

UNIVERSIDAD POPULAR DEL CESAR
FACULTAD DE INGENIERÍAS Y TECNOLÓGICAS
ESPECIALIZACIÓN EN INGENIERÍA DE SOFTWARE

TALLER No. 2

Diseño de solución con patrones de diseño
Sistema de Gestión Agrícola para Fincas Cafeteras

INTEGRANTES GRUPO 05:

Olimpo Castro Thorne
Juan de la Cruz Luque
Jhonnier de la Cruz Guzmán
Jesús Hernández
Wilmer Herrera

DOCENTE: Ing. Jairo Francisco Seoanes León

ASIGNATURA: Patrones de Diseño de Software

Junio de 2025 - Periodo 2025-I

Indice

- 1. Introducción**
- 2. Análisis del Escenario y Requerimientos**
 - 2.1. Contexto del Problema
 - 2.2. Propuesta de Solución Tecnológica
 - 2.3. Requerimientos Funcionales (RF)
 - 2.4. Requerimientos No Funcionales (RNF)
- 3. Diseño de la Solución y Arquitectura del Sistema**
 - 3.1. Elección de la Arquitectura
 - 3.2. Servicios Identificados en la Solución
 - 3.3. Aplicación de Patrones de Diseño
 - 3.4. Stack Tecnológico Seleccionado
 - 3.5. Integración de Patrones con Spring Boot
- 4. Arquitectura Lógica del Sistema (Componentes y Microservicios)**
 - 4.1. Descripción General
 - 4.2. Componentes Principales del Sistema
 - 4.3. Servicios Transversales
 - 4.4. Bases de Datos y Persistencia
 - 4.5. Comunicación entre Servicios
 - 4.6. Seguridad y Mantenimiento
 - 4.7. Diagrama de Componentes
- 5. Modelado de Clases y Representación Gráfica de Patrones**
 - 5.1. Diagrama de Secuencia: Generación de Reporte Agrícola (Builder)
 - 5.2. Diagrama de Actividad del Flujo General del Sistema
 - 5.3. Diagrama de Clases UML con Patrones Aplicados
- 6. Conclusiones**
- 7. Referencias Bibliográficas**

1. Introducción

La industria agrícola, especialmente en regiones rurales como el departamento del Cesar (Colombia), enfrenta desafíos crecientes relacionados con la eficiencia, sostenibilidad y trazabilidad de sus procesos productivos. En el contexto de la caficultura, muchas fincas pequeñas y medianas aún gestionan sus cultivos de forma empírica, con escasa adopción tecnológica, lo que limita su productividad y competitividad.

Frente a esta realidad, el diseño e implementación de sistemas de software especializados puede marcar una diferencia sustancial. Sin embargo, para lograr soluciones robustas, modulares y escalables, es esencial aplicar buenas prácticas de ingeniería de software, como los patrones de diseño. Estos patrones, especialmente los creacionales y estructurales, permiten construir sistemas que sean reutilizables, fácilmente extensibles y con una arquitectura sólida.

Este informe documenta el proceso completo de diseño e implementación de un sistema de gestión agrícola enfocado en la caficultura, empleando patrones de diseño creacionales (como Factory Method, Builder y Singleton) y estructurales (como Facade, Composite y Adapter), integrados con tecnologías modernas como Java, Spring Boot, y una arquitectura basada en microservicios.

El trabajo se desarrolla bajo una perspectiva práctica, con el objetivo de simular un proyecto real, fortaleciendo tanto los conocimientos técnicos como la capacidad de análisis, diseño e implementación de soluciones de software de calidad.

Objetivo General

Diseñar e implementar una solución de software modular para la gestión de fincas cafeteras, basada en una arquitectura de microservicios y la aplicación de patrones de diseño creacionales y estructurales, que permita automatizar tareas agrícolas, generar reportes técnicos y mejorar la toma de decisiones en el sector agroindustrial.

Objetivos Específicos

1. Analizar los requerimientos funcionales y no funcionales del sistema agrícola enfocado en el manejo de cultivos de café.
2. Seleccionar una arquitectura de software adecuada (microservicios) que garantice escalabilidad, modularidad y mantenimiento a largo plazo.
3. Aplicar patrones de diseño como Factory Method, Builder, Singleton, Decorator y Abstract Factory para optimizar la estructura del sistema y desacoplar componentes.
4. Modelar la solución utilizando diagramas de clases, componentes, secuencia y actividades que representen visualmente el diseño e interacción del sistema.
5. Integrar tecnologías como Java 17, Spring Boot y PostgreSQL en una arquitectura orientada a servicios, aplicando principios SOLID.
6. Evaluar la solución propuesta a través del análisis de sus ventajas arquitectónicas y su alineación con buenas prácticas de desarrollo de software.

2. Análisis del Escenario y Requerimientos

2.1. Contexto del Problema

La caficultura representa uno de los pilares económicos y sociales más importantes en el departamento del Cesar, Colombia. Esta actividad no solo genera empleo y dinamiza la economía rural, sino que también es parte integral del patrimonio cultural y alimentario de la región. Sin embargo, la mayoría de las pequeñas y medianas fincas cafeteras operan sin el respaldo de tecnologías digitales, apoyándose en métodos tradicionales y empíricos para la planificación, ejecución y control de sus labores agrícolas.

Este enfoque presenta serias limitaciones:

- No existe un control centralizado sobre las actividades de cultivo, lo que dificulta la trazabilidad de las decisiones técnicas tomadas en campo.
- Las decisiones sobre el uso de recursos como fertilizantes, pesticidas o agua se basan en la experiencia empírica del productor, lo que genera ineficiencias y sobrecostos.
- La ausencia de sistemas de monitoreo ambiental impide la detección temprana de eventos críticos como plagas, enfermedades o estrés hídrico.
- No se generan reportes sistemáticos que permitan analizar la productividad por lote, la eficiencia en el uso de insumos o la planificación de futuras cosechas.

En consecuencia, los productores enfrentan problemas de baja productividad, calidad inconsistente en el grano y dificultades para acceder a mercados de valor agregado que exigen sistemas de trazabilidad y gestión sostenible.

2.2. Propuesta de Solución Tecnológica

Con el objetivo de mejorar la eficiencia operativa y facilitar la toma de decisiones en las fincas cafeteras, se propone el desarrollo de una plataforma de software especializada en la gestión agrícola, que permita digitalizar los procesos clave asociados al manejo del cultivo de café.

