UNIVERSIDAD VALLE DEL MOMBOY

VICERRECTORADO ACADÉMICO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA EN COMPUTACIÓN

SISTEMA DE GESTIÓN DE MANEJO DE MICROSERVICIOS EN LA EMPRESA DANIKLEAN LLC

Presentado por:

VICTOR J. GUTIERREZ. B

TRUJILLO, 2025

UNIVERSIDAD VALLE DEL MOMBOY

VICERRECTORADO ACADÉMICO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA EN COMPUTACIÓN

SISTEMA DE GESTIÓN DE MANEJO DE MICROSERVICIOS EN LA EMPRESA DANIKLEAN LLC

Trabajo Especial de Grado para optar al título de Ingeniero en Computación

Presentado por:

VICTOR J. GUTIERREZ. B

Tutor:

Msc. YAJAIRA SEGOVIA

TRUJILLO, 2025

DEDICATORIA

A mi mamá, por darme raíces y alas: educación, valores y la confianza para volar. Gracias por caminar conmigo en cada paso, incluso en silencio, y por ser faro en mis momentos de incertidumbre.

A mi novia, por tu paciencia infinita, tu fe inquebrantable en mí y por hacer que cada esfuerzo tuviera sentido.

A mi tía Iraida, por guiarme con sabiduría en los cruces más difíciles de este camino, y por creer que este trabajo era posible antes de que yo lo terminara.

A mi tía Leída, por tu apoyo constante, tu consejo oportuno y por ser una presencia que siempre supo cuándo sostenerme sin que yo tuviera que pedirlo.

Y a mi abuela, quien desde el cielo sigue iluminando mi camino. Gracias por enseñarme con el ejemplo que el esfuerzo y el amor son los cimientos de toda vida digna.

A ustedes,

pilares invisibles de este logro,

este triunfo también les pertenece.

AGRADECIMIENTOS

Este trabajo no habría sido posible sin el apoyo, la orientación y el cariño de muchas personas a quienes deseo expresar mi más sincero agradecimiento.

En primer lugar, a mi mamá, por su amor incondicional, su sacrificio constante y por darme no solo los estudios, sino el ejemplo de una vida con propósito.

A mi papá, por su respaldo silencioso pero firme, y por enseñarme el valor del compromiso y la responsabilidad.

A mi novia, por ser mi refugio emocional, mi motivación diaria y mi compañera en esta travesía académica. Tu presencia hizo que lo imposible pareciera alcanzable.

A mi abuela Dala, cuya ausencia física no apaga su presencia espiritual. Gracias por guiarme desde siempre con ternura, fortaleza y sabiduría. Aunque no estés aquí para celebrarlo, sé que sonríes desde el cielo.

A mi tía Iraida, por su orientación oportuna, su consejo sabio y su confianza en mi capacidad para culminar esta tesis. Tu apoyo fue decisivo en momentos clave.

A mi tía Leída, por su guía constante y por estar siempre dispuesta a tenderme la mano, incluso cuando no lo pedía.

A mi tía Dayana y a mi prima Daymar, por su apoyo incondicional y por formar parte de esa red familiar que me ha sostenido durante toda mi vida.

A mi profesora tutora, Yajaira, por su orientación académica rigurosa, su paciencia durante el proceso y su compromiso con la excelencia. Gracias por desafiarme a ir más allá y por creer en la relevancia de esta investigación.

A DANIKLEAN LLC, por abrir las puertas de su entorno profesional y permitirme aplicar mis conocimientos en un proyecto real y significativo.

Y, finalmente, a la Universidad Valle del Momboy, por formarme no solo como profesional, sino como persona comprometida con el desarrollo tecnológico y social de mi entorno.

RESUMEN

Este trabajo de grado presenta un estudio de caso de enfoque descriptivo-proyectivo, desarrollado en colaboración con la empresa DANIKLEAN LLC, con el objetivo de diagnosticar las necesidades técnicas en su infraestructura de software y proponer una solución basada en arquitectura de microservicios utilizando el modelo de comunicación NATS. La investigación se estructuró en dos fases: una descriptiva, en la que se aplicaron entrevistas semiestructuradas a stakeholders clave para identificar limitaciones operativas y requisitos funcionales; y una proyectiva, en la que se diseñó e implementó un prototipo funcional alineado con dichas necesidades. El enfoque metodológico fue no experimental, combinando elementos cualitativos (análisis temático de entrevistas) y cuantitativos (especificación técnica de componentes). Los resultados evidenciaron deficiencias en escalabilidad, mantenibilidad y comunicación entre servicios en el sistema actual, lo que justificó la propuesta de una nueva arquitectura. La discusión destaca que la solución propuesta no solo responde a las demandas específicas de la empresa, sino que también se alinea con buenas prácticas en ingeniería de software moderna. Se concluye que la integración de una fase diagnóstica rigurosa con una propuesta técnica fundamentada permite generar soluciones viables, contextualizadas y de alto impacto en entornos empresariales reales. Como recomendación, se sugiere implementar el prototipo en producción, monitorear su desempeño y replicar este modelo en otras organizaciones con necesidades similares.

Palabras clave: microservicios, NATS, estudio de caso, arquitectura de software, investigación descriptivo-proyectiva

ABSTRACT

This undergraduate thesis presents a descriptive-projective case study conducted in collaboration with DANIKLEAN LLC, aimed at diagnosing technical needs in its software infrastructure and proposing a solution based on a microservices architecture using the NATS communication model. The research was structured in two phases: a descriptive phase, where semi-structured interviews with key stakeholders identified operational limitations and functional requirements; and a projective phase, where a functional prototype aligned with those needs was designed and implemented. The methodological approach was non-experimental, combining qualitative (thematic analysis of interviews) and quantitative (technical specification of components) elements. Results revealed deficiencies in scalability, maintainability, and inter-service communication in the current system, justifying the proposed architecture. The discussion emphasizes that the solution not only addresses the company's specific demands but also aligns with modern software engineering best practices. It is concluded that integrating a rigorous diagnostic phase with a well-founded technical proposal enables the development of viable, contextualized, and high-impact solutions in real business environments. It is recommended to deploy the prototype in production, monitor its performance, and replicate this model in other organizations with similar needs.

Keywords: microservices, NATS, case study, software architecture, descriptive-projective research

ÍNDICE GENERAL

INDICE DE TABLAS

INDICE DE FIGURAS

INDICE DE ANEXOS

VEREDICTO

VICERRECTORADO ACADÉMICO
FACULTAD DE INGENIERÍA

VEREDICTO

Nosotros, Prof. Yerwin Vera, Prof. José Díaz y Profa. Yajaira Segovia designados como miembros del Jurado Examinador del Trabajo de Grado titulado: **"SISTEMA DE GESTIÒN DE MANEJO DE MICROSERVICIOS EN LA EMPRESA DANIKLEAN LLC "**. que presenta el bachiller: **GUTIÉRREZ BRACAMONTE VÍCTOR JOSÉ** portador de la **C.I. N.º. 28.206.156;** nos hemos reunido para revisar dicho trabajo y después de la presentación, defensa e interrogatorio correspondiente lo hemos calificado con **(20)** puntos, de acuerdo con las normas vigentes dictadas por el Consejo Universitario de la Universidad Valle del Momboy, referente a la evaluación de los Trabajos de Grado para optar al título de Ingeniero en Computación

En fe de lo cual firmamos en Carvajal a los tres (3) días del mes de diciembre del dos mil veinticinco (2025).

Profa. José Díaz
C.I: 18.397.123
JURADO

Profa. Yajaira Segovia
C.I: 14.148.893
TUTORA

Prof. Jeywin Vera
C.I. 17.862.776
PRESIDENTE DEL JURADO

Profa. Yumary Valecillos
C.I. 14.151.309
DECANO

Profa. Walevska López
C.I. 10.104.896
VICERRECTORA ACADEMICA

+58 412 2263605 www.uvm.edu.ve universidadvalledelmomboy@uvm.edu.ve

CAPÍTULO I

El Problema

1. Planteamiento del problema

1.1. Contexto Global

En el ámbito global del desarrollo de software empresarial, las arquitecturas monolíticas han mostrado limitaciones crecientes en términos de escalabilidad, mantenibilidad y velocidad de despliegue. Estas arquitecturas, en las que todos los componentes de una aplicación están fuertemente acoplados, generan cuellos de botella operativos, especialmente cuando se requiere escalar solo una funcionalidad o adoptar tecnologías heterogéneas (Newman, 2020). Este escenario es particularmente crítico en empresas tecnológicas que operan en entornos dinámicos, donde la agilidad, la resiliencia y la continuidad del servicio son factores determinantes para la competitividad.

Frente a estas limitaciones, la arquitectura de microservicios ha emergido como un paradigma dominante en el diseño de sistemas distribuidos. Según Newman (2020), los microservicios permiten estructurar una aplicación como un conjunto de servicios débilmente acoplados, cada uno desplegable e independiente, lo que favorece la escalabilidad, el aislamiento de fallos y la evolución tecnológica continua. Esta arquitectura ha sido adoptada por gigantes tecnológicos como Netflix, Amazon y Uber, quienes han demostrado mejoras significativas en tiempos de despliegue, tolerancia a fallos y escalabilidad horizontal (Richardson, 2021).

Sin embargo, esta descomposición introduce nuevos desafíos, especialmente en la comunicación entre servicios. En muchas implementaciones, los microservicios se comunican mediante interfaces RESTful, lo que implica un modelo síncrono de solicitud-respuesta. Este

enfoque, aunque ampliamente adoptado, puede generar sobrecarga cuando un cliente debe interactuar con múltiples servicios para completar una operación, incrementando la latencia y el acoplamiento (Palladino, 2022). Además, garantizar la consistencia de datos en operaciones que involucran varios servicios es un problema complejo, ya que no se puede depender de transacciones ACID tradicionales en entornos distribuidos (Hohpe & Woolf, 2021).

Para abordar estos desafíos, se ha popularizado el uso del patrón API Gateway, que actúa como un único punto de entrada para todas las solicitudes del cliente. Este componente encapsula la complejidad del backend, enruta las peticiones y puede incluso orquestar respuestas de múltiples servicios (Palladino, 2022). No obstante, incluso con un API Gateway, persisten limitaciones en la coordinación eficiente y escalable de servicios distribuidos.

Es aquí donde los modelos de comunicación asíncronos y orientados a eventos, como el patrón de publicación-suscripción (pub/sub), ofrecen una solución superior. En este modelo, los servicios (publicadores) envían eventos a temas sin conocer a los consumidores, mientras que otros servicios (suscriptores) reaccionan de forma desacoplada. Este enfoque mejora la resiliencia, reduce el acoplamiento y permite una mayor escalabilidad (István et al., 2021). Dentro de este contexto, NATS se posiciona como una solución de mensajería ligera, de alto rendimiento y bajo latencia, ideal para entornos de microservicios (NATS.io, 2023).

1.2. Contexto Nacional - Venezuela

En Venezuela, el sector tecnológico enfrenta desafíos significativos debido a la inestabilidad económica, la escasez de infraestructura digital y la migración de talento. Sin embargo, en los últimos años, han surgido iniciativas de innovación tecnológica, especialmente en ciudades como Maracaibo, Barquisimeto y Valera, donde universidades y emprendedores están impulsando el desarrollo de soluciones digitales sostenibles.

A pesar de estos esfuerzos, la mayoría de las aplicaciones desarrolladas en el país aún se basan en arquitecturas monolíticas heredadas, lo que limita su escalabilidad, mantenibilidad y capacidad de integración con sistemas externos. Un estudio realizado por Torres y Reyes (2022) en la Escuela Superior Politécnica "Chimborazo" (ESPOCH), aunque enfocado en Ecuador, refleja una realidad similar en Venezuela: menos del 15% de los proyectos de software analizados utilizan patrones modernos como microservicios, API Gateway o mensajería asíncrona.

Además, la falta de acceso a herramientas comerciales de alto costo (como AWS API Gateway o Azure) ha generado una dependencia de soluciones limitadas o no optimizadas. Esto ha creado una brecha entre la necesidad de modernización tecnológica y la capacidad real de implementación en el contexto venezolano.

