Catallactical

Informe de Auditoría Técnica

Plataforma SIG-COMPLY

Sigillum Knowledge Solutions S.L.

Updated 2026-03-04

Contents

Objeto del Informe

El presente informe documenta los resultados de la auditoría técnica independiente realizada por **Catallactical SL** sobre la plataforma **SIG-COMPLY**, propiedad de **Sigillum Knowledge Solutions S.L.** (en adelante, SKS).

Contexto

SKS desarrolla y comercializa SIG-COMPLY, una plataforma SaaS dirigida al sector cosmético que facilita la gestión del ciclo de vida de productos cosméticos. La plataforma cubre los siguientes ámbitos funcionales:

- **Gestión de Product Information Files (PIF):** Elaboración automatizada de la documentación técnica exigida por la normativa europea para la comercialización de productos cosméticos en la Unión Europea.
- **Gestión de fórmulas y sustancias:** Registro y seguimiento de la composición química de los productos, incluyendo materias primas, sustancias procesadas, impurezas y especificaciones.
- **Evaluaciones toxicológicas:** Cálculos automatizados de margen de seguridad y evaluaciones de riesgo para cada ingrediente.
- **Cumplimiento normativo:** Verificación de conformidad con la normativa europea y gestión de entidades reguladas.
- **Gestión multi-organización:** Soporte para múltiples organizaciones y suborganizaciones con control de acceso por roles.

La plataforma se implementa como una aplicación web basada en Spring Boot (backend Java) y Angular (frontend), desplegada como un servicio SaaS con persistencia en PostgreSQL y autenticación delegada en Auth0.

SKS ha contratado a Catallactical SL para llevar a cabo una auditoría técnica integral de la plataforma. El objetivo de esta auditoría es doble: por un lado, identificar áreas de mejora arquitectónica y proponer soluciones concretas con estimaciones de esfuerzo; por otro, fortalecer las capacidades técnicas del equipo de desarrollo de SKS mediante mentorización y transferencia de conocimiento.

Partes implicadas

Rol	Entidad
Cliente	Sigillum Knowledge Solutions S.L. (SKS)
Auditor técnico	Catallactical SL

Alcance general

La auditoría abarca cinco tareas principales que cubren desde la revisión de la arquitectura actual hasta propuestas detalladas de mejora en subsistemas críticos: permisos y autorización, generación de documentos PIF y versionado de fórmulas cosméticas. Adicionalmente, Catallactical ha proporcionado orientación en diseño dirigido por dominio (DDD), ha asistido en la implantación de una metodología ágil de desarrollo y ha asignado un ingeniero senior dedicado a la supervisión y mentorización del equipo de desarrollo de SKS.

Metodología y Procedimiento

Enfoque general

La auditoría se ha abordado con un enfoque pragmático orientado a resultados, combinando múltiples técnicas de análisis:

- **Análisis estático del código fuente:** Revisión sistemática del repositorio SCATHA, incluyendo estructura de paquetes, patrones de diseño, dependencias, configuración de seguridad y calidad del código.
- **Revisión de la arquitectura desplegada:** Evaluación de la topología de despliegue, dependencias externas, estrategia de persistencia y flujos de datos críticos.
- **Sesiones de trabajo colaborativas:** Reuniones periódicas con el equipo de SKS para presentar hallazgos, validar propuestas y alinear expectativas.
- **Análisis de dominio:** Identificación de dominios de negocio y bounded contexts siguiendo principios de Domain-Driven Design.

Se ha priorizado la identificación de riesgos técnicos con impacto directo en la escalabilidad, seguridad y mantenibilidad de la plataforma, con especial atención a los requisitos regulatorios del sector cosmético.

Procedimiento seguido

El trabajo se ha desarrollado siguiendo las siguientes fases:

1. **Acceso al repositorio de código**: Se ha proporcionado acceso completo al repositorio del proyecto SCATHA (nombre interno de SIG-COMPLY), incluyendo el backend (Java/Spring Boot) y el frontend (Angular).
2. **Análisis de código y arquitectura**: Revisión sistemática de la estructura de paquetes, modelos de datos, capas de servicio, controladores REST, configuración de seguridad, estrategia de persistencia y flujos de generación de documentos.
3. **Análisis de dominios (DDD)**: Identificación de dominios y contextos delimitados (bounded contexts) siguiendo principios de Domain-Driven Design, con el objetivo de orientar la evolución hacia una arquitectura más modular y alineada con el negocio.
4. **Elaboración de informes técnicos**: Generación de informes detallados para cada area auditada, con hallazgos, problemas identificados y recomendaciones concretas.
5. **Sesiones de trabajo y mentorización**: Reuniones periódicas con el equipo de SKS para presentar hallazgos, discutir alternativas y validar propuestas. Asignación de un ingeniero senior de nivel staff para supervisar y guiar al equipo de desarrollo en la adopción de buenas prácticas.
6. **Implantación de metodología ágil**: Asistencia en la implantación de prácticas ágiles de desarrollo, incluyendo organización del trabajo en iteraciones, definición de épicas y gestión del backlog.
7. **Estimación de alcance**: Dimensionamiento del esfuerzo necesario para implementar cada una de las mejoras propuestas.
8. **Documentación y guías**: Elaboración de documentación técnica y guías de buenas prácticas para el equipo de desarrollo, incluyendo guías de convenios de entidades JPA (estrategias de fetch, relaciones bidireccionales, referencias FK circulares, tipos de cascade) y guías de arquitectura por capas (modelo de tres capas Entity/Domain/DTO con MapStruct).

Alcance de las actividades

La siguiente tabla resume las cinco tareas principales que componen la auditoría:

Tarea	Descripción	Fuente
1	Revisión de la arquitectura actual de SIG-COMPLY	Análisis de código
2	Propuesta de mejora de arquitectura	Síntesis de Tarea 1
3	Sistema de permisos y autorización	Análisis especializado
4	Sistema de generación de PIF	Análisis especializado
5	Versionado de fórmulas cosméticas	Análisis especializado

Adicionalmente, como actividades transversales, se ha proporcionado:

- **Orientación en Domain-Driven Design (DDD)**: Análisis de dominios, identificación de bounded contexts y recomendaciones para la alineación del modelo de datos con los procesos de negocio.
- **Implantación de metodología ágil**: Definición de flujos de trabajo, organización en sprints y gestión de backlog.
- **Mentorización del equipo**: Asignación de un ingeniero senior (nivel staff) dedicado a la supervisión técnica, revisiones de código, generación de documentación y guías de buenas prácticas.

Tarea 1: Revisión de la Arquitectura Actual

Esta sección presenta los hallazgos de la revisión arquitectónica del proyecto SCATHA, abordando la estructura general, seguridad, persistencia de datos, escalabilidad, mantenibilidad y observabilidad.

Estructura general de la aplicación

Observación: La aplicación opera actualmente como un backend monolítico de Spring Boot que sirve un frontend de Angular dentro de una única unidad desplegable.

