Facultad de Ingeniería
Escuela de Ingeniería

Universidad del Valle – Sede Tuluá

Documento Proyecto Final

AUTOR

Johann Andrey Gonzalez Gonzalez - 2511006

Luis Manuel Cardona Trochez - 2059942

Yessica Fernanda Villa Nuñez - 2266301

DOCENTE

Juan Pablo Pinillos Reina

Tuluá – COLOMBIA

2025–2

Sistema Integral de Gestión de Laboratorios

1. Descripción del Problema

Contexto

Las universidades modernas enfrentan desafíos significativos en la administración de sus laboratorios académicos. Específicamente, la Universidad del Valle (Univalle) requería una solución integral que permitiera centralizar y automatizar los procesos relacionados con la gestión de infraestructura de laboratorios, acceso de estudiantes y control de recursos.

Usuario Final

- **Estudiantes**: Necesitan agendar prácticas, solicitar préstamo de equipos y registrar su asistencia
- **Administradores de laboratorio**: Requieren gestionar espacios, equipos disponibles, horarios y reportes de uso
- **Personal de seguridad/control de acceso**: Necesitan verificar quién ingresa a los laboratorios y cuándo
- **Docentes/Coordinadores**: Requieren reportes sobre el uso de laboratorios y disponibilidad de recursos

Necesidad que Resuelve

Antes de esta aplicación, los procesos eran **manuales, dispersos y sin automatización**:

- Registro de asistencias en papel o sistemas desconectados
- Falta de control sobre préstamo de equipos
- Imposibilidad de consultar disponibilidad de laboratorios en tiempo real
- Ausencia de reportes consolidados sobre uso de recursos
- Integración deficiente con sistemas existentes de autenticación institucional

Esta aplicación **centraliza, automatiza y proporciona visibilidad completa** sobre todas las operaciones del laboratorio.

2. Objetivo General

Desarrollar una aplicación web basada en arquitectura de microservicios que permita gestionar integralmente los laboratorios académicos de la Universidad del Valle,

automatizando procesos de asistencia, agendamiento de prácticas y control de préstamo de equipos, con persistencia de datos en PostgreSQL y despliegue containerizado mediante Docker y Docker Compose.**

3. Objetivos Específicos

- **Diseñar la estructura de base de datos relacional** con tablas para laboratorios, equipos, estudiantes, asistencias, prácticas y préstamos, estableciendo integridad referencial mediante claves foráneas.
- **Implementar una API REST completa** con endpoints CRUD para cada entidad del sistema (laboratorios, usuarios, equipos, prácticas y préstamos).
- **Desarrollar un servicio de autenticación** integrado con los sistemas de la universidad, implementando JWT para autorización y control de acceso basado en roles.
- **Integrar microservicios con patrón API Gateway** para centralizar las solicitudes y enrutar el tráfico hacia los servicios correspondientes (autenticación, laboratorios, prácticas, equipos).
- **Implementar control de acceso mediante roles y permisos** que permita navegación dinámica en el frontend según el tipo de usuario (estudiante, administrador, docente).
- **Crear una interfaz web responsiva con Angular** que incluya listados, formularios interactivos, sistema de navegación dinámico y visualización de reportes.
- **Contenerizar todos los servicios con Docker** y orquestar su despliegue mediante Docker Compose para garantizar reproducibilidad y facilitar la ejecución del sistema completo.
- **Automatizar la inicialización de datos** mediante un Data Initializer que cargue configuraciones por defecto, laboratorios, equipos y usuarios de prueba en la base de datos.
- **Documentar todos los endpoints** mediante Swagger/OpenAPI para facilitar el consumo de la API y la integración futura con otros sistemas.

4. Justificación de la Aplicación

¿Por qué se eligió esta temática?

Se eligió el tema de **gestión de laboratorios académicos** porque:

1. **Necesidad Real Institucional**: La Universidad del Valle manifestó la necesidad explícita de un sistema que centralizara la administración de sus laboratorios.

