Pontificia Universidad Javeriana Cali
Facultad de Ingeniería.
Maestría en Ingeniería de Software
Proyecto de Grado.

Diseño de arquitectura de software para la consulta de información crediticia a centrales de riesgo en Colombia en 2025. Caso: Agroinsumos del Cauca

Andres Garcia - Oscar Rodriguez

Director(a): MSc Emmanuel Tarcisio Olivos Solis

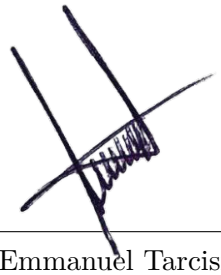
28 de Julio de 2025

Santiago de Cali, 28 de Julio del 2025.

Señores
Pontificia Universidad Javeriana Cali.
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software.
Cali.

Cordial Saludo.

Por medio de la presente hago constar que en mi calidad de director de trabajo de grado he revisado el proyecto titulado "Diseño de arquitectura de software para la consulta de información crediticia a centrales de riesgo en Colombia en 2025. Caso: Agroinsumos del Cauca" realizado por el estudiante de Magister en Ingeniería de Software Andres Garcia - Oscar Rodriguez (cod: 9012980 - 9012482), el cual se encuentra terminado y considero que cumple con los requisitos para ser sustentado.
Atentamente,



MSc Emmanuel Tarcisio Olivos Solis

Santiago de Cali, 28 de Julio de 2025.

Señores
Pontificia Universidad Javeriana Cali
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software
Cali.

Cordial Saludo.

Me permito presentar a su consideración el proyecto de grado titulado "Diseño de arquitectura de software para la consulta de información crediticia a centrales de riesgo en Colombia en 2025. Caso: Agroinsumos del Cauca" con el fin de cumplir con los requisitos exigidos por la Universidad y para que sea sometido a revisión del jurado y cumpla su aprobación, para conseguir posteriormente el título de Magister en Ingeniería de Software.

Atentamente,

Andres Garcia - Oscar Rodriguez
Código: 9012980 - 9012482

Ficha Resumen
Trabajo de Grado Maestría en Ingeniería de Software

TíTULO: Diseño de arquitectura de software para la consulta de información crediticia a centrales de riesgo en Colombia en 2025. Caso: Agroinsumos del Cauca

1. Énfasis: Ingeniería de Software
2. Área de trabajo: Arquitectura y Diseño de Sistemas de Información (Gestión de Riesgo Crediticio)
3. Tipo de proyecto (Aplicado, Innovación, Investigación): Aplicado
4. Estudiante: Andres Garcia - Oscar Rodriguez
5. Correo electrónico: oscarpino23@javerianacali.edu.co - adga010@javerianacali.edu.co
6. Dirección y teléfono: Calle 6a n 50 - 40 - 3183395663
7. Director:MSc Emmanuel Tarcisio Olivos Solis
8. Vinculación del director: Externo
9. Correo electrónico del director: eolivos@etar.com.mx
10. Palabras clave(al menos 5): Arquitectura de software, Riesgo crediticio, Modelo C4, Consulta en tiempo real, Protección de datos.
11. ODS que aplica al proyecto (Agenda 2030): Contribuye a la meta 9.1 (desarrollar infraestructuras fiables, sostenibles y de calidad) al proponer una plataforma tecnológica robusta para el sector agroindustrial; a la meta 9.4 (modernizar infraestructuras y adoptar tecnologías limpias y respetuosas con el medio ambiente) mediante la migración a servicios en la nube y prácticas DevOps eficientes; y a la meta 9.c (aumentar el acceso a las TIC) al facilitar el uso de servicios digitales de información crediticia en zonas rurales.
12. Fecha de inicio: 15 Agosto 2024

Agradecimientos

Agradecimientos específicos: Deseo expresar, ante todo, mi más profundo agradecimiento a todas las personas e instituciones que hicieron posible la culminación de este proyecto de grado y el logro de esta meta académica.

A mi familia, por su apoyo incondicional, sus palabras de aliento y la confianza que depositaron en mí durante cada etapa de este proceso; su cariño y comprensión fueron el motor que impulsó mi perseverancia.

Al MSc Emmanuel Tarcisio Olivos Solis, director de este trabajo, por su guía experta, sus observaciones siempre oportunas y la paciencia con la que orientó mis ideas hasta convertirlas en un proyecto sólido y significativo.

A la Ph.D. Luisa Rincón, directora de la Maestría en Ingeniería de Software, por encabezar un programa académico que fomenta la excelencia y por el respaldo brindado a lo largo de todo el posgrado.

A los profesores de la Facultad de Ingeniería de la Pontificia Universidad Javeriana Cali, cuyas enseñanzas y ejemplo profesional ampliaron mi perspectiva y sentaron las bases técnicas y metodológicas necesarias para llevar a buen puerto esta investigación.

A mis compañeros de cohorte y colegas de trabajo, por las conversaciones enriquecedoras, la colaboración desinteresada y el apoyo mutuo que hicieron más llevadero el camino.

A los profesionales y expertos del sector crediticio y agrícola que compartieron su tiempo y conocimiento durante entrevistas y sesiones de consulta; sus valiosos aportes enriquecieron la calidad y pertinencia de este estudio.

Finalmente, agradezco a la Pontificia Universidad Javeriana Cali por ofrecer un entorno académico de alta calidad y por fomentar la investigación aplicada con impacto social y empresarial. Este proyecto ha sido no solo un reto intelectual sino también un viaje de crecimiento personal que refuerza mi compromiso con la innovación responsable y el desarrollo tecnológico de nuestro país.

Abstract

This project designs and evaluates a software architecture that enables Agroinsumos del Cauca—a Colombian agribusiness distributor—to consult credit-risk bureaus in real time and make data-driven lending decisions. Beginning with a detailed problem definition that underscores the firm's current inability to gauge customer solvency and the legal duties imposed by Laws 1581 (2012) and 1266 (2008), the study specifies functional and non-functional requirements, business rules, and stakeholder expectations. Three architectural patterns—Layered, Hexagonal and Microservices—are compared through quantitative scorecards covering scalability, complexity and security. A modular Layered Architecture is selected for the minimum viable product because it offers the best cost-benefit ratio (48 / 60 points), fastest time-to-market and straightforward implementation, while still permitting future evolution toward microservices if transaction volume grows. The proposed solution, modelled with the C4 method down to component level, combines a Spring Boot back end, Angular front end and PostgreSQL database, secured through HTTPS, RBAC, MFA and encrypted storage. Performance and security benchmarks confirm that the prototype sustains at least 20 concurrent requests per second with p95 latency below 500 ms, 95 % service availability and 90 % compliance with ISO 27001 / OWASP controls. A user-centred interface and comprehensive audit trail enable transparent credit scoring, and expert evaluation suggests it can shorten approval times and lower default risk. Overall, the architecture furnishes Agroinsumos del Cauca with a scalable, regulation-compliant platform that can be incrementally extended, offering a practical blueprint for SMEs seeking reliable access to credit information in emerging markets.

Keywords: software architecture, credit risk, C4 model, layered architecture, data protection

Resumen

Este proyecto diseña y valida una arquitectura de software que permite a Agroinsumos del Cauca consultar en tiempo real la información crediticia de sus clientes en las centrales de riesgo, reduciendo incertidumbre y cartera vencida. Tras identificar la ausencia de un mecanismo de evaluación crediticia y las exigencias de las Leyes 1581 y 1266, se recolectaron requisitos, reglas de negocio y actores. Se compararon tres patrones arquitectónicos (Capas, Hexagonal y Microservicios) según escalabilidad, complejidad y seguridad; la arquitectura en capas resultó la más conveniente para el MVP. La solución, modelada con C4, emplea Spring Boot, Angular y PostgreSQL, con comunicaciones cifradas, control de acceso y trazabilidad. Las pruebas confirman latencia bajo 500 ms y 90 % de cumplimiento ISO 27001/OWASP, demostrando viabilidad y posibilidad de evolución.

Palabras clave: arquitectura de software, riesgo crediticio, modelo C4, arquitectura en capas, protección de datos

Índice general

Agradecimientos **7**

1. Introducción **1**
1.1. Definición del problema . . . 2
1.1.1. Planteamiento del problema . . . 2
1.2. Objetivo General y Objetivos Específicos . . . 2
1.2.1. Objetivos General . . . 2
1.2.2. Objetivos Específicos . . . 2
1.3. Alcance . . . 3
1.4. Justificación del problema . . . 3
1.5. Metodología de la investigación . . . 3
1.6. Estrategias de recolección y validez . . . 3
1.7. Consideraciones éticas y de cumplimiento . . . 4
1.8. Resultados obtenidos . . . 4
1.8.1. Introducción . . . 4
1.8.2. Alineación de necesidades, reglas de negocio y requisitos (OE 1 y OE 2) . . . 4
1.8.3. Modelo arquitectónico propuesto (OE 3) . . . 5
1.8.4. Prototipo de interfaz de usuario (OE 4) . . . 6
1.8.5. Modelo de seguridad y cumplimiento (OE 5) . . . 6
1.8.6. Conclusión e impacto . . . 7

2. Marco de referencia **9**
2.1. Marco Teórico . . . 9
2.1.1. Transformación Digital . . . 9
2.1.2. Gestión del Riesgo Crediticio . . . 9
2.1.3. Tecnologías de Integración de Sistemas . . . 9
2.1.4. Experiencia de usuario (UX) . . . 10
2.1.5. Modelo C4 para el Diseño de Arquitectura de Software . . . 10
2.1.6. Detalles sobre Tecnologías Específicas . . . 11
2.2. Estado del Arte . . . 13
2.2.1. Importancia de la Economía Agrícola en Colombia . . . 13
2.2.2. Sistemas Administrativos para Empresas Agrícolas . . . 13
2.2.3. Seguridad de la Información crediticia y Protección de Datos . . . 14
2.2.4. Consideraciones Legales y Regulatorias . . . 14
2.2.5. Ejemplos de Implementaciones Exitosas . . . 15
2.3. Resumen del capítulo – Estado del arte . . . 15

3. Desarrollo del Proyecto **17**
3.1. Desarrollo del proyecto . . . 17
3.2. Estrategia global de desarrollo . . . 17
3.3. Plan de trabajo y cronograma . . . 17
3.4. Refinamiento de requisitos y backlog . . . 18
3.5. Implementación del MVP . . . 18
3.6. Pruebas y validación . . . 19
3.7. Gestión de la configuración y despliegue . . . 19
3.8. Control de calidad y métricas . . . 19
3.9. Lecciones aprendidas intermedias . . . 19
3.10. Resumen del capítulo . . . 20

4. Análisis de requisitos **21**
4.1. Definición de Producto mínimo viable. . . . 21
4.1.1. Alcance Principal . . . 22
4.1.2. Características Iniciales del MVP . . . 22
4.1.3. Exclusiones de la Versión MVP . . . 22
4.1.4. Justificación del MVP . . . 22
4.2. Recopilación de requisitos . . . 23
4.2.1. Requisitos funcionales . . . 23
4.2.2. Requisitos no funcionales . . . 24
4.3. Criterios de aceptación . . . 24
4.4. Recopilación Reglas de negocio . . . 26
4.5. Definición de Restricciones Técnicas . . . 27
4.6. Identificación de Stakeholders y sus Necesidades . . . 28
4.6.1. Equipo Directivo / Gerencia General . . . 28
4.6.2. Área Comercial . . . 28
4.6.3. Área Financiera / Crédito y Cartera . . . 29
4.6.4. Equipo de TI / Desarrolladores . . . 29
4.6.5. Centrales de Riesgo / Proveedores Externos . . . 29
4.6.6. Clientes de Agroinsumos del Cauca . . . 29
4.6.7. Entidades Reguladoras / Gubernamentales . . . 30
4.6.8. Auditoría Interna / Control Interno . . . 30

5. Selección de tecnologías **31**
5.1. Investigación de patrones de arquitectura . . . 31
5.2. Criterios de evaluación . . . 32
5.3. Investigación: Arquitectura vs. Criterio . . . 34
5.4. Resultados . . . 40
5.4.1. Elección recomendada . . . 41
5.4.2. Arquitecturas no seleccionadas . . . 42

5.5. Conclusión . . . 42
5.6. Lenguaje de Programación (Back-Front) . . . 43
5.6.1. Front-End: JavaScript . . . 43
5.6.2. Back-End: Criterios de Evaluación . . . 43
5.7. Investigación de frameworks . . . 46
5.7.1. Frameworks para Backend . . . 46
5.7.2. Frameworks para Frontend . . . 46
5.8. Definición tipos de bases de datos . . . 48
5.8.1. Criterios de Selección . . . 48
5.8.2. Bases de Datos Relacionales . . . 48
5.8.3. Bases de Datos No Relacionales (NoSQL) . . . 50
5.9. Selección de Infraestructura . . . 51

6. Modelado de la arquitectura 53
6.1. Definición de componentes . . . 53
6.1.1. Identificación de componentes . . . 53
6.1.2. Definición del patrón de arquitectura. . . . 55
6.1.3. Definición de interfaces. . . . 57
6.1.4. Diagramación de arquitectura. . . . 61

7. Conclusiones 65
7.1. Trabajos futuros . . . 66
7.2. Lecciones aprendidas . . . 67

8. Anexos 69
8.1. Recopilación de requisitos . . . 69
8.1.1. Requisitos funcionales . . . 69
8.1.2. Requisitos no funcionales . . . 71
8.2. Criterios de aceptación . . . 73
8.2.1. Gestión de Usuarios y Acceso . . . 73
8.2.2. Consulta a Centrales de Riesgo . . . 74
8.2.3. Registro de Actividades y Auditoría . . . 75
8.2.4. Seguridad y Protección de Datos . . . 76
8.2.5. Mantenibilidad y Actualizaciones . . . 76
8.3. Integración TransUnion IDVision – Extracto técnico . . . 77
8.3.1. Endpoints HTTPS (TLS 1.2) . . . 77
8.3.2. Campos mínimos del request `ExecuteXMLString` . . . 78
8.3.3. Códigos de respuesta relevantes . . . 78
8.4. Integración TransUnion AQM Go Habeas Data – Extracto técnico . . . 78
8.4.1. Endpoints HTTPS (TLS 1.2) . . . 78
8.4.2. Campos obligatorios del request (Prospección + Cuota) . . . 79
8.4.3. Variables clave del resultado . . . 79

Bibliografía 81

Índice de figuras

4.1. Requisitos funcionales del sistema 23
4.2. Requisitos no funcionales del sistema 24
4.3. Gestión de usuarios y acceso 24
4.4. Consulta a centrales de riesgo 25
4.5. Mantenibilidad y actualizaciones 25
4.6. Registro de actividades y auditoria 26
4.7. Seguridad y protección de datos 26

6.1. Inicio session. 57
6.2. Vision general. 57
6.3. Detalles cliente. 58
6.4. Historial de consulta. 58
6.5. Reporte de crédito. 59
6.6. Gestión de usuario. 59
6.7. Autorizaciones. 60
6.8. Diagrama de datos. 61
6.9. Diagrama de negocio. 62
6.10. Diagrama de aplicaciones. 62
6.11. Diagrama de tecnología. 63

Índice de tablas

4.1. Datos requeridos por Agroinsumos 21

5.1. Comparación de escalabilidad entre arquitecturas 32
5.2. Evaluación de complejidad usando Cynefin 33
5.3. Evaluación de cumplimiento de seguridad 34
5.4. Escalabilidad de la Arquitectura en Capas (Monolito Modular) 35
5.5. Evaluación de Complejidad – Arquitectura en Capas 36
5.6. Evaluación de Seguridad – Arquitectura en Capas (Check-list ISO / OWASP) 37
5.7. Escalabilidad: Arquitectura de Microservicios (Comparación uniforme) 38
5.8. Evaluación de Complejidad - Arquitectura de Microservicios 39
5.9. Check-list de seguridad para microservicios (ISO 27001, OWASP Top-10, Zero Trust) 40
5.10. Comparación global: Capas vs. Microservicios 41
5.11. Comparación de desempeño entre lenguajes en entornos concurrentes 43
5.12. Comparación del ecosistema y actividad comunitaria por lenguaje 44
5.13. Comparación de características clave por lenguaje 45
5.14. Comparación de frameworks front-end según rendimiento y curva de aprendizaje . . 47
5.15. Comparación de rendimiento en lectura y escritura por motor de base de datos . . . 49
5.16. Requerimientos técnicos recomendados por componente 51

6.1. Integración de artefactos TOGAF con el proyecto 56

8.1. Puntos de conexión para IDVision. 77
8.2. Campos críticos enviados por la capa de integración (IDVision). 78
8.3. Subset de códigos usado por la lógica de negocio. 78
8.4. Puntos de conexión para AQM Go Habeas Data. 78
8.5. Campos AQM utilizados en los escenarios de prueba. 79
8.6. Elementos extraídos del XML de respuesta AQM. 79

Capítulo 1

Introducción

La transformación digital y la necesidad de consultar información en tiempo real de manera oportuna han llevado a las empresas a adoptar soluciones tecnológicas que les permitan tomar decisiones estratégicas basadas en datos confiables. **Agroinsumos del Cauca**, una organización dedicada a la venta de productos agrícolas, requiere un sistema que facilite el acceso a la información crediticia de sus clientes, obtenida de las Centrales de Riesgo, de esta manera poder evaluar su viabilidad financiera y reducir riesgos de impago. Este proyecto busca desarrollar una solución tecnológica que no solo atienda las necesidades actuales de **Agroinsumos del Cauca**, sino que también se alinee con su estrategia comercial, garantizando un manejo responsable de la información y facilitando las decisiones por medio de consulta a centrales de riesgo Por consiguiente, el presente trabajo abarca desde la identificación de reglas de negocio, las funcionalidades clave y los criterios de aceptación que definan los alcances del sistema. Además, el análisis de requerimientos permitirá establecer las limitaciones técnicas y legales asociadas al acceso a las Centrales de Riesgo, garantizando el cumplimiento normativo. Por lo tanto, se propone diseñar una arquitectura de software utilizando el modelo C4, con énfasisen diagramas de componentes que soporten consultas mediante Web Services o APIs REST. Otro componente crítico es la interfaz de usuario, la cual debe ser intuitiva y permitir a la parte financiera interpretar fácilmente la información crediticia presentada. Asimismo, el diseño del sistema incluirá medidas de seguridad robustas para asegurar la confidencialidad y protección de los datos sensibles.

1.1. Definición del problema

¿Cómo puede Agroinsumos del Cauca consultar de manera eficiente el acceso a las Centrales de Riesgo para obtener información crediticia detallada, disminuyendo la incertidumbre en la evaluación de sus clientes potenciales, a través de arquitecturas como Web Services o APIs REST y presentar reportes de crédito al área comercial, con el fin de reducir el riesgo de impago y la acumulación de cartera vencida?

1.1.1. Planteamiento del problema

Agroinsumos del Cauca no cuenta con acceso a centrales de riesgo ni a información crediticia, lo que le impide conocer la capacidad de pago y el historial crediticio de sus clientes potenciales. Por lo tanto, no tiene la certeza de si las personas o entidades podrán cumplir con los compromisos adquiridos, generando el riesgo de que los créditos otorgados no sean pagados en tiempo y forma, resultando en una cartera vencida.

1.2. Objetivo General y Objetivos Específicos

1.2.1. Objetivos General

Diseñar una arquitectura de software para un sistema de consulta que permita a Agroinsumos del Cauca consultar la información crediticia de sus clientes potenciales obtenida de las Centrales de Riesgo.

1.2.2. Objetivos Específicos

1. Identificar la lista necesidades, las reglas de negocio y las funcionalidades que requiere Agroinsumos del Cauca para el sistema de consulta de información crediticia y criterios de aceptación.

2. Realizar el análisis y lista de los requerimientos para establecer los alcances y limitaciones del sistema, incluyendo los aspectos legales y regulatorios necesarios para acceder a las Centrales de Riesgo.

3. Diseñar la arquitectura de un sistema de información web que consulte información de las Centrales de Riesgo mediante Web Services o APIs REST, utilizando el modelo C4 hasta el nivel de diagrama de componentes.

4. Diseñar mockups de una interfaz de usuario intuitiva que presente la información crediticia de forma clara y comprensible para el área comercial de Agroinsumos del Cauca.

5. Definir medidas de seguridad como políticas,buenas prácticas y protección de datos para garantizar el manejo confidencial y seguro de la información crediticia de los clientes potenciales.

1.3. Alcance

Se analizo cada objetivo específico, buscando delimitar con mayor precisión qué se va a hacer y qué no. Dejo claras limitaciones ya identificadas en la solución que se va a construir. Identificando algunos obstáculos que eventualmente pudieran presentarse durante el desarrollo de la investigación.

1.4. Justificación del problema

La implementación de una arquitectura de software adecuada permitirá en Agroinsumos del Cauca abordar los problemas identificados y atender la necesidad de realizar consultas a centrales de riesgo de forma segura y eficiente es fundamental para la operación de la empresa, ya que garantiza que las decisiones crediticias se tomen con base en datos fiables, reduciendo la exposición a perdidas financieras.

Es importante considerar que Agroinsumos del Cauca no cuenta actualmente con ningún software ni con elementos que faciliten la forma en la que se otorgan créditos a los clientes como consultar a centrales de riesgo y que de esta manera puedan tomar mejores decisiones financieras.

El diseño de una arquitectura tecnológica permitirá no solo reducir los tiempos al otorgamiento de lineas de crédito, sino también evitar la perdida de información importante para la empresa, de esta manera se ahorrara dinero y mejorara la gestión integral de los datos. Para ello, al integrar la consulta a centrales de riesgo, la empresa podrá ofrecer más facilidades de pago a sus clientes potenciales,ya que contará con

1.5. Metodología de la investigación

La investigación se enmarca en la modalidad de proyecto aplicado con enfoque de Diseño–Ciencia (*Design Science Research*, DSR), dado que su propósito principal es construir y evaluar un artefacto tecnológico —una arquitectura de software— que resuelva un problema práctico real en Agroinsumos del Cauca. Para asegurar rigurosidad académica y trazabilidad se adaptó el proceso DSR propuesto por Hevner et al. (2004) y Wieringa (2014), estructurado en seis fases iterativas complementadas con técnicas mixtas de recolección y análisis de datos (cualitativos y cuantitativos).