Este sistema ofrecerá funcionalidades para:

- Registrar información detallada de cada finca, lote y cultivo.
- Planificar y documentar todas las actividades agrícolas realizadas en campo.
- Ingresar datos ambientales manuales o mediante sensores (en futuras versiones).
- Generar reportes técnicos de producción, consumo de insumos y comportamiento ambiental.
- Detectar condiciones críticas y emitir alertas inteligentes al caficultor.
- Organizar el conocimiento agronómico y facilitar su análisis para mejorar la toma de decisiones.

La arquitectura seleccionada será de tipo microservicios, lo cual permite un diseño modular y desacoplado, donde cada servicio (gestión de cultivos, actividades, reportes, alertas, etc.) puede evolucionar, escalar y desplegarse de manera independiente. Esto habilita un sistema robusto, adaptable y fácil de integrar con tecnologías emergentes como IoT, drones agrícolas o servicios meteorológicos externos.

2.3. Requerimientos Funcionales (RF)

Estos requerimientos definen las funcionalidades que el sistema debe ofrecer al usuario final para satisfacer las necesidades del sector caficultor:

| ID | Requerimiento Funcional |
|-----|--|
| RF1 | El sistema debe permitir registrar múltiples fincas y lotes, incluyendo ubicación, área y tipo de suelo. |
| RF2 | Debe poder registrar los cultivos establecidos por lote: tipo de café, variedad, fecha de siembra y densidad. |
| RF3 | Permitir el registro de actividades agrícolas como fertilización, riego, fumigación, poda, recolección, etc. |
| RF4 | Soportar la entrada de parámetros ambientales (temperatura, humedad, precipitaciones) manualmente o por integración futura con sensores. |
| RF5 | Registrar aparición de plagas o enfermedades, con detalles del diagnóstico, nivel de infestación y tratamiento aplicado. |
| RF6 | Generar reportes técnicos por finca o lote: productividad, consumo de insumos, historial de labores y proyecciones. |
| RF7 | Emitir alertas automatizadas sobre condiciones críticas (riesgo de plaga, necesidad de riego, etc.). |

| | |
|--------------------|---|
| id | RF1 |
| Nombre | Registro de Fincas y Lotes |
| Descripción | El sistema debe permitir registrar múltiples fincas y sus respectivos lotes agrícolas. Cada registro debe incluir información como la ubicación geográfica, el área cultivable y el tipo de suelo. Esta funcionalidad permite gestionar diferentes unidades productivas de manera independiente y organizada. |
| Importancia | Alta |
| Prioridad | Alta |

| | |
|--------------------|---|
| id | RF2 |
| Nombre | Registro de Cultivos por Lote |
| Descripción | El sistema debe ofrecer la capacidad de registrar los cultivos asociados a cada lote, incluyendo el tipo de café sembrado, su variedad, la fecha de siembra y la densidad de plantas. Esta información es clave para la trazabilidad del cultivo y la planificación de actividades agrícolas. |
| Importancia | Alta |
| Prioridad | Alta |

| | |
|--------------------|---|
| id | RF3 |
| Nombre | Registro de Actividades Agrícolas |
| Descripción | El sistema debe permitir registrar y documentar actividades agrícolas tales como fertilización, riego, fumigación, poda, recolección, entre otras. Cada actividad debe quedar asociada a un lote o cultivo específico, incluyendo detalles como la fecha, tipo de insumo utilizado y observaciones. |
| Importancia | Alta |
| Prioridad | Alta |

| | |
|---------------|-----------------------------------|
| id | RF4 |
| Nombre | Ingreso de Parámetros Ambientales |

| | |
|-------------|---|
| Descripción | El sistema debe soportar la entrada de datos ambientales relevantes, como temperatura, humedad y precipitaciones. Estos parámetros pueden ser ingresados manualmente o integrarse de forma automática mediante sensores IoT en futuras versiones. |
| Importancia | Media |
| Prioridad | Media |

| | |
|-------------|--|
| id | RF5 |
| Nombre | Gestión de Plagas y Enfermedades |
| Descripción | El sistema debe permitir registrar la aparición de plagas o enfermedades en los cultivos. Esta funcionalidad debe incluir información detallada como el diagnóstico realizado, el nivel de infestación y los tratamientos aplicados. |
| Importancia | Alta |
| Prioridad | Media |

| | |
|-------------|--|
| id | RF6 |
| Nombre | Generación de Reportes Técnicos |
| Descripción | El sistema debe ser capaz de generar reportes técnicos detallados por finca o lote. Los reportes incluirán datos de productividad, consumo de insumos, historial de labores agrícolas y proyecciones de rendimiento. Esta funcionalidad será fundamental para la toma de decisiones. |
| Importancia | Alta |
| Prioridad | Alta |

| | |
|-------------|---|
| id | RF7 |
| Nombre | Emisión de Alertas Automatizadas |
| Descripción | El sistema debe emitir alertas automáticas en función de condiciones críticas como riesgo de plagas, necesidad de riego o condiciones climáticas extremas. Las alertas deben ser generadas con base en reglas predefinidas y notificadas al usuario en tiempo oportuno. |
| Importancia | Alta |
| Prioridad | Alta |

2.4. Requerimientos No Funcionales (RNF)

Estos requerimientos aseguran que el sistema cumpla estándares de calidad, usabilidad y operación técnica:

| ID | Requerimiento No Funcional |
|------|--|
| RNF1 | La interfaz debe ser intuitiva y usable por agricultores con bajo nivel de alfabetización digital. |
| RNF2 | El sistema debe tener alta capacidad de respuesta, incluso con grandes volúmenes de datos históricos. |
| RNF3 | Se deben garantizar la integridad, confidencialidad y disponibilidad de los datos agrícolas. |
| RNF4 | La solución debe ser fácilmente escalable horizontalmente para atender a múltiples fincas o regiones. |
| RNF5 | El sistema debe ser tolerante a fallos, con mecanismos de respaldo automático y recuperación ante errores. |
| RNF6 | El código fuente debe estar debidamente documentado, estructurado en módulos y seguir principios SOLID. |
| RNF7 | La aplicación debe ser compatible con distintos dispositivos (PC, tablet, móvil) y navegadores web. |

| | |
|--------------------|--|
| id | RNF1 |
| Nombre | Interfaz Intuitiva |
| Descripción | La interfaz debe ser intuitiva y usable por agricultores con bajo nivel de alfabetización digital. |
| Importancia | Alta |
| Prioridad | Alta |

