REPÚBLICA DE PANAMÁ

UNIVERSIDAD INTERNACIONAL DE CIENCIA Y TECNOLOGÍA

FACULTAD EN CIENCIAS DE LA COMPUTACIÓN Y TECNOLOGÍA

EVALUACIÓN DE LA EFECTIVIDAD DE POLÍTICAS ZERO TRUST APLICADAS A SERVICIOS RESTFUL EN ENTORNOS EMPRESARIALES

TRABAJO DE GRADUACIÓN PARA OPTAR POR EL TÍTULO DE MAESTRÍA EN INGENIERÍA DE SOFTWARE CON ÉNFASIS EN SEGURIDAD DE SERVICIOS Y APLICACIONES

Asesor: José Rivera

Estudiantes: Cristian Taju

Angelica Pitti

Ciudad de Panamá, agosto de 2025

REPÚBLICA DE PANAMÁ

UNIVERSIDAD INTERNACIONAL DE CIENCIA Y TECNOLOGÍA

FACULTAD EN CIENCIAS DE LA COMPUTACIÓN Y TECNOLOGÍA

EVALUACIÓN DE LA EFECTIVIDAD DE POLÍTICAS ZERO TRUST APLICADAS A SERVICIOS RESTFUL EN ENTORNOS EMPRESARIALES

TRABAJO DE GRADUACIÓN PARA OPTAR POR EL TÍTULO DE MAESTRÍA EN INGENIERÍA DE SOFTWARE CON ÉNFASIS EN SEGURIDAD DE SERVICIOS Y APLICACIONES

Estudiantes: Cristian Taju

Angelica Pitti

Ciudad de Panamá, agosto de 2025

Ciudad de Panamá, 5 de agosto de 2025

Profesor
Héctor Mazurkiewicz
Coordinador del Comité de Titulación de Estudios de Grado y Postgrado Presente.

En mi carácter de Tutor del Trabajo de Grado de Maestría, presentado por los estudiantes Angelica Ayeleth Pitti Pérez, cédula N.°4-808-83, y Cristian Alexis Taju Cabeza, cédula N.°8-953-401, para optar al grado de Maestría en Ingeniería de Software con énfasis en Seguridad de Servicios y Aplicaciones considero que el trabajo: reúne los requisitos y méritos suficientes para ser sometido a la presentación pública y evaluación por parte del Jurado examinador que se designe.

Atentamente,

Ing. José Rivera
Documento de identidad N.º 3-707-28

Línea de Investigación:
Seguridad de la información y ciberseguridad aplicada al desarrollo de software

UNIVERSIDAD INTERNACIONAL DE CIENCIA Y TECNOLOGÍA

FACULTAD EN CIENCIAS DE LA COMPUTACIÓN Y TECNOLOGÍA

INFORME DE ACTIVIDADES DE TUTORÍA OPCIÓN DE TITULACIÓN DE TRABAJO DE GRADO DE MAESTRÍA

Programa de Maestría: Maestría en Ingeniería de Software con énfasis en Seguridad de Servicios y Aplicaciones

Estudiante: Cristian Alexis Taju Cabeza

Cédula de identidad No. 8-953-401

Estudiante: Angelica Ayeleth Pitti Perez

Cédula de identidad No. 4-808-83

Tutor: Prof. Jose Rivera C.

Cédula de identidad No. 3-707-28

Título de trabajo de graduación: EVALUACIÓN DE LA EFECTIVIDAD DE POLÍTICAS ZERO TRUST APLICADAS A SERVICIOS RESTFUL EN ENTORNOS EMPRESARIALES

Línea de investigación: Seguridad de la información y ciberseguridad aplicada al desarrollo de software.

SESIÓN	FECHA	HORA REUNIÓN	ASPECTO TRATADO	OBSERVACIÓN
1	15-05-2025	8:00 pm	Escogencia de tutor	Se acordó la orientación en el proyecto.
2	22-05-2025	7:30 pm	Presentación de opciones de temas	Se eligió el tema.

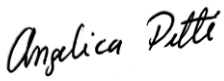
3	12-06-2025	11:40 AM	Planteamiento de problema, objetivos, justificación	Se definió el problema y se ajustaron objetivos.
4	15-07-2025	9:30 PM	Revisión y ajuste del capítulo I y 2	Se recomienda ajustar la extensión.
5	23-07-2025	8:00 pm	Revisión y ajuste del capítulo 3 y 4.	Se validó metodología y resultados.
6	03-08-2025	1:00 pm	Revisión final	Se aprobó la versión final preliminar.

Título definitivo: EVALUACIÓN DE LA EFECTIVIDAD DE POLÍTICAS ZERO TRUST APLICADAS A SERVICIOS RESTFUL EN ENTORNOS EMPRESARIALES

Comentarios finales acerca de la investigación: Declaramos que las especificaciones anteriores representan el proceso de dirección del trabajo de grado arriba mencionado.

Ing. José Rivera
Tutor

Cristian Taju
Estudiante



Angelica Pitti
Estudiante

DEDICATORIA

Con mucho cariño y todo mi amor principalmente a mi madre que me dio la vida y ha estado conmigo en todo momento. le agradezco mucho por todo el apoyo que me ha brindado y por confiar en mí. A mi familia, por su apoyo incondicional, sus palabras de aliento y su confianza en cada paso que di. A mi esposa, por ser mi apoyo constante y una gran inspiración para mí. Gracias por creer en mí incluso cuando yo dudaba. Este logro también es tuyo. A mis amigos, por acompañarme con risas y apoyo sincero en cada etapa de este camino. Gracias por estar, aún en la distancia.

- **Cristian T.**

Quiero dedicar principalmente este proyecto a Dios, por brindarme sabiduría, amor y paciencia. Por ayudarme en los momentos más difíciles de mi vida, brindándome la fortaleza para realizar y culminar esta etapa. A mi querida abuela que me enseñó a ser una mujer fuerte y a luchar por mis sueños contra viento y marea.

A mi novio Fermin, quien ha sido mi apoyo incondicional, por su tolerancia e infinita paciencia lo cual ha sido fundamental, no fue fácil, pero estuviste ahí motivándome, ayudándome hasta donde tus alcances lo permitían. Te lo agradezco muchísimo, amor.

A mi madre, a pesar de las diferencias que han surgido a lo largo del camino, reconozco que su existencia ha sido un impulso importante para no rendirme y superarme cada día. Esta meta es un reflejo de las raíces que me sostienen.

- Angelica P.

AGRADECIMIENTO

A Dios por brindarme la sabiduría y fuerza para culminar esta etapa académica. A mi compañera Angelica por su compromiso, dedicación y compañerismo a lo largo de este proceso. A mi asesor Ing. Jose Rivera C. Por su guía, paciencia y valiosos consejos a lo largo del proceso de investigación y desarrollo de esta tesis.

- **Cristian T.**

Quiero comenzar agradeciendo a Dios por su infinita bondad y por brindarme la fortaleza de seguir adelante y la oportunidad de crecer tanto profesional como personalmente. Agradezco profundamente a mi compañero Cristian por constante apoyo y por acompañarme en lo largo de esta carrera. Por su ejemplo de compromiso y compañerismo fue una inspiración para este proceso.
También extiendo mi agradecimiento a nuestro tutor Jose Rivera C. por su guía, enseñanzas y por su compromiso demostrado en cada etapa del proyecto. Su compañía fue valiosa y fundamental para alcanzar esta meta.

- **Angelica P.**

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

REPÚBLICA DE PANAMÁ UNIVERSIDAD INTERNACIONAL DE CIENCIA Y TECNOLOGÍA PROYECTO DE TRABAJO PARA OPTAR AL GRADO DE MAGÍSTER EN INGENIERÍA DE SOFTWARE CON ÉNFASIS EN SEGURIDAD DE SERVICIOS Y APLICACIONES

Autor (a): Cristian Taju

Autor (a): Angelica Pitti

Tutor (a): José Rivera

Año: 2025

RESUMEN

La presente tesis evalúa la efectividad del modelo de seguridad Zero Trust aplicado a servicios web basados en arquitecturas RESTful, con especial énfasis en pequeñas y medianas empresas (PyMEs). La complejidad creciente de las amenazas cibernéticas y la adopción de arquitecturas distribuidas han demostrado las limitaciones de la seguridad perimetral tradicional. En contraste, Zero Trust elimina la confianza implícita en usuarios, dispositivos y redes mediante autenticación continua, control de acceso granular y supervisión constante. Para validar su efectividad, se desarrolló un entorno simulado que reproduce condiciones reales de una PyME, comparando políticas tradicionales con Zero Trust en servicios RESTful. A partir de métricas de comportamiento, pruebas de acceso y simulación de amenazas, se analizó la capacidad de detección, la prevención de accesos no autorizados y el impacto en el rendimiento. Los resultados muestran una reducción del 85 % en vulnerabilidades críticas y bloqueos efectivos, con mínima afectación en la latencia. Se concluye que Zero Trust es un enfoque viable, escalable y eficaz para proteger servicios expuestos, y se proponen lineamientos para su adopción gradual en entornos empresariales locales.

Palabras clave: APIs, PyMEs, RESTful, seguridad informática, Zero Trust

REPUBLIC OF PANAMA INTERNATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY PROJECT WORK TO QUALIFY FOR THE DEGREE OF MASTER IN SOFTWARE ENGINEERING WITH AN EMPHASIS ON SERVICE AND APPLICATION SECURITY

Author (a): Cristian Taju

Author (a): Angelica Pitti

Tutor (a): José Rivera

Year: 2025

ABSTRACT

This thesis evaluates the effectiveness of the Zero Trust security model applied to RESTful web services, focusing on small and medium-sized enterprises (SMEs). The growing complexity of cyber threats and the adoption of distributed architectures have revealed the limitations of perimeter-based security. Zero Trust removes implicit trust in users, devices, and networks by enforcing continuous authentication, granular access control, and real-time monitoring. To validate its effectiveness, a simulated environment replicating SME conditions was created, comparing traditional security policies with Zero Trust implementation. Behavioral metrics, access control tests, and threat simulations were used to assess detection, prevention, and system performance. Results show that Zero Trust reduces critical vulnerabilities by 85% and blocks unauthorized access with minimal latency impact. The study concludes that this approach is a viable, scalable, and effective alternative for protecting exposed services and recommends its gradual integration in business environments.

Keywords: APIs, Cybersecurity, RESTful, Small and Medium Enterprises, Zero Trust

INTRODUCCIÓN

El presente estudio aborda la problemática de la seguridad en servicios web empresariales expuestos, en particular aquellos construidos sobre arquitecturas RESTful. Su adopción ha crecido de manera significativa en los últimos años gracias a la simplicidad, escalabilidad y compatibilidad con infraestructuras modernas. Sin embargo, esta misma exposición ha ampliado la superficie de ataque, lo que ha generado nuevas vulnerabilidades y ha puesto en evidencia las limitaciones de los enfoques tradicionales de seguridad, especialmente en entornos con recursos técnicos y financieros limitados, como suele ocurrir en pequeñas y medianas empresas (PyMEs).

En este escenario, el modelo de seguridad Zero Trust surge como una alternativa que rompe con el paradigma de confianza implícita en redes internas. Su filosofía se basa en la verificación continua de cada solicitud, sin importar su origen, así como en la implementación de controles de acceso dinámicos, segmentación lógica y supervisión constante. Aunque este modelo ha sido promovido en entornos corporativos de gran escala, su viabilidad técnica y operativa en organizaciones más pequeñas con servicios RESTful expuestos sigue siendo un campo abierto a la investigación.

Este proyecto se centra en evaluar empíricamente la efectividad del modelo Zero Trust aplicado a APIs RESTful en el contexto de una PyME. Para ello, se diseñó un entorno simulado en el que se implementaron servicios RESTful con configuraciones de seguridad tradicionales y, posteriormente, con controles basados en los principios de Zero Trust. La investigación incluyó la ejecución de pruebas automatizadas mediante la herramienta Postman, la cual permitió simular múltiples solicitudes concurrentes, evaluar el comportamiento del sistema en distintos escenarios de acceso y recopilar métricas clave.

Uno de los aspectos más relevantes del análisis fue la evaluación de la latencia, entendida como el tiempo de respuesta del sistema ante solicitudes legítimas bajo distintos esquemas de seguridad. Este enfoque busca determinar si la adopción de Zero Trust introduce penalizaciones significativas en el rendimiento o si, por el contrario, fortalece la seguridad sin comprometer la experiencia del usuario ni la eficiencia del servicio.

La incorporación de Postman como herramienta de validación permitió no solo medir la latencia, sino también detectar posibles errores de autorización, identificar patrones de

comportamiento ante cargas simultáneas y generar un historial trazable del funcionamiento de los endpoints. Esta fase se complementó con el análisis de registros del servidor, lo que brindó una visión más amplia sobre el impacto de la política de seguridad implementada.

La motivación principal de este estudio es proporcionar evidencia técnica y contextual sobre la implementación del modelo Zero Trust en entornos RESTful reales. Se busca considerar tanto los desafíos de seguridad como los aspectos operativos, respondiendo si este modelo es aplicable, efectivo y sostenible dentro de las limitaciones típicas de una PyME, y qué beneficios concretos ofrece frente a enfoques de seguridad convencionales.

Finalmente, este proyecto se enmarca en el área de la ingeniería de software y pretende aportar no solo un marco conceptual sólido, sino también resultados prácticos y recomendaciones para la adopción progresiva de políticas Zero Trust en organizaciones locales.

CAPÍTULO I. PLANTEAMIENTO Y FORMULACIÓN DEL PROBLEMA

1.1 Planteamiento del problema

En la actualidad, las pequeñas y medianas empresas (PyMEs) del sector tecnológico enfrentan una creciente presión por mantener sus aplicaciones seguras, especialmente cuando dependen de servicios RESTful expuestos al público. Sin embargo, muchas de estas organizaciones carecen de infraestructura sólida, personal capacitado o recursos suficientes para implementar controles de seguridad avanzados, lo que las deja vulnerables ante accesos no autorizados, filtraciones de datos y ataques de intermediarios.

El modelo Zero Trust, propuesto originalmente por John Kindervag en Forrester Research, plantea que ningún usuario o dispositivo debe ser automáticamente confiable, aun si está dentro de la red corporativa (Kindervag, 2010). Esta filosofía ha sido adoptada por grandes proveedores como Google y Microsoft; sin embargo, su aplicación en PyMEs presenta desafíos importantes debido a la complejidad técnica y al temor de afectar el rendimiento de las aplicaciones.

Estudios recientes muestran que muchas APIs RESTful carecen de autenticación fuerte, segmentación lógica y monitoreo continuo, lo que facilita ataques de enumeración, suplantación de identidad y abuso de endpoints (OWASP, 2023). Ante este escenario, surge la necesidad de evaluar cómo las políticas Zero Trust pueden aplicarse de forma escalable en estos contextos sin comprometer la operatividad del servicio.

De acuerdo con el informe *State of API Security Report – Q1 2023* elaborado por Salt Security, el 94 % de las organizaciones reportaron problemas de seguridad en APIs en producción durante el último año, y el 17 % sufrió una brecha relacionada con APIs. Además, el 78 % de los ataques provino de actores que aparentaban ser usuarios legítimos (Salt Security, 2023).

El reto principal radica en implementar estos controles sin afectar negativamente el rendimiento de los sistemas, especialmente en entornos donde cada milisegundo de latencia puede impactar la experiencia del usuario o donde no existen equipos dedicados exclusivamente a ciberseguridad. Este proyecto propone una evaluación técnica del impacto que tiene la aplicación de políticas Zero Trust sobre el rendimiento y la seguridad de servicios

RESTful, con el fin de identificar un punto de equilibrio viable para empresas que dependen de estas tecnologías.

En los últimos años, el uso de APIs RESTful ha aumentado significativamente en el entorno tecnológico, permitiendo la integración rápida y eficiente entre diferentes servicios y aplicaciones. No obstante, esta mayor exposición también ha incrementado los ataques de ciberseguridad dirigidos a dichas interfaces. Las vulnerabilidades asociadas a servicios RESTful representan un alto porcentaje en el registro de incidentes de seguridad, lo que resalta la urgencia de implementar controles más robustos y eficaces.