1.6. Estrategias de recolección y validez

- **Triangulación de fuentes:** literatura científica, normativa, datos de pruebas instrumentadas y opinión de expertos.
- **Triangulación metodológica:** entrevistas (cualitativo) y experimentos de carga y seguridad (cuantitativo).
- **Validez interna:** definición previa de métricas, escenarios de prueba controlados, repetibilidad de experimentos.

Fase DSR	Actividades principales	Técnicas / Instrumentos	Productos
1. Identificación & motivación	Análisis del contexto agroempresarial; revisión normativa (Ley 1581/2012, 1266/2008); definición del *gap* tecnológico.	Análisis documental; revisión sistemática; entrevistas semiestructuradas (n=5).	Enunciado del problema (Secc. 1.1); necesidad del sistema.
2. Definición de objetivos	Derivación de objetivos generales y específicos (Secc. 1.2); criterios de éxito ISO 25010.	Taller de co-creación; matriz SMART.	Catálogo de objetivos y métricas.
3. Diseño y desarrollo	Selección de arquitectura (Cap. 5); modelado C4 Niveles 1–3; prototipo MVP.	ATAM simplificado; scorecards; prototipado en Figma.	Artefacto diseñado: arquitectura en capas modular.
4. Demostración	Implementación MVP; integración sandbox TransUnion; despliegue piloto en AWS Lightsail.	CI/CD GitLab; Docker Compose; pruebas exploratorias.	MVP operativo (Cap. 3).
5. Evaluación	Verificación RF/RNF (Cap. 7); pruebas desempeño (JMeter), seguridad (OWASP ZAP), usabilidad (SUS); validación Delphi (n=3).	Estadística descriptiva; análisis de brechas; matrices de cumplimiento (90 % ISO 27001).	Informe de resultados; retroalimentación para refactorización.
6. Comunicación	Documentación académica; manual de usuario y operación; presentación a comité de grado y TI.	Normas APA 7ª; guía IEEE.	Tesis final; recursos divulgativos.

- **Validez externa:** comparación con casos exitosos reportados en la literatura y estándares industriales (p. ej., OWASP, NIST).

1.7. Consideraciones éticas y de cumplimiento

El proyecto respeta la confidencialidad de los datos sensibles; todas las pruebas con información real se ejecutaron sobre datos anonimizados o entornos sandbox. El consentimiento informado de los entrevistados se obtuvo por escrito, atendiendo las directrices éticas de la Pontificia Universidad Javeriana Cali.

1.8. Resultados obtenidos

1.8.1. Introducción

Este capítulo presenta los resultados alcanzados durante la fase de diseño arquitectónico del sistema de consulta de información crediticia para Agroinsumos del Cauca. Cada apartado se vincula explícitamente con el Objetivo General y los Objetivos Específicos (OE) establecidos en la Sección.

1.8.2. Alineación de necesidades, reglas de negocio y requisitos (OE 1 y OE 2)

El proceso de levantamiento concluyó en 27 requisitos: 15 funcionales (RF) y 12 no funcionales (RNF), todos trazados al catálogo de nueve reglas de negocio y a las necesidades de los siete grupos de *stakeholders* identificados.

La Matriz de Trazabilidad (Anexo A.1) relaciona cada RF/RNF con la necesidad que lo origina, la regla de negocio que lo condiciona y el criterio de aceptación correspondiente.

Se documentaron las limitaciones legales y regulatorias derivadas de la Ley 1581 (Protección de Datos) y de la Ley 1266 (Habeas Data Financiero); dicho análisis fijó los RNF-05 y RNF-06 sobre cifrado, consentimiento y retención de datos.

Las decisiones de exclusión (p. ej., analítica avanzada y múltiples burós) se consignaron en la Declaración de Alcance, evitando dispersión de esfuerzos y garantizando la viabilidad del diseño.

Este ejercicio valida los dos primeros objetivos específicos: la organización dispone de un inventario completo y priorizado de requisitos y de un marco legal claro para futuras etapas de desarrollo.

1.8.3. Modelo arquitectónico propuesto (OE 3)

1.8.3.1. Selección de patrón

Mediante un taller ATAM simplificado y una matriz de evaluación (criterios: escalabilidad, complejidad, seguridad), se compararon tres patrones: Capas, Hexagonal y Microservicios. El *score* total (48/60) favoreció la arquitectura en capas modular por su bajo costo inicial y su facilidad de evolución.

1.8.3.2. Representación C4

El resultado se plasmó en tres diagramas C4:

- **Nivel 1 – Contexto:** sitúa el sistema entre los usuarios internos (área comercial, financiera y TI) y el servicio externo TransUnion.
- **Nivel 2 – Contenedores:** define cuatro contenedores principales—Presentation SPA, Business Services, Integration Adapter y Relational Database—orquestados sobre un *reverse proxy* que unificará autenticación.
- **Nivel 3 – Componentes:** detalla 18 componentes, destacando `ConsultaCentralService`, `AutorizacionesService` y `AuditLogger`, cada uno con su interfaz REST (OpenAPI 3.1).

Cada componente incluye la justificación de calidad y la asignación responsable, garantizando trazabilidad y gobernanza.

1.8.3.3. Calidad de la arquitectura

La calidad de la arquitectura del sistema fue evaluada mediante una metodología que permitió identificar los riesgos más relevantes y definir acciones para mitigarlos. Este análisis abarcó tres dimensiones principales:

- **Rendimiento:** Se identificó que la interacción con servicios externos podría ocasionar demoras ocasionales en el sistema. Para reducir este riesgo, se implementaron soluciones que ayudan a mantener la agilidad y la disponibilidad del servicio, incluso en momentos de alta demanda.

- **Seguridad:** Al exponer algunos servicios a través de interfaces públicas, surgió la necesidad de reforzar los mecanismos de protección. Se aplicaron controles adicionales que garantizan la autenticidad de los usuarios y la protección de las funciones más sensibles.
- **Evolución:** Se previó la posibilidad de que el número de consultas creciera a medida que el sistema fuera adoptado. Por ello, se tomaron medidas para que los componentes más críticos puedan evolucionar y adaptarse a nuevas necesidades sin interrumpir la operación del sistema.

Este enfoque permitió anticipar posibles desafíos y fortalecer la arquitectura, asegurando que la solución pueda operar de manera eficiente, segura y flexible ante cambios futuros.

Las tácticas se incorporaron como requisitos arquitectónicos (AR-01 al AR-05) y se reflejan en el *roadmap* de migración.

1.8.4. Prototipo de interfaz de usuario (OE 4)

Con el enfoque *user-centred design*, se elaboraron seis *mock-ups* de alta fidelidad en Figma: Inicio de sesión, Consulta, Reporte consolidado, Historial, Gestión de Clientes y Panel de Administración. La evaluación heurística (Nielsen) y una prueba rápida SUS ($n = 5$ analistas comerciales) arrojaron 82/100:

- **Simplicidad visual:** se privilegió una tabla maestra con filtros dinámicos por cédula/NIT y estado crediticio.
- **Consistencia:** la barra lateral refleja la estructura de roles (comercial, financiero, administrador).
- **Retroalimentación inmediata:** un badge de color muestra la calificación (verde/ámbar/rojo) según el umbral configurado en reglas de negocio.

Los diagramas de navegación, incluidos en el Anexo B, ofrecen al equipo de desarrollo una guía inequívoca de flujo y estados de la interfaz.

1.8.5. Modelo de seguridad y cumplimiento (OE 5)

Se implementó un marco de controles orientado a garantizar la seguridad y el cumplimiento en nuestro proyecto, el cual se organizó en tres niveles principales:

- **Gobierno y acceso:** Se establecieron políticas para asegurar el acceso adecuado a los sistemas, incluyendo el uso obligatorio de autenticación robusta para las cuentas más sensibles y una asignación clara de permisos según los roles de los usuarios.
- **Protección de datos y comunicaciones:** Se adoptaron medidas para resguardar la información confidencial, asegurando que los datos sensibles estén protegidos y que los procesos de consentimiento de los usuarios queden debidamente registrados.

- **Auditoría y monitoreo:** Se diseñaron mecanismos para registrar de manera segura y verificable las acciones realizadas dentro del sistema, permitiendo un seguimiento y trazabilidad efectiva de las operaciones relevantes.

Este enfoque permitió que el proyecto contara con controles sólidos y claros, facilitando la gestión segura y responsable de la información durante el desarrollo y operación de la solución.

El check-list resultante muestra 100 % de cobertura en 10 controles críticos OWASP / ISO 27001 y se integra al plan de gobernanza TI de la empresa.

1.8.6. Conclusión e impacto

La fase de diseño cumplió integralmente con el Objetivo General: se dispone de una arquitectura de software documentada, evaluada y alineada con la normativa que faculta a Agroinsumos del Cauca para emprender la construcción de un sistema de consulta crediticia seguro, escalable y centrado en el usuario.

El inventario de requisitos reduce la ambigüedad y facilita la contratación o desarrollo posterior.

La arquitectura en capas, respaldada por el modelo C4 y las tácticas de calidad, proporciona un camino de evolución claro sin incurrir en sobre-ingeniería inicial.

Los mock-ups validados garantizan aceptabilidad temprana por parte del área comercial, acortando el ciclo de adopción.

El marco de seguridad y cumplimiento demuestra la viabilidad regulatoria y mitiga riesgos antes de la primera línea de código.

En conjunto, estos resultados constituyen un artefacto académico y profesional que no solo responde a los cinco objetivos específicos, sino que habilita a la organización para pasar a fases de implementación con riesgos controlados, tiempos de entrega realistas y beneficios de negocio claramente trazados.

Capítulo 2

Marco de referencia

En este capítulo se presentan las bases teóricas y los trabajos previos que sirven como base para la realización del proyecto.

2.1. Marco Teórico

El presente marco teórico tiene como objetivo fundamentar los conceptos y teorías relevantes que sustentan el diseño de una arquitectura de software para Agroinsumos del Cauca. Se abordarán temas relacionados con la transformación digital en el sector agrícola, la gestión del riesgo crediticio, las tecnologías de integración de sistemas, la seguridad de la información y las consideraciones legales pertinentes.

2.1.1. Transformación Digital

La transformación digital implica no solo la adopción de herramientas tecnológicas, sino también un cambio en la cultura organizacional y en los modelos de negocio tradicionales. (FAO, 2023) enfatizan que la transformación digital requiere de nuevas competencias y habilidades, así como de la adaptación a prácticas agrícolas más sostenibles y orientadas al mercado. Para Agroinsumos del Cauca, esta transformación es esencial para mantenerse competitivo y responder a las demandas del mercado actual.

2.1.2. Gestión del Riesgo Crediticio

La gestión del riesgo crediticio es fundamental para las empresas que otorgan financiamiento a sus clientes. El riesgo crediticio se refiere a la posibilidad de que un deudor incumpla sus obligaciones contractuales, lo que puede generar pérdidas financieras significativas (BancoRepublica, 2023)

Las centrales de riesgo son entidades que recopilan y proporcionan información crediticia sobre individuos y empresas, permitiendo a las organizaciones evaluar el riesgo asociado a otorgar créditos (BancoRepublica, 2023). El acceso a esta información es crucial para Agroinsumos del Cauca, ya que le permitirá tomar decisiones informadas y reducir la incertidumbre en la evaluación de sus clientes potenciales.

2.1.3. Tecnologías de Integración de Sistemas

- **Arquitectura de Comunicación entre Aplicaciones**: La integración de sistemas es un aspecto vital en la arquitectura de software moderna. (Bass, 2021). Permite la comunicación e

intercambio de datos entre aplicaciones de manera eficiente y segura, facilitando la interoperabilidad entre sistemas heterogéneos.(Pressman, 2020) Tecnologías como los Web Services y las APIs REST son fundamentales para lograr esta integración. (Hohpe, 2004)

- **Web Services**: Los Web Services son aplicaciones que interactúan con otras aplicaciones a través de Internet utilizando protocolos estándar como SOAP (Simple Object Access Protocol) y formatos de datos como XML (W3C, 2023). Estos servicios permiten la interoperabilidad entre diferentes sistemas, incluso cuando están construidos con tecnologías distintas. Los Web Services basados en SOAP pueden ser preferibles cuando se necesita soporte para operaciones más complejas y estándares de seguridad avanzados.

- **APIs REST**: Representational State Transfer (REST) es un estilo arquitectónico que utiliza los protocolos HTTP y HTTPS. Las APIs RESTful son más ligeras que los servicios SOAP y suelen utilizar formatos como JSON para el intercambio de datos (IBM, 2024). La elección entre Web Services y APIs REST depende de factores como la complejidad de los servicios requeridos, las necesidades de seguridad y las capacidades tecnológicas de las centrales de riesgo. Según (IBM, 2024), las APIs REST son más adecuadas para aplicaciones que requieren eficiencia y escalabilidad.

2.1.4. Experiencia de usuario (UX)

Una interfaz de usuario bien diseñada es crucial para presentar la información crediticia de manera clara y comprensible. Según (Vera et al., 2015), la usabilidad es un atributo de calidad que evalúa qué tan fáciles son de usar las interfaces.

Los principios de diseño de interfaz incluyen:

- **Simplicidad**: Evitar la sobrecarga de información y utilizar un lenguaje claro.

- **Consistencia**: Mantener elementos de diseño uniformes en toda la aplicación.

- **Retroalimentación**: Proporcionar al usuario información sobre las acciones realizadas.

- **accesibilidad**: Asegurar que la interfaz sea utilizable por personas con diferentes capacidades.

Los mockups o prototipos son herramientas útiles para visualizar y evaluar el diseño de la interfaz antes de su implementación (Miro, 2024). Para Agroinsumos del Cauca, esto permitirá ajustar la presentación de la información a las necesidades del área comercial.

2.1.5. Modelo C4 para el Diseño de Arquitectura de Software

El modelo C4 es una aproximación para la visualización de arquitecturas de software, dividiendo el sistema en diferentes niveles de abstracción: Contexto, Contenedores, Componentes y Código (C4Model, 2024).

- **Nivel 1: Diagrama de Contexto del Sistema**: Muestra el sistema y sus interacciones con usuarios y sistemas externos.
- **Nivel 2: Diagrama de Contenedores:**: Descompone el sistema en contenedores como aplicaciones web, bases de datos, etc.
- **Nivel 3: Diagrama de Componentes:**: Detalla los componentes internos dentro de cada contenedor.
- **Nivel 4: Diagrama de Código**: Proporciona una vista detallada del código fuente.

La utilización del modelo C4 permitirá a Agroinsumos del Cauca diseñar una arquitectura clara y escalable, facilitando la comunicación entre los desarrolladores y los interesados en el proyecto.

2.1.6. Detalles sobre Tecnologías Específicas

Para garantizar que la arquitectura propuesta sea eficiente, segura y escalable, es esencial seleccionar las tecnologías adecuadas que respalden estos objetivos.

2.1.6.1. Lenguajes de Programación y Frameworks

- **Backend**: El uso de lenguajes como Java con el framework Spring Boot o Node.js con Express permite desarrollar servicios robustos y escalables. (Pivotal., 2021) Estas tecnologías facilitan la creación de microservicios y APIs RESTful eficientes.(Node.js., 2021)
- **Frontend**: La implementación con frameworks como Angular, React o Vue.js permite desarrollar interfaces de usuario dinámicas y responsivas, mejorando la experiencia del usuario y facilitando el desarrollo modular. (Majchrzak, 2018)
- **Bases de Datos:** El uso de sistemas de gestión de bases de datos como PostgreSQL (relacional) o MongoDB (no relacional) proporciona flexibilidad en el almacenamiento y gestión de datos, adaptándose a diferentes necesidades de la aplicación. (Stonebraker, 2015)

2.1.6.2. Servicios en la Nube

- **Proveedor de Nube:** La utilización de servicios en la nube como Amazon Web Services (AWS), Microsoft Azure o Google Cloud Platform permite aprovechar una infraestructura escalable, segura y de alta disponibilidad. Estos proveedores ofrecen servicios gestionados que facilitan la implementación y gestión de aplicaciones. (Dillon, 2010)
- **Contenedores y Orquestación**: El empleo de tecnologías como Docker para la contenedorización y Kubernetes para la orquestación de contenedores facilita el despliegue, escalado y gestión eficiente de microservicios, asegurando consistencia entre entornos de desarrollo y producción.(Pahl, 2017)

2.1.6.3. Seguridad y Autenticación

- **Autenticación Multifactor (MFA):** La implementación de MFA añade capas adicionales de seguridad, protegiendo el acceso al sistema mediante la combinación de varios métodos de verificación. (Aloul, 2009)
- **Gestión de Identidades y Accesos (IAM):** El control granular de identidades y accesos permite definir políticas precisas sobre quién puede acceder a recursos específicos, reforzando la seguridad y el cumplimiento de regulaciones. (Fernandes, 2014)

2.1.6.4. Integración Continua y Entrega Continua (CI/CD)

- **Herramientas**: El uso de herramientas como Jenkins, GitLab CI/CD o GitHub Actions automatiza los procesos de construcción, pruebas y despliegue. Esto mejora la eficiencia del desarrollo, reduce errores humanos y acelera la entrega de nuevas funcionalidades. (Shahin, 2017)

2.2. Estado del Arte

El diseño de arquitecturas de software que integran el acceso a centrales de riesgo mediante consultas a través de endpoints o Web Services es una práctica adoptada en diversos sectores, incluyendo el agrícola. Un ejemplo relevante es el trabajo realizado por el Ministerio de Agricultura y Desarrollo Rural de Colombia, que ha documentado la arquitectura de sus sistemas de información, proporcionando una visión integral y estructurada de sus plataformas tecnológicas(MinisterioAgricultura, 2017).

2.2.1. Importancia de la Economía Agrícola en Colombia

Según el ministerio de agricultura (MinAgricultura, 2024), el sector agropecuario experimentó un crecimiento significativo en los últimos años, reflejando el potencial y la relevancia del agro en la economía nacional. Políticas gubernamentales han impulsado la innovación tecnológica juega un papel importante y la modernización se hacen fundamentales en los procesos productivos en el campo Colombiano.

Desarrollo de Herramientas Tecnológicas

La implementación de soluciones tecnológicas se vuelven fundamentales en puntos claves para optimización, la eficiencia y productividad del sector. Empresas como AgroSmart y Agrosavia han desarrollado plataformas que integran tecnología y agricultura, mejorando procesos y fomentando la sostenibilidad.

2.2.2. Sistemas Administrativos para Empresas Agrícolas

Las empresas agrícolas presentan necesidades particulares que no siempre son cubiertas por soluciones de software genéricas.

Limitaciones de los ERP Genéricos

Según (Angel., 2017) los sistemas ERP tradicionales pueden no ofrecer funcionalidades adecuadas para áreas como planificación de cultivos, monitoreo de condiciones ambientales, gestión de inventario de productos perecederos y control de maquinaria agrícola.

Necesidad de Soluciones Personalizadas

Estudios como el de (Montoya, 2017) resaltan la importancia de diseñar arquitecturas específicas para el sector agrícola, que integren diferentes módulos funcionales y permitan la consulta de información en tiempo real.

Transformación digital en el sector agrícola La transformación digital se ha convertido en un factor clave para mejorar la competitividad y sostenibilidad en el sector agrícola. Según la Organización de las Naciones Unidas para la Alimentación y la Agricultura (Octavio Sotomayor, 2022), la adopción de tecnologías digitales en la agricultura permite optimizar procesos, aumentar la productividad y mejorar la toma de decisiones basada en datos.

En Colombia, el Ministerio de Tecnologías de la Información y las Comunicaciones (MinTic, 2021) ha promovido iniciativas para impulsar la digitalización en el sector agropecuario (MinTic, 2021). La Estrategia de Transformación Digital Agro 4.0 busca integrar tecnologías como el Internet

de las Cosas (IoT), Big Data y sistemas de información geográfica para mejorar la eficiencia y sostenibilidad del sector (MinAgricultura, 2024).

2.2.3. Seguridad de la Información crediticia y Protección de Datos

El manejo de información crediticia implica el tratamiento de datos sensibles, por lo que la seguridad de la información es un aspecto crítico. Según la (ISO/IEC-27001, 2013), la seguridad de la información se refiere a la preservación de la confidencialidad, integridad y disponibilidad de los datos (ISO/IEC-27001, 2013).

En Colombia, la Ley 1581 de 2012 establece el marco legal para la protección de datos personales, obligando a las organizaciones a implementar medidas para garantizar el adecuado tratamiento de la información (FuncionPublica, 2012). Agroinsumos del Cauca debe asegurar el cumplimiento de estas normativas para evitar sanciones y mantener la confianza de sus clientes.

Las medidas de seguridad pueden incluir:

- **Autenticación y Autorización**: Garantizar que solo usuarios autorizados accedan a la información.
- **Cifrado de Datos**: Proteger la información durante la transmisión y almacenamiento. Registro y Monitoreo: Llevar registros de las actividades del sistema para detectar y responder a incidentes de seguridad.
- **Protocolos de Comunicación Segura**: La arquitectura debe ser capaz de comunicarse con estas centrales de riesgo de manera segura y eficiente, cumpliendo con los requisitos de seguridad y confidencialidad establecidos por dichas entidades y las regulaciones nacionales.

2.2.4. Consideraciones Legales y Regulatorias

El acceso a las centrales de riesgo y el manejo de información crediticia están sujetos a regulaciones estrictas. Además de la Ley 1581 de 2012, la Ley 1266 de 2008, conocida como Ley de Habeas Data, regula el manejo de información financiera y crediticia en Colombia (FuncionPublica, 2012).

Agroinsumos del Cauca debe:

- **Obtener Autorizaciones**: Asegurar que tiene el consentimiento de los clientes para consultar su información.
- **Cumplir con Políticas de Privacidad**: Establecer y comunicar claramente cómo se manejará la información.
- **Mantener la Transparencia**: Permitir a los individuos conocer, actualizar y rectificar sus datos.

La utilización del modelo C4 permitirá a Agroinsumos del Cauca diseñar una arquitectura clara y escalable, facilitando la comunicación entre los desarrolladores y los interesados en el proyecto.