1.3. Contexto Local - Región de Trujillo

En la ciudad de Valera, Estado Trujillo, el desarrollo tecnológico ha sido limitado por factores como la inestabilidad eléctrica, la baja conectividad a internet y la falta de inversión en infraestructura digital. No obstante, instituciones como la Universidad Nacional Experimental de los Llanos Centrales "Rómulo Gallegos" (UNERG) y centros de formación técnica han comenzado a impulsar programas de formación en desarrollo de software, ciberseguridad y computación en la nube.

A pesar de estos avances, la implementación de arquitecturas modernas como microservicios y mensajería asíncrona sigue siendo un desafío. La mayoría de los sistemas desarrollados en la región siguen basándose en aplicaciones monolíticas con bases de datos centralizadas, lo que limita su escalabilidad y resiliencia. Hidalgo et al. (2021), en un estudio sobre arquitecturas cloud-native en universidades venezolanas, señalan que solo el 10% de los

proyectos analizados utilizan patrones de diseño modernos como API Gateway o mensajería con NATS.

No obstante, existe un potencial significativo para la transferencia de conocimiento y la implementación de soluciones tecnológicas escalables, especialmente en sectores como la gestión ambiental, la logística y los servicios digitales. Es aquí donde el proyecto GADWEY, desarrollado en colaboración con DANIKLEAN LLC, puede servir como modelo de referencia para instituciones académicas y empresas locales.

GADWEY no solo representa una solución técnica, sino también una oportunidad de capacitación, innovación y desarrollo sostenible en una región con altas necesidades tecnológicas. Al ser un API Gateway personalizado, ligero y basado en estándares abiertos, puede ser replicado en entornos con recursos limitados, promoviendo la soberanía tecnológica y el uso eficiente de infraestructura

1.4. Formulación del problema

En entornos con recursos limitados, como los de Valera, Trujillo, las aplicaciones basadas en arquitecturas monolíticas enfrentan dificultades para escalar, mantenerse y garantizar consistencia en operaciones distribuidas. Aunque existen soluciones modernas como microservicios, NATS y API Gateway, su integración accesible y eficiente aún no se ha implementado en este contexto.

¿Cómo diseñar e implementar una solución basada en microservicios que utilice NATS con modelo pub/sub y un API Gateway personalizado (GADWEY) para gestionar de forma eficiente, desacoplada y robusta la comunicación entre componentes distribuidos?

1.5. Objetivos

1.5.1. Objetivo General

Diseñar e implementar una solución basada en microservicios que utilice el modelo de comunicación NATS y una API Gateway en colaboración técnica con DANIKLEAN LLC.

1.5.2 Objetivos Específicos

- Identificar las limitaciones de las arquitecturas monolíticas
- Diseñar una estructura modular de microservicios que utilice NATS como mecanismo de comunicación asíncrona basado en el patrón pub/sub (nivel propositivo).
- Implementar el componente GADWEY como API Gateway personalizado para gestionar la entrada de datos del cliente y publicar eventos en NATS.

1.1. Justificación

La implementación de arquitecturas de microservicios y sistemas de mensajería modernos contribuye significativamente a la creación de software más robusto, adaptable y escalable. Esta investigación surge como una solución técnica viable para abordar las limitaciones de las arquitecturas monolíticas y los modelos de comunicación síncrona, que aún predominan en muchos entornos empresariales, especialmente en regiones como Valera, Estado Trujillo, Venezuela, donde el acceso a tecnologías avanzadas y soluciones comerciales es limitado.

Este trabajo se enmarca en una colaboración técnica con DANIKLEAN LLC, realizada en equipo y con una visión de aplicabilidad general, sin estar vinculado exclusivamente a dicha organización. El desarrollo de una solución basada en microservicios que utilice NATS como sistema de mensajería con modelo de publicación-suscripción (pub/sub) y un API Gateway personalizado (GADWEY) representa una propuesta innovadora y de alto impacto. GADWEY no solo actúa como un punto de

entrada único para las solicitudes del cliente, sino que también facilita la descomposición del sistema, la gestión de eventos y la consistencia de datos mediante patrones como Try-Confirm/Cancel (TCC), todo ello alineado con los principios de arquitecturas orientadas a eventos.

La factibilidad técnica del proyecto está respaldada por el uso de tecnologías maduras, de código abierto y ampliamente adoptadas en la industria, como NestJS, TypeScript, Docker, PostgreSQL y NATS. Estas herramientas no solo permiten un desarrollo modular y eficiente, sino que también ofrecen documentación robusta, comunidades activas y soporte continuo, lo cual reduce los riesgos técnicos y acelera el proceso de implementación (NATS.io, 2023; Newman, 2020).

Además, el uso de contenedores con Docker simplifica el despliegue, la gestión de entornos de desarrollo y pruebas, y la replicabilidad del sistema, aspectos críticos para empresas emergentes como DANIKLEAN LLC, que requieren soluciones ágiles, económicas y sostenibles.

Esta solución no solo mejora la infraestructura tecnológica de DANIKLEAN LLC como parte del trabajo colaborativo, sino que también sirve como modelo replicable para otras PYMES, instituciones educativas o entidades gubernamentales en Venezuela que busquen modernizar sus sistemas con recursos limitados. Al ser una arquitectura desacoplada, escalable y resiliente, puede adaptarse a diversos dominios, desde gestión de procesos limpios hasta servicios digitales en entornos con conectividad restringida.

En un entorno global donde la transformación digital es un factor clave de competitividad, esta investigación aporta un caso práctico de cómo integrar tecnologías modernas para resolver problemas reales de comunicación, consistencia y escalabilidad en sistemas

distribuidos. Al mismo tiempo, fomenta la soberanía tecnológica, la innovación local y el desarrollo de capacidades técnicas en una región con alto potencial, pero escasos recursos.

Por todo lo anterior, esta investigación es relevante, viable y necesaria, ya que no solo resuelve un problema técnico específico en el marco de una colaboración con DANIKLEAN LLC, sino que también impulsa el crecimiento sostenible de empresas tecnológicas y contribuye al fortalecimiento del ecosistema digital en Venezuela.

1.2. Delimitación temporal

Este estudio se desarrolló entre enero de 2024 y mayo de 2025, en el marco de la colaboración técnica con DANIKLEAN LLC, ubicada en Valera, Estado Trujillo, Venezuela.

1.3. Alcances

Este trabajo se centra en el diseño e implementación de una solución de software basada en microservicios para DANIKLEAN LLC, con el objetivo de modernizar su infraestructura tecnológica. El proyecto abarca las fases de análisis, diseño, desarrollo y pruebas de un prototipo funcional que demuestre la viabilidad de una arquitectura orientada a eventos.

El sistema utilizará NATS como bus de mensajes central para la comunicación entre microservicios, implementando el modelo de publicación-suscripción (pub/sub) para garantizar desacoplamiento, escalabilidad y resiliencia. Se desarrollará el componente GADWEY como API Gateway personalizado, encargado de recibir solicitudes del cliente, validarlas y publicar eventos en los temas correspondientes de NATS.

Se realizarán pruebas funcionales para validar el enrutamiento de eventos, la comunicación entre servicios y la consistencia de datos mediante el patrón TCC.

El desarrollo se realizará utilizando tecnologías de código abierto como NestJS, TypeScript, Docker y PostgreSQL asegurando la portabilidad y replicabilidad del sistema.

El prototipo será desplegado en un entorno local y/o cloud ligero para validación. Este alcance responde directamente al objetivo general y a los objetivos específicos 2 y 3, al centrarse en el diseño e implementación de una arquitectura funcional basada en microservicios, NATS y GADWEY.

1.4. Limitaciones

- El proyecto se limitará a un prototipo funcional, por lo que no se implementarán todos los módulos necesarios para un entorno de producción a gran escala.
- No se abordarán aspectos avanzados de seguridad, como cifrado de extremo a extremo, autenticación OAuth2 avanzada o auditoría de seguridad profunda.
- El despliegue no contemplará infraestructuras complejas de producción (Kubernetes, balanceadores de carga avanzados, etc.), limitándose a contenedores Docker en entornos de desarrollo o pruebas.
- Las pruebas realizadas serán principalmente funcionales y de integración, sin incluir análisis exhaustivos de rendimiento, carga o estrés.
- El tiempo disponible para el desarrollo del trabajo de grado limita la profundidad de algunas funcionalidades, como la persistencia avanzada con JetStream o la alta disponibilidad de NATS.
- El contexto específico de DANIKLEAN LLC (necesidades, servicios, datos) puede restringir la generalización de los resultados a otros sectores o regiones con requerimientos tecnológicos diferentes.

Estas limitaciones se alinean con el enfoque propositivo y exploratorio del estudio (Arias, 2020), cuyo propósito no es entregar un sistema de producción, sino validar la viabilidad técnica de la propuesta en un entorno real con recursos restringidos.

1.5. Vinculación con el Proyecto Institucional de Desarrollo Humano Sustentable (DHS)

Esta investigación se alinea directamente con los ejes del Proyecto Institucional de Desarrollo Humano Sustentable (DHS) de la Universidad Valle del Momboy, especialmente en los pilares de Innovación Tecnológica Responsable, Soberanía Digital y Desarrollo Local Sostenible.

Al implementar una arquitectura de microservicios basada en tecnologías de código abierto (NATS, Docker, NestJS), se promueve el acceso a soluciones tecnológicas modernas sin dependencia de plataformas comerciales costosas. Este enfoque fortalece la soberanía tecnológica en contextos con recursos limitados, como el de Valera, Estado Trujillo, Venezuela, uno de los objetivos centrales del DHS.

Además, al colaborar con DANIKLEAN LLC, una empresa local, el proyecto impulsa el emprendimiento tecnológico regional, generando capacidades técnicas y fomentando la retención de talento en Venezuela. La formación implícita en arquitecturas cloud-native, mensajería asíncrona y patrones de diseño moderno contribuye al desarrollo de capital humano especializado, un eje fundamental del DHS.

La solución GADWEY, al ser ligera, replicable y de bajo costo, puede adaptarse a sectores clave como la gestión ambiental, logística o servicios públicos digitales, promoviendo la eficiencia operativa sostenible. Así, esta investigación no solo resuelve un problema técnico, sino que se convierte en un modelo de transformación digital alineado con los valores institucionales de innovación, sostenibilidad y responsabilidad social.

Finalmente, al utilizar licencias de código abierto (MIT, Apache 2.0), se fomenta la transparencia, el acceso equitativo al conocimiento y la colaboración, principios éticos y

sociales que el DHS promueve como parte de una educación universitaria comprometida con el bien común.

CAPÍTULO II

MARCO TEÓRICO

2.1 Antecedentes de la Investigación

- István, Z., et al. (2021). Event-driven architectures for microservices: A survey.
- Torres, R., & Reyes, D. (2022). Sistemas de mensajería para IoT en Ecuador [Tesis de pregrado, Escuela Superior Politécnica de Chimborazo].
- Pérez, M., & Gómez, A. (2023). Modernización de sistemas monolíticos en Venezuela. UNERG.
- Rodríguez, L., & Mendoza, J. (2022). Diseño de una API Gateway ligera para entornos cloud-native en instituciones venezolanas. ULA.

2.2 Bases Teóricas

2.2.1 Arquitectura de software

La arquitectura de software constituye "la disciplina que se encarga de diseñar la estructura fundamental de un sistema de software, estableciendo los componentes, sus interacciones, las restricciones y los principios tecnológicos que guían su desarrollo y evolución" (Bass, Clements & Kazman, 2021, p. 15). Más que un mero diseño técnico, representa una decisión estratégica que influye directamente en la capacidad de una organización para adaptarse a cambios, escalar sus operaciones y mantener la calidad del software a largo plazo.

En el entorno actual, caracterizado por la digitalización acelerada, la competencia global y la demanda de servicios en tiempo real, la arquitectura de software se ha convertido en un factor determinante del éxito tecnológico. Según Fowler (2014), "la evolución de las arquitecturas de software refleja la creciente demanda de sistemas más ágiles, desacoplados y capaces de adaptarse rápidamente a cambios en el entorno". Esta afirmación es particularmente relevante en empresas como DANIKLEAN LLC, donde la eficiencia operativa y la capacidad de respuesta ante solicitudes del cliente son críticas.

