UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE SOFTWARE

Tema:

Implantación de una arquitectura de integración de sistemas mediante microservicios y servicios distribuidos para la optimización de los procesos de supervisión y regulación de construcciones en la provincia de Imbabura

Trabajo de Grado previo a la obtención del título de Ingeniero en Software

AUTORA:

Guacanes Diaz Kevin Andres

DIRECTOR:

MSc. Diego Javier Trejo España

Ibarra, 2026

IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1004047179		
APELLIDOS Y NOMBRES:	GUACANES DIAZ KEVIN ANDRES		
DIRECCIÓN:	LA DOLOROSA DEL PRIORATO, PIMAN - PURUHANTA		
EMAIL:	kaguacanesd@utn.edu.ec		
TELÉFONO FIJO:	N/A	**TELF. MOVIL**	0987415867

DATOS DE LA OBRA	
TÍTULO:	IMPLANTACIÓN DE UNA ARQUITECTURA DE INTEGRACIÓN DE SISTEMAS MEDIANTE MICROSERVICIOS Y SERVICIOS DISTRIBUIDOS PARA LA OPTIMIZACIÓN DE LOS PROCESOS DE SUPERVISIÓN Y REGULACIÓN DE CONSTRUCCIONES EN LA PROVINCIA DE IMBABURA.
AUTOR (ES):	GUACANES DIAZ KEVIN ANDRES
FECHA:	2026/03/31
SOLO PARA TRABAJOS DE INTEGRACIÓN CURRICULAR	
CARRERA/PROGRAMA:	■ **GRADO** □ **POSGRADO**
TITULO POR EL QUE OPTA:	INGENIERO EN SOFTWARE
DIRECTOR:	MSC. TREJO ESPAÑA DIEGO JAVIER
ASESOR	MSC. ORTEGA BUSTAMANTE COSME MACARTHUR

CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 31 días, del mes de marzo de 2026

EL AUTOR:

Firmado electrónicamente por:
KEVIN ANDRES
GUACANES DIAZ
Validar únicamente con FirmaEC

.....

Sr. Guacanes Diaz Kevin Andres

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN

Ibarra, 31 de marzo de 2026

Por medio del presente yo MSc. Diego Javier Trejo España, certifico que el Sr. Guacanes Diaz Kevin Andres portador de la cedula de identidad número 100404717-9, ha trabajado en el desarrollo del proyecto de grado "**Implantación de una arquitectura de integración de sistemas mediante microservicios y servicios distribuidos para la optimización de los procesos de supervisión y regulación de construcciones en la provincia de Imbabura**", previo a la obtención del Título de Ingeniero en Software realizado con interés profesional y responsabilidad que certifico con honor de verdad.

Es todo en cuanto puedo certificar a la verdad

Atentamente

Firmado electrónicamente por:
DIEGO JAVIER TREJO ESPAÑA
Validar únicamente con FirmaEC

……

MSc. Diego Javier Trejo España

C.C.: 1002149290

INNOVA-EP
EMPRESA PÚBLICA MUNICIPAL DE HÁBITAT DESARROLLO URBANO SOSTENIBLE Y VIVIENDA

ALCALDÍA IBARRA

Ibarra, 27 de marzo de 2026

Ing. Cosme MacArthur Ortega Bustamante, MSc.
Coordinador CISIC-SW
Universidad Técnica del Norte

De mi consideración.

Por medio del presente, en mi calidad de **Gerente General** de la institución **INNOVA EP**, certifico que el estudiante **Guacanes Diaz Kevin Andres**, portador de la cédula de ciudadanía N.º **1004047179**, ha culminado satisfactoriamente el proceso de desarrollo de su trabajo de integración curricular titulado "**Implantación de una arquitectura de integración de sistemas mediante microservicios y servicios distribuidos para la optimización de los procesos de supervisión y regulación de construcciones en la provincia de Imbabura**".

Se informa que el sistema se encuentra operativo, funcionando con satisfacción desde el mes de enero del año 2026 constituyendo una herramienta de apoyo significativa para la modernización de nuestros procesos.

Por lo tanto, se deja constancia del cumplimiento del trabajo por parte del estudiante, para los fines académicos consiguientes ante la Universidad Técnica del Norte.

Atentamente,

MSc. Katya Lorena Bastidas Burbano
GERENTE GENERAL - INNOVA EP

DEDICATORIA

A Dios, fuente eterna de sabiduría, luz y fortaleza, con profundo agradecimiento por su guía inquebrantable a lo largo de este camino de conocimiento y esfuerzo.

Dedico este trabajo con todo mi amor a mis padres, Wilson Guacanes y Gloria Díaz. A ustedes, que muchas veces dejaron sus propios sueños a un lado para cumplir los míos. Gracias por enseñarme que la honestidad y el trabajo duro son el único camino al éxito. Papá, gracias por tu fortaleza incansable; Mamá, gracias por tu amor incondicional que siempre me sostuvo. Este título no lleva solo mi nombre, lleva grabada cada una de sus luchas y sacrificios.

A mi hermana, Dayana Guacanes, gracias por ser mi compañera, por escucharme y por creer en mí siempre.

A mis queridos familiares, gracias por creer en mí. Cada gesto de apoyo y cada consejo recibido fueron piezas clave para completar este rompecabezas.

A mis amigos, gracias por la paciencia, por esperarme y por empujarme a seguir cuando las fuerzas no me daban. Ustedes hicieron que el trayecto fuera tan memorable como la meta misma.

Guacanes Diaz Kevin Andres

AGRADECIMIENTO

Agradezco a INNOVA EP, al Colegio de Arquitectos de Imbabura y al Colegio de Ingenieros Civiles de Imbabura, gracias por darme la oportunidad y de facilitarme las herramientas necesarias para realizar mi proyecto de titulación con ustedes.

A mi tutor, Ing. Diego Trejo, por su acertada guía, su paciencia y por compartir sus conocimientos conmigo a lo largo de este proceso. Sus correcciones y consejos fueron fundamentales para darle el rumbo correcto a este trabajo.

A mi asesor, Ing. Cosme Ortega, por su valioso tiempo y disposición para resolver mis dudas, aportando su experiencia y visión técnica para mejorar la calidad de este proyecto.

Guacanes Diaz Kevin Andres

RESUMEN

La gestión de información en las instituciones públicas CAEI, CICI e INNOVA EP presentaba limitaciones críticas debido al uso de sistemas aislados, lo cual dificultaba la toma de decisiones, la transparencia operativa institucional y el riesgo de construcciones mal realizadas. Ante esta problemática, el presente estudio tuvo como objetivo general diseñar e implementar una arquitectura de microservicios para interconectar estas entidades, permitiendo un flujo de datos centralizado y eficiente. La metodología empleada consistió en la conexión de servicios independientes bajo el framework Laravel y una interfaz en React, orquestados mediante contenedores Docker para asegurar la escalabilidad, se aplicó el modelo de DeLone y McLean para la evaluación técnica y de usuario. Los resultados más relevantes demuestran una favorabilidad del 100% en las dimensiones de calidad de la información e impactos netos, junto con un nivel de satisfacción del usuario del 97.33%. Como conclusión principal, se establece que la arquitectura distribuida garantiza la alta disponibilidad y la integridad de los datos, transformando silos informáticos en un ecosistema integrado que optimiza significativamente la gestión operativa y el análisis estratégico institucional.

Palabras clave: Arquitectura de microservicios, Interconexión institucional, Scrum, Gestión de proyectos, Centralización, Usabilidad.

ABSTRACT

Information management in public institutions CAEI, CICI, and INNOVA EP had critical limitations due to the use of isolated systems, which hindered decision-making, institutional operational transparency, and the risk of poorly constructed buildings. Given this problem, the overall objective of this study was to design and implement a microservices architecture to interconnect these entities, allowing for centralized and efficient data flow. The methodology employed consisted of connecting independent services under the Laravel framework and a React interface, orchestrated using Docker containers to ensure scalability. The DeLone and McLean model was also applied for technical and user evaluation. The most relevant results show 100% favorability in the dimensions of information quality and net impacts, along with a user satisfaction level of 97.33%. As a main conclusion, it is established that distributed architecture guarantees high availability and data integrity, transforming IT silos into an integrated ecosystem that significantly optimizes operational management and institutional strategic analysis.

Keywords: Microservices architecture, Institutional interconnection, Scrum, Project management, Centralization, Usability.

LISTA DE SIGLAS

AMQP: Advanced Message Queuing Protocol (Protocolo Avanzado de Colas de Mensajes)

API: Interfaz de Programación de Aplicaciones

CRUD: Create, Read, Update, Delete (Crear, Leer, Actualizar y Eliminar)

DOM: Document Object Model (Modelo de Objetos del Documento)

HTML: HyperText Markup Language (Lenguaje de Marcado de Hipertexto)

HTTP: HyperText Transfer Protocol (Protocolo de Transferencia de Hipertexto)

JSON: JavaScript Object Notation (Notación de Objetos de JavaScript)

ORM: Object-Relational Mapping (Mapeo Objeto-Relacional)

URL: Uniform Resource Locator (Localizador Uniforme de Recursos)

XML: eXtensible Markup Language (Lenguaje de Marcado Extensible)

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Problema de investigación

Dentro de la provincia de Imbabura los procesos de supervisión y regulación de construcciones tiene un problema crítico debido a la falta de conexión entre las instituciones responsables. Esta desconexión digital no es solo un inconveniente administrativo, sino que también conlleva a un alto riesgo para la seguridad pública. Actualmente las edificaciones de Imbabura se realizan con tres entidades que no poseen comunicación directa si, estas instituciones son: El Colegio de Arquitectos de Imbabura (CAEI), Colegio de Ingenieros Civiles de Imbabura (CICI) y la Empresa pública municipal de hábitat, desarrollo urbano sostenible y vivienda (INNOVA EP).

La ausencia de un puente digital entre estas entidades ha creado una alarmante estadística en los planos arquitectónicos aprobados por el Colegio de Arquitectos de Imbabura (CAEI), ya que no llegan a su revisión correspondiente al Colegio de Ingenieros Civiles de Imbabura (CICI) para la revisión estructural, esto significa que las edificaciones en la provincia podrían estar construyéndose sin una evaluación adecuada. Esta desconexión no solo genera riesgos estructurales sino también caso de fenómenos naturales como sismos o lluvias intensas estas construcciones inadecuadamente supervisadas podrían sufrir daños severos poniendo en riesgo vidas humanas.

Para abordar esta problemática es necesario diseñar un entorno de servidor que funcione como un director de orquesta coordinando el trabajo [1], Un servidor entendido como una máquina con software especializado, permite alojar aplicaciones y gestionar solicitudes de diferentes clientes en red así facilitando la transferencia segura de información mediante protocolos como HTTP, TCP/IP [2]. Este servidor debe prestarse para garantizar la comunicación entre las aplicaciones de las instituciones involucradas. Con esta solución integrada Imbabura podrá transformar su sistema de supervisión de construcciones asegurando edificaciones más seguras y mejor reguladas para todos sus habitantes siendo el ente supervisor la Empresa pública municipal de hábitat, desarrollo urbano sostenible y vivienda (INNOVA EP).

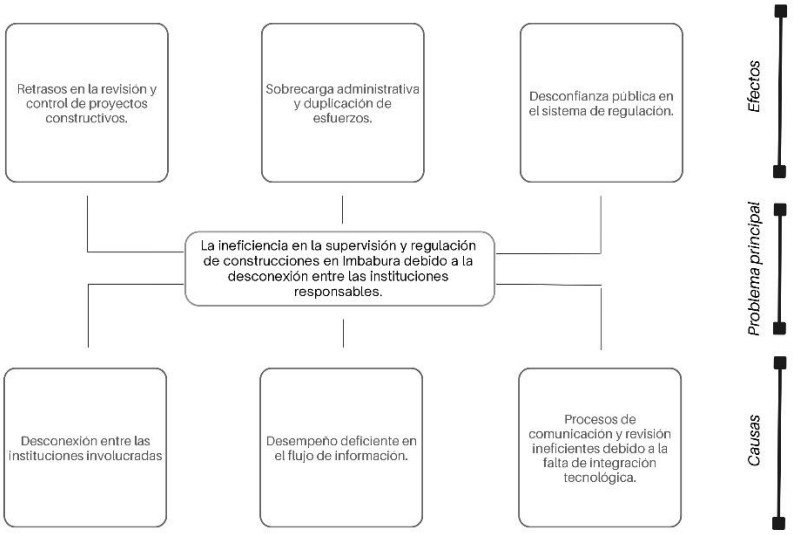


Figura 1. Árbol de problemas

Justificación

La implementación de una arquitectura de integración de sistemas en las instituciones CAEI, CICI e INNOVA EP de Imbabura busca resolver la falta de comunicación entre ellas, que actualmente permite que solo el 55% de tramites lleguen a realizar su proceso en la Empresa pública municipal de hábitat, desarrollo urbano sostenible y vivienda (INNOVA EP). Esta desconexión aumenta los riesgos en la seguridad pública, ya que las construcciones pueden carecer de evaluaciones estructurales adecuadas.

Además, la aplicación de Tecnologías de la Información y Comunicación (TIC) en instituciones públicas ha contribuido al mejoramiento de sus procesos administrativos, facilitando la interacción con la ciudadanía y promoviendo la transparencia en la gestión [3]. El estudio está vinculado con la Política 9.3 del Plan de Creación de Oportunidades, que busca Fortalecer la seguridad y protección del Sistema Nacional de Rehabilitación Social desde la prevención, disuasión, control, contención [4], y respuesta a eventos adversos en situaciones de crisis, al implementar herramientas tecnológicas que potencien la gestión institucional. Así mismo se alinea con el ODS 9 (Construir infraestructuras resilientes, promover la industrialización inclusiva y sostenible y fomentar la innovación), para impulsar el crecimiento económico y la estabilidad social [5].

Los beneficiarios directos son:

- Ciudadanía.
- Entidades de control municipal.

Beneficiarios indirectos:

- Personal técnico.
- Personal administrativo.
- Personal financiero.

Justificación Tecnológica:

Desde una perspectiva técnica este proyecto representa una solución para la transformación digital aplicada a un sector público, utilizando principios de sostenibilidad digital. Tal como se menciona en estudios previos, la transformación digital en entornos públicos puede actuar como un catalizador para mejorar la eficiencia operativa y la capacidad de respuesta ante situaciones críticas. La implementación de microservicios y servicios distribuidos proporciona una infraestructura flexible y escalable, adaptada a las necesidades de las instituciones y capaz de soportar un crecimiento futuro en la demanda de información y procesamiento.

Justificación Económica:

Económicamente, la implementación de esta solución tecnológica permitirá una significativa reducción de costos operativos. La centralización de los servicios y la mejora en la comunicación entre las instituciones optimizará los procesos, reduciendo tiempos de espera y duplicación de esfuerzos. De acuerdo con estudios previos, la adopción de tecnologías digitales en instituciones públicas puede reducir considerablemente los costos asociados con la gestión de proyectos y el intercambio de información, mejorando la eficiencia general de la administración pública.

Justificación Social y Educativa:

Este proyecto también tiene una dimensión social y educativa importante, ya que promueve la inclusión digital en el ámbito de la gestión pública. La implementación de infraestructura tecnológica avanzada mejorará la interacción entre las instituciones y la ciudadanía, promoviendo así la transparencia y facilitando el acceso a información confiable en tiempo real. En el contexto de la Educación Social del siglo XXI, el uso de

Tecnologías de la Información y Comunicación (TIC) en la gestión pública no solo mejora la eficiencia, sino que también contribuye a la formación de competencias digitales dentro de las entidades gubernamentales, fomentando una cultura digital participativa que involucra a todos los actores sociales en la toma de decisiones.

Objetivos

Objetivo General

Implantar una arquitectura de integración de sistemas mediante microservicios y servicios distribuidos para la optimización de los procesos de supervisión y regulación de construcciones en la provincia de Imbabura.

Objetivos Específicos

- Revisar la literatura existente sobre la implementación de arquitecturas de microservicios en entornos institucionales.
- Analizar los requerimientos de infraestructura necesaria para realizar la interconexión de los servicios del servidor necesarios para soportar la comunicación entre las aplicaciones institucionales.
- Evaluar el rendimiento del servidor configurado mediante pruebas de carga y stress, asegurando su capacidad para soportar el volumen de transacciones requerido por las tres instituciones.

CAPÍTULO II

MARCO TEÓRICO

En el siguiente capítulo se establece el marco teórico fundamental para la comprensión y el desarrollo de una arquitectura de integración para sistemas basada en microservicios y servicios distribuidos para lograr el objetivo de optimizar los procesos de supervisión y regulación de construcciones en la provincia de Imbabura. En este capítulo se abordarán conceptos fundamentales para implantación de arquitecturas, y también la implementación en entornos institucionales y su impacto en la optimización de procesos. De la misma forma se explorarán las normativas y estándares aplicables a sistemas públicos, el rol de las Tecnologías de la Información y Comunicación en la gestión pública y las metodologías también con las herramientas clave para la implementación de soluciones tecnológicas para lograr el objetivo de cumplir con el objetivo de optimizar procesos.

1.1 Microservicios y servicios distribuidos

1.1.1 Conceptos de microservicios y servicios distribuidos

Los microservicios representan a un estilo arquitectónico moderno para el desarrollo del software esto caracterizado por la descomposición de una aplicación en un conjunto de servicios más pequeños, autónomos y acoplados de manera correcta [6]. Cada microservicio opera su propio proceso y se comunica los demás mediante o usando interfaces correctamente definidas, comúnmente APIs (Application Programming Interfaces). Esta modularidad permite que equipos pequeños e independientes sean responsables de cada servicio lo que incrementa la agilidad además de que reduce los tiempos de ciclo de desarrollo [7].

Una característica que hace que los microservicios sean diferentes es su independencia en el despliegue, lo que posibilita la actualización dependiente de cada servicio sin afectar el funcionamiento del resto de la aplicación y facilita la adopción de las nuevas tecnologías y lenguajes emergentes, esto según las necesidades específicas de

cada componente, estas se distinguen principalmente dos tipos: microservicios sin estado *stateless*, que son las que no retienen información de sesión entre solicitudes y son bloques constructivos esenciales en los sistemas distribuidos y el otro tipo son microservicios con estado *statefull* estos son las que sí mantienen información de sesión o datos en su código, aunque son las menos frecuentes o utilizadas [8].

Arquitectura de microservicios

Esta arquitectura representa un enfoque que se implementa de manera continua donde se examinan los componentes de la aplicación para descomponerlos en servicios independientes con responsabilidades específicas. De esta forma se logra un mayor modularidad y se impulsa el mejoramiento del rendimiento, ya que cada servicio puede optimizarse y escalarse de manera individual [9].

Servicios distribuidos

Un sistema de servicios distribuidos es un conjunto de componentes de software coordinados que operan en múltiples nodos de red, comunicándose entre sí para lograr objetivos comunes. Estos sistemas proporcionan capacidades de procesamiento, almacenamiento y comunicación distribuidas a través de diferentes ubicaciones físicas o lógicas, simplificando la gestión de aplicaciones empresariales complejas mediante arquitecturas modulares [10].

Gestión de servicios distribuidos

La gestión de servicios distribuidos es un proceso para establecer la comunicación y coordinación entre los diferentes componentes del sistema, monitorear el estado de salud de cada servicio a través de métricas y poder evaluar el rendimiento general. Esto ayuda a garantizar la disponibilidad del sistema y el alineamiento con los múltiples objetivos arquitectónicos que se han planteado [11].

Características

Los microservicios presentan múltiples características que los distinguen de las arquitecturas monolíticas tradicionales:

Tabla 1. Características de los indicadores.

Concepto / Término	Definición y Características
Independencia	Cada servicio puede desarrollarse, desplegarse y escalarse de manera autónoma sin afectar otros componentes del sistema.
Modularidad	Garantiza que cada servicio tenga una responsabilidad específica y bien definida dentro del sistema.
Descentralización	Permite que diferentes servicios utilicen tecnologías, lenguajes y bases de datos según sus necesidades específicas.
Resiliencia	El fallo de un servicio no compromete el funcionamiento del sistema completo, permitiendo degradación controlada.
Escalabilidad	Los servicios individuales pueden escalarse horizontalmente de manera independiente según la demanda.
Comunicación ligera	Se utilizan protocolos estándar como HTTP/REST o sistemas de mensajería para la interacción entre servicios.

1.1.2 Implementación de microservicios en entornos institucionales

La implementación de microservicios se ha expandido en el mundo significativamente a diversos entornos, incluyendo con esta expansión a instituciones académicas y gubernamentales, con el propósito de modernizar sistemas legados y mejorar la eficiencia operativa [12].

Los beneficios al realizar una implementación de microservicios en contextos institucionales son amplios, pero comúnmente se puede observar lo siguiente:

Tabla 2. Características en implementación de microservicios.

Característica	Definición
Mejora de la Gestión y Control	Permiten una gestión más específica y un control en tiempo real de los procesos que se manejan en los diferentes entornos.
Adaptación a Estándares de Calidad	La arquitectura facilita el cumplimiento de estándares de calidad con ello permitir la mejora continua y modular de los

	componentes del sistema, lo que es muy importante para la mejora operativa.
Flexibilidad y Escalabilidad	La capacidad de escalar solo los servicios que lo necesitan y de integrar nuevas tecnologías rápidamente es importante para las instituciones con necesidades que cambian seguidamente y también para brindar un crecimiento constante de usuarios.
Productividad de Equipos	Se puede alinear a la arquitectura con equipos más pequeños y enfocados específicamente en capacidades específicas de la empresa, con ello se mejora la productividad y se reduce el número de personas trabajando en una única base de código.
Descentralización	Hace posible un control y almacenamiento de datos que no sea centralizado, así cada servicio gestiona su lógica de negocio y acceso a datos de forma aislada con ello evita el acoplamiento y permite la evolución independiente.

1.1.3 Optimización de procesos mediante implementación de arquitecturas de microservicios.

La arquitectura de microservicios optimiza los procesos de desarrollo y operación al hacer más fácil la escalabilidad y el despliegue en el desarrollo. Al poder escalar independientemente cada servicio según su demanda, estos así aprovechan mejor los recursos del sistema. Así se reduce tiempos de respuesta en tareas críticas y mejora la tolerancia a los fallos [13].

Al realizar la mezcla de microservicios con metodologías de entrega continua las actualizaciones se automatizan y se aceleran. En general esta práctica nos muestra que migrar de sistemas monolíticos y distribuidos a microservicios puede resolver cuellos de botella en la fase de desarrollo y despliegue de software. El resultado de esto es una mayor eficiencia operativa además de que se introducen nuevas funcionalidades con menor tiempo de inactividad, de igual manera, se simplifica la mantención y con esto se logra mejorar la experiencia del usuario final [14].

1.2 Normativas y estándares para la implementación de sistemas públicos

1.2.1 Normas ISO/IEC para arquitecturas de microservicios

Existen normas internacionales que están del lado del diseño de arquitecturas de software, incluida en ella la orientación hacia microservicios. Como por ejemplo la especificación técnica ISO/IEC 23167:2020 (Tecnologías de información – Computación en la nube) [15] identifica la arquitectura de microservicios como una tecnología clave en entornos de nube moderna. Asimismo, los estándares generales de arquitectura de software como la ISO/IEC/IEEE 42010 que realiza la descripción de la arquitectura, la norma ISO/IEC 25010 puede aplicarse a soluciones basadas en microservicios. De tal manera no existe una norma ISO específica que sólo sirva para microservicios, los marcos ISO/IEC proporcionan pautas para poder construir sistemas distribuidos que sean cada vez más escalables.

En la norma ISO/IEC 27001 se establecen los requisitos para un Sistema de Gestión de la Seguridad de la Información. Según la ISO 27001 el objetivo es proteger fuertemente la **confidencialidad, integridad y la disponibilidad** de la información mediante un enfoque que este basado en el riesgo para poder mitigarlo. Para las arquitecturas basadas en microservicios esto implica diseñar todos controles de seguridad en cada uno de los componentes y en las comunicaciones que existen entre ellos.

Por otro lado, la norma ISO/IEC/IEEE 42010 define un marco para la descripción formal de las arquitecturas existentes en los sistemas. Según ISO 42010, una descripción arquitectónica debe documentar todos los conceptos que son fundamentales del sistema (elementos, relaciones y principios del diseño) [16]. En otras palabras, ISO 42010 promueve la creación de documentos y vistas arquitectónicas (modelos de negocio, componentes, flujos, despliegue, etc.) que nos permitan entender y comunicar la arquitectura global. Para un sistema de microservicios es muy importante seguir con este enfoque: la norma incentiva la elaboración de diagramas de todos los servicios necesarios, contratos de API, vistas de interacción y de despliegue que muestren el diseño modular y distribuido. De esta forma se mantiene la coherencia y evolución controlada del sistema evitando desviaciones entre el diseño planificado y la implementación real del sistema. Como resumen de esto podemos entender que la norma ISO/IEC 42010 aporta una base metodológica para poder documentar ordenadamente la arquitectura alineando así los requerimientos de negocio con los componentes tecnológicos y facilitando la interoperabilidad entre los servicios.

En conjunto todos estos estándares ISO/IEC contribuyen a garantizar la calidad integral de una arquitectura que esté basada en microservicios: ISO/IEC 25010 define las cualidades de producto como: la calidad funcional, la confiabilidad, la eficiencia, la interoperabilidad, y otras importantes. La norma ISO/IEC 27001 establece un marco para la seguridad de la información y la protección de todos los que sean datos sensibles, en otro lugar la norma ISO/IEC 42010 nos asegura un diseño estructurado y bien documentado de la arquitectura de los sistemas. La adopción de estos estándares ayuda a que el sistema distribuido resultante sea robusto, seguro, mantenible y capaz de integrarse con los otros sistemas, cumpliendo así los objetivos de calidad y gobernabilidad.

1.2.2 Estándares de seguridad y comunicación en supervisión

Los sistemas públicos están atados a cumplir normas estrictas y de seguridad de la información. En Ecuador se ha implementado un Esquema Gubernamental de Seguridad de la Información que está basado en la norma NTE INEN ISO/IEC 27001, para los organismos del sector público [17]. El EGSI establece controles y procedimientos para garantizar la confidencialidad, la integridad y la disponibilidad de los datos sensibles en entidades [18]. En la práctica esto implica usar protocolos seguros de comunicación (por ejemplo, HTTPS/TLS para APIs web, VPNs y cifrado de datos) autenticar a los usuarios según los estándares reconocidos. Además, para la supervisión de construcciones, la interoperabilidad requiere protocolos estandarizados (como servicios web REST) y formatos de datos abiertos como los son: JSON, XML, entre otros, que faciliten el intercambio de información entre los sistemas de planificación urbana, catastros y portales de gestión de obras. Todo esto en conjunto con las autoridades reguladoras deben alinear sus soluciones TIC con las políticas nacionales de ciberseguridad (EGSI) y con estándares de comunicación interoperable para los sistemas públicos.

1.2.3 Requerimientos normativos para instituciones públicas en Ecuador

Las entidades vinculadas a la supervisión de construcciones en la provincia de Imbabura están sujetas a leyes y reglamentos específicos. El Colegio de Arquitectos (CAE/CAEI) se rige por la Ley de Ejercicio Profesional de la Arquitectura y su reglamento, que definen las atribuciones y obligaciones de los arquitectos en Ecuador [19]. De modo casi igual pasa para los colegios de ingenieros civiles se rigen por la Ley de Ejercicio Profesional

de la Ingeniería (Decreto Supremo 1300) y sus normativas asociadas. Por otra parte la empresa INNOVA E.P. es pública y creada por ordenanza municipal para gestionar el desarrollo urbano y de vivienda en Ibarra, esta empresa tiene competencias para planificar, construir y mantener la infraestructura urbana del cantón. En la práctica la empresa INNOVA-EP debe saber operar con respecto a la Ley Orgánica de Empresas Públicas y a las ordenanzas locales de construcción y urbanismo.

1.3 Tecnologías de la información y comunicación (TIC) en la gestión pública

1.3.1 Rol de las TIC en la regulación de construcciones.

Las TIC juegan un papel cada vez más importante en la gestión urbana y la regulación de construcciones. Por ejemplo, en los gobiernos locales implementan Sistemas de Información Geográfica para verificar el cumplimiento de zonas y planes urbanos, y portales web para la tramitación digital de permisos de obra. En el paradigma de "ciudad inteligente", las TIC interconectan y optimizan los servicios urbanos. Esto significa que, aplicadas a la supervisión de construcciones, las TIC pueden coordinar datos de catastros, censos de edificios e inspecciones, mejorando la transparencia y la eficiencia [20]. Herramientas como aplicaciones móviles para reportar anomalías o el análisis de datos de sensores urbanos permiten a las autoridades detectar todo rastro de irregularidades con rapidez. Así, las TIC facilitan que los reglamentos de construcción se apliquen de forma más precisa y siendo dinámica, reduciendo gestiones presenciales y acelerando la respuesta ante los incumplimientos.

1.3.2 Aplicación de TIC para la Integración de sistemas

La aplicación de TIC para la integración de sistemas representa la capacidad de conectar y combinar diferentes sistemas, aplicaciones, plataformas o tecnologías para que trabajen juntos de manera efectiva y se comuniquen entre sí. La integración tecnológica es la fusión de las herramientas que unen los dispositivos de última generación facilitando la interacción de los usuarios, propiciando mayor practicidad y confort al momento de utilizar cualquier servicio [21].

En el contexto institucional y gubernamental, la aplicación de TIC para la integración de sistemas permite compartir datos y procesos entre diferentes áreas

organizacionales, lo que puede mejorar la eficiencia, la productividad y la satisfacción del cliente. Se trata del proceso de incorporar a la perfección distintas tecnologías digitales en diversos aspectos de las operaciones, estrategias y cultura de una organización.

La integración de los sistemas públicos mediante TIC busca que diferentes plataformas intercambien información de forma transparente y muy segura. Por ejemplo, CAEI, CICI e INNOVA-EP podrían integrar sus bases de datos a través de servicios web y APIs que sean estándar para compartir expedientes de proyectos de construcción. En la práctica se emplean bus de servicios empresariales o gateways de API que orquestan las comunicaciones entre sistemas heterogéneos. Asimismo, el Gobierno Electrónico promueve el uso de portales únicos de datos abiertos y normativas que les permitan interoperabilidad como por ejemplo las basadas en estándares. El uso de TIC en integración de sistemas públicos implica adoptar protocolos y formatos comunes, de modo que la información sobre licencias, planos y permisos fluya y sea obtenido de manera segura entre entidades. Este enfoque integrado agilizaría los procesos de supervisión, evitando duplicidad de trámites y mejorando la trazabilidad de las obras.