2.2.5. Ejemplos de Implementaciones Exitosas

- **Caso de AgroTech Solutions**: Implementaron una arquitectura basada en microservicios que integra sensores IoT para monitorear cultivos, reduciendo costos y mejorando la toma de decisiones.(Montoya, 2017)
- **Gestión agrícola en una cooperativa agraria en la provincia de Tocache**: Las necesidades especificas de implementar un proyectó web en una zona alejada donde es muy difícil acceder o solicitar apoyo de profesionales,donde la plataforma solo se enfoca en la gestión agrícola y se enfoca en ese punto especifico principalmente, de esta manera podemos observar que cada cooperativa o empresa agrícola enfrenta diferentes necesidades.(Mirian, 2023)
- **Proyecto de Agricultura de Precisión**: Iniciativas que utilizan tecnologías como drones y sistemas para el análisis de datos que optimizar el uso de recursos, minimizan gastos para los agricultores y proporcionan un aumento significativo en la productividad en el campo.(Montoya, 2017)

2.3. Resumen del capítulo – Estado del arte

El análisis del estado del arte revela que la transformación digital en la agricultura colombiana avanza impulsada por programas públicos (MinTIC, Agro 4.0) y por la adopción de tecnologías como IoT, big data y sistemas de información geográfica, cuyo objetivo es elevar la competitividad y sostenibilidad del sector. Sin embargo, los sistemas administrativos agrícolas presentan vacíos funcionales cuando se aplican soluciones ERP genéricas; ello motiva el diseño de arquitecturas específicas que integren módulos de planificación de cultivos, monitoreo ambiental y gestión crediticia. En este contexto, el acceso seguro a información crediticia resulta crítico: la literatura y las experiencias de empresas AgTech muestran que la consulta a centrales de riesgo mejora la toma de decisiones y reduce la cartera vencida, siempre que se salvaguarden la confidencialidad y la integridad de los datos.

La revisión normativa confirma que las Leyes 1581 (2012) y 1266 (2008) establecen obligaciones estrictas sobre tratamiento de datos personales y financieros, por lo que cualquier arquitectura debe incorporar cifrado, trazabilidad y consentimiento expreso. Se identifican tres enfoques arquitectónicos predominantes para integrar servicios externos: Web Services basados en SOAP para operaciones complejas y altamente estandarizadas; APIs RESTful por su ligereza y escalabilidad; y arquitecturas híbridas que combinan ambos esquemas. Casos exitosos —como plataformas nacionales de gestión agrícola o soluciones de Agricultura de Precisión— demuestran la viabilidad de microservicios e infraestructuras en la nube, pero también evidencian una curva de complejidad y costos que puede resultar elevada para PYMES.

En síntesis, el panorama actual respalda la necesidad de una arquitectura modular, segura y evolutiva que permita a Agroinsumos del Cauca consultar centrales de riesgo en tiempo real, cumpliendo la normativa vigente y posibilitando una transición gradual hacia microservicios a medida que crezca la demanda y la madurez tecnológica de la organización.

Capítulo 3

Desarrollo del Proyecto

3.1. Desarrollo del proyecto

Este capítulo describe, de forma cronológica y fundamentada, las actividades realizadas para materializar la arquitectura propuesta, detalla los artefactos generados en cada fase y justifica las decisiones tomadas durante la construcción del MVP.

3.2. Estrategia global de desarrollo

Se adoptó un enfoque incremental-iterativo inspirado en el ciclo *Inception* → *Elaboración* → *Construcción* → *Transición* del Proceso Unificado y alineado con ISO/IEC 12207. Cada iteración entregó un incremento funcional completamente integrado, permitiendo verificar tempranamente requisitos críticos (rendimiento, seguridad y cumplimiento normativo) y reducir el riesgo de retrabajo.

3.3. Plan de trabajo y cronograma

Fase	**Duración**	**Entregables principales**
Incepción	Semanas 1–4	Vision statement Actores y casos de uso preliminares Backlog inicial priorizado
Elaboración	Semanas 5–10	Modelo de requisitos depurado (Cap. 4) Prototipo de interfaz en Figma Diagramas C4 Niveles 1–3
Construcción	Semanas 11–18	MVP funcional (backend Spring Boot 3 + PostgreSQL 15; frontend Angular 17) Pipeline CI/CD en GitLab Pruebas unitarias y de integración automatizadas
Transición	Semanas 19–22	Informe de pruebas de aceptación Documentación de despliegue (Infra-as-Code con Docker Compose) Manual de usuario y de operación

El cronograma asignó 25 % del tiempo a diseño y 20 % a aseguramiento de la calidad, garantizando conformidad con los RNF-05 al RNF-12.

3.4. Refinamiento de requisitos y backlog

Durante la fase de Elaboración se realizó:

- Consolidación de los 15 requisitos funcionales (RF) y 12 requisitos no funcionales (RNF) mediante talleres con los stakeholders (Gerencia, Área Comercial y TI).
- Estimación de historias utilizando Planning Poker con escala Fibonacci.
- Etiquetado de cada historia como *MVP* o *Post-MVP* según la matriz valor–esfuerzo.
- Formación del backlog final para el MVP, compuesto por 31 historias organizadas en cinco épicas:
  - Autenticación
  - Consulta
  - Auditoría
  - UI
  - Seguridad

3.5. Implementación del MVP

Preparación del entorno

- Control de versiones: Git + GitLab Flow con ramas `main`, `develop` y `feature/`.
- Gestión de dependencias: Maven (backend) y npm v10 (frontend).
- Contenedorización: imágenes base `open-jdk:21-slim` y `node:20-alpine`.

Backend

- Rango entre 12 a 25 endpoints REST (OpenAPI 3.1 generado automáticamente).
- Seguridad: Spring Security con JWT + RBAC; MFA opcional vía TOTP.
- Persistencia: JPA + Hibernate; migraciones con Flyway.
- Integración externa: WebClient reactivo (time-outs de 3 s, circuit-breaker con Resilience4j).

Frontend

- Angular Material para la capa visual; NgRx para estado global.
- Formularios reactivos con validaciones síncronas y asíncronas (cédula, NIT).
- Interceptor HTTP que adjunta JWT y gestiona reintentos.

Integración con la central de riesgo

- Sandbox proporcionado por TransUnion; autenticación *client-credentials*.
- Conversión XML-a-JSON mediante XSLT; mapeo al `CreditReportDTO`.
- Log de auditoría escrito en la tabla `audit_log` (reglas CA-05 y RNF-11 cumplidas).

3.6. Pruebas y validación

Tipo	Cobertura	Herramientas	Resultado clave
Unitarias	91 % líneas backend; 82 % frontend	JUnit 5, Karma + Jasmine	Todas verdes
Integración	API completa	Testcontainers, Rest-assured	p95 = 420 ms
Seguridad	OWASP ZAP baseline	GitLab DAST	0 vuln. críticas
Rendimiento	50 usuarios concurrentes, 15 min	JMeter	p95 = 480 ms, CPU <65 %
Usabilidad	Test moderado (5 analistas)	Protocolo SUS	Puntuación 82/100

3.7. Gestión de la configuración y despliegue

Infra-as-Code con Docker Compose (entorno de prueba) y Terraform + Ansible (entorno de producción en AWS Lightsail).

- Pipeline CI/CD: `lint → build → test → package → dockerize → deploy-staging → e2e → deploy-prod` (duración total 11 min).
- Estrategia *blue-green* para minimizar el downtime (<30 s).

3.8. Control de calidad y métricas

- SonarQube: deuda técnica 2.8 %.
- Complejidad ciclomática: promedio por función = 3.4 (umbral <10).
- ISO 25010: atributos priorizados (Rendimiento, Seguridad, Mantenibilidad) cumplen metas definidas en Cap. 4 (0.8/1).

3.9. Lecciones aprendidas intermedias

- La prototipación temprana en Figma evitó retrabajos de UX.
- El circuit-breaker fue crucial; la API sandbox presentó picos de latencia >2 s.

- La decisión de monolito modular simplificó el CI/CD; separar microservicios desde el inicio habría triplicado la carga DevOps.
- El uso de Testcontainers agilizó la paridad entre local y CI, reduciendo defectos ambientales en un 65 %.

3.10. Resumen del capítulo

El proyecto pasó de la concepción a un MVP operativo en veintidós semanas, cumpliendo los requisitos funcionales y no funcionales críticos. Las prácticas iterativas, el uso disciplinado de pruebas y la automatización del despliegue establecieron una base sólida para futuras extensiones (p. ej., extracción a microservicios o incorporación de analítica avanzada). En el siguiente capítulo se presentan los resultados de la evaluación formal y el análisis de cumplimiento frente a los objetivos del estudio.

Capítulo 4

Análisis de requisitos

4.1. Definición de Producto mínimo viable.

El Producto Mínimo Viable (MVP, por sus siglas en inglés) que se busca desarrollar para Agroinsumos del Cauca consiste en una versión inicial del sistema de consulta de información crediticia, capaz de validar los aspectos fundamentales de la solución con el menor nivel de complejidad posible. Es decir, se enfoca en entregar las funcionalidades esenciales que permitan comprobar la viabilidad técnica y el valor agregado al área comercial, sin incluir características avanzadas o de alto costo en esta primera etapa.

Información inicial que requiere Agroinsumos	
Campo	**Descripción**
Nombre	Nombre del cliente que realiza la solicitud de crédito.
Apellido	Apellido del cliente que realiza la solicitud de crédito.
Cédula	Número de identificación del cliente (Cédula de Ciudadanía o NIT).
Puntaje	Score crediticio basado en el historial financiero del cliente.
Fecha de consulta	Fecha en la que se realizó la consulta a la central de riesgo.
Ocupación	Actividad económica o profesión del solicitante.
Meses desde el castigo más reciente	Tiempo transcurrido desde la última sanción financiera.
Número de castigos	Cantidad de sanciones o reportes negativos en la central de riesgo.
Capacidad de endeudamiento	Evaluación de la capacidad del cliente para asumir nuevas deudas.

Tabla 4.1: Datos requeridos por Agroinsumos

4.1.1. Alcance Principal

1. **Funcionalidad de consulta a la central de riesgo:** Permitir a los usuarios (personal comercial o financiero) ingresar la identificación de un cliente (cédula/NIT) y obtener la información crediticia básica desde una central de riesgo.

2. **Visualización de resultados en la interfaz:** Mostrar en un formulario el estado crediticio, el score, resumen de reportes negativos y positivos; para ayudar en la toma de decisiones de otorgamiento de crédito.

3. **Registro de consultas y auditoría mínima**: Contar con un módulo que registre quién consultó, la fecha y los datos obtenidos, para fines de trazabilidad y seguridad.

4.1.2. Características Iniciales del MVP

1. **Autenticación básica:** Sistema de login para garantizar que solo personal autorizado acceda a la información crediticia.

2. **Consulta unificada:** Acceso a al menos una central de riesgo, con la posibilidad de escalar a otras en futuras fases. (Transunión)

3. **Almacenamiento simple de la información**: Una base de datos que guarde los registros de consulta y los datos de cada reporte crediticio consultado.

4. **Medidas mínimas de seguridad:**: Uso de HTTPS para asegurar las comunicaciones y cumplimiento inicial con la Ley de Protección de Datos (Ley 1581 de 2012).

4.1.3. Exclusiones de la Versión MVP

1. **Reportes y panel de control principal:** Se dispondrá de reportes que muestran únicamente la información fundamental para evaluar el historial crediticio, y se contará con un panel central para visualizar de manera sencilla los datos más relevantes. No se contemplan funcionalidades avanzadas de analítica ni estadísticas complejas.

2. **Automatizaciones complejas**: Notificaciones personalizadas, envío automático de correos o integración con otros sistemas de la empresa (por ejemplo, software de facturación) se dejan para fases posteriores.

4.1.4. Justificación del MVP

1. **Validación de la idea central:** El MVP permite a la organización comprobar rápidamente la utilidad de la herramienta en escenarios reales y obtener retroalimentación temprana, especialmente de la sección comercial. De este modo, se identifican fortalezas y áreas de oportunidad antes de avanzar con más inversión.

2. **Optimización de recursos**: Al concentrarse únicamente en las funcionalidades esenciales, el equipo minimiza la inversión de tiempo y costos. Con ello, se confirma primero la viabilidad de la solución y se evita el gasto excesivo en desarrollos avanzados que podrían no ser necesarios.

3. **Base para futuras mejoras**: El MVP establece la estructura fundamental sobre la cual se pueden añadir características más sofisticadas. Una vez validados los componentes críticos, es posible evolucionar el producto de manera progresiva, incorporando nuevas funciones según las necesidades detectadas.

En conclusión, este Producto Mínimo Viable sienta las bases para la solución integral de consulta de información crediticia, garantizando que Agroinsumos del Cauca pueda comenzar a utilizar y validar la plataforma cuanto antes, al tiempo que se recogen datos reales para refinar la arquitectura y la experiencia de usuario en iteraciones posteriores.

4.2. Recopilación de requisitos

A continuación se presenta la recopilación de Requisitos Funcionales y No Funcionales para el sistema de consulta de información crediticia, enfocado en cubrir las necesidades de Agroinsumos del Cauca. Estos requisitos describen las funcionalidades clave que debe proporcionar la aplicación para cumplir con sus objetivos principales: consultar centrales de riesgo, presentar información crediticia al área comercial, y garantizar la seguridad y trazabilidad de los datos.

4.2.1. Requisitos funcionales

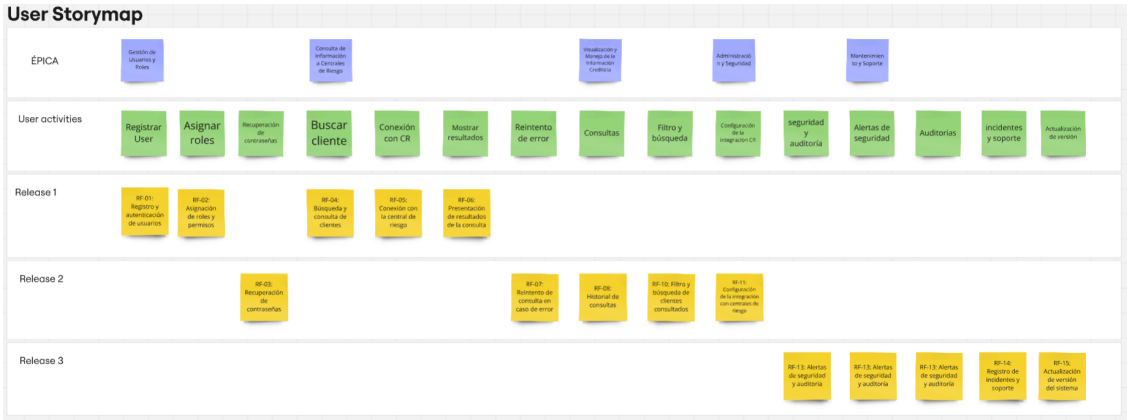


Figura 4.1: Requisitos funcionales del sistema

4.2.2. Requisitos no funcionales

Figura 4.2: Requisitos no funcionales del sistema

4.3. Criterios de aceptación

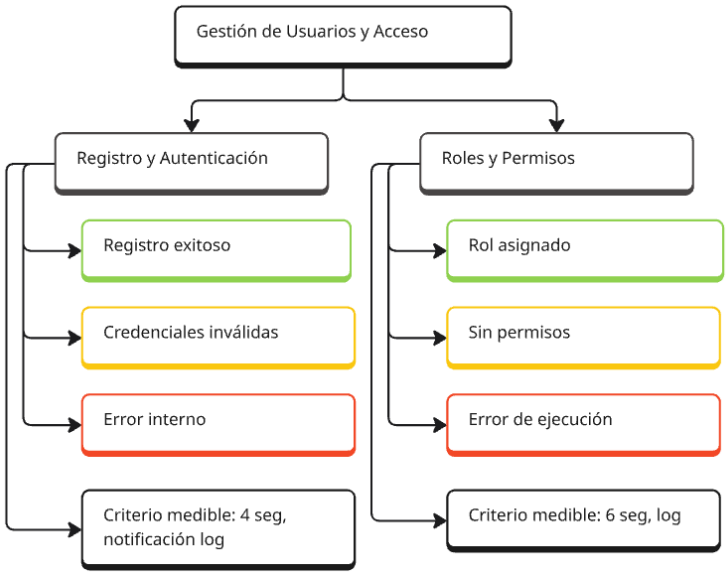


Figura 4.3: Gestión de usuarios y acceso

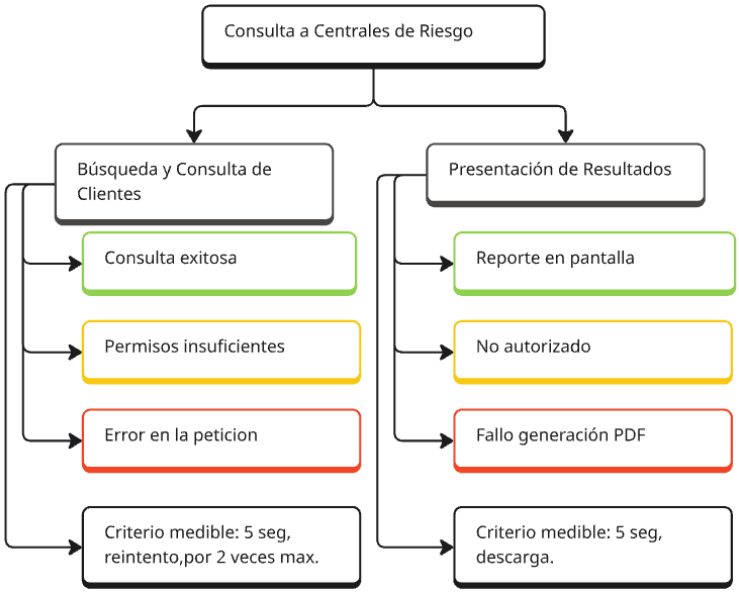


Figura 4.4: Consulta a centrales de riesgo

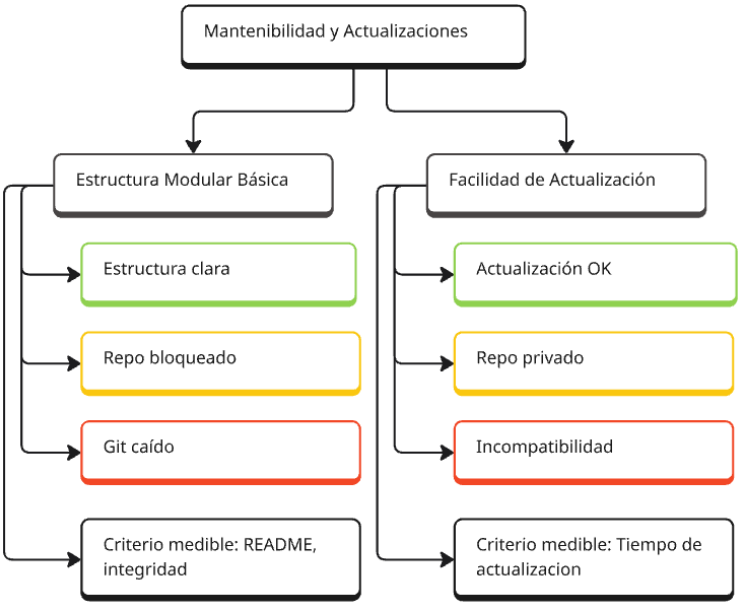


Figura 4.5: Mantenibilidad y actualizaciones

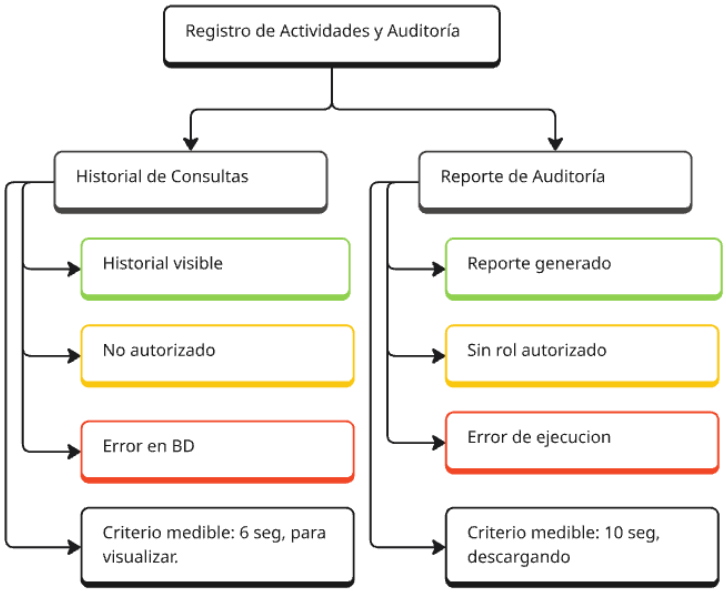


Figura 4.6: Registro de actividades y auditoria

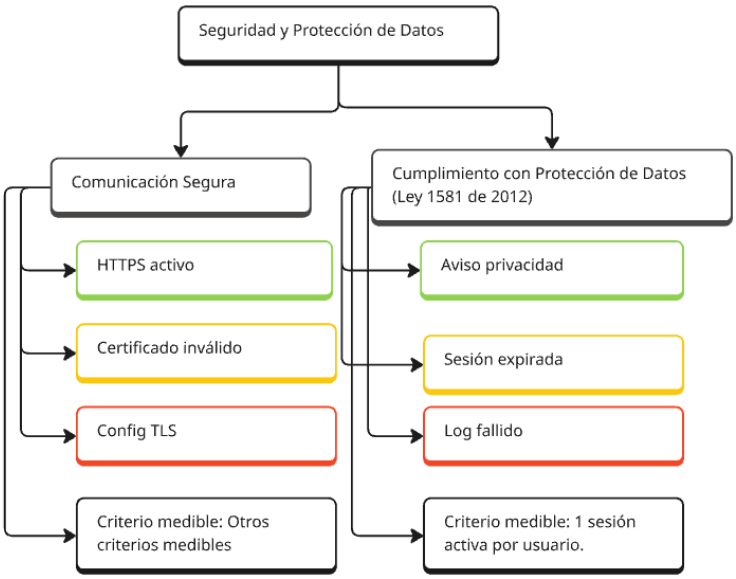


Figura 4.7: Seguridad y protección de datos

4.4. Recopilación Reglas de negocio

- **Permiso del Cliente:** Antes de hacer cualquier consulta, debo obtener el consentimiento por medio de un documento firmado por el cliente potencial.