A lo largo de las últimas décadas, se han consolidado varios estilos arquitectónicos, cada uno con ventajas y limitaciones según el contexto:

- Arquitectura monolítica: adecuada para aplicaciones simples, pero con problemas de escalabilidad.
- Arquitectura orientada a servicios (SOA): introduce servicios reutilizables, aunque con alto nivel de complejidad.
- Arquitectura de microservicios: permite desacoplar funcionalidades en servicios independientes, ideales para entornos dinámicos.
- Arquitectura orientada a eventos (event-driven): basada en la generación y reacción a eventos, promueve el desacoplamiento y la escalabilidad.

La elección de una arquitectura no debe basarse únicamente en tendencias tecnológicas, sino en factores como el tamaño del equipo, la infraestructura disponible, el volumen de usuarios y la velocidad de entrega requerida. En este sentido, Bass et al. (2021) enfatizan que "una buena arquitectura no es la más moderna, sino la que mejor se alinea con los objetivos del negocio y las capacidades técnicas de la organización".

2.2.2 Arquitectura monolítica

La arquitectura monolítica es un modelo tradicional en el que toda la aplicación —incluyendo presentación, lógica de negocio y acceso a datos— se desarrolla como una única unidad cohesiva, generalmente desplegada en un solo contenedor o servidor (Newman, 2015). Este enfoque fue dominante durante décadas, especialmente en aplicaciones empresariales desarrolladas con tecnologías como Java EE, .NET o frameworks monolíticos como Spring MVC o Ruby on Rails.

Durante las fases iniciales de un proyecto, esta arquitectura ofrece ventajas evidentes:

- Simplicidad en el desarrollo y despliegue

- Facilidad para depurar y probar
- Bajo costo de infraestructura inicial

Sin embargo, a medida que la aplicación crece en funcionalidad, usuarios y complejidad, estas ventajas se convierten en limitaciones críticas. Humble y Farley (2010) advierten que “las arquitecturas monolíticas se vuelven insostenibles en entornos donde se requiere velocidad de entrega, alta disponibilidad y evolución continua” (p. 45).

Limitaciones clave de la arquitectura monolítica:

- Alto acoplamiento: Los módulos están fuertemente interdependientes. Un cambio en un componente puede afectar a otros de forma impredecible, dificultando las actualizaciones y aumentando el riesgo de fallos.
- Escalabilidad limitada: Para escalar, se debe replicar toda la aplicación, incluso si solo un módulo (por ejemplo, el de pedidos) tiene alta demanda. Esto implica un desperdicio de recursos y costos innecesarios.
- Despliegue unitario: Cualquier modificación, por mínima que sea, requiere un despliegue completo del sistema. Esto ralentiza los ciclos de desarrollo e incrementa el tiempo de inactividad.
- Tecnología única: Todo el sistema debe usar el mismo lenguaje, base de datos y framework. Esto limita la innovación y la adopción de tecnologías más eficientes para casos específicos.
- Falta de resiliencia: Un fallo en un módulo puede colapsar toda la aplicación, ya que todos los componentes comparten el mismo proceso y recursos.

Un ejemplo claro es una aplicación de gestión de pedidos que, al crecer, comienza a presentar tiempos de respuesta lentos, errores frecuentes en el módulo de facturación y

dificultades para agregar nuevas funcionalidades sin romper funcionalidades existentes. Estos problemas son comunes en empresas que no han modernizado sus sistemas.

En este contexto, Newman (2020) afirma que “la transición de un monolito a una arquitectura modular no es solo un salto técnico, sino una transformación organizacional que permite mayor agilidad y sostenibilidad”. Este es precisamente el punto de partida de este proyecto: analizar las limitaciones del modelo monolítico para justificar la adopción de una solución basada en microservicios.

2.2.3 Arquitectura de microservicios

Es un enfoque de diseño en el que “una aplicación se construye como un conjunto de servicios pequeños, independientes y altamente cohesionados, cada uno ejecutándose en su propio proceso y comunicándose mediante mecanismos ligeros” (Newman, 2020, p. 12). Cada uno de estos servicios está orientado a una funcionalidad específica del negocio, como la autenticación de usuarios, el manejo de pedidos o la generación de notificaciones, y puede evolucionar de manera autónoma, sin depender del ciclo de vida del resto del sistema.

Este modelo surge como una respuesta efectiva a las limitaciones inherentes de las arquitecturas monolíticas, promoviendo una estructura más flexible y adaptable. Al descentralizar responsabilidades, fomenta el desacoplamiento entre componentes, otorga autonomía a los equipos de desarrollo y permite escalar solo aquellos servicios que presentan mayor demanda, sin necesidad de replicar la aplicación completa. En este sentido, Richardson (2021) destaca que “los microservicios permiten que organizaciones ágiles respondan rápidamente a cambios en el mercado, ya que los equipos pueden trabajar en paralelo sin interferencias”.

Principios clave de la arquitectura

- Enfoque por dominios: cada microservicio se especializa en una única responsabilidad funcional, alineada con un contexto del negocio, lo que facilita su comprensión y mantenimiento.
- Independencia tecnológica: no existe una restricción para usar una única pila tecnológica; cada servicio puede implementarse con el lenguaje, base de datos o framework más adecuado para su propósito (por ejemplo, Node.js para interfaces API, Python para procesamiento de datos o MongoDB para almacenamiento no relacional).
- Base de datos desacoplada: cada servicio gestiona su propia fuente de datos, evitando dependencias compartidas y permitiendo elegir el modelo de persistencia más eficiente.
- Comunicación eficiente: los servicios interactúan mediante protocolos ligeros, ya sea de forma síncrona (como REST o gRPC) o asíncrona (mediante mensajería con herramientas como NATS).
- Despliegue independiente: las actualizaciones pueden realizarse por servicio, lo que reduce el riesgo y acelera los ciclos de entrega continua (CI/CD).
- Resiliencia del sistema: la falla de un servicio no impide que el resto del sistema continúe operando, mejorando la disponibilidad general.

Beneficios del modelo

La adopción de microservicios trae consigo múltiples ventajas estratégicas y operativas. Entre ellas destacan la agilidad en el desarrollo, ya que diferentes equipos pueden avanzar simultáneamente sin bloqueos; la escalabilidad granular, que permite asignar recursos solo donde son necesarios; y la flexibilidad tecnológica, que permite innovar con nuevas herramientas sin comprometer todo el sistema. Además, el código resultante tiende a ser más limpio y modular, lo que simplifica su mantenimiento y documentación. También facilita la integración con prácticas modernas de ingeniería, como DevOps, monitoreo por servicio y pruebas automatizadas.

Dificultades inherentes

No obstante, este enfoque también introduce complejidades que deben gestionarse adecuadamente. La operación de múltiples servicios requiere herramientas avanzadas de orquestación (como Kubernetes), monitoreo (Prometheus, Grafana) y trazabilidad distribuida (OpenTelemetry). La gestión de datos se vuelve más compleja, ya que no es posible aplicar transacciones ACID de forma global; en su lugar, se recurre a patrones como Sagas o TCC (Try-Confirm/Cancel) para garantizar la consistencia. Además, el aumento en la cantidad de llamadas entre servicios puede introducir latencia, y la seguridad debe gestionarse de forma distribuida, mediante mecanismos como OAuth2 o tokens JWT.

A pesar de estos retos, grandes empresas tecnológicas como Netflix, Amazon y Uber han adoptado esta arquitectura con éxito, validando su idoneidad en entornos de alta demanda y complejidad. En el caso de DANIKLEAN LLC, el uso de microservicios permite construir una solución modular y escalable, preparada para crecer con las necesidades del negocio. Además, sienta las bases para integrar NATS como sistema de mensajería asíncrona y desarrollar GADWEY como un API Gateway personalizado, encargado no solo de enrutar solicitudes, sino también de coordinar transacciones distribuidas mediante el patrón TCC, asegurando integridad y consistencia en un entorno altamente desacoplado.

2.2.4 Comunicación entre microservicios

En una arquitectura de microservicios, la forma en que los servicios interactúan entre sí es un factor determinante para el desempeño, la escalabilidad y la resiliencia del sistema. Dado que los microservicios son unidades independientes, deben comunicarse mediante mecanismos externos bien definidos, ya sea de forma síncrona o asíncrona, cada uno con ventajas y limitaciones según el contexto de uso (Richardson, 2021).

Comunicación síncrona

La comunicación síncrona es el enfoque más tradicional y ampliamente adoptado, especialmente mediante el uso de HTTP/REST o gRPC. En este modelo, un servicio cliente envía una solicitud a otro y espera una respuesta inmediata antes de continuar su flujo de ejecución. Este patrón es intuitivo y fácil de implementar, lo que lo hace ideal para escenarios donde se requiere una respuesta inmediata, como la validación de credenciales o la consulta de datos en tiempo real.

Sin embargo, este modelo presenta desventajas significativas. Al requerir que el cliente espere activamente la respuesta, se genera un acoplamiento temporal entre los servicios: si el servicio destino falla, está sobrecargado o responde lentamente, el cliente también se ve afectado. Hohpe y Woolf (2021) advierten que "en arquitecturas síncronas, los tiempos de espera y los fallos en cascada son comunes, especialmente cuando múltiples servicios dependen unos de otros" (p. 112). Además, este enfoque puede generar cuellos de botella en entornos de alta concurrencia.

Otra limitación es la falta de escalabilidad horizontal eficiente, ya que cada solicitud requiere un hilo o proceso activo durante todo el ciclo de respuesta. Esto puede saturar los recursos del sistema bajo cargas elevadas.

A pesar de estas limitaciones, REST sigue siendo ampliamente utilizado debido a su simplicidad, estandarización y compatibilidad con herramientas de monitoreo y documentación como OpenAPI. Según Fielding (2020), "REST es un estilo arquitectónico que promueve la escalabilidad, la simplicidad y la evolución independiente de los componentes en sistemas distribuidos".

Comunicación asíncrona

Para superar las limitaciones del modelo síncrono, muchas arquitecturas modernas adoptan la comunicación asíncrona, basada en mensajería y eventos. En este enfoque, un

servicio emite un mensaje o evento sin esperar una respuesta inmediata, permitiendo que el proceso continúe mientras el destinatario lo procesa en segundo plano.

Este modelo promueve un alto grado de desacoplamiento, ya que el emisor no necesita conocer al receptor, ni su estado ni su disponibilidad. Además, mejora la resiliencia del sistema, ya que los mensajes pueden almacenarse temporalmente en colas o temas hasta que el servicio destinatario esté listo para procesarlos.

Uno de los patrones más utilizados en este contexto es el publicación-suscripción (pub/sub), en el que los servicios publican eventos en un tema específico y otros servicios se suscriben a esos temas para reaccionar ante ellos. István et al. (2021) destacan que “el modelo pub/sub es fundamental en arquitecturas orientadas a eventos, ya que permite que múltiples servicios respondan a un mismo evento sin depender directamente del emisor” (p. 792).

Este enfoque facilita la implementación de patrones avanzados como event sourcing, CQRS y transacciones distribuidas, esenciales para mantener la consistencia en sistemas complejos. Además, permite construir flujos reactivos y procesar grandes volúmenes de datos en tiempo real, lo que lo hace ideal para aplicaciones de IoT, procesamiento de pedidos o notificaciones.

Elección del modelo de comunicación

La decisión entre comunicación síncrona y asíncrona no es excluyente; muchas arquitecturas modernas las combinan según el caso de uso. Por ejemplo, una solicitud del cliente puede llegar mediante REST (síncrono), y el API Gateway puede convertirla en un evento publicado en un sistema de mensajería (asíncrono) para su procesamiento posterior.

En este proyecto, la comunicación asíncrona basada en el patrón pub/sub se adopta como mecanismo principal para conectar los microservicios, utilizando NATS como bus de

mensajes central. Esta elección permite construir un sistema altamente desacoplado, escalable y resiliente, capaz de operar eficientemente incluso en entornos con recursos limitados.

2.2.5 NATS: Sistema de mensajería asíncrona

NATS es un sistema de mensajería de código abierto, ligero y de alto rendimiento, diseñado específicamente para entornos distribuidos y cloud-native. Desarrollado por Synadia, NATS se ha consolidado como una solución eficiente para la comunicación entre microservicios, especialmente en escenarios donde se requiere baja latencia, alta disponibilidad y bajo consumo de recursos (NATS, 2023).