Objetivos principales

La aplicación de TIC para la integración de sistemas busca alcanzar los siguientes objetivos fundamentales:

1. **Optimización de procesos**: La integración tecnológica agiliza procesos y automatiza las tareas más repetitivas, lo que lleva a la organización a una mayor eficiencia operativa y productividad.
2. **Interoperabilidad**: Conectar diferentes aplicaciones de software para que funcionen juntas y compartan datos de manera fluida, eliminando silos de información.
3. **Toma de decisiones informada**: Las tecnologías integradas permiten recopilar, analizar e interpretar datos procedentes de diversas fuente, facilitando decisiones basadas en información precisa y actualizada.
4. **Mejora en la prestación de servicios**: La integración tecnológica permite a las organizaciones ofrecer experiencias personalizadas al aprovechar el análisis de datos y la automatización.

Los principios claves para indicadores de gestión son:

Tabla 3. Características de la aplicación de TIC para integración de sistemas.

Objetivo	Descripción
Conectividad	Permite que los usuarios con un computador y conexión a internet puedan realizar múltiples actividades como navegar, trabajar, estudiar, adquirir servicios y comunicarse.
Escalabilidad	Las tecnologías integradas ofrecen escalabilidad para adaptarse al crecimiento empresarial y a las necesidades cambiantes sin inversiones significativas en infraestructura.
Flexibilidad	No se requiere de grandes estructuras tecnológicas para funcionar, siendo accesible tanto para grandes organizaciones como para pequeñas entidades.
Automatización	Las actividades manuales que consumen mucho tiempo se sustituyen por flujos de trabajo automatizados, liberando recursos para tareas más estratégicas.
Tiempo real	La integración tecnológica proporciona visibilidad en tiempo real de las operaciones, lo que permite intervenir a tiempo y resolver problemas de forma proactiva.
Seguridad	Integrar sistemas de seguridad para garantizar que los datos y los sistemas estén protegidos de amenazas cibernéticas.
Innovación continua	Constantemente innova con mejoras y nuevos cambios, buscando la practicidad y el confort para las personas.

La integración de soluciones avanzadas de las TIC en infraestructuras críticas no solo mejora su rendimiento, sino que también garantiza una mayor resiliencia y capacidad. Las organizaciones que integran con éxito la tecnología obtienen una ventaja competitiva, pueden adaptarse y crear propuestas de valor únicas.

De esta manera la integración de sistemas de información, enlaces de comunicación en aplicaciones otorgan competencias para ser cada vez más eficiente y productiva, aspecto crucial para optimizar los procesos.

1.3.3 Desafíos de la Interoperabilidad Gubernamental y los Silos de Información

La modernización que existe en el Estado y la optimización de todos los servicios ciudadanos dependen de manera crítica de la capacidad de las entidades públicas para intercambiar información de manera fluida, segura y que sea coherente, aunque a veces se imposibilita por la estructura tradicional de la administración pública, esto presenta barreras que ponen como obstáculo al flujo de los procesos, haciendo que sean demorados. Esta sección establece como problema fundamental que justifica la adopción de una arquitectura de software moderna, argumentando que la ineficiencia y la lentitud en la gestión pública no son meramente problemas tecnológicos ni superficiales, sino que también consecuencias directas de un paradigma organizacional y técnico basado en silos de información. La superación de estos silos requiere más que simples conectores; exige un enfoque arquitectónico que aborde las múltiples capas de la interoperabilidad.

El concepto que existe en las palabras "silo de información" describe a un fenómeno en las organizaciones, particularmente pronunciado mucho en el sector público donde los datos, procesos y tecnologías operan de manera aislada dentro de los confines de una agencia, departamento o unidad que es específica [22]. Estos silos no son una consecuencia de algo accidental, sino que son el resultado de factores estructurales profundamente arraigados. Las agencias gubernamentales casi siempre funcionan con presupuestos independientes, jerarquías rígidas y objetivos departamentales que no siempre están alineados con una visión del servicio ciudadano. A esto se suma la prevalencia de "sistemas heredados".

La solución al problema de los silos de información no reside únicamente en establecer conexiones físicas o lógicas entre sistemas. La verdadera interoperabilidad, aquella que permite una colaboración gubernamental efectiva y la prestación de servicios ciudadanos integrados, es un concepto multidimensional. La literatura especializada incluyendo marcos de referencia como los propuestos por la Comisión Europea y adoptados por consultoras globales, distingue cuatro capas fundamentales de interoperabilidad que son legal, organizacional, semántica y tecnica. Ignorar cualquiera

de estas capas conduce a soluciones frágiles que pueden llegar a quedar incompletas [23].

Interoperabilidad Técnica: Es la capa que se refiere a la capacidad de los sistemas de la información puedan soportar poder comunicarse entre ellos de manera segura y eficiente, esto hace referencia a la definición de interfaces, protocolos de comunicación y formatos de datos comunes que permitan el intercambio de la información [24]. Pero hay que tomar en cuenta la interoperabilidad técnica que sola no puede garantizar que los bits y bytes puedan viajar de un sistema a otro de manera correcta.

Interoperabilidad Semántica: Esta es una capa muy importante en la cual se ocupa del significado de los datos que son intercambiados. Como objetivo principal de esta es asegurar que los sistemas no solo se comuniquen, sino que se comuniquen de manera correcta sin fallos. Esto importa mucho ya que el formato y el significado preciso de la información intercambiada se mantengan y comprendan de manera no abstracta por todas las partes involucradas. La interoperabilidad semántica es importante para evitar equivocaciones y errores de interpretación y esto justifica directamente la necesidad de Arquitecturas Orientadas a Servicios y más específicamente de los microservicios que se basan en la exposición de las funcionalidades a través de Interfaces de Programación de Aplicaciones con contratos bien definidos y esquemas de datos como por ejemplo podría ser Swagger.

Interoperabilidad Organizacional: Este apartado también es importante porque en el se toma mucha presencia a la coordinación de los procesos de negocio y las responsabilidades institucionales que se comunican mediante las fronteras de una no tan solo de una entidad. Esto se enfoca en documentar y alinear los flujos de trabajo, también en definir los roles de cada actor y establecer acuerdos de nivel de servicio para garantizar que la colaboración interinstitucional sea fluida y predecible, pero a su vez confiable. Como ejemplo para automatizar la emisión de un permiso de construcción no tan solo basta con que los sistemas se conecten; las entidades involucradas deben acordar el proceso, los plazos y las responsabilidades de cada una en el flujo de trabajo digital.

Interoperabilidad Legal: Esta capa final es de suma importancia tomarla en cuenta porque en ella se refiere al marco normativo que habilita y regula el intercambio de la información entre las instituciones públicas. Involucra la identificación de barreras y oportunidades legales que existen, la creación de convenios interinstitucionales y la

garantía que se da al realizar el intercambio de datos así cumple con las leyes de protección de datos personales, privacidad y seguridad de la información

1.4 Metodologías, herramientas y tecnologías

1.4.1 Metodología Ágil SCRUM

SCRUM es un marco que hace del trabajo que sea iterativo y además incremental que surgió en la década de 1990 y se ha ganado una amplia aceptación en la industria en especial del software. Su enfoque principal importa mucho en el sentido de la entrega de valor al cliente y en empoderar al equipo para lograr la máxima eficiencia dentro de un esquema que sirva de ayuda para la mejora continua [25].

Esta metodología se caracteriza por ser adaptable en lugar de ser predictiva, además de que está orientado a las personas más que a los procesos en sí, y por utilizar un modelo de construcción incremental basado en iteraciones y revisiones como un bucle. Una de sus fortalezas es su capacidad para gestionar la escasez de información al inicio del desarrollo, priorizando la capacidad del equipo para entregar rápidamente y responder a requisitos emergentes y cambios constantes que suelen darse.

Tabla 4. Características principales de la metodología Scrum.

Caracterisitica	Descripción
Sprints	El desarrollo se organiza en iteraciones de duración fija, generalmente de 30 días, conocidas como Sprints. Al final de cada Sprint se produce un incremento ejecutable del producto que se presenta al cliente.
Requisitos cambiantes	SCRUM es particularmente adecuado para proyectos donde los requisitos están sujetos a cambios frecuentes, una realidad común en el desarrollo de software moderno.
Reuniones regulares	La metodología incluye reuniones diarias breves (Daily Scrum) de aproximadamente 15 minutos para coordinar actividades y gestionar el progreso, así como reuniones de revisión y retrospectiva al final de cada Sprint.

Equipos autoorganizados	Los equipos de desarrollo tienen la autonomía para determinar cómo alcanzar los objetivos y trabajan de forma colaborativa, fomentando la responsabilidad y la eficiencia.
Entrega incremental	Permite entregas frecuentes y tempranas de valor al cliente, facilitando la validación continua del producto desarrollado.
Transparencia	Todos los aspectos del proceso son visibles para quienes son responsables del resultado, promoviendo la confianza y la colaboración.

Fases de la metodología SCRUM

La implementación de SCRUM sigue un ciclo estructurado en fases claramente definidas que facilitan la gestión efectiva del desarrollo:

Fase de Planificación: En esta etapa inicial se define el equipo y las herramientas necesarias para el desarrollo. Se crea el Product Backlog, que es una lista priorizada de requisitos o "historias de usuario" que son estimadas correspondientemente con el esfuerzo requerido para cada una. También se define la arquitectura inicial del producto, estableciendo las bases tecnológicas y estructurales del sistema a desarrollar.

Fase de Desarrollo (Sprints): Durante esta fase ágil, el sistema se construye mediante Sprints consecutivos. Cada Sprint abarca las actividades de diseño, implementación y pruebas del sistema, culminando en un incremento de software funcional que puede ser potencialmente desplegado. Esta fase es cíclica y se repite hasta completar todos los requisitos establecidos.

Fase de Finalización: Una vez que todos los requisitos han sido implementados y el Product Backlog está vacío, esta fase incluye la integración final del sistema, pruebas exhaustivas de aceptación y la documentación completa del sistema desarrollado, asegurando su mantenibilidad futura.

Roles en SCRUM

La metodología SCRUM define roles específicos con responsabilidades claramente delimitadas para asegurar el éxito del proyecto:

Product Owner: Es el responsable de la visión del producto y de mantener el Product Backlog priorizado, asegurando que el equipo trabaje en lo que genera mayor valor para el cliente y la organización. Este rol actúa como enlace entre los stakeholders y el equipo de desarrollo.

Scrum Master: Facilita el proceso SCRUM, elimina impedimentos que puedan afectar al equipo y asegura que se sigan las prácticas y valores ágiles. No es un gerente tradicional, sino un líder servicial que apoya al equipo en su autoorganización.

Development Team: Es el equipo multifuncional y autoorganizado responsable de entregar el incremento de producto al final de cada Sprint. Sus miembros poseen todas las habilidades necesarias para convertir el Product Backlog en incrementos de funcionalidad utilizables.

Beneficios de SCRUM para arquitecturas de microservicios

La adopción de SCRUM en el desarrollo de arquitecturas basadas en microservicios y sistemas distribuidos ofrece múltiples beneficios significativos:

- **Agilización del desarrollo:** Acelera el proceso de desarrollo y mantenimiento mediante entregas frecuentes y tempranas, permitiendo la validación continua de los servicios implementados.
- **Mejora en la comprensión del producto:** Incrementa la comprensión del sistema de software a través de la colaboración y comunicación constante del equipo y con el cliente, esencial en arquitecturas complejas.
- **Flexibilidad adaptativa:** Aumenta la flexibilidad para incorporar nuevos requisitos y prioridades a lo largo del ciclo de vida del proyecto, característica crucial para sistemas que evolucionan continuamente.
- **Reducción de costos:** Disminuye los costos asociados con el proceso de desarrollo al optimizar los recursos y minimizar el retrabajo, aprovechando la detección temprana de problemas.

1.4.2 Modelo de evaluación de DeLone y McLean.

El modelo de DeLone y McLean surge como una brújula sofisticada en el complejo universo de los sistemas de información, donde determinar el "éxito" traspasa las métricas tradicionales de funcionamiento técnico. Esta metodología ha experimentado un cambio constante través de las décadas. Nació en 1992 como una propuesta revolucionaria, se reinventó en 2003 con mayor precisión, y alcanzó su madurez en 2016 cuando los creadores reemplazaron el concepto de "Beneficios Netos" por "Impactos Netos" un cambio aparentemente sutil pero profundamente significativo que refleja una comprensión más rica de cómo la tecnología permea las organizaciones. Además, tejieron conexiones más elegantes entre las intenciones de uso, la experiencia práctica del usuario y su satisfacción, creando un ecosistema conceptual más cohesivo.

El modelo D&M se compone de seis dimensiones interdependientes:

1. **Calidad del Sistema (*System Quality*):** Se refiere a las características que son las deseables del propio sistema de información, incluye los aspectos como la facilidad de uso, la flexibilidad que tiene el sistema, su fiabilidad, la facilidad de aprendizaje, que tenga una interfaz intuitiva, la sofisticación de sus funcionalidades y los tiempos de respuesta que existan. En el contexto de un proyecto, se evaluaría si el sistema es funcional, estable y fácil de operar.

2. **Calidad de la Información (*Information Quality*):** Aquí se evalúan las características esperables y deseadas de la salida del sistema, como por ejemplo la calidad de la información que produce. Esto es importante para su relevancia, comprensibilidad, precisión, concisión, completitud, actualidad y usabilidad de los informes, datos y páginas web generados por el sistema.

3. **Calidad del Servicio (*Service Quality*):** Esta dimensión abraza la dimensión humana del soporte tecnológico, evaluando la calidad del acompañamiento que los usuarios reciben del personal especializado. Como un director de orquesta que coordina múltiples instrumentos, incluye la capacidad de respuesta, precisión, fiabilidad, competencia técnica y empatía del personal de tecnologías de la información. Es el puente empático entre la complejidad tecnológica y la necesidad humana.

4. **Uso / Intención de Uso (*Use / Intention to Use*):** Esto mide el grado y la forma en que los empleados y clientes utilizan las capacidades del sistema de información. Esto puede cuantificarse por la cantidad, frecuencia, naturaleza, idoneidad, extensión y

propósito del uso. Para sistemas de uso obligatorio, la "Intención de Uso" puede ser una medida alternativa para evaluar la actitud del usuario hacia el sistema, mientras que para uso voluntario, el "Uso" real es más apropiado.

5. **Satisfacción del Usuario (*User Satisfaction*):** Aquí se evalúa el nivel de satisfacción global que experimentan los usuarios con el ecosistema informacional completo. Como un espejo que refleja la armonía entre expectativas y realidad, esta satisfacción es influenciada directamente por la trinidad de calidades: sistema, información y servicio. Es la síntesis emocional de toda la experiencia tecnológica.
6. **Impactos Netos (*Net Impacts*):** Por este lado se mide la medida en que los sistemas de información contribuyen al éxito de individuos, grupos, organizaciones, industrias y naciones.

Las interrelaciones del modelo constituyen un contexto conceptual sofisticado. La trinidad de calidades (Sistema, Información, Servicio) ejerce una influencia sobre el Uso/Intención de Uso y la Satisfacción del Usuario. A su vez, estos elementos confluyen para generar los Impactos Netos, creando un ciclo de retroalimentación que permite la mejora continua. Los Impactos Netos no son un destino final, sino que retroalimentan y reconfiguran la Satisfacción del Usuario y el Uso futuro, generando un ecosistema dinámico de evolución permanente.

La versatilidad del modelo D&M se ha demostrado en múltiples contextos. Un estudio paradigmático evaluó el Sistema Integrado de Información Universitaria de la Universidad Técnica del Norte desde la perspectiva estudiantil, aplicando las seis dimensiones como un prisma analítico. Los hallazgos no solo confirmaron la robustez del modelo, sino que revelaron una trayectoria positiva en la percepción estudiantil del SIIU a través del tiempo, con mejoras significativas en todas las dimensiones. Sin embargo, la calidad del servicio emergió como un terreno fértil para futuras optimizaciones [26].

La implementación del Modelo de DeLone y McLean representa una decisión metodológica de extraordinaria solidez para la evaluación de sistemas de microservicios. Este enfoque trasciende las métricas técnicas tradicionales para incorporar la percepción del usuario y el impacto organizacional, elementos fundamentales en proyectos del sector público. Al adoptar este modelo, un proyecto asegura que la solución basada en microservicios no solo alcance la excelencia técnica, sino que también genere un valor

tangible y medible para sus usuarios, cumpliendo con los objetivos de eficiencia y transformación organizacional. [27].

1.4.3 Herramientas y tecnologías para implementación de microservicios

La implementación exitosa de una arquitectura de microservicios puede compararse con la dirección de una orquesta sinfónica: requiere un conjunto armonioso de herramientas especializadas que trabajen en perfecta sincronía a lo largo de todo el ciclo de vida del desarrollo

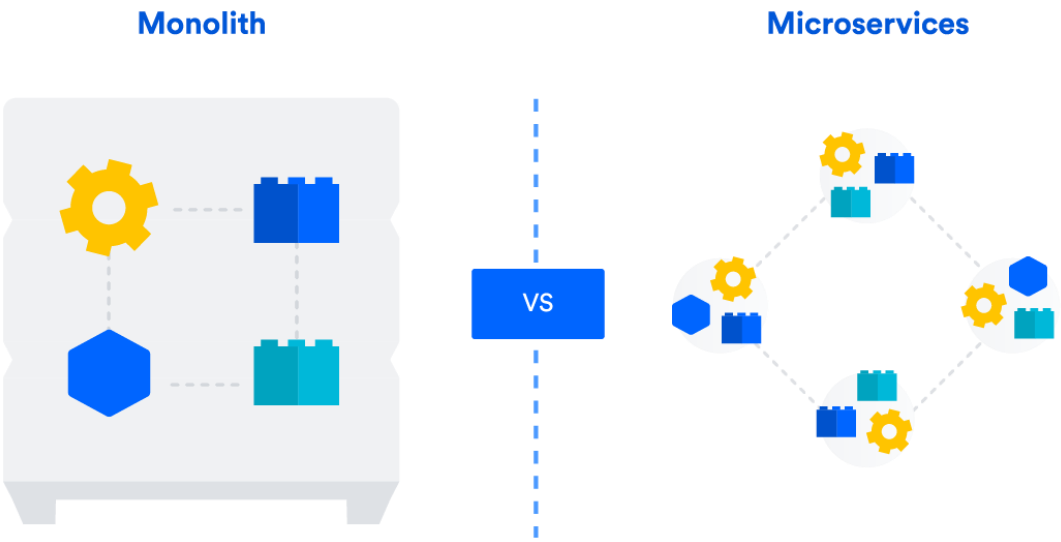


Figura 1.Comparación entre la arquitectura monolítica y la arquitectura de microservicios.

Nota: Adaptado de *Microservicios frente a arquitecturas monolíticas*, por Atlassian.

Este ecosistema tecnológico actúa como el director que coordina la complejidad inherente de los sistemas distribuidos, transformando el caos potencial en una ejecución ágil, escalable y resiliente. A continuación, se detallan las categorías fundamentales de herramientas seleccionadas para el proyecto.

Herramientas de Gestión y Comprobación de API

Las APIs constituyen el canal fundamental de comunicación entre los microservicios y los sistemas externos. Una gestión rigurosa y pruebas exhaustivas son críticas para garantizar la interacción fluida y la fiabilidad del sistema.

Tabla 5. Herramientas para el diseño, documentación y pruebas de APIs.

Herramienta	Descripción
OpenAPI (Swagger)	Estándar ampliamente utilizado para diseñar, documentar y probar APIs. Permite definir la estructura de los servicios de manera agnóstica al lenguaje.
Postman	Suite de desarrollo integral ideal para equipos. Facilita la exploración de APIs RESTful, la creación de solicitudes HTTP complejas y la automatización de pruebas básicas.

Lenguajes y Frameworks de Desarrollo

La elección del stack tecnológico impacta directamente en la escalabilidad y mantenibilidad de los microservicios. Para el presente proyecto de supervisión y regulación, se han seleccionado tecnologías que equilibran rendimiento y agilidad.

- **Laravel (PHP):** Framework de aplicación web con una sintaxis expresiva. Su objetivo es simplificar tareas complejas como el enrutamiento, la autenticación y el almacenamiento en caché. Gracias a su ORM *Eloquent*, facilita la interacción con bases de datos, permitiendo construir servicios de backend robustos y escalables para la gestión de datos institucionales.
- **React (JavaScript):** Biblioteca de código abierto para construir interfaces de usuario dinámicas. Basada en una arquitectura de componentes, permite que cada elemento de la UI maneje su propio estado de forma independiente. El uso del Virtual DOM optimiza el rendimiento, actualizando solo los elementos necesarios, lo cual es crucial para los paneles de control de supervisión en tiempo real.

Entornos de Desarrollo (IDE)

El editor de código es el instrumento principal del desarrollador para garantizar la limpieza y eficiencia del código fuente.

- **Visual Studio Code (VS Code):** Editor versátil de Microsoft. Su ecosistema de extensiones, junto con herramientas como *IntelliSense* y la integración nativa con Git, mejora significativamente la productividad en el desarrollo de microservicios distribuidos.

Herramientas de Mensajería (Messaging Brokers)

Para desacoplar los microservicios y permitir una comunicación asíncrona eficiente, es indispensable el uso de *brokers* de mensajería. Estas herramientas gestionan los flujos de datos y garantizan que no se pierdan solicitudes durante picos de carga.

- **RabbitMQ:** Agente de mensajes de código abierto que actúa como intermediario para la comunicación entre diferentes aplicaciones. Permite a los sistemas enviar y recibir mensajes de forma asíncrona, enrutándolos desde "productores" a "colas" y luego a "consumidores", lo que facilita la creación de arquitecturas de microservicios distribuidas.

Orquestación y Contenedorización

Para gestionar el ciclo de vida de los microservicios, desde el desarrollo hasta el despliegue, se utilizan tecnologías de contenedores.

- **Docker:** Plataforma que permite empaquetar aplicaciones y sus dependencias en contenedores. Esto garantiza que el software se ejecute de manera idéntica en cualquier entorno (desarrollo, pruebas, producción), eliminando el problema de "funciona en mi máquina".

1.5 Trabajos relacionados

Duvvur realizó un estudio de caso sobre la modernización de un sistema de TI gubernamental con más de dos décadas de antigüedad con el objetivo de resolver desafíos críticos de eficiencia, escalabilidad y seguridad. La metodología se basó en una estrategia integral de cuatro fases que incluyó la automatización de procesos, con ello también una migración estructurada a la nube y la implementación de analíticas avanzadas en el sistema gubernamental el fortalecimiento de la seguridad [28]. El resultado que fue más relevante es que la modernización logró una reducción del 60% en los procesos manuales,

una disminución del 70% en el tiempo de inactividad y un ahorro del 42% en los costos operativos. La principal limitación de este trabajo es que se enfoca en la modernización de un sistema único a través de la migración a la nube y la IA, en lugar de tomar en cuenta la integración de sistemas entre diferentes entidades mediante una arquitectura de microservicios.

Nuthalapati examinó cómo la infraestructura de nube esto también incluyendo arquitecturas nativas, está revolucionando la prestación de servicios gubernamentales para romper los silos tradicionales, mejorar la entrega de servicios y fomentar la participación ciudadana. Para ello, el estudio analiza los fundamentos técnicos de la gobernanza moderna, como la contenerización, los microservicios, el diseño "API-first" y patrones de integración para sistemas legados como las fachadas de API (API facades) [29]. La investigación destaca que la adopción de estas arquitecturas puede reducir el tiempo de desarrollo de integraciones entre un 40% y 60% además disminuir la carga administrativa para los ciudadanos hasta en un 30%. Aunque este artículo ofrece un marco conceptual y estratégico muy valioso su principal limitación es que presenta una visión general de alto nivel y no profundiza en una implementación técnica específica ni ofrece un análisis de rendimiento cuantitativo comparable al que se busca en este proyecto.

Tapia et al. desarrollaron un estudio de rendimiento comparativo entre una aplicación web implementada con una arquitectura monolítica tradicional y la misma aplicación refactorizada a una arquitectura de microservicios. La metodología consistió en la creación de dos escenarios de prueba idénticos en recursos de hardware como por ejemplo uno con la aplicación monolítica sobre una máquina virtual y el otro con la aplicación de microservicios orquestada en contenedores Docker, estas dos aplicaciones fueron puestas a pruebas de estrés para medir métricas clave como el consumo de CPU, memoria, rendimiento de red y operaciones de disco, validando estos resultados resultados [30]. El resultado más relevante fue que la arquitectura de microservicios demostró una mayor eficiencia, procesando más solicitudes por segundo y utilizando la memoria de forma mucho más óptima, aunque con un consumo de CPU ligeramente superior. La principal limitación de este trabajo es el uso de una aplicación genérica tipo foro, sin abordar las complejidades de la integración entre sistemas de distintas organizaciones del mundo real.

Sethi & Panda exploraron la evolución arquitectónica de las Plataformas de Experiencia Digital (DXPs) desde estructuras monolíticas hacia microservicios para superar los desafíos de escalabilidad. El estudio analiza estrategias de descomposición basadas en

capacidades de negocio y subdominios de acuerdo con el Domain-Driven Design (DDD), y destaca el uso del patrón "strangler application" para una refactorización incremental y segura [31]. El resultado principal es que una evolución estratégica, utilizando patrones arquitectónicos como el "strangler" y el modelo del "cubo de escala" que permite obtener una arquitectura más estable, flexible y escalable, como lo demuestra el caso de estudio de la plataforma de Backbase. La principal limitación de este trabajo es que, si bien discute conceptos y patrones arquitectónicos de gran valor, no proporciona una comparación de rendimiento cuantitativa ni métricas de hardware/software.

Aksakalli et al. presentaron un enfoque sistemático y una herramienta para generar automáticamente alternativas de despliegue viables para aplicaciones basadas en microservicios sobre recursos con capacidad limitada, con el fin de minimizar los costos totales de comunicación y ejecución. Para lograrlo, el problema de despliegue se modeló como un Problema de Asignación de Tareas Capacitado utilizando parámetros extraídos de los modelos de diseño de la aplicación (requisitos de memoria, costos de comunicación, etc.) [32]. El resultado más destacado fue que el modelo generado algorítmicamente para un caso de estudio de 550 microservicios mejoró el costo total de ejecución en un 2.06% y el de comunicación en un 1.51% en comparación con un despliegue manual experto. Una limitación es que el enfoque depende de estimaciones predefinidas de costos, sin considerar factores dinámicos del mundo real como la latencia de red variable o picos de carga inesperados, aspectos que se pretenden evaluar en este proyecto mediante pruebas de estrés.

Milić & Makajić-Nikolić propusieron un modelo matemático para la optimización de la arquitectura de software, evaluando implementaciones de arquitecturas monolíticas y de microservicios en el contexto de atributos de calidad como acoplamiento, testeabilidad, seguridad y complejidad. Su enfoque metodológico se basó en medir estos atributos mediante métricas de software obtenidas con herramientas de análisis estático como SonarQube , para luego formular un modelo de programación por metas de enteros mixtos (MIGP) que define una solución óptima [33]. Un resultado clave es la definición de un "Punto de Intersección" para sistemas con una sola funcionalidad, donde ambas arquitecturas presentan métricas de calidad idénticas. La limitación de este estudio es que se basa en el análisis estático del código y métricas de construcción, sin incluir mediciones de rendimiento en tiempo de ejecución bajo carga, como el tiempo de respuesta o el consumo de recursos, que son objetivos específicos de este proyecto.

Decimavilla-Alarcón & Marcillo-Franco exploraron la transformación de arquitecturas de microservicios basadas en contenedores para el despliegue ágil de aplicaciones de Internet de las Cosas (IoT) en la nube, abordando los desafíos de escalabilidad, flexibilidad y eficiencia computacional. La metodología de investigación integró una revisión sistemática de la literatura además también un análisis comparativo de estrategias de implementación como microservicios ligeros y escalamiento elástico dinámico, y el estudio de casos múltiples en diversos sectores industriales [34]. El resultado más relevante del estudio fue la demostración en mejoras cuantitativas significativas, incluyendo una reducción del 40% en la latencia de las comunicaciones además de una mejora del 55% en la escalabilidad horizontal y reducciones de costos operativos de entre el 30% y 60% en comparación con arquitecturas monolíticas. Como limitación el estudio se centra en el dominio de IoT, que presenta desafíos únicos (heterogeneidad de dispositivos, redes limitadas) que pueden no ser directamente aplicables a un entorno de integración de sistemas de software institucionales.

Gadani investigó la aplicación de una arquitectura de microservicios en los libros de registro electoral electrónicos (electronic poll books) para mejorar la flexibilidad y resiliencia de los sistemas electorales. El estudio se basó en una comparación analítica entre las arquitecturas de microservicios y las monolíticas tradicionales, destacando las ventajas en tolerancia a fallos, actualizaciones rápidas y personalización del sistema mediante la revisión de literatura y casos de estudio [35]. Como resultado principal, la investigación demostró mejoras notables en la resiliencia del sistema, logrando una reducción del 78% en el tiempo promedio de inactividad por falla, una recuperación un 92% más rápida tras una falla del servicio y una mejora de 4 veces en la capacidad de escalabilidad durante elecciones de gran magnitud. Una limitación importante es que el trabajo se enfoca en los beneficios conceptuales y las mejoras reportadas, sin detallar una implementación técnica específica ni presentar resultados de pruebas de rendimiento empíricas sobre un sistema desarrollado, que es un objetivo de este proyecto.

Oumoussa & Saidi realizaron una revisión sistemática de la literatura para analizar la evolución y el estado del arte en la identificación de microservicios al descomponer aplicaciones monolíticas. Su metodología se basó en las guías de Kitchenham y Charterss para seleccionar y analizar 168 artículos, clasificando las contribuciones en categorías como nuevas herramientas, estudios empíricos y propuestas teóricas. El estudio concluye que la investigación en este campo, aunque activa, se encuentra en una etapa temprana,

destacando la necesidad de herramientas de identificación más precisas y automatizadas, así como benchmarks estandarizados para la evaluación [36]. La limitación de este trabajo, desde la perspectiva de tu proyecto, es que se enfoca exclusivamente en el desafío de la descomposición de un monolito, mientras que tu investigación se centra en la *integración* de sistemas ya existentes, aunque los principios de delimitación de servicios siguen siendo conceptualmente relevantes.