2. **Complejidad Académica Apropiada**: El dominio del problema permite demostrar competencias en:
    - Arquitectura de microservicios
    - Integración con sistemas legacy
    - Manejo de múltiples roles y permisos
    - Procesos de negocio complejos (reservas, préstamos, reportes)
3. **Aplicabilidad Transferible**: Los patrones implementados pueden replicarse en otros sistemas institucionales (salas de videoconferencia, recursos bibliográficos, equipos científicos, etc.).

¿Qué problema real resuelve?

- **Automatiza procesos manuales**: Eliminó la necesidad de registros en papel o sistemas desconectados
- **Proporciona control de acceso**: Integra autenticación institucional con autorización granular
- **Mejora la trazabilidad**: Registra quién accedió, cuándo y qué equipos utilizó
- **Facilita la planificación**: Los administradores pueden ver disponibilidad de laboratorios y equipos en tiempo real
- **Genera inteligencia de datos**: Permite reportes sobre uso, patrones de asistencia y recursos más demandados

¿Quién lo usaría?

1. **Estudiantes de Ingeniería**: Agendan prácticas, solicitan equipos, registran asistencia
2. **Administradores de Laboratorio**: Gestionar espacios, equipos, horarios, ver reportes
3. **Personal de Control de Acceso**: Validar identidad usando lectores de código de barras
4. **Docentes/Coordinadores**: Consultar reportes de asistencia y uso de laboratorios
5. **Directivos Académicos**: Analizar métricas de utilización de recursos

5. Patrón de Microservicio Utilizado

API Gateway

Seleccionado: ✅

El patrón **API Gateway** fue implementado como punto central de entrada para todas las solicitudes. Este gateway:

- Enruta solicitudes hacia los microservicios correspondientes
- Centraliza la autenticación y validación de tokens JWT
- Abstrae la complejidad de múltiples servicios del cliente
- Facilita la evolución independiente de servicios sin afectar clientes

Implementación: Spring Cloud API Gateway en puerto 8080

Arquitectura de Microservicios

Seleccionado: ✅

Se utilizó una arquitectura basada en microservicios, dividiendo el sistema en servicios independientes para autenticación, gestión de laboratorios, prácticas y préstamos de equipos. Cada microservicio puede desplegarse, escalarse y mantenerse de forma aislada, comunicándose con los demás mediante APIs bien definidas, lo que reduce el acoplamiento y facilita la evolución del sistema.

Implementación: Múltiples servicios Spring Boot desplegados y orquestados mediante Docker y Docker Compose.