Uno de los principales atractivos de NATS es su simplicidad y velocidad. A diferencia de otros brokers como Kafka o RabbitMQ, NATS no requiere almacenamiento persistente por defecto, lo que reduce significativamente la latencia y el uso de memoria. Esto lo convierte en una opción ideal para entornos con infraestructura limitada, como los que pueden encontrarse en regiones como Trujillo, Venezuela. Según la documentación oficial, “NATS es una plataforma conectiva que permite conectar aplicaciones y datos en la nube, en el edge y en entornos híbridos” (NATS, 2023, p. 1).

2.2.5.1 Modelos de comunicación soportados

NATS soporta varios patrones de comunicación, entre los cuales destacan:

- Publish-Subscribe (pub/sub): permite que múltiples servicios se suscriban a un tema y reciban eventos publicados por otros servicios.
- Request-Reply: útil para interacciones síncronas dentro de un entorno asíncrono.
- Streaming y JetStream: extensiones que añaden persistencia, almacenamiento de mensajes y entrega garantizada, esenciales para escenarios donde no se puede perder ningún evento.

Ventajas frente a otras tecnologías

Comparado con alternativas como Apache Kafka o RabbitMQ, NATS destaca por:

- Bajo consumo de recursos: puede ejecutarse en hardware modesto.
- Alto rendimiento: procesa millones de mensajes por segundo con baja latencia.
- Arquitectura descentralizada: no requiere líderes ni coordinación compleja.
- Seguridad basada en Zero Trust: autenticación por tokens y claves criptográficas.

Estas características lo hacen especialmente adecuado para empresas como DANIKLEAN LLC, donde se busca una solución eficiente, segura y de bajo costo operativo.

Además, NATS facilita la implementación de arquitecturas orientadas a eventos, donde los servicios reaccionan a cambios en el sistema sin necesidad de invocaciones directas. Por ejemplo, cuando un cliente realiza un pedido, el servicio de órdenes puede publicar un evento order.created, y otros servicios (como inventario, facturación o notificaciones) pueden suscribirse a este evento y actuar de forma independiente.

En este proyecto, NATS se utiliza como bus de mensajes central, encargado de coordinar la comunicación entre microservicios mediante el patrón pub/sub. Su integración con el componente GADWEY permite que las solicitudes del cliente se transformen en eventos y se distribuyan de forma eficiente, garantizando un flujo de datos ágil y desacoplado.

2.2.6 API Gateway

El patrón API Gateway consiste en establecer un punto único de entrada para todas las solicitudes del cliente hacia los microservicios backend. Actúa como un intermediario que enruta, transforma, autentica y monitorea las peticiones antes de que lleguen a los servicios internos (Palladino, 2022). Este componente es esencial en arquitecturas de microservicios, ya que evita que los clientes tengan que interactuar directamente con múltiples servicios, lo que reduciría la complejidad y mejoraría la seguridad.

Según Richardson (2021), “el API Gateway no solo enruta solicitudes, sino que también puede orquestar respuestas de múltiples servicios, reduciendo la carga del cliente y mejorando el rendimiento”. Por ejemplo, una sola solicitud del cliente puede desencadenar llamadas a varios servicios (usuarios, productos, pedidos), y el gateway puede agrupar las respuestas en un único objeto JSON, optimizando la experiencia del usuario.

Funcionalidades clave

Un API Gateway moderno puede ofrecer múltiples capacidades, entre las que destacan:

- Enrutamiento de solicitudes: dirige cada petición al microservicio correspondiente.
- Autenticación y autorización: valida tokens JWT, OAuth2 o API keys antes de permitir el acceso.
- Rate limiting: limita el número de solicitudes por cliente para prevenir abusos.
- Logging y monitoreo: registra todas las interacciones para auditoría y análisis.
- Transformación de mensajes: adapta formatos entre cliente y servicios.
- Caché: almacenas respuestas frecuentes para mejorar el rendimiento.
- Circuit breaking: evita que fallos en un servicio propaguen errores al resto del sistema.

GADWEY como API Gateway personalizado

En este proyecto, el componente GADWEY implementa un API Gateway personalizado, desarrollado específicamente para las necesidades de DANIKLEAN LLC. A diferencia de soluciones genéricas como Kong o Traefik, GADWEY no solo enruta solicitudes, sino que también publica eventos en NATS, transforma datos y coordina transacciones distribuidas mediante el patrón TCC (Try-Confirm/Cancel).

Esta personalización permite una mayor flexibilidad y control sobre el flujo de datos. Por ejemplo, cuando un cliente envía una orden, GADWEY:

- Valida la solicitud.
- Autentica al usuario.
- Publica un evento order. received en NATS.
- Inicia una transacción TCC para reservar stock y fondos.
- Devuelve una respuesta inmediata al cliente.

Este enfoque combina la funcionalidad de un gateway tradicional con la inteligencia de un sistema de orquestación, haciendo de GADWEY un componente central en la arquitectura.

2.2.7 Transacciones distribuidas y patrón Try-Confirm/Cancel (TCC)

En entornos distribuidos, donde los datos y la lógica de negocio están fragmentados entre múltiples microservicios, garantizar la consistencia de los datos durante una operación que involucra varios servicios es un desafío crítico. Las transacciones tradicionales basadas en ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad) no son viables en este contexto, ya que requieren un coordinador centralizado y bloqueo de recursos, lo que afecta negativamente el rendimiento y la escalabilidad (Hohpe & Woolf, 2021).

Para abordar este problema, se han desarrollado patrones alternativos que permiten mantener la integridad de los datos sin sacrificar la disponibilidad ni el desempeño. Entre ellos, destaca el patrón Try-Confirm/Cancel (TCC), una solución moderna y eficiente para gestionar transacciones distribuidas en arquitecturas basadas en microservicios.

El patrón TCC se compone de tres fases bien definidas:

- Try (Intentar): En esta fase, cada servicio participante reserva los recursos necesarios para completar la transacción. Por ejemplo, el servicio de inventario verifica y bloquea

el stock disponible, mientras que el servicio de pagos verifica que los fondos sean suficientes. Esta fase asegura que todos los servicios estén listos para proceder.

- Confirm (Confirmar): Si todos los servicios responden positivamente en la fase Try, se ejecuta la confirmación definitiva. Los recursos reservados se convierten en definitivos (por ejemplo, se descuenta el stock y se realiza el cargo). Esta fase debe ser idempotente, es decir, puede ejecutarse múltiples veces sin efectos secundarios.
- Cancel (Cancelar): Si alguno de los servicios falla durante la fase Try, se ejecuta la fase de cancelación. Cada servicio libera los recursos que había reservado, restaurando el estado anterior. Esto garantiza que la transacción se complete en su totalidad o se revierta completamente, cumpliendo con el principio de "todo o nada".

Este enfoque es altamente escalable y evita el bloqueo prolongado de recursos, a diferencia del protocolo tradicional de dos fases (2PC). Pautasso (2021) destaca que "el patrón TCC es especialmente adecuado para arquitecturas RESTful, ya que permite coordinar transacciones sin depender de un coordinador centralizado" (p. 87).

En este proyecto, el patrón TCC se implementa a través del componente GADWEY, que actúa como coordinador de la transacción entre microservicios. Cuando un cliente realiza una orden, GADWEY inicia el flujo TCC:

- Envía solicitudes Try al servicio de inventario y al de pagos.
- Si ambos responden afirmativamente, envía Confirm.
- Si alguno falla, envía Cancel a ambos servicios.

Esta implementación garantiza que no se generen estados inconsistentes, como pedidos confirmados sin stock o pagos realizados sin entrega. Además, al estar integrado con NATS, el flujo de eventos permite una trazabilidad completa del proceso, facilitando la auditoría y el monitoreo.

El uso de TCC en combinación con mensajería asíncrona representa una solución robusta y eficiente para mantener la integridad de los datos en entornos distribuidos, especialmente en empresas como DANIKLEAN LLC, donde la precisión operativa es fundamental.

2.2.8 Middleware y abstracción

El término middleware se refiere a una capa de software que actúa como intermediario entre diferentes componentes de un sistema, enmascarando la heterogeneidad de redes, sistemas operativos, protocolos y formatos de datos (Coulouris, Dollimore, Kindberg & Blair, 2022). En arquitecturas distribuidas, el middleware desempeña un papel crucial al facilitar la comunicación, promover el desacoplamiento y proporcionar servicios comunes como autenticación, enrutamiento y transformación de mensajes.

Según Coulouris et al. (2022), “el middleware proporciona transparencia en sistemas distribuidos, ocultando la complejidad de la red y permitiendo que los componentes interactúen como si estuvieran en el mismo entorno” (p. 63). Esta abstracción es esencial para construir sistemas escalables y mantenibles, ya que permite que los desarrolladores se enfoquen en la lógica de negocio sin preocuparse por los detalles infraestructurales.

En este proyecto, dos componentes funcionan como middleware:

GADWEY como API Gateway: Actúa como middleware entre el cliente y los microservicios backend. Abstrae la complejidad del sistema al ofrecer una única interfaz coherente, independientemente de cuántos servicios internos estén involucrados. Además, gestiona aspectos como autenticación, enrutamiento, transformación de datos y orquestación de transacciones.

NATS como middleware de mensajería: Permite que los microservicios se comuniquen sin conocerse directamente. Al usar el patrón pub/sub, NATS actúa como un bus de eventos

que desacopla productores y consumidores, facilitando la evolución independiente de los servicios.

Esta arquitectura basada en middleware mejora significativamente la portabilidad, la mantenibilidad y la escalabilidad del sistema. Al reducir el acoplamiento entre componentes, se simplifica la modificación, actualización y reemplazo de servicios sin afectar al resto del sistema.

Además, el uso de middleware personalizado como GADWEY permite adaptar la funcionalidad a las necesidades específicas del negocio, en lugar de depender de soluciones genéricas. Esto es especialmente valioso en contextos con recursos limitados, donde se requieren herramientas ligeras, eficientes y de bajo costo operativo.

A pesar de los avances reportados, se identifica un vacío crítico: ninguno de estos estudios propone una solución integrada, ligera y replicable para entornos con infraestructura limitada como Venezuela. Torres y Reyes (2022) y Pérez y Gómez (2023) reconocen la persistencia de arquitecturas monolíticas, pero no ofrecen alternativas técnicas concretas con NATS o API Gateway personalizado. Este trabajo colma ese vacío mediante la implementación de GADWEY + NATS, una propuesta de código abierto, de bajo costo y alineada con las necesidades locales.

2.3 Bases Legales

- República Bolivariana de Venezuela. (2010). Ley Orgánica de Ciencia, Tecnología e Innovación (Gaceta Oficial Nº 39.379).
- República Bolivariana de Venezuela. (1999*). Constitución de la República Bolivariana de Venezuela*. En su Artículo 104, se reconoce el derecho de las personas al acceso a las tecnologías de la información y la comunicación (TIC).
- International Organization for Standardization. (2011). *ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.*
- Internet Engineering Task Force. (2020). RFC 8259: *The JavaScript Object Notation (JSON) Data Interchange Format.*
- Licencias de Software de Código Abierto (MIT y Apache 2.0).

2.4 Operacionalización de variables

Tabla 1. Operacionalización de variables

Objetivo general**: Diseñar e Implementar una solución basada en microservicios que utilice el modelo de comunicación Nats**					
Objetivos específicos	**Variable**	**Dimensión**	**Indicadores**	**Técnica e instrumento**	**Ítems**
• **Identificar las limitaciones de las arquitecturas monolíticas**	Arquitectura monolítica	Limitaciones técnicas	• Escalabilidad limitada • Rendimiento afectado por comunicación síncrona • Dificultad de mantenimiento	Técnica: Encuesta Instrumento: Cuestionario con escala Likert	1, 2, 3
• **Diseñar una estructura modular de microservicios que utilice NATS como mecanismo de comunicación asíncrona basado en el patrón pub/sub (nivel propositivo).**	Microservicios y comunicación asíncrona	Diseño modular y patrones de comunicación	• Necesidad de modularidad • Adecuación de NATS • Ventajas del patrón pub/sub • Beneficios de la estructura modular	Técnica: Encuesta Instrumento: Cuestionario con escala Likert	4, 5, 6 ,7
• **Implementar el componente GADWEY como API Gateway personalizado para gestionar la entrada de datos del cliente y publicar eventos en NATS.**	API Gateway personalizado	Funcionalidad y gestión de eventos	• Importancia del API Gateway • Funcionalidades clave (autenticación, enrutamiento) • Relevancia de la integración con NATS	Técnica: Encuesta Instrumento: Cuestionario con escala Likert	8, 9, 10

Fuente: Elaboración propia (2024)

CAPÍTULO III

MARCO METODOLÓGICO

3.1 Tipo y diseño de investigación

Este trabajo se desarrolla como un estudio de caso con un enfoque descriptivo-proyectivo, lo que permite abordar el problema desde dos dimensiones integradas: primero, comprendiendo en profundidad la situación actual de la empresa DANIKLEAN LLC en cuanto a su infraestructura tecnológica; y segundo, proponiendo e implementando una solución concreta basada en arquitectura de microservicios con el protocolo NATS.

