PRODUCT REQUIREMENTS DOCUMENT

Sistema de Gestión de Laboratorio (SGL)

VERSIÓN	FECHA	AUTOR
1.0	24 de Mayo, 2024	Alan Flores
LABORATORIO	**ESTADO**	
Argon Laboratory	Publicado	

1. RESUMEN EJECUTIVO

El Sistema de Gestión de Laboratorio (SGL) es una aplicación backend robusta, diseñada para centralizar y gestionar las operaciones clave de un entorno de investigación. El sistema permite el registro y seguimiento de investigadores y sus experimentos, culminando en un proceso automatizado de generación de reportes de rendimiento mensual.

Este documento detalla los requisitos funcionales, la arquitectura técnica y los flujos de trabajo de la plataforma, sirviendo como una fuente única de verdad para todas las partes interesadas (stakeholders). El objetivo principal es proporcionar una solución escalable que garantice la integridad de los datos, automatice tareas repetitivas y ofrezca información valiosa sobre el rendimiento del laboratorio.

2. OBJETIVOS DEL PRODUCTO

Centralización de Datos: Establecer una única fuente de verdad para toda la información relacionada con investigadores y experimentos, eliminando la dependencia de hojas de cálculo o sistemas dispares.

Integridad de Datos: Asegurar que los datos sean consistentes y fiables mediante validaciones a nivel de API y transacciones de base de datos.

Automatización de Reportes: Eliminar el proceso manual y propenso a errores de generar reportes de rendimiento mensuales, liberando tiempo valioso del personal.

Escalabilidad: Construir una base técnica sólida que pueda crecer para acomodar futuros requisitos, como la gestión de inventario, presupuestos o dashboards en tiempo real.

3. ALCANCE DEL PRODUCTO

Funcionalidades Incluidas

- ✓ Gestión de Investigadores (CRUD)
- ✓ Gestión de Experimentos (CRUD)
- ✓ Sistema de Eventos Asíncronos
- ✓ Proceso Batch de Reporte Mensual
- ✓ API RESTful Documentada

Funcionalidades Excluidas

- ✗ Interfaz de Usuario (UI/Frontend)
- ✗ Autenticación y Autorización
- ✗ Gestión de Inventario
- ✗ Dashboards en tiempo real

4. ARQUITECTURA TÉCNICA Y PILA TECNOLÓGICA

El sistema está construido sobre una arquitectura de microservicios moderna, utilizando tecnologías probadas para garantizar rendimiento y mantenibilidad.

COMPONENTE	TECNOLOGÍA/LIBRERÍA	PROPÓSITO
Lenguaje	Java 17+	Lenguaje de programación principal
Framework	Spring Boot 3.x	Desarrollo rápido y robusto de la aplicación
API	Spring Web (MVC)	Creación de endpoints RESTful
Documentación API	Springdoc-OpenAPI v2	Generación automática de documentación (Swagger UI)
Base de Datos (Transaccional)	Spring Data JPA / Hibernate	Persistencia de entidades principales
Base de Datos (Documental)	Spring Data MongoDB	Almacenamiento de reportes mensuales
Procesamiento por Lotes	Spring Batch	Orquestación del job de reporte mensual
Eventos de Aplicación	Spring Application Events	Comunicación desacoplada entre componentes
Base de Datos Relacional	MySQL 8.x	Almacén de datos primario
Base de Datos NoSQL	MongoDB	Almacén para documentos flexibles
Build Tool	Apache Maven	Gestión de dependencias y ciclo de vida
Utilidades	Lombok	Reducción de código repetitivo (boilerplate)

5. FLUJO DE TRABAJO Y FUNCIONALIDADES

5.1. Flujo de Datos End-to-End

El flujo de trabajo completo, desde la entrada de datos hasta la generación de valor, sigue estos pasos:

1. Población de Datos (Onboarding)

Investigadores: Carga masiva de perfiles utilizando archivo CSV y Postman Collection Runner, consumiendo el endpoint POST /api/investigators.

Experimentos: Segunda carga masiva asociando cada experimento a un investigador existente a través de su licenseNumber usando POST /api/experiments. Los datos permiten especificar el estado (PLANNING, COMPLETED, etc.) para simular un entorno realista.

2. Operación Diaria

Los usuarios interactúan con los endpoints para crear, actualizar y consultar investigadores y experimentos según sea necesario. El estado de los experimentos se actualiza a lo largo de su ciclo de vida (ej. de IN_PROGRESS a COMPLETED).

3. Generación de Reporte Mensual

Un administrador o proceso automatizado realiza una petición POST al endpoint /api/jobs/launch/monthly-report. Esto dispara el job de Spring Batch, que ejecuta el flujo de reporte de forma asíncrona en segundo plano.

5.2. Detalle del Proceso de Reporte (Spring Batch)

Este es el corazón de la lógica de negocio automatizada:

Paso 1: Disparo (Trigger)

El JobLauncherController recibe la petición y utiliza el JobLauncher de Spring para iniciar el monthlyReportJob.

Paso 2: Lectura (Reader)

El reportStep del job activa el MonthlyExperimentReader. Este lector utiliza un JpaPagingItemReader para conectarse a MySQL y ejecutar una consulta JPQL optimizada para leer, de forma paginada, únicamente los experimentos con status COMPLETED.

Paso 3: Procesamiento (Writer/Aggregator)

Los experimentos se pasan en lotes al DepartamentalReportWriter, que agrega los datos en un Map en memoria. Para cada experimento, se obtiene la Specialization del investigador asociado y se calcula una puntuación ponderada basada en el levelRisk (HIGH=3 pts, MEDIUM=2 pts, LOW=1 pt).

Paso 4: Finalización y Escritura (Listener)

Una vez procesados todos los experimentos, el JobCompletionNotificationListener obtiene el Map final y utiliza MongoTemplate para guardar cada entrada como documento en la colección monthly_reports de MongoDB. El documento incluye un ID único (ej. BIOLOGY_2024-04) para evitar duplicados.

6. REQUISITOS NO FUNCIONALES

Rendimiento: Las operaciones de carga masiva deben ser eficientes. El uso de saveAll() en la capa de servicio y el JpaPagingItemReader en el batch garantizan alto rendimiento.

Fiabilidad: El job de Spring Batch es transaccional. Si ocurre un error durante el procesamiento de un chunk, el chunk se revierte, garantizando la consistencia.

Mantenibilidad: El código está estructurado en capas (Controller, Service, Repository) y utiliza DTOs para separar la representación de la API del modelo de dominio, facilitando futuras modificaciones.

Seguridad: Aunque no está en el alcance actual, la arquitectura basada en Spring Boot permite una integración sencilla con Spring Security para implementar autenticación y autorización en el futuro.

7. FUTURAS MEJORAS

Automatización del Job: Implementar @Scheduled para que el job de reporte se ejecute automáticamente el primer día de cada mes, eliminando la necesidad de un disparo manual.

API de Reportes: Crear un nuevo conjunto de endpoints (GET /api/reports) para consultar los reportes almacenados en MongoDB por fecha, especialización, etc.

Notificaciones: Integrar un servicio de correo electrónico para que, al finalizar el job, se envíe un resumen del reporte a los supervisores del laboratorio.