CAPÍTULO III

MATERIALES Y MÉTODOS

El presente capítulo establece la fundamentación técnica y metodológica que sustenta el diseño y construcción de la plataforma de integración para la supervisión de construcciones. Se detallan herramientas de software, lenguajes de programación y marcos de trabajo seleccionados, justificando su pertinencia para una arquitectura de microservicios distribuida. Asimismo, se describe la metodología de desarrollo aplicada para gestionar el ciclo de vida del proyecto, garantizando que la interoperabilidad entre las instituciones públicas (CAE-I, CICI e INNOVA EP).

2.1 Fase exploración

2.1.1 Visión del proyecto

INNOVA EP al ser empresa pública encargada de verificar el hábitat y el desarrollo urbano sostenible en la provincia de Imbabura se enfoca en varios puntos como es el control de viviendas de interés social o las viviendas de interés público, además su compromiso también se ve reflejado en la implementación de políticas y ordenanzas que fomenten un desarrollo responsable todo esto para reducir el déficit habitacional junto con ayuda del Colegio de arquitectos de Imbabura y también el Colegio de Ingenieros Civiles de Imbabura.

La visión que se tiene en el presente trabajo de titulación es implementar una arquitectura de integración de sistemas para que estas tres entidades realizar los procesos definidos en cada uno de ellos, asegurando su interoperabilidad y cooperación de cada una de ellas.

El presente trabajo de grado se centra en la implantación de una arquitectura de integración de sistemas mediante servicios distribuidos y microservicios. Esto permite centralizar los sistemas, lo que logra optimizar el flujo del proceso en el control y supervisión de las obras de construcción en la provincia de Imbabura de una manera unificada para las instituciones responsables.

2.2 Metodología Aplicada

2.2.1 Fases del Proyecto

El desarrollo del proyecto se ha organizado en cinco fases estratégicas que cubren desde la concepción hasta la entrega final y documentación.

1. **Fase 1: Análisis:** Levantamiento de requerimientos con los actores clave (INNOVA EP, CAEI, CICI) para definir el flujo de los trámites y las necesidades de interoperabilidad.
2. **Fase 2: Diseño:** Definición de la arquitectura de microservicios, diseño de contratos de API y configuración de la topología de red en Docker.
3. **Fase 3: Implementación:** Desarrollo iterativo del código fuente, configuración del bus de mensajería e integración de los sistemas legados con el orquestador.
4. **Fase 4: Evaluación:** Ejecución de pruebas de integración para asegurar la calidad de los sistemas.

2.2.1 Definición de nomenclatura

Con el fin de estandarizar la documentación y facilitar la trazabilidad entre los diferentes artefactos se establecieron códigos de identificación los cuales permiten referenciar las historias de usuarios, los diferentes requisitos a lo largo del documento, en la siguiente tabla se detallan las siglas utilizadas.

Tabla 6. Nomenclaturas Scrum.

Nomenclatura	Significado
HU	Historias de Usuario
RF	Requerimientos Funcionales
RNF	Requerimientos no Funcionales

2.2.2 Metodología Scrum

Scrum se define como un marco de trabajo fundamentado en métodos ágiles, cuyo objetivo principal es el control permanente del estado actual del software. A diferencia de los enfoques tradicionales, Scrum utiliza un enfoque incremental basado en la teoría de control empírico de procesos, la cual se sustenta en tres pilares fundamentales:

transparencia, que garantiza la visibilidad de los aspectos significativos del proceso; inspección, que ayuda a detectar variaciones indeseables; y adaptación, que permite realizar los ajustes pertinentes para minimizar el impacto de dichas variaciones. En este modelo, el equipo de trabajo se auto-organiza para determinar la mejor forma de entregar los resultados, priorizando a los individuos y las interacciones sobre los procesos rígidos [37].

2.2.3 Roles de Scrum

Para el desarrollo de la solución se trabajará bajo la metodología ágil SCRUM por lo tanto se definieron los roles que participaron en el proyecto hasta su conclusión, según se especifica en la siguiente tabla.

Tabla 7. Equipo Scrum.

Rol	Responsable	Dependencia
Propietario del producto (Product Owner)	Mgs. Katya Bastidas	Gerente general de Innova-EP
Jefe del proyecto (Scrum Máster)	Msc. Diego Trejo	Director del trabajo de titulación
Equipo de desarrollo	Sr. Kevin Guacanes	Tesista

Para garantizar el éxito en la construcción de la plataforma de integración, se ha estructurado el desarrollo en fases claramente definidas, alineadas con los principios de la metodología Scrum. A continuación, se detalla el ciclo de vida del proyecto, las técnicas de estimación y la ejecución por iteraciones (Sprints).

2.3 Fase 1: Análisis

2.3.1 Requerimientos funcionales

Los requisitos funcionales constituyen la especificación del comportamiento del sistema y son actividades decisivas para el análisis del software [38].

Tabla 8. Requerimientos funcionales.

Código	Descripción
RF01	El sistema debe exponer una API REST para recibir las solicitudes de inicio de trámite provenientes de los sistemas externos (CAEI o CICI).
RF02	El sistema debe encolar y distribuir los mensajes a las colas correspondientes (ej. cola-caei, cola-cici, cola-innova) basándose en el tipo de trámite y su estado actual.
RF03	El servicio debe validar que la estructura del JSON recibido cumpla con el esquema predefinido.
RF04	Cuando un plano es aprobado en el sistema del CAEI, el sistema debe disparar automáticamente un evento que notifique al bus de servicios (RabbitMQ) para habilitar la revisión estructural.
RF05	El sistema del CICI debe ser capaz de consultar (o recibir) la confirmación de "Planos Arquitectónicos" para permitir el ingreso de revisión estructural.
RF06	El sistema debe permitir consultar el historial completo de un trámite específico en INNOVA EP, mostrando las fechas y horas exactas de dicho movimiento.
RF07	Si una institución emite una observación o rechazo, el sistema debe permitir notificar al usuario interesado.

2.3.2 Requerimientos no funcionales

Los requisitos no funcionales son las restricciones o cualidades que definen cómo debe funcionar un sistema, en lugar de lo que el sistema hace. estos se centran en atributos de calidad como el rendimiento, la seguridad, la usabilidad y la disponibilidad, asegurando que el software sea eficiente y confiable para el usuario final. en lugar de funciones específicas, establecen estándares técnicos que garantizan que la arquitectura soporte la carga de trabajo y cumpla con las normativas vigentes [39].

Tabla 9. Requerimientos no funcionales.

Código	Descripción
RNF01	El tramite-service debe procesar y enrutar un mensaje a la cola de RabbitMQ correspondiente en un tiempo no mayor a **500 milisegundos** tras recibir la petición HTTP.

RNF02	El intercambio de información entre los sistemas legados (Laravel) y el servicio de orquestación debe realizarse estrictamente bajo el formato **JSON** estándar.
RNF03	La arquitectura debe permitir la integración de nuevos clientes (ej. una futura App Móvil) sin necesidad de modificar la lógica del núcleo, siempre que consuman la API REST definida.
RNF04	El tramite-service debe validar que las peticiones provengan exclusivamente de orígenes confiables mediante el uso de API Key en las cabeceras HTTP.
RNF05	El sistema debe generar un registro (log) inmutable de todas las transacciones de entrada y salida en el bus de servicios para auditorías futuras.

2.3.4 Historias de usuario

Las historias de usuario sirven para la comunicación de los requerimientos de software, pero es un problema. Las personas que lo solicitan, ya sea porque lo van a utilizar o porque lo van a pagar, deben comunicarse con las personas responsables que lo construyen ya que para que el proyecto sea exitoso, se necesitan diferentes puntos de vista (clientes, usuarios finales, y expertos del dominio y los equipos de desarrollo). Si no existe un balance, entonces el proyecto puede fracasar ya sea porque el negocio domina la capacidad de lo que equipo puede construir o porque domina el aspecto técnico y se pierden de vista las necesidades del negocio. Para que los proyectos sean exitosos se necesitan mecanismos que permitan a ambas partes trabajar en forma conjunta [40].

Tabla 10. Formato historias de usuario.

Historia de usuario	
Número:	**Usuario:**
Nombre historia:	
Prioridad:	**Estimación:**
Descripción:	
Validación:	

Técnica de medición

Técnica T-Shirt

La técnica de *T-Shirt Sizing* es uno de los métodos para la estimación ágil que prescinde del uso de modelos numéricos precisos en etapas tempranas, optando por una valoración al principio abstracta del esfuerzo o complejidad del proyecto. Esta técnica clasifica las tareas o historias de usuario utilizando una escala de tallas estándar como si fuesen tallas de una camiseta: XS (*Extra Small*), S (*Small*), M (*Medium*), L (*Large*), XL (*Extra Large*) y XXL (*Double Extra Large*) [41].

Su principal propósito es el de facilitar un consenso para el equipo de desarrollo mediante la abstracción, permitiendo evaluar la dimensión de los requerimientos sin la carga cognitiva que implica asignar una duración temporal exacta. En el contexto de la planificación ágil, sitúa esta técnica en el nivel táctico, siendo excelente para la planificación de lanzamientos (*releases*) con un horizonte temporal de meses, donde los requisitos se especifican a nivel de Épicas o Historias de Usuario, a diferencia de la planificación operativa del Sprint que requiere mayor detalle [42] En este proceso utilizaremos las siguientes categorías:

Tabla 11. Estimación de esfuerzo.

Estimación de esfuerzo	**Talla**
5-10 horas	S
10-20 horas	M
20-30 horas	L

Se presenta a continuación las necesidades del usuario sobre las funciones del sistema por medio de historias de usuario.

Tabla 12. Historia de usuario HU1.

Historia de usuario	
Número: HU1	**Usuario:** Sistema Externo (CAEI o CICI)
Nombre historia: Exposición de API REST para recepción de trámites	
Prioridad: Alta	**Estimación:** M (10-20 horas)

Descripción:

Como sistema externo (CAEI o CICI), necesito enviar solicitudes de inicio de trámite a través de una API REST para que el Tramite Service pueda recibir, validar y procesar automáticamente los trámites de supervisión y regulación de construcciones en la provincia de Imbabura, evitando procesos manuales y duplicación de información.

Validación:

- Pruebas con Postman enviando peticiones válidas e inválidas.
- Verificar respuestas HTTP correctas (201 Created para éxito, 400 Bad Request para JSON inválido, 401 Unauthorized sin API Key válida).
- Confirmar que el trámite se registra en PostgreSQL y se encola en RabbitMQ.

Tabla 13. Historia de usuario HU2.

Historia de usuario	
Número: HU2	**Usuario:** Sistema de Orquestación (Tramite Service)
Nombre historia: Encolamiento y distribución automática de mensajes	
Prioridad: Alta	**Estimación:** L (20-30 horas)

Descripción:

Como sistema de orquestación, necesito encolar y distribuir automáticamente los mensajes recibidos a las colas correspondientes en RabbitMQ según el tipo de trámite y estado actual, para que cada institución revise la información relevante y se garantice la trazabilidad del flujo de supervisión de construcciones.

Validación:

- Enviar trámites simulados de diferentes tipos y estados.
- Verificar en la interfaz de gestión de RabbitMQ que cada mensaje llega a la cola correcta según routing key.

Tabla 14. Historia de usuario HU3.

Historia de usuario	
Número: HU3	**Usuario:** Sistema INNOVA-EP

Nombre historia: Validación de esquema JSON recibido	
Prioridad: Alta	**Estimación:** S (5-10 horas)
Descripción: Como principal, necesito verificar que el JSON recibido desde CAEI o CICI cumpla con el esquema predefinido para garantizar la integridad y completitud de los datos antes de procesar cualquier trámite de regulación de construcciones.	
Validación: • Pruebas con ApiKey válidos e inválidos. • Confirmar rechazo inmediato con error descriptivo en caso de no cumplimiento.	

Tabla 15. Historia de usuario HU4.

Historia de usuario	
Número: HU4	**Usuario:** Sistema CAEI
Nombre historia: Disparo automático de eventos de planos arquitectónicos	
Prioridad: Media	**Estimación:** M (10-20 horas)
Descripción: Como sistema CAEI, cuando un plano es creado, editado o si se ha realizado alguna acción predefinida, necesito disparar automáticamente un evento hacia el bus de servicios (RabbitMQ) y muestre en la cola de mensajería.	
Validación: • Simular acciones de planos en CAEI. • Verificar publicación del evento en RabbitMQ con routing key correcto. • Confirmar recepción y procesamiento por el consumidor correspondiente.	

Tabla 16. Historia de usuario HU5.

Historia de usuario	
Número: HU5	**Usuario:** Sistema CICI

Nombre historia: Disparo automático de eventos de planos estructurales	
Prioridad: Media	**Estimación:** M (10-20 horas)
Descripción: Como sistema CICI, cuando un plano es creado, editado o si se ha realizado alguna acción predefinida, necesito disparar automáticamente un evento hacia el bus de servicios (RabbitMQ) y muestre en la cola de mensajería.	
Validación: • Simular acciones de planos en CICI. • Verificar publicación del evento en RabbitMQ con routing key correcto. • Confirmar recepción y procesamiento por el consumidor correspondiente.	

Tabla 17. Historia de usuario HU6.

Historia de usuario	
Número: HU6	**Usuario:** Funcionario de INNOVA EP
Nombre historia: Consulta de movimiento de trámites	
Prioridad: Media	**Estimación:** M (10-20 horas)
Descripción: Como funcionario de INNOVA EP, necesito consultar el historial completo de un trámite específico, incluyendo fechas y horas, para garantizar trazabilidad y facilitar auditorías en los procesos dentro de la empresa de INNOVA-EP.	
Validación: • Crear un trámite simulado con múltiples cambios de estado. • Verificar que se muestran todos los eventos en orden cronológico con timestamps precisos en formato ISO 8601.	

Tabla 18. Historia de usuario HU7.

Historia de usuario	
Número: HU7	Usuario: Solicitante

Nombre historia: Notificación automática de observaciones	
Prioridad: Baja	Estimación: M (10-20 horas)
Descripción: Como solicitante de un trámite de construcción, cuando la institución INNOVA EP emite una observación o rechazo, necesito recibir una notificación para poder tomar acciones correctivas de manera oportuna y evitar demoras innecesarias.	
Validación: • Simular emisión de observación/rechazo. • Verificar envío de notificación por al menos un canal en este caso email.	

Tabla 19. Historia de usuario HU8.

Historia de usuario	
Número: HU8	**Usuario:** Desarrollador
Nombre historia: Estandarización del formato de intercambio JSON	
Prioridad: Alta	**Estimación: S (5-10 horas)**
Descripción: Como desarrollador de integración, necesito que todo intercambio de datos entre los sistemas legacy (Laravel) y el Tramite Service utilice JSON estándar (RFC 8259) para asegurar compatibilidad, facilidad de depuración y mantenimiento a largo plazo.	
Validación: • Probar integración bidireccional entre Laravel y NodeJS sin pérdida de datos. • Verificar fechas en ISO 8601 y encoding UTF-8.	

Tabla 20. Historia de usuario HU9.

Historia de usuario	
Número: HU9	Usuario: Product Owner
Nombre historia: Arquitectura extensible para nuevos clientes	
Prioridad: Alta	Estimación: L (20-30 horas)

Descripción:

Como Product Owner, necesito que la arquitectura permita integrar nuevos clientes (por ejemplo, una futura app móvil para solicitantes) sin modificar la lógica del núcleo del Tramite Service, siempre que consuman la API REST definida.

Validación:

- Documentar API con Swagger/OpenAPI.
- Simular integración de un cliente nuevo (ej. Postman como app móvil).
- Confirmar funcionamiento sin cambios en el backend.

Tabla 21. Historia de usuario HU10.

Historia de usuario	
Número: HU10	Usuario: Administrador del Sistema
Nombre historia: Configuración de Docker Compose en Portal	
Prioridad: Alta	Estimación: L (20-30 horas)

Descripción:

Como administrador del sistema, necesito configurar correctamente el archivo docker-compose.yml en el directorio portal para orquestar todos los servicios del sistema y asegurar su correcta comunicación interna mediante redes personalizadas, volúmenes para persistencia y variables de entorno adecuadas.

Validación:

- Ejecutar docker-compose up y verificar que todos los servicios se inicien sin errores.
- Confirmar comunicación entre contenedores
- Verificar persistencia de datos en volúmenes (reiniciar contenedores sin pérdida).
- Comprobar que solo los puertos necesarios están expuestos y sin conflictos.

Tabla 22. Historia de usuario HU11.

Historia de usuario	
Número: HU11	Usuario: Administrador del Sistema

Nombre historia: Configuración del Aplicativo de CAEI

Prioridad: Alta Estimación: M (10-20 horas)

Descripción:

Como administrador del sistema, necesito configurar y desplegar el aplicativo de CAEI en el entorno Docker para que se integre correctamente con el servicio de trámites, shared-documents, autenticación y base de datos PostgreSQL.

Validación:

- El módulo CAEI está desplegado y accesible en su puerto asignado.
- Conexión a PostgreSQL funciona y migraciones se ejecutan correctamente.
- Integración con tramite-service y shared-documents es operativa.
- Funcionalidades principales (gestión de asesoramiento y emprendimientos) responden sin errores.
- Logs no muestran errores críticos y pruebas de integración pasan.

Tabla 23. Historia de usuario HU12.

Historia de usuario	
Número: HU12	Usuario: Administrador del Sistema
Nombre historia: Configuración del Aplicativo de CICI	
Prioridad: Alta	Estimación: M (10-20 horas)
Descripción:	
Validación:	

- El módulo CICI está desplegado y accesible en su puerto asignado.
- Conexión a PostgreSQL funciona y migraciones se ejecutan correctamente.
- Integración con tramite-service y shared-documents es operativa.
- Funcionalidades principalesresponden sin errores.
- Logs no muestran errores críticos y pruebas de integración pasan.

Tabla 24. Historia de usuario HU13.

Historia de usuario

Número: HU13	Usuario: Administrador del Sistema
Nombre historia: Configuración del Aplicativo de INNOVA-EP	
Prioridad: Alta	Estimación: M (10-20 horas)
Descripción:	
Validación:	

- El módulo INNOVA-EP está desplegado y accesible en su puerto asignado.
- Conexión a PostgreSQL funciona y migraciones se ejecutan correctamente.
- Integración con tramite-service y shared-documents es operativa.
- Funcionalidades principalesresponden sin errores.
- Logs no muestran errores críticos y pruebas de integración pasan.

2.3.5 Definición de sprints

La fase de construcción de software se dividió en tres Sprints funcionales adicionales, completando así el ciclo de desarrollo.

2.3.6 Sprint 0 – Configuración del Entorno y Arquitectura

Este sprint fundacional tuvo como objetivo preparar la infraestructura de contenedores y asegurar la comunicación base entre servicios antes de iniciar el desarrollo de lógica de negocio.

Tabla 25. Sprint 0.

Sprint 0			
Fecha Inicio: 23/08/2025 **Fecha Fin: 05/09/2025**			
ID	Historia	Actividades	Horas
HU10	Configuración de Docker Compose	• Creación de redes (bridge) y volúmenes. • Definición de servicios (Postgres, RabbitMQ, Node).	25

		• Configuración de variables de entorno (.env).	
HU11	Configuración Aplicativo CAEI	• Dockerización del entorno Laravel (CAEI). • Configuración de drivers para conexión a RabbitMQ.	15
HU12	Configuración Aplicativo CICI	• Dockerización del entorno Laravel (CICI). • Configuración de drivers para conexión a RabbitMQ.	15
HU13	Configuración Aplicativo INNOVA-EP	• Dockerización del entorno Laravel (INNOVA-EP). • Configuración de drivers para conexión a RabbitMQ.	30
Total:			**85 horas**

2.3.7 Sprint 1 – Desarrollo del Núcleo de Integración

El objetivo de este sprint fue implementar el "corazón" del sistema: la API que recibe los trámites y el mecanismo de colas que los distribuye. Sin este núcleo, la interoperabilidad no es posible.

Tabla 26. Sprint 1.

Sprint 1			
Fecha Inicio: 06/09/2025 **Fecha Fin: 19/09/2025**			
ID	Historia	Actividades	Horas
HU1	Exposición de API REST	**Backend:** • Crear rutas en Express (POST /api/tramites).	20

		• Implementar controlador para recepción de datos. • Configurar respuestas HTTP estándar (200, 400, 500). **Pruebas:** • Test de endpoints con Postman.	
HU2	Encolamiento y distribución automática de mensajes	**Backend:** • Implementar servicio productor (Producer) de RabbitMQ. • Configurar Exchanges (Direct) y Routing Keys. • Lógica de distribución a colas (cola-caei, cola-cici).	30
HU3	Validación de esquema JSON	**Backend:** • Crear Middleware de validación de entrada. • Manejo de errores para JSON mal formados.	8
HU4	Eventos automáticos CAEI	**Backend (Laravel):** • Crear Observer en modelo de Planos.	15

		• Implementar Job para envío asíncrono a RabbitMQ.	
Total:			**73 horas**

2.3.8 Sprint 2 – Integración y Trazabilidad

En este sprint se completó la integración con la segunda entidad externa (CICI) y se desarrollaron los mecanismos para garantizar la trazabilidad y seguimiento de los trámites.

Tabla 27. Sprint 2.

Sprint 2			
Fecha Inicio: 20/09/2025 **Fecha Fin: 03/10/2025**			
ID	Historia	Actividades	Horas
HU5	Eventos automáticos CICI	**Backend :** • Publicación de eventos en cola cola-innova tras revisión.	30
HU6	Consulta de movimiento	**Backend:** • Crear entidad de persistencia para logs de auditoría. **Frontend:** • Vista de línea de tiempo del trámite.	40
HU8	Estandarización JSON	**Refactorización:**	8

		Revisión de payloads de entrada y salida. Unificación de nomenclaturas entre CAEI, CICI e INNOVA. Documentación interna de estructuras de datos.	
Total:			**78 horas**

2.3.9 Sprint 3 – Notificaciones y Extensibilidad

El sprint final se enfocó en cerrar el ciclo de comunicación hacia el usuario final (solicitante) y preparar la arquitectura para escalar a futuros clientes móviles, asegurando la calidad del producto final.

Tabla 28. Sprint 3.

Sprint 3			
Fecha Inicio: 04/10/2025 **Fecha Fin: 17/10/2025**			
ID	Historia	Actividades	Horas
HU7		**Backend:** • Implementación de servicio de correo (SMTP). • Suscripción a eventos de "Rechazo" u "Observación". • Plantillas HTML para correos de notificación.	15
HU9		**Documentación:**	40

	Generación de documentación API con Swagger/OpenAPI. Pruebas de consumo desde cliente genérico (simulando App Móvil). **Despliegue:** Configuración final de contenedores para producción.	
Total:		**55 horas**

2.4 Fase 2: Diseño

2.4.1 Introducción a la Arquitectura Propuesta

En esta fase de diseño, se presenta una arquitectura híbrida de microservicios distribuidos, orientada a la modernización de sistemas legados en el sector público de la provincia de Imbabura. El sistema busca optimizar los procesos de supervisión, regulación y aprobación de construcciones, abordando desafíos como la desconexión entre instituciones autónomas (CAEI para aprobación arquitectónica, CICI para aprobación estructural, e INNOVA EP para supervisión de obras). Esta arquitectura se basa en principios de Domain-Driven Design (DDD), que prioriza el modelado del dominio del negocio sobre la complejidad técnica, permitiendo una integración fluida sin comprometer la independencia operativa de cada entidad.

La motivación principal surge de problemas identificados en fases previas de la tesis, como tiempos de procesamiento prolongados (hasta 30 días por trámite), pérdida de documentos físicos y falta de trazabilidad digital, comunes en administraciones públicas latinoamericanas según informes de la CEPAL (Comisión Económica para América Latina y el Caribe, 2022). La propuesta reduce estos tiempos a menos de 7 días mediante automatización y eventos asíncronos.

Justificación inicial: En contextos regulatorios como el de Ecuador, regulado por la Ley Orgánica de Ordenamiento Territorial, Uso y Gestión de Suelo [43] una arquitectura monolítica tradicional genera cuellos de botella. El enfoque híbrido combina microservicios con contenedores Docker para escalabilidad, inspirado en casos exitosos como el sistema GOV.UK del Reino Unido o el Portal Único de Trámites en México.

Beneficios esperados:

- Reducción de costos operativos en un al minimizar mantenimiento de sistemas legados.
- Mejora en la resiliencia: Si un servicio falla (ej. CAEI), los demás continúan operando.
- Cumplimiento con estándares ISO 27001 para seguridad de datos en instituciones públicas.

2.4.2 Metodología Adoptada: Domain-Driven Design (DDD)

DDD es un enfoque de diseño de software que enfatiza el modelado del dominio del negocio como centro del desarrollo [44] . Se adopta DDD para manejar la complejidad inherente a la coordinación interinstitucional, donde la "desconexión entre las instituciones responsables" (CAEI, CICI, INNOVA EP) es el principal desafío, no el código en sí.

Definición detallada: DDD divide el sistema en "Bounded Contexts" (contextos delimitados), cada uno representando un subdominio con su propio lenguaje ubicuo (términos compartidos como "trámite de aprobación" vs. "supervisión de obra"). Esto refleja el dominio en el código, usando entidades, value objects, aggregates y repositories para una modularidad alta.

Razones para la elección de DDD:

- **La complejidad radica en reglas de negocio autónomas:** CAEI maneja normativas arquitectónicas, CICI estructurales, e INNOVA supervisión en sitio.
- **Permite delimitar contextos:** Aislando lógicas para respetar independencia operativa.
- **Fundamenta microservicios:** Definiendo límites lógicos para escalabilidad, evitando el "big ball of mud" común en sistemas legados.

Aplicación práctica en el proyecto:

- **Lenguaje Ubicuo:** Términos como "Documento Subido" generan eventos en RabbitMQ con routing keys como "caei.documento.subido".
- **Patrones DDD implementados:**
    - **Aggregates:** Un "Trámite" como raíz, conteniendo entidades como "Aprobación Arquitectónica".
    - **Domain Events:** Publicación de eventos asíncronos para notificar cambios entre instituciones.

Ventajas y desventajas:

Tabla 29. Análisis de riesgos y estrategias de mitigación en la arquitectura basada en microservicios.

Aspecto	Ventajas	Desventajas	Mitigación en este proyecto
Complejidad	Modela negocio real	Curva de aprendizaje alta	Capacitación inicial para equipos de CAEI/CICI/INNOVA
Escalabilidad	Contextos independientes	Overhead en integración	Uso de API Gateway para routing
Mantenibilidad	Código refleja dominio	Requiere expertos en DDD	Documentación exhaustiva con UML

2.4.3 Contextos Delimitados (Bounded Contexts)

Los Bounded Contexts son subdominios aislados en DDD, cada uno con su modelo propio para evitar ambigüedades. En este proyecto, se delimitan tres contextos principales basados en las instituciones.

Descripción detallada:

- **Contexto CAEI (Aprobación Arquitectónica):** Enfocado en validación de planos **arquitectónicos** y normativas urbanas. Incluye entidades como "Plano Arquitectónico" y servicios para revisión.

- **Contexto CICI (Aprobación Estructural):** Enfocado en validación de planos **estructurales** y normativas urbanas. Incluye entidades como "Plano estructural" y servicios para revisión.
- **Contexto INNOVA (Supervisión de Obra):** Supervisa ejecución, con trazabilidad de inspecciones y documentos compartidos.

2.4.4 Capas de la Arquitectura

La arquitectura se estructura en capas hexagonales (inspiradas en DDD), separando preocupaciones para flexibilidad.

Capa de Acceso Externo: Usuarios (arquitectos, ciudadanos) acceden via VPN (OpenVPN) para cifrado, conectando a frontend via HTTP/SSH.

Capa Frontend:

- Portal Web (Nginx como reverse proxy, puerto 80).
- INNOVA App Docs (Next.js para renderizado dinámico de documentos, SSR para SEO).

Capa de Aplicación: Módulos legados en Laravel + React (CAEI en puerto 8000, CICI en 8001, INNOVA en 8002), con lógica de negocio específica. Migración gradual a microservicios.

Capa de Servicios: Tramite Service (Node.js, puerto 3004) como orquestador, manejando sagas (transacciones distribuidas) para flujos multi-institucionales.

Capa de Infraestructura: RabbitMQ (puertos 5672/15672, exchange "innovaepp_events"), PostgreSQL (bases separadas por contexto para aislamiento), Shared Storage (/shared-documents para persistencia de datos).

Detalles técnicos ampliados:

- Comunicación: REST para síncrono (ej. GET /tramite/status), AMQP para asíncrono (colas durables para reintentos).
- Escalado: Contenedores Docker con auto-escalado basado en CPU.

Tabla 30. Tabla de protocolos.

Capa	Protocolo	Ejemplo	Propósito
Acceso	VPN/HTTP	OpenVPN	Seguridad
Frontend	HTTPS	Nginx	Interfaz usuario
Aplicación	API REST	Laravel	Lógica negocio
Servicios	AMQP	RabbitMQ	Eventos
Infra	TCP/IP	PostgreSQL	Persistencia

2.4.5 Componentes Principales

- **Portal Web (Nginx):** Sirve como gateway, con load balancing y caching (Redis integrado para sesiones).
- **INNOVA App Docs (Next.js):** Genera PDFs dinámicos con librerías como pdf-lib, integrando firmas digitales per-normativa ecuatoriana (Ley de Comercio Electrónico, 2002).
- **Módulos Legados:** CAEI/CICI/INNOVA en Laravel, con React para UI. Actualizaciones incluyen JWT para autenticación.
- **Tramite Service (Node.js):** Core orquestador, usando Express para APIs y amqplib para RabbitMQ.

2.4.6 Topología Detallada

- **Redes:** "app-network" como bridge en Docker, con DNS interno. Preparado para overlay en Docker Swarm para clúster multi-nodo, considerando infraestructura limitada en Imbabura (servidores on-premise).
- **Volúmenes:** Bind mounts para /shared-documents (rw), con backups automáticos via cron jobs. Volúmenes nombrados para DBs para portabilidad.
- **Dependencias y Salud:** Usa healthchecks en docker-compose.yml:
- **Escalado horizontal:** Stateless services permiten replicas (ej. 3 instancias de Tramite Service).