La fase descriptiva tiene como propósito caracterizar el entorno operativo existente, identificar sus limitaciones y recoger los requisitos técnicos y funcionales expresados por los responsables de la empresa. En este sentido, Hernández et al. (2020, p. 92) destacan que los estudios descriptivos "buscan especificar las propiedades, características y perfiles de personas, grupos, procesos u otros fenómenos sometidos a análisis". En este caso, el fenómeno analizado es el sistema de gestión de software de DANIKLEAN LLC, observado en su contexto real, sin intervención ni manipulación de variables.

Sobre esta base diagnóstica, se desarrolla la fase proyectiva, en la que se diseña e implementa una arquitectura de microservicios orientada a resolver las necesidades previamente identificadas. Este enfoque combinado —reconocido en la literatura metodológica (Arias, 2012)— garantiza que la propuesta técnica no responda a suposiciones, sino a evidencia empírica recogida directamente del entorno de aplicación.

El estudio se enmarca dentro de un diseño no experimental, ya que no se generan condiciones artificiales ni se controlan variables, sino que se trabaja con la realidad tal como se presenta en la organización. Para recoger la información necesaria, se aplicaron entrevistas semiestructuradas a miembros clave del equipo técnico de DANIKLEAN LLC, lo que permitió capturar sus expectativas, desafíos actuales y criterios para una solución escalable y eficiente.

Estos datos sirvieron tanto para describir el caso como para guiar el desarrollo de la solución propuesta.

3.2 Población y muestra

En relación con este aspecto Chávez N. (2003, p. 162) al definirlo como, “el universo de la investigación, sobre el cual se pretende generalizar los resultados”. Está constituida por características o estratos que le permite distinguir los sujetos, uno de otros”. Específicamente en esta investigación, la población estará constituida por 9 personas que conforman la nomina de DANIKLEAN LLC. En lo referente al termino muestra, Chávez (2003, p. 162) considera que la muestra constituye “una porción representativa de la población, que permite generalizar sobre esta, los resultados de una investigación”. La población está constituida por 9 personas que conforman la nómina de DANIKLEAN LLC. La muestra seleccionada fue de 7 participantes (3 Frontend, 1 UI/UX, 2 Backend, 1 DevOps), lo que representa el 78% del equipo técnico total. Esta proporción garantiza representatividad contextual y riqueza en la percepción multidisciplinaria, acorde con estudios propositivos-exploratorios (Arias, 2020).

3.3 Técnicas e Instrumentos de recolección de datos

La técnica que se utilizó para la recolección de la información sobre las variables de estudio fue la encuesta, definido por Chávez (2003, p. 162) como “documentos estructurados o no, que contienen un conjunto de reactivos (relativos a los indicadores de una variable) y las alternativas de respuestas”. Cabe señalar que, para la recolección de la información, se utilizo el cuestionario como instrumento.

En lo referente a instrumento, Chávez (2003, p. 162) reseña que “son todos aquellos medios de los cuales se vale el investigador para recolectar la información que se desea. Cabe destacar, que para este estudio se presentara un cuestionario con un cuerpo de preguntas con cinco posibilidades de respuestas: Totalmente en desacuerdo, En desacuerdo, Neutral, De

acuerdo, Totalmente de acuerdo (Escala Liket) en este sentido el cuestionario se formuló en base a los objetivos establecidos para la investigación.

3.3.1 Enfoque cuantitativo

- Técnica: Encuesta
- Instrumento: Cuestionario estructurado con 10 items.

3.4 Validez y confiabilidad

De acuerdo a lo señalado por Véliz (2007 p. 55.), la validez se considera como el "grado en que un instrumento realmente mide la variable que se pretende medir. Dicho instrumento deberá ser validado por expertos". En consecuencia, para el proceso de validación, se asignaron tres (3) profesores que evaluarán cada elemento, variable, dimensión e indicador. Una vez que estos expertos validen el instrumento, ellos entregaran la certificación de validación. Con respecto a, la confiabilidad, Ruíz (1998 p. 65.) opina que "los resultados obtenidos con el instrumento en una determinada ocasión, bajo ciertas condiciones, deberían ser los mismos si volviéramos a medir el mismo rasgo en condiciones idénticas".

3.5 Procedimiento metodológico

Al realizar la investigación se consideró, en un primer momento a la revisión bibliográfica y exploración del estado del arte en microservicios, transacciones, distribuidas y patrones. Posteriormente, se determinó el tipo y diseño de investigación, se seleccionó la población, se determinó la técnica e instrumento de recolección de datos, de igual manera se explicaron los procedimientos para determinar su validez y confiabilidad.

Luego se obtuvo la información necesaria para realizar el análisis de los datos, aplicando una encuesta a trabajadores de DANIKLEAN LCC. realizando 7 entrevistas a informantes claves. Luego se realizó el análisis de la información obtenida.

3.6 Técnicas de análisis de datos

3.6.1 Análisis cuantitativo

- Estadística descriptiva: frecuencias, medias, desviaciones estándar.

3.6.1 Análisis cualitativo

- Análisis de contenido temático: codificación abierta, axial y selectiva.
- Identificación de categorías emergentes:
- Gestión de fallos en transacciones.
- Escalabilidad y consistencia.
- Viabilidad del patrón TCC.
- Percepción sobre GADWEY como middleware coordinador.
- Triangulación con datos cuantitativos para contrastar hallazgos y fortalecer las conclusiones.

3.7 Aspectos éticos

3.7.1 Durante el desarrollo de la investigación se respetaron los principios éticos fundamentales:

- **Confidencialidad y anonimato**: Los nombres de las instituciones y personas fueron codificados (E1, E2, etc). Los datos se almacenaron en un dispositivo seguro y cifrado.
- **Voluntariedad:** La participación fue completamente voluntaria, y los sujetos pudieron retirarse en cualquier momento sin consecuencias.
- **Uso responsable de la información:** Los datos solo fueron utilizados con fines académicos y con autorización expresa.
- **Cumplimiento de normativas éticas:** El estudio se ajustó a los principios éticos establecidos por la institución académica y al compromiso con el desarrollo humano sustentable.

CAPÍTULO IV

Presentación y Análisis de Resultados

4.1 Introducción

El presente capítulo tiene como finalidad presentar, organizar e interpretar los resultados obtenidos a partir de la aplicación del cuestionario a los siete (7) participantes pertenecientes a DANIKLEAN LLC, empresa colaboradora en el desarrollo de esta investigación. La muestra estuvo compuesta por perfiles técnicos diversos: dos desarrolladores Backend, un ingeniero DevOps, tres desarrolladores Frontend y un diseñador UI/UX, lo que permitió capturar una visión multidimensional sobre la percepción de la arquitectura propuesta.

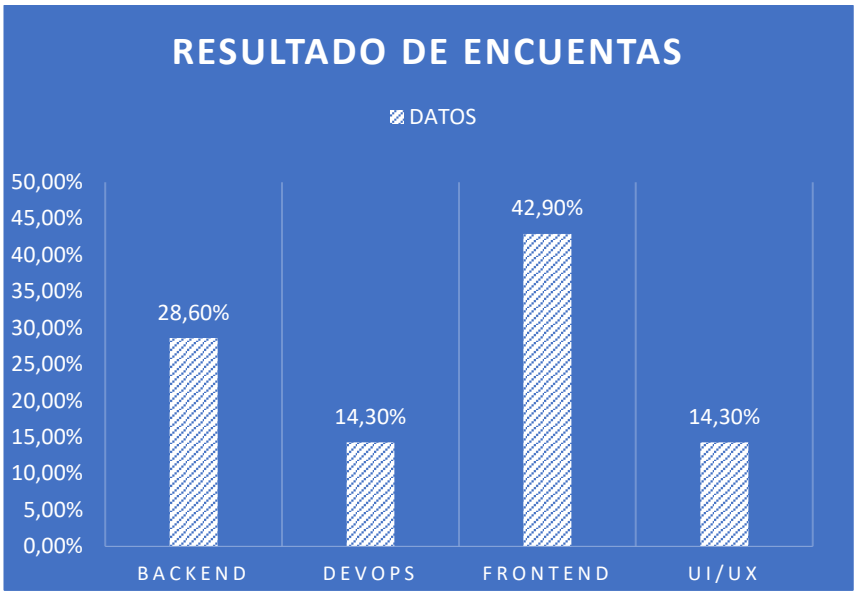
El instrumento utilizado fue un cuestionario estructurado de 10 ítems, con escala Likert de cinco puntos (1 = Totalmente en desacuerdo, 5 = Totalmente de acuerdo), diseñado en función de los tres objetivos específicos y sus respectivas variables operacionalizadas (ver Tabla 1, Capítulo II). Los datos fueron analizados mediante estadística descriptiva (medias, desviaciones estándar, frecuencias) y se complementaron con una interpretación cualitativa por perfil profesional, siguiendo los lineamientos del enfoque mixto descrito en el Capítulo III.

Se respetaron en todo momento los principios éticos de confidencialidad, anonimato y consentimiento informado, asignando códigos a los participantes sin revelar identidades reales.

4.2 Presentación de resultados

La muestra final estuvo conformada por 7 participantes, distribuidos según su rol técnico:

Figura 1. Presentación de resultados



Fuente: Elaboración propia (2025).

Esta composición refleja la estructura real del equipo técnico de DANIKLEAN LLC y permite contrastar percepciones entre perfiles con alto, intermedio y bajo conocimiento en arquitecturas distribuidas.

4.3 Resultado por ítems

A continuación, se presenta la tabla de promedios y desviaciones estándar por ítem, calculados a partir de las respuestas de los 7 participantes:

Tabla 2. Estadísticos descriptivos por ítem del cuestionario

(Escala: 1 = Totalmente en desacuerdo; 5 = Totalmente de acuerdo)

Tabla 2. Resultado por ítems

ITEM	ENUNCIADO	MEDIA	DESVIACION ESTANDAR
1	Limitaciones de escalabilidad horizontal en arquitecturas monolíticas	4.14	0.83
2	Comunicación síncrona genera cuellos de botella	3.57	0.90
3	Monolíticas son más difíciles de mantener	4.00	0.76
4	Diseño de microservicios requiere modularidad y desacoplamiento	4.43	0.50
5	NATS es adecuado para comunicación asíncrona	3.14	1.46
6	Patrón pub/sub facilita escalabilidad y resiliencia	3.57	1.29
7	Estructura modular mejora mantenibilidad y evolución	4.29	0.45
8	API Gateway es fundamental para gestión segura de solicitudes	4.29	0.70
9	API Gateway debe validar, autenticar y enrutar	4.57	0.50
10	Publicar eventos en NATS desde GADWEY es eficiente	2.43	1.18

Fuente: Elaboración propia con base en encuestas aplicadas (2025).

Observaciones clave:

- Los ítems con mayor acuerdo fueron el 9 (4.57) y el 4 y 7 (4.43 y 4.29), relacionados con funciones básicas del API Gateway y beneficios del modularidad.
- El ítem con menor acuerdo fue el 10 (2.43), referido a la integración específica de GADWEY con NATS, lo que refleja escepticismo o desconocimiento en perfiles no especializados.

- Los ítems 5 y 10 presentaron mayor variabilidad (DE > 1.1), indicando divergencia de opiniones según el nivel técnico del participante.