| | |
|--------------------|---|
| id | RNF2 |
| Nombre | Alto Rendimiento |
| Descripción | El sistema debe tener alta capacidad de respuesta, incluso con grandes volúmenes de datos históricos. |
| Importancia | Alta |
| Prioridad | Alta |

| | |
|--------------------|--|
| id | RNF3 |
| Nombre | Seguridad de los Datos |
| Descripción | Se deben garantizar la integridad, confidencialidad y disponibilidad de los datos agrícolas. |
| Importancia | Alta |
| Prioridad | Alta |

| | |
|--------------------|---|
| id | RNF4 |
| Nombre | Escalabilidad Horizontal |
| Descripción | La solución debe ser fácilmente escalable horizontalmente para atender a múltiples fincas o regiones. |
| Importancia | Media |
| Prioridad | Alta |

| | |
|--------------------|--|
| id | RNF5 |
| Nombre | Tolerancia a Fallos |
| Descripción | El sistema debe ser tolerante a fallos, con mecanismos de respaldo automático y recuperación ante errores. |
| Importancia | Alta |
| Prioridad | Alta |

| | |
|--------------------|---|
| id | RNF6 |
| Nombre | Código Fuente Documentado |
| Descripción | El código fuente debe estar debidamente documentado, estructurado en módulos y seguir principios SOLID. |
| Importancia | Media |
| Prioridad | Media |

| | |
|-------------|---|
| id | RNF7 |
| Nombre | Compatibilidad Multidispositivo |
| Descripción | La aplicación debe ser compatible con distintos dispositivos (PC, tablet, móvil) y navegadores web. |
| Importancia | Alta |
| Prioridad | Alta |

3. Diseño de la Solución y Arquitectura del Sistema

3.1. Elección de la Arquitectura

Se ha seleccionado una arquitectura basada en microservicios como fundamento para el desarrollo del sistema de gestión agrícola. Esta elección responde a la necesidad de contar con una solución escalable, modular, distribuida y fácilmente integrable con tecnologías emergentes (como sensores IoT, drones o APIs meteorológicas).

Justificación técnica de la elección:

- Modularidad y Escalabilidad: Cada componente del sistema (por ejemplo, gestión de fincas, actividades agrícolas, reportes o monitoreo ambiental) se implementa como un servicio independiente, facilitando el desarrollo paralelo, el mantenimiento y la escalabilidad específica de cada módulo.
- Flexibilidad Tecnológica: Se pueden usar diferentes tecnologías, bases de datos o lenguajes para cada microservicio, optimizando el rendimiento según el contexto del dato (ej. MongoDB para registros no estructurados y PostgreSQL para datos relacionales).
- Resiliencia: La caída o fallo de un servicio no afecta necesariamente al resto del sistema, favoreciendo la disponibilidad y tolerancia a fallos.
- Desarrollo Distribuido: Esta arquitectura permite que diferentes equipos o desarrolladores trabajen de manera independiente sobre servicios distintos sin generar conflictos, ideal para escalar el proyecto a futuro.

3.2. Servicios Identificados en la Solución

| Microservicio | Responsabilidad Principal |
|----------------|---|
| ms-fincas | Gestión de datos de fincas y lotes: ubicación, área, tipo de suelo, agrupación geográfica. |
| ms-cultivos | Registro y trazabilidad de cultivos: variedad, fechas de siembra, densidad, ciclo fenológico. |
| ms-actividades | Administración de tareas agrícolas: fertilización, riego, cosecha, |

| Microservicio | Responsabilidad Principal |
|-----------------------|---|
| | poda, etc. |
| ms-parametros | Registro manual o automático de condiciones ambientales: temperatura, humedad, precipitaciones. |
| ms-plagas | Gestión de incidencias fitosanitarias: plagas, enfermedades, tratamientos aplicados. |
| ms-alertas | Sistema de alertas basadas en reglas: condiciones críticas, tareas pendientes, etc. |
| ms-reportes | Generación de reportes por lote, finca o cultivo: productividad, consumo, proyecciones. |
| ms-usuarios | Gestión de usuarios, roles, autenticación y permisos. |
| ms-gateway (opcional) | Puerta de enlace de APIs para controlar acceso, enruteamiento y seguridad. |

3.3. Aplicación de Patrones de Diseño

◊ **Patrones Creacionales**

- Abstract Factory
Aplicación: Creación de familias de insumos agrícolas o sensores de distintas marcas.
Justificación: Permite crear objetos agrupados sin acoplarse a sus clases concretas, facilitando la inclusión de nuevos proveedores.
- Builder
Aplicación: Generación personalizada de reportes técnicos (por fechas, lotes, métricas).
Justificación: Separa la construcción compleja del objeto del proceso de presentación, evitando constructores inflexibles.
- Singleton
Aplicación: Gestión centralizada de configuración global o conexión a la base de datos.
Justificación: Garantiza que exista una única instancia, evitando redundancias y consumo innecesario de recursos.
- Factory Method
Aplicación: Creación dinámica de distintas actividades agrícolas (riego, poda, fumigación).

Justificación: Facilita la extensión de nuevas actividades sin modificar el código cliente.

◊ **Patrones Estructurales**

- Adapter

Aplicación: Consumo de APIs meteorológicas o sistemas externos de gestión de insumos.

Justificación: Permite adaptar interfaces incompatibles para integrarlas sin modificar el código del núcleo.

- Facade

Aplicación: Exponer una interfaz simplificada para la gestión de datos agrícolas complejos.

Justificación: Oculta la complejidad de las operaciones internas y provee una interfaz limpia para los consumidores.

- Composite

Aplicación: Modelado jerárquico de finca → lotes → parcelas.

Justificación: Permite tratar árboles de componentes como una única unidad, simplificando operaciones de cálculo o visualización.

- Decorator

Aplicación: Extensión de funcionalidades sobre objetos base, como un reporte con opciones de exportación (PDF, Excel, con gráficos).

Justificación: Permite añadir comportamiento sin modificar la clase base, útil para manejar múltiples combinaciones de salida.