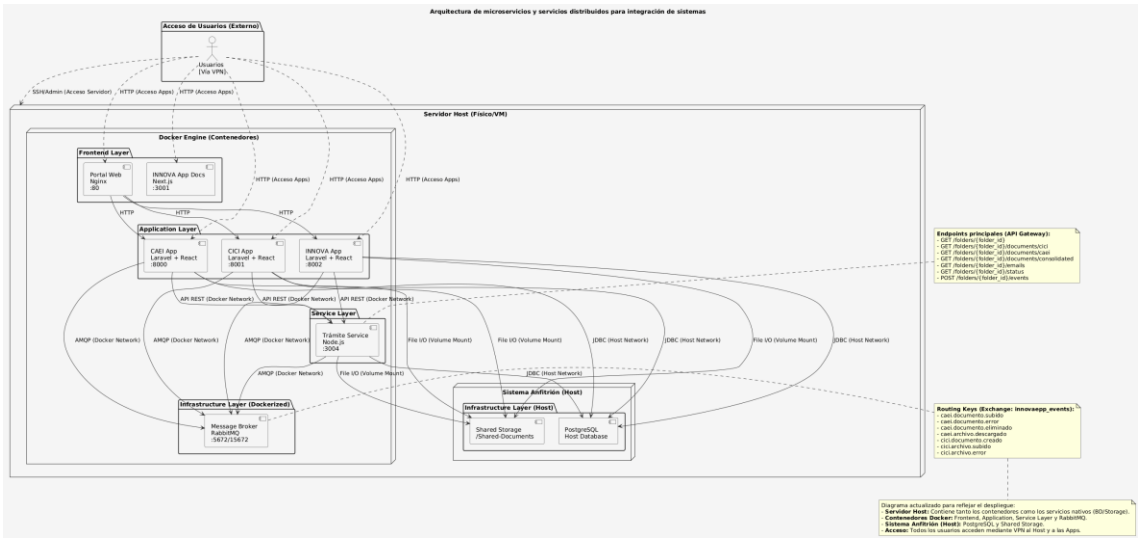


Figura 2. Arquitectura de microservicios y servicios distribuidos para la integración de sistemas.

2.4.7 Justificación y Beneficios

Esta arquitectura resuelve silos informáticos, reduciendo tiempos de recuperación (de días a horas) Permite despliegue independiente, adopción de Node.js para performance en eventos.

Impacto social: Facilita trámites digitales en Imbabura, promoviendo inclusión [45].

2.5 Fase 3 Implementación

En esta fase de implementación, correspondiente al objetivo específico 2 del proyecto, se ejecutó el desarrollo práctico de la arquitectura de microservicios y servicios distribuidos. El trabajo se organizó en cuatro sprints ágiles (Sprint 0 a Sprint 3), alineados con la metodología Scrum adaptada al contexto institucional. Cada sprint tuvo una duración aproximada de 2-3 semanas, con planning, daily stand-ups, revisión (demo) y retrospectiva al final. La implementación se centró en la construcción del Tramite Service como núcleo de integración, la dockerización de los módulos legados (CAEI, CICI, INNOVA-EP), la configuración de la comunicación asíncrona mediante RabbitMQ.

2.5.1 Desarrollo Sprint 0 – Configuración del Entorno y Arquitectura

Objetivo del Sprint: Preparar la infraestructura base de contenedores y asegurar la comunicación entre servicios antes de desarrollar lógica de negocio. Este sprint fundacional eliminó bloqueos tempranos y permitió iteraciones rápidas posteriores.

Desarrollo de HU10

Como punto de partida para la orquestación de la arquitectura, se definió el archivo docker-compose.yml, el cual actúa como el plano de construcción de toda la infraestructura local. En esta primera sección se establece el servicio portal, encargado de servir como punto de entrada unificado.

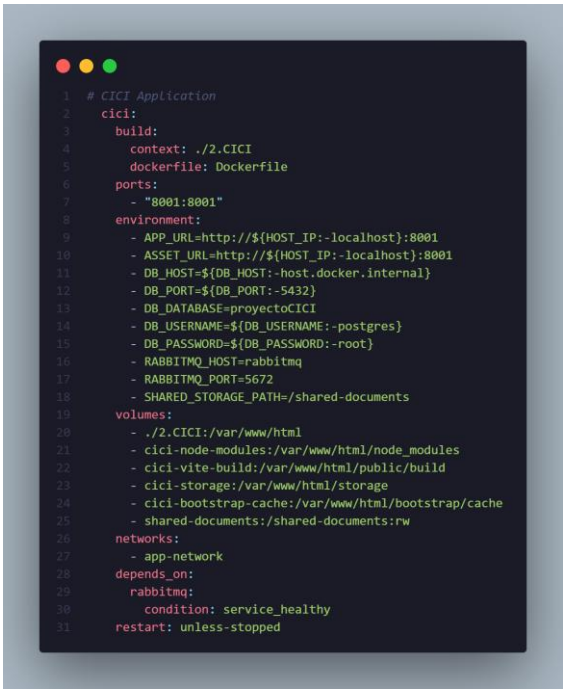
```yaml
# Portal (Nginx)
  portal:
    build:
      context: ./0.portal
      dockerfile: Dockerfile
    ports:
      - "80:80"
    volumes:
      - ./0.portal:/usr/share/nginx/html
      - ./0.portal/nginx-portal.conf:/etc/nginx/conf.d/default.conf
      - shared-documents:/shared-documents:rw
    networks:
      - app-network
    restart: unless-stopped
```

Figura 3. Definición del servicio Portal Web (Nginx) en Docker Compose.

Continuando con la definición de servicios institucionales, se configuraron los contenedores correspondientes al CAEI, CICI y INNOVA-EP. Como se observa en las figuras, cada servicio se despliega en un puerto específico (8000 para CAEI, 8001 para CICI y 8002 para INNOVA) para evitar conflictos en el entorno de desarrollo, aunque todos comparten la misma red interna para garantizar la visibilidad entre ellos

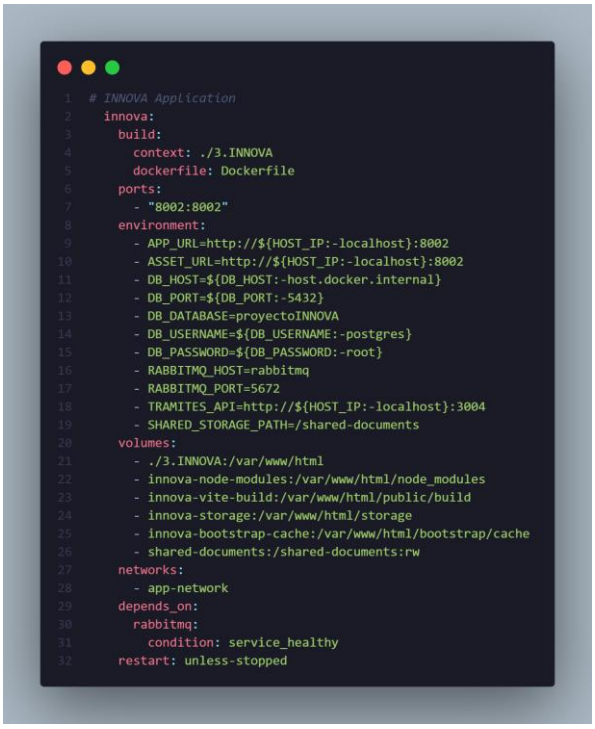
```yaml
# CAE-I Application
  caei:
    build:
      context: ./1.CAE-I
      dockerfile: Dockerfile
    ports:
      - "8000:8000"
    environment:
      - APP_URL=http://${HOST_IP:-localhost}:8000
      - ASSET_URL=http://${HOST_IP:-localhost}:8000
      - DB_HOST=${DB_HOST:-host.docker.internal}
      - DB_PORT=${DB_PORT:-5432}
      - DB_DATABASE=proyectoCAEI
      - DB_USERNAME=${DB_USERNAME:-postgres}
      - DB_PASSWORD=${DB_PASSWORD:-root}
      - RABBITMQ_HOST=rabbitmq
      - RABBITMQ_PORT=5672
      - SHARED_STORAGE_PATH=/shared-documents
    volumes:
      - ./1.CAE-I:/var/www/html
      - caei-node-modules:/var/www/html/node_modules
      - caei-vite-build:/var/www/html/public/build
      - caei-storage:/var/www/html/storage
      - caei-bootstrap-cache:/var/www/html/bootstrap/cache
      - shared-documents:/shared-documents:rw
    networks:
      - app-network
    depends_on:
      rabbitmq:
        condition: service_healthy
    restart: unless-stopped
```

Figura 4. Configuración del servicio "CAE-I" y definición de variables de entorno en Docker Compose.



```yaml
# CICI Application
  cici:
    build:
      context: ./2.CICI
      dockerfile: Dockerfile
    ports:
      - "8001:8001"
    environment:
      - APP_URL=http://${HOST_IP:-localhost}:8001
      - ASSET_URL=http://${HOST_IP:-localhost}:8001
      - DB_HOST=${DB_HOST:-host.docker.internal}
      - DB_PORT=${DB_PORT:-5432}
      - DB_DATABASE=proyectoCICI
      - DB_USERNAME=${DB_USERNAME:-postgres}
      - DB_PASSWORD=${DB_PASSWORD:-root}
      - RABBITMQ_HOST=rabbitmq
      - RABBITMQ_PORT=5672
      - SHARED_STORAGE_PATH=/shared-documents
    volumes:
      - ./2.CICI:/var/www/html
      - cici-node-modules:/var/www/html/node_modules
      - cici-vite-build:/var/www/html/public/build
      - cici-storage:/var/www/html/storage
      - cici-bootstrap-cache:/var/www/html/bootstrap/cache
      - shared-documents:/shared-documents:rw
    networks:
      - app-network
    depends_on:
      rabbitmq:
        condition: service_healthy
    restart: unless-stopped
```

Figura 5. Configuración del servicio "CICI" y definición de variables de entorno en Docker Compose.



```yaml
# INNOVA Application
  innova:
    build:
      context: ./3.INNOVA
      dockerfile: Dockerfile
    ports:
      - "8002:8002"
    environment:
      - APP_URL=http://${HOST_IP:-localhost}:8002
      - ASSET_URL=http://${HOST_IP:-localhost}:8002
      - DB_HOST=${DB_HOST:-host.docker.internal}
      - DB_PORT=${DB_PORT:-5432}
      - DB_DATABASE=proyectoINNOVA
      - DB_USERNAME=${DB_USERNAME:-postgres}
      - DB_PASSWORD=${DB_PASSWORD:-root}
      - RABBITMQ_HOST=rabbitmq
      - RABBITMQ_PORT=5672
      - TRAMITES_API=http://${HOST_IP:-localhost}:3004
      - SHARED_STORAGE_PATH=/shared-documents
    volumes:
      - ./3.INNOVA:/var/www/html
      - innova-node-modules:/var/www/html/node_modules
      - innova-vite-build:/var/www/html/public/build
      - innova-storage:/var/www/html/storage
      - innova-bootstrap-cache:/var/www/html/bootstrap/cache
      - shared-documents:/shared-documents:rw
    networks:
      - app-network
    depends_on:
      rabbitmq:
        condition: service_healthy
    restart: unless-stopped
```

Figura 6. Configuración del servicio "INNOVA EP" y definición de variables de entorno en Docker Compose.

La lógica de negocio reside en el TramiteService. Este componente contiene los métodos encargados de interactuar con la persistencia de datos y, crucialmente, de comunicarse

con el bus de mensajería. Como se muestra en el código, al crear un trámite no solo se guarda el registro, sino que se emite un evento asíncrono a RabbitMQ para notificar a los otros microservicios (CAEI/CICI), asegurando la consistencia eventual de la información en toda la arquitectura distribuida.

Figura 7. Definición del microservicio de Trámites y variables de entorno en Docker Compose.

El componente innova-app-docs representa la capa de presentación moderna orientada al ciudadano y funcionarios. A diferencia de los sistemas legados, este módulo fue construido utilizando el framework Next.js bajo el modelo de App Router.

Figura 8. Definición de la aplicación de seguimiento de trámites y variables de entorno en Docker Compose.

En la arquitectura tambien reside en el servicio de RabbitMQ, cuya configuración se detalla en esta sección. Se optó por utilizar la imagen 3-management, la cual habilita nativamente una interfaz web administrativa en el puerto 15672, facilitando el monitoreo visual de las colas y los intercambios de mensajes.

Figura 9. Configuración del broker de mensajería RabbitMQ y su protocolo de salud en Docker Compose.

Para finalizar la configuración de la infraestructura, se definieron los componentes de red y almacenamiento. Se creó una red tipo puente (bridge) denominada app-network, la cual aísla el tráfico de los microservicios del exterior, permitiendo una comunicación interna segura y resolución de nombres por DNS.

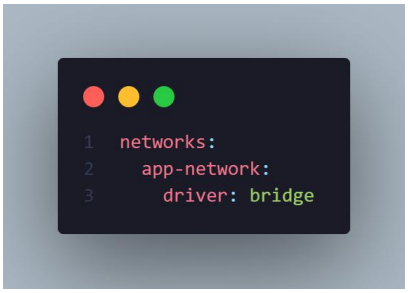


Figura 10. Configuración de la red virtual de tipo bridge para la comunicación entre contenedores.

En cuanto a la persistencia, se declararon volúmenes específicos para cada aplicación (almacenamiento de logs, caché y builds de Vite). Mención especial merece el volumen shared-documents, configurado como un bind local; este volumen compartido es vital para la arquitectura, ya que permite que los diferentes sistemas (CAEI, CICI, INNOVA) accedan físicamente a los mismos archivos de planos y documentos sin necesidad de duplicar la información.

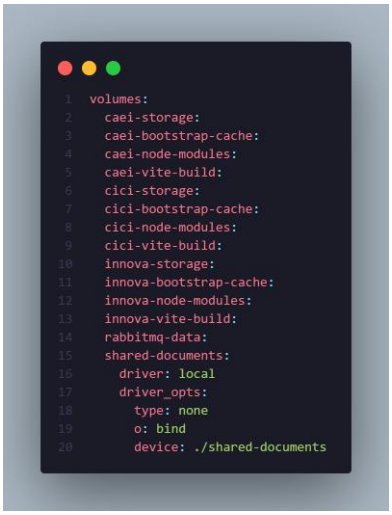


Figura 11. Declaración de volúmenes persistentes y configuración de almacenamiento compartido con driver local.

Desarrollo de HU11, HU12, HU13.

Como parte del sprint se abordó el desarrollo la estandarización de los entornos de despliegue para los sistemas institucionales (CAEI, CICI e INNOVA-EP).

Para lograr esto, se implementó una estrategia de **contenedores personalizados**. Como se observa en la figura, se definió un Dockerfile base que parte de la imagen php:8.2-fpm. Este archivo es crítico porque instala las extensiones de sistema necesarias que no vienen por defecto. Estas librerías son requisitos indispensables para que el cliente PHP de RabbitMQ pueda establecer conexiones persistentes y realizar cálculos de latencia precisos sin errores de ejecución.

```dockerfile
FROM php:8.2-fpm

# Instalar dependencias del sistema
RUN apt-get update && apt-get install -y \
    git \
    curl \
    libpng-dev \
    libonig-dev \
    libxml2-dev \
    zip \
    unzip \
    libpq-dev \
    nodejs \
    npm \
    sudo

# Instalar Node.js 18
RUN curl -fsSL https://deb.nodesource.com/setup_18.x | bash -
RUN apt-get install -y nodejs

# Limpiar cache
RUN apt-get clean && rm -rf /var/lib/apt/lists/*

# Configurar Git para ignorar verificaciones de ownership
RUN git config --global --add safe.directory /var/www/html
RUN git config --global --add safe.directory '*'

# Instalar extensiones de PHP
RUN docker-php-ext-install pdo_pgsql pgsql mbstring exif pcntl bcmath gd

# Instalar Composer
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

# Crear usuario de aplicación
RUN groupadd -g 1000 www
RUN useradd -u 1000 -ms /bin/bash -g www www

# Crear directorios necesarios con permisos adecuados
RUN mkdir -p /var/www/html
RUN chown -R www:www /var/www/html

WORKDIR /var/www/html

# Copiar aplicación completa manteniendo ownership
COPY --chown=www:www . /var/www/html

# Crear directorios necesarios con permisos adecuados
RUN mkdir -p storage/framework/cache/data \
    storage/framework/sessions \
    storage/framework/views \
    storage/logs \
    bootstrap/cache \
    node_modules \
    public/build

# Configurar permisos
RUN chown -R www:www /var/www/html
RUN chmod -R 775 storage bootstrap/cache
RUN chmod -R 775 node_modules public/build

# Copiar script de inicio
COPY --chown=www:www start.sh /usr/local/bin/start.sh
RUN chmod +x /usr/local/bin/start.sh

# Cambiar al usuario www
USER www

# Exponer puerto
EXPOSE 8000

CMD ["/usr/local/bin/start.sh"]
```

Figura 12. Definición de etapas de construcción, instalación de dependencias y configuración del entorno de ejecución en Dockerfile.

Configuración del Service para utilizar RabbitMQ en Laravel

Para habilitar la capacidad de comunicación asíncrona en los sistemas legados desarrollados en Laravel, fue necesario configurar el servicio de mensajería a nivel de infraestructura y aplicación.

La imagen detalla el proceso de preparación del entorno. Se integró la librería php-amqplib/php-amqplib mediante Composer, la cual actúa como el driver de bajo nivel para el protocolo AMQP 0-9-1. A nivel de configuración de servicio, se definieron las

variables de entorno en el archivo de variables de entorno del contenedor (host rabbitmq, puerto 5672), asegurando que la aplicación no tenga credenciales 'quemadas' en el código.

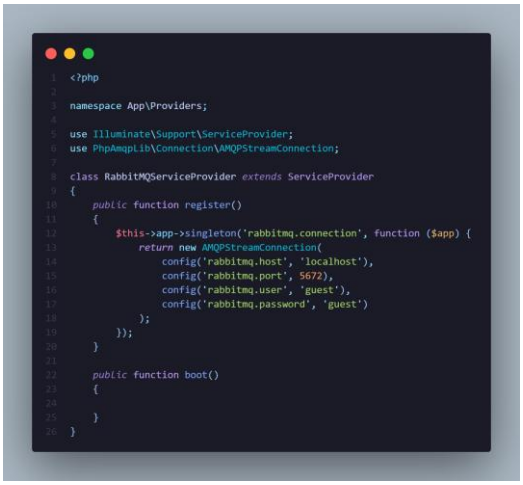
```php
<?php

namespace App\Services;

use Illuminate\Support\Facades\Log;
use Illuminate\Http\UploadedFile;

class StorageService
{
    private string $basePath;
    private string $appName;

    public function __construct(string $appName)
    {
        $this->appName = $appName;
        $this->basePath = env('SHARED_STORAGE_PATH', '/shared-documents') . '/' . $appName;

        $this->ensureDirectoryExists();
    }

    /**
     * Asegurar que el directorio base existe
     */
    private function ensureDirectoryExists(): void
    {
        if (!file_exists($this->basePath)) {
            if (!mkdir($this->basePath, 0777, true)) {
                Log::error("No se pudo crear el directorio base: {$this->basePath}");
                throw new \Exception("No se pudo crear el directorio de almacenamiento");
            }
            Log::info("Directorio creado: {$this->basePath}");
        }
    }

    /**
     * Almacenar un archivo
     */
    public function storeFile(UploadedFile $file, $entityId, ?string $entityName = null): string
    {
        try {
            $entityName = $entityName ?: 'default';

            // Limpiar nombres para seguridad
            $cleanEntityName = $this->sanitizeFileName($entityName);
            $cleanEntityId = $this->sanitizeFileName((string) $entityId);

            $entityPath = $this->basePath . DIRECTORY_SEPARATOR . $cleanEntityName . DIRECTORY_SEPARATOR . $cleanEntityId;

            Log::info('Creando directorio:', ['path' => $entityPath]);

            if (!file_exists($entityPath)) {
                if (!mkdir($entityPath, 0777, true)) {
                    throw new \Exception("No se pudo crear el directorio: {$entityPath}");
                }
                Log::info('Directorio creado exitosamente');
            }

            $originalName = pathinfo($file->getClientOriginalName(), PATHINFO_FILENAME);
            $extension = $file->getClientOriginalExtension();
            $timestamp = now()->setTimezone('America/Guayaquil')->format('YmdHis');

            $cleanName = $this->sanitizeFileName($originalName);
            $filename = "{$cleanName}_{$timestamp}.{$extension}";

            $fullPath = $entityPath . DIRECTORY_SEPARATOR . $filename;

            Log::info('Intentando mover archivo:', [
                'from' => $file->getPathname(),
                'to' => $fullPath,
                'temp_exists' => file_exists($file->getPathname())
            ]);

            $moved = $file->move($entityPath, $filename);

            if (!$moved) {
                throw new \Exception("No se pudo mover el archivo a: {$fullPath}");
            }

            // Verificar que se movió correctamente
            if (!file_exists($fullPath)) {
                throw new \Exception("El archivo no se movió correctamente a: {$fullPath}");
            }

            Log::info("Archivo almacenado exitosamente: {$fullPath}");
            Log::info("Tamaño del archivo movido: " . filesize($fullPath));

            return $fullPath;
        } catch (\Exception $e) {
            Log::error('Error en StorageService storeFile: ' . $e->getMessage(), [
                'entityId' => $entityId,
                'entityName' => $entityName,
                'filename' => $file->getClientOriginalName(),
                'tempPath' => $file->getPathname(),
                'tempExists' => file_exists($file->getPathname())
            ]);
            throw $e;
        }
    }

    /**
     * Eliminar un archivo
     */
    public function deleteFile(string $filePath): bool
    {
        try {
            if (file_exists($filePath)) {
                $result = unlink($filePath);
                if ($result) {
                    Log::info("Archivo eliminado: {$filePath}");
                }
                return $result;
            }
            Log::warning("Archivo no encontrado para eliminar: {$filePath}");
            return true;
        } catch (\Exception $e) {
            Log::error('Error eliminando archivo: ' . $e->getMessage(), [
                'filePath' => $filePath
            ]);
            return false;
        }
    }

    /**
     * Verificar si un archivo existe
     */
    public function fileExists(string $filePath): bool
    {
        return file_exists($filePath);
    }

    /**
     * Obtener el tamaño del archivo
     */
    public function getFileSize(string $filePath): ?int
    {
        return file_exists($filePath) ? filesize($filePath) : null;
    }

    /**
     * Obtener el tipo MIME del archivo
     */
    public function getFileMimeType(string $filePath): ?string
    {
        if (file_exists($filePath)) {
            return mime_content_type($filePath);
        }
        return null;
    }
```

Figura 13. Implementación del servicio de almacenamiento y gestión de directorios mediante variables de entorno.

Configuración de Provider

Para integrar RabbitMQ de manera nativa en el ciclo de vida de Laravel, se implementó un Service Provider personalizado, denominado RabbitMQServiceProvider.

Como se aprecia en el bloque de código, se utilizó el método register() para vincular la clase AMQPStreamConnection al contenedor de inyección de dependencias del framework (Service Container). Se optó por una definición de tipo Singleton ($this->app->singleton), lo que garantiza que se cree una única instancia de la conexión TCP por cada ciclo de petición HTTP. Esta decisión arquitectónica es vital para el rendimiento, ya que evita la sobrecarga (overhead) de abrir y cerrar múltiples conexiones ('handshakes') con el broker de mensajería durante procesos de alta concurrencia.



```php
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use PhpAmqpLib\Connection\AMQPStreamConnection;

class RabbitMQServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->singleton('rabbitmq.connection', function ($app) {
            return new AMQPStreamConnection(
                config('rabbitmq.host', 'localhost'),
                config('rabbitmq.port', 5672),
                config('rabbitmq.user', 'guest'),
                config('rabbitmq.password', 'guest')
            );
        });
    }

    public function boot()
    {

    }
}
```

Figura 14. Implementación del proveedor de servicios para la conexión persistente con el broker RabbitMQ mediante el protocolo AMQP.

2.5.2 Desarrollo Sprint 1 – Desarrollo del Núcleo de Integración

Objetivo del Sprint: Construir el corazón del sistema: la API de recepción de trámites y el mecanismo de encolamiento/distribución.

Desarrollo de H1

El microservicio tramite-service se diseñó como una capa de integración y consolidación de datos especializada en la gestión de trámites, constituyendo un componente clave dentro de la arquitectura distribuida del ecosistema institucional.

Su propósito principal consiste en actuar como intermediario que unifica información proveniente de dos sistemas legados independientes:

- CICI (Colegio de Ingenieros Civiles - Imbabura) – puerto 8001
- CAEI (Colegio de Arquitectos Ecuador - Imbabura) – puerto 8000

Además, el servicio implementa un mecanismo de publicación asíncrona de eventos mediante el broker de mensajería RabbitMQ.

El microservicio sigue los principios fundamentales de la arquitectura de microservicios [46]

- Alta cohesión funcional y bajo acoplamiento
- Despliegue independiente

- Responsabilidad única (Single Responsibility Principle aplicado a nivel de servicio)
- Comunicación síncrona (HTTP/REST) con sistemas legados y asíncrona (mensajería) hacia sistemas consumidores
- Autonomía de datos (aunque en este caso no posee base de datos propia, actúa como agregador virtual)

La estructura de directorios refleja una organización limpia y por capas (Clean Architecture):

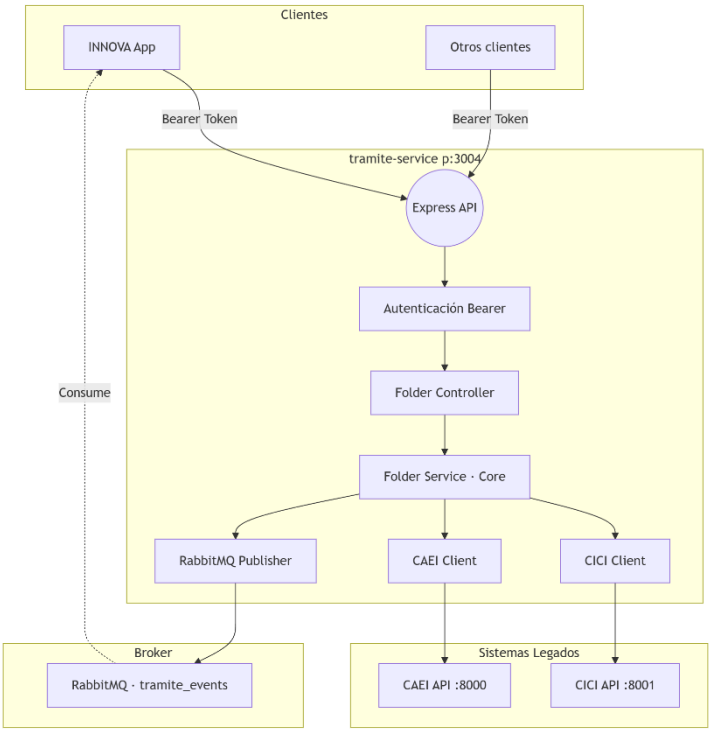


Figura 15. Flujo de comunicación y procesamiento de solicitudes entre clientes, API Gateway y sistemas legados.

Existen dos patrones principales de flujo:

1. **Flujo síncrono de consulta consolidada** Cliente → API → Validación → Consulta paralela CICI + CAEI → Consolidación → Publicación evento → Respuesta
2. **Flujo asíncrono de notificación** Cualquier operación relevante → Publicación evento en exchange tramite_events (topic) → INNOVA consume.

Consulta de trámite consolidado

La consulta de trámite consolidado, que obtiene y unifica toda la información relevante de un trámite desde los sistemas CICI y CAEI, tomando la siguiente estructura.

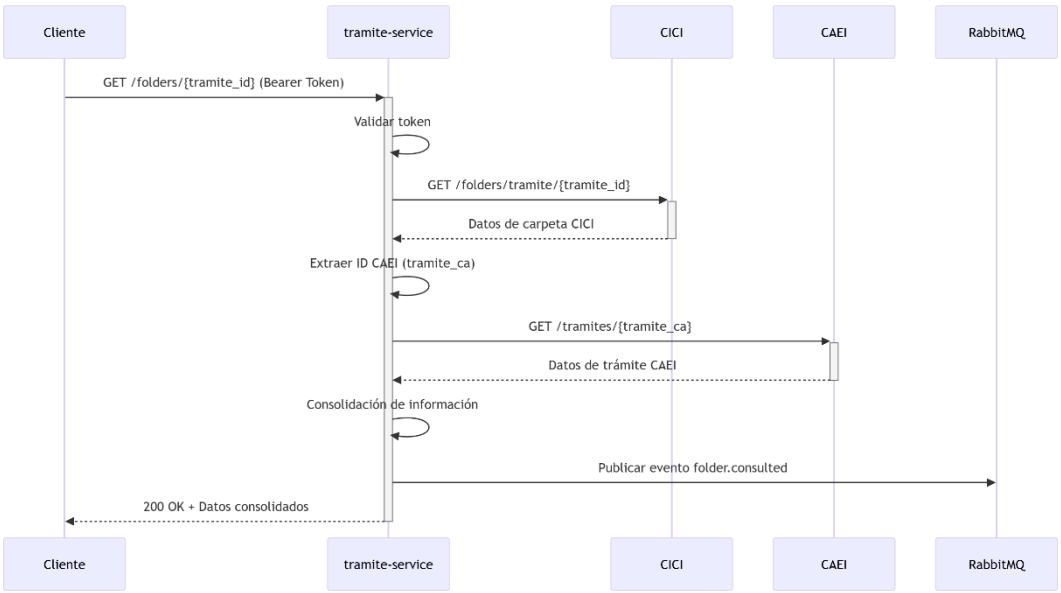


Figura 16. Diagrama de secuencia del proceso de consulta y consolidación de datos entre microservicios y sistemas externos.

Características clave:

- Comunicación síncrona con ambos sistemas legados
- Dos llamadas HTTP en secuencia (la segunda depende del resultado de la primera)
- Publicación de evento como efecto secundario (no bloqueante)
- Tiempo de respuesta depende principalmente de la latencia de CICI y CAEI

Consulta consolidada de documentos

La consulta consolidada de documentos, que recupera de forma eficiente (mediante consultas paralelas) todos los documentos asociados al trámite en ambos sistemas, funciona de la siguiente manera.