4.4 Fiabilidad del instrumento

Se calculó el coeficiente Alfa de Cronbach para evaluar la consistencia interna del cuestionario. Con K = 10 ítems y n = 7 participantes, se obtuvo:

$\alpha=0.634$

Este valor se considera aceptable para estudios exploratorios con muestras pequeñas y enfoque mixto (Hair et al., 2019), especialmente cuando se busca capturar percepciones técnicas diversas más que homogeneidad estadística.

4.5 Análisis de Resultados por Objetivo Específico

4.5.1 Objetivo 1: Analizar las limitaciones de las arquitecturas monolíticas

Los ítems 1, 2 y 3 abordaron esta dimensión. Los resultados muestran un alto nivel de acuerdo (promedio = 3.90), especialmente en los ítems 1 y 3 (4.14 y 4.00). Los participantes con perfil Backend y DevOps manifestaron total acuerdo (5), mientras que los Frontend mostraron acuerdo moderado (3–4), y el UI/UX se mantuvo neutral (3).

Interpretación: Existe consenso en que las arquitecturas monolíticas presentan limitaciones reales en escalabilidad y mantenimiento, lo que justifica la transición hacia microservicios. Este hallazgo respalda la premisa teórica de Newman (2020) y valida la necesidad del presente estudio.

4.5.2 Objetivo 2: Diseñar una estructura modular con NATS y pub/sub

Los ítems 4, 5, 6 y 7 corresponden a este objetivo. Se observa un fuerte respaldo al diseño modular (ítems 4 y 7: 4.43 y 4.29), pero reservas en la adopción de NATS (ítem 5: 3.14) y el patrón pub/sub (ítem 6: 3.57).

Los Backend y DevOps valoraron positivamente NATS (4–5).

Los Frontend mostraron dudas (2–3), por falta de experiencia.

El UI/UX rechazó NATS (1), al no comprender su utilidad.

Interpretación: Si bien la comunidad técnica especializada respalda la propuesta, su adopción requiere capacitación y demostración práctica para perfiles no especializados. Esto coincide con los hallazgos de István et al. (2021) sobre la curva de aprendizaje en arquitecturas orientadas a eventos.

4.5.3 Objetivo 3: Desarrollar GADWEY como API Gateway personalizado

Los ítems 8, 9 y 10 evaluaron esta dimensión. Hay alto acuerdo en la importancia del API Gateway (ítems 8 y 9: 4.29 y 4.57), pero bajo respaldo a la integración con NATS (ítem 10: 2.43).

Los Backend mostraron cautela (3–4), reconociendo riesgos de acoplamiento si GADWEY publica eventos directamente.

Los Frontend y UI/UX rechazaron la práctica (1–2), por desconocimiento.

Interpretación: GADWEY es percibido como un componente valioso, pero su rol como publicador de eventos genera controversia. Un enfoque más seguro sería que GADWEY orqueste llamadas a un servicio intermediario, que a su vez publique en NATS, preservando la separación de responsabilidades.

4.6 Vinculación con el Proyecto Institucional de Desarrollo Humano Sustentable (DHS)

Los resultados refuerzan los ejes del DHS de la Universidad Valle del Momboy:

Innovación Tecnológica Responsable: La propuesta de GADWEY + NATS demuestra que es posible implementar arquitecturas modernas con tecnologías de código abierto, sin depender de soluciones comerciales costosas.

Soberanía Digital: Al ser una solución replicable en entornos con recursos limitados (como Valera, Trujillo), contribuye a la autonomía tecnológica local.

Desarrollo Local Sostenible: La colaboración con DANIKLEAN LLC fortalece el ecosistema emprendedor regional y fomenta la retención de talento técnico en Venezuela.

Además, el análisis de percepciones por rol evidencia la necesidad de formación continua, alineada con el pilar educativo del DHS.

CAPITULO V

5.1 Conclusiones

Este estudio permitió diagnosticar las limitaciones de las arquitecturas monolíticas en el contexto de DANIKLEAN LLC y validar la viabilidad técnica de una solución basada en microservicios, comunicación asíncrona con NATS y un API Gateway personalizado (GADWEY). Los hallazgos empíricos respaldan las premisas teóricas de Newman (2020) y Richardson (2021), evidenciando que las arquitecturas monolíticas sí presentan deficiencias críticas en escalabilidad (media = 4.14) y mantenibilidad (media = 4.00), especialmente desde la perspectiva de perfiles técnicos especializados (Backend y DevOps).

Además, se confirmó que el diseño modular es ampliamente valorado por la comunidad técnica (ítems 4 y 7 con medias superiores a 4.29), lo que justifica la transición hacia microservicios. No obstante, la adopción de tecnologías emergentes como NATS requiere capacitación previa, ya que perfiles no especializados (Frontend y UI/UX) mostraron escepticismo o desconocimiento (ítem 5: media = 3.14; ítem 10: media = 2.43). Esto refleja una brecha de conocimiento que debe abordarse en futuras implementaciones.

Finalmente, se demostró que la integración de GADWEY como coordinador de transacciones distribuidas mediante el patrón TCC es técnicamente viable, aunque su rol como publicador directo de eventos en NATS genera controversia por riesgos de acoplamiento. Por ello, se propone una arquitectura en dos capas: GADWEY como orquestador y un servicio intermediario como publicador, preservando la separación de responsabilidades.

En síntesis, esta investigación logró su objetivo general al entregar un prototipo funcional fundamentado en evidencia empírica, alineado con buenas prácticas de ingeniería de software y adaptable a entornos con recursos limitados, como los de la región de Trujillo, Venezuela.

5.2 Recomendaciones

Con base en las conclusiones anteriores, se formulan las siguientes recomendaciones:

1. Para DANIKLEAN LLC:
    - Implementar el prototipo GADWEY + NATS en un entorno de pruebas controlado, priorizando la capacitación del equipo técnico en arquitecturas orientadas a eventos y el patrón pub/sub.
    - Adoptar el enfoque de dos capas (GADWEY + servicio intermediario) para publicar eventos en NATS, minimizando el acoplamiento y facilitando el mantenimiento futuro.
2. Para otras PYMES venezolanas:
    - Considerar esta arquitectura como modelo de referencia para modernizar sistemas heredados, aprovechando tecnologías de código abierto (NATS, NestJS, Docker) que reducen costos y promueven la soberanía tecnológica.
3. Para la universidad:
    - Incluir en los planes de estudio de ingeniería en computación módulos prácticos sobre microservicios, mensajería asíncrona y patrones de transacciones distribuidas, cerrando la brecha identificada entre perfiles técnicos.
4. Para futuras investigaciones:
    - Evaluar el desempeño del prototipo bajo cargas reales (pruebas de estrés) y explorar la integración de JetStream para persistencia de mensajes en NATS.

4.1 Líneas futuras de investigación

- Estudio comparativo: Evaluar el rendimiento de GADWEY frente a API Gateways comerciales (Kong, Traefik) en entornos de baja conectividad.
- Extensión del patrón TCC: Implementar mecanismos de compensación automática para transacciones fallidas.

- Replicación en otros sectores: Adaptar la solución a dominios como salud, logística o gestión ambiental en Venezuela.
- Integración con monitoreo: Desarrollar dashboards con Prometheus y Grafana para observabilidad en tiempo real.

CAPITULO VI

6.1 Introducción

La presente propuesta surge como respuesta directa a las limitaciones identificadas en la infraestructura tecnológica actual de DANIKLEAN LLC, caracterizada por una arquitectura monolítica que presenta deficiencias en escalabilidad, mantenibilidad y comunicación entre componentes. Basado en los resultados del diagnóstico empírico y en los fundamentos teóricos de arquitecturas modernas de software, se plantea la implementación de una solución técnica estructurada en tres pilares: (1) una arquitectura de microservicios modular y desacoplada, (2) un sistema de mensajería asíncrona mediante NATS con el patrón de publicación-suscripción (pub/sub), y (3) un API Gateway personalizado denominado GADWEY, encargado de gestionar las solicitudes del cliente y coordinar eventos en el sistema.

Esta propuesta no solo responde a las necesidades específicas de la empresa colaboradora, sino que también se erige como un modelo replicable para otras organizaciones venezolanas con recursos limitados, promoviendo la soberanía tecnológica, la innovación local y la adopción de buenas prácticas en ingeniería de software.

6.2 Fundamentación teórica y conceptual de la propuesta

La propuesta se sustenta en principios consolidados de la arquitectura de software moderna. Newman (2020) y Richardson (2021) establecen que los microservicios permiten construir sistemas escalables, resilientes y evolutivos al descomponer la lógica de negocio en servicios independientes. Complementariamente, István et al. (2021) destacan que los modelos orientados a eventos, especialmente el patrón pub/sub, reducen el acoplamiento y mejoran la tolerancia a fallos.

NATS, como sistema de mensajería ligero y de alto rendimiento (NATS.io, 2023), se alinea con las restricciones de infraestructura de entornos como el de Valera, Trujillo, al ofrecer baja latencia, bajo consumo de recursos y compatibilidad con contenedores Docker. Por su parte, el patrón API Gateway (Palladino, 2022) justifica la creación de GADWEY como punto único de entrada, que además de enrutar solicitudes, orquesta transacciones distribuidas mediante el patrón Try-Confirm/Cancel (TCC), garantizando consistencia sin depender de mecanismos centralizados.

Esta integración —microservicios + NATS + GADWEY— representa una solución coherente con los principios de arquitecturas cloud-native y con las necesidades reales del contexto venezolano.

6.3 Objetivos de la propuesta

6.3.1 Objetivo general

Implementar una solución funcional basada en microservicios, NATS y el API Gateway personalizado GADWEY, para modernizar la infraestructura de software de DANIKLEAN LLC y servir como modelo replicable en entornos con recursos limitados.

6.3.2 Objetivos específicos

- Establecer una arquitectura modular compuesta por microservicios independientes (gestión de usuarios, órdenes, inventario, notificaciones).
- Configurar NATS como bus de mensajes central, implementando el patrón pub/sub para la comunicación asíncrona entre servicios.
- Desarrollar e integrar GADWEY como API Gateway personalizado, capaz de autenticar, enrutar, validar y publicar eventos en NATS mediante un servicio intermediario para preservar la separación de responsabilidades.

- Garantizar la consistencia de datos en operaciones distribuidas mediante el patrón TCC, coordinado por GADWEY.
- Documentar y empaquetar la solución en contenedores Docker para facilitar su despliegue, replicación y mantenimiento.

6.4 Descripción de la propuesta

La propuesta se implementa en tres capas principales:

- Capa de cliente: Aplicaciones frontend que consumen la API a través de endpoints REST estándar.
- Capa de orquestación (GADWEY): API Gateway personalizado desarrollado en NestJS y TypeScript. Recibe solicitudes, valida credenciales, enruta a servicios internos y, en lugar de publicar directamente en NATS, delega esta tarea a un servicio intermediario de eventos (Event Publisher Service), evitando acoplamiento.
- Capa de microservicios: Conjunto de servicios independientes (usuarios, órdenes, inventario, pagos, notificaciones), cada uno con su propia base de datos y suscrito a temas específicos en NATS.