Problemas clave:

- **Escalabilidad limitada:** Las operaciones que consumen muchos recursos (por ejemplo, la generación de PIF) pueden consumir los recursos de toda la aplicación, impidiendo el escalado independiente de los componentes.
- **Flexibilidad de despliegue:** Las actualizaciones del frontend requieren un redespliegue completo del backend, lo que ralentiza los ciclos de lanzamiento.

Recomendaciones:

- Evaluar la extracción de tareas altamente intensivas en recursos a servicios dedicados para permitir el escalado independiente.
- En el momento en que existan más de dos equipos de desarrollo, desacoplar el frontend y el backend, desplegándolos como aplicaciones separadas (Angular como activos estáticos, Spring Boot como API REST pura).

Implementación de seguridad

Observación: La autenticación se maneja de forma robusta utilizando Auth0 (OpenID Connect) y Spring Security.

Problemas clave:

- **Dependencia de autenticación externa:** La dependencia de Auth0 para la autenticación introduce una herramienta de terceros, lo que podría dificultar el despliegue en escenarios sin acceso a red externa.
- **Autorización duplicada e incompleta:** La lógica de autorización se encontraba duplicada y dispersa a través de los endpoints de la aplicación, con casos en los que la autorización estaba completamente ausente, lo que representa un riesgo de seguridad significativo en un entorno multi-tenant.

Recomendaciones:

- Permitir la autenticación utilizando JWT de forma independiente a Auth0, disponiendo de un servicio de autenticación que pueda ser desplegado independientemente para escenarios de despliegue en zonas bajo control exhaustivo o sin acceso a red externa.
- Centralizar la lógica de autorización mediante un patrón PEP/PDP con AOP, eliminando la duplicación y garantizando cobertura completa en todos los endpoints. Se ha formado al equipo en modelos RBAC, ABAC y NGAC para fundamentar las decisiones de diseño.

Estrategia de persistencia de datos

Observación: Spring Data JPA estándar con Hibernate, respaldado por PostgreSQL. H2 se utiliza para desarrollo y pruebas.

Problemas clave:

- **Falta crítica de auditoría:** Las entidades clave (Substance, Pif, Regulation) carecen de la anotación de auditoría, a pesar de que la dependencia spring-data-envers está presente en el

proyecto. Esto crea una brecha significativa en la trazabilidad y el mantenimiento de registros históricos, crucial para el cumplimiento normativo.
- **Potencial de problemas de rendimiento:** Muchas entidades utilizan campos de texto grandes, lo que, combinado con una auditoría exhaustiva, podría afectar el rendimiento y el almacenamiento de la base de datos.
- **Carga eager:** Algunas relaciones utilizan FetchType.EAGER, lo que podría llevar a problemas de consultas N+1 y a la recuperación de datos excesivos.

Recomendaciones:

- Implementar auditoría en todas las entidades críticas que requieran seguimiento de cambios históricos para el cumplimiento normativo.
- Revisar todas las relaciones OneToMany y ManyToMany, estableciendo por defecto FetchType.LAZY y recuperando datos explícitamente solo cuando sea necesario para optimizar el rendimiento.
- Asegurar una indexación adecuada de la base de datos y considerar la optimización de consultas para campos de texto grandes a los que se accede con frecuencia.

Cuellos de botella de escalabilidad y rendimiento

Observación: Las causas fundamentales incluyen la estructura monolítica, la generación síncrona de PIF y las posibles ineficiencias en la persistencia de datos.

Problemas clave:

- **Monolito:** Limita el escalado independiente de las partes intensivas en recursos.
- **Generación síncrona de PIF:** Ocupa los hilos web, provocando largos tiempos de respuesta.
- **Rendimiento de la base de datos:** Grandes campos de texto, carga eager y la sobrecarga futura de la auditoría podrían afectar el rendimiento de las consultas y escrituras.
- **Cálculos complejos:** Los servicios de cálculo de toxicología podrían convertirse en cuellos de botella intensivos en CPU.

Recomendaciones:

- Implementar procesamiento asíncrono para tareas pesadas para liberar hilos web y mejorar la capacidad de respuesta.
- Optimizar las interacciones con la base de datos mediante la indexación, la carga lazy y el ajuste de consultas.
- Introducir cache para datos de acceso frecuente y que cambian lentamente para reducir la carga de la base de datos.
- Introducir mecanismos de perfilado y monitorización para identificar y abordar proactivamente los cuellos de botella.

Mantenibilidad y testabilidad del código

Observación: La base de código generalmente tiene una buena estructura modular con una clara separación de paquetes y utiliza Lombok. Existen algunas clases de prueba.

Problemas clave:

- **Lógica de filtrado compleja y duplicada:** CosmeticProductService contiene una lógica de filtrado altamente compleja, duplicada y difícil de mantener.
- **Longitud y cohesión del método:** PIFReportController.generateReport es excesivamente largo, lo que afecta la legibilidad y mantenibilidad.
- **Inyección de campos:** La inyección de campos mediante anotaciones puede dificultar las pruebas unitarias sin un contexto completo de Spring.

Recomendaciones:

- Refactorizar la lógica de filtrado compleja utilizando Spring Data JPA Specifications, Querydsl o Criteria API para mejorar la mantenibilidad y reducir la duplicación.

- Dividir los métodos largos en unidades más pequeñas y enfocadas para mejorar la legibilidad y la testabilidad.
- Preferir la inyección por constructor para todas las dependencias para mejorar la testabilidad y promover una gestión de dependencias más clara.

Manejo de errores y observabilidad

Observación: Registro básico con Log4j2 y un manejador de excepciones global limitado para una única excepción personalizada.

Problemas clave:

- **Manejo de excepciones global incompleto:** No hay un manejador de reserva genérico para excepciones no manejadas, lo que podría exponer información sensible o devolver respuestas de error inconsistentes.
- **Falta de métricas y monitoreo:** No hay una integración evidente de métricas para monitorear la salud y el rendimiento de la aplicación.
- **Falta de trazado distribuido:** La falta de trazado distribuido dificulta la depuración de flujos de solicitudes entre componentes.

Recomendaciones:

- Implementar un manejador de excepciones global completo con una reserva genérica, devolviendo respuestas de error consistentes y no sensibles a los usuarios.
- Integrar Spring Boot Actuator y Micrometer para recopilar y exponer métricas de la aplicación, e integrar con un sistema de monitoreo.
- Implementar trazado distribuido para la mejor visibilidad de los flujos de solicitudes, la latencia y una depuración más sencilla en todo el sistema.

Análisis de dominios (DDD)

Como parte de la auditoría, se ha realizado un análisis de dominios siguiendo principios de Domain-Driven Design para identificar los contextos delimitados (bounded contexts) del sistema y orientar su evolución futura. Este análisis constituye la base para la modularización del sistema y ha sido complementado con sesiones de orientación al equipo de SKS sobre conceptos y prácticas de DDD.