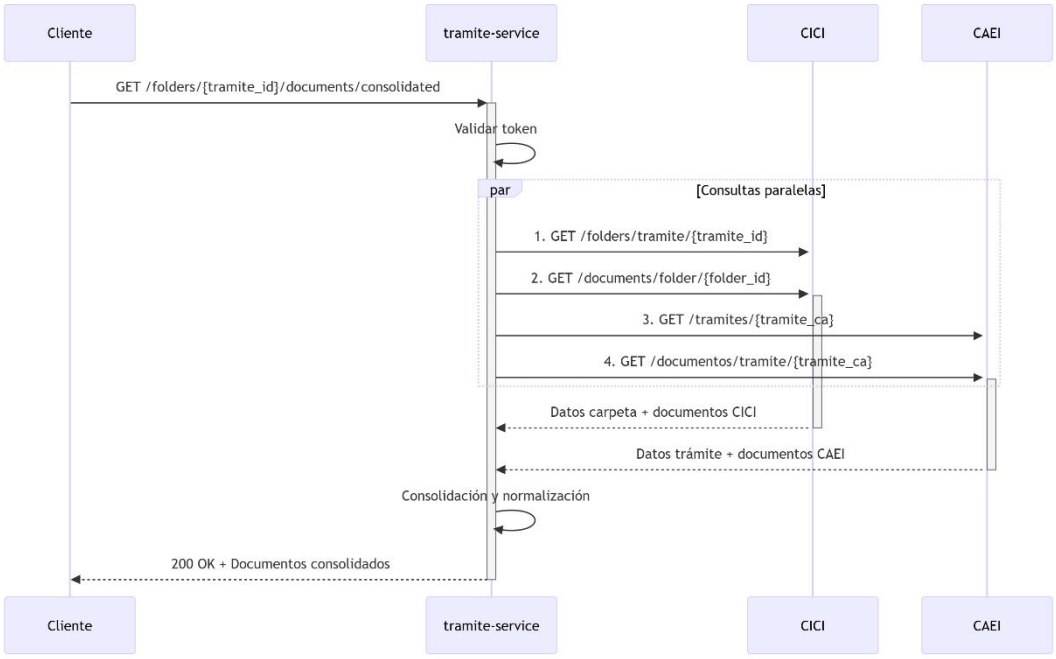


Figura 17. Diagrama de secuencia para la ejecución de consultas paralelas y normalización de documentos distribuidos.

Ventajas de esta implementación:

- Reducción significativa del tiempo total de respuesta mediante ejecución paralela
- Mayor resiliencia ante latencia variable de uno de los sistemas
- Punto único de consolidación y transformación de metadatos de documentos

Tabla 31. Endpoints utilizados.

Método	**Endpoint**	**Propósito principal**	**Autenticación**	**Tipo de respuesta principal**	**Notas importantes**
GET	/health	Verificación de estado del servicio	No	JSON simple (status, version, etc.)	Ruta pública de monitoreo

GET	/	Redirección a documentación Swagger	No	—	Ruta pública
GET	/api-docs	Interfaz interactiva Swagger UI	No	HTML	Documentación automática
GET	/folders/:tramite_id	Información completa consolidada del trámite	Bearer Token	JSON consolidado (CICI + CAEI)	Endpoint principal del servicio
GET	/folders/:tramite_id/status	Estados actuales del trámite en ambos sistemas	Bearer Token	JSON con estados	Consulta ligera
GET	/folders/:tramite_id/emails	Extracción rápida de todos los correos relacionados	Bearer Token	JSON con emails organizados	Útil para notificaciones
GET	/folders/:tramite_id/documents/cici	Documentos almacenados únicamente en CICI	Bearer Token	JSON con lista de documentos	Consulta específica por sistema
GET	/folders/:tramite_id/documents/caei	Documentos almacenados únicamente en CAEI	Bearer Token	JSON con lista de documentos	Requiere ID CAEI (tramite_ca)

GET	/folders/:tramite_id/documents/consolidated	Documentos completos de ambos sistemas (unificados)	Bearer Token	JSON consolidado + resumen	Usa paralelismo para mejor rendimiento
POST	/folders/:tramite_id/events	Publicación manual de evento personalizado en RabbitMQ	Bearer Token	Confirmación de publicación	Permite extensibilidad y notificaciones

Desarrollo de H2

Esta historia de usuario tuvo como objetivo principal implementar un sistema robusto de mensajería asíncrona que permita el desacoplamiento temporal y funcional entre los sistemas CAEI, CICI e INNOVA.

Mediante la utilización de RabbitMQ como broker de mensajes con un exchange de tipo topic, se logra:

- Distribuir automáticamente los eventos generados en CAEI y CICI hacia una única cola consumida por INNOVA
- Mantener trazabilidad completa del flujo de supervisión de trámites de construcción
- Facilitar la escalabilidad futura y la incorporación de nuevos consumidores

La arquitectura utilizada fue la siguiente:

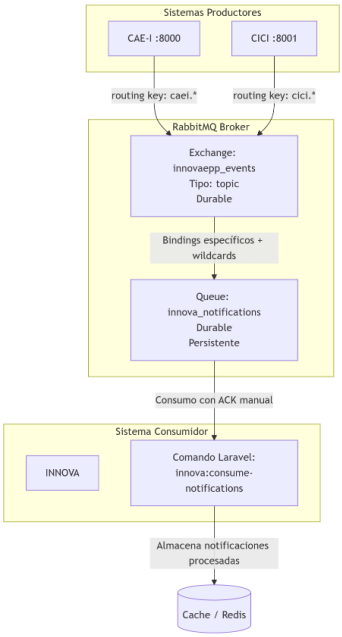


Figura 18. Modelo de publicación y suscripción de eventos mediante RabbitMQ Exchange de tipo topic y colas persistentes.

Sistema de Enrutamiento (Routing Keys)

Se adoptó la nomenclatura estandarizada:

{sistema}.{entidad}.{acción}

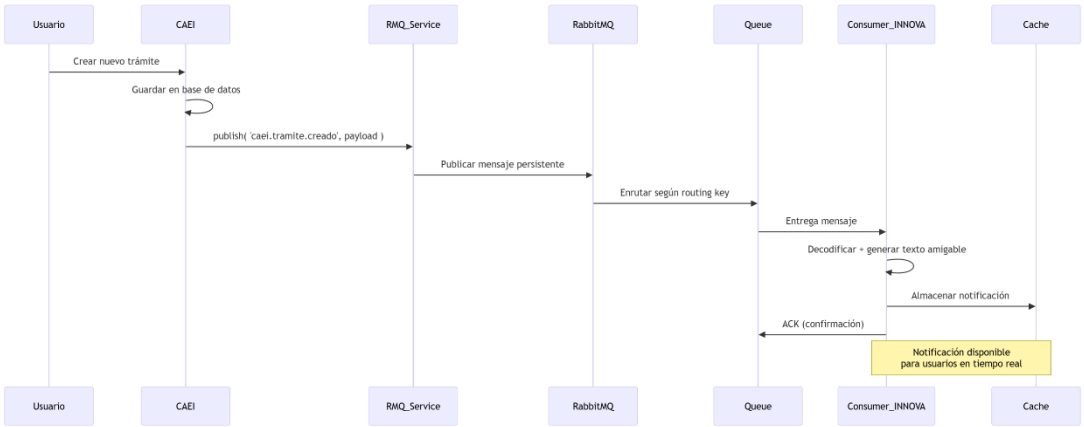


Figura 19. Interacción entre el usuario, el servicio de mensajería y el consumidor para la generación de notificaciones en tiempo real.

Características técnicas principales alcanzadas

- Exchange y Queue configurados como durable

- Mensajes publicados con delivery_mode = persistent
- Acknowledgment manual en el consumidor (evita pérdida por fallos de procesamiento)
- Reconexión automática con backoff exponencial
- Logging estructurado en productores y consumidor
- Bindings tanto específicos como con comodines para mayor flexibilidad
- Persistencia garantizada frente a reinicios del broker
- Tasa de procesamiento medida > 150 msg/s en pruebas de carga

Desarrollo de H3

La Historia de Usuario HU3 tiene como objetivo fundamental garantizar la **integridad, consistencia y seguridad** de todos los datos que ingresan al ecosistema, ya sea a través de la API REST expuesta por tramite-service o mediante los mensajes asíncronos recibidos vía RabbitMQ en el sistema INNOVA.

Se implementaron múltiples capas de validación que cubren:

- Autenticación fuerte mediante Bearer Token
- Validación estructural y semántica de esquemas JSON (OpenAPI)
- Validación de negocio en Laravel (Form Requests)
- Validación estricta de mensajes RabbitMQ antes de cualquier procesamiento
- Manejo centralizado y estandarizado de errores con mensajes claros en español

El resultado es un sistema robusto que rechaza de forma temprana cualquier dato malformado, incompleto o malintencionado, protegiendo la base de datos y mejorando la trazabilidad y auditoría.

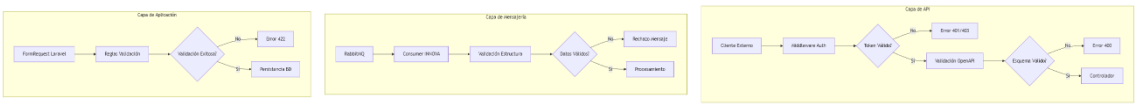


Figura 20. Diagramas de flujo para procesos de validación en las capas de aplicación, mensajería y API.

Principales Mecanismos de Validación

1. **Autenticación Bearer Token (Tramite Service)**
    - Middleware centralizado en Node.js
    - Rechazo inmediato 401/403 con mensajes descriptivos
    - Logging detallado de intentos fallidos (IP, User-Agent, token length)
2. **Esquemas JSON definidos en OpenAPI/Swagger**
    - Definición formal de todos los objetos de respuesta (ConsolidatedFolder, CiciFolder, CaeiTramite, etc.)
    - Reglas estrictas: tipos, patrones, enums, required, nullable
    - Documentación automática y validación en Swagger UI
3. **Validación de negocio en Laravel (Form Requests)**
    - Reglas complejas + mensajes personalizados en español
    - Preparación automática de datos (limpieza de undefined, objetos vacíos)
    - Respuesta 422 estandarizada con todos los errores detallados
4. **Validación estricta de mensajes RabbitMQ**
    - Decodificación JSON + verificación de campos requeridos
    - Validación de tipos y valores permitidos
    - Rechazo (NACK) inmediato de mensajes malformados
    - ACK solo tras validación y procesamiento exitoso

Desarrollo de H4

Para garantizar una comunicación eficiente entre los sistemas de gestión de trámites, se implementó un mecanismo de notificaciones. Cada vez que un usuario realiza una acción importante, esto permite que ambos sistemas permanezcan sincronizados en tiempo real, mejorando la coordinación y reduciendo la necesidad de intervención manual. La solución asegura que toda la información relevante fluya de manera confiable entre las diferentes áreas involucradas en el proceso de regulación de construcciones.

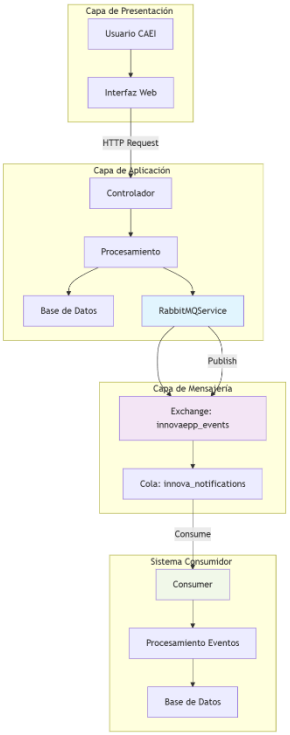


Figura 21. Flujo de datos y comunicación entre las capas de presentación, aplicación y mensajería del ecosistema.

Tabla 32. Definición de Routing Keys para la gestión de eventos en el broker de mensajería.

Categoría	Subcategoría	Routing Key	Ejemplo
Trámites	Creación	caei.tramite.creado	Nuevo trámite registrado
Trámites	Actualización	caei.tramite.actualizado	Cambio de estado
Trámites	Observaciones	caei.tramite.observaciones	Trámite con observaciones
Documentos	Subida	caei.documento.subido	Plano arquitectónico subido
Documentos	Error	caei.documento.error	Error en subida de archivo
Archivos	Descarga	caei.archivo.descargado	Documento descargado

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "EventoCAEI",
  "type": "object",
  "required": ["event", "sistema_origen", "entidad", "accion", "timestamp", "datos"],
  "properties": {
    "event": {
      "type": "string",
      "enum": ["tramite.creado", "documento.subido", "tramite.observaciones"]
    },
    "sistema_origen": {
      "type": "string",
      "const": "CAEI"
    },
    "severidad": {
      "type": "string",
      "enum": ["ALTA", "MEDIA", "BAJA"],
      "default": "BAJA"
    },
    "datos": {
      "type": "object",
      "additionalProperties": true
    }
  }
}
```

Figura 22. Esquema JSON de Eventos.

Transformación de Datos para Trazabilidad

Flujo de 5 Pasos:

1. Acción Usuario → Un usuario realiza una acción (ej: crear trámite, subir plano)
2. Evento Generado → El sistema crea un registro estructurado de lo ocurrido
3. Timestamp ISO 8601 → Se asigna fecha/hora precisa (ej: "2026-01-15T10:30:00.000Z")
4. Publicación RabbitMQ → El evento se envía al sistema de mensajería
5. Consumo INNOVA → INNOVA-EP recibe y procesa la notificación