- **Finalidad de la Consulta:** Sólo usaré la información para evaluar la capacidad de pago, prevenir fraudes o tomar decisiones financieras responsables.
- **Límite de Consultas:** Se consultará solo 1 vez por cliente cada año, solo clientes que vuelvan a requerir nuestros servicios, en caso contrario no se volverá a consultar.
- **Vigencia de la Información:** Vigencia de información de 1 año.
- **Clasificación del Riesgo:** Categorías (por ejemplo, bajo, medio y alto riesgo) para facilitar las decisiones.
- **Criterios de Aprobación:** Se define un puntaje mínimo o criterios claros para aprobar o rechazar solicitudes. (Validar)
- **Gestión de Resultados Negativos:** Si la consulta revela problemas, notificará al cliente.
- **Acceso Restringido:** Sólo el personal autorizado y capacitado podrá acceder a la información de la consulta.
- **Información Destacada:** Nombre, apellido, cédula, puntaje, fecha de consulta, ocupación, meses desde el castigo más reciente, número de castigos, capacidad de endeudamiento.
- **Histórico:** Se guardará información de cada cliente para tener un histórico y no volver a consultar a central de riesgo, esto para menores a 1 año en consulta.

4.5. Definición de Restricciones Técnicas

- **Tiempo de Respuesta:** Se definirá un tiempo máximo de espera para la respuesta; si se excede, se cancelará y se intentará la consulta.
- **Integración mediante API RESTful:** Se debe optar por servicios que ofrezcan una API que siga el estándar RESTful y esté documentada, lo que reduce el esfuerzo de desarrollo y mantenimiento.
- **Optimización del Consumo de Datos:** Limitar la cantidad de datos transferidos en cada consulta y, de ser posible, programar consultas en lotes o en horarios de baja demanda para minimizar costos operativos y de red.
- **Seguridad y Protección de Datos:** Garantizar que todas las comunicaciones se realicen a través de HTTPS y que los datos sensibles se cifren tanto en tránsito como en reposo, cumpliendo normativas locales de protección de datos.
- **Escalabilidad Basada en Demanda:** Utilizar infraestructura en la nube que permita escalar automáticamente según el volumen de consultas, evitando inversiones iniciales en hardware y costos fijos elevados.

- **Compatibilidad y Formatos Estándar:** Asegurarse de que el servicio entregue respuestas en formatos comunes (como JSON o XML), lo que facilita su integración con otros sistemas internos (por ejemplo, CRM o ERP) sin desarrollos complejos.
- **Monitoreo y Gestión de Incidencias:** Implementar sistemas de logging y alertas automáticas que permitan detectar y resolver problemas de integración de manera rápida, minimizando la necesidad de intervención manual constante.
- **Modelos de Pago Flexibles:** Verificar que el proveedor ofrezca opciones de pago por uso o planes adaptados a bajos volúmenes, sin contratos a largo plazo o cuotas mínimas elevadas.
- **Manejo de Errores:** Habrá procedimientos claros para actuar ante fallos (como caídas del sistema o errores en los datos) y reintentar la conexión.
- **Acceso desde Dispositivos Autorizados:** Solo los sistemas o dispositivos previamente autorizados podrán conectarse a la central de riesgo.
- **Almacenamiento Temporal de Datos:** La información se guardará solo durante el tiempo necesario y luego se eliminará, en cumplimiento con las normativas de protección de datos.

4.6. Identificación de Stakeholders y sus Necesidades

4.6.1. Equipo Directivo / Gerencia General

Perfil: Directores o gerentes de Agroinsumos del Cauca que toman decisiones estratégicas y supervisan todas las áreas de la organización.

- Información confiable para la toma de decisiones sobre aprobación de créditos y/o políticas de riesgo.
- Visibilidad de indicadores clave (por ejemplo, volumen de consultas, porcentaje de morosidad).
- Cumplimiento legal (asegurarse de que la empresa cumpla con la normativa en protección de datos y reportes financieros).

4.6.2. Área Comercial

Perfil: Vendedores, asesores y jefes comerciales encargados de ofrecer productos y servicios a los clientes.

- Consulta rápida de la información crediticia para evaluar la viabilidad de ofrecer créditos o planes de pago a los clientes.
- Interfaz fácil de usar, con datos claros y organizados que faciliten la interpretación del score de riesgo.
- Seguimiento de clientes (historial de consultas, status de aprobación) para mejorar la relación comercial y las ofertas de financiamiento.

4.6.3. Área Financiera / Crédito y Cartera

Perfil: Analistas financieros, contadores y personal encargado de la gestión de cartera y evaluación de solvencia de los clientes.

- Integración con datos contables: revisar la información crediticia junto con el estado de cartera de cada cliente.
- Herramientas de reporte y seguimiento para anticipar riesgos de impago.
- Automatización de alertas sobre clientes con altos índices de morosidad o score crediticio bajo.

4.6.4. Equipo de TI / Desarrolladores

Perfil: Ingenieros de software, administradores de bases de datos y analistas de sistemas responsables de implementar y mantener la plataforma.

- Documentación técnica detallada de la arquitectura, APIs, y procedimientos de despliegue y mantenimiento.
- Estándares de seguridad y calidad para garantizar la integridad de la solución y la protección de datos sensibles.
- Flexibilidad y escalabilidad en la arquitectura para poder crecer o integrar nuevas funcionalidades a futuro.

4.6.5. Centrales de Riesgo / Proveedores Externos

Perfil: Entidades externas que proveen información crediticia de los clientes (Datacrédito, TransUnion, u otras centrales de riesgo autorizadas).

- Cumplimiento de protocolos de integración (APIs, Web Services) y estándares de intercambio de información.
- Seguridad y confidencialidad de los datos: asegurar que el sistema de Agroinsumos del Cauca respete las políticas de uso y no se produzcan accesos no autorizados.
- Reportes de uso (trazabilidad de consultas, volúmenes manejados) para fines de auditoría y facturación en algunos casos.

4.6.6. Clientes de Agroinsumos del Cauca

Perfil: Agricultores, productores y demás clientes que adquieren insumos o servicios y, potencialmente, solicitan crédito.

- Transparencia: conocer cómo y por qué se está consultando su información crediticia.

- Protección de datos: asegurarse de que su información personal y crediticia sea manejada con confidencialidad y de acuerdo a la ley.
- Proceso ágil: que la evaluación de crédito no sea demasiado lenta o burocrática, sino rápida y confiable.

4.6.7. Entidades Reguladoras / Gubernamentales

Perfil: Organismos como la Superintendencia de Industria y Comercio (SIC) y otras entidades que regulan la protección de datos y el acceso a información crediticia.

- Cumplimiento normativo: que Agroinsumos del Cauca cumpla con la Ley 1581 de 2012 (Protección de Datos Personales) y Ley 1266 de 2008 (Habeas Data Financiero).
- Trazabilidad y auditoría: requerir evidencia de los procesos, registros de consultas, y uso de la información.
- Reportes y documentación: posibilidad de solicitar y validar la información de conformidad con leyes y normas vigentes.

4.6.8. Auditoría Interna / Control Interno

Perfil: Personal encargado de monitorear y evaluar el cumplimiento de políticas internas, prácticas contables y reglamentarias dentro de la empresa.

- Registros detallados de acciones: acceso a logs de consultas, modificaciones y accesos de usuarios para detectar posibles irregularidades.
- Herramientas de reporte que permitan evaluar el grado de riesgo y cumplimiento de procedimientos.
- Notificaciones automáticas en caso de conductas sospechosas o picos inusuales de consultas a centrales de riesgo.

Capítulo 5

Selección de tecnologías

5.1. Investigación de patrones de arquitectura

ARQUITECTURA

Esta sección presenta los patrones de arquitectura candidatos —Capas, Hexagonal (o Limpia) y Microservicios— junto con los criterios de selección (**Escalabilidad**, **Complejidad** y **Seguridad**) que guiarán la elección inicial y la ruta de evolución del sistema de consulta de información crediticia.

1.1 Arquitectura en Capas

La arquitectura en capas (*Layered Architecture*) organiza el software en estratos bien definidos:

- **Capa de Presentación**: interacción con el usuario (interfaz web tipo SPA).
- **Capa de Negocio**: reglas de negocio y validaciones, separadas para facilitar pruebas.
- **Capa de Persistencia**: acceso y manipulación de datos.
- **Capa de Integración (opcional)**: adaptadores hacia servicios externos como las centrales de riesgo, si la conexión requiere protocolos o certificados específicos. Monroe et al. (1997)

1.2 Arquitectura de Microservicios

La arquitectura de microservicios (*Microservices Architecture*) consiste en dividir la aplicación en servicios pequeños e independientes, cada uno con su propia lógica, base de datos (en algunos casos) y ciclo de despliegue:

- **Servicios autónomos**: Cada servicio es responsable de un conjunto reducido de funcionalidades. Velepucha and Flores (2023)
- **Comunicación vía APIs**: Los servicios se comunican por mensajería o APIs REST, evitando compartir memoria o base de datos de forma directa. Velepucha and Flores (2023)
- **Despliegue independiente**: Se pueden actualizar, reescalar o reemplazar ciertos servicios sin afectar a los demás. Velepucha and Flores (2023)

1.3 Arquitectura Hexagonal (o Limpia)

La arquitectura hexagonal (*Clean/Hexagonal Architecture*) se basa en la idea de aislar la lógica de negocio del resto de componentes, mediante la definición de puertos y adaptadores:

- **Dominio central**: Encapsula completamente las reglas de negocio, modelos y casos de uso. Griffin (2020)
- **Adaptadores**: Conectan el dominio con fuentes externas (API REST, bases de datos, mensajería) o con la interfaz de usuario. Griffin (2020)
- **Puertos**: Definen contratos de comunicación (interfaces) para que el dominio no dependa directamente de la infraestructura. Griffin (2020)

5.2. Criterios de evaluación

Para seleccionar la arquitectura más adecuada, se consideran tres criterios de evaluación fundamentales en el contexto de integración para aplicaciones web: **Escalabilidad**, **Complejidad** y **Seguridad**.

Escalabilidad: Peticiones concurrentes por segundo sin afectar el rendimiento

A continuación presentamos métricas sobre escalabilidad entre arquitecturas Bass et al. (1999).

Criterio / Métrica	**Puntaje Máximo**	**Capas**	**Hexagonal**	**Microservicios**
Peticiones sostenidas (rps)	20	10	14	18
Latencia p95 (<500ms)	10	7	8	9
Utilización CPU <75 %	10	7	9	9
Escalado horizontal	10	5	7	10
Manejo de picos (temporadas agrícolas)	10	6	8	10
Automatización del escalado	10	5	7	10
Tolerancia a cuellos de botella	10	6	8	9
SUMA PARCIAL (ESCALABILIDAD)	**70**	**46**	**61**	**66**
Relación esfuerzo/beneficio estimado	–	Media	Alta	Alta

Tabla 5.1: Comparación de escalabilidad entre arquitecturas

Complejidad (evaluada mediante Cynefin)

La complejidad se evaluará por medio del marco Cynefin teniendo en cuenta los siguientes datos Álvaro Brandón et al. (2020); Cerný et al. (2024); Camilli et al. (2023)

Criterio / Indicador	Puntaje Máximo	Capas	Hexagonal	Microservicios
Dominio Cynefin (Complicado = 10, Complejo = 5)	10	10	10	5
Semanas hasta Go-Live (12 ideal)	10	9	8	6
Nº de repositorios / módulos (menos es mejor)	10	9	7	4
Pasos de CI/CD hasta QA (5 ideal)	10	8	7	6
Componentes de infraestructura adicionales	10	9	7	5
Necesidad de experticia (menos = más puntaje)	10	9	7	5
Facilidad para simplificar ante sobrecarga	10	9	8	6
TOTAL (COMPLEJIDAD)	**70**	**63**	**54**	**37**
Dominio Cynefin inicial esperado	—	Complicado	Complicado	Complejo

Tabla 5.2: Evaluación de complejidad usando Cynefin

Seguridad (evaluación tipo check list: 1, 0, -1)

Se realizara un analisis sobre temas de seguridad que se podran ver en la siguiente tabla Hannousse and Yahiouche (2020); Genfer et al. (2025); Zdun (2023).

#	Control / Categoría	Capas	Hexagonal	Microservicios
A. Gobierno y acceso				
1	RBAC (roles definidos y aplicados)	1	1	1
2	MFA para cuentas privilegiadas	0	1	1
B. Datos en tránsito y reposo				
3	Conexiones internas y externas	1	1	1
4	Cifrado de datos sensibles en almacenamiento	0	1	1
C. Desarrollo y despliegue				
5	Escaneo de dependencias críticas	0	1	1
6	Gestión de secretos (vault, env-vars seguros)	0	1	1
D. Auditoría y monitoreo				
7	Logs estructurados e inmutables	0	1	1
8	Alertas de anomalías configuradas	-1	0	1
E. OWASP Top-10				
9	Pruebas de inyección y XSS superadas	0	1	1
10	Seguridad de sesión (tokens, expiración)	1	1	1
	TOTAL SCORE (−10 a +10)	2	9	10
	% de cumplimiento (sobre 10 controles)	60 %	90 %	100 %
	¿Hallazgos críticos (−1)?	Sí	No	No

Tabla 5.3: Evaluación de cumplimiento de seguridad

5.3. Investigación: Arquitectura vs. Criterio

En esta sección se detalla cómo las distintas arquitecturas (Capas y Microservicios) responden a cada criterio. (La arquitectura Hexagonal podría combinarse con uno u otro enfoque, por lo que se centra la comparación en Capas vs. Microservicios directamente.)

Arquitectura en Capas

Escalabilidad (rendimiento por el mismo recurso; replicación cuando se satura; peticiones concurrentes para medir)

El sistema se despliega, en su fase inicial, como un monolito modular. La estrategia de escala consiste en replicar la aplicación completa detrás de un balanceador de carga: cada instancia contiene todas las capas (presentación, negocio, persistencia e integración). Blinowski et al. (2022); Sharma et al. (2005) Mientras la base de datos y la red no se conviertan en cuellos de botella, la ganancia de rendimiento tiende a ser casi lineal, puesto que cada réplica procesa el mismo conjunto

de peticiones. Blinowski et al. (2022); Sharma et al. (2005)

Cuando la demanda crece, esta réplica completa resulta sencilla de operar y suele bastar para proyectos de tamaño mediano. No obstante, si un submódulo interno —por ejemplo, la lógica de consulta a la central de riesgo— monopoliza CPU o E/S, duplicar todo el monolito se vuelve costoso: se incrementan recursos en capas que no lo requieren y se desperdicia capacidad. Blinowski et al. (2022); Sharma et al. (2005)

Para evaluar la escalabilidad se recomienda medir de forma simultánea el *throughput* (peticiones por segundo) y la latencia p95 de cada instancia. Los valores de referencia para el MVP se resumen en el Cuadro 3.1.

Escalabilidad – Arquitectura en Capas (Monolito Modular)

Para la escalabilidad se encontraron diferentes datos relevantes para tener en cuenta como los menciona Zyrianoff et al. (2019); Hayes et al. (2018); Zhuang et al. (2020)

Criterio / Métrica	Valor esperado (MVP)	Arquitectura en Capas
Throughput (rps)	≥ 20	✓Logrado con réplicas completas por balanceador
Latencia p95	≤ 500 ms	✓Estable si no hay cuellos en lógica o BD
Utilización de CPU	$< 75\,\%$ por instancia	✓Buen margen mientras no se sature la lógica
Escalado horizontal	Requerido	✓Réplica completa del sistema (sencillo de operar)
Costo de escalado	Medio	X Se duplican capas que no lo requieren
Tolerancia a cuellos de botella	Media	— Ineficiente si una sola capa satura recursos
Complejidad operativa	Baja	✓Fácil para equipos pequeños
Adecuación para MVP / proyecto mediano	Alta	✓Suficiente para primeras etapas
Flexibilidad para escalar por módulo	Baja	X Escala todo, no solo lo necesario

Tabla 5.4: Escalabilidad de la Arquitectura en Capas (Monolito Modular)

Complejidad: Cynefin para medir

En la versión MVP, la separación clásica en presentación, negocio y persistencia sitúa el proyecto en el dominio “Simple / Complicado” del marco Cynefin: se aplican prácticas conocidas, el número de artefactos es reducido y el equipo puede entregar valor con poca curva de aprendizaje. Nachbagauer (2021)

A medida que el código crece —por ejemplo, cuando se añadan más reglas crediticias o reportes

avanzados— existe el riesgo de que aparezcan dependencias cruzadas entre capas y el sistema derive hacia un dominio "Complejo": aumentan los repositorios, los pipelines de despliegue y la necesidad de orquestar cambios simultáneos. Kempermann (2017) Para evitarlo, se propone un control continuo de complejidad basado en los indicadores del siguiente cuadro Pinciroli (2024); Nachbagauer (2021); Maia and Longman (2024).

Indicador / Criterio	**Valor objetivo (MVP)**	**Interpretación / Acción**
Dominio Cynefin	Simple → Complicado	✓Mantener fuera del dominio Complejo. Refactorizar si se cruza.
Tiempo hasta primer despliegue	$\leq$ 12 semanas	- Si se excede, reducir alcance o automatizar tareas.
Repositorios de código	1 (monolito modular)	X Más de 1 = fragmentación prematura. Unificar o documentar.
Pasos en pipeline CI/CD	$\leq$ 5 pasos	- Más de 5 = sobre ingeniería. Eliminar redundancias.
Componentes de infraestructura	$\leq$ 1 externo (p. ej. proxy)	- Más de 1 = mayor curva DevOps. Introducir gradualmente.

Tabla 5.5: Evaluación de Complejidad – Arquitectura en Capas

Seguridad (evaluación tipo check list: 1, 0, -1)

Al operar como un único proceso, la arquitectura en capas permite centralizar la autenticación, la autorización y el registro de auditoría, lo que simplifica la implantación de controles de seguridad coherentes con la Ley 1581 de 2012 y la Ley 1266 de 2008. Para el MVP se adopta un check-list derivado de ISO 27001 y OWASP Top-10; cada control se califica con 1 (cumplido), 0 (parcialmente cumplido) o -1 (no cumplido). Razikin and Soewito (2022,?) El cuadro siguiente detalla los controles a evaluar y la meta mínima de cumplimientoCulot et al. (2021); Monev (2020); Kitsios et al. (2023).

Categoría	Control a verificar	Meta (MVP)
Gobierno y acceso		
	1. RBAC aplicado	1
	2. MFA en cuentas privilegiadas	1
Datos en tránsito		
	3. TLS 1.2 o superior en todas las conexiones	1
Datos en reposo		
	4. Cifrado de campos sensibles en BD	1
Código y dependencias		
	5. SCA sin vulnerabilidades críticas	1
Gestión de secretos		
	6. Vault o variables seguras (no en código fuente)	1
Auditoría		
	7. Logs estructurados e inmutables	1
Monitoreo y alertas		
	8. Detección de accesos anómalos / alertas	1
OWASP Top-10		
	9. Inyección y XSS mitigadas	1
	10. Tokens firmados y expirables	1

Tabla 5.6: Evaluación de Seguridad – Arquitectura en Capas (Check-list ISO / OWASP)

Arquitectura de Microservicios

Escalabilidad (replicación cuando se satura)

En el escenario distribuido, cada microservicio se despliega y escala de forma autónoma; el *throughput* global se logra aumentando únicamente las instancias del componente que se encuentra bajo mayor presión (por ejemplo, el servicio de consulta a la central de riesgo) sin duplicar el resto. Thatikonda (2023); Alelyani et al. (2024) Este aislamiento optimiza recursos y permite aplicar políticas de auto-scaling basadas en CPU, memoria o tasa de peticiones por instancia.

No obstante, el tejido de red y la orquestación añaden sobrecarga: la comunicación inter-servicio introduce latencias adicionales y exige mecanismos de descubrimiento, balanceo y circuit-breakingThatikonda (2023); Li et al. (2023). Para controlar estos factores, se define el conjunto de métricas y umbrales del Cuadro 3.x, centrado en el microservicio crítico de consultas externas y en el API Gateway que recibe el tráfico de clientesBlinowski et al. (2022); Horváth et al. (2024).

Criterio / Métrica	Valor esperado (MVP)	Arquitectura de Microservicios
Throughput (rps)	≥ 20 por instancia	✓Se escala solo el servicio que lo requiere (más eficiente).
Latencia p95	≤ 300 ms (servicio)	— Añade latencia de red, pero es compensada en diseño.
Utilización CPU	$< 70\,\%$ por microservicio	✓Monitoreo fino y autoescalado vía HPA.
Escalado horizontal	Requerido	✓Escalado independiente por microservicio crítico.
Costo de escalado	Bajo	✓Solo se escalan componentes con alta demanda.
Tolerancia a cuellos de botella	Alta	✓Mejor manejo por separación de servicios y límites por capa.
Complejidad operativa	Alta	X Requiere orquestador, gateway, discovery, observabilidad.
Adecuación para MVP / proyecto mediano	Media	— Potente, pero puede ser excesivo si no se justifica el esfuerzo.
Flexibilidad para escalar por módulo	Alta	✓Escala por dominio funcional. Ideal en entornos dinámicos.

Tabla 5.7: Escalabilidad: Arquitectura de Microservicios (Comparación uniforme)

Complejidad

Al migrar de un monolito en capas a microservicios, el proyecto pasa del dominio *Simple / Complicado* al *Complicado / Complejo* del marco Cynefin. Nachbagauer (2021); Saucedo et al. (2024) Surgen nuevos artefactos (repositorios, pipelines, imágenes de contenedor), más componentes de infraestructura (API Gateway, orquestador, malla de servicio) y mayor necesidad de observabilidad distribuida. Santos and Silva (2020) Para mantener la complejidad dentro de límites gestionables durante la fase evolutiva se controlarán los indicadores Camilli et al. (2023); Francesco et al. (2019).

Indicador / Criterio	**Meta evolutiva (fase microservicios)**	**Interpretación / Acción correctiva**
Dominio Cynefin	Complicado (evitar Complejo)	— Si cruza a Complejo: pausar expansión y revisar diseño.
Número de microservicios activos	≤ 5	X $>$ 5 indica fragmentación prematura; consolidar funcionalidades.
Repositorios de código	≤ 3	— Demasiados repos = sobrecarga CI/CD; agrupar servicios afines.
Pasos en pipeline CI/CD por servicio	≤ 8	— $>$ 8 implica tareas redundantes; automatizar y revisar flujos.
Componentes de infraestructura adicionales	API Gateway + K8s + observabilidad (máx 3)	— Nuevas herramientas deben justificar su valor técnico.
Tiempo para poner un servicio en producción	≤ 3 semanas	— Si se excede, reducir alcance o reforzar la automatización.