Se han identificado tres dominios core y varios dominios de soporte:

Dominios core:

Dominio de Materiales y Fórmulas: Es el dominio con mayor complejidad de negocio. Abarca la gestión de materias primas, la composición de fórmulas cosméticas y el versionado de formulaciones. Este dominio es central para la plataforma, ya que las fórmulas son el eje sobre el que giran los cálculos, la documentación regulatoria y los productos comerciales. Entidades clave: Formula, FormulaVersion, RawMaterial, Impurity, FinalFormula, Specifications, Packaging.

Dominio Toxicológico: Gestiona las evaluaciones toxicológicas, cálculos de margen de seguridad, datos de exposición y perfiles toxicológicos de las sustancias presentes en las formulaciones. Este dominio consume datos del dominio de Sustancias y del dominio de Materiales y Fórmulas para producir las evaluaciones de seguridad exigidas por la normativa. Entidades clave: Toxicology, Exposition, cálculos de MoS (Margin of Safety).

Dominio Regulatorio: Gestiona el cumplimiento normativo, la generación de Product Information Files (PIF), la gestión de mercados de destino, los productos comerciales y la documentación regulatoria exigida por la normativa europea. Este dominio orquesta información de los dominios anteriores para producir la documentación necesaria para la comercialización. Entidades clave: Pif, Market, CosmeticProduct, ComercialProduct, CertificateType, Event.

Dominios de soporte:

Dominio de Sustancias: Gestiona el catálogo de sustancias químicas que conforman la base de las formulaciones cosméticas. Es responsable de la identificación, clasificación y datos regulatorios de

cada sustancia individual (números CAS, INCI names, propiedades físico-químicas). Aunque fundamental para el funcionamiento de la plataforma, este dominio actúa como soporte proporcionando datos de referencia que consumen los dominios core. Entidades clave: Substance, ProcessedSubstance, RegulatedEntity, Cas, Ec.

Gestión de Identidad y Accesos (IAM): Maneja la autenticación, autorización, perfiles de usuario, roles, grupos y la jerarquía organizativa (organizaciones y suborganizaciones). Este dominio está íntimamente relacionado con las propuestas de la Tarea 3, ya que la evolución del sistema de autorización debe alinearse con la estructura organizativa. Entidades clave: User, Organization, Group, Suborganization.

Plataforma Técnica: Cubre los aspectos técnicos fundacionales, incluyendo la infraestructura backend (Java/Maven), la aplicación frontend principal (Angular), el sistema de generación de documentos y los componentes UI genéricos.

Hallazgo clave: La estructura actual del código no refleja adecuadamente estos dominios, lo que genera acoplamiento entre contextos que deberían evolucionar de forma independiente. Por ejemplo, las entidades del dominio de Sustancias están mezcladas con las del dominio Toxicológico en los mismos paquetes, y el dominio Regulatorio no tiene límites claros frente al dominio de Materiales y Fórmulas. Se recomienda alinear progresivamente la estructura de paquetes y módulos con estos dominios, comenzando por la separación a nivel de paquete dentro del monolito actual.

Tarea 2: Propuesta de Mejora de Arquitectura

A partir de los hallazgos de la Tarea 1, se sintetizan las siguientes direcciones estratégicas de mejora.

Dirección estratégica

La evolución arquitectónica de SIG-COMPLY debe seguir un enfoque incremental y pragmático, priorizando las mejoras que aporten mayor valor en términos de seguridad, escalabilidad y mantenibilidad, sin interrumpir la operación actual de la plataforma. Es fundamental que cada mejora sea implementable de forma aislada, validable independientemente y desplegable sin riesgo para los usuarios existentes.

Las seis líneas de actúación principales, ordenadas por prioridad estratégica, son:

1. Desacoplamiento de la aplicación
2. Mejora del sistema de autorización
3. Implementación de auditoría de datos
4. Procesamiento asíncrono
5. Observabilidad y monitorización
6. Alíneación con Domain-Driven Design

Desacoplamiento de la aplicación

El monolito actual, si bien funcional para el tamaño actual del equipo, presenta limitaciones crecientes a medida que la plataforma evoluciona. El enfoque propuesto no pasa por extraer funcionalidad a servicios independientes de forma prematura, sino por modularizar el monolito existente con contextos bien definidos e interfaces claras.

Se propone alinear progresivamente la estructura interna del backend con los dominios identificados en el análisis DDD: los dominios core (Materiales y Fórmulas, Toxicológico, Regulatorio) y los dominios de soporte (Sustancias, IAM, Plataforma). Cada dominio se encapsularía en un módulo con límites explícitos, exponiendo interfaces bien definidas hacia los demás módulos y ocultando sus detalles de implementación internos.

Este enfoque de modularización interna aporta los beneficios de una arquitectura desacoplada (independencia de desarrollo, claridad de responsabilidades, testabilidad por módulo) sin incurrir en la complejidad operativa de los microservicios. Si en el futuro el crecimiento del equipo o la carga de la plataforma lo justifican, los módulos bien definidos podrán extraerse a servicios independientes con un esfuerzo significativamente menor.

Mejora de autorización

El sistema actual de autorización requiere una evolución significativa para soportar los requisitos de multi-tenancy y granularidad que exige la plataforma. Se detalla en profundidad en la Tarea 3 (Sección 5).

Las líneas principales incluyen:

- Unificación de anotaciones de seguridad
- Modelo de autorización en tres capas (Authenticated, Org, Suborg)
- Introducción de capacidades como eje ortogonal a los roles
- Filtrado automático de datos por tenant

Auditoría de datos

La activación de auditoría sobre las entidades críticas del dominio es una prioridad alta dada la naturaleza regulatoria del sector cosmético. La normativa europea exige trazabilidad completa de

los cambios en la formulación de productos cosméticos, y la ausencia actual de auditoría en entidades clave como Substance, Pif y Regulation representa un riesgo de cumplimiento significativo.

Se recomienda:

- Activar auditoría mediante spring-data-envers (ya presente como dependencia del proyecto) en todas las entidades del dominio PLM: Formula, Substance, Pif, Regulation, RawMaterial, Impurity, Toxicology y Exposition.
- Implementar estrategias de particionamiento temporal o archivado para gestionar el crecimiento de las tablas de auditoría, dado que los campos TEXT grandes presentes en muchas entidades generarán un volumen significativo de datos históricos.
- Garantizar que los registros de auditoría incluyan el usuario que realizó el cambio, el timestamp de la operación y el contexto organizativo (organización y suborganización).
- Considerar la creación de vistas o endpoints específicos para consultar el historial de cambios, facilitando tanto la operativa interna como posibles inspecciones regulatorias.

Procesamiento asíncrono

La generación síncrona de documentos PIF es el cuello de botella más evidente de la plataforma. El método generateReport del PIFReportController ejecuta todo el proceso de forma síncrona en el hilo de la solicitud HTTP: recopilación de datos, generación del documento DOCX mediante Docx4j, conversión a PDF mediante LibreOffice y devolución del resultado. Este flujo puede tardar varios segundos bajo condiciones normales y constituye un riesgo de agotamiento de hilos bajo carga concurrente.