3.4. Stack Tecnológico Seleccionado

| Tecnología | Uso Principal |
|---------------------|--|
| Java 17 | Lenguaje base para el backend. |
| Spring Boot 3.x | Framework para desarrollo rápido de microservicios. |
| Maven | Gestor de dependencias y construcción del proyecto. |
| PostgreSQL | Almacenamiento relacional de datos estructurados. |
| MongoDB (opcional) | Almacenamiento flexible de eventos, logs o datos no estructurados. |
| Docker | Contenedорización de cada microservicio. |
| Kubernetes (futuro) | Orquestación para despliegue y escalamiento. |

| Tecnología | Uso Principal |
|-------------------------|-------------------------|
| IntelliJ IDEA / VS Code | Entornos de desarrollo. |

3.5. Integración de Patrones con Spring Boot

Spring Boot, gracias a su motor IoC (Inversión de Control), facilita la implementación de múltiples patrones de diseño de forma natural y desacoplada. A continuación, se explica cómo:

- Factory Method / Abstract Factory:
Las fábricas pueden declararse como beans (@Component, @Bean) e injectarse dinámicamente con @Autowired o @Qualifier, permitiendo cambiar la implementación en tiempo de ejecución sin modificar el código consumidor.
- Builder:
Los builders pueden injectarse como componentes reutilizables en servicios que requieran construir reportes o configuraciones compuestas.
- Singleton:
Spring maneja todos los beans como Singleton por defecto, lo que hace innecesaria su implementación manual salvo en casos especiales.
- Decorator:
Los decoradores pueden componerse e injectarse envolviendo otros beans. Spring permite configurar cadenas de responsabilidad y beans compuestos.
- Adapter / Facade:
Se implementan como servicios (@Service) injectados en controladores o servicios centrales, simplificando el consumo de APIs o la complejidad interna del dominio.

Anotaciones clave utilizadas:

- @Component, @Service, @Repository, @Controller: Para registrar beans en el contenedor.
- @Autowired: Inyección automática de dependencias.
- @Qualifier: Diferenciar múltiples implementaciones de una misma interfaz.
- @Primary: Definir un bean preferido cuando hay más de uno.

4. Arquitectura Lógica del Sistema (Componentes y Microservicios)

4.1. Descripción General

El sistema se ha diseñado utilizando una arquitectura de microservicios desacoplados, donde cada servicio aborda una responsabilidad específica dentro del dominio agrícola. Esta estructura permite que los servicios se desarrollen, escalen y desplieguen de forma independiente, mejorando la flexibilidad, el mantenimiento y la resiliencia del sistema.

El acceso al sistema comienza desde el usuario, quien interactúa mediante una interfaz web o aplicación móvil conectada al API Gateway. Este gateway actúa como punto de entrada centralizado, encargándose del enrutamiento de solicitudes, seguridad, autenticación y balanceo de carga.

4.2. Componentes Principales del Sistema

A continuación, se describen los microservicios y componentes que conforman la solución:

API Gateway

- Intermediario entre el cliente y los servicios.
- Aplica políticas de seguridad, autenticación, límites de tráfico y enrutamiento.
- Facilita la exposición controlada de los microservicios hacia el exterior.

ms-usuarios

- Gestión de autenticación, autorización y perfiles de usuario.
- Responsable de emitir y validar tokens JWT para acceso seguro a los servicios.

ms-fincas

- Registra información de fincas y lotes agrícolas.
- Permite asociar ubicación, área, tipo de suelo y propietario.

ms-cultivos

- Gestiona los cultivos por lote: variedad, fechas de siembra, densidad de siembra.
- Establece relación entre lotes y actividades agronómicas asociadas.

ms-actividades

- Planifica y registra actividades agrícolas (siembra, fertilización, cosecha, etc.).
- Puede incluir evidencias (comentarios, imágenes, condiciones observadas).

ms-parametros

- Registra condiciones climáticas como temperatura, humedad o precipitaciones.
- Admite ingreso manual o integración futura con sensores IoT o APIs externas (mediante patrón Adapter).

ms-plagas

- Registra incidentes fitosanitarios: tipo de plaga/enfermedad, tratamiento aplicado, evolución.
- Posibilita análisis histórico para toma de decisiones preventivas.

ms-alertas

- Sistema de reglas y notificaciones.
- Genera alertas proactivas basadas en condiciones críticas: clima extremo, riesgo de plagas, tareas vencidas.

ms-reportes

- Construye reportes personalizados por cultivo, lote, finca o periodo.
- Utiliza el patrón Builder para estructurar reportes complejos.
- Soporta exportación en múltiples formatos (PDF, Excel, HTML).

4.3. Servicios Transversales

Además de los microservicios funcionales, se integran componentes transversales para habilitar buenas prácticas de arquitectura distribuida:

- Config Server: Centraliza la configuración de todos los microservicios, facilitando la gestión de entornos (dev, test, prod) y cambios en caliente.
- Service Discovery: Permite a los servicios registrarse y encontrarse entre sí dinámicamente, habilitando escalabilidad horizontal, balanceo de carga y tolerancia a fallos.

4.4. Bases de Datos y Persistencia

Cada microservicio gestiona su propia base de datos, respetando el principio de persistencia distribuida y eliminando dependencias cruzadas. Se utilizan diferentes motores de almacenamiento según las necesidades del servicio:

- PostgreSQL para almacenamiento estructurado: fincas, cultivos, usuarios, actividades, reportes.
- MongoDB para almacenamiento no estructurado: logs, adjuntos, eventos, configuraciones dinámicas.
- En futuras versiones se considerará el uso de almacenamiento distribuido tipo MinIO o Amazon S3 para evidencias y archivos multimedia.

4.5. Comunicación entre Servicios

Los microservicios se comunican utilizando principalmente API RESTful sobre HTTP. Cada servicio expone endpoints que son consumidos por otros servicios o por el API Gateway. Esta comunicación puede ser evolucionada a esquemas asíncronos mediante colas de mensajes (ej. RabbitMQ, Apache Kafka) para procesos desacoplados como procesamiento de alertas o generación de reportes automáticos.

4.6. Seguridad y Mantenimiento

- Autenticación basada en tokens JWT emitidos por ms-usuarios.
- Registro centralizado de logs y trazabilidad de eventos por microservicio.
- Contenerización con Docker para portabilidad y despliegue ágil.
- Soporte futuro para Kubernetes como orquestador de microservicios, asegurando disponibilidad, escalado automático y gestión de fallos.