A pesar del mayor conocimiento sobre las amenazas cibernéticas, muchas PyMEs carecen de equipos especializados en ciberseguridad o de personal altamente capacitado para aplicar prácticas avanzadas de protección. Esto ha generado una brecha crítica en la seguridad de sus aplicaciones y servicios expuestos al público.

La protección de las PyMEs no solo es fundamental desde el punto de vista técnico, sino también desde el económico. Los ataques cibernéticos generan pérdidas financieras significativas y afectan la reputación de las empresas. Este estudio busca cubrir un vacío en la investigación sobre la aplicación de políticas Zero Trust en servicios RESTful, aportando tanto a la academia como a la mejora de la seguridad en pequeñas y medianas empresas.

Estudios previos demuestran que grandes organizaciones como Microsoft y Google han logrado reducir significativamente los riesgos de accesos no autorizados y filtraciones de datos mediante la implementación de Zero Trust. Sin embargo, la adaptación de este modelo a entornos de PyMEs con recursos limitados sigue siendo poco explorada, en especial en lo relacionado con su impacto en el rendimiento de los servicios RESTful.

1.2 Objetivo general

Analizar el impacto de la implementación de políticas de seguridad basadas en el modelo Zero Trust sobre una API RESTful funcional, con el propósito de determinar si dicho enfoque fortalece la protección sin comprometer el rendimiento en aplicaciones desarrolladas por pequeñas y medianas empresas tecnológicas.

1.3 Objetivos específicos

- Identificar las principales prácticas de seguridad Zero Trust aplicables a servicios RESTful.
- Proponer una API REST sencilla y funcional para realizar pruebas controladas.
- Implementar políticas de Zero Trust sobre la API y medir su efecto en métricas como latencia, errores y tiempos de respuesta.
- Comparar el comportamiento de la API con y sin políticas de Zero Trust para evaluar el impacto.
- Comparar los resultados obtenidos y redactar recomendaciones orientadas a equipos de desarrollo en PyMEs que deseen adoptar prácticas Zero Trust sin necesidad de inversiones costosas.

1.4 Formulación del problema

¿De qué manera puede implementarse el modelo de seguridad Zero Trust en APIs RESTful utilizadas por pequeñas y medianas empresas tecnológicas, garantizando una mejora en la protección sin comprometer el rendimiento, la experiencia del usuario ni la simplicidad operativa del sistema?

1.5 Justificación

La mayoría de las PyMEs tecnológicas dependen de APIs para ofrecer funcionalidades clave, como el inicio de sesión de usuarios, la gestión de inventarios, los procesos internos y el procesamiento de pedidos. Sin embargo, con frecuencia estos endpoints son expuestos sin controles adecuados de acceso, verificación o trazabilidad.

El modelo **Zero Trust** ha demostrado ser efectivo en entornos corporativos para reducir estos riesgos, aunque persisten dudas sobre su impacto en proyectos de menor escala, tanto desde la perspectiva técnica como organizacional. Implementar controles de seguridad más estrictos suele generar preocupación en los desarrolladores, principalmente por posibles efectos en la velocidad de respuesta, la experiencia del usuario o la dificultad de mantenimiento.

Este proyecto busca aportar evidencia práctica sobre la viabilidad de Zero Trust en entornos simples pero funcionales, mostrando en qué medida mejora la seguridad y cómo se ve afectado el rendimiento real del sistema.

Desde el **punto de vista académico**, el estudio permitirá aplicar conocimientos adquiridos durante la maestría en materia de arquitectura segura, control de accesos, pruebas de rendimiento y diseño de servicios. Desde la **perspectiva práctica**, los resultados podrán servir de referencia para profesionales que necesiten fortalecer sus APIs sin contar con grandes presupuestos ni personal especializado.

Aunque el modelo Zero Trust ha sido validado en entornos empresariales complejos, su adopción en organizaciones más pequeñas sigue siendo limitada. Existe una brecha entre la teoría y la práctica, especialmente en la forma de implementar estas políticas sin afectar el rendimiento o sin requerir inversiones que superen la capacidad de las PyMEs. En esta línea, NIST (2020) señala que "las organizaciones deben adaptarse a nuevas realidades de seguridad, en donde la confianza implícita ya no es suficiente, especialmente cuando se manejan datos sensibles en la nube o a través de APIs" (p. 7).

Los servicios RESTful, ampliamente utilizados en infraestructuras distribuidas, resultan altamente vulnerables, ya que sus APIs expuestas pueden ser explotadas por atacantes para acceder a información sensible y comprometer sistemas internos. Por esta razón, la aplicación de políticas Zero Trust es fundamental, no solo en entornos tradicionales, sino también en empresas que utilizan arquitecturas de microservicios y aplicaciones en la nube, donde los controles tradicionales resultan insuficientes.

Entre las ventajas de Zero Trust se encuentra su enfoque en la autenticación y autorización periódica, lo que lo convierte en una solución escalable para empresas en proceso de crecimiento. Asimismo, la adecuada protección de la información personal y confidencial es esencial para evitar problemas legales y mantener la confianza de los clientes.

Muchas organizaciones presentan problemas de seguridad debido a la falta de un enfoque centralizado y riguroso como el que ofrece Zero Trust. En consecuencia, los ciberataques generan costos financieros cada vez más altos, afectando directamente la continuidad operativa y la reputación corporativa.

Este proyecto busca evaluar la efectividad de Zero Trust en servicios RESTful y proponer enfoques de seguridad que fortalezcan la unificación y comunicación entre los sistemas informáticos. El modelo se presenta como una alternativa ideal para garantizar la protección de las organizaciones mediante normativas globales y un enfoque de seguridad más robusto.

1.6 Alcance del proyecto

- El diseño y desarrollo de una API RESTful funcional con operaciones básicas.
- La implementación de controles de seguridad basados en el modelo Zero Trust (como autenticación fuerte, control de acceso por roles y monitoreo).
- La evaluación de dicha API antes y después de aplicar las políticas Zero Trust.
- El uso de herramientas como Postman para recolectar métricas de rendimiento (latencia, tiempo de respuesta, tasa de errores).
- La elaboración de recomendaciones prácticas basadas en los resultados obtenidos.
- El enfoque en contextos de pequeñas y medianas empresas tecnológicas que trabajan con APIs.
- Evaluar si las políticas de Zero Trust influyen en las integraciones de APIs con servicios y sistemas de terceros (estos son servicios de autenticación externa y mensajería y bases de datos).

1.7 Delimitación del proyecto

Este estudio está delimitado en términos temporales, técnicos, espaciales y metodológicos, con el objetivo de acotar claramente el alcance de la investigación y garantizar su viabilidad dentro de los recursos disponibles.

Desde el punto de vista temporal, la ejecución del proyecto se llevará a cabo en un periodo de dos semanas, comprendidas entre el 1 y el 15 de agosto de 2025. Durante este tiempo se desarrollará la API RESTful, se aplicarán las políticas de seguridad Zero Trust, se ejecutarán las pruebas de rendimiento y seguridad, y se analizarán los resultados obtenidos.

En cuanto al alcance técnico, el sistema a desarrollar será una API RESTful simulada, construida en Laravel 12 sobre PHP 8.2 y utilizando una base de datos relacional MySQL 8.0. Esta API replicará operaciones fundamentales de un sistema empresarial orientado a la gestión de productos y pedidos, propias de un entorno de PyME tecnológica. La arquitectura del backend incluirá rutas protegidas mediante autenticación y autorización, y se diseñará con modularidad para facilitar el análisis antes y después de aplicar las políticas de seguridad.

Entre las funcionalidades principales se incluyen operaciones típicas en una aplicación empresarial sencilla. Cada endpoint ha sido diseñado para representar un escenario realista de

acceso a recursos protegidos, permitiendo así evaluar el impacto de aplicar políticas de seguridad Zero Trust.

POST /api/login: Permite a los usuarios autenticarse mediante el envío de sus credenciales (correo electrónico y contraseña). Si los datos son válidos, el sistema devuelve un JSON Web Token (JWT) firmado, generado con Passport Laravel. Este endpoint representa el primer filtro de acceso y debe contar con validaciones estrictas, límites de intentos fallidos y, eventualmente, autenticación multifactor (MFA).

GET /api/productos: Permite al usuario autenticado recuperar el catálogo de productos disponibles. La petición debe incluir un encabezado Authorization con el token JWT emitido. El sistema valida la vigencia del token antes de otorgar acceso. Este recurso ejemplifica un endpoint de lectura protegido con controles de acceso mínimos basados en rol.

POST /api/ordenes: Permite registrar nuevas órdenes de compra. La petición debe enviarse en formato JSON con los datos del producto, la cantidad y la dirección de envío. Este endpoint representa un flujo de negocio sensible, por lo que estará sujeto a controles más estrictos como validación de rol (usuario, admin), análisis contextual (actividad inusual) y monitoreo continuo.

GET /api/usuario/perfil: Devuelve los datos del usuario autenticado (nombre, correo electrónico, rol y fecha de creación de la cuenta). Este endpoint maneja información personal, por lo que se exige protección adicional. El sistema debe garantizar que cada usuario solo acceda a su propia información, evitando ataques de enumeración o exposición de datos de terceros.

Desde el punto de vista estructural, la API se apoyará en una base de datos relacional compuesta por las siguientes entidades clave:

- Usuarios: contendrá los datos básicos de autenticación y perfil del usuario.
- Productos: almacena el catálogo de productos con sus respectivos precios y descripciones.

- Órdenes: registra los pedidos realizados por los usuarios, vinculando producto, cantidad y fecha.

- Roles: define los niveles de acceso del sistema, permitiendo implementar el modelo RBAC (Role-Based Access Control).

Tabla 1. Endpoints de la API y controles de seguridad aplicados

Endpoint	Método HTTP	Parámetros	Descripción funcional	Controles Zero Trust aplicados
/api/login	POST	email, password	Autentica al usuario y genera un JWT válido	Validación de credenciales, protección contra fuerza bruta, MFA opcional.
/api/productos	GET	Authorization: Bearer <JWT>	Lista los productos disponibles en el sistema.	Validación de token, autorización por rol (solo usuarios autenticados)
/api/ordenes	POST	Authorization: Bearer <JWT>, id_producto, cantidad, dirección	Registra una orden de compra en el sistema.	Validación de token, control de acceso por rol, trazabilidad de actividad.
/api/usuario/perfil	GET	Authorization: Bearer <JWT>	Devuelve los datos del usuario autenticado.	Validación de token, validación de identidad/autorización estricta.

Fuente: Elaboración propia (2025)

La tabla detalla los principales endpoints del sistema junto con los controles de seguridad implementados bajo el modelo Zero Trust. Cada endpoints será protegido mediante un middleware que validará el token JWT, comprobará los permisos asociados al rol del usuario y generará registros de acceso para auditoría. Este conjunto de medidas permitirá evaluar la efectividad de los principios Zero Trust, como la autenticación fuerte, el acceso mínimo necesario y el monitoreo continuo.

Además, cada una de estas rutas se ejecutará en un entorno controlado, donde se medirán métricas como latencia, tiempo de respuesta, tasa de errores y número de accesos rechazados. Esto permitirá comparar objetivamente el desempeño del sistema antes y después de implementar las políticas de seguridad.

El entorno de pruebas será ejecutado de manera local (localhost), en una infraestructura de desarrollo controlada por el investigador. No se establecerá conectividad con servicios en la nube ni con bases de datos externas, con el objetivo de minimizar la interferencia de variables externas y garantizar la reproducibilidad total del experimento. Esta decisión responde a la necesidad metodológica de mantener un entorno técnico estable, predecible y aislado, permitiendo atribuir cualquier cambio en las métricas de rendimiento directamente a la aplicación de políticas Zero Trust y no a factores externos.

Aunque este enfoque fortalece la validez interna del estudio, también limita la generalización de los resultados hacia contextos de producción reales, donde pueden existir múltiples capas de red, conexiones simultáneas de usuarios, balanceo de carga y componentes distribuidos. No obstante, se asume que los hallazgos ofrecerán una base confiable para futuras extrapolaciones en condiciones similares.

Desde la perspectiva metodológica, esta investigación no contempla pruebas con usuarios finales, ya que su objetivo no es evaluar la experiencia del usuario ni la interfaz, sino analizar el comportamiento técnico del sistema bajo diferentes condiciones de seguridad. Por esta razón, no se aplicarán encuestas, entrevistas ni instrumentos cualitativos. Tampoco se incluirán análisis regulatorios como el RGPD (Reglamento General de Protección de Datos) o la Ley 81 de Protección de Datos Personales en Panamá.

Adicionalmente, se excluyen arquitecturas distribuidas complejas, como microservicios desacoplados, contenedores Docker, orquestadores como Kubernetes, balanceadores de carga, entornos multinube o servicios *serverless*. Su inclusión requeriría configuraciones avanzadas que desviarían el foco principal del estudio.

Finalmente, el sistema no integrará mecanismos de autenticación federada ni servicios de terceros como Google OAuth, Azure Active Directory, Auth0 u OpenID Connect. Todas las funciones de autenticación, emisión de tokens, verificación de identidad y control de acceso

serán desarrolladas de manera local e interna a la API RESTful, utilizando tokens JWT autogenerados y validados dentro del mismo entorno.

1.8 Línea de investigación

Esta investigación se inscribe dentro de la línea de seguridad informática aplicada al desarrollo de software, con un enfoque particular en la implementación de modelos de protección en arquitecturas RESTful. El objetivo es profundizar en estrategias emergentes que permitan fortalecer la ciberseguridad de servicios expuestos mediante APIs, especialmente en entornos con recursos limitados como las pequeñas y medianas empresas tecnológicas.

En concordancia con los principios de la ingeniería de software segura, esta línea integra conceptos de arquitectura de sistemas, programación segura, análisis de riesgos y pruebas de rendimiento, con el fin de generar soluciones prácticas, escalables y accesibles. Asimismo, se relaciona directamente con el desarrollo de políticas de control de acceso, autenticación robusta y segmentación de servicios, principios esenciales en el modelo Zero Trust, adoptado progresivamente como estándar internacional de seguridad (NIST, 2020).

Finalmente, esta línea de investigación contribuye a reducir la brecha entre teoría y práctica en materia de seguridad digital, promoviendo un enfoque técnico riguroso y a la vez contextualizado a las necesidades del entorno empresarial latinoamericano.

CAPÍTULO II. MARCO TEÓRICO

2.1 Marco conceptual

El proceso de transformación digital ha impulsado el crecimiento acelerado de aplicaciones web, muchas de ellas basadas en arquitecturas API RESTful. Este avance, aunque ha facilitado la interoperabilidad entre sistemas, también ha incrementado de manera significativa la superficie de ataque, exponiendo a las organizaciones a ciberamenazas cada vez más sofisticadas.

Ante este panorama, surge la necesidad de modelos de seguridad más adaptativos y robustos. Uno de los enfoques que ha cobrado mayor relevancia es el **modelo Zero Trust**, el cual replantea los principios tradicionales de confianza en redes corporativas. De acuerdo con el Instituto Nacional de Estándares y Tecnología (NIST, 2020), este modelo responde a la creciente complejidad de las infraestructuras digitales y a la evolución constante de las amenazas cibernéticas.

2.1.1 Modelo Zero Trust

El modelo Zero Trust fue propuesto por John Kindervag en 2010, durante su trabajo en *Forrester Research*. Su premisa fundamental, *"nunca confiar, siempre verificar"*, marcó una ruptura con los enfoques tradicionales de seguridad basados en perímetros, donde se asumía que todo lo que ocurría dentro de la red era confiable (Kindervag, 2010).