Se recomienda:

- Implementar generación asíncrona de PIF mediante Spring Async o un sistema de colas de mensajes. La solicitud del usuario retornaría inmediatamente con un identificador de trabajo, y el documento se generaría en segundo plano.
- Proporcionar al usuario un mecanismo de notificación (polling, Server-Sent Events o WebSocket) para informarle cuando el documento esté listo para descarga.
- Considerar la migración a un motor de generación de documentos más ligero y rápido (Typst), como se detalla en la Tarea 4 (Sección 6), lo que reduciría el tiempo de generación de 5 segundos a 500 milisegundos, mitigando significativamente el problema incluso sin procesamiento asíncrono.

Observabilidad y monitorización

La observabilidad actual de SIG-COMPLY se limita a logging básico con Log4j2, lo que resulta insuficiente para operar una plataforma SaaS de forma proactiva. Se recomienda implementar una estrategia de observabilidad completa basada en los tres pilares:

Métricas: Integrar Spring Boot Actuator y Micrometer para exponer métricas de la JVM, del servidor HTTP, de la base de datos y métricas personalizadas de negocio (número de PIFs generados, tiempos de generación, fórmulas creadas, usuarios activos). Estas métricas deben alimentar un sistema de monitorización como Prometheus/Grafana para visualización y alertas.

Logging estructurado: Evolucionar el registro básico actual hacia logging estructurado en formato JSON con correlación de solicitudes. Cada entrada de log debe incluir un identificador de correlación único (trace-id), el contexto de tenant (organización y suborganización) y el usuario. Esto facilita enormemente la depuración de problemas en producción y el análisis post-mortem de incidentes.

Trazado distribuido: Implementar trazado distribuido mediante OpenTelemetry para visualizar el flujo completo de cada solicitud a través de los diferentes componentes del sistema. Esto es especialmente valioso para diagnosticar problemas de rendimiento en flujos complejos como la generación de PIF, que involucra múltiples servicios y operaciones de base de datos.

Alertas: Configurar alertas proactivas sobre métricas críticas: tiempos de respuesta que superan umbrales definidos, tasas de error por encima de lo normal, uso de recursos (CPU, memoria, conexiones de base de datos) y métricas de negocio anómalas. El objetivo es detectar y resolver problemas antes de que impacten a los usuarios finales.

Tarea 3: Sistema de Permisos y Autorización

Contexto y situación actual

La revisión del código de SCATHA reveló que la lógica de autorización se encontraba duplicada y dispersa a través de los endpoints de la aplicación. En algunos casos, la autorización estaba completamente ausente. Para una plataforma multi-tenant que gestiona datos sensibles de múltiples organizaciones, esta situación representaba un riesgo de seguridad prioritario.

El primer paso fue formar al equipo de SKS en los principales modelos de control de acceso — RBAC (Role-Based Access Control), ABAC (Attribute-Based Access Control) y NGAC (Next Generation Access Control) — para que pudieran tomar decisiones informadas sobre la dirección del sistema de permisos.

Con ese marco teórico establecido, se desarrolló un piloto que centralizaba la autorización utilizando un patrón PEP/PDP (Policy Enforcement Point / Policy Decision Point) mediante AOP (Programación Orientada a Aspectos). El piloto introdujo los siguientes componentes:

- **AccessControlAspect (PEP):** Punto de enforcement que intercepta métodos de controlador anotados.
- **AccessPolicyService (PDP):** Punto de decisión que evalúa rol, operación y propiedad del recurso.
- **RequireService / RequireRole:** Anotaciones declarativas para verificar servicio contratado y rol del usuario respectivamente.
- **SuborganizationContextService:** Resolución de la suborganización activa desde el header X-Suborganization-Id.

Este piloto demostró la viabilidad del enfoque centralizado y validó el patrón PEP/PDP como base arquitectónica. Sin embargo, al profundizar en el análisis junto con el equipo, se identificó que los requisitos reales del sistema de permisos — multi-tenancy con scopes diferenciados, capacidades granulares por usuario, filtrado automático de datos por tenant — exceden el alcance del piloto y requieren un diseño más completo. La propuesta que se describe a continuación recoge ese diseño.

El enum de roles del piloto sigue una jerarquía simple: Admin > Write > WriteSelf > Read. El análisis del código revela que el rol Admin aparece siempre junto a Write/WriteSelf en las anotaciones y no tiene ningún endpoint exclusivo, funcionando en la práctica de forma idéntica a Write.

Scopes de autorización:

El sistema opera con cuatro scopes diferenciados, cada uno con su propia ruta de verificación independiente:

Scope	Descripción
AUTHENTICATED	Usuario autenticado. Aplicable a datos de referencia de lectura pública u operaciones que solo requieren un usuario válido.
PLATFORM	Administración global de la plataforma. Gestión de organizaciones y datos de referencia transversales.
ORG	Nivel de organización. Gestión de usuarios, suborganizaciones, asignación de servicios y vistas cross-suborg.
SUBORG	Scope principal para operaciones de negocio. Utilizado por la mayoría de los endpoints de la aplicación.

Requisitos y casos de uso identificados

El análisis del sistema actual, junto con las sesiones de trabajo con el equipo de SKS, ha permitido identificar los siguientes requisitos clave que el sistema de autorización debe cubrir:

1. **Control de acceso a servicios por usuario:** Como administrador de una organización, necesito poder determinar qué usuarios dentro de cada suborganización tienen acceso a cada servicio (por ejemplo, PIF, gestión de base de datos), en lugar del modelo actual donde todos los usuarios de una organización acceden a todos los servicios contratados.
2. **Acciones privilegiadas por usuario:** Como responsable de calidad, necesito que solo determinados usuarios puedan realizar acciones críticas como la firma de PIFs, sin que ello implique elevar su rol completo. El sistema debe soportar permisos especiales asignables individualmente, ortogonales a la jerarquía de roles.
3. **Autorización diferenciada por scope:** Como plataforma multi-tenant, el sistema debe diferenciar claramente entre operaciones a nivel de plataforma (administración global), organización (gestión organizativa) y suborganización (operaciones de negocio), aplicando rutas de verificación independientes para cada scope.
4. **Aislamiento de datos entre tenants:** Como organización cliente, necesito la garantía de que mis datos son inaccesibles para otras organizaciones. El aislamiento debe ser sistemático y centralizado, no dependiente de la implementación individual de cada consulta.
5. **Visibilidad cross-suborg para administradores:** Como administrador de organización, necesito poder consultar y gestionar datos de todas las suborganizaciones bajo mi organización, manteniendo al mismo tiempo el aislamiento para los usuarios regulares que solo deben operar dentro de su suborganización.
6. **Jerarquía de roles clara y coherente:** Como desarrollador, necesito un modelo de roles sin ambigüedades ni duplicaciones, donde cada rol tenga un propósito diferenciado y la lógica de autorización sea predecible en todos los escenarios de combinación de permisos.