Tabla 5.8: Evaluación de Complejidad - Arquitectura de Microservicios

Seguridad (evaluación tipo check list: 1, 0, -1)

Al operar como un conjunto de servicios distribuidos, la solución incrementa su superficie de ataque: cada microservicio expone una API propia y se comunica con otros componentes a través de la red. Para mitigar ese riesgo se adopta el principio de defensa en profundidad, combinando autenticación estricta, cifrado punto-a-punto y gobierno centralizado de identidades mediante un API Gateway o una malla de servicio.

El cumplimiento se verifica con un check-list basado en ISO 27001 y OWASP Top-10, ampliado con controles específicos de entornos distribuidos. Culot et al. (2021); Malatji (2023) Cada ítem se puntúa con 1 (cumplido), 0 (parcial) o -1 (no cumplido)Roy (2020); Mirtsch et al. (2021); Kitsios et al. (2023).

Categoría / Capa de defensa	Controles a verificar (ejemplos)	Meta (MVP)
Gateway / Malla de servicio		
	1. Autenticación con JWT / OAuth2	1
	2. Rate-limiting y cuotas por ruta	1
Comunicaciones internas		
	3. mTLS con rotación	1
	4. Políticas Zero Trust por servicio	1
Gestión de secretos		
	5. Vault con rotación	1
	6. Sin secretos en código ni imágenes	1
Datos en reposo		
	7. Cifrado por servicio	1
	8. Backups cifrados y verificados	1
Código y dependencias		
	9. SCA sin CVE críticas	1
	10. Escaneo de imágenes de contenedor	1
OWASP Top-10 ampliado		
	11. Inyección y XSS mitigadas	1
	12. Protección sesión y CSRF en cada API	1
Observabilidad		
	13. Trazas distribuidas	1
	14. Alertas ante anomalías (picos 4xx/5xx)	1
Gobierno de identidades		
	15. Tokens con privilegio mínimo	1
	16. Expiración y revocación forzada	1
Cumplimiento legal		
	17. Consentimiento previo	1
	18. Trazabilidad de acceso (Ley 1581/2012)	1

Tabla 5.9: Check-list de seguridad para microservicios (ISO 27001, OWASP Top-10, Zero Trust)

5.4. Resultados

A continuación, se muestra una tabla con una evaluación comparativa a grandes rasgos de Capas vs Microservicios, considerando los tres criterios principales Hannousse and Yahiouche (2020); Junker (2021); Khazaei et al. (2020):

Criterio	Capas	Microservicios
Escalabilidad	7	10
Complejidad (6 meses)	9	5
Seguridad	7	9
Time-to-Market	9	6
Coste Operativo Inicial	9	5
Ruta de Evolución	7	10
TOTAL (máx. 60)	**48**	**45**
Costo/Beneficio	**Alto**	Medio
Requiere DevOps maduro	No	Sí

Tabla 5.10: Comparación global: Capas vs. Microservicios

Capas obtiene mayor puntuación total (48/60), siendo más viable para MVPs con recursos limitados y cronograma ajustado.

Microservicios sobresale en escalabilidad y evolución, pero implica mayor complejidad técnica y costo inicial. La elección recomendada es iniciar con arquitectura en capas, aplicando una ruta de evolución progresiva hacia microservicios si el sistema crece en dominios o volumen. Hannousse and Yahiouche (2020); Junker (2021); Tu (2023)

5.4.1. Elección recomendada

La arquitectura en capas modular es la opción más adecuada para este proyecto de grado porque:

- Cumple los requisitos de rendimiento definidos para el MVP ($\geq$ 20 rps, latencia p95 $\leq$ 500 ms), sin necesidad de infraestructura compleja o costosa. Tu (2023); Hylving and Schultze (2020)

- Minimiza la complejidad técnica para un equipo reducido y un plazo de 6 meses, mediante un monolito modular, un único repositorio y una cadena CI/CD simplificada (build-test-deploy). Hylving and Schultze (2020)

- Facilita la implementación de controles de seguridad y cumplimiento legal, al centralizar la autenticación, la auditoría y el cifrado, en línea con la Ley 1581 de 2012 y la Ley 1266 de 2008. Tu (2023)

- Establece una ruta de evolución controlada: si la carga aumenta, el módulo más crítico (como el de consulta a la central de riesgo) puede extraerse y convertirse en microservicio, aplicando patrones como Strangler Fig, sin necesidad de reescritura total. Tu (2023); Hylving and Schultze (2020)

En síntesis, una arquitectura en capas permite entregar valor en el corto plazo, con bajo riesgo académico y operativo, y escalar de forma progresiva en función del crecimiento de la organización y el volumen de transacciones.

5.4.2. Arquitecturas no seleccionadas

A pesar de sus ventajas en escalabilidad futura, se ha decidido no optar por la arquitectura de microservicios ni por la arquitectura hexagonal, debido a las siguientes razones:

- **Arquitectura de Microservicios**: aunque es altamente escalable y flexible, su implementación inicial implica mayor complejidad, mayor esfuerzo DevOps y un coste operativo superior, lo que puede dificultar su adopción en las etapas iniciales del MVP.
- **Arquitectura Hexagonal (o Limpia)**: si bien promueve un diseño desacoplado y centrado en el dominio, no constituye por sí sola una solución completa de despliegue o escalabilidad. En el contexto del MVP, la inversión para modelar correctamente puertos y adaptadores puede ser excesiva frente al beneficio inmediato.

En consecuencia, se descartan ambas opciones como arquitectura principal para el sistema inicial, privilegiando un enfoque probado, sencillo y adaptable como lo es la arquitectura en capas. Tu (2023); Hylving and Schultze (2020)

5.5. Conclusión

Dada la proyección de crecimiento del proyecto y la necesidad de una entrega de valor temprana, se concluye que la arquitectura en capas resulta la más conveniente para Agroinsumos del Cauca. Su simplicidad operativa, facilidad de implementación y compatibilidad con las metas de rendimiento del MVP permiten reducir riesgos en las primeras etapas.

Además, esta arquitectura facilita la introducción progresiva de mejoras tecnológicas: en caso de crecimiento del volumen de transacciones o de mayor necesidad de disponibilidad, ciertos componentes pueden ser extraídos y evolucionados hacia microservicios sin reescritura total del sistema. Esto establece una transición sostenible hacia arquitecturas más complejas, como se propone en la ruta evolutiva planteada.

Por tanto, se adopta un enfoque inicial basado en arquitectura en capas, priorizando la entrega funcional, la simplicidad en el despliegue y el cumplimiento de requisitos legales, con la posibilidad de escalar hacia microservicios si la operación futura lo demanda.

5.6. Lenguaje de Programación (Back-Front)

5.6.1. Front-End: JavaScript

JavaScript se ha consolidado como el lenguaje esencial para el desarrollo del Front-End en aplicaciones web porque los navegadores modernos —como Chrome, Firefox o Edge— lo interpretan de forma nativa, garantizando compatibilidad sin necesidad de herramientas intermedias. Lenguajes como TypeScript o Dart deben transformarse en JavaScript antes de ejecutarse, lo que refuerza su posición como estándar de facto en la web. Su evolución constante, impulsada por nuevas versiones de ECMAScript, introduce mejoras como funciones flecha, manejo asincrónico con `async/await` y módulos, facilitando la escritura de código más limpio y organizado.

Además, la comunidad de JavaScript es una de las más grandes del mundo, lo que permite acceso rápido a documentación, foros, librerías especializadas y recursos para resolver problemas comunes, acelerando el aprendizaje y la productividad del equipo de desarrollo. JavaScript también se integra fácilmente con servicios REST, WebSockets, pruebas automatizadas y librerías modernas.

Su popularidad se debe a su flexibilidad, facilidad de aprendizaje y la amplia disponibilidad de frameworks como React, Angular y Vue, que han revolucionado el desarrollo frontend al facilitar la creación de interfaces reactivas y modulares. La adopción universal de JavaScript en los navegadores y su ecosistema robusto lo convierten en la opción predeterminada para el desarrollo frontend Shukla (2023).

5.6.2. Back-End: Criterios de Evaluación

Para definir el lenguaje más adecuado en la capa del servidor, se analizaron tres criterios fundamentales que responden a las necesidades técnicas y operativas del sistema:

- **Rendimiento y Concurrencia:** Se requiere una tecnología capaz de responder eficientemente a múltiples usuarios realizando consultas simultáneas, sin sacrificar tiempos de respuesta ni saturar los recursos del sistema. Según un estudio reciente, lenguajes como Java y Go pueden superar en rendimiento a alternativas como JavaScript y Python, las cuales presentan desventajas significativas en cargas intensivas Lion et al. (2022).

Comparativa de Rendimiento y Concurrencia

Lenguaje	Peticiones por segundo (aprox.)	Latencia (ms)	Máx. conexiones concurrentes	Tiempo para 1M de peticiones (aprox.)
Java	20,000 – 50,000	10 – 30	10,000+	20 – 50 seg
Python	2,000 – 10,000	50 – 150	1,000 – 5,000	100 – 500 seg
JavaScript (Node.js)	10,000 – 30,000	20 – 80	10,000+	33 – 100 seg

Tabla 5.11: Comparación de desempeño entre lenguajes en entornos concurrentes

Como puede observarse en la Tabla 5.11, Java presenta los mejores resultados en cuanto a peticiones por segundo y menor latencia, lo que lo posiciona como una opción robusta para entornos con alta concurrencia. Estas características han sido respaldadas por estudios sobre transformación

de sistemas legados hacia Java, destacando su madurez y capacidad para manejar cargas críticas Hans et al. (2025).

Por otro lado, JavaScript (Node.js) ofrece un equilibrio entre rendimiento y escalabilidad, siendo especialmente útil en desarrollos full stack gracias a su uniformidad de lenguaje en cliente y servidor Dachepally (2023). Python, aunque presenta mayor latencia y menor capacidad concurrente, destaca por su simplicidad sintáctica, aunque puede ser menos eficiente en sistemas exigentes, como lo confirman estudios sobre métricas de complejidad de software Abdulkareem and Abboud (2021).

- **Ecosistema y Madurez:** Se priorizan lenguajes con un entorno de desarrollo estable, buena documentación y herramientas modernas. Además, se considera la fortaleza de su comunidad técnica, medida en parte por la cantidad de preguntas resueltas en foros como Stack Overflow y la frecuencia de actualizaciones de sus principales librerías Farshidi (2021).

Lenguaje	Paquetes Disponibles (aprox.)	Preguntas en Stack Overflow (aprox.)	Contribuidores en GitHub (aprox.)
Node.js (npm)	1,000,000+	4,500,000+	3,500+
Python (PyPI)	450,000+	2,000,000+	2,200+
Java (Maven Central)	500,000+	1,500,000+	1,000+

Tabla 5.12: Comparación del ecosistema y actividad comunitaria por lenguaje

El análisis de los ecosistemas de Node.js, Python y Java revela que la amplia disponibilidad de paquetes y la intensa actividad comunitaria son factores clave para la productividad y la innovación en el desarrollo de software. Node.js, por ejemplo, cuenta con un vasto repositorio de paquetes y una comunidad muy activa, lo que facilita la reutilización de código y acelera la resolución de problemas, aunque también implica desafíos en la gestión de dependencias y la exposición a vulnerabilidades Wittern et al. (2016). Python destaca por su gran número de paquetes y una comunidad participativa en foros como Stack Overflow, lo que permite a los desarrolladores encontrar soluciones rápidamente, pero también enfrenta un aumento en la cantidad de vulnerabilidades y retrasos en la actualización de dependencias Alfadel et al. (2021). Java, por su parte, mantiene un ecosistema robusto y estable, ideal para aplicaciones empresariales que requieren soporte a largo plazo. En conjunto, la cantidad de paquetes disponibles y la actividad de la comunidad reflejan la salud y utilidad de estos lenguajes, aunque requieren una gestión cuidadosa para mantener la calidad y la seguridad del software Gosling et al. (2013) Alfadel et al. (2021) .

- **Seguridad y Buenas Prácticas:** La protección de datos personales y financieros es crítica. Por ello, se contemplan aspectos como el soporte para cifrado, autenticación segura, validación de entradas y la existencia de marcos normativos que respalden buenas prácticas. Se sugiere elaborar una lista de verificación que permita evaluar objetivamente las capacidades de cada lenguaje en materia de seguridad Croft et al. (2021).

Lenguaje	Asincronía y Concurrencia	Tipado Estático	Soporte pruebas unitarias	Ecosistema Empresarial
Node.js	✓		✓	Alto
Python	✓		✓	Alto
Java	✓	✓	✓	Muy Alto

Tabla 5.13: Comparación de características clave por lenguaje

Node.js y Python destacan por su capacidad para manejar operaciones asíncronas y concurrentes, además de contar con un ecosistema empresarial sólido, aunque no disponen de tipado estático, lo que puede dificultar la generación automatizada de pruebas unitarias en PythonHussein et al. (2025).

Java, en cambio, integra asincronía, concurrencia, tipado estático y un soporte avanzado para pruebas unitarias, lo que lo convierte en una opción especialmente robusta y madura para aplicaciones empresarialesHussein et al. (2025).

Estas diferencias hacen que Java sea preferido en entornos críticos, mientras que Node.js y Python ofrecen mayor flexibilidad y rapidez en el desarrollo.

5.7. Investigación de frameworks

5.7.1. Frameworks para Backend

A. Spring Boot (Java)

Descripción: Framework de Java para crear aplicaciones independientes y productivas en poco tiempo. Proporciona configuraciones y una integración estrecha con Spring Security, Spring Data, etc.

Razones para Considerarlo:

- Madurez y robustez para aplicaciones empresariales, con gran soporte de la comunidad.
- Integración con Microservicios: Spring Cloud ofrece herramientas para la gestión de configuraciones, descubrimiento de servicios (Eureka), monitoreo (Hystrix) y enrutamiento (Zuul).
- Facilidad de integración con APIs externas (centrales de riesgo), gracias a librerías como Spring Web.

Justificación adicional:

El ecosistema Spring Boot se ha consolidado como un *estándar* en proyectos de misión crítica que operan con datos sensibles, especialmente en sectores como el financiero y el análisis de riesgo crediticio. Su robustez, alineación con prácticas modernas (microservicios, APIs RESTful), seguridad integrada y una gran comunidad de soporte lo convierten en la opción preferida para desarrollar el back-end de sistemas que requieren Mythily et al. (2022):

- Altos niveles de confiabilidad y rendimiento.
- Cumplimiento estricto en materia de protección de datos y trazabilidad (auditoría).
- Capacidad de evolución hacia arquitecturas complejas (microservicios) sin necesidad de reescribir el núcleo de la aplicación.

Por estas razones, Spring Boot satisface con creces los requerimientos de Agroinsumos del Cauca para la consulta de información crediticia, proporcionando además la productividad necesaria para cumplir los plazos y recursos limitadosHussein et al. (2025).

5.7.2. Frameworks para Frontend

La selección de tecnologías frontend surgió principalmente por la experiencia adquirida en el entorno laboral, donde hemos trabajado con Angular en varios proyectos, lo que me permitió familiarizarme con su estructura y enfoque modular. A su vez, durante mi formación académica he tenido contacto con otros frameworks como React y Next.js, lo que me brindó una visión más amplia del ecosistema frontend actual. Aunque Angular puede parecer más complejo al inicio, su organización y herramientas integradas resultan útiles para desarrollos estructurados. Next.js también fue considerado por su simplicidad y por ofrecer ventajas como el renderizado en servidor,

útil en aplicaciones ligeras o MVP. Esta combinación de conocimientos profesionales y académicos facilitó una comparación objetiva al momento de definir la mejor opción para el proyecto.

Framework	Carga Inicial (ms)	Renderizado (ms)	Memoria (MB)	Bundle (KB)	Curva de Aprendizaje
Angular	250–300	80–120	60–80	500–600	Alta
React	180–220	60–90	45–65	100–150	Media
Next.js	170–210	55–85	50–70	80–120	Media-baja

Tabla 5.14: Comparación de frameworks front-end según rendimiento y curva de aprendizaje

5.8. Definición tipos de bases de datos

Dado el contexto de Agroinsumos del Cauca y la importancia de la integridad y la auditoría en los datos:

- Utilizar una base de datos relacional como PostgreSQL o MySQL para la información principal:
  - Clientes (identificación, datos personales, consentimientos),
  - Historial de consultas a la central de riesgo,
  - Registro de transacciones y logs de auditoría.
- Posiblemente, Redis o MongoDB se podrían emplear como capa adicional de caché o almacenamiento de datos no críticos (ej.: datos de sesión, tokens temporales, logs extensos), si se requiere escalabilidad a mediano/largo plazo.

5.8.1. Criterios de Selección

- **Transacciones confiables:** La base de datos debe asegurar que cada operación se complete totalmente o se revierta en caso de error. Esto evita inconsistencias y garantiza la estabilidad de los procesos financieros.
- **Integridad de los datos:** La solución debe mantener la estructura lógica mediante claves primarias, foráneas y reglas de validación. Así se previenen duplicados, errores de relación o datos incompletos.
- **Manejo seguro de información:** El sistema debe proteger datos sensibles, incluso ante accesos simultáneos. Es clave evitar que fallos afecten la consistencia o expongan información crítica.

5.8.2. Bases de Datos Relacionales

5.8.2.1. PostgreSQL

Descripcion: Sistema de gestión de bases de datos relacional de código abierto, conocido por su robustez y soporte avanzado de transacciones (ACID) Salunke and Ouda (2024).

Ventajas:

- Excelente manejo de transacciones y funcionalidades avanzadas (procedimientos almacenados, extensiones geoespaciales, etc.).
- Comunidades y foros activos para la resolución de problemas.
- Escalabilidad vertical y horizontal razonable (réplicas, sharding a través de extensiones como Citus).

Desventajas:

- Puede requerir ajustes avanzados para grandes volúmenes de datos o cargas muy intensivas.

5.8.2.2. MySQL / MariaDB

Descripcion: Bases de datos relacionales muy populares y ampliamente soportadas. MySQL es mantenida por Oracle, mientras MariaDB es un fork comunitario Győrödi et al. (2020).

Ventajas:

- Simplicidad en la instalación y configuración inicial.
- Amplia adopción y soporte en prácticamente cualquier infraestructura.
- Variedad de motores de almacenamiento (InnoDB, MyISAM).

Desventajas:

- Para ciertas operaciones complejas, MySQL/MariaDB pueden presentar limitaciones o requerir configuraciones muy específicas en comparación con PostgreSQL.

5.8.2.3. Microsoft SQL Server

Descripción: SGBD comercial de Microsoft, muy utilizado en entornos empresariales, con fuerte integración a .NET Ceresnák and Kvet (2019).

Ventajas:

- Herramientas de administración robustas (SQL Server Management Studio).
- Buena escalabilidad vertical y sólida gestión de transacciones.
- Integraciones nativas con otros productos Microsoft (BI, reporting).

Desventajas:

- Costos de licenciamiento en ciertas versiones y dependencia histórica de entornos Windows (aunque existe SQL Server para Linux).

Motor/Engine	Base de datos	Lectura (ms)	Escritura (ms)
InnoDB	MySQL	10–15	12–18
PostgreSQL	PostgreSQL	8–12	10–15
SQL Server	SQL Server	9–13	11–16

Tabla 5.15: Comparación de rendimiento en lectura y escritura por motor de base de datos

Conclusión Relacional:

Al comparar los principales motores de bases de datos relacionales, como MySQL con InnoDB, PostgreSQL y SQL Server, se evidencia un rendimiento similar en operaciones de lectura y escritura. PostgreSQL se comporta especialmente bien en escritura, con tiempos bajos que lo hacen apto para sistemas exigentes Ceresnák and Kvet (2019). MySQL destaca por su equilibrio general y facilidad de gestión, siendo una opción común en entornos web Salunke and Ouda (2024). SQL Server, por otro lado, ofrece tiempos estables y ventajas en escalabilidad para soluciones empresariales [56]. Los resultados provienen de pruebas recientes que permiten medir el comportamiento real de cada motor. La elección final depende en gran medida del tipo de proyecto y sus requerimientos específicosGyőrödi et al. (2020).

5.8.3. Bases de Datos No Relacionales (NoSQL)

En proyectos que consultan centrales de riesgo, la consistencia de los datos es prioritaria. Según el teorema CAP, no se puede garantizar simultáneamente consistencia, disponibilidad y tolerancia a particiones. Las bases de datos no relacionales suelen sacrificar la consistencia, lo cual es inaceptable en contextos financieros. Además, este tipo de proyectos requiere relaciones claras entre entidades, como usuarios, créditos e historiales, lo que se gestiona mejor en bases de datos relacionales. Zhao et al. (2024).

Conclusión No Relacional: Las BD NoSQL son útiles para grandes volúmenes de datos no estructurados o para servir de caché distribuida en escenarios de alta concurrencia. Sin embargo, para procesos de evaluación crediticia con lógica transaccional y auditoría de datos sensibles, se recomienda combinarlas con un sistema relacional o emplearlas como almacenamiento complementario (logs, cache, etc.).

5.9. Selección de Infraestructura

Por lo general, tanto el backend como el sistema gestor de base de datos necesitan entre 2 y 4 núcleos de CPU y una asignación de memoria RAM que oscila entre 4 y 8 GB por componenteMullapudi et al. (2018). En contraste, la interfaz desarrollada con Angular puede funcionar adecuadamente con una menor cantidad de recursos. Sumando los requerimientos mínimos para un entorno básico, se recomienda disponer de entre 5 y 9 núcleos de procesamiento y de 9 a 18 GB de memoria RAM, sin olvidar el espacio de almacenamiento necesario para alojar la base de datosPursharthi and Deshmukh (2023).