4.7. Diagrama Arquitectónico

A continuación, se muestra el diagrama lógico que representa la interacción entre microservicios, el API Gateway, el usuario final y los componentes transversales de infraestructura:

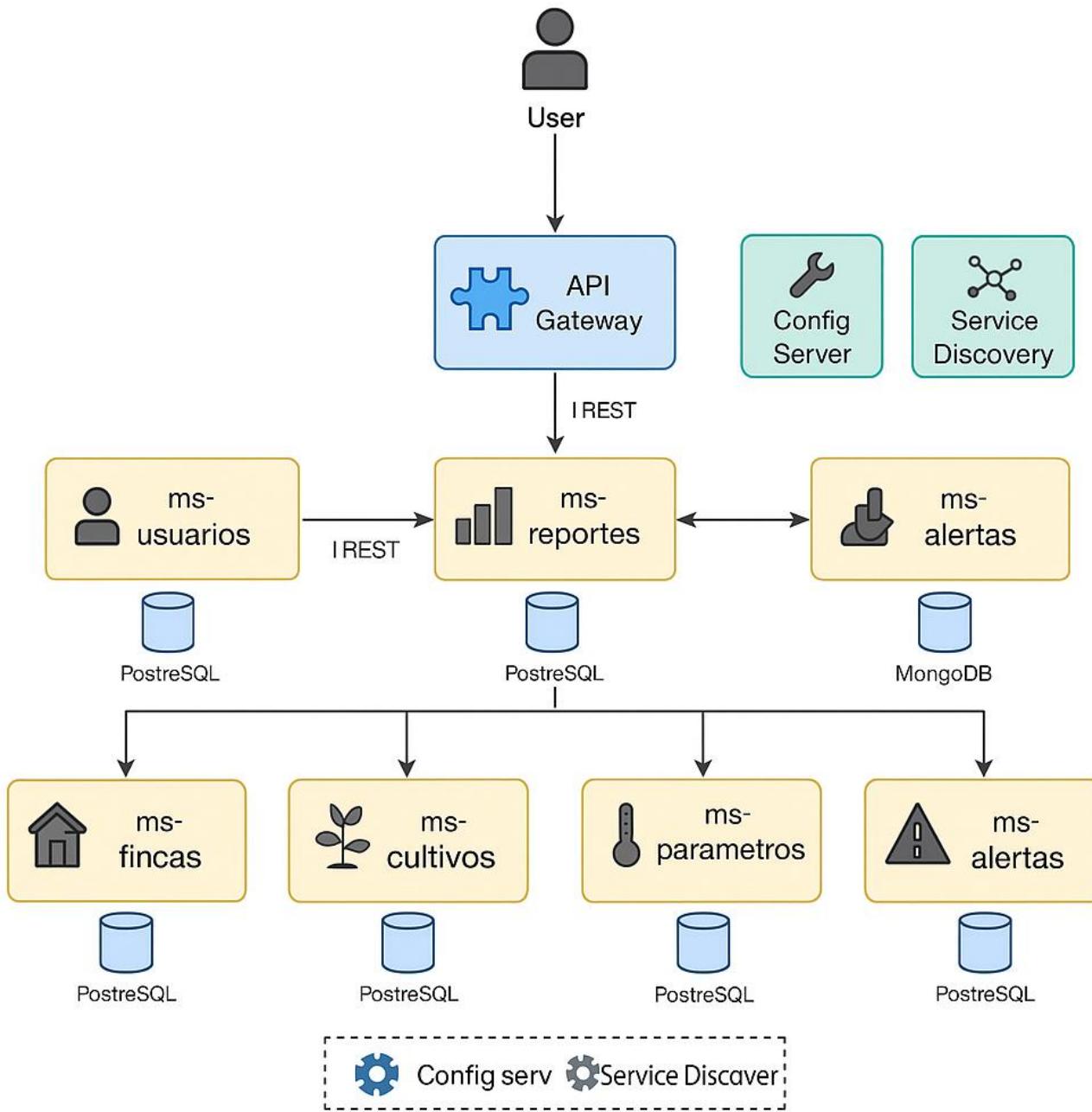


Diagrama de Componentes:

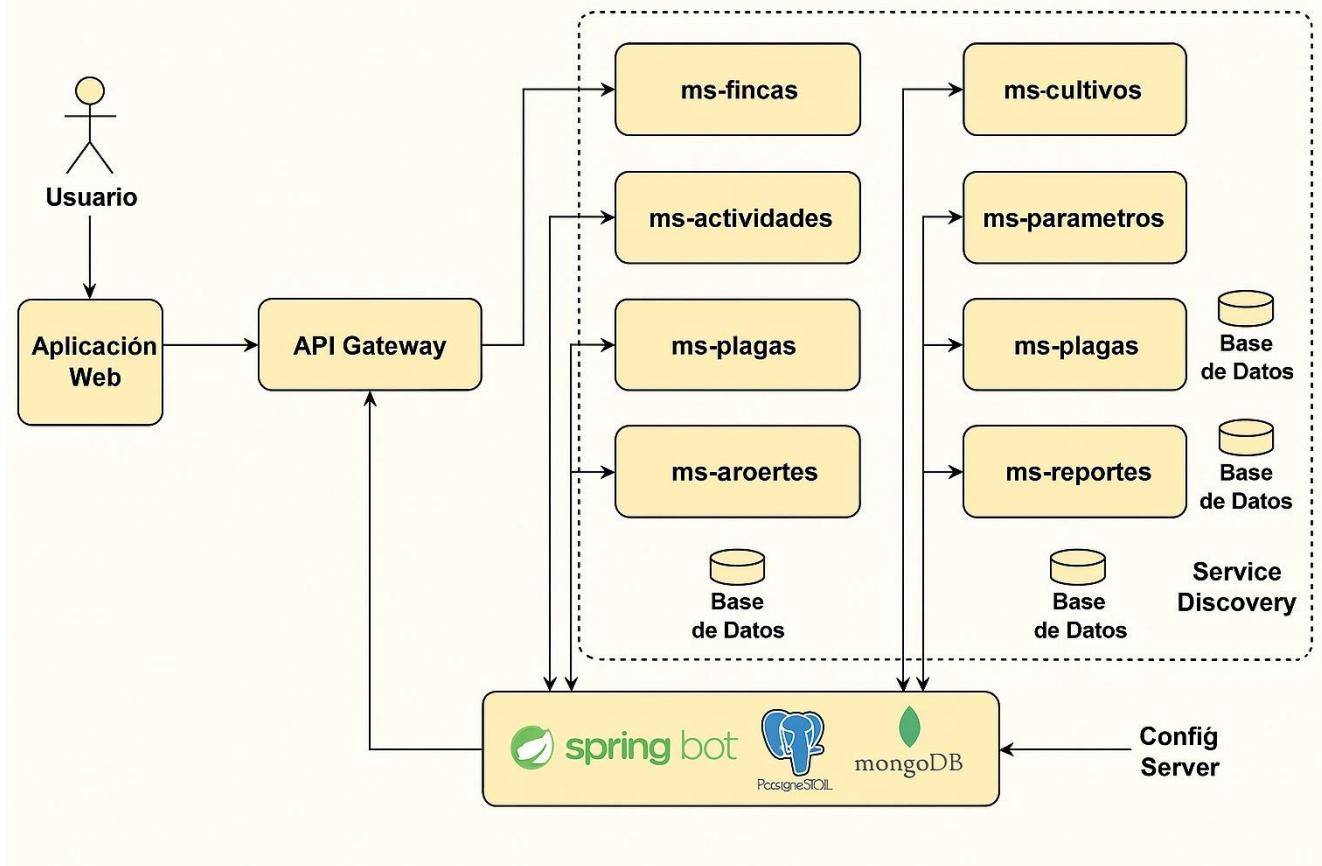


Diagrama de secuencia: Este diagrama muestra cómo se aplican los pasos del patrón Builder cuando un usuario genera un reporte personalizado de productividad agrícola a través del sistema.

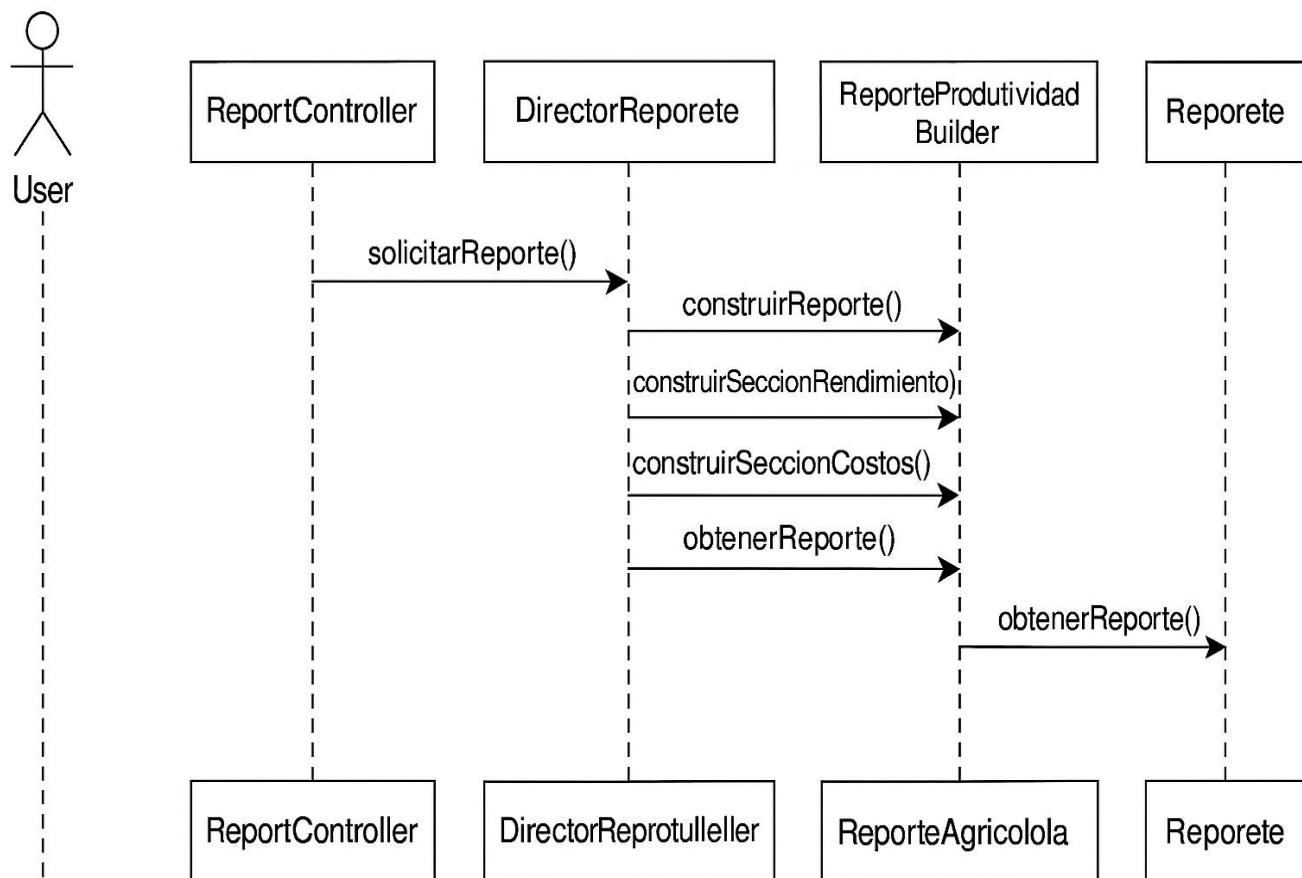
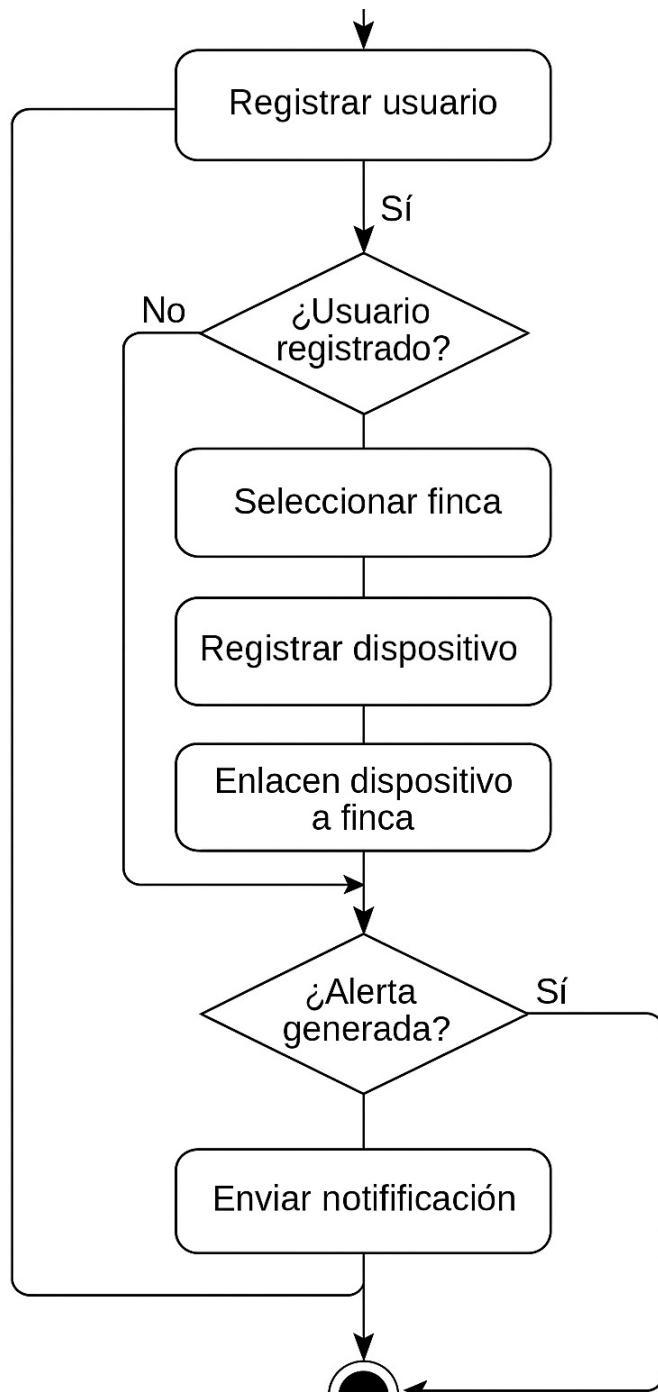