Recomendaciones propuestas

Anotación unificada con scope obligatorio

Actualmente, la autorización de cada endpoint requiere dos anotaciones separadas: RequireService (que verifica que la organización tenga contratado el servicio) y RequireRole (que verifica el rol del usuario en la suborganización). Esta duplicación genera inconsistencias y aumenta la probabilidad de errores al configurar nuevos endpoints.

Se propone reemplazar ambas anotaciones por una única anotación Secured con campo scope obligatorio. El scope determina la ruta de autorización que se aplica, y los parámetros adicionales (rol requerido, servicio, capacidad) se especifican como atributos de la misma anotación.

La migración consiste en un refactor mecánico: cada par RequireService + RequireRole en los controladores existentes se convierte en un único Secured con el scope correspondiente. Los tres advices actuales del aspecto se refactorizan en un único advice que despacha según el scope declarado.

Eliminación de Admin y promoción a OrgAdmin

El análisis del código revela que el rol Admin a nivel de suborganización no tiene ningún endpoint exclusivo y funciona idénticamente a Write en el AccessPolicyService. Esta duplicación genera confusión en el modelo de dominio y complica la lógica de autorización sin aportar valor.

Se propone eliminar Admin del enum de roles de suborganización, simplificando los roles a tres: Read, Write y WriteSelf. Lo que era Admin se transforma en OrgAdmin, un rol a nivel de organización. El OrgAdmin tiene la capacidad de omitir todas las verificaciones de rol, servicio y capacidad para cualquier endpoint dentro de su organización, lo que refleja adecuadamente su función real de administrador organizativo.

Esta reclasificación resuelve el desajuste actual entre el modelo de dominio y el código, y establece una jerarquía de roles clara y coherente.

Capacidades como tercer eje de autorización

El sistema actual no tiene un mecanismo para otorgar permisos especiales a usuarios individuales que no encajen en la jerarquía de roles. Por ejemplo, la firma de PIFs es una acción que solo determinados usuarios deberían poder realizar, independientemente de su rol general.

Se propone modelar estos permisos especiales como capacidades: permisos sparse a nivel de endpoint, ortogonales a los roles. Los tres ejes para autorización SUBORG serían:

1. **Rol:** Read / Write / WriteSelf (baseline CRUD). Define el nivel base de acceso del usuario.
2. **Servicio:** Determina a qué servicios puede acceder el usuario (ej. "PIF", "Database Management"). Permite segmentar funcionalidad por usuario.
3. **Capacidad:** Acciones especiales asignables individualmente (ej. SIGN_PIF). Extensible para futuras necesidades sin modificar la estructura de roles.

Dado que el servidor utiliza endpoints estilo RPC (una acción por endpoint), las capacidades se mapean limpiamente a la anotación Secured como verificación declarativa.

Lógica OR entre roles de org y suborg

El modelo propuesto otorga a los usuarios dos roles independientes: uno a nivel de organización (opcional) y otro a nivel de suborganización. El rol de org es opcional – un usuario puede no tener rol a nivel de org, en cuyo caso solo se comprueba el rol de suborg.

Cuando ambos roles existen, se aplica una lógica OR: el sistema evalúa el acceso usando el rol de org y, si este no es suficiente, verifica el rol de suborg. El rol de org actúa como un piso – un usuario con Write a nivel de org puede escribir en cualquier suborganización de esa organización, aunque su rol específico de suborg sea Read.

Se ha elaborado una matriz de verificación exhaustiva que cubre todos los escenarios posibles de combinación de roles org/suborg, recursos propios/ajenos y recursos de la misma/diferente suborganización, garantizando que el comportamiento es predecible y correcto en todos los casos.

Filtrado de datos por tenant mediante Hibernate Filter

El aislamiento de datos entre tenants es crítico para una plataforma multi-organización. Actualmente, el filtrado se realiza de forma manual e inconsistente a nivel de consulta. Se propone implementar filtrado automático a nivel de persistencia usando Hibernate Filter, lo que garantiza que las restricciones de tenant se aplican de forma transparente a todas las consultas.

Se implementarán dos filtros:

- **Filtro de organización:** Siempre habilitado (excepto para PlatformAdmin). Garantiza que una organización nunca accede a datos de otra organización. Este filtro se aplica a todas las entidades con scope ORG o inferior.
- **Filtro de suborganización:** Habilitado condicionalmente. Se activa para usuarios regulares con X-Suborganization-Id presente, y se desactiva para OrgAdmins (que necesitan ver datos cross-suborg) o cuando el header está ausente.

Se ha evaluado y descartado el uso de PostgreSQL Row Level Security (RLS) por las siguientes razones: requiere un pool de conexiones separado para PlatformAdmin, dificulta los tests unitarios y divide la lógica de seguridad entre Java y scripts de migración de base de datos.

Como prerrequisitos para la implementación, es necesario convertir las 14 queries nativas existentes (que usan ILIKE + LIMIT) a JPQL, ya que los filtros de Hibernate solo se aplican a consultas gestionadas por el ORM. Se recomienda adicionalmente añadir tests arquitectónicos (por ejemplo, mediante ArchUnit) para prevenir futuras queries nativas en entidades con scope de tenant.

Header X-Tenant-Id para contexto de organización

El sistema actual deriva la organización únicamente del objeto de usuario autenticado. Esta aproximación resulta limitante para escenarios donde un OrgAdmin necesita operar en el contexto de una organización específica.

Se propone añadir un header X-Tenant-Id junto al existente X-Suborganization-Id. La organización ya no se derivará únicamente del usuario, sino que vendrá del header HTTP y se validará contra la membresía de organización del usuario. Este enfoque permite:

- Validación explícita del contexto organizativo en cada solicitud.
- Soporte para usuarios que pertenecen a múltiples organizaciones.
- Trazabilidad clara del contexto de tenant en los logs de la aplicación.

Endpoints unificados para todos los tipos de usuario

Se propone mantener un único conjunto de endpoints REST para todos los tipos de usuario. Usuarios regulares, OrgAdmins y PlatformAdmins usarán los mismos endpoints, diferenciando el comportamiento de filtrado según el rol y los headers proporcionados:

- **Usuario regular:** X-Tenant-Id y X-Suborganization-Id requeridos. Ve solo datos pertenecientes a su suborganización.
- **OrgAdmin:** X-Tenant-Id requerido, X-Suborganization-Id opcional. Sin header de suborg, ve todas las suborganizaciones de su organización. Con header de suborg, se limita a esa suborganización.
- **PlatformAdmin:** Ningún header requerido. Ve datos a través de todas las organizaciones.

Este diseño funciona porque las entidades de transferencia ya incluyen metadatos de suborganización en la respuesta, permitiendo vistas cross-suborg sin necesidad de endpoints dedicados para administradores.