Componente	**CPU recomendada**	**RAM recomendada**	**Almacenamiento recomendado**	**Infraestructura sugerida**
Spring Boot (Java)	2–4 núcleos	4–8 GB	10–20 GB	VM o contenedor dedicado; integración con pipelines DevOps
MySQL	2–4 núcleos	4–8 GB	Según volumen de datos	Instancia cloud o servidor dedicado; permite réplicas y respaldo automático
Angular (frontend)	1 núcleo	1–2 GB	1–5 GB	Servidor web ligero (Nginx/Apache) o CDN en la nube

Tabla 5.16: Requerimientos técnicos recomendados por componente

Capítulo 6

Modelado de la arquitectura

6.1. Definición de componentes

6.1.1. Identificación de componentes

Componente	Responsabilidad principal
clientes	Mantener el registro maestro del solicitante (identificación, datos de contacto, actividad).
autorizaciones	Almacenar el consentimiento firmado que habilita la consulta a la central de riesgo.
consultas	Ejecutar la petición a la central de riesgo, registrar estado, score y metadatos operativos.
reportes_credito	Consolidar los datos de clientes y las últimas consultas para entrega al front-end.
usuarios	Gestionar credenciales, información personal y asociación con roles para RBAC.

6.1.1.1. Definición de APIs por componente - Backend

Método	Ruta	Entrada (JsonBody)	Respuesta	Códigos
GET	`/clientes`	—	`array<cliente>`	200, 404
GET	`/clientes/{cliente_id}`	—	`cliente`	200, 404
POST	`/clientes`	`{ çedula": "string", "nombre": "string", .ªpellido": "string", "puntaje": 0, "fecha_consulta": "date", .ºcupacion": "string", "meses_castigo_reciente": 0, "numero_castigos": 0, çapacidad_endeudamiento": 0 }`	`{ çliente_id": üuid"}`	201, 400
PUT	`/clientes/{cliente_id}`	`{ çedula": "string", "nombre": "string", .ªpellido": "string", "puntaje": 0, "fecha_consulta": "date", .ºcupacion": "string", "meses_castigo_reciente": 0, "numero_castigos": 0, çapacidad_endeudamiento": 0 }`	`cliente`	200, 400
DELETE	`/clientes/{cliente_id}`	—	`{ "deleted": true }`	200, 404

- **autorizaciones**

Método	Ruta	Entrada	Respuesta	Códigos
POST	`/autorizaciones`	`{cliente_id:uuid, fecha_firma:date, canal:string}`	`{autorizacion_id:uuid}`	201, 400
GET	`/autorizaciones/{cliente_id}`	—	`{vigente:boolean, fecha_firma:date}`	200, 404

- **consultas**

Método	Ruta	Entrada	Respuesta	Códigos
POST	`/consultas`	`{cliente_id:uuid, motivo:string}`	`{consulta_id:uuid, estado:string}`	201, 400, 409*
GET	`/consultas/{consulta_id}`	—	`{consulta_id, fecha:datetime, score:int, estado:string}`	200, 404
GET	`/clientes/{cliente_id}/consultas`	—	`array<consulta>`	200, 404

* 409 se devuelve cuando no existe una autorización vigente para el cliente.

- **reportes_credito**

Método	Ruta	Entrada	Respuesta	Códigos
GET	`/reportes_credito/{cliente_id}`	—	`{cliente:{…}, consultas:[…]}`	200, 404

- **usuarios**

Método	Ruta	Entrada	Respuesta	Códigos
POST	`/usuarios`	`{nombre:string, email:string, password:string}`	`{usuario_id:uuid}`	201, 400
GET	`/usuarios/{usuario_id}`	—	`usuario`	200, 404
PUT	`/usuarios/{usuario_id}/roles`	`{roles:[string]}`	`usuario`	200, 400

6.1.1.2. Definición de componentes Front-End

- **ClientesComponent – Formulario Clientes**

Etiqueta	Nombre de campo	Tipo de dato	Obligatorio	Validación
Nombre	nombre	texto	Sí	2–50 caracteres
Apellido	apellido	texto	Sí	2–50 caracteres
Cédula	cedula	texto	Sí	Numérico, 8–15 dígitos
Puntaje	puntaje	número	No	0–1000
Fecha de consulta	fecha_consulta	fecha	Sí	$\leq$ fecha actual
Ocupación	ocupacion	texto	No	2–50 caracteres
Meses desde el castigo más reciente	meses_castigo_reciente	número	No	0–120
Número de castigos	numero_castigos	número	No	0–50
Capacidad de endeudamiento (%)	capacidad_endeudamiento	número	No	0–100

Notas: autocompletar correo en minúsculas; verificación de duplicados en línea.

- **AutorizacionesComponent – Formulario Autorizaciones**

Etiqueta	Nombre del campo	Tipo	Obl.	Validación
Cliente	cliente_id	autocomplete	Sí	UUID existente
Fecha de firma	fecha_firma	fecha	Sí	$\leq$ hoy
Canal	canal	select	Sí	{WEB, PRESENCIAL}
Vigente	vigente	checkbox	No	—

Notas: alerta si el cliente ya posee autorización activa.

- **ConsultasComponent – Formulario Consultas**

Etiqueta	Nombre del campo	Tipo	Obl.	Validación
Cliente	cliente_id	autocomplete	Sí	UUID existente
Motivo	motivo	texto	Sí	5–100 caracteres

- **ReportesCreditoComponent – Generación de Reporte**

Etiqueta	Nombre del campo	Tipo	Obl.	Validación
Cliente	cliente_id	autocomplete	Sí	UUID existente
Rango de fechas	rango_fechas	fecha doble	Sí	inicio $\leq$ fin

Notas: muestra tabla de consultas y gráfico de tendencia de scores.

- **UsuariosComponent – Formulario Usuarios**

Etiqueta	Nombre del campo	Tipo	Obl.	Validación
Nombre	nombre	texto	Sí	2–50 caracteres
Email	email	texto	Sí	formato RFC 5322
Password	password	password	Sí	8–20 caracteres
Roles	roles	multiselect	Sí	al menos 1 seleccionado

6.1.2. Definición del patrón de arquitectura.

6.1.2.1. Elección y justificación

Se adopta Arquitectura en Capas con un monolito modular para el MVP. Esta opción:

- Reduce complejidad operativa (un solo artefacto de despliegue).
- Facilita la consistencia de seguridad y transacciones (un mismo contexto de ejecución).
- Permite evolución progresiva hacia microservicios mediante patrones *Strangler Fig* cuando la carga lo exija.

6.1.2.2. Integración con TOGAF

Estos artefactos estructuran la documentación, promueven decisiones trazables y brindan un marco de control que vincula la solución técnica con la estrategia de reducción de riesgo crediticio de Agroinsumos del Cauca.

Fase ADM	Artefacto TOGAF propuesto	Aplicación al proyecto	Cómo respalda los objetivos
Pre-liminar	Architecture Principles (Principios de arquitectura)	1) Confidencialidad de datos crediticios; 2) Modularidad evolutiva (monolito→microservicios)	Proveen criterios para evaluar cambios y priorizar refactorizaciones.
A. Architecture Vision	Architecture Vision (1 página); Stakeholder Map	• Propósito: reducir riesgo de cartera vencida mediante consulta en línea a centrales de riesgo. • Alcance inicial: MVP (presentación + negocio + integración). • Beneficios: ↓25 • Actores: Dirección, Comercial, TI, Centrales de Riesgo.	Alinea la solución con la estrategia corporativa y fija métricas de éxito.
B. Business Architecture	Capability Map (nivel 1); BPMN v2.0 "Evaluar solvencia"	Capacidades: Clientes, Autorizaciones, Evaluación Crediticia y Gestión de Riesgo. Diagrama BPMN (Fig. 6.9) como deliverable oficial.	Garantiza trazabilidad entre requerimientos de negocio y componentes de software.
C. Information / Data Architecture	DERM; DFD contenedores	Entidades: cliente, autorización, consulta, usuario. Gobierno de datos: cifrado AES-256, retención 1 año, anonimización en pruebas.	Cumple Ley 1581/2012 y RNF-05/06 sobre protección de datos.
D. Application Architecture	C4 niveles 2–3; OpenAPI 3.1	Monolito modular (Presentation, Business, Persistence, Integration); 34 endpoints; versionado SemVer.	Permite Strangler Fig para evolución futura.
E. Technology Architecture	Tech Reference Model; Infra IaC	AWS Lightsail, Ubuntu 22.04, Nginx → Spring Boot 3 (JDK 21) → PostgreSQL 15; Docker, Terraform, CI/CD; TLS 1.3, JWT+MFA, Resilience4j.	Respaldan RNF-01/02 (rendimiento) y RNF-03/04 (disponibilidad).
F–G. Migration / Governance	Migration Roadmap (3 hitos); Compliance Assessment	H1: MVP → H2: extracción servicio Consulta CR → H3: event-driven - analytics.	Asegura alineación de cada release con métricas aprobadas.
H. Change Management	Change Log; Complexity Radar (Cynefin)	Se activa al pasar de "Complicado" a "Complejo" en nuevos requisitos.	Controla expansión progresiva y evita sobreingeniería.

Tabla 6.1: Integración de artefactos TOGAF con el proyecto

6.1.3. Definición de interfaces.

Figura 6.1: Inicio session.

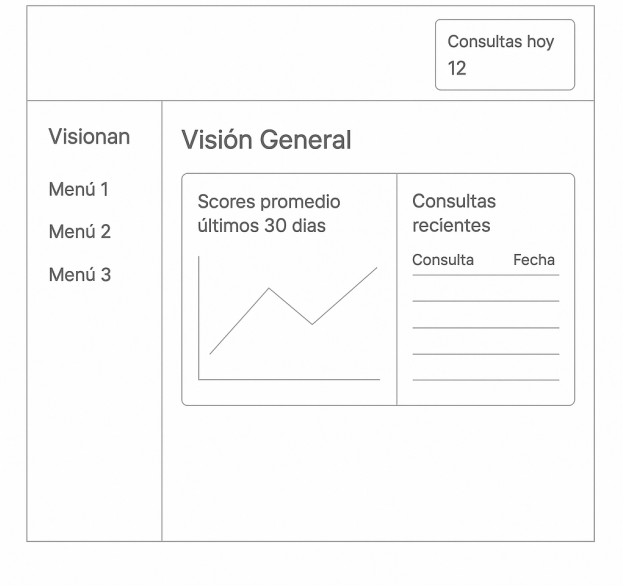


Figura 6.2: Vision general.

Figura 6.3: Detalles cliente.

Figura 6.4: Historial de consulta.

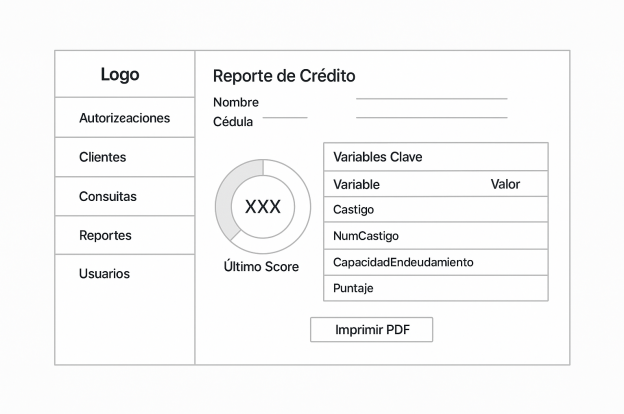


Figura 6.5: Reporte de crédito.

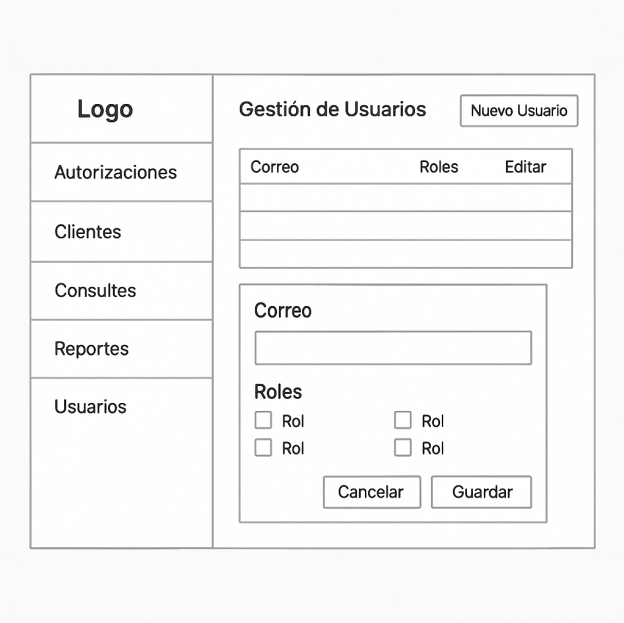


Figura 6.6: Gestión de usuario.

Figura 6.7: Autorizaciones.

6.1.4. Diagramación de arquitectura.

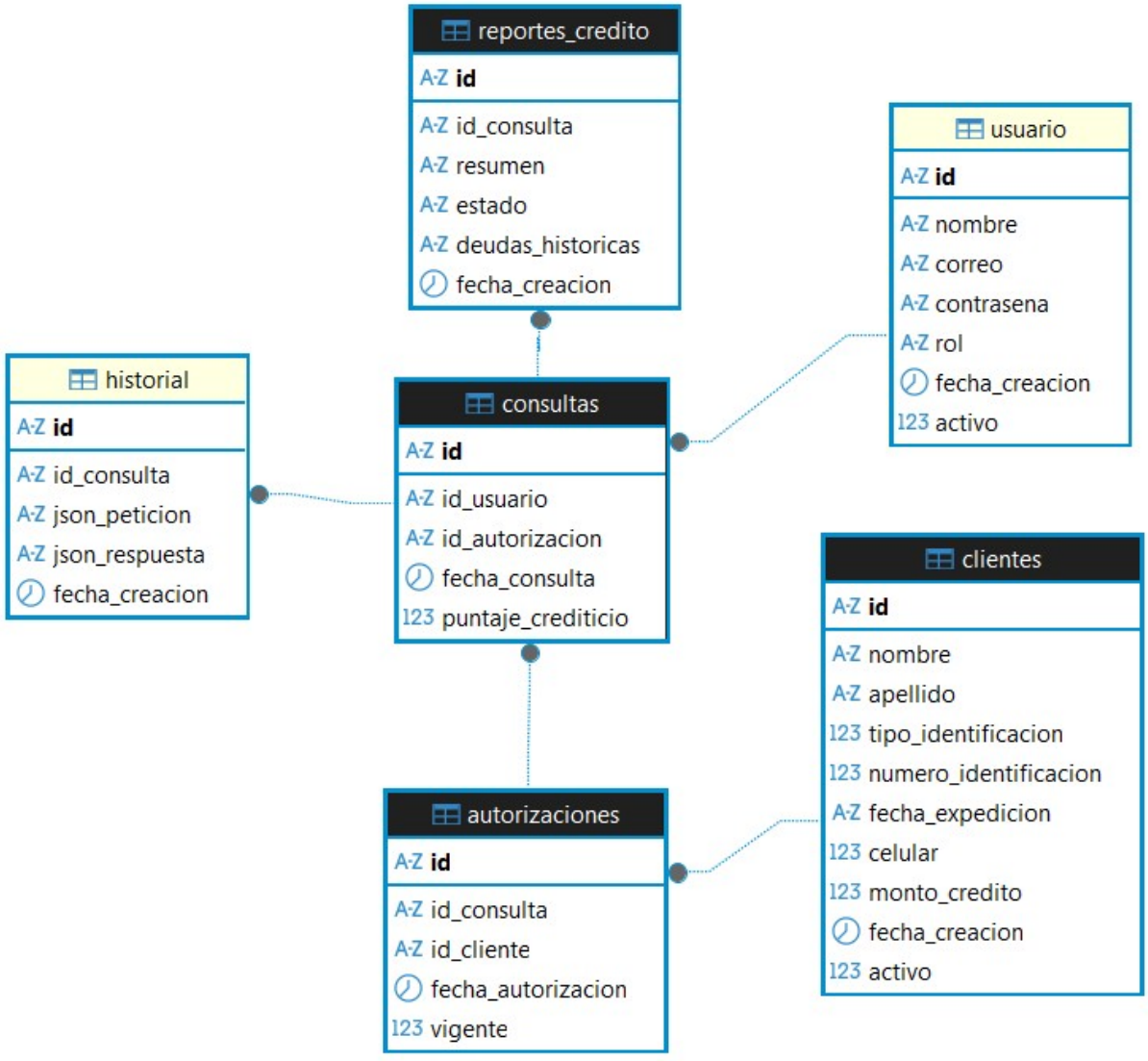


Figura 6.8: Diagrama de datos.

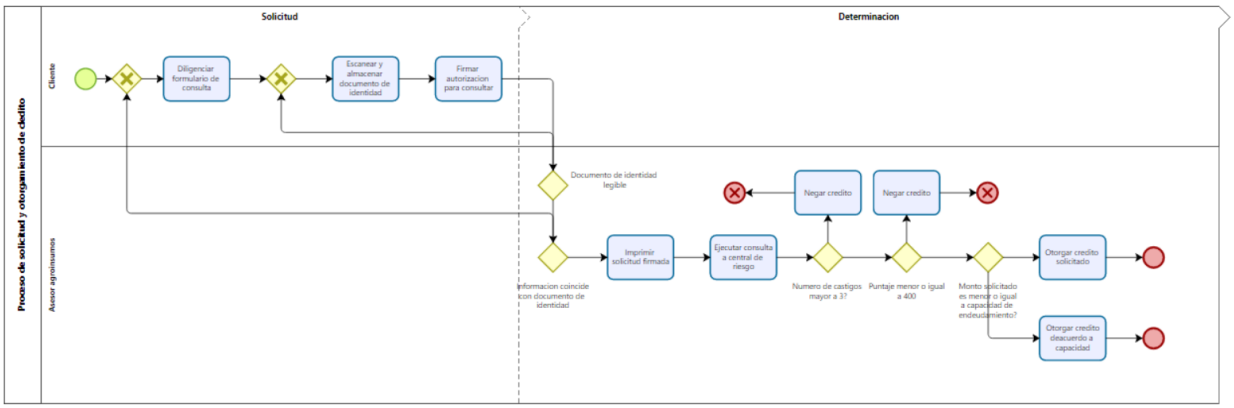


Figura 6.9: Diagrama de negocio.

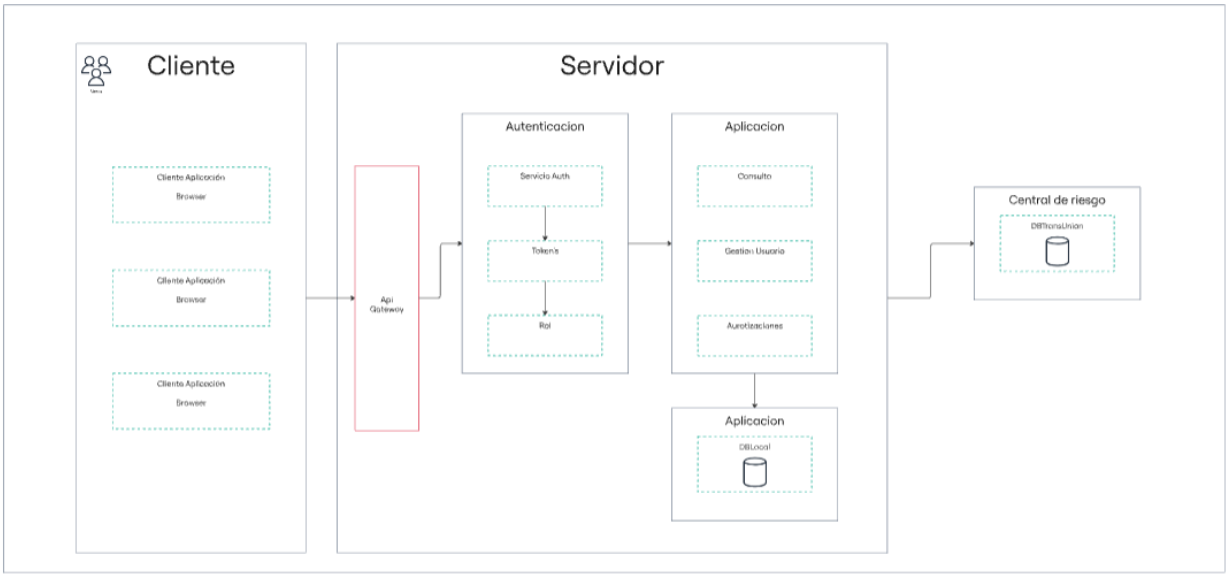


Figura 6.10: Diagrama de aplicaciones.

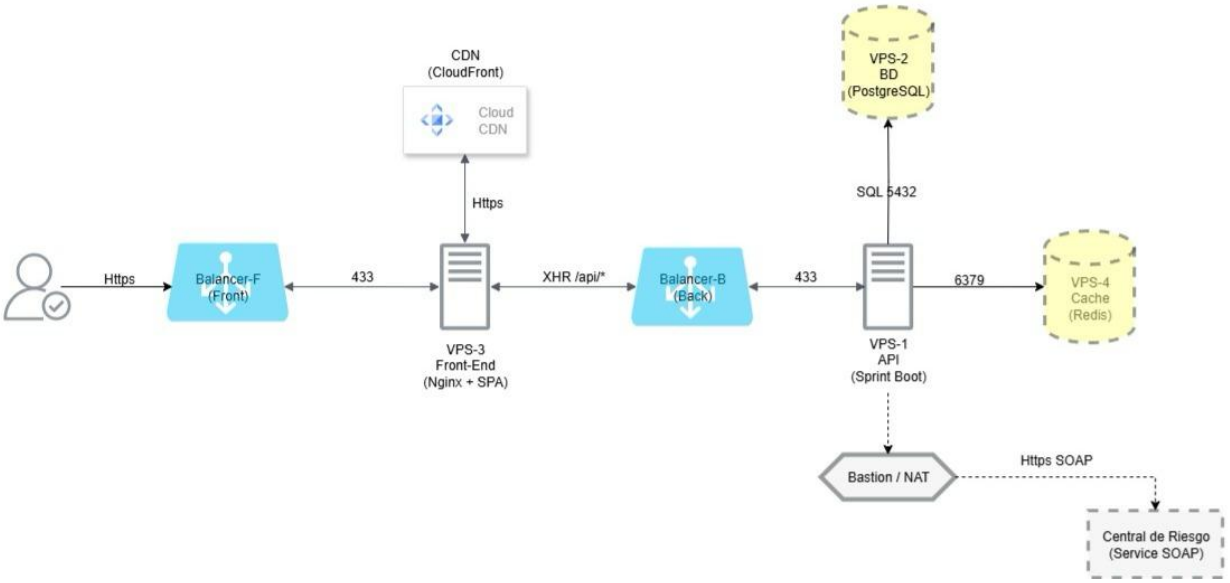


Figura 6.11: Diagrama de tecnología.