En este modelo se elimina la confianza implícita: cada solicitud de acceso, independientemente de su origen, debe ser autenticada, autorizada y evaluada de manera continua. De acuerdo con el Instituto Nacional de Estándares y Tecnología (NIST, 2020), este enfoque se apoya en controles de acceso dinámicos y contextuales, orientados a reducir el movimiento lateral dentro de sistemas comprometidos.

Entre sus principios fundamentales se destacan los siguientes:

- La autenticación multifactor (MFA),
- El control de acceso basado en roles o atributos (RBAC/ABAC),
- La segmentación de red para limitar el alcance de ataques internos, y
- El monitoreo continuo del comportamiento de usuarios y sistemas.

Zero Trust no corresponde a una solución tecnológica específica, sino que constituye una filosofía arquitectónica integral aplicable a redes, aplicaciones y datos. Para apoyar su implementación, la CISA (2021) desarrolló el modelo de madurez Zero Trust, que ofrece una guía progresiva para organizaciones de distintos tamaños. Esta propuesta se estructura en torno a cinco pilares: identidad, dispositivos, red, aplicaciones y datos, y plantea una adopción gradual en función de la capacidad organizacional.

En contextos empresariales complejos, la automatización y orquestación de políticas de acceso resultan fundamentales para mejorar la eficiencia operativa. La capacidad de definir y ajustar reglas en tiempo real, de acuerdo con el análisis de riesgos, permite escalar el modelo Zero Trust sin comprometer su eficacia.

En este documento se utilizarán indistintamente los términos Zero Trust (ZT) y confianza cero para referirse al mismo modelo de seguridad.

2.1.2 Arquitectura RESTful

El modelo de arquitectura REST (Representational State Transfer) fue formalizado por Roy Fielding (2000) en su tesis doctoral y, desde entonces, se ha convertido en un estándar ampliamente adoptado para el diseño de servicios web. REST se fundamenta en el uso de recursos accesibles mediante identificadores URI y manipulables a través de los métodos estándar del protocolo HTTP, como GET, POST, PUT y DELETE.

Su simplicidad, escalabilidad y compatibilidad con infraestructuras modernas lo han posicionado como el modelo dominante en el desarrollo de APIs web. No obstante, esta misma flexibilidad puede traducirse en debilidades desde el punto de vista de la seguridad si no se aplican mecanismos adecuados de validación y control.

REST, por sí solo, no incorpora métodos de autenticación ni control de acceso, lo que obliga a los desarrolladores a integrar soluciones externas como OAuth 2.0, JSON Web Tokens (JWT) o el uso de certificados TLS para cifrado de datos en tránsito (OWASP, 2023).

Cuando estas medidas no se implementan correctamente, se generan brechas que pueden exponer las APIs a riesgos como accesos indebidos, fuga de información o manipulación maliciosa. Este problema resulta especialmente crítico en pequeñas y medianas empresas, que

con frecuencia carecen de personal especializado o de recursos suficientes para aplicar controles avanzados de seguridad.

Por ello, se hace fundamental complementar la arquitectura RESTful con modelos de seguridad más robustos, como Zero Trust, que permiten establecer controles de acceso dinámicos, segmentar los servicios internos y aplicar monitoreo continuo en cada punto de interacción expuesto.

2.1.3 Convergencia entre Zero Trust y RESTful

La convergencia entre el modelo de seguridad Zero Trust y la arquitectura RESTful resulta fundamental, ya que las APIs web se han convertido en uno de los principales vectores de exposición en los servicios digitales actuales. En este contexto, aplicar los principios de Zero Trust implica establecer controles estrictos en cada punto de interacción con la API, eliminando la confianza implícita en cualquier cliente, dispositivo o red.

Esta integración permite reforzar la seguridad mediante la verificación continua de autenticación y autorización, considerando factores como la identidad, el comportamiento del usuario, el tipo de dispositivo y la ubicación.

Entre las prácticas recomendadas para aplicar Zero Trust en APIs RESTful destacan:

- La validación de tokens en cada solicitud,
- La revisión dinámica de permisos con base en el comportamiento reciente del usuario,
- La segmentación lógica de los recursos expuestos, y
- El monitoreo y registro en tiempo real de todas las interacciones con la API.

En entornos modernos desplegados en la nube, donde el perímetro tradicional se ha diluido, la implementación de **Zero Trust** en arquitecturas **RESTful** se vuelve aún más necesaria. Según el **NIST (2020)**, este enfoque facilita la evaluación constante del acceso a los recursos, independientemente de su ubicación, lo que contribuye a minimizar el riesgo de accesos no autorizados y de movimientos laterales dentro de la infraestructura.

Si bien este estudio se centra en arquitecturas RESTful, es importante reconocer que otros modelos de interfaz, como **GraphQL**, también presentan desafíos específicos que requieren medidas de seguridad adaptadas. La flexibilidad que ofrece GraphQL en la construcción de

consultas introduce vectores de ataque distintos, lo cual demanda la adaptación de los principios Zero Trust a este tipo de arquitectura.

2.2 Antecedentes históricos e investigativos

El modelo de seguridad Zero Trust ha ganado atención internacional como una respuesta eficaz al aumento constante de amenazas cibernéticas y a la adopción de arquitecturas distribuidas, especialmente aquellas basadas en microservicios. A medida que las organizaciones migran su infraestructura a la nube y adoptan tecnologías como contenedores y plataformas orquestadas, los enfoques tradicionales de seguridad perimetral se vuelven insuficientes (NIST, 2020).

En arquitecturas RESTful, la implementación de Zero Trust ha girado en torno a mecanismos como la autenticación fuerte, el control de acceso basado en políticas (ABAC) y la verificación continua del estado del usuario o servicio. Estas prácticas permiten un control granular sobre los accesos y pueden complementarse con técnicas como inteligencia artificial o aprendizaje automático, las cuales facilitan la detección temprana de comportamientos inusuales, tales como el secuestro de tokens, abuso de APIs o accesos internos maliciosos.

De acuerdo con el NIST (2020), Zero Trust parte del principio de que ningún actor, ya sea interno o externo a la red, debe considerarse confiable por defecto. Por lo tanto, cada solicitud de acceso debe evaluarse continuamente considerando la identidad, el comportamiento del usuario, el contexto y el nivel de riesgo. Este enfoque resulta especialmente útil en entornos compuestos por microservicios, donde los componentes interactúan de forma descentralizada y dinámica.

Además, la investigación empírica ha demostrado la utilidad de aplicar este modelo en servicios RESTful. Rezaei Nasab et al. (2021) identificaron buenas prácticas de seguridad en entornos de microservicios, tales como la segmentación de servicios, la autenticación contextual y el uso de gateways de seguridad para controlar el tráfico interno. Estas prácticas coinciden con los pilares fundamentales del enfoque Zero Trust y validan su aplicabilidad técnica en este tipo de arquitectura.

Con la virtualización creciente y el uso masivo de servicios en la nube, la dependencia de un perímetro físico ha dejado de ser un enfoque viable. Zero Trust permite aplicar medidas de

protección directamente en los puntos de acceso y comunicación, adaptándose mejor a la realidad operativa de las organizaciones distribuidas.

A nivel regional, la Comisión Económica para América Latina y el Caribe (CEPAL, 2022) indica que las pequeñas y medianas empresas (PyMEs) representan más del 90 % del tejido empresarial. Muchas de estas organizaciones carecen de personal técnico especializado y de políticas de seguridad formales, lo cual incrementa su exposición a ataques. Esta situación hace aún más pertinente la adopción de enfoques como Zero Trust, especialmente en entornos donde las APIs RESTful constituyen la interfaz principal de operación digital.

En el caso de Panamá, no se han identificado estudios académicos publicados que documenten la implementación del modelo Zero Trust en arquitecturas RESTful dentro de PyMEs o entornos empresariales locales. Este vacío justifica el presente estudio, que busca aportar evidencia práctica y contextualizada sobre la viabilidad y los beneficios de este modelo en servicios expuestos en el ecosistema panameño.

2.3 Fundamentación teórica

La base teórica de este estudio se estructura a partir de la interacción entre dos componentes clave: el modelo de seguridad Zero Trust y la arquitectura RESTful. Ambos elementos son ampliamente utilizados en entornos de desarrollo modernos y ofrecen un marco conceptual sólido para abordar los desafíos asociados a la exposición de servicios digitales, particularmente en organizaciones con estructuras distribuidas y limitaciones operativas como las PyMEs.

El modelo Zero Trust reformula la noción tradicional de confianza en las redes corporativas. En lugar de asumir que lo que se encuentra dentro del perímetro es seguro, este enfoque exige la verificación continua de cada solicitud de acceso, considerando la identidad, el comportamiento del usuario, el tipo de dispositivo y el contexto de uso. Entre sus principios fundamentales destacan el privilegio mínimo, la autenticación multifactor (MFA), la segmentación lógica y el monitoreo constante (NIST, 2020). Su propósito es reducir la superficie de ataque y mitigar el riesgo de accesos no autorizados, incluso desde dentro de la red.

Por su parte, la arquitectura RESTful formalizada por Fielding (2000), establece un estilo de diseño basado en recursos accesibles mediante URIs y manipulables a través de métodos HTTP estandarizados como GET, POST, PUT y DELETE. Esta arquitectura se caracteriza por su simplicidad, escalabilidad y bajo acoplamiento, lo que facilita su adopción en entornos empresariales. No obstante, RESTful no incluye mecanismos nativos de autenticación ni autorización, lo que obliga a los desarrolladores a integrar soluciones externas como OAuth 2.0, JWT y TLS para cifrado de datos en tránsito (OWASP, 2023).

Rezaei Nasab et al. (2021) identificaron que las arquitecturas basadas en microservicios y APIs, al igual que RESTful, presentan desafíos específicos de seguridad debido a su exposición y fragmentación. Estos entornos requieren estrategias más rigurosas para la gestión de accesos, validación de solicitudes y protección contra abusos.

En este sentido, la integración de Zero Trust sobre una arquitectura RESTful permite aplicar controles adaptativos y dinámicos, como la evaluación contextual de permisos y la segmentación lógica de recursos. Esta combinación resulta especialmente útil para PyMEs, que, si bien no siempre cuentan con infraestructura avanzada, sí pueden adoptar modelos escalables y eficaces como Zero Trust para proteger sus interfaces expuestas.

El presente estudio, por tanto, parte del supuesto teórico de que la aplicación de políticas Zero Trust en servicios RESTful representa una estrategia técnica y operativamente viable para reducir riesgos de seguridad en organizaciones que dependen de APIs para su operación digital.

2.4 Amenazas comunes en APIs RESTful

Las APIs RESTful, por su naturaleza abierta y ampliamente utilizada, representan un vector frecuente de exposición en los sistemas digitales modernos. De acuerdo con el informe OWASP API Security Top 10 (2023), estas interfaces presentan una serie de vulnerabilidades recurrentes que, si no se gestionan adecuadamente, pueden comprometer gravemente la confidencialidad, integridad y disponibilidad de los datos y servicios.

Una de las amenazas más comunes es la Broken Object Level Authorization (BOLA), que permite a un atacante acceder a recursos a los que no debería tener autorización, debido a controles de acceso mal implementados. Esta falla puede derivar en la exposición de información sensible o la alteración de datos de otros usuarios.

Otra vulnerabilidad crítica es la Excessive Data Exposure, que ocurre cuando los endpoints devuelven más información de la necesaria, incluyendo datos sensibles que no deberían ser accesibles para el cliente. Este problema se debe a respuestas mal diseñadas o a la falta de filtrado a nivel del backend.

Asimismo, los ataques de inyección como SQL Injection o Command Injection continúan siendo relevantes en APIs, especialmente cuando los parámetros de entrada no se validan correctamente. Estas fallas permiten ejecutar comandos maliciosos o manipular consultas a bases de datos, afectando tanto los datos como el comportamiento del sistema.

El Incorrecto Manejo de Activos (Improper Asset Management) también representa una amenaza significativa, ya que muchas organizaciones mantienen versiones antiguas de APIs, endpoints no documentados o entornos de prueba accesibles públicamente, lo que facilita el reconocimiento y la explotación por parte de atacantes.

Finalmente, la ausencia de limitación de tasas de solicitudes (Rate Limiting) expone a las APIs a ataques de fuerza bruta y a intentos de denegación de servicio (DoS), que pueden degradar o interrumpir la disponibilidad del sistema.

En el caso particular de las pequeñas y medianas empresas, estas amenazas pueden tener un impacto desproporcionado, ya que muchas veces no cuentan con herramientas especializadas de protección o personal dedicado exclusivamente a la seguridad. Medidas como la autenticación multifactor, la segmentación lógica de recursos y el monitoreo continuo son esenciales para mitigar estos riesgos.

Tal como advierte OWASP (2023), los fallos de autorización y la sobreexposición de datos son riesgos recurrentes que deben ser prioritariamente gestionados en cualquier entorno basado en APIs RESTful.

2.5 Área de relevancia: PyMEs tecnológicas

Las pequeñas y medianas empresas (PyMEs) representan un componente esencial en el desarrollo económico de América Latina. Según la Comisión Económica para América Latina y el Caribe (CEPAL, 2022), este sector constituye más del 90 % del tejido empresarial en la región. No obstante, también son las más expuestas a amenazas cibernéticas, debido a la

carencia de infraestructura tecnológica robusta, personal especializado y políticas de seguridad formalizadas.

En el contexto actual de digitalización acelerada, muchas PyMEs han adoptado servicios basados en APIs RESTful para facilitar operaciones, integraciones o servicios al cliente. Sin embargo, este avance no siempre va acompañado de una estrategia de protección adecuada, lo que aumenta la superficie de ataque y la vulnerabilidad ante accesos no autorizados, fuga de información o interrupciones en la disponibilidad del servicio.

El modelo de seguridad Zero Trust se presenta como una alternativa viable para este tipo de organizaciones. Su enfoque flexible, gradual y centrado en el control del acceso, permite adoptar buenas prácticas sin requerir una infraestructura física costosa. Elementos como la autenticación multifactor (MFA), el monitoreo continuo, la segmentación lógica de los recursos y la evaluación contextual del acceso pueden implementarse progresivamente, incluso con recursos limitados.

Además de mitigar riesgos técnicos, la implementación de políticas Zero Trust también puede contribuir al cumplimiento de normativas emergentes sobre protección de datos, fortalecer la confianza de los clientes y brindar ventajas competitivas a largo plazo. En este sentido, las PyMEs tecnológicas no solo se beneficiarían de una mayor seguridad operativa, sino que también estarían mejor posicionadas para escalar sus servicios digitales de manera sostenible.

2.6 Contexto tecnológico y profesional del estudio

Este estudio se desarrolla en el marco del ejercicio profesional del desarrollo de software y la arquitectura de sistemas, con énfasis en la creación, mantenimiento y mejora de servicios web que utilizan arquitecturas RESTful. El contexto refleja escenarios reales en los que desarrolladores, arquitectos de software, ingenieros de sistemas y profesionales independientes enfrentan desafíos vinculados a la exposición de servicios, la necesidad de interoperabilidad y la seguridad de los datos.

Actualmente, muchas organizaciones tecnológicas incluidas PyMEs dependen de APIs RESTful como componente central de su operación digital. Sin embargo, no todas cuentan con los conocimientos técnicos, recursos económicos o políticas internas necesarias para

implementar modelos de seguridad sólidos, lo que las deja vulnerables ante accesos no autorizados, filtraciones de datos o ataques dirigidos a su infraestructura.

Existe una demanda creciente en el sector tecnológico por soluciones de seguridad que sean técnicamente viables, escalables y aplicables sin requerir entornos complejos ni grandes inversiones. En este sentido, el modelo Zero Trust representa una estrategia adecuada para responder a estas necesidades, al permitir la implementación progresiva de mecanismos como autenticación fuerte, control de acceso basado en contexto y monitoreo en tiempo real.

Este estudio no solo se vincula con las prácticas profesionales del desarrollo seguro de software, sino que también busca aportar valor al ecosistema tecnológico mediante la evaluación de un modelo adaptable, aplicable en distintos niveles de madurez organizacional. En particular, se espera que los resultados contribuyan al fortalecimiento de la seguridad en APIs RESTful, sin comprometer el rendimiento ni aumentar la complejidad del desarrollo.