Estimación de alcance

Area	Esfuerzo	Notas
Anotación Secured + reescritura aspecto	2-6 días	Reemplazar 3 advices por 1. Nueva anotación. Dispatch por scope.
Refactor AccessPolicyService	2 días	Lógica OR entre roles org/suborg. Capacidades. Bypass OrgAdmin.
Cambios modelo de datos	4-6 días	Nuevas tablas: user_service_in_suborg, user_capability, rol org. Eliminar Admin. Migraciones.
Conversión queries nativas	2-4 días	Convertir 14 queries ILIKE+LIMIT a JPQL.
Setup Hibernate Filter	2-4 días	Filtros en 5 entidades. Lógica de habilitación. Header X-Tenant-Id.
Migración anotaciones controlador	2-4 días	Reemplazar RequireService+RequireRole por Secured. Anotar endpoints desprotegidos.
Total estimado	**16-32 días**	**Cambio arquitectónico fundacional**

Nota: El trabajo de UI de administración (pantallas para asignar roles, servicios y capacidades a usuarios) deberá ser entregado de forma coordinada con el trabajo de backend. La estimación de UI se realizará una vez validado el diseño funcional.

Tarea 4: Sistema de Generación de PIF

Contexto y arquitectura actual

El sistema de generación de Product Information Files (PIF) es uno de los componentes más críticos de SIG-COMPLY, ya que produce los documentos regulatorios exigidos por la normativa europea.

Backend (Java/Spring Boot):

La generación de PIF se apoya en dos componentes principales:

- **ReportingService:** Orquestador principal que coordina la recopilación de datos y la generación del documento.
- **EngineDocx4j:** Motor de renderizado basado en Docx4j con aproximadamente 1.700 líneas de código, que genera documentos Word que posteriormente se convierten a PDF mediante LibreOffice.

El contenedor de datos PIFReport tiene más de 60 campos y el constructor de secciones BuilderPIFReport comprende aproximadamente 1.300 líneas de código. Todo el modelo de datos se distribuye en 17 clases Java.

Frontend (Angular):

El módulo pif-generator contiene un componente principal de aproximadamente 1.600 líneas de código, organizado en 6 pasos de wizard (título, producto principal, descripción, seguridad, parte B y otra información), 7 componentes de anexos y utilidades auxiliares de aproximadamente 600 líneas de código.

Limitaciones actuales

Problema	Impacto
PIFReport monolítico	Más de 60 campos en una única clase, difícil de mantener
Secciones acopladas	No es posible renderizar secciones de forma independiente
Estilos embebidos en código	Estilos hardcodeados en Java (EngineDocx4j)
Inflexibilidad de plantillas	La plantilla DOCX requiere el documento completo
Reutilizabilidad limitada	Headers, footers y tablas duplicados entre secciones
Lógica de renderizado compleja	Más de 2.000 líneas de manipulación Docx4j
Sin selección de secciones	Todas las secciones se incluyen siempre
Verbosidad de Docx4j	API compleja para operaciones simples
Dependencia de LibreOffice	La conversión DOCX a PDF introduce dependencias de sistema y problemas de robustez
Proceso síncrono	La generación bloquea hilos web, provocando tiempos de espera bajo carga

Dirección propuesta: Plantillas Typst modulares

Se propone migrar el sistema de generación de PIF de Docx4j/LibreOffice a una arquitectura basada en plantillas Typst modulares.

Arquitectura propuesta:

La nueva arquitectura se estructura en las siguientes capas:

- **Frontend (Angular):** El wizard PIF existente se extiende con un componente de selección de secciones que permite al cliente elegir que secciones incluir en el PIF.
- **PifController:** Nuevos endpoints para generación completa y preview por sección.
- **PifOrchestrationService:** Coordinador que gestiona el ensamblaje de secciones y la selección.
- **Servicios de sección modulares:** Cada sección del PIF se implementa como un servicio independiente (CoverPageSectionService, ProductDescriptionSectionService, SafetyReportPartASectionService, etc.), cada uno responsable de extraer y validar los datos de su sección.
- **TypstRenderingEngine:** Servicio que compila plantillas Typst y genera la salida PDF directamente, eliminando la dependencia de LibreOffice.
- **Sistema de plantillas Typst:** Plantillas modulares organizadas en una biblioteca de componentes reutilizables (tema, tipografía, layout, tablas) y plantillas de sección individuales.

Estructura del documento PIF:

El PIF es un documento regulatorio complejo que se compone de las siguientes secciones principales:

- **Portada:** Título, logo, versión del documento.
- **Definición de producto principal:** Datos identificativos del producto cosmético.
- **Sección 1 – Descripción del Producto:** Incluye descripción general, fórmulas de equivalencia y fórmulas de comparación.
- **Sección 2 – Informe de Seguridad Parte A:** Con 15 subsecciones que cubren composición cualitativa y cuantitativa, características físico-químicas, estabilidad, compatibilidad con packaging, PAO (periodo después de apertura), calidad microbiológica, challenge test, impurezas y trazas, material de embalaje, uso normal y previsible, evaluación de exposición, perfil toxicológico, margen de seguridad, efectos indeseables e información de producto cosmético.
- **Sección 3 – Informe de Seguridad Parte B:** Conclusión de la evaluación, advertencias de etiquetado, razonamiento (con más de 10 subsecciones) y credenciales del evaluador.
- **Sección 4 – Otra Información:** Método de fabricación, buenas prácticas de fabricación, prueba de efectos alegados e información sobre ensayos con animales.
- **Anexos I-VII:** Documentación complementaria.

La modularización propuesta permite que cada una de estas secciones se renderice de forma independiente, facilitando tanto la generación del documento completo como la previsualización de secciones individuales durante la edición.

Sistema de diseño Typst:

Las plantillas se apoyan en un sistema de diseño completo implementado en Typst que garantiza consistencia visual en todo el documento. Este sistema define:

- **Paleta de colores:** Colores primarios, secundarios, de acento, texto, bordes y estados (éxito, advertencia, error).
- **Escala tipográfica:** Desde 8pt (extra small) hasta 28pt (display), con tamaños intermedios para cada nivel jerárquico del documento.
- **Espaciado:** Escala de espaciado consistente para márgenes, padding y separación entre elementos.
- **Estilos de bordes:** Definición de grosores y colores para tablas, separadores y cajas.
- **Componentes reutilizables:** Tablas estándar con cabeceras, tablas clave-valor, tablas de composición de ingredientes, encabezados de sección, pies de página y utilidades de layout.

Este sistema de diseño permite que cualquier miembro del equipo pueda crear o modificar plantillas de sección sin necesidad de conocimientos de Java, utilizando únicamente la sintaxis de markup de Typst y los componentes predefinidos.