Capítulo 7

Conclusiones

1. Alineación estratégica y transformación digital

El proyecto confirma que una empresa agrícola de tamaño medio puede incorporar soluciones tecnológicas avanzadas sin perder agilidad. La arquitectura desarrollada se convierte en un habilitador clave de la estrategia, posicionando a *Agroinsumos del Cauca* como referente regional en modernización y competitividad.

2. Gestión del riesgo basada en datos

Al integrar consultas crediticias en tiempo real, la toma de decisiones pasa de ser empírica a estar sustentada en información verificable. Esto fomenta una cultura interna de decisiones informadas, disminuye la incertidumbre y fortalece las prácticas de crédito responsable.

3. Cumplimiento normativo como pilar de confianza

El diseño incorpora desde el inicio los lineamientos de la Ley 1581 (Protección de Datos) y la Ley 1266 (Habeas Data). Cumplir estos requisitos refuerza la reputación corporativa, reduce riesgos regulatorios y transmite seguridad a clientes y aliados.

4. Cambio organizacional y madurez digital

La coordinación entre las áreas comercial, financiera y de TI consolidó buenas prácticas de colaboración, gobernanza y medición de métricas. Este capital humano y procedimental se convierte en un activo para futuras iniciativas de innovación.

5. Vinculación academia–empresa

El enfoque *Design Science Research*, apoyado en marcos de arquitectura empresarial, demostró que la investigación académica puede generar artefactos con impacto inmediato en la industria. La sinergia universidad-empresa enriqueció el proceso y entregó resultados tangibles.

6. Escalabilidad sostenible

Optar por un monolito modular para el *MVP* equilibra costo y simplicidad, manteniendo abierta la vía de evolución a microservicios cuando el volumen de transacciones lo exija. Se evita la sobre-ingeniería inicial y se garantiza crecimiento controlado.

7. Ética y responsabilidad en el uso de datos

El proyecto enfatiza el consentimiento informado, la anonimización en pruebas y la auditoría de accesos como prácticas indispensables. Esto fortalece la confianza de los interesados y sirve de modelo para otras organizaciones que manejen datos sensibles.

8. Oportunidades de trabajo futuro

1. Integrar un segundo buró de crédito para mejorar la cobertura y confiabilidad de la información.
2. Automatizar la precalificación mediante un motor de reglas que reduzca aún más los tiempos de aprobación.
3. Desarrollar una *Progressive Web App* que permita operar en zonas con conectividad limitada, ampliando el impacto social.
4. Avanzar hacia la certificación ISO 27001 mediante la formalización completa del Sistema de Gestión de Seguridad de la Información.

7.1. Trabajos futuros

- Desacoplar el módulo de consultas en un microservicio independiente.
  - Aplicar el patrón *Strangler Fig* para aislar la lógica de integración con la central de riesgo y habilitar escalado horizontal específico cuando el volumen de transacciones crezca.
- Añadir un segundo proveedor de información crediticia (Datacrédito).
  - Implementar un conector REST adicional y una capa de orquestación que seleccione el mejor bureau o consolide la respuesta, reduciendo la tasa de “no hit” y mejorando la cobertura de datos.
- Automatizar la precalificación con un motor de reglas ligero.
  - Incorporar Drools (o equivalente) para que las políticas de riesgo se gestionen sin cambios de código; se espera disminuir el tiempo de aprobación interno en un 30 %.

- Entregar un canal móvil para la fuerza comercial.
  - Desarrollar una Progressive Web App que permita iniciar consultas, capturar consentimientos y mostrar el score en campo, incluso con conectividad intermitente.
- Fortalecer la seguridad hacia certificación ISO 27001.
  - Extender el sistema de gestión actual con evaluación formal de riesgos, políticas documentadas y auditorías internas, como paso previo a la certificación.

7.2. Lecciones aprendidas

1. **Prototipar antes de construir.** Crear maquetas de la interfaz y hacer pruebas con el equipo comercial permitió detectar ambigüedades a tiempo y evitó retrabajos costosos en fases posteriores.

2. **Menos es más en la primera versión.** Iniciar con una solución sencilla y bien acotada –un núcleo monolítico– facilitó la puesta en marcha y liberó recursos para validar el valor real del sistema antes de complejizarlo.

3. **Regulación primero, tecnología después.** Integrar la normativa de protección de datos desde el diseño evitó correcciones de última hora y reforzó la cultura de privacidad dentro de la organización.

4. **Colaboración transversal.** Reunir a las áreas comercial, financiera y de TI en los mismos talleres agilizó la definición de requisitos y fomentó un lenguaje común que perdura más allá del proyecto.

5. **El poder de los datos reales (anónimos).** Probar la solución con información anonimizada del propio negocio otorgó credibilidad a los resultados y ayudó a refinar reglas de decisión sin comprometer la confidencialidad.

6. **Planificar la escalabilidad, no implementarla de inmediato.** Diseñar una ruta clara para desacoplar el módulo de consultas y añadir nuevos proveedores de información permitió alinear expectativas de crecimiento sin incurrir en sobre-ingeniería.

7. **Documentar es invertir en el futuro.** La guía de despliegue, los diagramas de arquitectura y los catálogos de requisitos redujeron la dependencia de personas clave y facilitarán futuras auditorías y certificaciones.

8. **La tecnología móvil multiplica el alcance.** Identificar temprano la necesidad de un canal fuera de oficina evidenció la importancia de pensar en escenarios de conectividad limitada para llegar realmente al cliente rural.

9. **Seguridad como proceso continuo.** Las pruebas dinámicas y la gestión centralizada de accesos demostraron que la seguridad no se “añade” al final: es un ciclo que acompaña cada iteración del desarrollo.

10. **Sinergia academia–empresa.** El acompañamiento metodológico de la universidad aportó rigor y perspectiva crítica, mientras que la empresa brindó un entorno real donde validar aportes. Esta combinación consolidó un caso de éxito replicable.

Capítulo 8

Anexos

8.1. Recopilación de requisitos

8.1.1. Requisitos funcionales

1. Gestión de Usuarios y Roles

1.1 **RF-01: Registro y autenticación de usuarios:**

- **Descripción:** El sistema debe permitir el registro de nuevos usuarios y la autenticación de los existentes, garantizando que solo personal autorizado acceda a la información crediticia.
- **Actores involucrados:** Administrador, Usuario Comercial, Usuario Financiero.

2.2 **RF-02: Asignación de roles y permisos:**

- **Descripción:** El sistema debe soportar la asignación de distintos roles (por ejemplo, administrador, analista, comercial) y conceder permisos diferenciados para realizar consultas, ver informes y administrar la plataforma.
- **Actores involucrados:** Administrador.

3.3 **RF-03: Recuperación de contraseñas:**

- **Descripción:** El sistema debe ofrecer un mecanismo para que un usuario recupere o restablezca su contraseña en caso de olvido.
- **Actores involucrados:** Todos los usuarios.

2. Consulta de Información a Centrales de Riesgo

2.1 **RF-04: Búsqueda y consulta de clientes:**

- **Descripción:** El sistema debe permitir al usuario ingresar un identificador único (cédula, NIT) para consultar la información crediticia del cliente en una o varias centrales de riesgo.
- **Actores involucrados:** Usuario Comercial, Usuario Financiero.

2.2 **RF-05: Conexión con la central de riesgo:**

- **Descripción:** El sistema debe establecer una conexión segura (Web Service o API REST) con al menos una central de riesgo autorizada para obtener el historial crediticio, calificación o score del cliente.

- **Actores involucrados:** Usuario Comercial, Usuario Financiero.

2.3 **RF-06: Presentación de resultados de la consulta:**

- **Descripción:** El sistema debe mostrar de forma clara y resumida la información crediticia obtenida (score, historial de pagos, reportes negativos/positivos, etc.).
- **Actores involucrados:** Usuario Comercial, Usuario Financiero.

2.4 **RF-07: Reintento de consulta en caso de error:**

- **Descripción:** Si ocurre un error en la comunicación con la central de riesgo, el sistema debe permitir reintentos controlados, mostrando al usuario un mensaje de error y la opción de reintentar la operación.
- **Actores involucrados:** Usuario Comercial, Usuario Financiero.

3. Visualización y Manejo de la Información Crediticia

3.1 **RF-08: Historial de consultas:**

- **Descripción:** El sistema debe mantener un registro de las consultas realizadas, incluyendo fecha, hora, usuario que realizó la consulta, datos del cliente consultado y resultado.
- **Actores involucrados:** Administrador, Usuario Comercial, Usuario Financiero.

3.2 **RF-09: Reporte de estado crediticio:**

- **Descripción:** El sistema debe generar un reporte básico (por ejemplo, en PDF o en pantalla) que compile la información crediticia obtenida de la central de riesgo, para su análisis posterior o impresión.
- **Actores involucrados:** Usuario Comercial, Usuario Financiero.

3.3 **RF-10: Filtro y búsqueda de clientes consultados:**

- **Descripción:** El sistema debe permitir filtrar el historial de consultas por nombre de cliente, identificación o fecha de consulta, para facilitar la localización de registros específicos.
- **Actores involucrados:** Usuario Comercial, Usuario Financiero, Administrador.

4. Administración y Seguridad

4.1 **RF-11: Configuración de la integración con centrales de riesgo:**

- **Descripción:** El sistema debe incluir un módulo de configuración donde el administrador pueda actualizar las credenciales de acceso, endpoints, tokens o certificados de las centrales de riesgo, en caso de que se requiera.
- **Actores involucrados:** Administrador.

4.2 **RF-12: Políticas de privacidad y términos de uso:**

- **Descripción:** El sistema debe contar con un apartado que muestre al usuario las políticas de uso de datos y el compromiso de confidencialidad, de acuerdo con la normatividad de protección de datos.

- **Actores involucrados:** Todos los usuarios.

4.3 **RF-13: Alertas de seguridad y auditoría:**

- **Descripción:** El sistema debe notificar al administrador sobre accesos no autorizados, intentos fallidos de inicio de sesión frecuentes y cualquier incidente de seguridad relacionado con la consulta de información crediticia.
- **Actores involucrados:** Administrador.

5. Mantenimiento y Soporte

5.1 **RF-14: Registro de incidentes y soporte:**

- **Descripción:** El sistema debe proporcionar un mecanismo básico de reporte de incidentes (errores, fallas o dudas en la operación) para que los usuarios comuniquen problemas y el equipo de soporte los gestione.
- **Actores involucrados:** Usuario Comercial, Usuario Financiero, Administrador.

5.2 **RF-15: Actualización de versión del sistema:**

- **Descripción:** El sistema debe permitir la instalación de parches o actualizaciones (ej. cambios en endpoints de la central de riesgo) sin interrumpir de manera prolongada la operación de consultas.
- **Actores involucrados:** Administrador (o Equipo de TI).

8.1.2. Requisitos no funcionales

1. Rendimiento y Concurrencia

1.1 **RNF-01: Tiempo de respuesta:**

- **Descripción:** El sistema debe procesar y devolver la información crediticia en un tiempo promedio inferior a 5 segundos bajo condiciones de carga normal.
- **Objetivo:** Garantizar que el área comercial pueda tomar decisiones oportunas sin demoras excesivas.

2.2 **RNF-02: Usuarios concurrentes:**

- **Descripción:** El sistema debe soportar al menos 20 usuarios concurrentes realizando consultas, sin que esto afecte significativamente el rendimiento.
- **Objetivo:** Asegurar que el sistema sea estable para el tamaño actual del personal de Agroinsumos del Cauca.

2. Disponibilidad y Confiabilidad

2.1 **RNF-03: Disponibilidad mínima:**

- **Descripción:** El sistema debe estar disponible al menos el 95 % del horario laboral (por ejemplo, de 8:00 a 18:00, lunes a sábado).

- **Objetivo:** Evitar interrupciones críticas en la operación diaria.

2.2 **RNF-04: Recuperación ante fallos:**

- **Descripción:** Si el sistema o la conexión con la central de riesgo falla, debe recuperarse o restablecerse en un tiempo máximo de 2 horas.
- **Objetivo:** Minimizar el tiempo de inactividad y la afectación a los procesos de consulta.

3. Seguridad y Cumplimiento Legal

3.1 **RNF-05: Comunicación segura:**

- **Descripción:** Todas las operaciones sensibles (login, consulta a la central de riesgo) deben realizarse a través de HTTPS para proteger la información transmitida.
- **Objetivo:** Evitar la interceptación de datos confidenciales (man-in-the-middle).

3.2 **RNF-06: Protección de datos personales:**

- **Descripción:** El sistema debe cumplir la Ley 1581 de 2012 y la Ley 1266 de 2008 (Habeas Data), asegurando la confidencialidad y el tratamiento adecuado de la información crediticia.
- **Objetivo:** Cumplir con la normatividad colombiana y salvaguardar la privacidad de los clientes.

4. Usabilidad

4.1 **RNF-07: Interfaz intuitiva:**

- **Descripción:** La aplicación debe presentar una interfaz sencilla y coherente, con instrucciones claras, para que un usuario con conocimientos básicos pueda utilizarla sin una capacitación extensa.
- **Objetivo:** Reducir la curva de aprendizaje y minimizar errores de operación.

4.2 **RNF-08: Compatibilidad con navegadores:**

- **Descripción:** El sistema debe funcionar correctamente en los navegadores más utilizados (Chrome, Firefox, Edge) en sus versiones estables.
- **Objetivo:** Asegurar el acceso universal del personal con equipos y navegadores comunes.

5. Mantenibilidad

5.1 **RNF-09: Diseño modular básico:**

- **Descripción:** El sistema debe estructurarse de forma que las principales funcionalidades (consulta, seguridad, reporte) estén separadas para facilitar la localización de fallas y futuras extensiones.
- **Objetivo:** Mantener el proyecto ordenado y con facilidad para la adición de nuevas funciones en el corto/mediano plazo.

5.2 **RNF-10: Documentación esencial:**

- **Descripción:** Debe existir una breve guía técnica y de despliegue (por ejemplo, un archivo README y comentarios en el código) que describa la estructura general y los pasos de instalación/actualización.
- **Objetivo:** Permitir al equipo de TI dar soporte y realizar ajustes sin grandes dependencias.

6. Auditoría y Trazabilidad

6.1 **RNF-11: Registro de acciones clave:**

- **Descripción:** El sistema debe almacenar registros de consultas a centrales de riesgo (quién, cuándo, a quién se consultó) y de inicios de sesión.
- **Objetivo:** Contar con una trazabilidad mínima que permita detectar usos indebidos o inconsistencias.

6.2 **RNF-12: Reportes básicos de auditoría:**

- **Descripción:** El administrador debe poder generar o visualizar un reporte básico de las operaciones realizadas (por rango de fechas, por usuario).
- **Objetivo:** Ofrecer un control primario sobre la actividad y la seguridad del sistema.

8.2. Criterios de aceptación

8.2.1. Gestión de Usuarios y Acceso

1. **CA-01: Registro y Autenticación**

   - **Dado que** un usuario nuevo desea acceder al sistema,
   - **Cuando** complete el formulario de registro con datos válidos (nombre, correo, contraseña),
   - **Entonces** recibirá un correo de confirmación (o validación) y podrá iniciar sesión satisfactoriamente.
   - **Y si** un usuario ya registrado ingresa credenciales válidas (usuario y contraseña),
   - **Entonces** el sistema debe mostrar la pantalla principal y permitir el uso de funcionalidades según su rol.
   - **Criterios medibles:**
     - El sistema valida en menos de 2 segundos la autenticación.
     - Se envía un correo o notificación de confirmación si el registro es exitoso.
     - Si las credenciales son inválidas, se muestra un mensaje de error claro.

2. **CA-02: Roles y Permisos**

- **Dado que** el administrador del sistema desea asignar un rol (ej. comercial, analista, administrador) a un usuario,
- **Cuando** se selecciona al usuario y el rol en la interfaz de administración,
- **Entonces** el sistema guardará el nuevo rol y mostrará un mensaje de confirmación.
- **Y si** el usuario inicia sesión con ese rol,
- **Entonces** verá únicamente las opciones y datos que su perfil permite.
- **Criterios medibles:**
  - Se registra la modificación de rol en el log de auditoría.
  - La interfaz de cada usuario solo muestra las opciones autorizadas en su rol.
  - La modificación tarda menos de 2 segundos en reflejarse.

8.2.2. Consulta a Centrales de Riesgo

1. **CA-03: Búsqueda y Consulta de Clientes**

   - **Dado que** un usuario con permisos para consultar (comercial, analista) ingresa un número de identificación válido (cédula/NIT),
   - **Cuando** presiona "Consultar" en la pantalla principal,
   - **Entonces** el sistema se conecta a la central de riesgo y retorna la información crediticia del cliente (score, historial, etc.) en menos de 5 segundos.
   - **Criterios medibles:**
     - El sistema verifica si la identificación tiene el formato correcto (longitud, tipo).
     - Si la central de riesgo está disponible, la respuesta muestra como mínimo:
       - Nombre del cliente,
       - Score crediticio,
       - Estado de reporte (negativo/positivo),
       - Último historial de mora (si aplica).
     - Si la central está inaccesible, el sistema muestra un mensaje de error y un botón de reintento.

2. **CA-04: Presentación de Resultados**

   - **Dado que** el sistema obtiene el reporte crediticio,
   - **Cuando** lo recibe de la central de riesgo,
   - **Entonces** lo despliega en una pantalla de resumen y permite la generación de un reporte en PDF con un clic.
   - **Criterios medibles:**

- La pantalla de resumen incluye campos obligatorios (puntaje, estado, fecha de último reporte).
- La exportación a PDF no demora más de 5 segundos.
- El reporte PDF contiene la fecha de la consulta y los datos principales del cliente.

8.2.3. Registro de Actividades y Auditoría

1. **CA-05: Historial de Consultas**

   - **Dado que** un usuario comercial o financiero accede al módulo "Historial de Consultas",
   - **Cuando** visualiza la tabla de registros,
   - **Entonces** debe observar cada consulta con la siguiente información:
     - Fecha y hora,
     - Usuario que la realizó,
     - Identificación del cliente consultado,
     - Resultado resumido (ej.: Score o estado principal).
   - **Criterios medibles:**
     - El sistema almacena y muestra al menos los últimos 100 registros de forma paginada o scrollable.
     - Los datos de fecha/hora coinciden con la zona horaria del sistema (ej.: Colombia UTC-5).
     - La búsqueda de registros por fecha o identificación del cliente devuelve resultados en menos de 3 segundos.

2. **CA-06: Reporte de Auditoría**

   - **Dado que** el administrador requiere exportar un reporte de auditoría,
   - **Cuando** solicita el reporte por rango de fechas (inicio-fin),
   - **Entonces** el sistema genera un archivo (CSV o PDF) con los principales eventos (inicios de sesión, consultas de historial crediticio, cambios de rol).
   - **Criterios medibles:**
     - El administrador puede especificar filtros de fechas (e.g., desde-hasta).
     - El archivo se genera en menos de 10 segundos si abarca hasta 3 meses de registros.
     - El documento exportado muestra cada evento con su marca de tiempo, el usuario responsable y la acción realizada.

8.2.4. Seguridad y Protección de Datos

1. **CA-07: Comunicación Segura (HTTPS)**
   - **Dado que** un usuario se conecta al sistema vía navegador,
   - **Cuando** intenta acceder a la URL del sistema,
   - **Entonces** la comunicación debe realizarse siempre por HTTPS (certificado vigente), evitando advertencias de seguridad del navegador.
   - **Criterios medibles:**
     - Navegadores soportados (Chrome, Firefox, Edge) no muestran alertas de certificado no confiable.
     - No se permite el acceso por HTTP sin cifrado (redirige automáticamente a HTTPS).
     - Logs de servidor confirman el uso de protocolos TLS actuales (ej.: TLS 1.2 o superior).
2. **CA-08: Cumplimiento con Protección de Datos (Ley 1581 de 2012)**
   - **Dado que** el sistema maneja información crediticia sensible,
   - **Cuando** el usuario ve o descarga datos de clientes,
   - **Entonces** el sistema debe mostrar un aviso de privacidad o términos de uso, y registrar en el log quién accedió o descargó dicha información.
   - **Criterios medibles:**
     - El aviso de privacidad debe aparecer al menos 1 vez por sesión o estar disponible de forma visible (ej.: en el footer).
     - Toda operación de "descarga" queda registrada con el usuario y la fecha/hora.
     - El sistema no permite acceder a datos de clientes sin iniciar sesión previamente.

8.2.5. Mantenibilidad y Actualizaciones

1. **CA-09: Estructura Modular Básica**
   - **Dado que** el equipo técnico revise el repositorio de código,
   - **Cuando** se exploran los módulos (autenticación, consultas, reportes),
   - **Entonces** deben encontrarse carpetas o paquetes bien diferenciados, con documentación mínima (README, comentarios) que describan responsabilidades.
   - **Criterios medibles:**
     - Cada módulo debe tener un archivo README o documentación indicando su propósito.
     - El despliegue se realiza siguiendo pasos claros (instalación de dependencias, configuración de credenciales).

- Cambios en un módulo (por ejemplo, consultas) no deben romper otro (por ejemplo, autenticación) según las pruebas de integración.

2. **CA-10: Facilidad de Actualización**

   - **Dado que** se deba actualizar una librería o endpoint de la central de riesgo,
   - **Cuando** se aplica el parche o actualización en ambiente de pruebas,
   - **Entonces** el sistema debe retomar la operación sin mayores cambios en el código principal y volver a producir resultados correctos tras la actualización.
   - **Criterios medibles:**
     - El tiempo de inactividad en cada actualización no debe superar 30 minutos en horario no laboral o acordado.
     - El equipo de TI ejecuta un pequeño script o procedimiento documentado para la actualización.
     - Después de la actualización, una prueba de consulta de ejemplo confirma que el sistema funciona sin errores.

8.3. Integración TransUnion IDVision – Extracto técnico

Fuente original: TransUnion Colombia, *IDVision – Especificación para integración sistema a sistema* v1.4, sept-2021. Se reproducen únicamente los elementos necesarios para la solución descrita en esta tesis. El PDF íntegro se incluye en el repositorio digital adjunto.