Diagrama de Actividad del Flujo General del Sistema: Este diagrama ilustra el flujo principal de interacción del usuario con el sistema agrícola, desde el ingreso hasta la generación del reporte final, reflejando procesos automatizados y decisiones del Sistema



5. Diagrama de Clases UML con Patrones de Diseño Aplicados

5.1. Introducción y Contexto

El diseño orientado a objetos del sistema agrícola propuesto se ha estructurado utilizando principios sólidos de ingeniería de software y la aplicación estratégica de patrones de diseño creacionales y estructurales. El objetivo es crear un modelo de clases que sea modular, reutilizable, fácilmente mantenible y escalable.

Este modelo refleja tanto las entidades del dominio agrícola (como fincas, cultivos, actividades y reportes), como también las soluciones arquitectónicas aplicadas para resolver problemas comunes de creación, estructuración y extensión de objetos.

El diagrama de clases que acompaña este apartado ilustra las relaciones entre entidades, el uso de clases abstractas, interfaces, herencia y composición, junto con los puntos exactos en los que se aplican patrones como Factory Method, Builder, Abstract Factory, Composite, Decorator y Singleton.

5.2. Clases del Dominio y su Rol

A continuación, se describen las principales clases del modelo y su relación con los patrones aplicados:

| Clase | Descripción Funcional |
|-------------------------------------|--|
| Finca | Representa una finca cafetera. Contiene una colección de objetos Lote. (Composite) |
| Lote | Subdivisión de la finca. Puede contener múltiples objetos Cultivo. |
| Cultivo | Define un cultivo específico (variedad, fechas), vinculado a actividades. |
| ActividadAgricola | Clase abstracta base para representar una actividad como riego, poda, fertilización. |
| ActividadRiego, ActividadPoda, etc. | Subclases concretas de ActividadAgricola. (Factory Method) |
| ReporteAgricola | Entidad compuesta que representa reportes productivos, ambientales o financieros. |

| Clase | Descripción Funcional |
|--|---|
| ReporteBuilder | Interfaz que define el proceso de construcción de reportes. (Builder) |
| ReporteProductividadBuilder | Builder concreto que construye reportes de rendimiento por finca/lote. |
| ReporteConsumoBuilder | Builder para reportes de consumo de insumos o recursos. |
| DirectorReporte | Encargado de orquestar el proceso de construcción de un reporte. |
| Sensor | Interfaz genérica para representar sensores climáticos o de suelo. |
| SensorProviderFactory | Fábrica abstracta para crear sensores de distintas marcas. (Abstract Factory) |
| ProveedorAFactory, ProveedorBFactory | Implementaciones concretas de SensorProviderFactory. |
| ReporteDecorador | Clase abstracta que permite extender funcionalidades del ReporteAgricola. (Decorator) |
| ReporteConGrafico, ReporteExportablePDF | Decoradores para añadir visualización o exportación. |
| ConexionBD | Implementa el patrón Singleton para gestionar una única instancia de conexión. |

5.3. Aplicación de Patrones y Justificación

| Patrón de Diseño | Clases Involucradas | Justificación Técnica |
|------------------|---|---|
| Factory Method | ActividadAgricola, ActividadRiego, etc. | Permite instanciar actividades específicas sin acoplar al cliente con clases concretas. |
| Builder | ReporteBuilder, DirectorReporte, | Facilita la construcción de reportes configurables paso a paso, separados |