Patrón DTO + Services + Repository

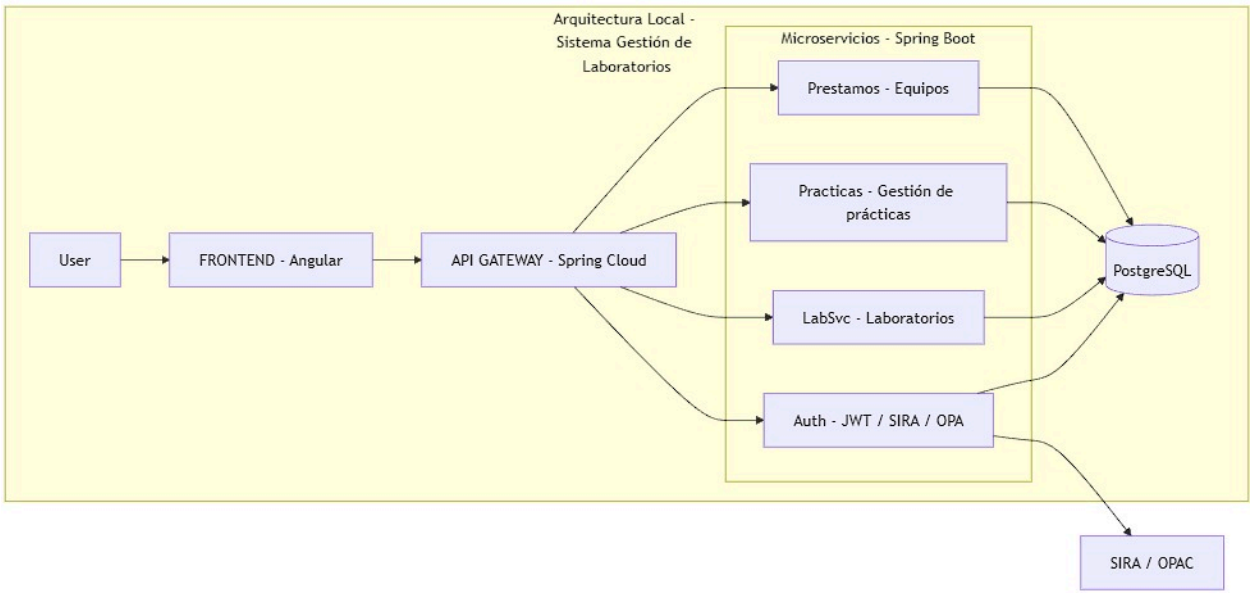
Seleccionado: ✅

En la capa backend se aplicó el patrón DTO + Services + Repository, separando la lógica en tres niveles bien definidos. Los DTO manejan la transferencia de datos hacia y desde el frontend, la capa de Services concentra la lógica de negocio y las reglas del dominio, y la capa Repository se encarga del acceso a datos mediante JPA, mejorando la mantenibilidad, testabilidad y claridad de la arquitectura.

Implementación: Clases DTO, servicios de negocio en Spring y repositorios JPA para la persistencia en PostgreSQL.

6. Arquitectura General

Diagrama de Arquitectura



Componentes Principales

1. Frontend (Angular)

- **Ubicación**: Puerto 4200
- **Responsabilidades**:
  - Interfaz de usuario responsiva
  - Consumo de API REST
  - Gestión de autenticación (tokens JWT)
  - Navegación dinámica según roles
  - Formularios CRUD para todas las entidades

2. API Gateway (Spring Boot)

- **Ubicación**: Puerto 8080
- **Responsabilidades**:
  - Punto de entrada único para todas las solicitudes
  - Autenticación y validación de JWT
  - Enrutamiento hacia microservicios
  - Control de permisos basado en roles

3. Microservicio de Autenticación

- **Ubicación**: Puerto 3000
- **Responsabilidades**:
  - Validación de credenciales contra CIRAN
  - Generación de tokens JWT
  - Consulta de información de usuarios via OPA
  - Gestión de roles y permisos

4. Microservicio de Laboratorios

- **Responsabilidades**:
  - CRUD de laboratorios
  - Gestión de disponibilidad
  - Información de capacidad y equipamiento
  - Consulta de horarios

5. Microservicio de Prácticas

- **Responsabilidades**:
  - CRUD de agendamiento de prácticas
  - Validación de disponibilidad de laboratorios
  - Registro de estudiantes por práctica
  - Reportes de prácticas realizadas

6. Microservicio de Préstamo de Equipos

- **Responsabilidades**:
  - CRUD de solicitudes de préstamo
  - Registro de entrega y devolución
  - Inventario de equipos
  - Historial de préstamos por usuario

7. Base de Datos PostgreSQL

- **Ubicación**: Puerto 5432
- **Responsabilidades**:
  - Persistencia de todos los datos
  - Integridad referencial mediante claves foráneas
  - Índices para optimización de consultas

Tecnologías por Componente

Componente	Tecnología	Puerto

Frontend	Angular + TypeScript + HTML/CSS	4200
API Gateway	Spring Boot + Spring Cloud	8080
Autenticación	OGS/CIRAN (Sistema Univalle)	3000
Backend Servicios	Spring Boot + Spring Web + JPA	8080
Base de Datos	PostgreSQL	5432
Contenedores	Docker	-
Orquestación	Docker Compose	-