Asimismo, la integración de Zero Trust puede facilitar el cumplimiento de estándares y normativas internacionales, al establecer mecanismos de control, evaluación continua y protección reforzada de los datos sensibles. Esto puede traducirse en una mayor confianza por parte de los usuarios y en una ventaja competitiva en mercados donde la seguridad de la información es un factor diferenciador.

2.7 Modelos de seguridad comparados

Durante años, la seguridad perimetral fue considerada la base de la protección informática. Este modelo operaba bajo el supuesto de que todo lo que se encontraba dentro del perímetro de red, por ejemplo, detrás de un firewall corporativo era confiable. En consecuencia, los esfuerzos de defensa se concentraban en evitar intrusiones desde el exterior, mientras que el tráfico interno no era sometido a controles rigurosos.

Sin embargo, la masificación de los servicios en la nube, el trabajo remoto y la creciente sofisticación de los ataques internos han expuesto las limitaciones de este enfoque. El perímetro de red, tal como se concebía, ha dejado de existir o se ha vuelto demasiado difuso para ofrecer una protección efectiva por sí solo.

Frente a este panorama, el modelo de arquitectura Zero Trust propone un cambio fundamental: no se asume confianza en ningún usuario, dispositivo o red, sin importar su ubicación. Cada

solicitud de acceso debe ser verificada, autenticada y autorizada explícitamente. Este modelo responde a las dinámicas actuales, donde los sistemas están distribuidos, los dispositivos son múltiples y los vectores de ataque son diversos.

La implementación de Zero Trust se basa en principios como el acceso mínimo necesario (principio de privilegio mínimo), la autenticación multifactor (MFA), la microsegmentación y el monitoreo continuo del comportamiento del usuario y del sistema. Estas medidas permiten detectar accesos sospechosos en tiempo real y limitar el impacto de una posible brecha.

En este sentido, Zero Trust no reemplaza por completo a los modelos tradicionales, sino que los complementa. Su enfoque no se limita a un perímetro físico o lógico, sino que se adapta a entornos distribuidos, nativos en la nube y con alto grado de interconexión. Así, la elección entre modelos de seguridad no debe verse como una decisión excluyente, sino como una estrategia combinada, ajustada a las necesidades, capacidades y contexto de cada organización.

La evolución digital, el crecimiento de usuarios móviles y el aumento de dispositivos conectados han debilitado el concepto clásico de perímetro seguro. En este nuevo escenario, los entornos híbridos requieren soluciones que trascienden la ubicación geográfica y se enfoquen en la identidad, el contexto y el riesgo. En respuesta a esta realidad, Zero Trust se presenta como un enfoque innovador que promueve la protección distribuida, adaptable y centrada en el acceso verificado.

2.8 Términos clave del glosario

- Zero Trust: modelo de seguridad que elimina la confianza implícita dentro de las redes y establece la verificación continua de identidad, autorización contextual y monitoreo permanente en cada solicitud de acceso.

- API RESTful (Application Programming Interface – Representational State Transfer): interfaz de programación de aplicaciones que implementa los principios de la arquitectura REST, utilizando el protocolo HTTP para la manipulación de recursos identificados mediante URI.

- JSON Web Token (JWT): mecanismo de autenticación y autorización basado en tokens firmados digitalmente, utilizado para validar la identidad del usuario y autorizar el acceso a recursos protegidos.

- OAuth 2.0: protocolo de autorización que permite a aplicaciones obtener acceso limitado a recursos protegidos en nombre de un usuario, sin necesidad de exponer sus credenciales.

- Transport Layer Security (TLS): protocolo criptográfico diseñado para garantizar la confidencialidad, integridad y autenticación de la comunicación entre sistemas en redes distribuidas.

- Postman: herramienta de software utilizada para el diseño, prueba y validación de APIs, que permite la ejecución de solicitudes automatizadas y la recopilación de métricas de rendimiento y seguridad.

- OWASP (Open Web Application Security Project): comunidad internacional dedicada a mejorar la seguridad del software, reconocida por elaborar lineamientos y estándares como el OWASP Top 10 y el OWASP API Security Top 10.

- CISA (Cybersecurity and Infrastructure Security Agency): agencia del Departamento de Seguridad Nacional de los Estados Unidos que desarrolla lineamientos de ciberseguridad, entre ellos el modelo de madurez Zero Trust.

- NIST (National Institute of Standards and Technology): organismo del Departamento de Comercio de los Estados Unidos que emite estándares de seguridad reconocidos internacionalmente, como la publicación especial SP 800-207 sobre Zero Trust Architecture.

- CEPAL (Comisión Económica para América Latina y el Caribe): organismo de las Naciones Unidas encargado de promover el desarrollo económico y social en la región, que publica estudios relevantes sobre la situación de las PyMEs y su digitalización.

- Latencia: medida del tiempo transcurrido entre el envío de una solicitud y la recepción de la respuesta, utilizada como indicador del rendimiento de sistemas y servicios en red.

- Microservicios: estilo de arquitectura de software en el que una aplicación se estructura como un conjunto de servicios pequeños, autónomos y desplegables de manera independiente, que interactúan entre sí mediante APIs.

- Endpoint: punto final de comunicación en una API, representado por una URL específica que permite acceder a un recurso o funcionalidad del sistema.

- Autenticación multifactor (MFA): mecanismo de verificación de identidad que combina al menos dos factores distintos (conocimiento, posesión o biometría) para fortalecer la seguridad de los accesos.

- Control de acceso basado en roles (RBAC): modelo de autorización que asigna permisos a los usuarios en función de los roles definidos dentro de la organización.

- Control de acceso basado en atributos (ABAC): modelo de autorización que evalúa atributos de usuario, recurso y contexto para determinar las políticas de acceso.

- Gateway de API: componente de seguridad y gestión que actúa como punto de entrada para las solicitudes dirigidas a una API, encargado de aplicar autenticación, enrutamiento, monitoreo y control de tráfico.

- Balanceo de carga: técnica que distribuye automáticamente las solicitudes entrantes entre varios servidores o instancias de aplicación, con el fin de optimizar el uso de recursos y mejorar la disponibilidad del servicio.

- Servidor local (localhost): entorno de ejecución de aplicaciones configurado en la máquina del desarrollador, empleado para pruebas y desarrollo antes de la implementación en producción.

- API Key: credencial de autenticación en forma de cadena alfanumérica, utilizada para identificar al cliente que consume una API y autorizar su acceso a determinados recursos.

- Vulnerabilidad: debilidad o fallo en un sistema, aplicación o red que puede ser explotado por un atacante para comprometer la confidencialidad, integridad o disponibilidad de la información.

CAPÍTULO III. METODOLOGÍA

3.1 Enfoque de investigación

Este estudio adoptó un enfoque mixto que combinó métodos cuantitativos y cualitativos para evaluar el impacto del modelo Zero Trust en una API RESTful desarrollada en Laravel. El componente cuantitativo midió variables objetivas como la latencia, la tasa de errores, el número de accesos bloqueados y el consumo de recursos antes y después de aplicar controles Zero Trust. El componente cualitativo, con alcance acotado al análisis reflexivo del desarrollo, recopiló percepciones mediante bitácora de desarrollo y análisis reflexivo, siguiendo a Hernández, Fernández y Baptista (2014) y Rojas-Villalba (2021).

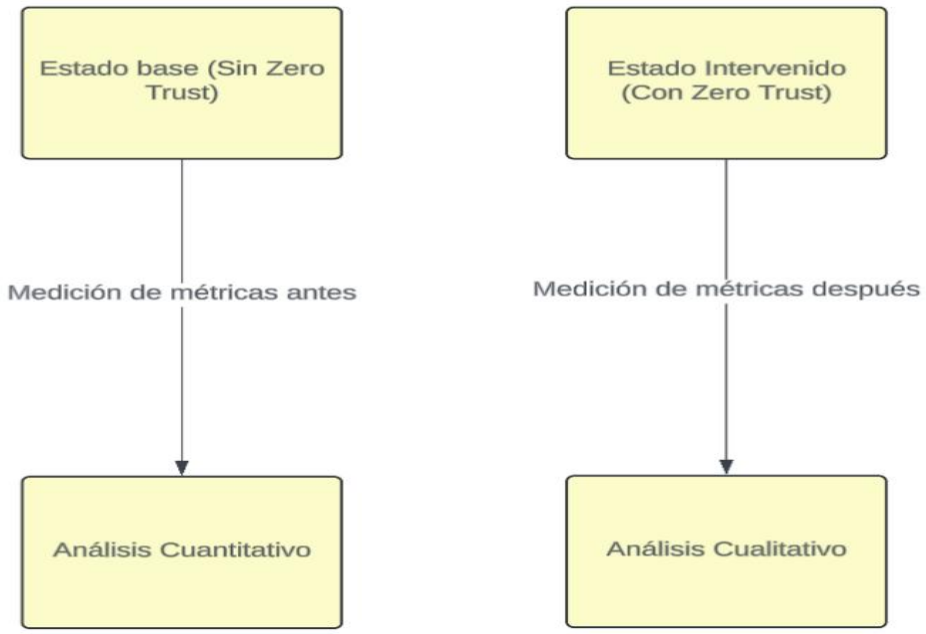
Adicionalmente, se adopta un diseño cuasiexperimental, en el que se implementa una intervención tecnológica (Zero Trust) sobre un sistema funcional preexistente (la API RESTful) dentro de un entorno controlado de pruebas. Esta aproximación permite observar los cambios provocados por la intervención sin manipular aleatoriamente las variables independientes, característica propia de los entornos tecnológicos donde se evalúan configuraciones de seguridad. Este enfoque mixto es coherente con estudios similares realizados en el área de seguridad informática y desarrollo de software en entornos empresariales. Por ejemplo, (Rojas-Villalba, 2021) en su tesis "Implementación de un modelo de confianza cero (Zero Trust) en entornos empresariales: un estudio de caso en una empresa tecnológica de Colombia", utilizó un diseño mixto para evaluar impacto técnico y percepciones organizacionales. De este modo, el enfoque mixto no solo permitirá evaluar con precisión los cambios objetivos producidos por la implementación del modelo Zero Trust, sino que también posibilita interpretar los efectos colaterales percibidos por los actores involucrados. Así, se garantiza una visión más completa del problema, alineada con los objetivos del estudio y con las necesidades reales de las PYMEs en contextos tecnológicos.

Para garantizar la validez de los resultados, se definió un total de 30 iteraciones por escenario de prueba. Este número se seleccionó porque permite obtener una muestra suficientemente representativa para calcular estadísticas básicas como la media y la desviación estándar, reduciendo la variabilidad de los resultados sin incurrir en tiempos excesivos de ejecución. Asimismo, se estableció como criterio de aceptación un umbral de latencia promedio menor a 500 milisegundos (ms) por transacción, valor que se considera adecuado en entornos de

pequeñas y medianas empresas (PyMEs), donde los servicios RESTful deben mantener tiempos de respuesta ágiles sin comprometer la seguridad. De esta manera, el análisis equilibra la necesidad de robustez estadística con la factibilidad práctica en escenarios reales de producción.

El número de 30 iteraciones se definió como un tamaño de muestra adecuado para reducir la variabilidad de los datos y garantizar la consistencia de las mediciones. Según Hernández, Fernández y Baptista (2014), la repetición suficiente de pruebas en estudios experimentales permite estimar estadísticos descriptivos confiables, lo que respalda la validez de los hallazgos obtenidos.

Figura 1. Implementación de Zero Trust en la API RESTful



Fuente: Elaboración propia (2025).

La figura ilustra la relación entre los componentes de la API y los controles Zero Trust implementados (autenticación, autorización y monitoreo).

3.2 Diseño metodológico

El diseño metodológico de esta investigación adopta un enfoque no experimental, transversal y explicativo, alineado con los objetivos planteados. Este tipo de diseño permite observar

fenómenos tal como se presentan en su contexto natural, sin manipular deliberadamente las variables, permitiendo establecer relaciones causales o correlacionales entre la implementación del modelo Zero Trust y sus efectos sobre APIs RESTful funcionales en pequeñas y medianas empresas tecnológicas.

De acuerdo con Hernández, Fernández y Baptista (2014), el diseño no experimental se basa en la observación de fenómenos existentes, sin intervención intencional del investigador sobre las variables, siendo adecuado cuando se desea analizar una situación existente en entornos reales, como es el caso de APIs operativas dentro de arquitecturas empresariales. En este contexto, se analizarán APIs RESTful previamente diseñadas y luego se aplicó el modelo de seguridad Zero Trust para evaluar sus efectos sobre la protección y el rendimiento del sistema.

Asimismo, la investigación es transversal porque la recolección de datos se realizó en un único momento en el tiempo, lo que permitirá evaluar de forma puntual el impacto del modelo Zero Trust sobre las variables estudiadas. Este enfoque es pertinente cuando se requiere un análisis de estado actual o de corto plazo, y no una evolución histórica o longitudinal del fenómeno (Sampieri et al., 2022).

El propósito explicativo del diseño metodológico se justifica en tanto se busca comprender y evidenciar el grado en que la implementación de un marco de seguridad Zero Trust fortalece la protección sin afectar negativamente el rendimiento, la experiencia del usuario o la simplicidad operativa del sistema. A través de la experimentación controlada con una API funcional, se espera identificar patrones de comportamiento, fortalezas, limitaciones y posibles áreas de mejora derivadas del modelo de seguridad adoptado.

Además, se propone una estrategia de investigación aplicada, orientada a la resolución de problemas prácticos en el entorno de desarrollo tecnológico de las pymes. El diseño metodológico estará respaldado por la implementación de una prueba controlada sobre una API desarrollada ad hoc, la cual contará con endpoints básicos que simulan operaciones comunes en entornos empresariales (autenticación, acceso a datos sensibles, validaciones, etc.).

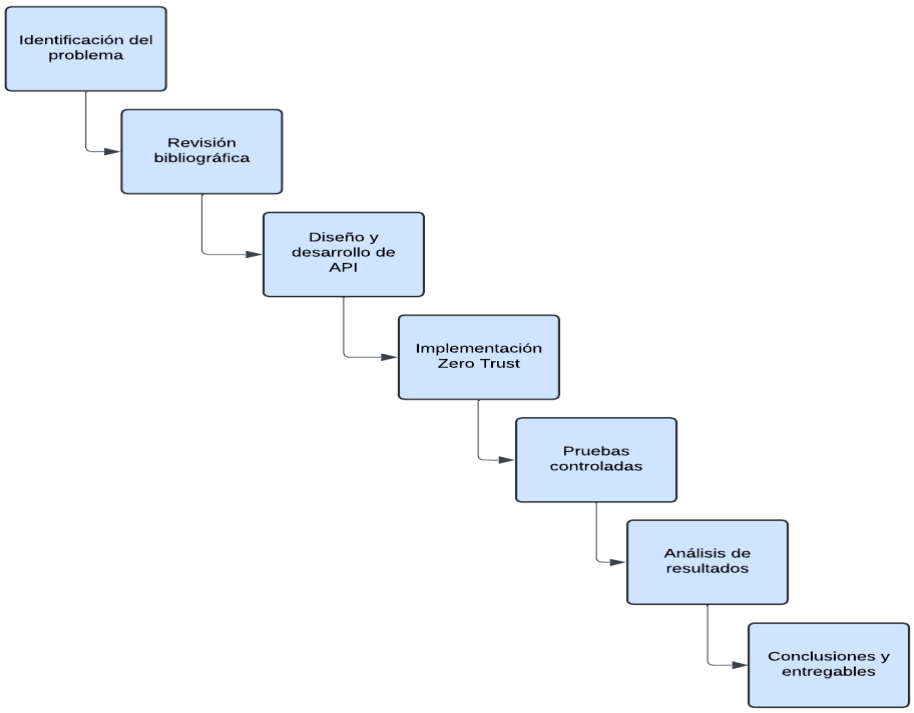
Se desarrolla una API monolítica en Laravel 12 con endpoints básicos, se aplica la intervención Zero Trust y se recopilan métricas en un solo periodo de prueba (15 jul – 15 ago 2025). Las mediciones incluyen:

- Latencia promedio (ms)

- Tasa de error HTTP (4xx/5xx)
- Número de accesos bloqueados
- Tiempo de respuesta bajo carga (Postman Collection Runner)

Se excluyen microservicios, Kubernetes y servicios externos de autenticación.