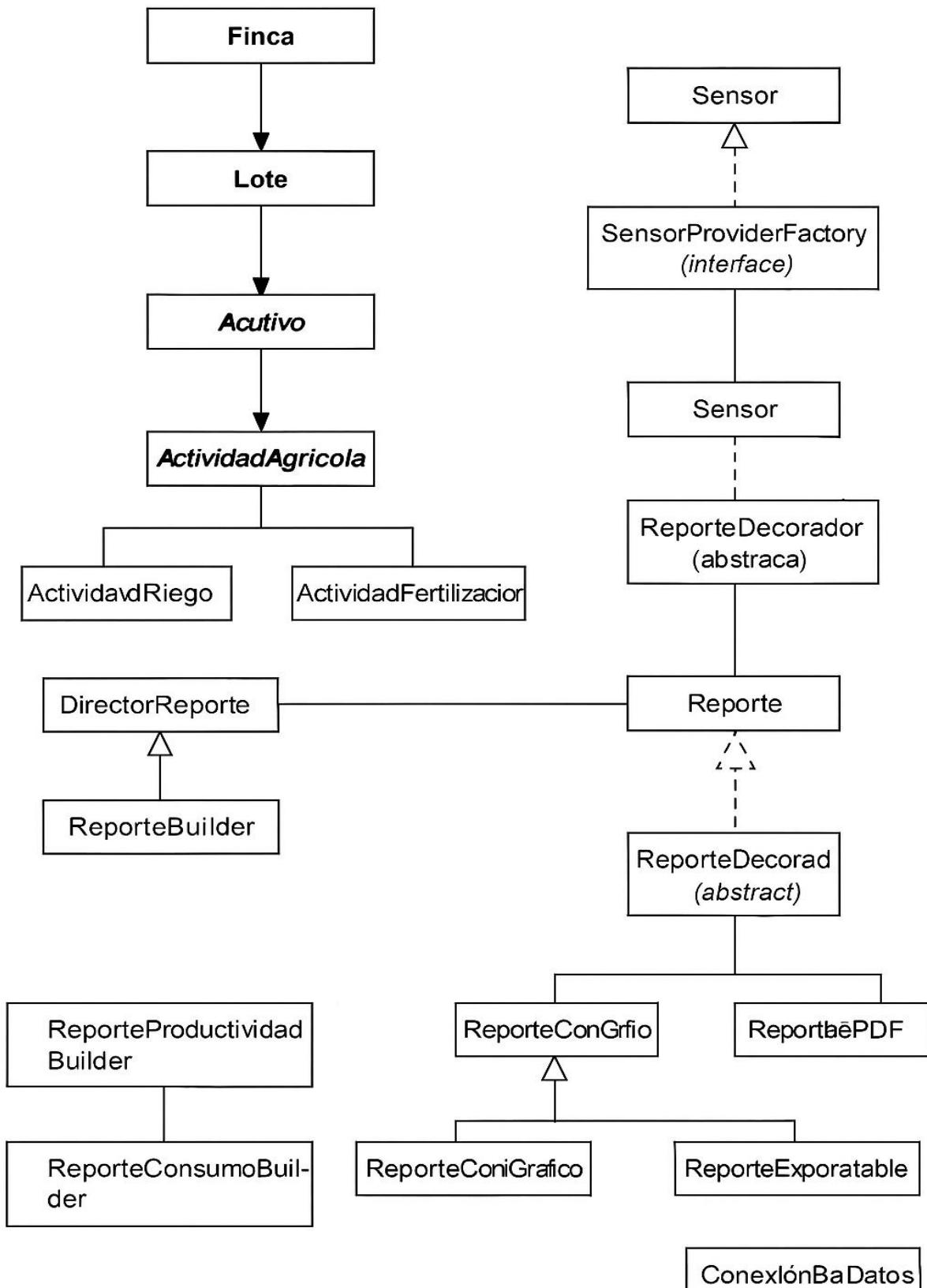
| Patrón de Diseño | Clases Involucradas | Justificación Técnica |
|------------------|---|---|
| | ReporteAgricola | de su estructura. |
| Abstract Factory | SensorProviderFactory, ProveedorAFactory, ProveedorBFactory | Provee una interfaz para familias de objetos (sensores) sin conocer sus implementaciones. |
| Singleton | ConexionBD | Asegura una única instancia compartida del gestor de conexiones. |
| Composite | Finca → Lote → Cultivo | Permite modelar estructuras jerárquicas y operar sobre ellas de manera uniforme. |
| Decorator | ReporteAgricola, ReporteConGrafico, ReporteExportablePDF | Agrega responsabilidades dinámicas a reportes sin modificar su clase base. |

5.4. Relación entre Clases (Resumen de Interacciones)

- Finca contiene una lista de Lote (composición).
- Lote contiene una lista de Cultivo.
- Cultivo está asociado a múltiples ActividadAgricola.
- ActividadAgricola es una clase abstracta con subclases concretas como ActividadRiego, ActividadPoda, etc.
- ReporteAgricola se construye con ayuda de DirectorReporte y un objeto ReporteBuilder.
- Sensor es una interfaz implementada por distintos sensores creados a través de SensorProviderFactory.
- ReporteAgricola puede ser extendido con decoradores como ReporteConGrafico o ReporteExportablePDF.
- ConexionBD es accedida globalmente como instancia única gracias al patrón Singleton.

Este diseño garantiza un sistema flexible y extensible, donde la adición de nuevos tipos de actividades, reportes, sensores o comportamientos no requiere reescribir estructuras existentes, cumpliendo así con el principio abierto/cerrado (Open/Closed) del diseño orientado a objetos.

Diagrama de clases UML con los patrones aplicados:



6. Conclusiones

La implementación de patrones de diseño en el desarrollo del sistema agrícola basado en microservicios demostró ser una estrategia altamente efectiva para mejorar la modularidad, mantenibilidad y escalabilidad del software.

Cada patrón abordó una necesidad específica del sistema:

- Factory Method permitió crear tipos de actividades agrícolas de forma flexible.
- Builder facilitó la construcción de reportes configurables.
- Singleton aseguró control sobre la conexión a la base de datos.
- Decorator hizo posible extender funcionalidades en los reportes sin alterar su estructura base.
- Abstract Factory brindó independencia tecnológica en la integración de sensores.

El uso de estos patrones no solo refuerza las buenas prácticas de diseño, sino que también prepara el sistema para cambios futuros, permitiendo extender funcionalidades, integrar nuevas tecnologías o adaptar componentes sin reescribir la lógica existente.

Finalmente, la estructura de microservicios y la orientación a patrones sentaron una base sólida para un desarrollo sostenible, escalable y profesional en el contexto de soluciones

7. Referencias Bibliográficas

1. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995).
Patrones de diseño: Elementos de software orientado a objetos reutilizable.
⌚ [https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%BAo_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%BAo_(inform%C3%A1tica))
2. Oracle (2023).
Documentación oficial de Java SE 17.
⌚ <https://docs.oracle.com/en/java/javase/17/>
3. Spring Framework (2024).
Documentación oficial de Spring Boot (en español).
⌚ <https://es.spring.io/projects/spring-boot>
4. Refactoring Guru.
Patrones de diseño orientados a objetos (en español). Excelente para estudio y consulta.
⌚ <https://refactoring.guru/es/design-patterns>
5. Educative & Platzi (sólo lectura básica).
Introducción a los microservicios y arquitectura desacoplada.
⌚ <https://platzi.com/tutoriales/2126-arquitectura/6497-que-es-la-arquitectura-de-microservicios/>
6. Microsoft Learn (2023).
Buenas prácticas para arquitecturas basadas en microservicios (en español).
⌚ <https://learn.microsoft.com/es-es/dotnet/architecture/microservices/>
7. Manning (Traducción Técnica).
Microservicios: Patrones con ejemplos en Java – Chris Richardson.
⌚ [\(edición traducida disponible\)](https://www.manning.com/books/microservicios-patrones)