Beneficios

Actual (Docx4j)	Propuesto (Typst)
1.700 LOC motor de renderizado	200 LOC servicio de renderizado
Plantillas DOCX binarias	Plantillas basadas en texto (compatibles con git)
API Java compleja	Sintaxis de markup limpia
Documento monolítico	Secciones modulares
Todas las secciones obligatorias	Selección por cliente
Compilación lenta (5s)	Compilación rápida (500ms)
Salida DOCX (conversión a PDF)	Salida PDF directa

Beneficios de negocio:

- **Flexibilidad:** Los clientes pueden personalizar que secciones incluir en sus PIFs.
- **Mantenibilidad:** Las plantillas se pueden actualizar sin conocimientos de Java.
- **Consistencia:** El sistema de diseño garantiza un estilo uniforme.
- **Rendimiento:** Generación de documentos 10x más rápida.
- **Calidad:** Tipografía y calidad de PDF superior a la conversión DOCX.

Ruta migratoria

Se propone una migración en 6 fases con una duración total estimada de 12 semanas:

Fase	Descripción	Duración
1	Infraestructura Typst: Añadir Typst a dependencias del proyecto, crear TypstRenderingEngine, implementar estructura básica de plantillas y test simple	1 semana
2	Sistema de diseño: Definir tokens de tema, implementar componentes reutilizables (tablas, header, footer), crear utilidades de tipografía y layout	1 semana
3	Plantillas de sección: Portada, descripción de producto, 15 subsecciones de Parte A, Parte B, otra información y plantillas de anexos	4 semanas
4	Servicios backend: Crear interfaces de servicio de sección, implementar cada servicio, construir servicio de orquestación y nuevos endpoints REST	2 semanas
5	Integración frontend: Componente de selección de secciones, actualización del servicio PIF, funcionalidad de preview y actualización del flujo del generador	2 semanas
6	Testing y migración: Comparación paralela Docx4j vs Typst, testing de regresión visual, benchmarking de rendimiento, rollout gradual con feature flag y deprecación de Docx4j	2 semanas

Tarea 5: Versionado de Fórmulas Cosméticas

Contexto

El versionado de fórmulas cosméticas es un requisito fundamental para la trazabilidad regulatoria en el sector. Actualmente, el modelo de datos de SCATHA trata la formula como una entidad única sin soporte nativo de versiones, lo que dificulta el seguimiento de la evolución de las formulaciones a lo largo del tiempo.

Limitaciones actuales

El modelo de datos actual presenta limitaciones significativas para la gestión del ciclo de vida de fórmulas cosméticas:

- **Ausencia de versionado:** No existe una entidad de versión de formula. Cualquier modificación a una formula sobreescribe los datos anteriores, perdiendo el histórico de cambios. En un sector regulado donde la trazabilidad es un requisito legal, esta carencia representa un riesgo de cumplimiento.
- **Composición no versionada:** La composición química (materias primas, sustancias, impurezas) está vinculada directamente a la entidad Formula, sin posibilidad de mantener composiciones históricas. Si se modifica un ingrediente, no hay forma de consultar la composición anterior.
- **Acoplamiento comercial-técnico:** Los PIFs, mercados y otros datos comerciales están acoplados directamente a la formula. Esto impide que múltiples productos comerciales compartan la misma formulación (una práctica habitual en el sector cosmético, donde un mismo producto se comercializa bajo diferentes marcas o presentaciones).
- **Ausencia de logs de cambio:** No hay un registro de log que documente los cambios realizados en las fórmulas y productos a lo largo del tiempo, dificultando tanto la auditoría interna como la respuesta ante inspecciones regulatorias.
- **Modelo de fabricante limitado:** La información de fabricación se almacena de forma dispersa, sin una entidad dedicada que permita gestionar diferentes tipos de fabricantes (fabricante principal, subcontratista, laboratorio de control).

Dirección propuesta: FormulaVersion como entidad principal

Se propone una refactorización del modelo de datos en 7 fases, con una duración total estimada de 6-7 semanas:

Fase	Enfoque	Duración
1	Entidad FormulaVersion	1 semana
2	Composición en FormulaVersion	1 semana
3	Introducción de ComercialProduct	1 semana
4	Refactorización de Manufacturer y Packaging	1 semana
5	División de Exposition	2-3 días
6	Endpoints de gestión de versiones	3-4 días
7	Tablas de Log (PiFLog, ProductLog)	3-4 días

Fase 1: Entidad FormulaVersion
La primera fase establece la base del versionado creando la entidad FormulaVersion y vinculándola a Formula mediante una relación OneToMany. Cada FormulaVersion representa un snapshot inmutable de la formulación en un momento dado. La relación de Markets se mueve de Formula a FormulaVersion, ya que los mercados de destino pueden variar entre versiones.

Un principio clave de diseño es la retrocompatibilidad: los endpoints existentes de la API continúan devolviendo la última versión por defecto, sin cambios en su interfaz pública. Se añade una propiedad opcional de versión en el cuerpo de las peticiones POST y como query param en las peticiones GET, permitiendo a los consumidores adoptar el versionado de forma progresiva.

Fase 2: Composición en FormulaVersion
La segunda fase traslada la composición química de la entidad Formula a FormulaVersion. Se crean tres entidades intermedias: FormulaVersion_RawMaterial, FormulaVersion_Substance y FormulaVersion_Impurity. Este cambio permite que cada versión de la formula tenga su propia composición independiente, lo que es fundamental para la trazabilidad regulatoria.

Los servicios de cálculo (CalculationService, ToxicologyCalculationService) se actualizan para recibir FormulaVersion como contexto en lugar de Formula, asegurando que los cálculos se realizan sobre la composición de la versión correcta.

Fase 3: ComercialProduct
La tercera fase introduce la entidad ComercialProduct, que desacopla los datos comerciales de la formulación técnica. Esta separación refleja la realidad del sector cosmético, donde una misma formulación puede comercializarse bajo múltiples nombres, marcas y presentaciones.

Los PIFs se mueven de Formula a ComercialProduct, ya que el PIF es un documento asociado a un producto comercial concreto. Se crea adicionalmente la entidad Label para gestionar el etiquetado. Múltiples productos comerciales pueden compartir la misma FormulaVersion, y cada producto puede vincular diferentes versiones de formula a lo largo de su ciclo de vida.

Fase 4: Manufacturer y Packaging
Se crea la entidad Manufacturer con discriminación de tipo (fabricante, subcontratista, etc.), consolidando la información de fabricación que actualmente se encuentra dispersa en varios campos. Los campos de Specifications se refactorizan a relaciones propias, mejorando la normalización del modelo. Packaging se modifica a una relación ManyToMany con ComercialProduct, reflejando que un mismo packaging puede utilizarse por múltiples productos.

Fase 5: División de Exposition
La entidad Exposition actual mezcla datos de exposición general con datos específicos por formulación. Se propone dividirla en dos entidades: una para los datos de exposición generales (comunes a múltiples productos) y otra para los datos específicos de cada formulación. Esta separación mejora la reutilizabilidad y reduce la redundancia de datos.

