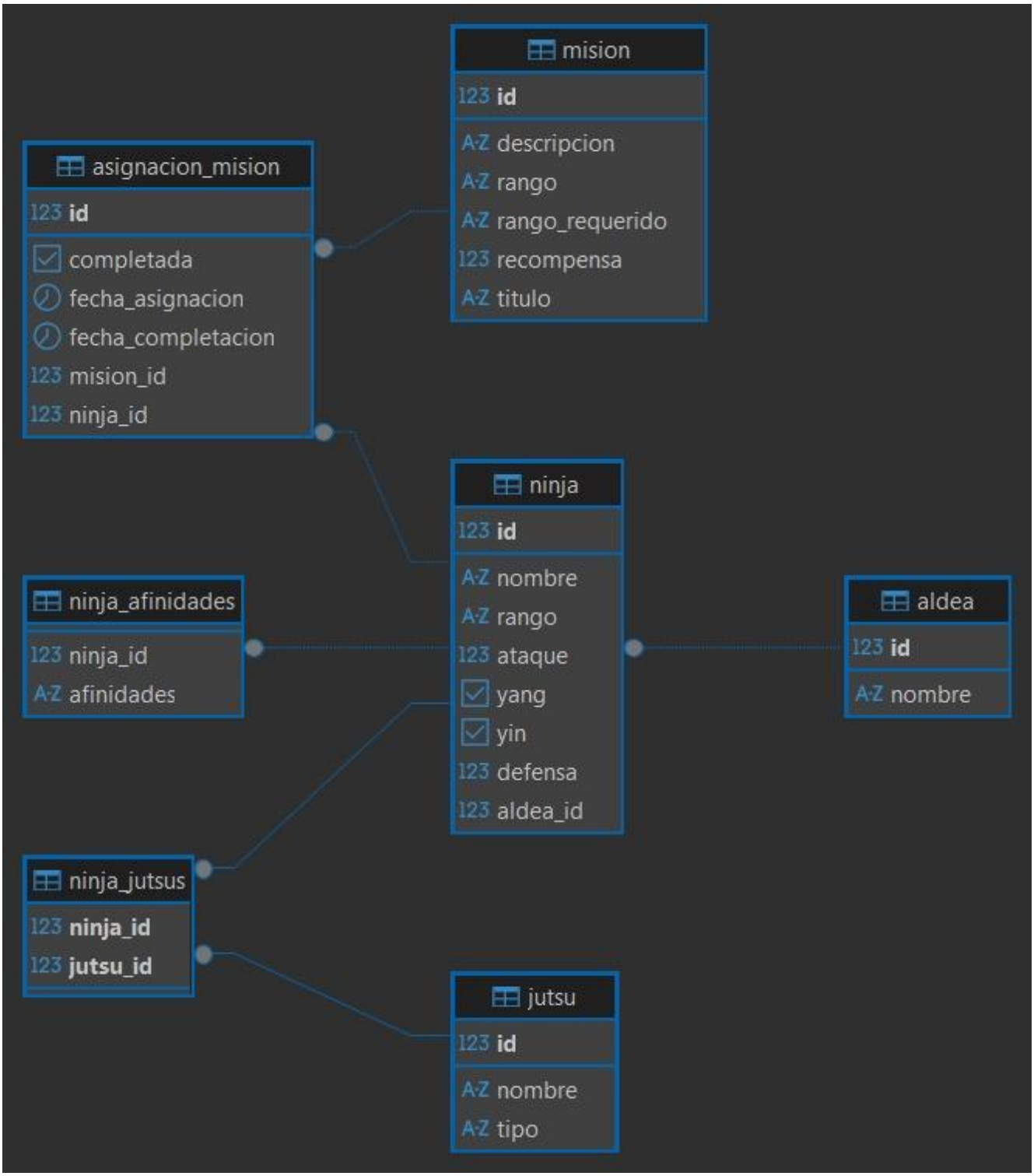
REQUERIMIENTOS TÉCNICOS

1. Arquitectura monolítica cliente-servidor.
2. Implementación monolítica cliente-servidor.
3. Patrón Repository-Service-Controller.
4. Persistencia en base de datos relacional con entidades JPA.
5. DTOs (Data Transfer Objects) para transferir entre capas.
6. Exportación de reportes mediante escritura de archivos en el servidor (java.nio.file).
7. Uso de @Autowired, @Service, @Repository y @RestController.
8. Exposición de endpoints REST:
   - /api/ninjas
   - /api/misiones
   - /api/reporte/txt

DIAGRAMA DE CLASES UML



MODELO ENTIDAD RELACIÓN



CONCLUSIONES

El proyecto demuestra la aplicación correcta de los principios de arquitectura monolítica y programación orientada a objetos, organizando las responsabilidades por capas y utilizando DTOs para aislar la lógica de negocio de la capa de presentación.

Además, la implementación con Spring Boot facilita la inyección de dependencias, la persistencia y la modularidad del código.

La exportación de reportes refuerza la capacidad del sistema para generar documentación dinámica a partir de la base de datos.

En conjunto, el sistema es escalable, mantenible y cumple los requerimientos funcionales y técnicos establecidos por el Hokage.