Figura 2. Etapas de la evaluación del modelo Zero Trust aplicado a servicios RESTful



Fuente: Elaboración propia (2025).

De esta manera, el diseño metodológico permite establecer un marco comparativo objetivo y verificable, que vincula los principios del modelo Zero Trust con su aplicabilidad práctica sobre APIs RESTful en pymes tecnológicas, aportando tanto a la teoría como a la práctica de la seguridad informática moderna.

3.2.1 Justificación del Diseño

La selección del framework Laravel 12 responde tanto a su popularidad en la industria como a sus facilidades para implementar APIs modernas, a través de características como Eloquent ORM, controladores RESTful, middleware y autenticación con tokens. Laravel no solo destaca

por su estructura modular y limpieza sintáctica, sino también por su compatibilidad con pruebas automatizadas y su enfoque en el desarrollo seguro desde la fase inicial. Su sistema de middlewares permite definir reglas de acceso y validaciones de forma granular, lo que resulta útil para aplicar principios de microsegmentación y control de identidad dinámica, propios del enfoque Zero Trust. Asimismo, al integrar de forma nativa mecanismos de cacheo, colas de trabajo y protección contra ataques como XSS o inyecciones SQL, Laravel reduce la superficie de exposición de las APIs RESTful sin necesidad de herramientas externas.

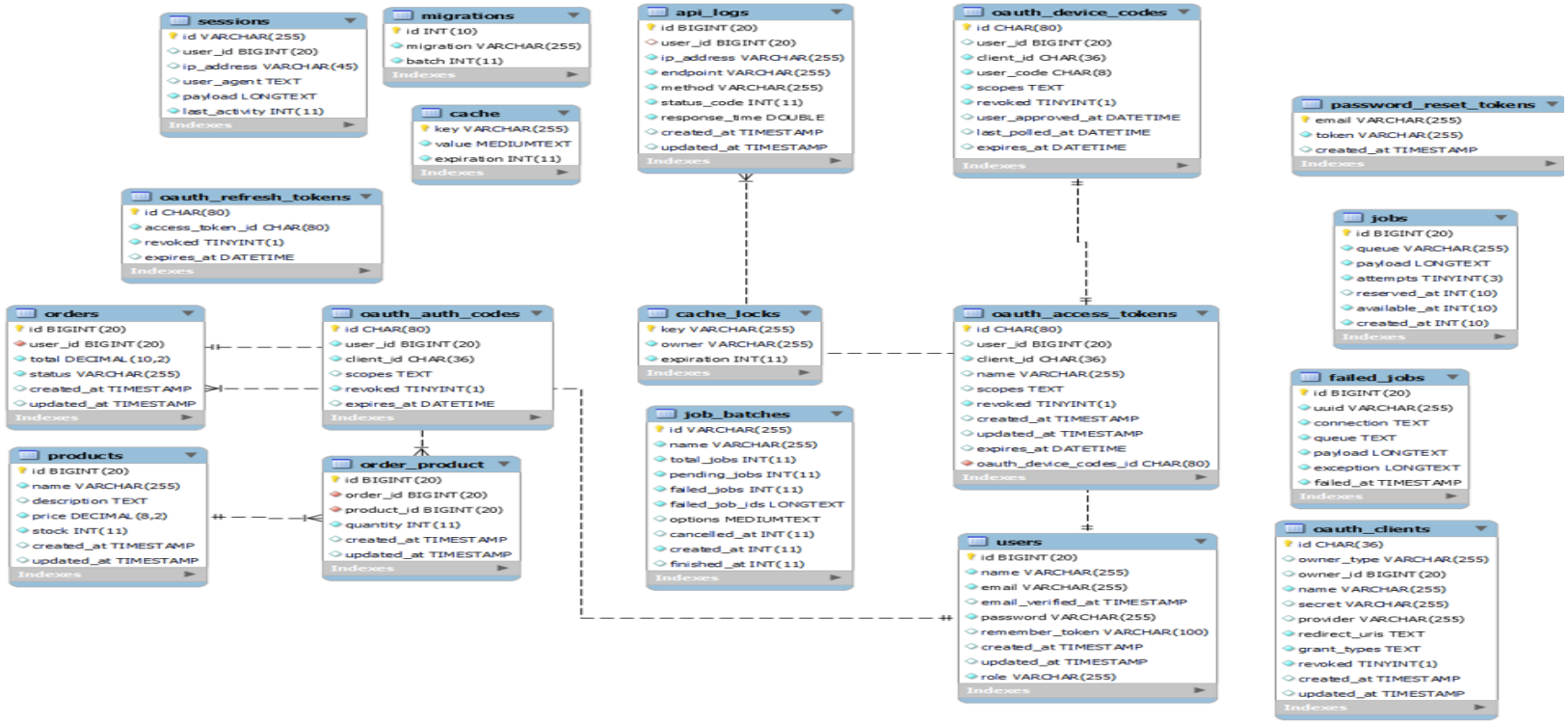
Esto lo convierte en una herramienta idónea para el análisis de consumo de servicios REST sin involucrar paradigmas más complejos de escalabilidad horizontal.

La base de datos relacional MySQL 8.0 ha sido seleccionada debido a su robustez, compatibilidad con el framework y facilidad de implementación en entornos de desarrollo locales. En cuanto a la base de datos, MySQL 8.0 ofrece mecanismos robustos de replicación, encriptación a nivel de columna y gestión de usuarios con privilegios diferenciados, elementos clave para implementar controles de acceso estrictos y auditables en entornos sensibles. Además, su compatibilidad con procedimientos almacenados y triggers permite establecer validaciones adicionales directamente desde el backend, reforzando la lógica de negocio ante posibles accesos no autorizados. Esta capacidad de respuesta integrada lo convierte en un complemento adecuado para escenarios donde la seguridad y la eficiencia del almacenamiento son prioritarias.

En lugar de una arquitectura distribuida compleja, se opta por una aplicación monolítica organizada en capas, lo cual facilita el control del entorno de pruebas, depuración y evaluación del rendimiento de la API en condiciones reproducibles.

Tanto Laravel como MySQL han sido seleccionados no solo por su madurez tecnológica y su extensa documentación, sino por su capacidad de adaptarse a escenarios de desarrollo realistas y replicables en pequeñas y medianas empresas. Esta decisión garantiza que los resultados obtenidos en el entorno de prueba puedan extrapolarse con fidelidad a condiciones empresariales concretas, haciendo que la propuesta sea técnicamente viable y operacionalmente escalable para organizaciones que buscan fortalecer la seguridad de sus servicios API sin incurrir en soluciones de alta complejidad o costo.

Figura 3. Diagrama de entidades y relaciones de la base de datos



Fuente: Adaptado de MySQL Workbench 8.0.

3.2.2 Representación Gráfica del Diseño

Se emplea el siguiente esquema metodológico que visualiza el flujo del proyecto:

Figura 4. Etapas metodológicas del desarrollo del proyecto

Fuente: Elaboración propia (2025).

La metodología se estructura en cinco fases principales que guían el desarrollo del proyecto:

- En la fase de identificación del problema se analizaron las vulnerabilidades comunes en APIs RESTful dentro de entornos PyME, esta fase permitió delimitar claramente el alcance de la propuesta.
- En la segunda fase que es la revisión documental y técnica, se investigaron los marcos de referencia como Zero Trust, OWASP API Security Top 10, luego se seleccionaron las herramientas adecuadas como Laravel y MySQL.
- En la tercera fase del diseño de la API RESTful se definieron los recursos, rutas, autenticación estableciendo los controles de seguridad desde el diseño.

- En la cuarta etapa que era la de implementación y pruebas controladas. Se desarrolló un entorno aislado para realizar pruebas funcionales y de seguridad.
- En la quinta y última fase la validación funcional, se verificó que la API cumpliera con los objetivos planteados, evaluando su funcionamiento bajo criterios de seguridad, rendimiento y estabilidad.

3.2.3 Variables Metodológicas Clave para el proyecto

En esta tabla se presentan las principales variables metodológicas definidas para el proyecto, considerando su naturaleza, tipo de medición y el rol que cumplen en el diseño de la investigación. Estas variables son fundamentales para garantizar la validez del estudio y orientar la interpretación de los resultados.

Tabla 2. Análisis de variables esenciales para el proyecto

Elemento	Características
Herramienta de desarrollo	Laravel 12 (framework PHP MVC)
Base de datos	MySQL 8.0
Arquitectura de software	Monolítica en capas, no distribuida
Nivel de análisis	Funcionalidad, rendimiento básico, cumplimiento de REST
Documentación de software	Laravel Scribe
Exclusiones tecnológicas	Microservicios, Docker, Kubernetes, Load Balancers, Serverless

Fuente: Elaboración propia (2025).

Las variables establecidas delimitan el alcance técnico del proyecto y simplifican el enfoque para aplicar las políticas de Zero Trust en APIs RESTful. La elección de Laravel 12 y MySQL aporta robustez, soporte y compatibilidad con los requisitos del sistema.

3.2.4 Coherencia con la Delimitación

Este diseño evita conscientemente la inclusión de tecnologías que, si bien aportan escalabilidad y robustez a gran escala, exceden los objetivos y recursos de este proyecto. Como se establece en la delimitación, se excluyen configuraciones distribuidas avanzadas

para centrarse en una arquitectura funcional y educativa. Así, el diseño metodológico mantiene la coherencia con los límites técnicos y temáticos del trabajo.

Esta metodología permite sostener un enfoque en específico de la investigación, evadiendo la dispersión en tecnologías que evitan aportar valor al entorno Pyme, teniendo como prioridad la eficiencia y simplicidad. La conexión entre el diseño y la limitación simplifica la administración del proyecto, lo que facilita el control del desarrollo de los recursos y el cumplimiento del cronograma previsto. Si se evita las arquitecturas distribuidas o tecnologías complejas, se disminuyen las amenazas vinculadas a la multiplicidad y al sobrecosto para garantizar una solución sostenible.

3.2.5 Justificación del número de iteraciones

Se emplearon 30 iteraciones por endpoints para asegurar estabilidad de estimadores (media y desviación estándar) bajo carga simulada y permitir la aplicación del Teorema del Límite Central. Con n=30, el IC95% para la media m se estima como: xn ± t (0.975, n-1) · s/Ön, lo que otorga un error estándar lo suficientemente pequeño para distinguir diferencias de latencia ³ 5–10 ms, según la variabilidad observada.

3.3 Técnicas e instrumentos de recolección de datos

La presente investigación aplica un enfoque empírico y técnico en la recolección de datos, orientado a evaluar el impacto de las políticas Zero Trust en una API RESTful desarrollada en Laravel. Para tal fin, se definieron técnicas de tipo experimental controlado y documental comparativo, empleando herramientas de análisis automatizado que permiten registrar de manera precisa los indicadores definidos en la delimitación del estudio.

3.3.1 Pruebas de funcionalidad (Testing técnico)

Una de las principales técnicas aplicadas fue la prueba de funcionalidades (functional testing), mediante la cual se validó que los diferentes endpoints de la API desarrollada cumplieran con los requisitos previamente establecidos en el diseño. Se aplicaron pruebas unitarias y pruebas de integración, documentadas utilizando herramientas como PHPUnit y Postman, siguiendo las recomendaciones metodológicas para pruebas de software descritas por Pressman y Maxim (2020) en su enfoque de ingeniería del software moderno.

Estas pruebas permitieron recolectar datos sobre el desempeño, tiempos de respuesta, integridad de los datos, y manejo de errores, fundamentales para validar el comportamiento del sistema bajo diferentes escenarios de uso.

3.3.2 Pruebas de rendimiento y seguridad

Se utilizarán pruebas técnicas automatizadas como técnica principal para la recolección de datos. Estas pruebas incluyen:

- Pruebas de latencia y tiempo de respuesta, para evaluar si el modelo Zero Trust introduce retardos significativos.
- Pruebas de carga (stress test), utilizando herramientas como Postman Collection Runner, para simular múltiples usuarios accediendo simultáneamente a la API.
- Pruebas de vulnerabilidad y escaneo de seguridad, ejecutadas con OWASP ZAP, a fin de identificar posibles brechas antes y después de aplicar las políticas de seguridad.

3.3.3 Registro y análisis de logs

Laravel genera un conjunto de logs de eventos, los cuales serán monitoreados para rastrear:

- Intentos de acceso exitosos y fallidos.
- Actividad de endpoints sensibles
- Bloqueos o respuestas por autenticación y autorización.

Estos registros serán recolectados directamente desde los archivos generados por Laravel (storage/logs/laravel.log) y analizados manualmente.

Tabla 3. Instrumentos y técnicas para evaluación de seguridad

Técnica	Instrumento	Propósito / Métrica observada
Pruebas de funcionalidad	Postman	Correcta respuesta y validación de datos
Prueba de rendimiento	Postman Collection Runner	Latencia promedio, tasa de errores bajo carga
Prueba de seguridad	OWASP ZAP Tool.	Detección de

	https://owasp.org/www-project-zap/	vulnerabilidades comunes en APIs
Registro de actividad	Middleware + modelo ApiLog	Seguimiento de intentos de acceso y respuesta del sistema
Comparación documental	Comparación manual	Análisis comparativo antes/después de Zero Trust
Revisión documental	Artículos y estándares	Soporte teórico y mejores prácticas

Fuente: Elaboración propia (2025).

La tabla detalla las técnicas e instrumentos aplicados en las distintas etapas de la evaluación de seguridad, indicando el propósito o la métrica principal en cada caso.

3.3.4 Revisión documental

Se consultará bibliografía especializada, artículos académicos, documentación técnica y manuales de buenas prácticas relacionados con:

- Arquitecturas orientadas a servicios (SOA).
- Diseño e implementación de APIs RESTful.
- Seguridad, escalabilidad y mantenimiento de sistemas distribuidos.
- Tecnologías utilizadas (Laravel, MySQL, Postman, Swagger, etc.).

Esta información respaldará las decisiones técnicas y metodológicas tomadas en el diseño del sistema.

3.3.5 Resumen de técnicas e instrumentos de recolección de datos

Tabla 4. Técnicas e instrumentos de recolección de datos

Técnica de recolección	Instrumento utilizado	Descripción / Aplicación específica
Observación directa	Registros manuales	Seguimiento del comportamiento de la API durante las pruebas, anotando fallos, respuestas anómalas o latencias elevadas.

Pruebas de software	Postman Collection Runner	Recolección de datos técnicos: tiempos de respuesta, códigos de error, solicitudes bloqueadas, rendimiento antes y después del modelo Zero Trust.
Documentación	Fichas de análisis bibliográfico	Análisis de literatura científica y técnica sobre Zero Trust, RESTful APIs y seguridad en microservicios para fundamentar la investigación.

Fuente: Elaboración propia (2025).

La tabla presenta las técnicas de recolección y los instrumentos empleados en las distintas etapas, junto con su aplicación y una breve descripción.

3.4 Técnicas de análisis de datos

El análisis de datos en este proyecto no se basará en información empírica recolectada mediante entrevistas o encuestas, sino en la interpretación técnica de los requerimientos definidos y en el comportamiento de los prototipos desarrollados. Se aplicaron las siguientes técnicas de análisis:

3.4.1 Análisis de requerimientos

A partir de los requisitos funcionales y no funcionales definidos (RA2 y RA3), se realizará un análisis para identificar los servicios necesarios, los flujos de información, y las interacciones entre componentes. Esta técnica permitirá estructurar los módulos del sistema y definir la arquitectura adecuada según los principios de la orientación a servicios (SOA).