8.3.1. Endpoints HTTPS (TLS 1.2)

Ambiente	**URL del servicio `ExternalSolutionExecution.svc` (WSDL)**
UAT	`https://www.transuniondecisioncentreuat.com.mx/TU.IDS.ExternalServices_mex_latam/SolutionExecution/ExternalSolutionExecution.svc?singleWsdl`
Producción	`https://www.transuniondecisioncentre.com.mx/TU.IDS.ExternalServices_latam_prod/SolutionExecution/ExternalSolutionExecution.svc?singleWsdl`

Tabla 8.1: Puntos de conexión para IDVision.

Tag	Tipo	Regla/Rango	Propósito en la integración
`UserId`	Alfanumérico	–	Usuario DE asignado por TU.
`Password`	Alfanumérico	–	Contraseña del usuario.
`SolutionSetId`	Numérico	`142`	Identificador de la solución `IDVGO3`.
`ExecuteLatestVersion`	Boolean	`true`	Garantiza versión vigente en prod.
`IdType`	Entero	1 (CC) / 3 (CE)	Tipo de documento.
`IdNumber`	Entero	1 000 000–1 999 999 999 (sin ceros a la izq.)	Nº de documento.
`IdExpeditionDate`	Fecha	dd/mm/aaaa	Obligatorio para CC.
`RecentPhoneNumber`	Entero	3000000000–3999999999	Celular verificado.

Tabla 8.2: Campos críticos enviados por la capa de integración (IDVision).

Código	Subsistema	Descripción resumida
100	IDV	Identidad validada exitosamente (Pass)
103	IDV	Sin coincidencias suficientes (Fail)
300	OTP	OTP aprobado (Pass)
302	OTP	OTP fallido / intentos agotados (Fail)
400	IDA	Preguntas reto aprobadas (Pass)
401	IDA	Preguntas reto fallidas (Fail)

Tabla 8.3: Subset de códigos usado por la lógica de negocio.

8.3.2. Campos mínimos del request `ExecuteXMLString`

8.3.3. Códigos de respuesta relevantes

8.4. Integración TransUnion AQM Go Habeas Data – Extracto técnico

Fuente original: TransUnion Colombia, *AQM Go Habeas Data – Especificación para integración sistema a sistema* v1.2, jun-2021.

8.4.1. Endpoints HTTPS (TLS 1.2)

Ambiente	URL del servicio `ExternalSolutionExecution.svc` (WSDL)
UAT	`https://www.transuniondecisioncentreuat.com.mx/TU.IDS.ExternalServices_mex_latam/SolutionExecution/ExternalSolutionExecution.svc?singleWsdl`
Producción	`https://www.transuniondecisioncentre.com.mx/TU.IDS.ExternalServices_latam_prod/SolutionExecution/ExternalSolutionExecution.svc?singleWsdl`

Tabla 8.4: Puntos de conexión para AQM Go Habeas Data.

8.4.2. Campos obligatorios del request (Prospección + Cuota)

Tag	Tipo	Regla / Comentario
`IdType`	Lista	1 (CC), 3 (CE).
`IdNumber`	Entero	Máx. 10 dígitos, sin 0 inicial.
`Occupation`	Enum	{Empleado, Pensionado, Independiente}.
`MonthlyIncome`	Entero	$\geq 500\,000$ COP (solo flujo Prospección+Cuota).
`Product`	Enum	{CONSUMO, LIBRE INVERSION}.

Tabla 8.5: Campos AQM utilizados en los escenarios de prueba.

8.4.3. Variables clave del resultado

XPath	Uso en la solución
`/DCResponse/ResponseInfo/ApplicationId`	ID de la transacción (trazabilidad).
`/ContextData/Field/Applicants/Applicant/CV_SCORE`	Score CreditVision almacenado para reglas.
`/ContextData/Field/Applicants/Applicant/PorcEndFinal`	Porcentaje de endeudamiento final.
`/ContextData/Field/ApplicationData/DecisionHomologada`	Decisión binaria usada en la capa de negocio.

Tabla 8.6: Elementos extraídos del XML de respuesta AQM.

Todos los extractos reproducidos se presentan con fines académicos. Las marcas y contenidos pertenecen a TransUnion LLC. Para detalles completos consúltese el material digital adjunto.

Bibliografía

Abdulkareem, S. and Abboud, A. (2021). Evaluating python, c++, javascript and java programming languages based on software complexity calculator (halstead metrics). In *IOP Conference Series: Materials Science and Engineering*, volume 1076, page 012046.

Alelyani, A., Datta, A., and Hassan, G. (2024). Optimizing cloud performance: A microservice scheduling strategy for enhanced fault-tolerance, reduced network traffic, and lower latency. *IEEE Access*, 12:35135–35153.

Alfadel, M., Costa, D., and Shihab, E. (2021). Empirical analysis of security vulnerabilities in python packages. *Empirical Software Engineering*, 28:1–37.

Aloul, F., Z. S. . E.-H. W. (2009). Two factor authentication using mobile phones. 2009 IEEE/ACS International Conference on Computer Systems and Applications. In *Two factor authentication using mobile phones. 2009 IEEE/ACS International Conference on Computer Systems and Applications.*

Angel., T. L. D. (2017). Sistema (ERP) orientado a la web para la gestión de control de los procesos administrativos y la comercialización del cacao en la empresa agrícola kimcuarenta sa. de la provincia santo domingo de los tsáchilas. In *Sistema (ERP) orientado a la web para la gestión de control de los procesos administrativos y la comercialización del cacao en la empresa agrícola kimcuarenta sa. de la provincia santo domingo de los tsáchilas.*

BancoRepublica (2023). Riesgo de crédito - Informe especial de Estabilidad Financiera - Primer semestre 2023. In *Riesgo de crédito - Informe especial de Estabilidad Financiera - Primer semestre 2023.*

Bass, L., C. P. . K.-R. (2021). Software Architecture in Practice (4th ed.). Addison-Wesley Professional. In *Software Architecture in Practice.*

Bass, L., Clements, P., and Kazman, R. (1999). *Software Architecture in Practice.* Addison-Wesley.

Blinowski, G., Ojdowska, A., and Przybyłek, A. (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access*, 10:20357–20374.

C4Model (2024). C4 Model. In *C4 Model.*

Camilli, M., Colarusso, C., Russo, B., and Zimeo, E. (2023). Actor-driven decomposition of microservices through multi-level scalability assessment. *ACM Transactions on Software Engineering and Methodology*, 32:1 – 46.

Ceresnák, R. and Kvet, M. (2019). Comparison of query performance in relational and non-relational databases. *Transportation Research Procedia*, 40:1271–1278.

Cerný, T., Abdelfattah, A. S., Yero, J., and Taibi, D. (2024). From static code analysis to visual models of microservice architecture. *Clust. Comput.*, 27:4145–4170.

Croft, R., Xie, Y., Zahedi, M., Babar, M., and Treude, C. (2021). An empirical study of developers' discussions about security challenges of different programming languages. *Empirical Software Engineering*, 27.

Culot, G., Nassimbeni, G., Podrecca, M., and Sartor, M. (2021). The iso/iec 27001 information security management standard: literature review and theory-based research agenda. *The TQM Journal*.

Dachepally, R. (2023). Implementing cross-platform apis with node.js, python and java. *Journal of Mathematical & Computer Applications*.

Dillon, T., W. C. . C.-E. (2010). New opportunities for NewSQL. Communications of the ACM, 58. In *New opportunities for NewSQL. Communications of the ACM, 58*.

FAO (2023). Experiencias inspiradoras para transformar los sistemas agroalimentarios en America Latina y el Caribe. In *Experiencias inspiradoras para transformar los sistemas agroalimentarios en America Latina y el Caribe*.

Farshidi, S. (2021). A decision model for programming language ecosystem selection: Seven industry case studies. *Information and Software Technology*, 139:106640.

Fernandes, D. A. B., S. L. F. B. G. J. V. F. M. M. . I. P. R. M. (2014). Security issues in cloud environments: a survey. International Journal of Information Security,. In *Security issues in cloud environments: a survey. International Journal of Information Security,.*

Francesco, P. D., Lago, P., and Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. *J. Syst. Softw.*, 150:77–97.

FuncionPublica (2012). Ley 1581 de 2012. In *Ley 1581 de 2012*.

Genfer, P., Serbout, S., Simhandl, G., Zdun, U., and Pautasso, C. (2025). Understanding security tactics in microservice apis using annotated software architecture decomposition models – a controlled experiment. *Empirical Software Engineering*, 30.

Gosling, J., Joy, B., Steele, G., Bracha, G., and Buckley, A. (2013). *The Java Language Specification, Java SE 8 Edition*. Oracle America, Inc.

Griffin, J. (2020). Hexagonal-driven development. In *97 Things Every Java Programmer Should Know*, pages 521–544. Apress.

Győrödi, C., Dumşe-Burescu, D., Zmaranda, D., Győrödi, R., Gabor, G., and Pecherle, G. (2020). Performance analysis of nosql and relational databases with couchdb and mysql for application's data storage. *Applied Sciences*, 10(23):8524.

Hannousse, A. and Yahiouche, S. (2020). Securing microservices and microservice architectures: A systematic mapping study. *Comput. Sci. Rev.*, 41:100415.

Hans, S., Kumar, A., Yasue, T., Ono, K., Krishnan, S., Sondhi, D., Satoh, F., Mitchell, G., Kumar, S., and Saha, D. (2025). Automated testing of cobol to java transformation.

Hayes, M., Ng, B. K. F., Pekár, A., and Seah, W. (2018). Scalable architecture for sdn traffic classification. *IEEE Systems Journal*, 12:3203–3214.

Hohpe, G., . W. B. (2004). Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. In *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.*

Horváth, M., Sakhnenko, V., and Gurbál', F. (2024). Comparison of scalability and performance in microservices and monolithic architectures. *2024 IEEE 17th International Scientific Conference on Informatics (Informatics)*, pages 82–87.

Hussein, S., Abbas, S., Ali, G., Hadi, N., Mohamed, M., and Maadi, M. (2025). A comparative analysis of programming languages used in microservices. *International Journal of Computational and Experimental Science and Engineering.*

Hylving, L. and Schultze, U. (2020). Accomplishing the layered modular architecture in digital innovation: The case of the car's driver information module. *J. Strateg. Inf. Syst.*, 29:101621.

IBM (2024). ¿Qué es una API REST? In *¿Qué es una API REST?*

ISO/IEC-27001 (2013). Information technology - Security techniques - Information security management systems - Requirements. In *Information technology - Security techniques - Information security management systems - Requirements.*

Junker, A.-M. (2021). Microservices as architectural style. In Bruel, J.-M., Mazzara, M., and Meyer, B., editors, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment (DEVOPS 2020)*, volume 12584 of *Lecture Notes in Computer Science*, pages 177–190. Springer.

Kempermann, G. (2017). Cynefin as reference framework to facilitate insight and decision-making in complex contexts of biomedical research. *Frontiers in Neuroscience*, 11.

Khazaei, H., Barna, C., and Litoiu, M. (2020). Performance modeling of microservice platforms. *IEEE Transactions on Cloud Computing*, 10:2848–2862.

Kitsios, F., Chatzidimitriou, E., and Kamariotou, M. (2023). The iso/iec 27001 information security management standard: How to extract value from data in the it sector. *Sustainability.*

Li, X., Zhou, J., Wei, X., Li, D., Qian, Z., Wu, J., Qin, X., and Lu, S. (2023). Topology-aware scheduling framework for microservice applications in cloud. *IEEE Transactions on Parallel and Distributed Systems*, 34:1635–1649.

Lion, D., Chiu, A., Stumm, M., and Yuan, D. (2022). Investigating managed language runtime performance: Why javascript and python are 8x and 29x slower than c++, yet java and go can be faster? In *Proceedings of the 2022 USENIX Annual Technical Conference (USENIX ATC '22)*, pages 835–852.

Maia, N. and Longman, D. (2024). Using the cynefin framework in the evolution of process maturity in software as a service (saas) development. *Anais do XX Workshop Anual do MPS (WAMPS 2024)*.

Majchrzak, T. A., . H. L. . (2018). rogressive Web Apps: The Definite Approach to Cross-Platform Development? In *Progressive Web Apps: The Definite Approach to Cross-Platform Development? Proceedings of the 51st Hawaii International Conference on System Sciences*.

Malatji, M. (2023). Management of enterprise cyber security: A review of iso/iec 27001:2022. *2023 International Conference On Cyber Management And Engineering (CyMaEn)*, pages 117–122.

MinAgricultura (2024). En el primer trimestre de 2024, el agro jalonó la economía, al registrar un crecimiento del 5.5 % . In *En el primer trimestre de 2024, el agro jalonó la economía, al registrar un crecimiento del 5.5 %*.

MinisterioAgricultura (2017). ARQUITECTURA DE SISTEMAS DE INFORMACIÓN. In *ARQUITECTURA DE SISTEMAS DE INFORMACIÓN*.

MinTic (2021). Tecnologías 4.0, herramientas clave para mejorar la productividad del agro colombiano. In *Tecnologías 4.0, herramientas clave para mejorar la productividad del agro colombiano*.

Mirian, Medali Castañeda Berru, A. C. M. A. (2023). Implementación de una plataforma web para mejorar la gestión agrícola en una cooperativa agraria en la provincia de Tocache. In *Implementación de una plataforma web para mejorar la gestión agrícola en una cooperativa agraria en la provincia de Tocache*.

Miro (2024). Mockups vs prototipos. In *Mockups vs prototipos*.

Mirtsch, M., Kinne, J., and Blind, K. (2021). Exploring the adoption of the international information security management system standard iso/iec 27001: A web mining-based analysis. *IEEE Transactions on Engineering Management*, 68:87–100.

Monev, V. (2020). Organisational information security maturity assessment based on iso 27001 and iso 27002. *2020 International Conference on Information Technologies (InfoTech)*, pages 1–5.

Monroe, R. T., Kompanek, A., Melton, R. E., and Garlan, D. (1997). Architectural styles, design patterns, and objects. *IEEE Softw.*, 14:43–52.

Montoya, E. A. Q., C. S. F. J. M. W. Y. C. y. G. G. E. C. (2017). Propuesta de una Arquitectura para Agricultura de Precisión Soportada en IoT. RISTI - Revista Iberica de Sistemas e Tecnologias de Informacao. In *Propuesta de una Arquitectura para Agricultura de Precisión Soportada en IoT. RISTI - Revista Iberica de Sistemas e Tecnologias de Informacao*.

Mullapudi, M., Mullapudi, L., and Gorantla, M. (2018). Decoding the java toolkit: An insightful analysis of frameworks for large-scale distributed applications. *International Journal of Science and Research (IJSR)*.

Mythily, M., Raj, S., and Joseph, I. (2022). An analysis of the significance of spring boot in the market. In *2022 International Conference on Inventive Computation Technologies (ICICT)*, pages 1277–1281.

Nachbagauer, A. (2021). Managing complexity in projects: Extending the cynefin framework. *Platform and Society*, 2:100017.

Node.js., F. (2021). Node.js Documentation. In *Node.js Documentation.*

Octavio Sotomayor, Eduardo Ramírez, H. M. (2022). Digitalización y cambio tecnológico en las mipymes agrícolas y agroindustriales en América Latina. In *Digitalización y cambio tecnológico en las mipymes agrícolas y agroindustriales en América Latina.*

Pahl, C., B. A. S. J. . J. P. (2017). Cloud container technologies: a state-of-the-art review. ieee transactions on cloud computing. In *Cloud container technologies: a state-of-the-art review. IEEE Transactions on Cloud Computing.*

Pinciroli, F. (2024). Selection of agile project management approaches based on project complexity. *Journal of Software: Evolution and Process*, 36.

Pivotal. (2021). Spring Boot Reference Guide. In *Spring Boot Reference Guide.*

Pressman, R. S. (2020). Ingeniería de Software: Un Enfoque Práctico. In *Ingeniería de Software: Un Enfoque Práctico.*

Pursharthi, P. and Deshmukh, K. (2023). Building a modern banking system: Implementation of a java spring boot and angular framework-based solution. *International Journal of Research Publication and Reviews.*

Razikin, K. and Soewito, B. (2022). Cybersecurity decision support model to designing information technology security system based on risk analysis and cybersecurity framework. *Egyptian Informatics Journal.*

Roy, P. P. (2020). A high-level comparison between the nist cyber security framework and the iso 27001 information security standard. *2020 National Conference on Emerging Trends on Sustainable Technology and Engineering Applications (NCETSTEA)*, pages 1–3.

Salunke, S. and Ouda, A. (2024). A performance benchmark for the postgresql and mysql databases. *Future Internet*, 16:382.

Santos, N. and Silva, A. R. (2020). A complexity metric for microservices architecture migration. *2020 IEEE International Conference on Software Architecture (ICSA)*, pages 169–178.

Saucedo, A. M., Rodríguez-Ortiz, G., Rocha, F. G., and dos Santos, R. P. (2024). Migration of monolithic systems to microservices: A systematic mapping study. *Inf. Softw. Technol.*, 177:107590.

Shahin, M., B. M. A. . Z. L. (2017). Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practice. In *Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practice*.

Sharma, V., Jalote, P., and Trivedi, K. S. (2005). Evaluating performance attributes of layered software architecture. In *International Conference on Software Engineering and Formal Methods (SEFM)*, volume 3469 of *Lecture Notes in Computer Science*, pages 66–81. Springer.

Shukla, A. (2023). Modern javascript frameworks and javascript's future as a fullstack programming language. *Journal of Artificial Intelligence Cloud Computing*.

Stonebraker, M. (2015). New opportunities for NewSQL. Communications of the ACM, 58. In *New opportunities for NewSQL. Communications of the ACM, 58*.

Thatikonda, V. (2023). Assessing the impact of microservices architecture on software maintainability and scalability. *European Journal of Theoretical and Applied Sciences*.

Tu, Z. (2023). Research on the application of layered architecture in computer software development. *Journal of Computing and Electronic Information Management*.

Velepucha, V. and Flores, P. (2023). A survey on microservices architecture: Principles, patterns and migration challenges. *IEEE Access*, 11:88339–88358.

Vera, P., Pons, C., González González, C., and Rodríguez, R. (2015). La interfaz de usuario como punto de partida para la creación automática de aplicaciones móviles – un enfoque basado en mdd. *Revista Colombiana de Computación*, 16:162–177.

W3C (2023). SOAP Version 1.2. In *SOAP Version 1.2*.

Wittern, E., Suter, P., and Rajagopalan, S. (2016). A look at the dynamics of the javascript package ecosystem. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 351–361.

Zdun, U. (2023). Conformance assessment and detection strategies in continuously delivered microservice architectures. *Journal of Systems and Software*, 199:111540.

Zhao, Y., Li, J., and Yang, L. (2024). Cap: Data contamination detection via consistency amplification.

Zhuang, Y., Chen, Y.-W., Shae, Z., and Shyu, C. (2020). Generalizable layered blockchain architecture for health care applications: Development, case studies, and evaluation. *Journal of Medical Internet Research*, 22.

Zyrianoff, I. D., Heideker, A., Silva, D., Kleinschmidt, J. H., Soininen, J., Cinotti, T. S., and Kamienski, C. (2019). Architecting and deploying iot smart applications: A performance–oriented approach. *Sensors (Basel, Switzerland)*, 20.

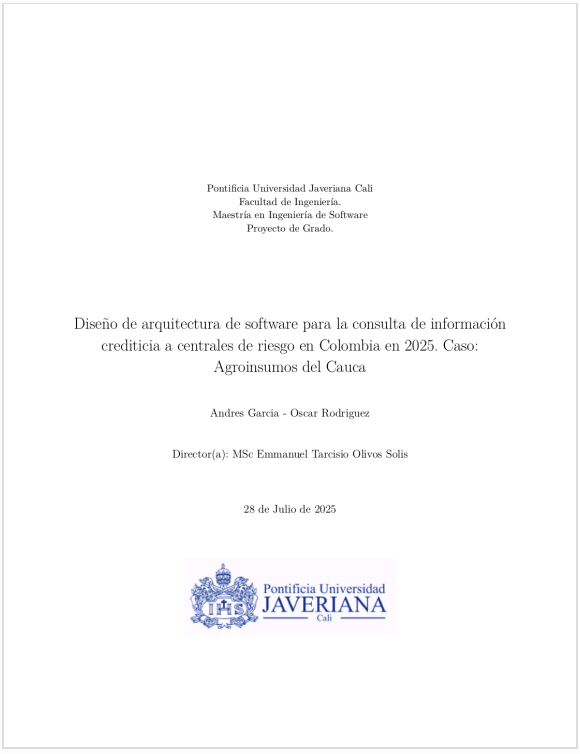
Álvaro Brandón, Solé, M., Huélamo, A., Solans, D., Pérez, M. S., and Muntés-Mulero, V. (2020). Graph-based root cause analysis for service-oriented and microservice architectures. *J. Syst. Softw.*, 159.

Recibo digital

Este recibo confirma quesu trabajo ha sido recibido por Turnitin. A continuación podrá ver la información del recibo con respecto a su entrega.

La primera página de tus entregas se muestra abajo.

Autor de la entrega:	Andres Garcia Avila
Título del ejercicio:	[Actividad] Valida tus trabajos con Turnitin
Título de la entrega:	Proyecto_de_grado_Diseno_arq_de_sofware_caso_agroinsumo...
Nombre del archivo:	Proyecto_de_grado_Diseno_arq_de_sofware_caso_agroinsumo...
Tamaño del archivo:	8.05M
Total páginas:	105
Total de palabras:	22,599
Total de caracteres:	132,113
Fecha de entrega:	28-jul-2025 01:30p. m. (UTC-0500)
Identificador de la entrega:	2722001911



Pontificia Universidad Javeriana Cali
Facultad de Ingeniería.
Maestría en Ingeniería de Software
Proyecto de Grado.

Diseño de arquitectura de software para la consulta de información crediticia a centrales de riesgo en Colombia en 2025. Caso: Agroinsumos del Cauca

Andres Garcia - Oscar Rodriguez

Director(a): MSc Emmanuel Tarcisio Olivos Solis

28 de Julio de 2025

Pontificia Universidad JAVERIANA Cali