7. Conclusiones

Dificultades Encontradas

1. **Integración con Sistemas Legacy**
   - Integrar los sistemas universitarios presentó desafíos debido a documentación limitada y cambios en credenciales
   - Solución: Se implementó un servicio wrapper que abstrae la complejidad de estas integraciones
2. **Seguridad de Credenciales**
   - Riesgo de exponer información sensible en solicitudes HTTP
   - Solución: Implementación de JWT y almacenamiento seguro de secretos en variables de entorno
3. **Configuración de Docker en Windows**
   - Incompatibilidades con Docker Desktop en sistemas Windows
   - Solución: Uso exclusivo de CLI y scripts de bash
4. **Dockerización del Frontend Angular**
   - Angular no estaba inicialmente containerizado
   - Solución: Se creó Dockerfile específico para Angular con servidor Nginx de producción

5. **Sincronización de Puertos**
    - Conflictos de puertos al ejecutar múltiples servicios localmente
    - Solución: Mapeo explícito de puertos en Docker Compose
6. **Performance de Consultas**
    - Algunas consultas a base de datos no utilizaban índices
    - Solución: Se optimizó con queries específicas en repositorios JPA

Aprendizajes Clave

1. **Arquitectura de Microservicios en Práctica**
    - Experiencia real implementando patrón API Gateway
    - Comprensión de desacoplamiento y comunicación entre servicios
    - Manejo de estado distribuido
2. **Integración de Autenticación**
    - JWT como estándar moderno de autenticación stateless
    - Coordinación entre múltiples sistemas de autenticación
    - Control de acceso basado en roles (RBAC)
3. **Frontend Moderno con Angular**
    - Arquitectura de componentes reutilizables
    - Consumo eficiente de APIs REST
    - Navegación dinámica basada en permisos
4. **Containerización y DevOps**
    - Docker como herramienta esencial para reproducibilidad
    - Docker Compose para orquestación local
    - Data Initializer para seed de datos de prueba
5. **Documentación como Requisito**
    - Importancia de documentar endpoints mediante Swagger/OpenAPI
    - README y guías de instalación críticas para transferencia de conocimiento
    - Diagramas visuales facilitan comprensión de arquitectura
6. **Consideraciones Académicas en Presentación**
    - Enfoque en patrones y tecnologías durante sustentación
    - Video demostrativo reduce necesidad de ejecución en vivo
    - Documentación sólida sustituye explicaciones detalladas de código

Recomendaciones

2. **Agregar Circuit Breaker (Resilience4j)**
    - Manejar fallos de comunicación entre microservicios
    - Mejorar resiliencia del sistema
3. **Implementar Validación de Entrada Robusta**
    - Bean Validation en todos los endpoints
    - Mensajes de error consistentes y claros
4. **Agregar Auditoría y Logging**
    - Registrar todas las operaciones críticas

    - Facilitar debugging y análisis de problemas
5. **Event-Driven Architecture con RabbitMQ**
    - Desacoplar aún más los microservicios
    - Implementar notificaciones asincrónicas (correos, SMS)
    - Crear sistema de eventos para auditoría
6. **Testing Automatizado Completo**
    - Unit tests para lógica de negocio
    - Integration tests para APIs
    - E2E tests para flujos críticos
7. **Integración con Más Sistemas**
    - Portal de estudiantes Univalle
    - Sistema de facturación
    - Plataforma de learning management (LMS)
8. **Análisis Predictivo**
    - Machine Learning para predicción de demanda de laboratorios
    - Análisis de patrones de uso
    - Optimización de asignación de recursos
9. **Escalabilidad Global**
    - Replicación de base de datos
    - Distribución geográfica de servicios
    - Sincronización de datos entre múltiples sedes

Conclusión General

Este proyecto de **gestión integral de laboratorios académicos** demostró la viabilidad de implementar una arquitectura moderna de microservicios en un contexto universitario real. La combinación de **Spring Boot, Angular, PostgreSQL y Docker** proporcionó una solución escalable, mantenible y fácil de desplegar.

Los patrones arquitectónicos implementados (API Gateway, autenticación con JWT, control de roles) son estándares de la industria y preparan el sistema para evolucionar hacia soluciones más complejas como event-driven architecture y machine learning.

La experiencia adquirida durante el desarrollo proporciona una base sólida para futuras iniciativas de transformación digital en la institución, demostrando que la automatización de procesos manuales genera valor real tanto para usuarios como para administradores.