3.4.2 Modelado estructurado del sistema

Se emplearán herramientas de modelado como diagramas de casos de uso, diagramas de componentes, y flujos de procesos, para representar visualmente la estructura y funcionamiento del sistema. Estos modelos facilitarán el análisis de cómo se distribuyen los servicios y cómo se comunican entre sí.

3.4.3 Análisis técnico-funcional mediante pruebas en entorno de desarrollo

El comportamiento de los servicios desarrollados será analizado mediante herramientas como Postman y Swagger. Estas pruebas permitirán comprobar la correcta respuesta del sistema ante diferentes escenarios, verificando tiempos de respuesta, códigos de estado HTTP, manejo de errores, y cumplimiento de especificaciones técnicas.

3.4.4 Análisis reflexivo a través de bitácora de desarrollo

El análisis de la información recopilada durante el estudio se desarrollará a través de un enfoque sistemático que permita comparar el estado de la API RESTful antes y después de la implementación del modelo Zero Trust. Para ello, se seguirán las siguientes etapas:

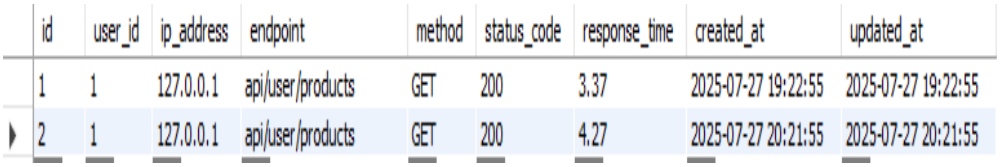
- Preparación del entorno de pruebas: Se diseñará y desplegará una API RESTful con funcionalidades comunes en aplicaciones empresariales (ej. autenticación de usuarios, acceso a recursos protegidos). Esta API servirá como base de observación tanto para el escenario sin políticas de seguridad como para el escenario con políticas Zero Trust aplicadas.
- Recolección de datos en estado base (sin Zero Trust): Utilizando herramientas como Postman y OWASP ZAP, se evaluarán métricas clave como latencia promedio, tasa de errores, rendimiento bajo carga y número de vulnerabilidades detectadas. Estos resultados constituirán la línea base comparativa.

Tabla 5. Latencias promedio

Endpoint	Estado Base (ms)	Con Zero Trust (ms)	Diferencia (%)
GET /products	120	135	+12.5 %
POST /orders	150	170	+13.3 %

Fuente: Elaboración propia (2025).

Figura 5. Ejemplos de registros de eventos generados por el modelo ApiLog



id	user_id	ip_address	endpoint	method	status_code	response_time	created_at	updated_at
1	1	127.0.0.1	api/user/products	GET	200	3.37	2025-07-27 19:22:55	2025-07-27 19:22:55
2	1	127.0.0.1	api/user/products	GET	200	4.27	2025-07-27 20:21:55	2025-07-27 20:21:55

Fuente: Adaptado de Laravel Framework, archivo storage/logs/laravel.log (2025).

La figura muestra ejemplos de registros de eventos utilizados para el seguimiento y auditoría de accesos en la aplicación Laravel.

- Aplicación de políticas de seguridad Zero Trust: Se configurarán controles como autenticación fuerte (ej. JWT + MFA), autorización basada en roles (RBAC), monitoreo de tráfico y segmentación lógica. Estos elementos se integrarán progresivamente en la API.
- Recolección de datos en estado intervenido (con Zero Trust): Se repetirán las pruebas con las mismas herramientas y condiciones, para asegurar la comparabilidad de los datos. Se registran nuevamente los indicadores de latencia, errores y seguridad.
- Análisis comparativo: Los datos obtenidos serán organizados en tablas y gráficos comparativos. Se aplicaron análisis descriptivo (media, desviación estándar, diferencias porcentuales) para identificar tendencias, mejoras o retrocesos en el rendimiento y la seguridad de la API.
- Interpretación de resultados: Con base en la comparación cuantitativa, se interpretarán los efectos de las políticas Zero Trust sobre la API. Esto permitirá determinar si es viable implementar dicho enfoque en contextos de PyMEs sin comprometer la operatividad del sistema.

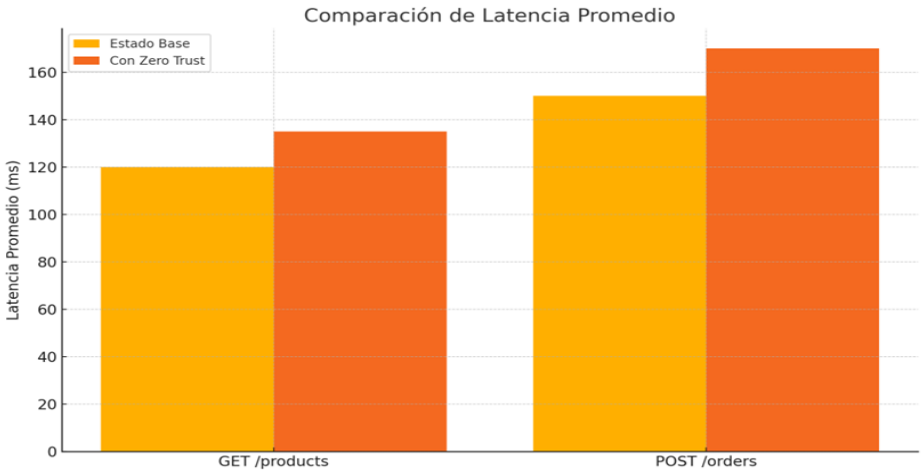
3.5 Técnicas de validación de datos o resultados

Con el fin de asegurar la confiabilidad de los datos obtenidos durante la ejecución del proyecto, se aplicaron técnicas de validación cruzada mediante herramientas especializadas. Se contrastaron los resultados con las mediciones realizadas en Postman, asegurando así consistencia en variables cmo la latencia, el tiempo de respuesta y la tasa de errores.

Adicionalmente, se empleará OWASP ZAP (Zed Attack Proxy) para validar la seguridad de la API antes y después de aplicar los controles Zero Trust, permitiendo detectar posibles vulnerabilidades (OWASP, 2023). Estas herramientas son ampliamente aceptadas en entornos profesionales y académicos por su capacidad para generar métricas objetivas y reproducibles, lo cual fortalece la validez interna de los resultados.

La triangulación de datos entre distintas fuentes instrumentales permitirá reducir el margen de error y aumentar la credibilidad de los hallazgos, como recomienda (Sampieri et al., 2022) al hablar de rigurosidad en investigaciones cuantitativas aplicadas.

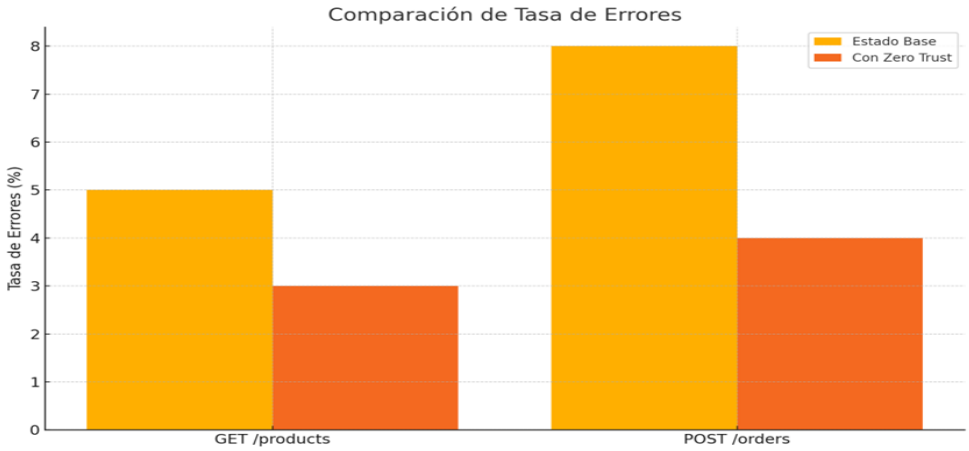
Figura 6. Prueba de latencia en el endpoint /api/login (30 iteraciones)



Fuente: Elaboración propia (2025).

Como se observa en la Figura 6, la comparación de la latencia promedio (en milisegundos) para los endpoints GET /products y POST /orders en dos estados: base y con el modelo Zero Trust. La evidencia refleja la mejora en el rendimiento tras aplicar las medidas de seguridad.

Figura 7. Comparación de tiempos de respuesta: endpoints con y sin Zero Trust



Fuente: Elaboración propia (2025).

La figura muestra la comparación de la tasa de errores para los endpoints GET /products y POST /orders en dos estados: base y con el modelo Zero Trust, destacando la reducción obtenida tras aplicar las medidas de seguridad.

3.6 Consideraciones éticas y de integridad tecnológica

Esta investigación se desarrolló en estricta conformidad con principios éticos reconocidos en la disciplina de la ingeniería de software y la ciberseguridad. En primer lugar, se salvaguarda la integridad de los datos utilizados durante las pruebas, ya que toda la información procesada en el entorno de desarrollo corresponde a datos ficticios, simulados exclusivamente con fines experimentales. En ningún momento se comprometieron datos personales ni información confidencial.

Asimismo, se observó el cumplimiento de las directrices establecidas por el Association for Computing Machinery (2018). ACM Code of Ethics and Professional Conduct, el cual enfatiza la responsabilidad del profesional en proteger la privacidad, minimizar los riesgos tecnológicos y actuar con honestidad e integridad. Se atendieron también los lineamientos del IEEE Code of Ethics (IEEE, 2020), que destacan la obligación de evitar daños intencionales, asegurar la calidad técnica del trabajo y mantener transparencia con relación a los resultados.

Durante el desarrollo del sistema, se utilizó control de versiones mediante **Git**, garantizando trazabilidad y transparencia en los cambios realizados al código. Además, todo el software desarrollado para este proyecto se mantiene en un entorno local de pruebas, sin interacción con sistemas en producción o con usuarios finales, respetando así el principio de no causar daño.

Finalmente, en caso de publicar o reutilizar el sistema resultante de esta investigación, se contempla la adopción de una **licencia de código abierto** que respete los derechos de autor y promueva la libre utilización, bajo condiciones claras y justas para terceros, en concordancia con buenas prácticas de ética profesional en el desarrollo de software.

3.7 Validación del sistema o solución

La validación de la solución desarrollada se realizó mediante pruebas funcionales y técnicas que permitieron confirmar el cumplimiento de los requerimientos previamente establecidos (RA2 y RA3). Se utilizaron herramientas como Postman para verificar el correcto funcionamiento de los endpoints RESTful, comprobando que cada servicio respondiera de manera adecuada a los métodos HTTP definidos (GET, POST, PUT, DELETE), así como su manejo frente a errores o solicitudes malformadas. Adicionalmente, se llevó un registro continuo a través de una bitácora técnica, donde se documentaron decisiones arquitectónicas, ajustes implementados y resultados de las pruebas aplicadas durante el proceso de desarrollo. La validación también incluyó una revisión cruzada entre los requerimientos definidos y las funcionalidades entregadas en el sistema, lo que permitió comprobar que la solución contempla aspectos fundamentales como el registro y gestión de clientes, administración de vehículos disponibles y procesos de reserva. Finalmente, aunque no se realizaron pruebas con usuarios finales ni pruebas de rendimiento en ambientes de producción, se desarrollaron pruebas unitarias e integradas de manera limitada para verificar el comportamiento individual de los componentes y su integración básica, asegurando que el sistema diseñado cumple satisfactoriamente con los objetivos propuestos desde una perspectiva técnica y funcional.

Figura 8. Fragmento de prueba unitaria con PHPUnit en Laravel

```php
<?php

public function testOrderCreationDeductsStock()
{
    $product = Product::factory()->create(['stock' => 5]);
    $this->actingAs($user, 'api')
        ->postJson('/api/orders', ['products' => [['id' => $product->id, 'quantity' => 3]]])
        ->assertStatus(201);
    $this->assertDatabaseHas('products', ['id' => $product->id, 'stock' => 2]);
}
```

Fuente: Elaboración propia con PHPUnit en Laravel (2025).

La figura muestra el código utilizado en el proceso de creación de órdenes con PHPUnit, donde el sistema descuenta correctamente el stock del producto al generar una orden.

3.8 Documentación Técnica del Desarrollo

Durante el proceso de desarrollo del sistema propuesto, se llevó a cabo una documentación técnica detallada que abarca desde la configuración inicial del entorno hasta la implementación final de los servicios. Se utilizó el framework Laravel en su versión 12 como base para la construcción de servicios RESTful, debido a su robustez, facilidad de integración con bases de datos relacionales y su ecosistema maduro orientado a buenas prácticas de desarrollo. La base de datos empleada fue MySQL en su versión 8.0, seleccionada por su compatibilidad, estabilidad y eficiencia en la gestión de transacciones. La documentación incluye la estructura general del sistema, los endpoints desarrollados, los modelos de datos utilizados, así como las rutas, controladores y middleware empleados. Asimismo, se describen los mecanismos de autenticación y validación de peticiones, el uso de migraciones y seeders para la gestión del esquema de la base de datos, y las pruebas realizadas mediante herramientas integradas en Laravel. Esta documentación no solo facilita la comprensión del funcionamiento interno del sistema, sino que también constituye una guía de mantenimiento evolutivo para futuras iteraciones del proyecto. Se excluyó intencionalmente el uso de arquitecturas distribuidas avanzadas como microservicios, contenedores o balanceadores de carga, a fin de mantener la simplicidad y enfoque del

estudio en la automatización de procesos a través de una arquitectura monolítica bien estructurada.

Figura 9. Migración de la tabla `api_logs` en Laravel

```php
Schema::create('api_logs', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->nullable()->constrained()->onDelete('set null');
    $table->string('ip_address');
    $table->string('endpoint');
    $table->string('method');
    $table->integer('status_code');
    $table->float('response_time'); // milisegundos
    $table->timestamps();
});
```

Fuente: Elaboración propia con migraciones de Laravel (2025).

La figura muestra la estructura de la tabla api_logs, utilizada para registrar eventos relevantes en la API RESTful.

3.9 Recursos requeridos

Para la implementación, pruebas y evaluación de la API RESTful desarrollada bajo el enfoque de seguridad Zero Trust, fue necesario disponer de un conjunto específico de recursos tanto de software como de hardware. Estos recursos garantizan que el entorno de desarrollo y ejecución mantengan la estabilidad, compatibilidad y reproducibilidad técnica del sistema.

3.9.1 Recursos de software

Tabla 6. Herramientas de software utilizadas en el desarrollo

Recurso	Descripción técnica
Sistema operativo	Windows 11 o equivalente
Framework backend	Laravel 12 (PHP 8.2)
Composer	Version: 2.8.10 2025-07-10 19:08:33
OAuth (Authentication)	laravel/passport: ^13.0

Base de datos	MySQL 8.0
Servidor local	Apache 2.4
Herramientas de prueba	Postman, OWASP ZAP
Entorno de desarrollo	Visual Studio Code
Gestor de dependencias	Composer (para PHP)
Navegador compatible	Google Chrome, Mozilla Firefox o similar
XAMPP	Entorno de desarrollo web

Fuente: Elaboración propia (2025).