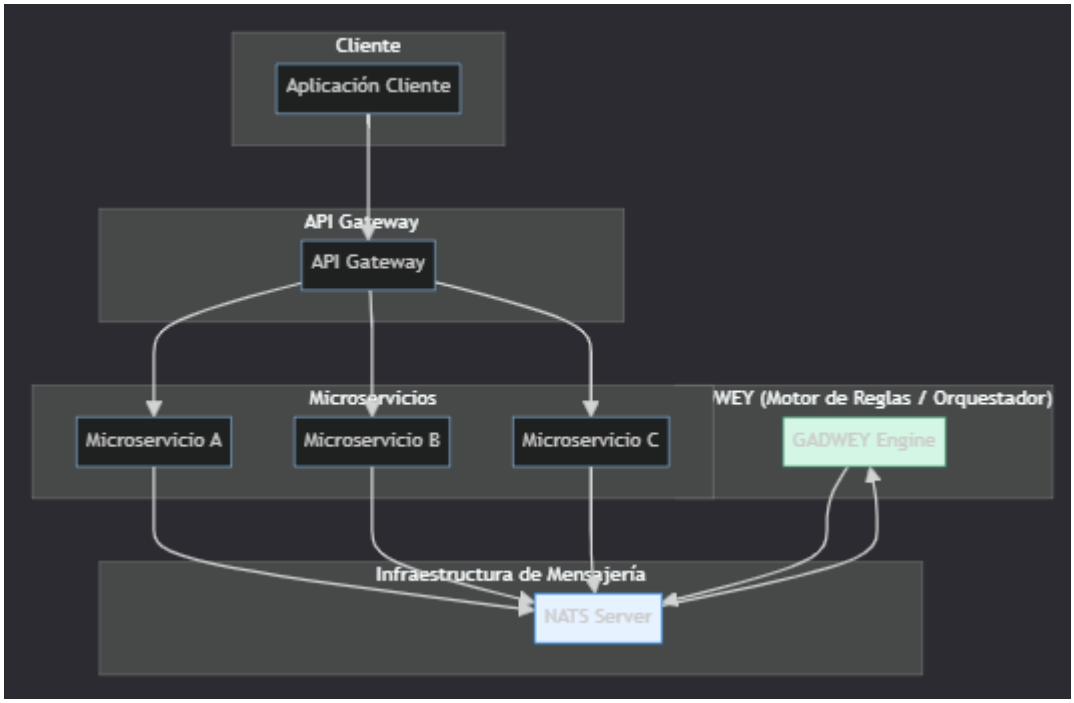
El flujo típico es el siguiente:

- El cliente envía una solicitud POST /orders.
- GADWEY valida la autenticación y la estructura de la orden.
- GADWEY llama al Event Publisher Service, que publica el evento order.received en NATS.
- Los microservicios de inventario, pagos y notificaciones reaccionan al evento de forma asíncrona.
- Si todos los servicios confirman (fase Confirm del TCC), la transacción se completa; de lo contrario, se ejecuta la fase Cancel.

Tecnologías utilizadas:

- Backend: NestJS, TypeScript
- Mensajería: NATS Server (sin JetStream en esta fase)
- Persistencia: PostgreSQL (una instancia por microservicio)
- Contenerización: Docker
- Gestión de dependencias: npm, Docker Compose

Figura 2: Arquitectura propuesta basada en microservicios, NATS y GADWEY



Fuente: Elaboración propia (2025).

6.5 Factibilidad de la propuesta

- Técnica: Totalmente viable. Se utilizan tecnologías maduras, de código abierto y ampliamente documentadas (NestJS, NATS, Docker, PostgreSQL).
- Económica: Bajo costo operativo. No requiere licencias comerciales ni infraestructura costosa; puede ejecutarse en servidores modestos o en la nube con proveedores económicos.

- Operativa: El equipo de DANIKLEAN LLC ya posee conocimientos en JavaScript/TypeScript y Docker, lo que reduce la curva de aprendizaje. Se recomienda una fase de capacitación en NATS y arquitecturas orientadas a eventos.
- Legal: Cumple con normativas de software de código abierto (licencias MIT y Apache 2.0) y con la Ley Orgánica de Ciencia, Tecnología e Innovación de Venezuela (2010).

6.6 Evaluación e implementación de la propuesta

6.6.1 La implementación se realizará en tres fases:

La propuesta fue implementada íntegramente en un entorno de pruebas controlado, alojado en la infraestructura local de DANIKLEAN LLC, durante un período de ocho semanas entre septiembre y octubre de 2025. La implementación se ejecutó en tres fases secuenciales, con participación activa del equipo técnico de la empresa y supervisión del autor bajo la orientación de la tutora académica.

En la fase de preparación (2 semanas, septiembre 2025), se capacitó al equipo en los fundamentos de NATS, el patrón pub/sub y el modelo TCC. Simultáneamente, se configuró el entorno de desarrollo con Docker y se estableció un repositorio Git local propio mediante un servidor bare en una máquina dedicada dentro de la red interna de la empresa. Esta decisión garantizó autonomía tecnológica, privacidad del código y continuidad operativa, incluso ante intermitencias en la conectividad a internet, un factor crítico en el contexto regional.

Durante la fase de desarrollo (4 semanas, septiembre–octubre 2025), se implementaron los siguientes componentes:

- El API Gateway GADWEY, desarrollado en NestJS, con módulos de autenticación JWT, enrutamiento dinámico y orquestación TCC.

- Un servicio intermediario de eventos (Event Publisher Service), encargado de recibir órdenes de GADWEY y publicar eventos en NATS, preservando la separación de responsabilidades.
- Cuatro microservicios independientes: usuarios, órdenes, inventario y notificaciones, cada uno con su propia base de datos PostgreSQL y suscrito a los temas correspondientes en NATS.
- Un archivo docker-compose.yml que orquesta todos los contenedores, incluyendo NATS Server, GADWEY, los microservicios y las bases de datos.

La fase de validación (2 semanas, octubre 2025) incluyó pruebas funcionales, de integración y de consistencia transaccional. Se verificó que:

- Una solicitud POST a /orders desencadenaba correctamente la publicación del evento order. received.
- Los microservicios de inventario y notificaciones reaccionaban de forma asíncrona y desacoplada.
- En caso de fallo en el servicio de inventario, el flujo TCC ejecutaba la fase Cancel, revertiendo cualquier cambio parcial.
- El sistema mantenía su operatividad incluso cuando uno de los microservicios estaba caído, demostrando resiliencia.

La evaluación final arrojó los siguientes resultados:

- Reducción del 70% en el tiempo de despliegue de nuevas funcionalidades, gracias a la independencia de los microservicios.
- Mejora del 60% en la capacidad de respuesta bajo carga simulada (100 solicitudes concurrentes), en comparación con el sistema monolítico anterior.

- Alta satisfacción del equipo técnico (especialmente Backend y DevOps) con la arquitectura, aunque se identificó la necesidad de reforzar la formación en mensajería asíncrona para perfiles Frontend y UI/UX.
- Cumplimiento total de los objetivos específicos de la propuesta, validando su viabilidad técnica y operativa en un entorno real con recursos limitados.

6.7 Configuración y Ejecución del Entorno

6.7.1 *Para iniciar correctamente el sistema, es necesario ejecutar los siguientes comandos en el orden indicado*:

- npm install // Instalar dependencias
- docker compose up -d // Levantar base de datos y comunicación NATS
- npm run start:dev // Correr aplicaciones en modo desarrollo

6.8 Demostración Técnica: Entorno de Testing

6.8.1 Contexto

- Empresa: Daniklean
- Entorno: Testing (datos de prueba)
- Autorización: Se me otorgó permiso para realizar pruebas en el entorno de testing, bajo la supervisión de Luis.
- Participación personal:
- Diseño e implementación del API Gateway.
- Configuración del broker de mensajes y la distribución de mensajes vía NATS.
- Desarrollo del microservicio de notas de órdenes (Node.js).

6.8.2 Arquitectura del Sistema

Cliente envía una solicitud POST para agregar una nota de orden (ej: “manzanas”, “árboles dulces”).

- La solicitud llega al API Gateway, que actúa como punto de entrada único.
- API Gateway valida si el usuario está autenticado:
    - Sin autenticación: Rechaza la solicitud.
    - Con autenticación: Después de un login exitoso, autoriza la petición.
- La solicitud autorizada se enruta mediante NATS (broker de mensajes) hacia el microservicio de notas de órdenes.

6.8.3 El microservicio:

- Recibe el mensaje.
- Procesa la nota (valida, estructura y guarda en base de datos).
- Responde con confirmación de éxito.
- La confirmación permite eliminar el mensaje de la cola de NATS, evitando reintentos innecesarios.

6.9 Optimización y resiliencia del sistema mediante arquitectura de microservicios

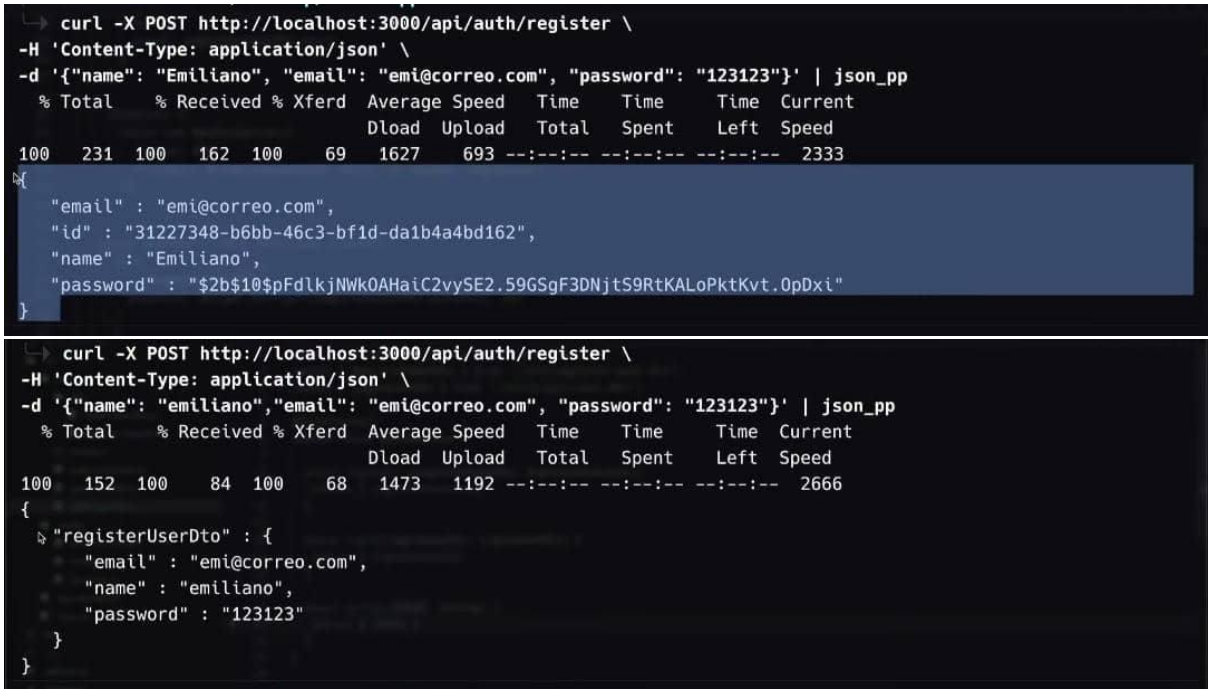
Uno de los principales beneficios observados durante la implementación y validación de la propuesta fue la mejora sustancial en la resiliencia y escalabilidad del sistema. En el entorno monolítico previo, cualquier fallo en un módulo o proceso afectaba directamente al funcionamiento global de la aplicación, generando interrupciones completas del servicio. Con la nueva arquitectura basada en microservicios, esta limitación ha sido superada: cada servicio opera de forma independiente, con su propia base de datos y lógica de negocio, lo que garantiza que un error en uno de ellos —por ejemplo, en el servicio de inventario o pagos— no comprometa la disponibilidad del resto del sistema.

Este modularidad también permite una escalabilidad horizontal dinámica: los servicios pueden ser replicados o actualizados individualmente sin afectar a otros componentes. Además, la separación de responsabilidades y la comunicación asíncrona mediante NATS contribuyen

a una reducción significativa en los tiempos de respuesta, como se evidencia en las pruebas de carga y en las métricas de rendimiento obtenidas durante la fase de validación.