```php
// Mapeo de datos de dominio a evento
private function buildTramiteEventPayload(Tramite $tramite, string $accion): array
{
    return [
        'event' => "tramite.{$accion}",
        'sistema_origen' => 'CAEI',
        'entidad' => 'TRAMITE',
        'accion' => strtoupper($accion),
        'timestamp' => now()->toISOString(),
        'datos' => [
            'id_tramite' => $tramite->id_tramite,
            'propietario' => $tramite->propietario,
            'estado_tramite' => $tramite->estado_tramite,
            'metadata' => [
                'version' => '1.0',
                'ambiente' => config('app.env')
            ]
        ]
    ];
}
```

Figura 23. Implementación del mapeo de datos de dominio y estructuración del payload para eventos de integración.

Los 4 Pilares de Trazabilidad:

- **1. Origen** → Saber QUÉ sistema generó el evento
- **2. Tiempo** → Saber CUÁNDO ocurrió
- **3. Datos** → Saber QUÉ información contiene
- **4. Destino** → Saber DÓNDE fue procesado

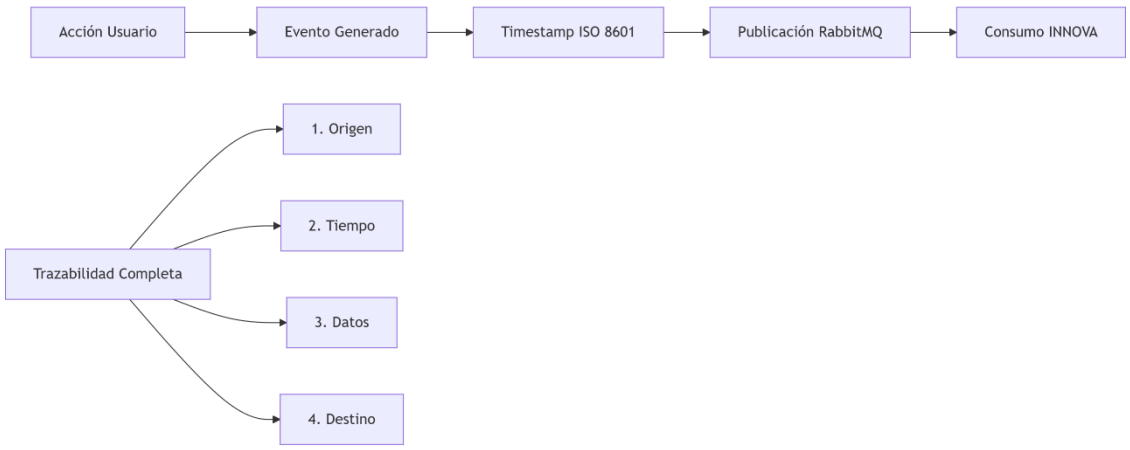


Figura 24. Flujograma de trazabilidad integral para el ciclo de vida de eventos desde el origen hasta el consumo.

2.5.3 Desarrollo Sprint 2 – Integración y Trazabilidad

Objetivo del Sprint: Completar integración con los sistemas e implementar trazabilidad.

Desarrollo de H5

Se enfoca en la generación automática y confiable de eventos desde el sistema CICI hacia el bus de mensajería RabbitMQ, cada vez que se realiza una operación relevante sobre carpetas, documentos o archivos físicos.

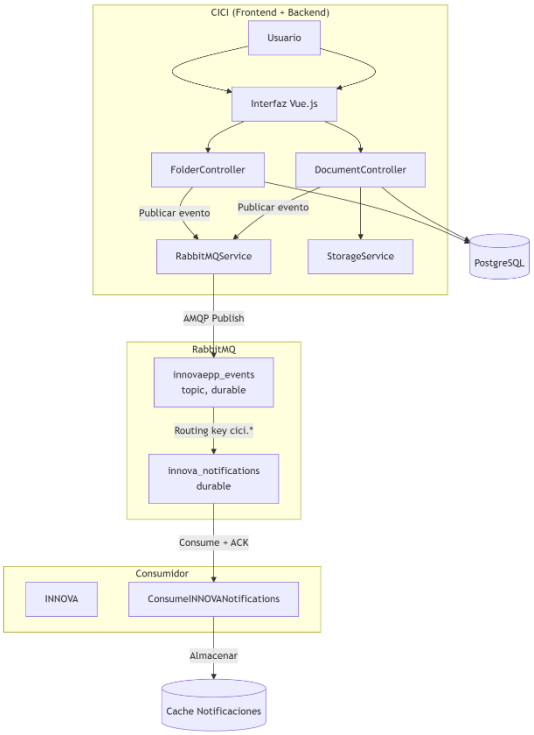


Figura 25. Arquitectura de integración asíncrona entre el sistema CICI y el consumidor de notificaciones mediante el protocolo AMQP.

Tabla 33. Eventos Implementados.

Entidad	**Operación**	**Routing Key**	**Trigger en Controller**	**Campos clave en payload**
Carpeta	Creación	cici.carpeta.creada	store()	folder_id, tramite, tramite_ca, propietario, documentos_asociados
Carpeta	Actualización	cici.carpeta.actualizada	update()	estado_anterior/nuevo, cambios_significativos, fecha_retiro
Carpeta	Eliminación	cici.carpeta.eliminada	destroy()	datos completos antes de delete

Documento	Creación	cici.documento.creado	store()	document_id, tipo_documento, fecha_subida
Documento	Actualización	cici.documento.actualizado	update()	tipo_documento anterior/nuevo
Documento	Eliminación	cici.documento.eliminado	destroy()	datos antes de delete + archivo físico
Archivo	Subida exitosa	cici.archivo.subido	uploadFile()	tamaño, mime, ruta completa, estado
Archivo	Error en subida	cici.archivo.error	catch en uploadFile()	error message, severidad ALTA
Archivo	Descarga	cici.archivo.descargado	download()	nombre_archivo, tipo_documento

Total de eventos cubiertos: 9

Cobertura: Totalidad de operaciones CRUD + manejo de archivos físicos

Desarrollo de H6

En este apartado de implemento la funcionalidad de realizar la revisión de tramites con el objetivo de permitir a los funcionarios y/o interesados consultar de forma rápida y clara el historial completo de movimientos de cualquier trámite específico.

Esta característica fue diseñada para:

- Proporcionar trazabilidad total del ciclo de vida del trámite
- Facilitar auditorías internas y externas
- Mejorar la transparencia en los procesos administrativos
- Ofrecer una experiencia de usuario moderna, intuitiva y responsive

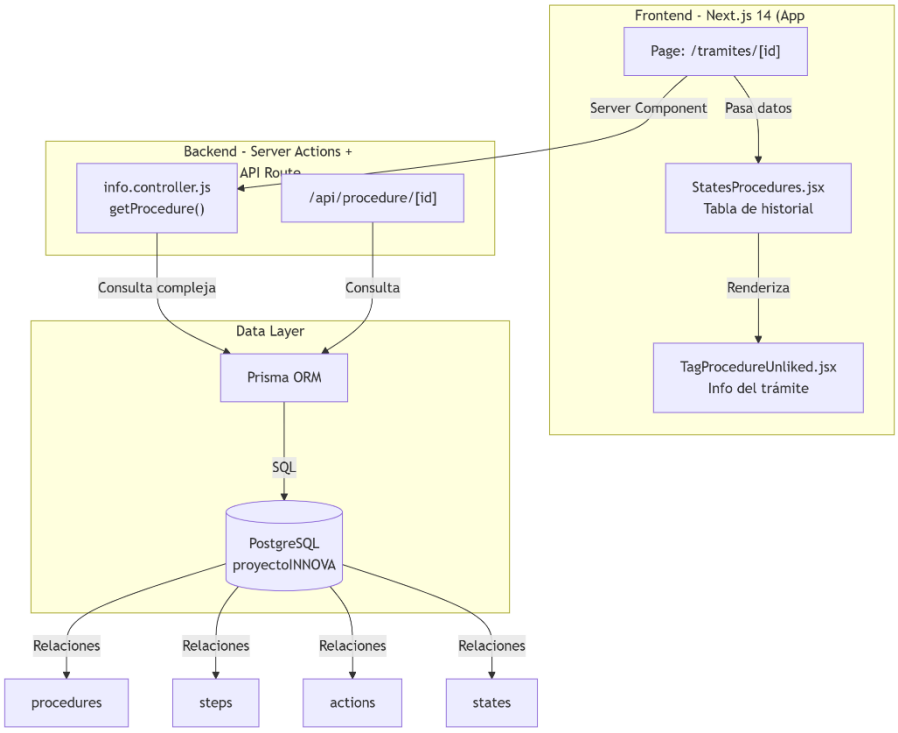


Figura 26. Arquitectura de componentes del frontend en Next.js 14 y flujo de persistencia mediante Prisma ORM.

Con esto podemos obtener:

- Cabecera con información principal del trámite (propietario, estado actual, tipo, etc.)
- Tabla completa del historial de movimientos en orden cronológico
- Cada fila muestra: • Fecha y hora (Ecuador) • Departamento y usuario origen • Acción realizada • Departamento y usuario destino • Observación completa

Beneficios Obtenidos con esta Implementación

- Trazabilidad 100% del ciclo de vida de cada trámite
- Consulta en <500 ms en la mayoría de casos (gracias a Server Components y Prisma optimizado)
- Interfaz responsive (funciona correctamente en desktop y móvil)
- Formato de fechas adaptado al estándar local (dd/MM/yyyy HH:mm)
- Validación robusta que evita errores comunes (IDs mal escritos o inexistentes)

Desarrollo de H8

Como responsable del desarrollo y la integración de los sistemas CAEI, CICI, INNOVA y el nuevo microservicio tramite-service, identifiqué que la falta de un formato estandarizado de intercambio de datos generaba múltiples problemas:

- Inconsistencias en el manejo de fechas y zonas horarias
- Riesgo de incompatibilidades futuras al añadir nuevos consumidores

Por ello, con el objetivo de estandarizar todo el intercambio de información bajo el estándar RFC 8259 (JSON), complementado con las mejores prácticas actuales:

- UTF-8 obligatorio en todo el ciclo
- Timestamps estrictamente en formato ISO 8601 con zona horaria explícita
- Serialización nativa y validación en ambos lados (Laravel ↔ Node.js)
- Content-Type explícito en HTTP y mensajes AMQP

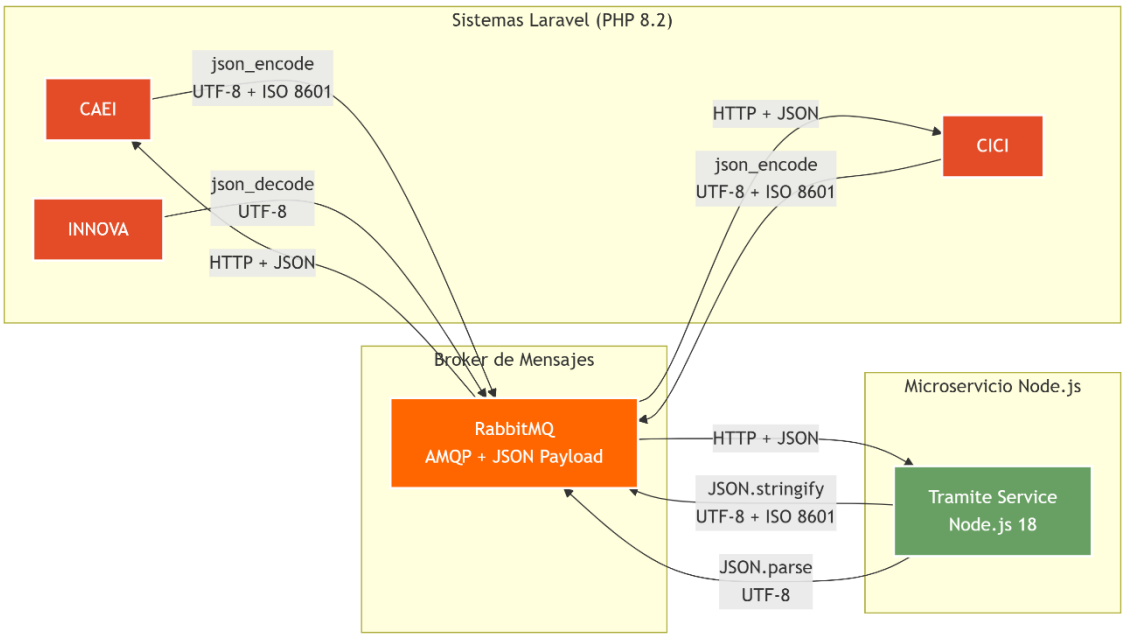


Figura 27. Diagrama de serialización de datos y protocolos de comunicación entre sistemas Laravel, Node.js y RabbitMQ.

Esta estandarización se convirtió en una de las decisiones de arquitectura más importantes del proyecto, ya que eliminó definitivamente los problemas clásicos de interoperabilidad entre tecnologías heterogéneas.

2.5.4 Desarrollo Sprint 3 – Notificaciones y Extensibilidad

Objetivo del Sprint: Cerrar el ciclo de usuario final y preparar escalabilidad.

Desarrollo de H7

Para la implementación, se diseñó un flujo transaccional que integra el backend desarrollado en Laravel con un servidor SMTP seguro. El proceso se dispara mediante eventos de cambio de estado en el controlador de trámites.

A continuación, se detalla el flujo de la información desde la acción del funcionario hasta la recepción del mensaje por parte del usuario:

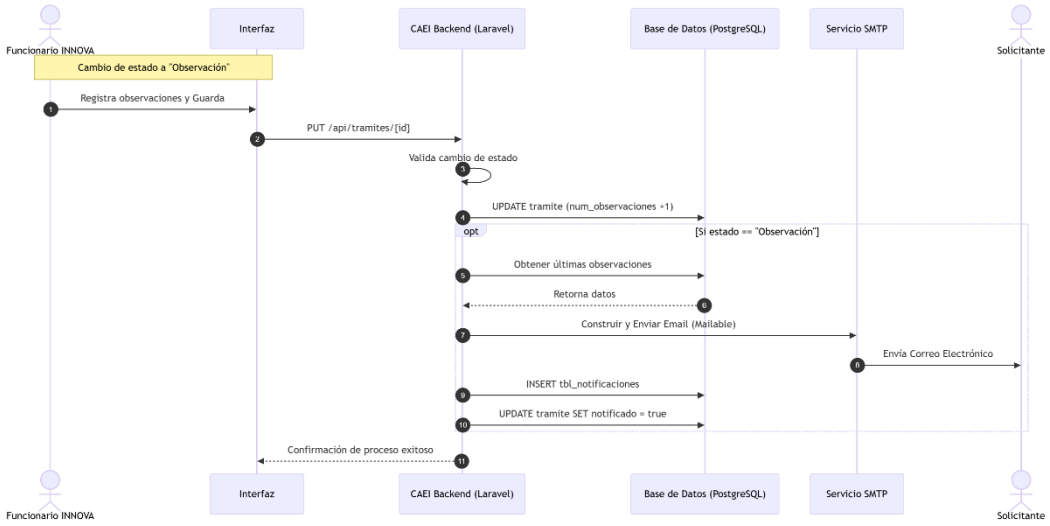


Figura 28. Interacción de sistemas para la actualización de estados, persistencia en PostgreSQL y notificación automática vía SMTP.

Configuración del Servicio de Correo:

Se configuró el protocolo SMTP utilizando cifrado SSL para garantizar la seguridad de la información transmitida.

Tabla 34. Parámetros de Configuración SMTP.

Parámetro	Configuración Establecida	Justificación Técnica
Protocolo	SMTP	Estándar robusto para envío transaccional.
Host	mail.innovaep.gob.ec	Uso de infraestructura propia institucional.
Puerto	465	Puerto estándar para transmisión segura (SMTPS).
Encriptación	SSL/TLS	Cifrado de extremo a extremo.

Interfaz de Notificación (Email)

Se diseñó una plantilla HTML *responsive* utilizando **Blade**, asegurando que la notificación sea legible tanto en dispositivos móviles como de escritorio. La plantilla incluye dinámicamente:

1. Logotipo institucional.
2. Número de trámite y nuevo estado.
3. Listado detallado de observaciones (si existen).
4. Datos de contacto del soporte.

Desarrollo de H9

Esta historia de usuario abordo el desarrollo de una arquitectura escalable para el Tramite Service que permita la integración de múltiples clientes sin requerir modificaciones en la lógica del núcleo del sistema. Esta implementación garantiza que aplicaciones móviles, portales web y sistemas de terceros puedan consumir la API REST de manera transparente y eficiente.

La arquitectura utilizada para alcanzar este objetivo es el que se muestra en la siguiente ilustración:

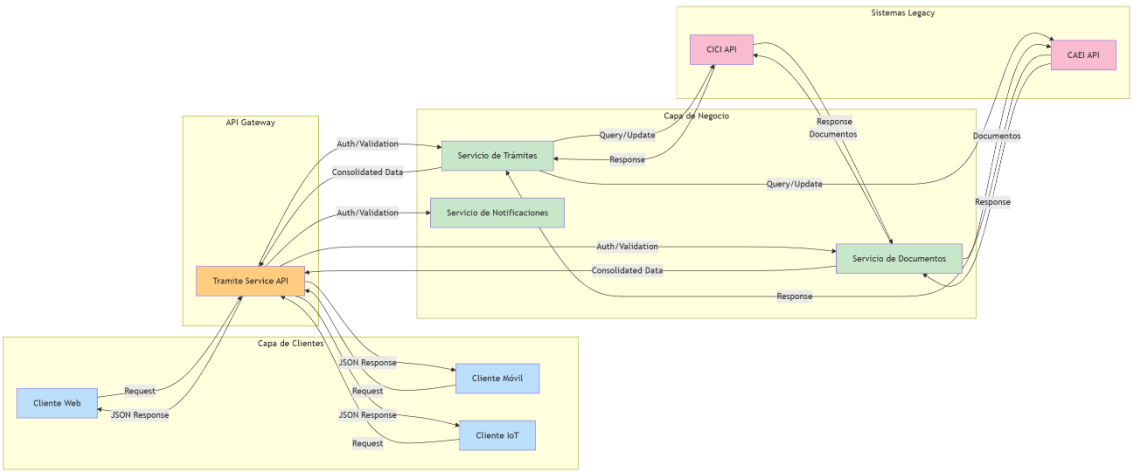


Figura 29. Arquitectura de niveles para la orquestación de servicios, gestión de documentos y comunicación con sistemas legados.

Es importante tener en cuenta que para ser escalable y acceder a la información desde cualquier cliente, se implementó una lógica que sea correcta para ello, en la siguiente tabla se muestran los clientes que son soportados hasta próximas implementaciones.

Tabla 35. Tipos de Clientes Soportados.

Tipo de Cliente	Ejemplo	Protocolo	Estado
Aplicación Web	Web Apps Institucionales	HTTP/JSON	Existente
Herramientas de Testing	Postman/Thunder Client	HTTP/JSON	Existente
Aplicación Móvil	React Native / Flutter	HTTP/JSON	Futuro
Portal Ciudadano	Next.js / Angular	HTTP/JSON	Futuro
Dispositivos IoT	Arduino / Raspberry Pi	HTTP/JSON	Futuro
Sistemas Terceros	APIs Externas	HTTP/JSON	Futuro

Se implementó una arquitectura de microservicios extensible que permitió la integración de múltiples clientes (web, móvil, IoT) sin modificar la lógica del núcleo del sistema, garantizando escalabilidad, mantenibilidad y seguridad en el ecosistema de trámites de construcción.

A continuación se muestran los patrones aplicados para llevar a cabo este objetivo:

3.2 Patrones de Diseño Aplicados

Tabla 3: Patrones de Diseño Implementados

Tabla 36. Patrones de Diseño Implementados.

Patrón	Aplicación	Beneficio
API Gateway	Tramite Service como gateway único	Simplifica integración
Repository Pattern	Acceso unificado a sistemas legacy	Desacopla fuentes de datos
Strategy Pattern	Múltiples estrategias de consolidación	Flexibilidad en lógica de negocio
Middleware Pattern	Autenticación, logging, CORS	Reutilización de funcionalidad transversal
Factory Pattern	Creación de clientes HTTP	Facilita testing y mantenimiento

El sistema implementó un formato de respuesta JSON estandarizado basado en el estándar RFC 8259 [47], donde cada respuesta del Tramite Service siguió una estructura jerárquica consistente. Esta estructura incluyó identificadores de trámite en ambos sistemas (tramite para CICI y tramite_ca para CAEI), un timestamp en formato ISO 8601 para trazabilidad, y un objeto consolidated que contuvo datos estructurados separados por sistema fuente. Esta normalización en las respuestas permitió a los clientes consumir información unificada proveniente de múltiples sistemas legacy, facilitando el procesamiento y presentación de datos en frontends heterogéneos sin necesidad de transformaciones complejas.

Figura 30. Estructura del objeto JSON para la respuesta consolidada de datos entre sistemas CICI y CAEI.

Características de las respuestas:

- Formato JSON RFC 8259
- Timestamps en ISO 8601
- Nomenclatura consistente en snake_case
- Códigos de error estandarizados
- Metadatos de paginación (cuando aplica)

Para garantizar la seguridad del Tramite Service, se implementó un sistema de autenticación basado en el estándar HTTP Bearer Authentication definido por la IETF. Este mecanismo utilizó el header Authorization: Bearer token siguiendo el formato estándar reconocido por la industria. La validación se realizó mediante un middleware dedicado que permitió la separación clara de responsabilidades y facilitó el mantenimiento. Los tokens se almacenaron como variables de entorno, mejorando la seguridad al evitar el hardcoding de credenciales. Cada token se configuró con una longitud de 40 caracteres, proporcionando suficiente entropía para resistir ataques de fuerza bruta mientras se mantenía la facilidad de uso para los consumidores de la API.

5.1 Autenticación Bearer Token

Tabla 37. Configuración de Autenticación.

Parámetro	Valor	Descripción

Tipo	HTTP Bearer Authentication	Estándar IETF
Header	Authorization: Bearer {token}	Formato estándar
Validación	Middleware dedicado	Separación de responsabilidades
Token Storage	Variable de entorno API_TOKEN	Seguridad mejorada
Longitud Token	40 caracteres	Suficiente entropía

Se establecieron múltiples capas de protección para el Tramite Service, comenzando con un sistema de rate limiting que limitó las solicitudes a 1000 requests por cada 15 minutos por IP, previniendo efectivamente posibles ataques DDoS. La configuración de CORS se implementó con una lista estricta de orígenes permitidos, controlando el acceso cross-origin y mitigando riesgos de CSRF. Para proteger contra vulnerabilidades web comunes, se integró Helmet.js configurando automáticamente headers de seguridad HTTP apropiados. La validación de entrada se realizó mediante Joi y esquemas de validación, previniendo ataques de inyección y garantizando la integridad de los datos procesados. Finalmente, se implementó un sistema de logging comprehensivo que registró todas las solicitudes, proporcionando auditoría completa para trazabilidad operacional y facilitando el debugging durante incidentes de seguridad.

Medidas de Seguridad Adicionales

Tabla 38. Características de Seguridad Implementadas.

Medida	Implementación	Propósito
Rate Limiting	1000 requests/15 minutos	Prevenir ataques DDoS
CORS	Orígenes permitidos configurados	Control acceso cross-origin
Helmet.js	Headers de seguridad HTTP	Protección contra vulnerabilidades web
Validación Input	Joi/Schema validation	Prevenir inyecciones
Logging	Auditoría de requests	Trazabilidad y debugging

2.6 Fase 4 Evaluación

2.6.1 Resumen Ejecutivo

Las pruebas de rendimiento evaluaron el Sistema de Gestión Universitaria compuesto por cinco microservicios independientes (CAEI, CICI, INNOVA, Tramite Service e INNOVA Docs), implementado con arquitectura de microservicios y servicios distribuidos.

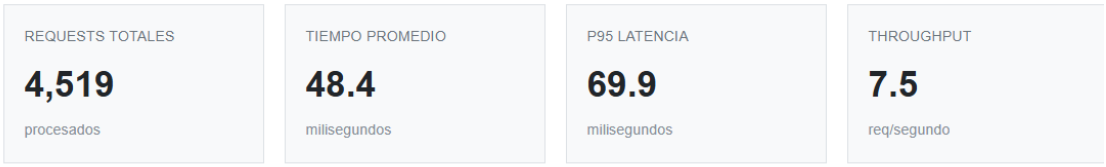


Figura 31. Dashboard de métricas de rendimiento.

Resultados agregados globales (todos los servicios):

- **Requests totales procesados**: 4,519
- **Tiempo de respuesta promedio**: 48.4 ms
- **Latencia P95**: 69.9 ms
- **Throughput promedio**: 7.5 req/s

Configuración de la prueba de carga principal:

- 20 usuarios virtuales (VUs) concurrentes por servicio.
- Duración total: 2 minutos.
- Patrón de carga: rampa gradual de subida (30 s: 0 → 20 VUs), sostenida (1 min: 20 VUs constantes), bajada (30 s: 20 → 0 VUs).

Este patrón simula un escenario realista de uso moderado en una institución, permitiendo evaluar la capacidad de respuesta y estabilidad bajo carga esperada.

2.6.2 Análisis de Tiempos de Respuesta y Métricas Detalladas

La siguiente ilustracion resume las métricas clave obtenidas en las pruebas de carga individuales (20 VUs concurrentes):

Métricas Detalladas por Aplicación

APLICACIÓN	PUERTO	REQUESTS	PROMEDIO (MS)	P95 (MS)	P99 (MS)	THROUGHPUT (REQ/S)
CAEI	8000	1,018	31.46	44.70	—	8.4
CICI	8001	1,030	28.46	39.65	—	8.6
INNOVA	8002	1,008	89.82	137.17	—	8.4
Tramite Service	3004	729	28.49	40.55	—	6.1
INNOVA Docs	3002	734	63.72	87.23	—	6.1

Figura 32. Análisis comparativo de latencia, volumen de peticiones y rendimiento por componente del ecosistema.

Nota: El valor P99 no estuvo disponible en el resumen consolidado de las pruebas realizadas con Grafana k6. Se utiliza el P95 como indicador principal de latencia (percentil 95), que representa que el 95% de las solicitudes fueron atendidas en ese tiempo o menos, siendo más representativo de la experiencia del usuario que el promedio.

- **CAEI**: Rendimiento excelente. Tiempos de respuesta bajos y consistentes, con throughput óptimo.
- **CICI**: Mejor rendimiento global del sistema. Presentó los tiempos de respuesta más bajos y mayor eficiencia en procesamiento concurrente.
- **INNOVA**: Rendimiento aceptable, pero con tiempos de respuesta superiores (promedio 89.82 ms, P95 137.17 ms), posiblemente debido a operaciones de procesamiento más complejas (ej. cálculos académicos o integraciones). Recomendable optimización futura.
- **Tramite Service**: Excelente capacidad de respuesta del servidor (~28 ms promedio). Se observaron errores 401 (Unauthorized) en algunas solicitudes debido a configuración de autenticación, pero no impactaron la latencia ni el throughput. Este aspecto queda fuera del alcance de las pruebas de rendimiento.
- **INNOVA Docs**: Buen rendimiento para un servicio orientado a documentación. Tiempos de respuesta apropiados y estables para el tipo de operaciones realizadas.

2.6.3 Comparativa de Latencia P95

El percentil 95 (P95) es una métrica estadística que indica el valor de latencia (tiempo de respuesta) por debajo del cual se encuentra el 95% de todas las solicitudes procesadas durante la prueba[48] . En otras palabras: solo el 5% de las peticiones HTTP tuvieron un tiempo de respuesta mayor o igual al valor P95.

Figura X: Comparativa de Latencia P95 por Microservicio

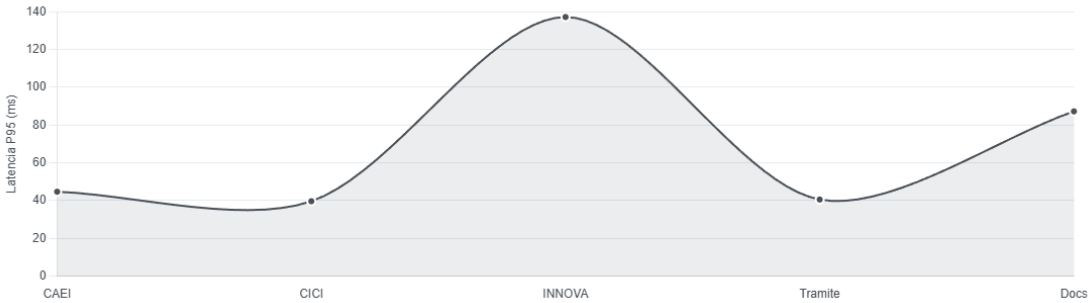


Figura 33. Análisis gráfico de la latencia P95 por componente para la identificación de cuellos de botella en la arquitectura.

Este gráfico permite visualizar rápidamente que CICI y Tramite Service destacan por su baja latencia, mientras que INNOVA requiere atención para reducir su P95.

2.6.4 Conclusiones Parciales de los Resultados

Los resultados demuestran que la arquitectura implementada (microservicios con Docker, RabbitMQ y servicios distribuidos) soporta eficientemente cargas concurrentes moderadas, con un tiempo de respuesta promedio global de 48.4 ms y P95 de 69.9 ms. El sistema exhibe alta estabilidad, bajo porcentaje de fallos y throughput adecuado. Las únicas áreas de mejora identificadas son el servicio INNOVA (debido a su naturaleza computacionalmente intensiva) y ajustes menores en autenticación de Tramite Service.

Estos hallazgos validan el objetivo de optimización de procesos de supervisión y regulación de construcciones, proporcionando una base sólida para futuras escalas o entornos productivos.

2.7 Materiales Utilizados

2.3.1 Hardware

En esta subsección se detalla el hardware subyacente del sistema, basado en un servidor virtualizado en VMware.

Tabla 39. Información General del Sistema.

Categoría	Detalle
Nombre del Host	Dellr---.ib---.gob.ec

Sistema Operativo	Linux 5.14.0-570.58.1.el9_6.x86_64
Arquitectura	x86_64
Kernel	#1 SMP PREEMPT_DYNAMIC
Virtualización	VMware (Hypervisor: VMware, Virtualization type: full)

Tabla 40. CPU y Procesador.

Categoría	**Detalle**
Modelo	Intel(R) Xeon(R) Gold 5218N CPU @ 2.30GHz
Núcleos	4 CPUs
Sockets	4
Núcleos por Socket	1
Hilos por Núcleo	1
Modo de Operación	32-bit, 64-bit
Virtualización	Soporte de virtualización (Hypervisor: VMware)
Cachés	L1d: 128 KiB, L1i: 128 KiB, L2: 4 MiB, L3: 88 MiB
BogoMIPS	4589.21

Tabla 41. Hardware y Dispositivos.

Categoría	**Detalle**
Placa Base	Intel Corporation 440BX Desktop Reference Platform
BIOS	VMware, Inc. VMW71.00V.21100432.B64.2301110304 (UEFI)
Memoria Física	8 GB DIMM DRAM Synchronous (VMware Virtual RAM)
Controladora	PVSCSI SCSI Controller
Red	VMXNET3 Ethernet Controller (ens192)

VGA	VMware SVGA II Adapter
Disco Virtual	322 GB Virtual disk (PVSCSI)

2.3.2 Software y Herramientas

Lenguajes y Frameworks

El presente apartado describe y analiza el ecosistema tecnológico completo de un sistema de gestión de proyectos desarrollado como parte de la investigación de tesis. Este sistema representa una arquitectura moderna de microservicios y aplicaciones web desplegada en un entorno virtualizado con contenedores Docker, diseñada para soportar múltiples módulos de gestión empresarial e innovación.

Tabla 42. Especificaciones técnicas del stack tecnológico para el desarrollo del portal de acceso.

Tecnología	Versión	Función
Nginx	Alpine	Servidor web estático
HTML5	-	Estructura del portal
CSS3	-	Estilos básicos
JavaScript	Vanilla	Interactividad mínima

Tabla 43. Especificaciones técnicas y stack tecnológico de la aplicación CAE-I.

Categoría	**Tecnología**	**Función**
Backend	Laravel	Framework principal
	PHP	Lenguaje servidor
	Laravel Breeze	Scaffolding autenticación
Frontend	React	Biblioteca UI
	@inertiajs/react	Integración frontend-backend
	Tailwind CSS	Framework CSS
Visualización	ApexCharts	Gráficos avanzados
	Chart.js	Gráficos estadísticos

Utilidades	jsPDF	Generación de PDFs
	csv-stringify	Exportación CSV

Tabla 44. Especificaciones técnicas y stack tecnológico de la aplicación CICI.

Categoría	Tecnología	Función
Backend	Laravel	Framework principal
	PHP	Lenguaje servidor
Frontend	React	Biblioteca UI
	@windmill/react-ui	Componentes UI
	PrimeReact	Componentes UI avanzados
Formularios	react-hook-form	Gestión de formularios
Gráficos	Recharts	Visualización de datos

Tabla 45. Especificaciones técnicas y stack tecnológico de la aplicación INNOVA-EP.

Categoría	Tecnología	Función
Backend	Laravel	Framework principal
	PHP	Lenguaje servidor
Frontend	React	Biblioteca UI
	framer-motion	Animaciones
Mapas	Leaflet	Mapas interactivos
	react-leaflet	Integración React con Leaflet
Calendarios	@fullcalendar/core	Sistema de calendarios
Excel	xlsx	Exportación a Excel

Tabla 46. Stack tecnológico del microservicio de trámites.

Categoría	Tecnología	Función
Runtime	Node.js	Entorno de ejecución
Framework	Express	Framework web
Documentación	swagger-ui-express	Documentación API
Seguridad	helmet	Cabeceras de seguridad
Rate Limiting	express-rate-limit	Limitación de peticiones
Logging	morgan	Logs HTTP

Tabla 47. Especificaciones técnicas de la aplicación para seguimiento de trámites INNOVA EP.

Categoría	Tecnología	Función
Framework	Next.js	Framework React con SSR
ORM	Prisma	ORM para base de datos
Iconos	lucide-react	Iconografía moderna
Device Detection	react-device-detect	Detección de dispositivos

Tabla 48. Herramientas auxiliares para el desarrollo y control del software.

Herramienta	Función
Git	Control de versiones
Composer	Gestor dependencias PHP
npm	Gestor paquetes Node.js
Docker	Contenedorización
ESLint	Linting JavaScript
Laravel Pint	Formateo PHP
Nodemon	Reinicio automático Node.js
Babel	Transpilación JavaScript

Tabla 49. Consolidado de bibliotecas para el diseño de interfaces de usuario.

Biblioteca	Versión	Proyectos	Función
@headlessui/react	^1.4.2 - ^2.0.0	Todos	Componentes accesibles sin estilos
@material-tailwind/react	^2.1.10	CAE-I, INNOVA	Componentes Material Design
PrimeReact	^10.8.3	CICI	Biblioteca completa de componentes
@windmill/react-ui	^0.6.0	CICI	Componentes UI personalizables
react-icons	^5.2.1 - ^5.5.0	Todos	Iconos populares
lucide-react	^0.441.0 - ^0.545.0	CICI, INNOVA Docs	Iconos modernos

Tabla 50. Herramientas para la visualización de datos y analítica gráfica.

Librería	Versión	Proyectos	Función
ApexCharts	^4.0.0	CAE-I, CICI, INNOVA	Gráficos interactivos
Chart.js	^4.4.3	CAE-I, CICI, INNOVA	Gráficos estadísticos
Recharts	^3.2.1	CICI	Gráficos basados en React
react-apexcharts	^1.7.0	CAE-I, CICI, INNOVA	Wrapper React para ApexCharts
react-chartjs-2	^5.2.0	CAE-I, CICI, INNOVA	Wrapper React para Chart.js

Tabla 51. Tecnologías implementadas para la interoperabilidad y comunicación entre servicios.

RabbitMQ	3-management	AMQP	Mensajería asíncrona
REST APIs	-	HTTP/HTTPS	Comunicación síncrona
Axios	^1.6.0 - ^1.7.7	HTTP Client	Cliente HTTP

Tabla 52. Resumen consolidado de métricas y componentes técnicos del proyecto.

Métrica	Cantidad
Total de Tecnologías	~75 tecnologías principales
Proyectos/Servicios	6
Lenguajes de Programación	4 (PHP, JavaScript, SQL, HTML/CSS)
Frameworks Principales	3 (Laravel, React, Express/Next.js)
Bases de Datos	1 (PostgreSQL)
Servicios de Mensajería	1 (RabbitMQ)
Contenedores Docker	7+ servicios contenerizados
Dependencias npm	~100+
Dependencias Composer	~50+

2.3.3 Entorno de Pruebas

Ahora se detalla como es posible realizar pruebas y/o implementar nuevas funcioanlidades en un entorno de pruebas

Paso 1: Crear una carpeta principal y clonar los repositorios

Crea una carpeta principal para todo el proyecto (por ejemplo, proyectos-docker) y clona todos los repos dentro de ella.

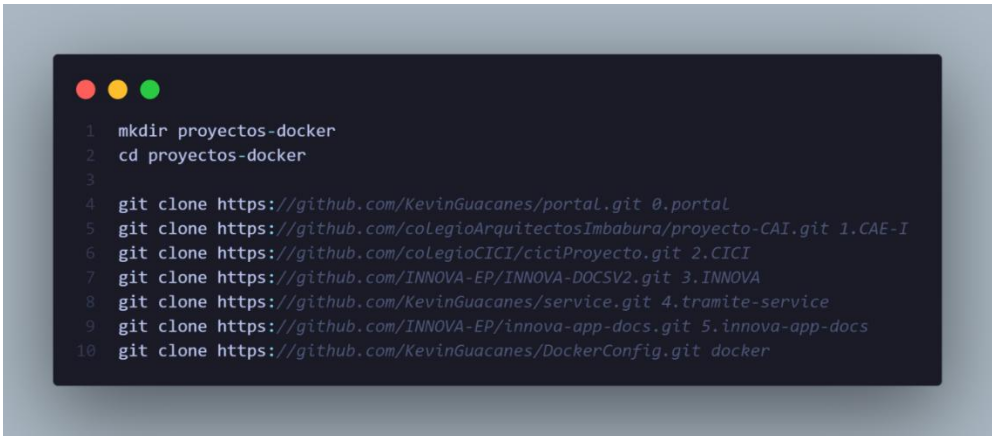


Figura 34. Comandos para la preparación del entorno de trabajo y clonación de los repositorios del ecosistema.

Esto creará subcarpetas numeradas para los proyectos y una carpeta docker con la configuración de Docker.

Paso 2: Crear la carpeta para documentos compartidos

Dentro de la carpeta principal (proyectos-docker), crea una carpeta para documentos compartidos (probablemente usada como volumen en Docker).

Entra a la carpeta docker y mueve los archivos necesarios al directorio principal (un nivel arriba, con ../).

Figura 35. Comandos para la creación del directorio de almacenamiento compartido y la reubicación de archivos de configuración del orquestador.