Fase 6: Endpoints de gestión de versiones
Se crean nuevos endpoints bajo la ruta /api/formula-version/ para las operaciones específicas de versionado: crear nueva versión (a partir de la última o de una versión específica), clonar versión e historial de versiones. Estos endpoints coexisten con los endpoints existentes de /api/formula/, que continúan operando sobre la última versión por defecto.

Fase 7: Tablas de Log
Se implementan las entidades PiFLog y ProductLog para proporcionar trazabilidad completa de las operaciones realizadas sobre fórmulas y productos a lo largo del tiempo. Cada entrada de log registra la acción realizada, el usuario, el timestamp y los metadatos relevantes. Se proporciona una interfaz de usuario para consultar el historial de cambios, facilitando la auditoría interna y la respuesta ante inspecciones regulatorias.

Análisis de dependencias

El impacto de la migración se ha analizado en detalle para minimizar disrupciones:

Comportamiento de la API:

- Los endpoints existentes permanecen sin cambios en su interfaz.
- Los endpoints POST aceptan una propiedad opcional de versión en el cuerpo de la petición.
- Los endpoints GET aceptan un query param opcional de versión.
- Si no se especifica versión, se opera sobre la última.

Dependencias de backend:

Controlador	Endpoints	Cambio
FormulaController	27	Añadir soporte de versiones
CalculationsController	5	Usar última versión
ToxicologyCalculationController	1	Usar última versión
PIFReportController	1	Usar última versión
Formula_RawMaterialController	2	Fase 2 - mover a FormulaVersion

Dependencias de servicios:

Servicio	Cambio
FormulaService	Añadir helper de resolución de versión
Formula_RawMaterialService	Fase 2 - mover a FormulaVersion
Formula_SubstanceService	Fase 2 - mover a FormulaVersion
CalculationService	Pasar contexto de FormulaVersion
ToxicologyCalculationService	Pasar contexto de FormulaVersion
PifTablesService	Usar última versión

Dependencias de frontend:

- FormulaService: Añadir versión opcional a métodos.
- FormulaResponse.ts: Añadir propiedad versión opcional.
- Otros servicios y componentes: Sin cambios inicialmente.

> **Principio de diseño:** La migración se diseña para ser completamente retrocompatible. Ningún endpoint existente cambia su comportamiento por defecto; el soporte de versiones se añade como funcionalidad opcional que los consumidores pueden adoptar progresivamente.

Conclusiones

La auditoría técnica realizada sobre la plataforma SIG-COMPLY ha permitido identificar tanto las fortalezas como las áreas de mejora críticas del sistema.

Fortalezas identificadas:

- Arquitectura backend bien estructurada con Spring Boot y buena separación de paquetes.
- Autenticación robusta mediante Auth0/OpenID Connect.
- Uso de tecnologías maduras y bien soportadas (Spring Data JPA, Hibernate, Angular).
- Base de código con estructura modular y uso adecuado de Lombok.

Áreas críticas de mejora:

- **Autorización:** El sistema actual carece de la granularidad necesaria para un entorno multi-tenant. La implementación propuesta de autorización en tres capas con capacidades resuelve esta carencia de forma integral.
- **Generación de documentos:** La dependencia de Docx4j y LibreOffice introduce fragilidad y limitaciones de rendimiento. La migración a Typst ofrece una mejora significativa en velocidad, mantenibilidad y flexibilidad.
- **Versionado de fórmulas:** La ausencia de versionado nativo es una limitación crítica para la trazabilidad regulatoria. La propuesta de FormulaVersion como entidad principal resuelve esta necesidad de forma retrocompatible.
- **Observabilidad:** La falta de métricas, trazado distribuido y manejo de errores completo limita la capacidad de operar la plataforma de forma proactiva.

Actividades transversales realizadas:

Además de las cinco tareas de auditoría, Catallactical ha contribuido al fortalecimiento del equipo de desarrollo de SKS mediante:

- **Orientación en Domain-Driven Design:** Análisis detallado de dominios y bounded contexts, con recomendaciones para alinear la arquitectura del código con los procesos de negocio del sector cosmético.
- **Implantación de metodología ágil:** Asistencia en la adopción de prácticas ágiles, organización del trabajo en iteraciones, definición de épicas y gestión del backlog de producto.
- **Mentorización técnica:** Asignación de un ingeniero senior de nivel staff para la supervisión continua del equipo de desarrollo, incluyendo revisiones de código, sesiones de formación y elaboración de documentación técnica. Entre las guías producidas destacan la guia de convenios de entidades JPA (estrategias de fetch, relaciones bidireccionales, tipos de cascade, gestión de referencias circulares) y la guia de arquitectura por capas (modelo Entity/Domain/DTO con MapStruct), ambas integradas en la documentación del proyecto.

Resumen de esfuerzo estimado:

Tarea	Esfuerzo estimado	Impacto principal
Sistema de autorización (Tarea 3)	16-32 días	Seguridad y multi-tenancy
Generación de PIF (Tarea 4)	12 semanas	Rendimiento y mantenibilidad
Versionado de fórmulas (Tarea 5)	6-7 semanas	Trazabilidad regulatoria
Auditoría de datos	1-2 semanas	Cumplimiento normativo
Observabilidad	2-3 semanas	Operabilidad en producción

Priorización recomendada:

1. **Corto plazo (inmediato):** Activar auditoría de datos en entidades críticas del dominio PLM, dado el riesgo de cumplimiento normativo que supone su ausencia. Completar el manejador de excepciones global para evitar filtraciones de información sensible en respuestas de error.
2. **Medio plazo (1-3 meses):** Implementar el nuevo sistema de autorización (16-32 días de desarrollo estimados), ya que es un prerrequisito para el crecimiento de la base de usuarios y la incorporación de nuevas organizaciones. Iniciar en paralelo la migración del generador de PIF a plantillas Typst modulares.
3. **Medio-largo plazo (3-6 meses):** Ejecutar el plan de versionado de fórmulas cosméticas (6-7 semanas), que es fundamental para la trazabilidad regulatoria a largo plazo. Implementar la estrategia de observabilidad completa (métricas, logging estructurado, trazado distribuido).
4. **Largo plazo (6-12 meses):** Desacoplamiento progresivo del monolito en función del crecimiento del equipo y la carga de la plataforma. Separación frontend/backend como primer paso, seguida de la extracción de servicios intensivos si el volumen de operaciones lo justifica.

Valoración global:

SIG-COMPLY cuenta con una base técnica sólida que, con las mejoras propuestas, puede evolucionar hacia una plataforma de referencia en el sector cosmético. Las recomendaciones se han diseñado para ser implementables de forma incremental, sin interrumpir la operación actual, y con un enfoque pragmático que prioriza el valor de negocio sobre la perfección técnica.

> La implementación de las mejoras propuestas posicionará a SIG-COMPLY como una plataforma robusta, escalable y preparada para el crecimiento, cumpliendo con los estándares de calidad y seguridad que exige el sector cosmético regulado.