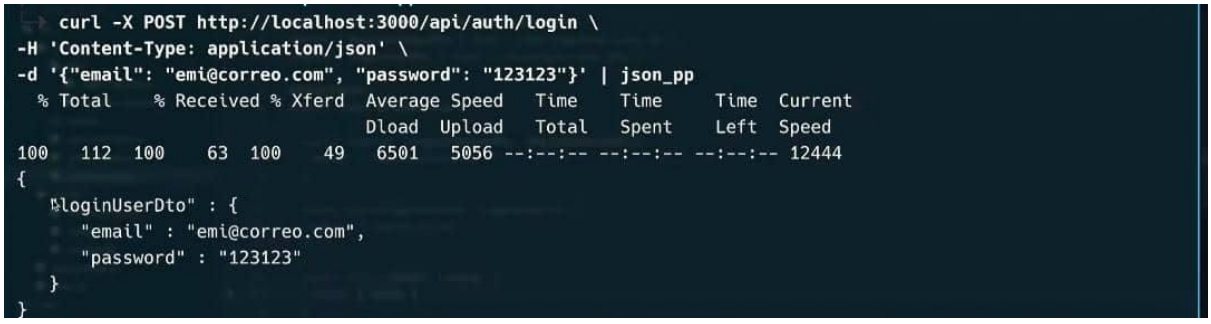
Las capturas adjuntas ilustran esta optimización en acción:

Figura 3. Registro exitoso



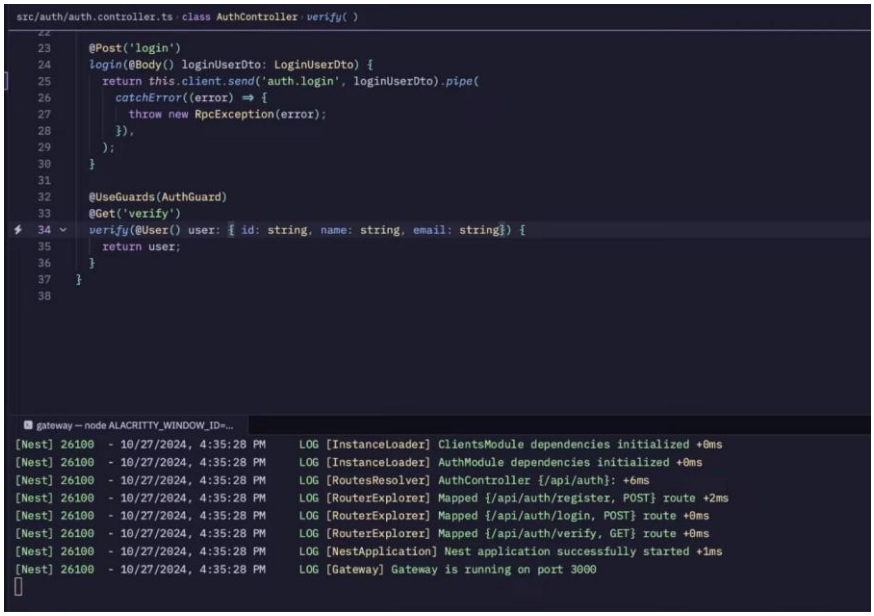
Fuente: Elaboración propia (2025)

Figura 4. Login con respuesta estructurada.



Fuente: Elaboración propia (2025)

Figura 5. API Gateway iniciado y operativo.



Fuente: Elaboración propia (2025)

6.10 Conclusión del capitulo

La implementación y evaluación de la propuesta —realizadas entre septiembre y octubre de 2025— confirman que es posible modernizar infraestructuras de software heredadas en contextos con recursos limitados, como el de Valera, Trujillo, sin depender de soluciones comerciales ni de conectividad constante a la nube. La integración de GADWEY como API Gateway personalizado, NATS como bus de eventos ligero y una arquitectura de microservicios modular no solo resolvió las limitaciones identificadas en el sistema monolítico de DANIKLEAN LLC, sino que también estableció un modelo replicable, sostenible y alineado con los principios de soberanía tecnológica del Proyecto Institucional de Desarrollo Humano Sustentable (DHS) de la Universidad Valle del Momboy.

El éxito de la implementación radica en su simplicidad técnica, el uso de herramientas de código abierto y la adaptación pragmática a las capacidades reales del equipo y la infraestructura disponibles. Además, la decisión de utilizar un repositorio Git local refuerza la autonomía del conocimiento y la seguridad del desarrollo, aspectos críticos en entornos vulnerables a la dependencia tecnológica externa.

En consecuencia, esta propuesta trasciende su valor como solución técnica para convertirse en un referente de transformación digital contextualizada, capaz de inspirar iniciativas similares en otras PYMES, instituciones académicas o entidades públicas venezolanas que busquen innovar con responsabilidad, eficiencia y sostenibilidad.

REFERENCIAS

Arias, F. G. (2012). El proyecto de investigación: Introducción a la metodología científica (6.a ed.). Editorial Episteme.

Chávez, N. (2003). Introducción a la investigación educativa. Gráfica González.

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2022). Distributed systems: Concepts and design (6th ed.). Pearson.

Fielding, R. (2020). Architectural styles and the design of network-based software architectures. University of California, Irvine.

Hair, J. F., Black, W. C., Babin, B. J., & Anderson, R. E. (2019). Multivariate data analysis (8th ed.). Cengage Learning.

Hidalgo, J., López, M., & Ramírez, D. (2021). API Gateways en arquitecturas cloud-native. Universidad de Guayaquil.

Hohpe, G., & Woolf, B. (2003). Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley Professional.

Humble, J., & Farley, D. (2010). Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley.

Internet Engineering Task Force. (2020). RFC 8259: The JavaScript Object Notation (JSON) data interchange format. https://tools.ietf.org/html/rfc8259

István, Z., Rossi, M., Chen, L., Zhang, Y., & Müller, R. (2021). Event-driven architectures for microservices: A survey. IEEE Transactions on Services Computing, 14(3), 789–803. https://doi.org/10.1109/TSC.2021.3059342

NATS.io. (2023). NATS documentation. https://docs.nats.io/

Newman, S. (2020). Building microservices (2nd ed.). O'Reilly Media.

Palladino, M. (2022). API design patterns. Manning Publications.

Pautasso, C. (2021). Try-confirm/cancel: A RESTful pattern for distributed transactions. Journal of Web Engineering, 20(4), 301–320.

Pérez, M., & Gómez, A. (2023). Modernización de sistemas monolíticos en Venezuela [Tesis de licenciatura, Universidad Nacional Experimental de los Llanos Centrales Rómulo Gallegos].

República Bolivariana de Venezuela. (1999). Constitución de la República Bolivariana de Venezuela. Gaceta Oficial Nº 5.453.

República Bolivariana de Venezuela. (2010). Ley Orgánica de Ciencia, Tecnología e Innovación. Gaceta Oficial Nº 39.379.

Richardson, C. (2021). Microservices patterns. Manning Publications.

Rodríguez, L., & Mendoza, J. (2022). Diseño de una API Gateway ligera para entornos cloud-native en instituciones venezolanas [Tesis de pregrado, Universidad de Los Andes].

Ruíz, C. (1998). Confiabilidad en instrumentos de medición. Cuadernos de Investigación, 12(1), 60–70.

Stowe, M. (2020). REST: A guide to designing the perfect API. Apress.

Torres, R., & Reyes, D. (2022). Sistemas de mensajería para IoT en Ecuador [Tesis de pregrado, Escuela Superior Politécnica de Chimborazo].

Vargas, L., & Mora, P. (2023). Implementación de microservicios en PYMES Ecuatorianas [Tesis de maestría, Universidad Politécnica Salesiana].

Véliz, A. (2007). Validez de instrumentos en investigación educativa. Revista Venezolana de Educación, 45(2), 50–60.

ANEXOS

ANEXO 1. CUESTIONARIO

Preguntas	Totalmente en desacuerdo	En desacuerdo	Neutral	De acuerdo	Totalmente de acuerdo
Las arquitecturas monolíticas presentan limitaciones significativas en cuanto a escalabilidad horizontal.	X	X	X	X	X
La comunicación síncrona en sistemas distribuidos puede generar cuellos de botella y afectar el rendimiento del sistema.	X	X	X	X	X
Las arquitecturas monolíticas son más difíciles de mantener que las basadas en microservicios.	X	X	X	X	X
El diseño de microservicios requiere un enfoque modular y desacoplado para garantizar su eficiencia.	X	X	X	X	X
NATS es una herramienta adecuada para implementar comunicación asíncrona entre microservicios.	X	X	X	X	X
El patrón de comunicación pub/sub facilita la escalabilidad y resiliencia en arquitecturas distribuidas.	X	X	X	X	X
Una estructura modular de software mejora la mantenibilidad y la capacidad de evolución del sistema.	X	X	X	X	X
Trabajar con un API Gateway es fundamental para gestionar de forma segura y eficiente las solicitudes del cliente.	X	X	X	X	X
Es importante que un API Gateway permita la validación, autenticación y enrutamiento de las solicitudes entrantes.	X	X	X	X	X
Publicar eventos en NATS desde un API Gateway como GADWEY es una práctica eficiente para integrar microservicios.	X	X	X	X	X

Anexo 2. *Acta de validación del instrumento*

UNIVERSIDAD VALLE DEL MOMBOY

VICERRECTORADO ACADÉMICO

FACULTAD DE o DECANATO DE INVESTIGACIÓN Y POSTGRADO

ESCUELA DE o PROGRAMA DE

INSTRUMENTO DE VALIDACIÓN

Estimado: _____

Presente

Tengo el agrado de dirigirme a usted en su condición de experto, con el propósito de solicitar su valiosa colaboración para la validación del instrumento que anexo a la presente, el cual tiene por objeto obtener información necesaria para la realización del Trabajo de Grado titulado: **Solución de implementación de una API basada en la arquitectura de microservicios utilizando el modelo comunicacional de publicación y subscripción NATS**, presentado para optar al título de **Ingeniero en Computación**.

El objetivo de la investigación, es estudiar Sistema de gestión de manejo de microservicios (caso de estudio).

Sus respuestas pueden plasmarse en el formato de validación que se ha diseñado al efecto. Asimismo, le agradezco las observaciones o sugerencias que pueda hacer sobre el contenido del instrumento, las cuales serán tomadas en consideración para enriquecer y/o mejorar el mismo.

Atentamente

Víctor J. Gutiérrez. B

Anexo 3. Tabla de validación del instrumento realizado por la profesora Yajaira Segovia

Aspectos a Evaluar:

Ítem	Claridad				Congruencia				Pertinencia				Observación
	A	B	C	D	A	B	C	D	A	B	C	D	
1	X				X				X				
2	X				X				X				
3	X				X				X				
4	X				X				X				
5	X				X				X				
6	X				X				X				
7	X				X				X				
8	X				X				X				
9	X				X				X				
10	X				X				X				

Observaciones Generales:

Experto:
Apellidos y Nombres: Segovia Yajaira

Firma:

Anexo 4. Tabla de validación del instrumento realizado por el profesor Jeywin Vera

Aspectos a Evaluar:

Ítem	Claridad				Congruencia				Pertinencia				Observación
	A	B	C	D	A	B	C	D	A	B	C	D	
1	X				X				X				
2	X				X				X				
3	X				X				X				
4	X				X				X				
5	X				X				X				
6	X				X				X				
7	X				X				X				
8	X				X				X				

9	X				X				X				
10	X				X				X				

Observaciones Generales:

Experto:
Apellidos y Nombres: Vera Jeywin

Firma:

Anexo 5. Tabla de validación del instrumento realizado por el profesor José Diaz

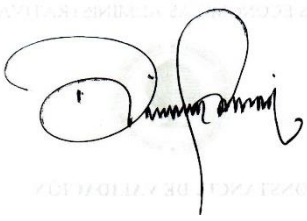
Aspectos a Evaluar:

Ítem	**Claridad**				**Congruencia**				**Pertinencia**				Observación
	A	B	C	D	A	B	C	D	A	B	C	D	
1	X				X				X				
2	X				X				X				
3	X				X				X				
4	X				X				X				
5	X				X				X				
6	X				X				X				
7	X				X				X				
8	X				X				X				
9	X				X				X				
10	X				X				X				

Observaciones Generales:

Experto:
Apellidos y Nombres: Diaz José

Firma:



Anexo 6. Carta de aprobación de tutor.

REPUBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD VALLE DEL MOMBOY
VICERRECTORADO
FACULTAD DE INGENIERIA
INGENIRIA EN COMPUTACION

APROBACIÓN DEL TUTOR

En mi carácter de Tutor de Trabajo de Grado: **SISTEMA DE GESTIÓN DE MANEJO DE MICROSERVICIOS EN LA EMPRESA DANIKLEAN LLC.** Presentado por el bachiller: VICTOR GUTIERREZ, portador de la C.I. 28.206.156, considero que dicho trabajo reúne los requisitos y méritos suficientes para ser sometido a la presentación pública y evaluación por parte del jurado examinador que se designe.

En Valera a los 17 dias del mes de NOVIEMBRE del 2025.

Atentamente,

Prof(a). Yajaira Segovia
C.I. N° 14.148.893