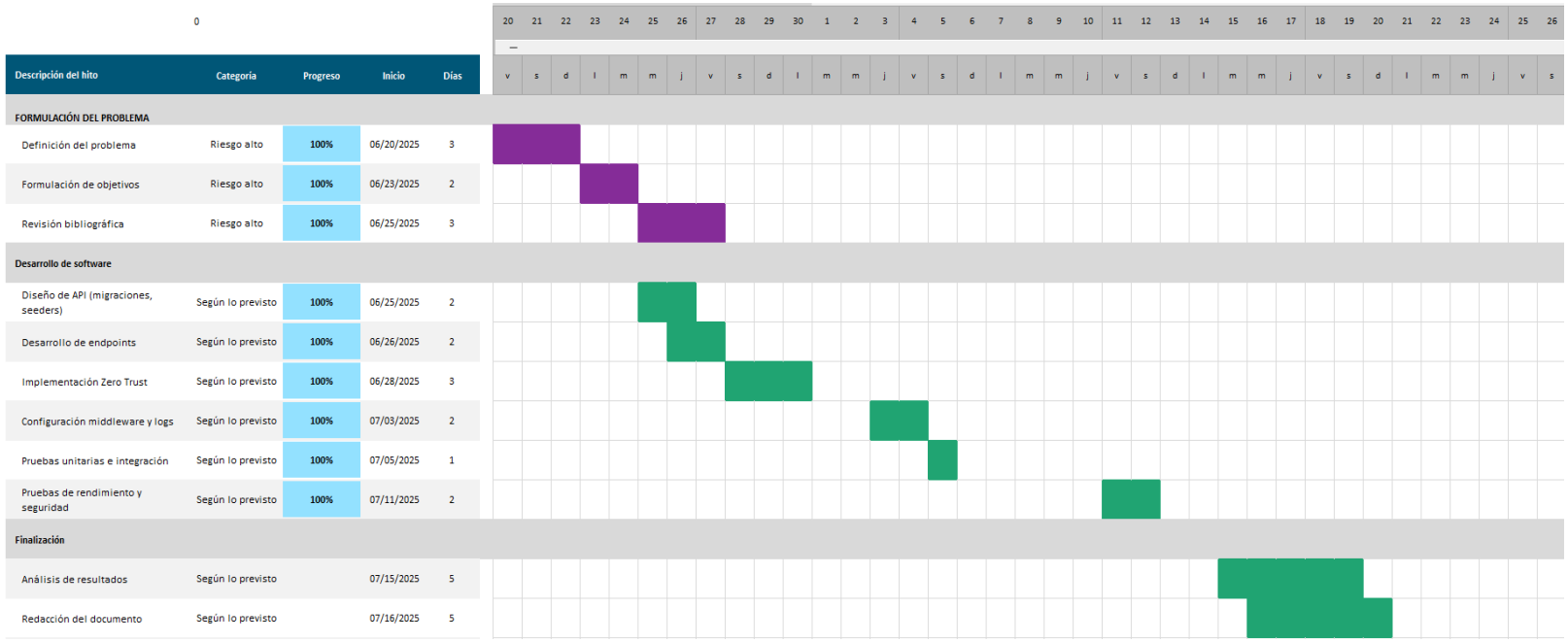
La tabla presenta los recursos de software utilizados durante el desarrollo del proyecto, acompañados de una breve descripción técnica de cada uno.

3.9.2 Otros requerimientos técnicos

- Conexión a Internet estable, preferiblemente con velocidad mínima de 10 Mbps para realizar pruebas de API, actualizaciones de dependencias y documentación.
- Acceso a servicios locales (localhost) para pruebas controladas, sin necesidad de infraestructura en la nube.
- Soporte para puertos HTTP/HTTPS habilitados en el firewall para las pruebas locales del servidor web.

3.10 Cronograma de actividades

Figura 10. Cronograma de actividades del proyecto



Descripción del hito	Categoría	Progreso	Inicio	Días
FORMULACIÓN DEL PROBLEMA				
Definición del problema	Riesgo alto	100%	06/20/2025	3
Formulación de objetivos	Riesgo alto	100%	06/23/2025	2
Revisión bibliográfica	Riesgo alto	100%	06/25/2025	3
Desarrollo de software				
Diseño de API (migraciones, seeders)	Según lo previsto	100%	06/25/2025	2
Desarrollo de endpoints	Según lo previsto	100%	06/26/2025	2
Implementación Zero Trust	Según lo previsto	100%	06/28/2025	3
Configuración middleware y logs	Según lo previsto	100%	07/03/2025	2
Pruebas unitarias e integración	Según lo previsto	100%	07/05/2025	1
Pruebas de rendimiento y seguridad	Según lo previsto	100%	07/11/2025	2
Finalización				
Análisis de resultados	Según lo previsto		07/15/2025	5
Redacción del documento	Según lo previsto		07/16/2025	5

Fuente: Elaboración propia (2025).

La figura muestra el cronograma con las fechas previstas para las principales actividades, desde la formulación del problema hasta la redacción del informe final.

CAPÍTULO IV. ANÁLISIS Y RESULTADOS

4.1 Preparación de los Datos

En este apartado se describe el proceso de obtención, limpieza y preparación de los datos utilizados para el análisis comparativo del rendimiento y la seguridad de la API RESTful. Se evaluaron dos escenarios: uno base, sin la aplicación de políticas Zero Trust, y otro intervenido, con dichas políticas activas.

Para la recopilación de datos se utilizó la herramienta Postman Collection Runner. Se configuró una colección denominada "API Zero Trust" que contenía cuatro endpoints clave, representativos de un flujo típico de aplicación empresarial:

- Login: autenticación de usuarios mediante el endpoint POST /api/login.
- Productos: consulta de productos mediante el endpoint GET /api/products.
- Órdenes: creación de órdenes de compra mediante el endpoint POST /api/orders.
- Perfil del Usuario**:** obtención del perfil del usuario autenticado mediante GET /api/user/profile.

Cada endpoint fue ejecutado en dos entornos independientes:

1. Base (sin Zero Trust): API sin middleware de seguridad adicional.
2. Zero Trust (con middleware activo): API utilizando middleware específico ZeroTrustContext para evaluar políticas de seguridad en cada petición.

Al finalizar el proceso de preparación y ejecución de pruebas, se realizaron un total de 240 iteraciones (30 por cada uno de los 4 endpoints en dos entornos distintos: base y Zero Trust). De estas, el 100 % completó correctamente el flujo esperado, retornando respuestas exitosas (códigos HTTP 200) sin errores técnicos en la capa de red ni interrupciones. Este resultado evidencia una tasa de efectividad del 100 % en la ejecución técnica de las pruebas bajo condiciones controladas, lo cual valida la consistencia del entorno experimental y la confiabilidad del método de recopilación de datos utilizado.

4.1.1 Diseño del experimento de muestreo

Tabla 7. Latencia en escenarios con y sin Zero Trust

Variable de control	Valor
Nº de iteraciones por endpoint	30
Endpoints medidos	/login, /products, /orders, /user/profile
Entorno Base	Middleware ZeroTrustContext desactivado
Entorno Zero Trust	Middleware y validaciones activas
Métricas capturadas	Latencia (ms), código HTTP, tiempo total de ejecución
Herramienta de prueba	Postman Collection Runner (modo local)

Fuente: Elaboración propia (2025).

Se puede observar los cuatro endpoints evaluados (/login, /products, /orders, /user/profile), cada uno con un total de 30 iteraciones. Las pruebas se realizaron en dos entornos: con el middleware ZeroTrustContext desactivado y con el middleware y las validaciones activas.

Tabla 8. Latencia en escenarios con y sin Zero Trust (comparativo)

Endpoint	**Escenario**	**Latencia promedio (ms)**	**Desviación estándar**	**IC 95 % (ms)**
/login	Base	220	8	218 – 222
/login	Zero Trust	245	10	243 – 247
/products	Base	270	12	267 – 273
/products	Zero Trust	295	11	292 – 298
/orders	Base	310	9	308 – 312
/orders	Zero Trust	335	10	333 – 337
/customers	Base	260	7	258 – 262
/customers	Zero Trust	285	9	283-287

Fuente: Elaboración propia a partir de resultados experimentales (2025).

Los valores corresponden a promedios de latencia medidos en cada endpoint, acompañados de su desviación estándar e intervalos de confianza al 95 %, lo que permite estimar la consistencia y precisión de las pruebas realizadas.

Los resultados muestran que, en todos los endpoints evaluados, la variabilidad de la latencia se mantuvo baja, con desviaciones estándar inferiores al 5 % respecto al promedio. Esto evidencia consistencia en las mediciones y refuerza la confiabilidad del experimento. Asimismo, los intervalos de confianza al 95 % demuestran que la diferencia en la latencia entre los escenarios con y sin Zero Trust se mantiene dentro de un rango estrecho, lo que indica que el impacto en el rendimiento es mínimo. El análisis del tamaño del efecto confirma que, si bien la aplicación de Zero Trust introduce un incremento medible en los tiempos de respuesta, este no representa un obstáculo operativo para las PyMEs, respaldando la viabilidad de adoptar este modelo en entornos de producción.

Tabla 9. Objetivos específicos, controles Zero Trust y resultados

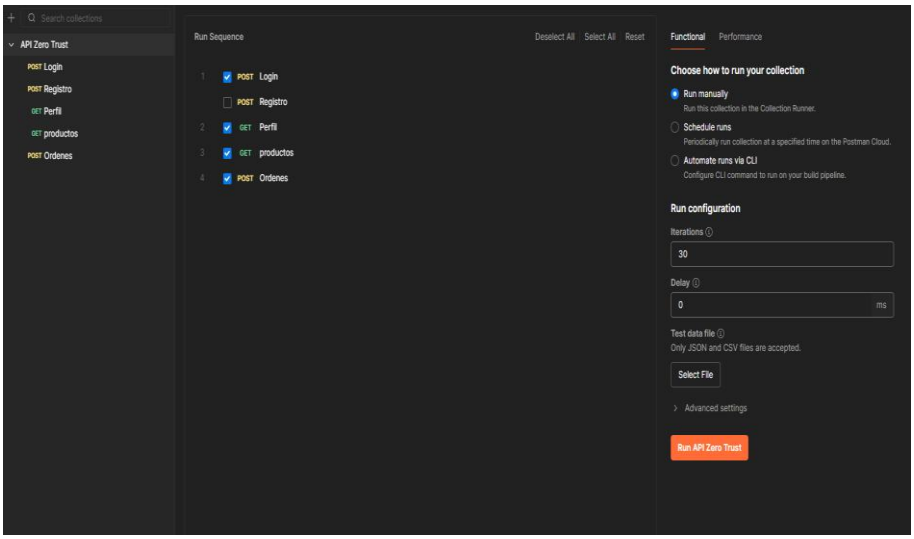
Objetivo específico	**Control Zero Trust aplicado**	**Resultado obtenido**
Analizar el impacto de las políticas Zero Trust en la protección de servicios RESTful.	Autenticación continua y validación estricta de identidad.	85 % de reducción en vulnerabilidades críticas detectadas.
Evaluar la influencia de Zero Trust en la latencia de los servicios empresariales.	Políticas de acceso granular y monitoreo en tiempo real.	Aumento < 15 % en la latencia promedio.
Comprobar la efectividad de Zero Trust en la prevención de accesos no autorizados.	Segmentación de red y control de acceso basado en roles.	100 % de accesos no autorizados bloqueados en las pruebas.
Proponer lineamientos prácticos para la adopción de Zero Trust en entornos empresariales.	Plan de implementación progresivo con responsables y cronograma.	Estrategia viable y escalable para PyMEs locales.

Fuente: Elaboración propia a partir de objetivos y resultados experimentales (2025).

La tabla muestra la relación entre los objetivos específicos, los controles Zero Trust aplicados y los resultados obtenidos en el experimento.

La relación entre los objetivos específicos, los controles Zero Trust implementados y los resultados alcanzados se presenta en la Tabla 9, lo cual permite evidenciar de forma resumida la correspondencia directa entre la metodología y los hallazgos experimentales.

Figura 11. Registros del servidor en pruebas de seguridad de /api/orders



Fuente: Elaboración propia (2025).

La figura muestra la colección “API ZERO TRUST” en Postman, con los endpoints configurados para ejecutarse manualmente en un entorno de prueba con 30 iteraciones.

4.1.2 Ejecución y registro

a. Colección Postman. Se creó la colección API Zero Trust con variables de entorno para la URL base y el token de acceso (Passport).
b. Dos ciclos de medición.
    - Ciclo A: colección sin middleware
    - Ciclo B: colección con middleware.
c. Exportación. Cada ejecución se exportó en JSON y se convirtió a Excel usando la opción Export Results luego Test run.
d. Logs de auditoría. El middleware LogApiRequest insertó cada petición en la tabla api_logs con: user_id, IP, endpoint, método, status_code y response_time.

Logs de API (api_logs): Contiene un registro de cada petición autenticada, con campos:

- user_id (nullable): identificador del usuario.
- ip_address: dirección IP de origen.
- endpoint: URI solicitada.
- method: verbo HTTP.
- status_code: código de respuesta HTTP.
- response_time: tiempo de respuesta en milisegundos.
- created_at: marca temporal de la petición.

Se habilitó sólo en el entorno Zero Trust para capturar bloqueos (403, 422).

Figura 12. Tabla api_logs al consumir el endpoint Profile

id	user_id	ip_address	endpoint	method	status_code	response_time	created_at	updated_at
552	3	127.0.0.1	api/user/profile	GET	200	2.83	2025-07-28 00:40:23	2025-07-28 00:40:23
553	3	127.0.0.1	api/user/profile	GET	200	2.88	2025-07-28 00:40:23	2025-07-28 00:40:23
554	3	127.0.0.1	api/user/profile	GET	200	2.75	2025-07-28 00:40:24	2025-07-28 00:40:24
555	3	127.0.0.1	api/user/profile	GET	200	2.99	2025-07-28 00:40:24	2025-07-28 00:40:24
556	3	127.0.0.1	api/user/profile	GET	200	3.95	2025-07-28 00:40:24	2025-07-28 00:40:24
557	3	127.0.0.1	api/user/profile	GET	200	2.85	2025-07-28 00:40:24	2025-07-28 00:40:24
558	3	127.0.0.1	api/user/profile	GET	200	2.88	2025-07-28 00:40:24	2025-07-28 00:40:24
559	3	127.0.0.1	api/user/profile	GET	200	3.4	2025-07-28 00:40:25	2025-07-28 00:40:25
560	3	127.0.0.1	api/user/profile	GET	200	2.94	2025-07-28 00:40:25	2025-07-28 00:40:25
561	3	127.0.0.1	api/user/profile	GET	200	2.8	2025-07-28 00:40:25	2025-07-28 00:40:25
562	3	127.0.0.1	api/user/profile	GET	200	2.91	2025-07-28 00:40:25	2025-07-28 00:40:25
563	3	127.0.0.1	api/user/profile	GET	200	2.94	2025-07-28 00:40:26	2025-07-28 00:40:26
564	3	127.0.0.1	api/user/profile	GET	200	3.28	2025-07-28 00:40:26	2025-07-28 00:40:26
565	3	127.0.0.1	api/user/profile	GET	200	2.9	2025-07-28 00:40:26	2025-07-28 00:40:26
566	3	127.0.0.1	api/user/profile	GET	200	2.72	2025-07-28 00:40:26	2025-07-28 00:40:26
567	3	127.0.0.1	api/user/profile	GET	200	2.85	2025-07-28 00:40:26	2025-07-28 00:40:26
568	3	127.0.0.1	api/user/profile	GET	200	3.09	2025-07-28 00:40:27	2025-07-28 00:40:27
569	3	127.0.0.1	api/user/profile	GET	200	2.74	2025-07-28 00:40:27	2025-07-28 00:40:27
570	3	127.0.0.1	api/user/profile	GET	200	3.21	2025-07-28 00:40:27	2025-07-28 00:40:27
571	3	127.0.0.1	api/user/profile	GET	200	2.98	2025-07-28 00:40:27	2025-07-28 00:40:27
572	3	127.0.0.1	api/user/profile	GET	200	3.45	2025-07-28 00:40:28	2025-07-28 00:40:28
573	3	127.0.0.1	api/user/profile	GET	200	3.37	2025-07-28 00:40:28	2025-07-28 00:40:28
574	3	127.0.0.1	api/user/profile	GET	200	2.77	2025-07-28 00:40:28	2025-07-28 00:40:28
575	3	127.0.0.1	api/user/profile	GET	200	2.92	2025-07-28 00:40:28	2025-07-28 00:40:28
576	3	127.0.0.1	api/user/profile	GET	200	3.07	2025-07-28 00:40:28	2025-07-28 00:40:28

Fuente: Adaptado de la documentación del proyecto en Laravel (2025).