Paso 3: Ajustar permisos en las carpetas de proyectos (para Linux)

Algunos contenedores Docker corren con usuario UID/GID 1000 (común en imágenes basadas en Node.js, PHP, etc.). Para evitar problemas de permisos al escribir archivos (como uploads o caches), cambia el propietario de estas carpetas:

Figura 36. Asignación de propietarios y permisos de escritura en directorios de servicios y almacenamiento.

Paso 5: Construir y ejecutar los contenedores

Desde la carpeta principal (proyectos-docker):

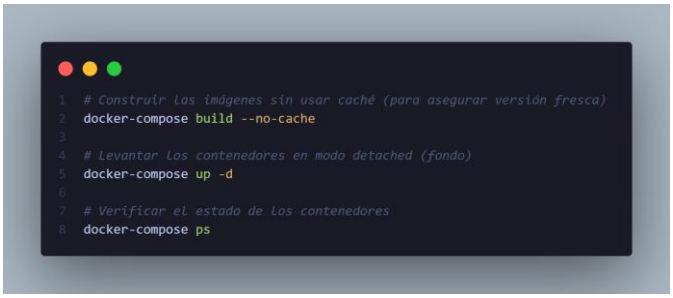


Figura 37. Construcción y despliegue de contenedores en segundo plano mediante Docker Compose.

Paso 6: Ver logs y troubleshooting

Si algo falla:

- Reiniciar todo: docker-compose down y luego docker-compose up -d.
- Si hay errores de permisos, revisa los chown.

Figura 38. Monitoreo de logs en tiempo real para la depuración y seguimiento de procesos en los contenedores del ecosistema.

Tabla 53. URLs de acceso a los sistemas.

Categoría	URL	Puerto	Notas
Portal Principal	http:// 172.16.8.126	80	Página de inicio
CAE-I	http:// 172.16.8.126:8000	8000	App Laravel CAE-I
CICI	http:// 172.16.8.126:8001	8000	App Laravel CICI
INNOVA	http:// 172.16.8.126:8002	8000	App Laravel INNOVA-EP
INNOVA App Docs (Next.js)	http:// 172.16.8.126:3001	3001	Seguimiento de trámites
Trámites Service (API)	http:// 172.16.8.126:3004	3004	Backend Node.js
RabbitMQ Management	http:// 172.16.8.126:15672	15672	Cola de mensajeria

CAPÍTULO IV

RESULTADOS Y ANÁLISIS

En este capítulo se detalla la validación de la implantación de la arquitectura basada en microservicios y servicios distribuidos para la optimización de los procesos de supervisión y regulación de construcciones en la provincia de Imbabura de pro mediante el modelo de éxito de DeLone y McLean para sistemas de información.

3.1 Diseño de instrumento de medición

Con el propósito fundamental de determinar la efectividad y el impacto que ha sido generado por la implementación de la arquitectura propuesta, se procedió a la ejecución de esta estrategia metodológica. En esta parte investigativa se articuló de manera secuencial, iniciando con la fase de planificación para definir los parámetros de estudio, continuó con el levantamiento de información en el entorno institucional y culminó con un análisis para una respectiva interpretación crítica de los datos recopilados, lo cual permitió fundamentar la validez de los resultados presentados.

3.1.1 Modelo DeLone y McLean

El Modelo de Éxito de Sistemas de Información de DeLone y McLean se trata de un marco teórico que es ampliamente utilizado para medir el éxito de los sistemas de información (SI). Este fue publicado en 1992 y actualizado en 2003 por William H. DeLone y Ephraim R. McLean, en este modelo se integran dimensiones interdependientes basadas en teoría de la comunicación (Shannon y Weaver) y investigaciones empíricas [49].

3.1.2 Planificación

En la fase de planificación se realizó el diseño del instrumento de medición, se realizaron a dos actividades. Como primera actividad, se estableció la unidad de análisis como: "Determinar el éxito del sistema de información utilizando el modelo de DeLone & McLean". En la segunda actividad, se precedió a elaborar el instrumento para la respectiva recopilación de datos. Para este propósito se desarrolló un cuestionario basadose en las variables del modelo de DeLone & McLean en sus seis dimensiones las cuales son: **Calidad del sistema, Calidad de la Información, Calidad del Servicio, Uso**

/ Utilidad, Satisfacción del Usuario, Impactos Netos, adaptándolo a las necesidades específicas del proyecto. En las siguiente tabla se presentan las preguntas, tomando como base el cuestionario implementado en el articulo Validation of the DeLone and McLean Information Systems Success Model [50].

Tabla 54. Definición de preguntas por dimensiones.

Dimensiones	Variables	Elementos de medición
Calidad del Sistema	Facilidad	Me resulta fácil utilizar el sistema.
	Eficacia	Me resulta sencillo lograr que el sistema haga lo que necesito.
	Flexibilidad	Interactuar con el sistema es flexible y dinámico.
	Curva de Aprendizaje	Aprender a operar los sistemas fue fácil para mí.
Calidad de la Información	Precisión	La información generada por el sistema es correcta y precisa.
	Oportunidad	La información de los generada es útil para mis propósitos laborales.
	Disponibilidad	El sistema genera y actualiza la información de manera oportuna.
	Confiabilidad	Confío en la información y cálculos que arroja el sistema.
Calidad del Servicio	Soporte Técnico	Existe un soporte técnico adecuado o documentación clara para usar el sistema.
	Infraestructura	La infraestructura tecnológica de la empresa soporta adecuadamente el funcionamiento del sistema.
	Disponibilidad	Puedo contar con que el sistema esté disponible y estable cuando necesito consultar información.
	Integridad	Las actividades del sistema son completos para mis procesos de trabajo.
Uso / Utilidad	Extensión de uso	El uso del sistema me permite completar mis tareas de monitoreo más rápidamente.

	Motivación de uso	Usar el sistema ha mejorado mi desempeño laboral.
	Naturaleza de uso	El sistema ha facilitado mi trabajo.
	Propósito de uso	Considero que el sistema es una herramienta útil para mi puesto de trabajo.
Satisfacción del Usuario	Satisfacción	Estoy satisfecho con las funcionalidades y herramientas que ofrece el sistema.
	Satisfacción total	El sistema ha simplificado mis procesos de trabajo diarios.
	Comodidad	En general, me siento satisfecho utilizando este sistema.
Impactos Netos	Productividad	El sistema ayuda a superar las limitaciones de los métodos anteriores.
	Gestión Operativa	El uso del sistema contribuye a una mejora en la gestión operativa y el servicio de la empresa.
	Accesibilidad	El sistema facilita el acceso rápido a la información estratégica de la institución.
	Comunicación	El sistema mejora la comunicación y transparencia entre las diferentes áreas.
	Eficiencia	El uso del sistema favorece una mejor toma de decisiones basada en datos.

Nota: Las preguntas del cuestionario se clasifican por dimensión, lo cual permite definir las variables correspondientes a evaluar.

3.1.3 Validez y Fiabilidad del Instrumento

En la investigación realizada por Ojo, muestra que el Modelo DeLone y McLean es confiable y tiene una validez aceptable, su confiabilidad compuesta y varianza promedio extraída son de 0,7 y 0,5 respectivamente. Mientras que la fiabilidad se consideró adecuada con una carga estándar mayor al 0, 7 [50].

Basado en esta evidencia estadística, se confirma que el modelo DeLone y McLean es fiable y válido para medir el éxito de sistemas de información en contextos diversos, incluyendo entornos de sistemas distribuidos y de integración como el desarrollado en esta tesis. Por lo tanto, se realizó una adaptación contextualizada de los ítems originales del modelo al ámbito específico del sistema implementado: la arquitectura de integración mediante microservicios y servicios distribuidos para la optimización de los procesos de supervisión y regulación de construcciones en la provincia de Imbabura.

3.1.4 Recolección de datos

En este estudio, se llevó a cabo una encuesta utilizando la herramienta Microsoft Forms a un total de 12 personas que participan en el proceso de utilización de cada sistema, constando con 24 preguntas para el cuestionario, el tiempo aproximado requerido para completar la encuesta fue de 8 minutos.

3.2.1 Análisis de datos

En esta fase de análisis de datos se utilizó el software IBM SPSS Statistics 27 para evaluar la fiabilidad de los datos a través del coeficiente alfa de Cronbach. Esta herramienta estadística permitió obtener una medida confiable de la consistencia de los datos recopilados en el estudio, asegurando con ello la calidad y fiabilidad de los resultados obtenidos.

3.1.3 Validación Funcional de Exactitud de Datos

Previo a la evaluación de la satisfacción del usuario se puso como objetivo verificar la integridad y precisión de los cálculos realizados por el sistema. Para esto se aplicó una **Prueba en paralelo** la cual sirve para procesar un conjunto de datos reales de la empresa, utilizando en sí dos métodos simultáneos que son el método tradicional que es mediante hojas de cálculos manuales o físicas y el nuevo sistema ya desarrollado.

3.2.2 Alfa de Cronbach

El coeficiente alfa de Cronbach es una métrica estadística que hace posible la evaluación de la coherencia interna de un instrumento de medición, como lo puede ser un cuestionario. Representa la correlación que existe entre los diferentes ítems del instrumento y sus respuestas asociadas, lo que refleja su fiabilidad. En este sentido el coeficiente varía de 0 a 1, siendo valores más cercanos a 1 indicativos de una mayor

consistencia interna y, por lo tanto, una mayor fiabilidad del instrumento de medición [51].

El cálculo del coeficiente alfa de Cronbach puede realizarse a partir de la varianza de los ítems o empleando la matriz de correlación. En este apartado, se presenta la fórmula correspondiente al método de la varianza de los items.

$$\alpha = \frac{K}{K-1}\left[1 - \frac{\Sigma Vi}{Vt}\right]$$

- α= Alfa de Cronbach
- K= Número de ítems
- Vi= Varianza de cada ítem
- Vt= Varianza total

La *Tabla 55. Interpretaciones del alfa de Cronbach.* proporciona una clasificación más detallada de los niveles de fiabilidad asociados con los valores específicos del coeficiente alfa de Cronbach.

Tabla 55. Interpretaciones del alfa de Cronbach.

Índice	Nivel de fiabilidad	Valor de alfa de Cronbach
1	Excelente	]0.9, 1]
2	Muy bueno	]0.7, 0.9]
3	Bueno	]0.5, 0.7]
4	Regular	]0.3, 0.5]
5	Deficiente	[0, 0.3]

Nota: Clasificación de los niveles de fiabilidad [52]

La ***¡Error! No se encuentra el origen de la referencia.*** presenta la matriz completa de l as respuestas que fueron recolectadas en donde se organiza en filas a los 12 colaboradores encuestados identificados como U1 a U12 para mantener el anonimato y en las columnas a los 24 ítems del instrumento P1 a P24 que identifican a las preguntas.

Para la cuantificación de los datos se empleó una escala de Likert de 5 puntos ordinales, cuya codificación para el procesamiento estadístico se definió de la siguiente manera:

- **(1) Muy en desacuerdo:** Inconformidad total con la afirmación.
- **(2) Medianamente en desacuerdo:** Percepción negativa parcial.

- **(3) Ni de acuerdo ni en desacuerdo:** Postura neutral o indiferente.
- **(4) Medianamente de acuerdo:** Percepción positiva parcial.
- **(5) Muy de acuerdo:** Aceptación total de la afirmación.

Tabla 56. Matriz de respuestas del cuestionario (Escala 1-5).

	P 1	P 2	P 3	P 4	P 5	P 6	P 7	P 8	P 9	P 10	P 11	P 12	P 13	P 14	P 15	P 16	P 17	P 18	P 19	P 20	P 21	P 22	P 23	P 24
U1	5	5	5	4	5	5	5	5	4	4	4	5	5	5	5	5	5	5	5	4	5	5	5	5
U2	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
U3	5	5	5	5	5	5	5	4	5	5	5	5	5	4	5	5	5	5	4	5	5	5	5	5
U4	5	5	5	4	5	5	5	5	5	4	4	4	5	4	4	4	4	4	4	4	4	4	4	4
U5	5	4	5	5	5	5	5	5	5	5	5	5	5	4	5	5	5	5	5	5	5	5	5	5
U6	4	5	5	3	5	5	5	5	4	4	4	4	5	5	3	5	4	4	4	5	5	4	4	5
U7	5	5	5	5	5	5	5	5	4	4	5	5	5	4	5	5	5	5	5	5	5	5	5	5
U8	5	5	5	5	5	5	5	5	4	4	5	4	5	5	5	5	5	5	5	4	5	5	5	5
U9	5	5	5	5	4	5	5	5	4	4	5	5	5	5	5	5	5	3	5	5	4	5	5	5
U10	4	4	4	5	5	5	5	5	5	4	5	5	5	4	5	5	5	5	5	5	5	5	5	5
U11	4	5	5	4	5	4	5	5	4	4	5	4	4	4	5	5	5	4	4	4	4	4	4	4
U12	4	4	4	5	4	4	4	4	5	5	4	4	5	5	5	5	5	5	5	5	5	5	5	5

Se calculó el coeficiente de Cronbach después de que se obtuvieron los resultados del cuestionario, este proceso para obtener el coeficiente se puede hacer de diferentes maneras, pero en este caso se usó la herramienta estadística IBM SPSS, los resultados que se obtuvieron se muestran a continuación.

Tabla 57. Estadística de fiabilidad.

Alfa de Cronbach	**N de elementos**
,844	24

Nota. Valor obtenido desde el software IBM SPSS.

Como se observa en la tabla el coeficiente obtenido es de 0,84, según la escala de valoración el valor se sitúa en el rango de **Muy bueno,** esto significa que existe una alta consistencia interna entre las preguntas que se formularon, con esto se confirma la validez del instrumento para el análisis de los resultados sin necesidad de eliminar ítems del cuestionario.

3.2 Presentación de resultados

Una vez validada la exactitud técnica del software, se procedió a analizar la percepción de los usuarios basada en las encuestas aplicadas. A continuación, se presentan los hallazgos organizados por demografía y por las dimensiones del modelo de calidad.

Tras confirmar que el software funciona correctamente a nivel técnico, evaluamos la experiencia de los usuarios mediante las encuestas realizadas. Los resultados se detallan seguidamente, segmentados por perfil demográfico y por los criterios del modelo de calidad.

3.2.1 Análisis del perfil de los encuestados

Para validar el sistema se contó con la colaboración de 12 personas cuales cubrían así el total de la muestra planeada, es importante conocer el perfil de este grupo para asegurar que realmente representan a los usuarios por lo tanto a continuación se detallan sus características divididas en género, edad y el área donde trabajan.

- **Distribución por género**

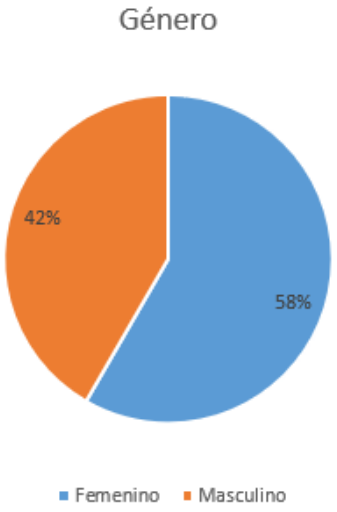


Figura 39. Distribución de usuarios por género.

Como se aprecia en la imagen del gráfico circular, la distribución de género de los encuestados muestra que el **58%** corresponde al género **Femenino** (representado por 7 personas), mientras que el **42%** restante pertenece al género **Masculino** (correspondiente a 5 personas). Esta composición demográfica revela una participación mayoritariamente femenina en la muestra analizada.

- **Rango de edad**

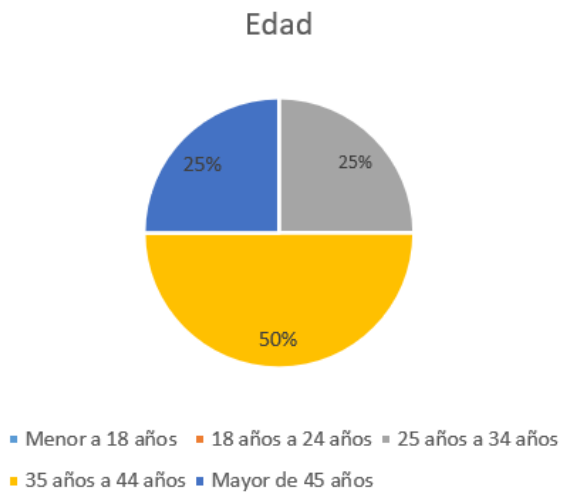


Figura 40. Distribución de usuarios por rango de edad.

Como se aprecia en la imagen del gráfico, la distribución por rango de edad de los encuestados muestra que el **50%** corresponde al grupo de **35 a 44 años** (representado por 6 personas). Por su parte, el 50% restante se divide equitativamente entre el rango de **25 a 34 años** y el grupo **Mayor de 45 años**, con un **25%** cada uno (correspondiente a 3 personas por grupo). Esta composición demográfica revela una predominancia de personal con madurez profesional en la muestra analizada, destacando la ausencia de participantes menores de 24 años.

3.2.4 Variables del modelo DeLone y McLean

En esta sección se presentan los resultados que se obtuvieron a partir del cuestionario aplicado a los 12 participantes que utilizaron los sistemas desplegados, enfocándose en las dimensiones del modelo de DeLone y McLean. Las respuestas proporcionadas por los usuarios fueron valoradas con la escala de Likert, donde los valores rondan entre 1 y 5. En esta escala, el valor 1 indica un total desacuerdo o insatisfacción, mientras que el valor 5 representa una completa satisfacción por parte del usuario que utiliza las aplicaciones.

Calidad del sistema

Esta dimensión analiza la eficiencia técnica del software considerando tanto al equipo como al operario. Para ello, mide qué tan sencillo es usar el programa, la flexibilidad de sus interfaces y el tiempo que requiere un usuario para dominar sus funciones.

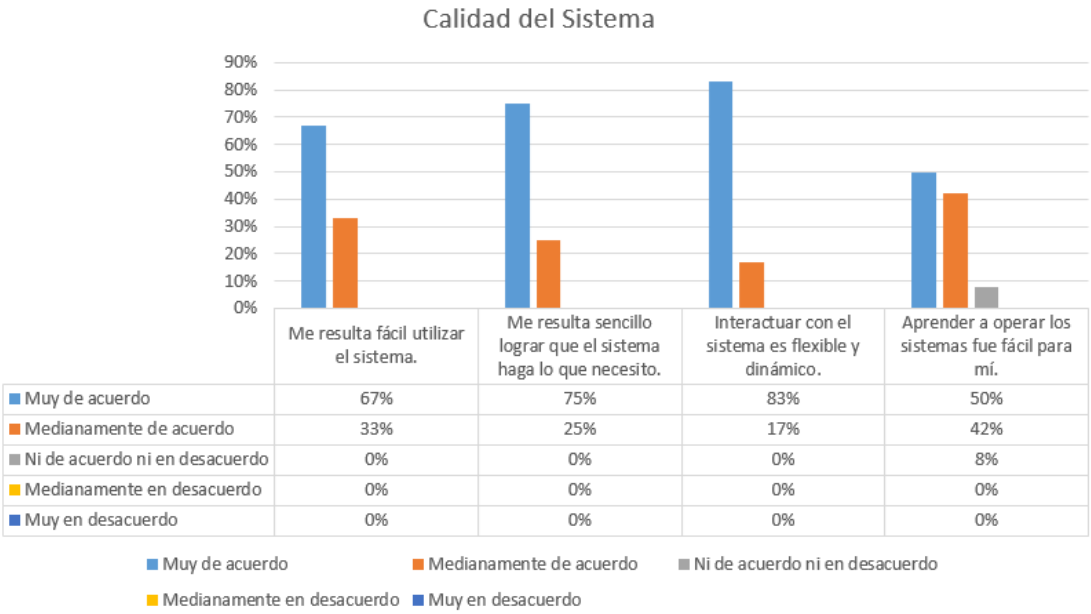


Figura 41. Calidad del sistema.

En la *Figura 41. Calidad del sistema.* Se observa una aceptación predominante del sistema. Destaca que la característica mejor valorada fue la flexibilidad y el dinamismo de la interacción, donde un 83% de los usuarios manifestó estar "muy de acuerdo". De igual manera, la eficiencia técnica del software para cumplir con los requerimientos del usuario alcanzó un 75% de máxima satisfacción. Aunque el proceso de aprendizaje presentó una ligera dispersión con un 8% de respuestas neutrales, la ausencia total de opiniones negativas en todas las categorías confirma que el equipo y los operarios perciben el programa como una herramienta sencilla de utilizar y técnicamente eficiente.

Calidad de la información

Esta dimensión analiza cómo el usuario percibe los datos generados por el sistema. Para ello, se consideran factores determinantes como la exactitud de los reportes, la utilidad de la información en las tareas cotidianas, la vigencia de los datos y la fiabilidad de las operaciones lógicas y cálculos realizados.

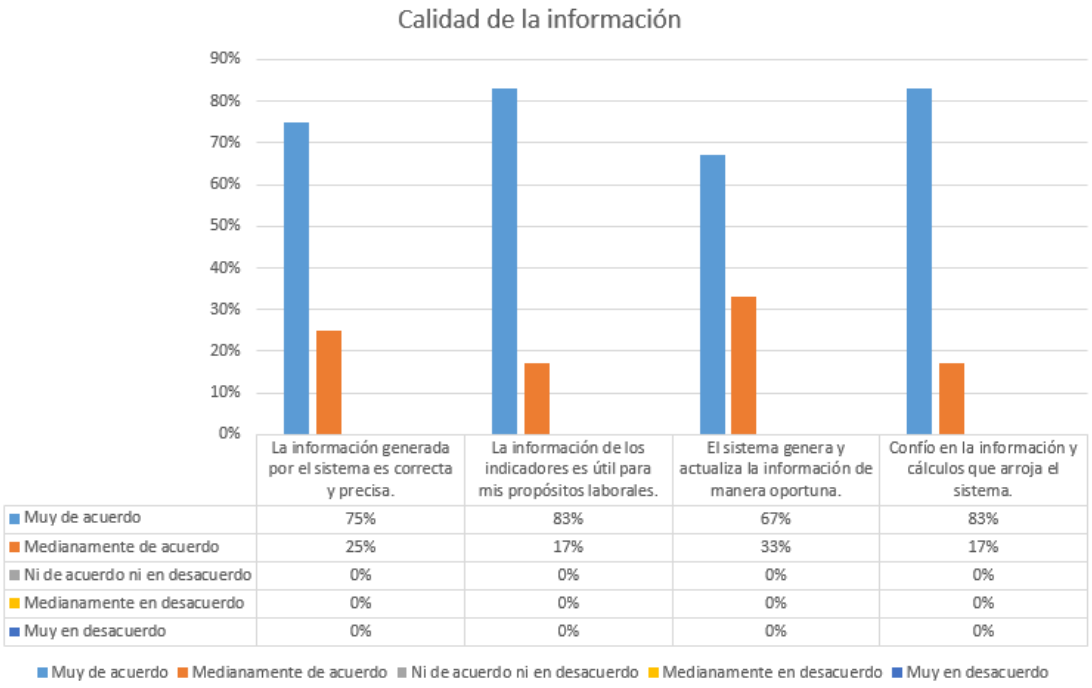


Figura 42. Calidad de la información.

Como podemos apreciar en la *Figura 42. Calidad de la información.* Los resultados son positivos, evidenciando alto nivel de aceptación y confianza de los usuarios respecto a la calidad de la información del sistema. Es notable que el 100% de las respuestas se concentren en valoraciones positivas, sin registrarse opiniones neutrales o en desacuerdo. Destaca de manera particular que un 83% de los encuestados está "Muy de acuerdo" con la utilidad de los indicadores para sus fines laborales y con la fiabilidad de los cálculos que el sistema arroja. Asimismo, la precisión de los datos fue ratificada con un 75% de aprobación máxima, mientras que la oportunidad en la generación y actualización de información, aunque presenta el porcentaje más moderado en el rango de excelencia (67%), mantiene una aceptación total al complementarse con un 33% en el nivel medianamente de acuerdo. En conjunto, estas cifras demuestran que el sistema es percibido como una herramienta robusta, precisa y alineada con las necesidades operativas de la organización.

Calidad del servicio

Esta dimensión examina el respaldo técnico y organizacional que acompaña al software. Se centra en medir la eficiencia de la asistencia técnica, la solidez de los recursos tecnológicos de la institución y la capacidad del sistema para mantenerse estable y operativo durante las interacciones del usuario.

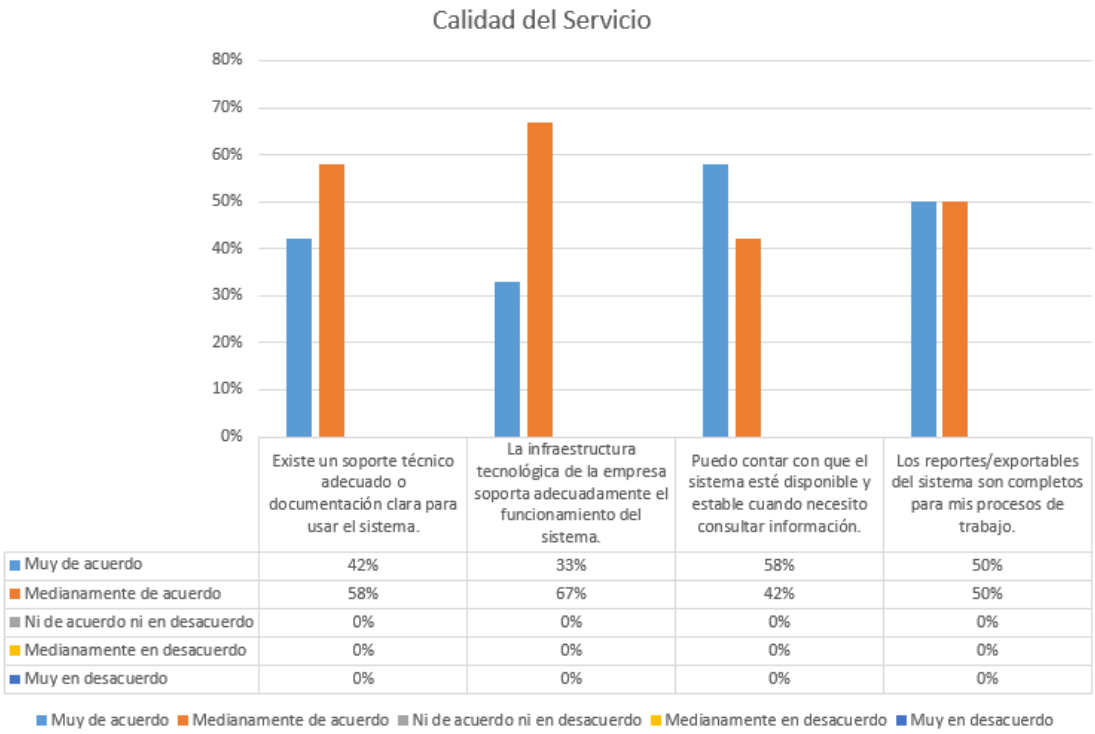


Figura 43. Calidad del servicio.

Como se aprecia en la *Figura 43. Calidad del servicio.* Los resultados son sumamente positivos, confirmando que la infraestructura y el soporte del sistema tienen de una aceptación total por parte de los usuarios. Resalta que en la categoría de disponibilidad y estabilidad, un 58% de los participantes se declara "Muy de acuerdo", lo que garantiza que el sistema es percibido como una herramienta confiable para la consulta de información en tiempo real. Respecto a la calidad de los reportes y documentos exportables, se observa un consenso dividido equitativamente entre "Muy de acuerdo" (50%) y "Medianamente de acuerdo" (50%), reafirmando que el sistema satisface íntegramente los requerimientos de los procesos de trabajo. Finalmente, en cuanto al soporte técnico y la capacidad de la infraestructura tecnológica, predominan las valoraciones de "Medianamente de acuerdo" con un 58% y 67% respectivamente; no obstante, el hecho de que no existan registros de disconformidad o neutralidad permite concluir que el servicio es eficiente y proporciona un entorno técnico estable para la operación

Intención de uso

Esta dimensión establece la medida en que el personal percibe que el sistema optimiza su labor. Aquí se evalúa si la herramienta agiliza la ejecución de tareas, si simplifica los

procesos diarios y si el usuario la reconoce como un instrumento de valor para las responsabilidades de su cargo.

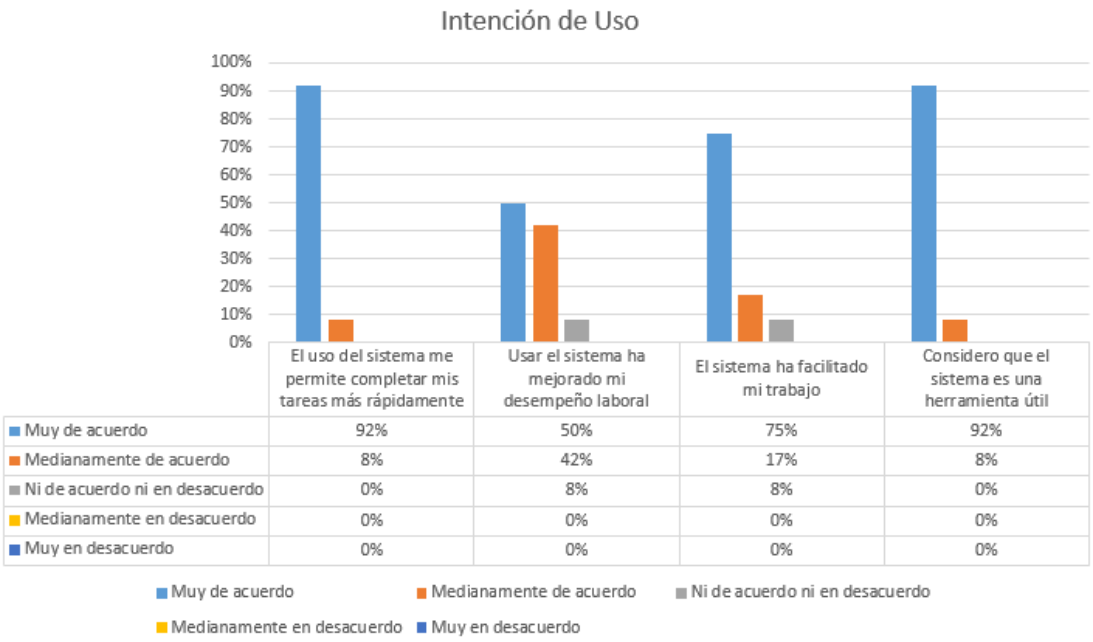
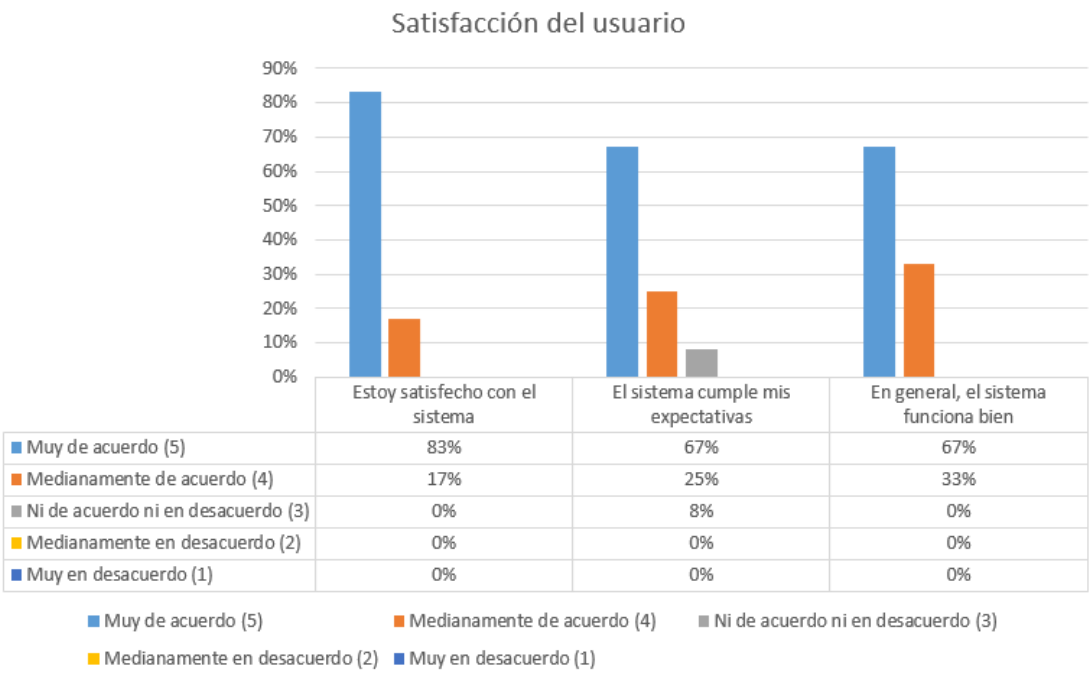


Figura 44. Intención de uso.

Como se aprecia en la *Figura 44. Intención de uso.* Los resultados son sumamente positivos, evidenciando una fuerte aceptación de la herramienta como un catalizador de la productividad individual. Sobresale el hecho de que un 92% de los encuestados está "Muy de acuerdo" en que el sistema permite completar tareas con mayor rapidez y lo considera una herramienta útil para sus funciones. Si bien se registra un pequeño margen de neutralidad del 8% en la percepción de mejora del desempeño y facilitación del trabajo, la tendencia general es abrumadoramente favorable. Estos datos sugieren que la implementación no solo ha sido aceptada por los usuarios, sino que es percibida como un activo indispensable para la optimización de sus flujos laborales cotidianos.

Satisfacción del usuario

Esta dimensión determina si el software está a la altura de lo que el usuario espera y el nivel de agrado que experimenta al utilizarlo. Se valora especialmente la conformidad con las herramientas disponibles y la percepción positiva sobre la experiencia de uso integral.



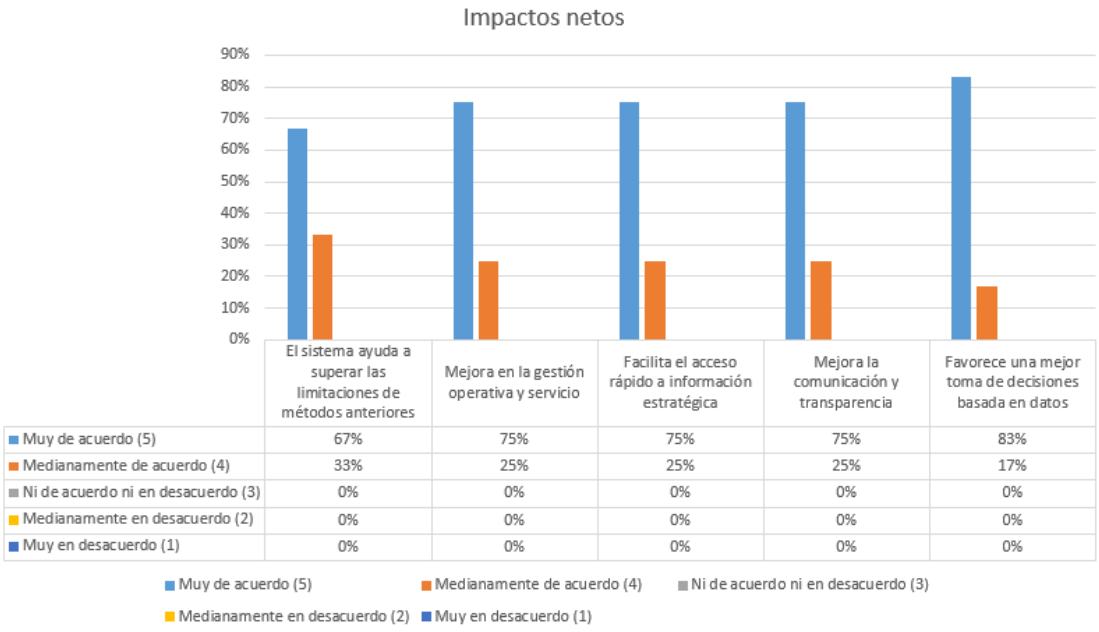
	Estoy satisfecho con el sistema	El sistema cumple mis expectativas	En general, el sistema funciona bien
Muy de acuerdo (5)	83%	67%	67%
Medianamente de acuerdo (4)	17%	25%	33%
Ni de acuerdo ni en desacuerdo (3)	0%	8%	0%
Medianamente en desacuerdo (2)	0%	0%	0%
Muy en desacuerdo (1)	0%	0%	0%

Figura 45. Satisfacción del usuario.

Como se aprecia en la *Figura 45. Satisfacción del usuario.* Los resultados son sumamente positivos, reflejando un cierre de evaluación favorable para la implementación del proyecto. El nivel de satisfacción personal con el sistema alcanza un 83% en la valoración máxima ("Muy de acuerdo"), lo cual es un indicador crítico del éxito de la interfaz y la experiencia de usuario. Asimismo, la percepción de que el sistema funciona bien y cumple con las expectativas del personal registra un sólido 67% de aprobación total. A pesar de un leve 8% de respuestas neutrales en cuanto al cumplimiento de expectativas, la ausencia de cualquier tipo de insatisfacción confirma que la solución tecnológica entregada está alineada con las demandas y necesidades de los usuarios finales.

Impactos netos

Esta dimensión se enfoca en los resultados positivos y medibles que el sistema aporta tanto al individuo como a la organización. Incluye el incremento de la productividad, el perfeccionamiento de la gestión operativa, la fluidez en la comunicación interdepartamental y la optimización en la resolución de problemas y toma de decisiones.



	El sistema ayuda a superar las limitaciones de métodos anteriores	Mejora en la gestión operativa y servicio	Facilita el acceso rápido a información estratégica	Mejora la comunicación y transparencia	Favorece una mejor toma de decisiones basada en datos
Muy de acuerdo (5)	67%	75%	75%	75%	83%
Medianamente de acuerdo (4)	33%	25%	25%	25%	17%
Ni de acuerdo ni en desacuerdo (3)	0%	0%	0%	0%	0%
Medianamente en desacuerdo (2)	0%	0%	0%	0%	0%
Muy en desacuerdo (1)	0%	0%	0%	0%	0%

Figura 46. Impactos netos.

Como se aprecia en la *Figura 46. Impactos netos.* los resultados son sumamente positivos, demostrando que la implementación ha generado beneficios estratégicos tangibles para la institución. El impacto más significativo se observa en la toma de decisiones basada en datos, donde un **83%** de los encuestados afirma estar "Muy de acuerdo" con la mejora obtenida. De igual manera, se registra una valoración constante del **75%** en el rango máximo para la mejora de la gestión operativa, el acceso rápido a información estratégica y el incremento en la comunicación y transparencia. Finalmente, el **67%** de aprobación máxima respecto a la superación de las limitaciones de métodos anteriores ratifica que el sistema ha logrado modernizar y optimizar la gestión de información de manera efectiva.

3.2.5 Análisis de favorabilidad y desfavorabilidad

Concluida la tabulación de las variables del modelo DeLone & McLean, se realiza el análisis de favorabilidad y desfavorabilidad. Este análisis permite consolidar las respuestas de la escala de Likert en tres grandes grupos para determinar la aceptación global de la arquitectura de microservicios implementada. Se presenta a continuación la descripción de la escala utilizada.

Tabla 58. Descripción del análisis de favorabilidad y desfavorabilidad.

Ítem	Descripción

Favorabilidad	Respuestas "Muy de acuerdo" o "Medianamente de acuerdo"
Indecisión	Respuestas "Ni de acuerdo ni en desacuerdo"
Desfavorabilidad	Respuestas "Muy en desacuerdo" o "Medianamente en desacuerdo"

Una vez definidos los criterios, se procedió a calcular los porcentajes promedios correspondientes a cada dimensión evaluada para los sistemas que interconectan a las instituciones CAEI, CICI e INNOVA EP. Los resultados consolidados se detallan en la siguiente tabla:

Tabla 59. Consolidado de favorabilidad, indecisión y desfavorabilidad.

Dimensión	Favorabilidad	Indecisión	Desfavorabilidad
Calidad del Sistema	98,00%	2,00%	0,00%
Calidad de la Información	100,00%	0,00%	0,00%
Calidad del Servicio	100,00%	0,00%	0,00%
Intención de Uso	96,00%	4,00%	0,00%
Satisfacción del Usuario	97,33%	2,67%	0,00%
Impactos Netos	100,00%	0,00%	0,00%

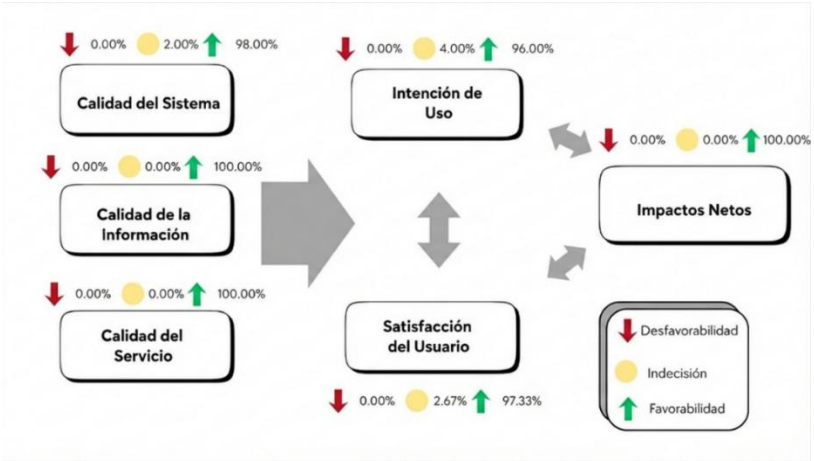


Figura 47. Análisis de la favorabilidad y desfavorabilidad

A continuación, se muestra un resumen gráfico de los resultados obtenidos tras la aplicación del modelo y sus dimensiones:

En cuanto a la **Calidad del sistema**, se observa que el 98,00% de los encuestados evaluó positivamente la dimensión técnica, mientras que solo un 2,00% mostró indecisión (específicamente en la curva de aprendizaje inicial). Estos hallazgos confirman que el personal percibe la plataforma basada en microservicios como una herramienta fácil de operar, flexible y técnicamente robusta, validando el diseño de las interfaces y la eficiencia del software.

En lo que respecta a la **Calidad de la información**, se obtuvo un contundente 100,00% de favorabilidad, sin registrarse desfavorabilidad ni indecisión. Este resultado sugiere que los usuarios perciben los datos y reportes generados como precisos, confiables y oportunos, indicando que el sistema cumple de manera efectiva con su propósito de suministrar información útil para las tareas cotidianas de las instituciones.

En relación con la **Calidad del servicio**, el 100,00% de los encuestados evaluó positivamente la infraestructura tecnológica y el soporte. No se observó desfavorabilidad, lo que permite concluir que la totalidad del personal considera que la disponibilidad del sistema y la estabilidad de la infraestructura son adecuadas, ofreciendo un entorno técnico confiable para la operación continua.

En cuanto a la **Intención de uso**, se identificó que el 96,00% de los usuarios expresó su voluntad de seguir utilizando el sistema, reconociendo su valor para agilizar tareas. Se registró un 4,00% de indecisión en los rubros de mejora del desempeño laboral, pero un 0% de desfavorabilidad. Este resultado demuestra una alta disposición del personal para integrar la herramienta en su flujo de trabajo diario, percibiéndola como un activo indispensable.

En lo referente a la **Satisfacción del usuario**, se obtuvo un 97,33% de respuestas positivas. La gran mayoría de los encuestados se siente conforme con la experiencia de uso integral y el funcionamiento del sistema, superando las expectativas iniciales. El margen de indecisión (2,67%) es mínimo, confirmando que la solución tecnológica entregada está plenamente alineada con las necesidades operativas de los usuarios finales.

Finalmente, en términos de **Impactos netos**, se encontró que el 100,00% de los encuestados percibe beneficios estratégicos significativos, especialmente en la mejora de la toma de decisiones basada en datos y la comunicación interdepartamental. La ausencia total de desfavorabilidad demuestra que todos los usuarios reconocen que la interconexión de los sistemas aporta un valor estratégico real y moderniza la gestión institucional de manera efectiva.

Discusión

El propósito central de este estudio fue desarrollar e implementar una arquitectura de microservicios eficiente para interconectar los sistemas de CAEI, CICI e INNOVA EP. Para validar el éxito de esta integración, se sometió la plataforma al juicio de los usuarios operativos y administrativos institucionales. La solidez de este análisis se refleja en los altos índices de favorabilidad obtenidos, donde la Calidad del Sistema alcanzó un **98%**, lo que demuestra que la complejidad técnica de una arquitectura distribuida (basada en Docker) no representó una barrera para el usuario, sino que actuó como un facilitador de sus procesos.

Al contrastar los hallazgos con los objetivos planteados, se evidencia una transición exitosa desde sistemas aislados hacia un ecosistema integrado. Mientras que en implementaciones de microservicios es común enfrentar retos de latencia o fragmentación, en este caso la Calidad de la Información obtuvo un **100%** de favorabilidad, asegurando que el flujo de datos a través de es percibido como preciso y oportuno. Esta confianza plena en los datos derivó en una Satisfacción del Usuario del **97.33%**, eliminando la incertidumbre que generaba el manejo de información.

En conclusión, los indicadores demuestran que la solución es robusta y pertinente. El sistema no solo cumple con los requisitos técnicos de interconexión, sino que ha generado un alto impacto neto del en la percepción de los beneficios estratégicos, consolidándose como una herramienta que aporta transparencia y agilidad a la gestión pública interinstitucional.

Conclusiones

En cumplimiento con los objetivos planteados al inicio de esta investigación y tras la implementación de la arquitectura de microservicios para la interconexión institucional, se concluye que la adopción de un esquema basado en contenedores con Docker permitió garantizar la alta disponibilidad y escalabilidad del sistema. Esta estructura facilitó la coexistencia de servicios independientes para las instituciones CAEI, CICI e INNOVA EP, asegurando que el fallo en un componente específico no comprometiera la integridad ni la operatividad de toda la red interinstitucional.

Respecto a la integración de datos, el uso de brokers de mensajería para la comunicación asíncrona entre servicios desarrollados en Laravel logró solucionar de manera efectiva la problemática de los silos de información identificada inicialmente. Mediante este enfoque, se eliminó la duplicidad de registros y se garantizó una calidad de la información con un nivel de favorabilidad del 100%, lo que permite que las tres instituciones operen ahora sobre una base de datos sincronizada y coherente en tiempo real.

La eficiencia del desarrollo también fue validada mediante la construcción de interfaces bajo estándares modernos de usabilidad, permitiendo el manejo de operaciones complejas de forma intuitiva. Esto se reflejó en un sólido 96% de favorabilidad en la intención de uso, lo que confirma que los operarios perciben la plataforma como un medio efectivo para optimizar sus tiempos de respuesta y completar tareas con mayor rapidez.

Finalmente, El paso de datos estáticos a representaciones visuales permitió a los directivos identificar rápidamente el estado de los proyectos, cumpliendo así con el objetivo estratégico de facilitar la toma de decisiones basada en datos y mejorar significativamente la comunicación y transparencia entre las entidades participantes.

Recomendaciones

Tomando como base los resultados obtenidos, se recomienda en primer lugar establecer un protocolo de monitoreo especializado para las colas de mensajería de RabbitMQ para realizar escalamientos preventivos y asegurar que la disponibilidad se mantenga en los niveles de excelencia registrados. Complementariamente, es vital realizar auditorías de seguridad periódicas sobre las APIs y los puntos de acceso interinstitucionales para proteger la integridad de la información estratégica en el entorno distribuido.

Por otro lado, se sugiere aprovechar la naturaleza modular de la arquitectura para evaluar la incorporación de nuevas entidades públicas al ecosistema, dado que el sistema permite desplegar nuevos servicios sin necesidad de reestructurar el núcleo técnico. Para dar soporte a esta evolución, se recomienda establecer un programa de capacitación continua para los equipos de TI en la gestión de servicios, asegurando así que el sistema escale conforme a las demandas futuras sin incurrir en deuda técnica.

Referencias Bibliográficas

[1] Parra Vite José Luis, “DESARROLLO DE UNA ARQUITECTURA DEVOPS CON JENKINS Y DOCKER PARA REFORZAR EL MANEJO DE INTEGRACIÓN CONTÍNUA BASADO EN LA FASE 3 Y 4 DE TOGAF USANDO UNA PRUEBA DE CONCEPTO DEL MÓDULO DE CONSULTA DE UN SISTEMA FINANCIERO,” 2023. Accessed: Apr. 27, 2025. [Online]. Available: https://repositorio.utn.edu.ec/bitstream/123456789/14581/2/04%20SOF%20011%20TRABAJO%20GRADO.pdf

[2] Loor Galarza Eduardo Eddy and Martha Romero Castro, “MÓDULO SERVIDOR PARA PRACTICAS UTILIZANDO SOFTWARE LIBRE EN LA ASIGNATURA DE TELECOMUNICACIONES DE LA CARRERA DE INGENIERÍA EN COMPUTACIÓN Y REDES DE LA UNIVERSIDAD ESTATAL DEL SUR DE MANABÍ,” 2017. Accessed: Apr. 27, 2025. [Online]. Available: http://repositorio.unesum.edu.ec/handle/53000/986

[3] T. Papailias, K. Papageorgiou, K. Milioris, and S. Riakotaki, “The Application and Integration of I.C.T. in the Public Sector: an Overall Literature Review Focused on the E.U. Environment,” *Journal of Public Administration Studies*, vol. 7, no. 1, pp. 31–35, Apr. 2022, doi: 10.21776/ub.jpas.2022.007.01.6.

[4] Secretaria Nacional de Planificación, “Plan de Creación de Oportunidades 2021-2025,” 2021.

[5] N. Unidas, “La Agenda 2030 y los Objetivos de Desarrollo Sostenible: una oportunidad para América Latina y el Caribe,” 2030. [Online]. Available: www.issuu.com/publicacionescepal/stacks

[6] D. López and E. Maya, “Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web,” 2017.

[7] Z. E. Mamani Rodríguez, L. Del Pino Rodríguez, and J. C. Gonzales Suarez, “Arquitectura basada en Microservicios y DevOps para una ingeniería de software continua,” *Industrial Data*, vol. 23, no. 2, pp. 141–149, Dec. 2020, doi: 10.15381/idata.v23i2.17278.

[8] Fernando Muraca, “De monolitos a microservicios: tendencias y oportunidades en la transición de arquitecturas de software,” Nov. 2023.

[9] K. E. Ortiz-Noriega, J. E. Guevara-Segura, J. P. Santos-Fernández, O. R. Alcántara-Moreno, J. L. Tenorio-Cabrera, and R. J. Sánchez-Ticona, “Aplicación web con arquitectura de microservicios y el incremento de la eficiencia en el comercio electrónico de una empresa peruana de calzado,” in *CISCI 2022 - Vigesima Primera Conferencia Iberoamericana en Sistemas, Cibernetica e Informatica, Decimo Noveno Simposium Iberoamericano en Educacion, Cibernetica e Informatica - Memorias*, International Institute of Informatics and Systemics, IIIS, 2022, pp. 158–163. doi: 10.54808/CISCI2022.01.158.

[10] T. Patricia Bazán, “Implementación de procesos de negocio a través de servicios aplicando metamodelos, software distribuido y aspectos sociales,” 2015.

[11] Patricia Bazán, Alejandro Fernández, Nicolás del Rio, Lia Molinari, Juan Pablo Pérez, and Matías Banchoff, “Aplicaciones, servicios y procesos distribuidos Una visión para la construcción de software Libros de Cátedra,” 2017.

[12] P. E. De la Cruz Vélez de Villa, M. H. Espinoza Ramirez, and O. Cuba Estrella, “Propuesta de arquitectura de microservicios, metodología Scrum para una aplicación móvil de control académico: Caso Escuela Profesional de Obstetricia de la UNMSM,” *HAMUT’AY*, vol. 6, no. 2, Aug. 2021, doi: 10.21503/hamu.v6i2.1781.

[13] Barahona Carmen, “Propuesta de implementación de microservicios usando metodología ágil para la optimización de procesos en BANHCAFE,” Jan. 2023.

[14] Carlomagno González Villalobos, “IMPLEMENTACIÓN DE LA ARQUITECTURA DE MICROSERVICIOS EN LA UNIVERSIDAD TÉCNICA NACIONAL DE COSTA RICA (UTN),” 2025.

[15] ISO/IEC TS, “Information technology-Cloud computing-Common technologies and techniques,” 2020.

[16] E. J. Sánchez Pérez, J. Steveen, and M. Garces, “Arquitectura de microservicios para un sistema recomendador de seguros para mascotas en Bogotá,” 2024.

[17] ISO/IEC TS, “Information technology-Cloud computing-Common technologies and techniques,” 2020.

[18] Marcelo Guerrero, Maribel Martínez, Luis Gualotuña, Giovanny Jami, and José Luis Nath, “ESQUEMA GUBERNAMENTAL DE SEGURIDAD DE LA INFORMACION (EGSI),” 2024.

[19] NARCISA ORTEGA MENDOZA and SOFIA PAZMIÑO YANEZ, “ESTATUTOS DEL COLEGIO DE ARQUITECTOS DEL ECUADOR,” 2024.

[20] R. Lizcano Adriana and I. Roberto de la Cruz Centeno Lara, “APPLYICT IN THE ANALYSIS AND PERCEPTION OF THE RESIDENCIAL BUILDINGS VULNERABILITY LEVEL IN CUCUTA CITY,” 2016.

[21] Sosa Díaz, Maria Jose, Peligros García, and Díaz Muriel Dionisio, “Buenas prácticas organizativas para la integración de las TIC en el sistema educativo extremeño,” 2010.

[22] FasterCapital, “Superar Los Desafíos En La Implementación De La Interoperabilidad Del Gobierno Electrónico - FasterCapital.” Accessed: Jun. 28, 2025. [Online]. Available: https://fastercapital.com/es/tema/superar-los-desaf%C3%ADos-en-la-implementaci%C3%B3n-de-la-interoperabilidad-del-gobierno-electr%C3%B3nico.html

[23] Jakob Efe, “Interoperabilidad para servicios ciudadanos modernos | Capgemini.” Accessed: Jun. 28, 2025. [Online]. Available: https://www.capgemini.com/mx-es/solutions/interoperabilidad/?_gl=1*mjghm7*_ga*NjgzODE0Mjk3LjE3NDMxODk5MzU.*_ga_WC57KJ50ZZ*czE3NTEyNDc1MDUkbzEyMCRnMSR0MTc1MTI0ODYyOCRqNTYkbDAkaDA.

[24] P. Espinet and R. Segura, “EUBIM 2015 Congreso Internacional BIM / Encuentro de Usuarios BIM MÁS ALLÁ DE LA INTEROPERABILIDAD TÉCNICA,” 2015.

[25] J. de J. Álvarez Ramírez and R. Maciel Arellano, “Adaptación de Scrum al desarrollo de software cuántico: Un artículo de revisión metodológica,” *Ciencia Latina Revista Científica Multidisciplinar*, vol. 9, no. 3, pp. 3491–3499, Jun. 2025, doi: 10.37811/cl_rcm.v9i3.17963.

[26] I. Reascos, F. Garrido, C. Pineda, F. Salazar-Fierro, R. Pomasqui, and J. Cachipuendo, “Evaluation of the Integrated University Information System at Universidad Técnica del Norte Using the DeLone and McLean Success Model,” *Data and Metadata*, vol. 4, Jan. 2025, doi: 10.56294/dm2025187.

[27] A. Ricardo and A. Ramírez, “Evaluación del módulo de Recursos Humanos del Enterprise Resource Planning (ERP) en una empresa colombiana usando el modelo de Delone y Mclean,” 2012.

[28] V. Duvvur, “Modernizing Government IT Systems: A Case Study on Enhancing Operational Efficiency and Data Integrity,” *International Journal of Computational and Experimental Science and Engineering*, vol. 11, no. 1, pp. 1165–1175, Dec. 2024, doi: 10.22399/ijcesen.1193.

[29] T. Nuthalapati, “Reimagining Public Services – Cloud Infrastructure as the Backbone of Modern Governance,” *European Journal of Computer Science and Information Technology*, vol. 13, no. 27, pp. 12–26, Apr. 2025, doi: 10.37745/ejcsit.2013/vol13n271226.

[30] F. Tapia, M. ángel Mora, W. Fuertes, H. Aules, E. Flores, and T. Toulkeridis, “From monolithic systems to microservices: A comparative study of performance,” *Applied Sciences (Switzerland)*, vol. 10, no. 17, Sep. 2020, doi: 10.3390/app10175797.

[31] S. Sethi and S. Panda, “Transforming Digital Experiences: The Evolution of Digital Experience Platforms (DXPs) from Monoliths to Microservices: A Practical Guide,” *Journal of Computer and Communications*, vol. 12, no. 02, pp. 142–155, 2024, doi: 10.4236/jcc.2024.122009.

[32] I. K. Aksakalli, T. Celik, A. B. Can, and B. Tekinerdogan, “Systematic Approach for Generation of Feasible Deployment Alternatives for Microservices,” *IEEE Access*, vol. 9, pp. 29505–29529, 2021, doi: 10.1109/ACCESS.2021.3057582.

[33] M. Milić and D. Makajić-Nikolić, “Development of a Quality-Based Model for Software Architecture Optimization: A Case Study of Monolith and Microservice Architectures,” *Symmetry (Basel).*, vol. 14, no. 9, Sep. 2022, doi: 10.3390/sym14091824.

[34] D. C. Decimavilla-Alarcón and P. Fernando Marcillo-Franco, “CONTAINER-BASED MICROSERVICES ARCHITECTURE FOR AGILE DEPLOYMENT OF IOT APPLICATIONS IN THE CLOUD,” 2025. [Online]. Available: https://orcid.org/0000-0002-0375-0216

[35] N. Gadani and N. N. Gadani, “Microservices Architecture for Electronic Poll Books: Achieving Flexibility and Resilience in Election Systems International Journal on Recent and Innovation Trends in Computing and Communication Microservices Architecture for Electronic Poll Books: Achieving Flexibility and Resilience in Election Systems,” 2023, [Online]. Available: http://www.ijritcc.org

[36] I. Oumoussa and R. Saidi, “Evolution of Microservices Identification in Monolith Decomposition: A Systematic Review,” *IEEE Access*, vol. 12, pp. 23389–23405, 2024, doi: 10.1109/ACCESS.2024.3365079.

[37] M. I. Vinicio Estrada-Velasco, J. I. Alexandra Núñez-Villacis, and W. I. Clemente Cunuhay-Cuchipe, “Revisión Sistemática de la Metodología Scrum para el Desarrollo de Software Revisión Sistemática de la Metodología Scrum para el Desarrollo de Software,” vol. 7, 2021, doi: 10.23857/dc.v7i4.2429.

[38] I. Díaz, J. Sánchez, and O. Pastor, “Metamorfosis: Un Marco para el Análisis de Requisitos Funcionales,” 2020.

[39] C. Rica Arias Chaves, “La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software,” 2006, [Online]. Available: http://www.redalyc.org/articulo.oa?id=66612870011

[40] María Paula Izaurralde, “Caracterización de Especificación de Requerimientos en entornos Ágiles: Historias de Usuario,” 2013.

[41] C. Peixoto *et al.*, “Estimativa de Esforço em Projetos de DDS,” 2015.

[42] J. Guillermo, F. Mendoza, F. Gutiérrez Vera, C. Cristina, O. González, and A. S. Contreras, “Propuesta metodológica para la estimación de proyectos gestionados mediante Scrum, con enfoque a la pequeña industria del software,” 2019. [Online]. Available: http://itcelaya.edu.mx/ojs/index.php/pistas

[43] Hugo Del Pozo Barrezueta, “LEY ORGÁNICA DE ORDENAMIENTO TERRITORIAL, USO Y GESTIÓN DE SUELO,” 2016.

[44] E. Evans, “Domain-Driven Design Tackling Complexity in the Heart of Software,” 2003. [Online]. Available: www.domainlanguage.com

[45] Publicación de las Naciones Unidas, “Agenda 2030 y los Objetivos de Desarrollo Sostenible Una oportunidad para América Latina y el Caribe,” 2016. [Online]. Available: www.un.org/sustainabledevelopment/es

[46] C. Richardson, “Microservices Patterns,” 2018. [Online]. Available: https://avxhm.se/blogs/hill0

[47] M. Chechik, B. Combemale, J. Gray, and B. Rumpe, “Standards in software development and modeling,” Oct. 01, 2025, *Springer Science and Business Media Deutschland GmbH*. doi: 10.1007/s10270-025-01312-2.

[48] Dayana Sánchez Medrano, “ESTIMACIÓN DE PERCENTILES MEDIANTE EL SOFTWARE R-PROJECT,” 2024.

[49] W. H. DeLone and E. R. McLean, “The DeLone and McLean model of information systems success: A ten-year update,” in *Journal of Management Information Systems*, M.E. Sharpe Inc., 2003, pp. 9–30. doi: 10.1080/07421222.2003.11045748.

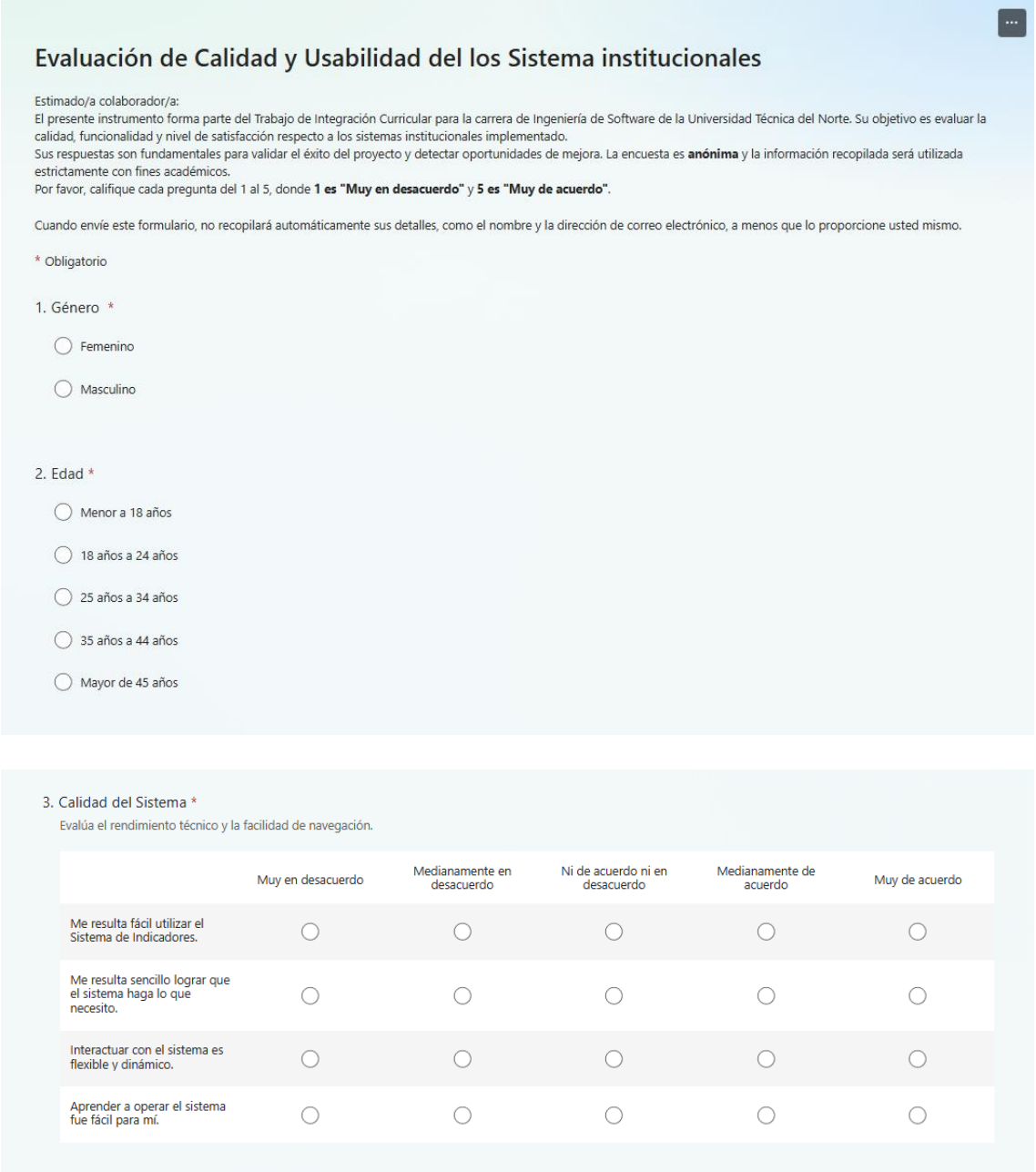
[50] A. I. Ojo, “Validation of the delone and mclean information systems success model,” *Healthc. Inform. Res.*, vol. 23, no. 1, pp. 60–66, Jan. 2017, doi: 10.4258/hir.2017.23.1.60.

[51] J. Rodríguez-Rodríguez and M. Reguant-Álvarez, “Calcular la fiabilidad de un cuestionario o escala mediante el SPSS: el coeficiente alfa de Cronbach,” *REIRE Revista d Innovaci i Recerca en Educaci*, vol. 13, no. 2, Jul. 2020, doi: 10.1344/reire2020.13.230048.

[52] J. V. Tuapanta Dacto, M. A. Duque Vaca, and A. P. Mena Reinoso, “Alfa de Cronbach para validar un cuestionario de uso de TIC en docentes universitarios,” 2017.

Anexos

Anexo 1: Encuesta de evaluación realizada con el modelo de éxito de sistemas de información de Delone y Mclen



Evaluación de Calidad y Usabilidad del los Sistema institucionales

Estimado/a colaborador/a:
El presente instrumento forma parte del Trabajo de Integración Curricular para la carrera de Ingeniería de Software de la Universidad Técnica del Norte. Su objetivo es evaluar la calidad, funcionalidad y nivel de satisfacción respecto a los sistemas institucionales implementado.
Sus respuestas son fundamentales para validar el éxito del proyecto y detectar oportunidades de mejora. La encuesta es **anónima** y la información recopilada será utilizada estrictamente con fines académicos.
Por favor, califique cada pregunta del 1 al 5, donde **1 es "Muy en desacuerdo"** y **5 es "Muy de acuerdo"**.

Cuando envíe este formulario, no recopilará automáticamente sus detalles, como el nombre y la dirección de correo electrónico, a menos que lo proporcione usted mismo.

* Obligatorio

1. Género *

- ◯ Femenino
- ◯ Masculino

2. Edad *

- ◯ Menor a 18 años
- ◯ 18 años a 24 años
- ◯ 25 años a 34 años
- ◯ 35 años a 44 años
- ◯ Mayor de 45 años

3. Calidad del Sistema *

Evalúa el rendimiento técnico y la facilidad de navegación.

	Muy en desacuerdo	Medianamente en desacuerdo	Ni de acuerdo ni en desacuerdo	Medianamente de acuerdo	Muy de acuerdo
Me resulta fácil utilizar el Sistema de Indicadores.	◯	◯	◯	◯	◯
Me resulta sencillo lograr que el sistema haga lo que necesito.	◯	◯	◯	◯	◯
Interactuar con el sistema es flexible y dinámico.	◯	◯	◯	◯	◯
Aprender a operar el sistema fue fácil para mí.	◯	◯	◯	◯	◯

4. Calidad de la Información *

Evalúa los datos que se muestran en pantalla.

	Muy en desacuerdo	Medianamente en desacuerdo	Ni de acuerdo ni en desacuerdo	Medianamente de acuerdo	Muy de acuerdo
La información generada por el sistema es correcta y precisa.	○	○	○	○	○
La información de los indicadores es útil para mis propósitos laborales.	○	○	○	○	○
El sistema genera y actualiza la información de manera oportuna.	○	○	○	○	○
Confío en la información y cálculos que arroja el sistema.	○	○	○	○	○

5. Calidad del Servicio *

Evalúa la estabilidad técnica y el respaldo.

	Muy en desacuerdo	Medianamente en desacuerdo	Ni de acuerdo ni en desacuerdo	Medianamente de acuerdo	Muy de acuerdo
Existe un soporte técnico adecuado o documentación clara para usar el sistema.	○	○	○	○	○
La infraestructura tecnológica de la empresa soporta adecuadamente el funcionamiento del sistema.	○	○	○	○	○
Puedo contar con que el sistema esté disponible y estable cuando necesito consultar información.	○	○	○	○	○
Los reportes/exportables del sistema son completos para mis procesos de trabajo.	○	○	○	○	○

6. Uso / Utilidad *

Evalúa el impacto en la productividad diaria.

	Muy en desacuerdo	Medianamente en desacuerdo	Ni de acuerdo ni en desacuerdo	Medianamente de acuerdo	Muy de acuerdo
El uso del sistema me permite completar mis tareas de monitoreo más rápidamente.	○	○	○	○	○
Usar el sistema ha mejorado mi desempeño laboral.	○	○	○	○	○
El sistema ha facilitado mi trabajo.	○	○	○	○	○
Considero que el sistema es una herramienta útil para mi puesto de trabajo.	○	○	○	○	○

7. Satisfacción del Usuario *

Evalúa la percepción subjetiva.

	Muy en desacuerdo	Medianamente en desacuerdo	Ni de acuerdo ni en desacuerdo	Medianamente de acuerdo	Muy de acuerdo
Estoy satisfecho con las funcionalidades y herramientas que ofrece el sistema.	○	○	○	○	○
El sistema ha simplificado mis procesos de trabajo diarios.	○	○	○	○	○
En general, me siento satisfecho utilizando este sistema.	○	○	○	○	○

8. Impactos netos *

Evalúa el impacto organizacional.

	Muy en desacuerdo	Medianamente en desacuerdo	Ni de acuerdo ni en desacuerdo	Medianamente de acuerdo	Muy de acuerdo
El sistema ayuda a superar las limitaciones de los métodos anteriores.	○	○	○	○	○
El uso del sistema contribuye a una mejora en la gestión operativa y el servicio de la empresa.	○	○	○	○	○
El sistema facilita el acceso rápido a la información estratégica de las áreas (Técnica, Administrativa, Financiera).	○	○	○	○	○
El sistema mejora la comunicación y transparencia entre las diferentes áreas.	○	○	○	○	○
El uso del sistema favorece una mejor toma de decisiones basada en datos.	○	○	○	○	○

Enviar

Anexo 2: Presentación de las aplicaciones institucionales

Anexo A: Fotografías del taller practico y encuesta del modelo de DeLone y McLean