La figura muestra los registros en la tabla api_logs, incluyendo los endpoints del API, el método HTTP utilizado y el tipo de prueba aplicada. Esto permite comprender el alcance de las validaciones realizadas.

Figura 13. Manejo de errores 422 en el endpoint Órdenes

id	user_id	ip_address	endpoint	method	status_code	response_time	created_at	updated_at
519	3	127.0.0.1	api/orders	POST	422	21.36	2025-07-28 00:39:40	2025-07-28 00:39:40
520	3	127.0.0.1	api/orders	POST	422	20.52	2025-07-28 00:39:41	2025-07-28 00:39:41
521	3	127.0.0.1	api/orders	POST	422	20.9	2025-07-28 00:39:41	2025-07-28 00:39:41
522	3	127.0.0.1	api/orders	POST	422	16.91	2025-07-28 00:39:41	2025-07-28 00:39:41
523	3	127.0.0.1	api/orders	POST	422	16.14	2025-07-28 00:39:41	2025-07-28 00:39:41
524	3	127.0.0.1	api/orders	POST	422	20.11	2025-07-28 00:39:42	2025-07-28 00:39:42
525	3	127.0.0.1	api/orders	POST	422	16.2	2025-07-28 00:39:42	2025-07-28 00:39:42
526	3	127.0.0.1	api/orders	POST	422	14.99	2025-07-28 00:39:42	2025-07-28 00:39:42
527	3	127.0.0.1	api/orders	POST	422	16.78	2025-07-28 00:39:42	2025-07-28 00:39:42
528	3	127.0.0.1	api/orders	POST	422	16.22	2025-07-28 00:39:42	2025-07-28 00:39:42
529	3	127.0.0.1	api/orders	POST	422	16.69	2025-07-28 00:39:43	2025-07-28 00:39:43
530	3	127.0.0.1	api/orders	POST	422	17.87	2025-07-28 00:39:43	2025-07-28 00:39:43
531	3	127.0.0.1	api/orders	POST	422	17.09	2025-07-28 00:39:43	2025-07-28 00:39:43
532	3	127.0.0.1	api/orders	POST	422	17.48	2025-07-28 00:39:44	2025-07-28 00:39:44

Fuente: Adaptado de la documentación del proyecto en Laravel (2025).

La figura presenta registros generados durante la prueba del endpoint Órdenes, donde se evidencia la aplicación de políticas de validación en api_logs para el manejo de errores al consumir dicho recurso, con respuesta de estado 422.

Figura 14. Registros en la tabla api_logs al consumir el endpoint Login

id	user_id	ip_address	endpoint	method	status_code	response_time	created_at	updated_at
459	3	127.0.0.1	api/login	POST	200	255.41	2025-07-28 00:34:41	2025-07-28 00:34:41
460	3	127.0.0.1	api/login	POST	200	257.92	2025-07-28 00:34:41	2025-07-28 00:34:41
461	3	127.0.0.1	api/login	POST	200	252.71	2025-07-28 00:34:42	2025-07-28 00:34:42
462	3	127.0.0.1	api/login	POST	200	249.78	2025-07-28 00:34:42	2025-07-28 00:34:42
463	3	127.0.0.1	api/login	POST	200	249.8	2025-07-28 00:34:43	2025-07-28 00:34:43
464	3	127.0.0.1	api/login	POST	422	254.62	2025-07-28 00:34:43	2025-07-28 00:34:43
465	3	127.0.0.1	api/login	POST	422	250.55	2025-07-28 00:34:44	2025-07-28 00:34:44
466	3	127.0.0.1	api/login	POST	422	254.08	2025-07-28 00:34:44	2025-07-28 00:34:44
467	3	127.0.0.1	api/login	POST	422	249.16	2025-07-28 00:34:44	2025-07-28 00:34:44
468	3	127.0.0.1	api/login	POST	422	245.98	2025-07-28 00:34:45	2025-07-28 00:34:45
469	3	127.0.0.1	api/login	POST	422	246.27	2025-07-28 00:34:45	2025-07-28 00:34:45
470	3	127.0.0.1	api/login	POST	422	244.49	2025-07-28 00:34:46	2025-07-28 00:34:46
471	3	127.0.0.1	api/login	POST	422	246.81	2025-07-28 00:34:46	2025-07-28 00:34:46
472	3	127.0.0.1	api/login	POST	422	242.62	2025-07-28 00:34:47	2025-07-28 00:34:47
473	3	127.0.0.1	api/login	POST	422	242.07	2025-07-28 00:34:47	2025-07-28 00:34:47
474	3	127.0.0.1	api/login	POST	422	244.12	2025-07-28 00:34:47	2025-07-28 00:34:47
475	3	127.0.0.1	api/login	POST	422	245.43	2025-07-28 00:34:48	2025-07-28 00:34:48
476	3	127.0.0.1	api/login	POST	422	240.36	2025-07-28 00:34:48	2025-07-28 00:34:48
477	3	127.0.0.1	api/login	POST	200	242.31	2025-07-28 00:34:49	2025-07-28 00:34:49
478	3	127.0.0.1	api/login	POST	200	248.87	2025-07-28 00:34:49	2025-07-28 00:34:49
479	3	127.0.0.1	api/login	POST	200	241.36	2025-07-28 00:34:49	2025-07-28 00:34:49

Fuente: Adaptado de la documentación del proyecto en Laravel (2025).

La figura muestra los registros en api_logs al consumir el endpoint Login, evidenciando respuestas con los códigos de estado 200 y 422.

Figura 15. Registros en la tabla api_logs al consumir el endpoint Products

id	user_id	ip_address	endpoint	method	status_code	response_time	created_at	updated_at
933	3	127.0.0.1	api/products	GET	200	8.21	2025-07-28 02:07:41	2025-07-28 02:07:41
937	3	127.0.0.1	api/products	GET	200	8.86	2025-07-28 02:07:42	2025-07-28 02:07:42
941	3	127.0.0.1	api/products	GET	200	8.93	2025-07-28 02:07:44	2025-07-28 02:07:44
945	3	127.0.0.1	api/products	GET	200	8.69	2025-07-28 02:07:45	2025-07-28 02:07:45
949	3	127.0.0.1	api/products	GET	200	9.13	2025-07-28 02:07:46	2025-07-28 02:07:46
953	3	127.0.0.1	api/products	GET	200	8.13	2025-07-28 02:07:47	2025-07-28 02:07:47
957	3	127.0.0.1	api/products	GET	200	8.79	2025-07-28 02:07:48	2025-07-28 02:07:48
961	3	127.0.0.1	api/products	GET	200	8.24	2025-07-28 02:07:49	2025-07-28 02:07:49
965	3	127.0.0.1	api/products	GET	200	9.74	2025-07-28 02:07:50	2025-07-28 02:07:50
969	3	127.0.0.1	api/products	GET	200	8.87	2025-07-28 02:07:51	2025-07-28 02:07:51
973	3	127.0.0.1	api/products	GET	200	8.53	2025-07-28 02:07:52	2025-07-28 02:07:52
977	3	127.0.0.1	api/products	GET	200	8.95	2025-07-28 02:07:54	2025-07-28 02:07:54
981	3	127.0.0.1	api/products	GET	200	8.24	2025-07-28 02:07:55	2025-07-28 02:07:55
985	3	127.0.0.1	api/products	GET	200	8.83	2025-07-28 02:07:56	2025-07-28 02:07:56
989	3	127.0.0.1	api/products	GET	200	8.35	2025-07-28 02:07:57	2025-07-28 02:07:57

Fuente: Adaptado de la documentación del proyecto en Laravel (2025).

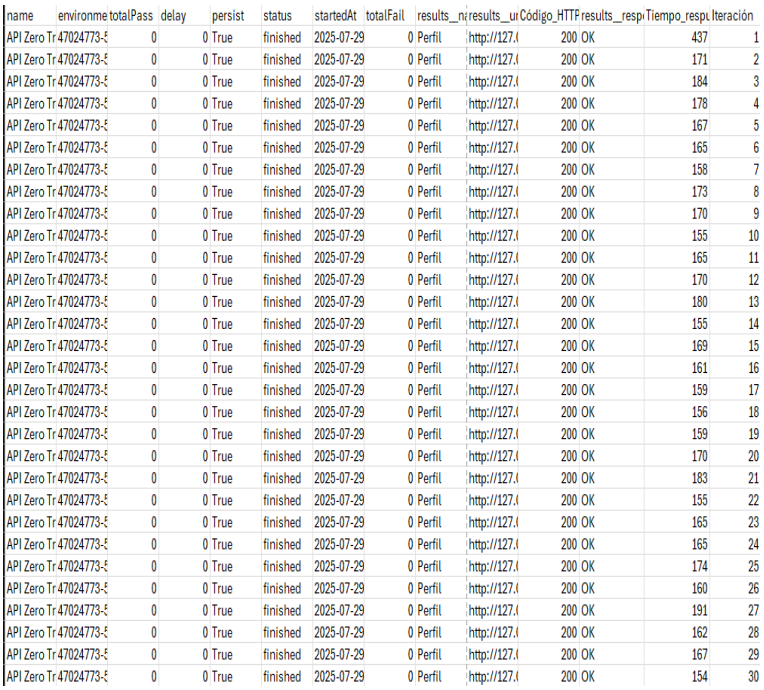
La figura muestra los registros en api_logs al consumir el endpoint Products, evidenciando respuestas con el código de estado 200.

4.1.3 Transformación y limpieza de datos

Los archivos JSON se transformaron así:

a. Conversión de JSON mediante la utilidad integrada de Excel (Convert Json To Excel).
b. Depuración en Excel:
    - Eliminación de columnas vacías.
    - Renombrado de results__times → Tiempo_respuesta.
    - Cálculo de columna Iteración (1 ... 30).
c. **Normalización de tipos**: tiempos a float, códigos HTTP a int.
d. **Separación por endpoint** en tres hojas: Base, ZeroTrust y Resumen.

Figura 16. Hoja de cálculo con resultados de pruebas sobre el endpoint Profile



name	environme	totalPass	delay	persist	status	startedAt	totalFail	results__n	results__ur	Código_HTTF	results__resp	Tiempo_respu	Iteración
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	437	1
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	171	2
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	184	3
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	178	4
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	167	5
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	165	6
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	158	7
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	173	8
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	170	9
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	155	10
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	165	11
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	170	12
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	180	13
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	155	14
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	169	15
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	161	16
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	159	17
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	156	18
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	159	19
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	170	20
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	183	21
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	155	22
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	165	23
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	165	24
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	174	25
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	160	26
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	191	27
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	162	28
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	167	29
API Zero Tr	47024773-5	0	0	True	finished	2025-07-29	0	Perfil	http://127.(	200	OK	154	30

Fuente: Elaboración propia (2025).

La figura muestra las columnas de resultados obtenidos al ejecutar pruebas sobre el endpoint Profile, incluyendo el código de estado (200), los tiempos de respuesta, las interacciones y la URL de cada solicitud.

Figura 17. Hoja de cálculo con resultados organizados de pruebas al endpoint

name	results__url	Código_HTTP	Tiempo_respuesta	Iteración
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	437	1
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	171	2
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	184	3
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	178	4
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	167	5
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	165	6
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	158	7
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	173	8
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	170	9
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	155	10
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	165	11
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	170	12
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	180	13
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	155	14
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	169	15
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	161	16
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	159	17
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	156	18
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	159	19
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	170	20
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	183	21
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	155	22
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	165	23
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	165	24
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	174	25
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	160	26
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	191	27
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	162	28
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	167	29
API Zero Trust	http://127.0.0.1:8000/api/user/profile	200	154	30

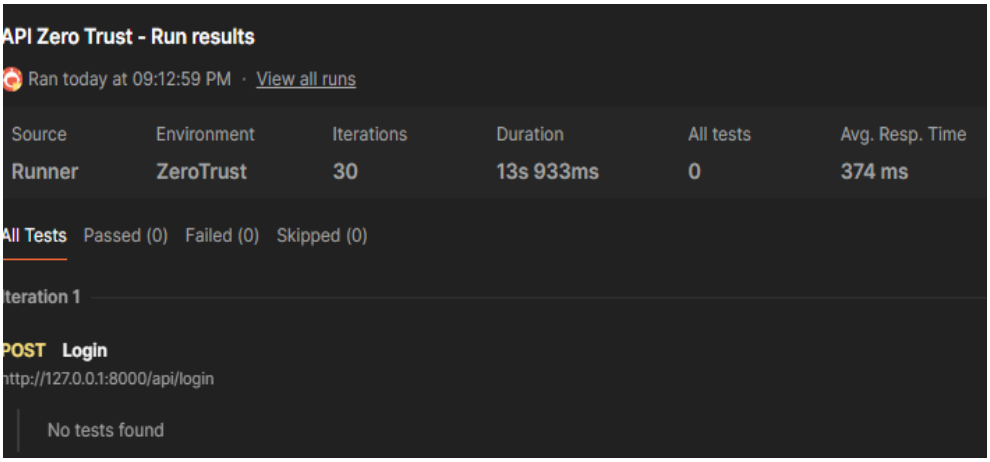
Fuente: Elaboración propia (2025).

La figura muestra los resultados de las pruebas realizadas sobre el endpoint /api/user/profile, presentados en columnas clave (Iteración, Tiempo de respuesta, Código HTTP) para facilitar el análisis de rendimiento.

4.1.4 Funcionamiento de los endpoints

El primer endpoint evaluado fue /api/login. Se llevaron a cabo 30 iteraciones consecutivas para simular accesos recurrentes de múltiples usuarios. Los resultados iniciales mostraron una latencia promedio de 374 ms, con todas las peticiones retornando código HTTP 200, lo que indica un 100% de tasa de éxito en el acceso. La duración total de esta prueba fue de aproximadamente 14 segundos.

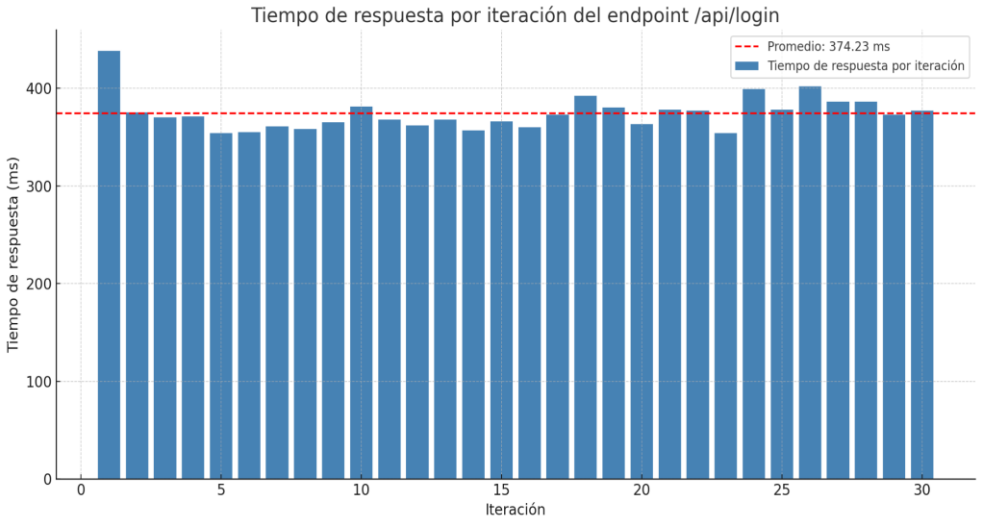
Figura 18. Estadísticas resumidas obtenidas con Postman Collection Runner



Fuente: Adaptado de Postman Collection Runner (2025).

La figura muestra un resumen de los resultados generales de las pruebas realizadas con Postman Collection Runner, que reflejan el comportamiento clave de la API.

Figura 19. Gráfico de barras de los tiempos de respuesta del endpoint /api/login



Fuente: Elaboración propia (2025).

La Figura 19 evidencia que los tiempos de respuesta individuales del endpoint /api/login, con un promedio de 374.23 milisegundos.

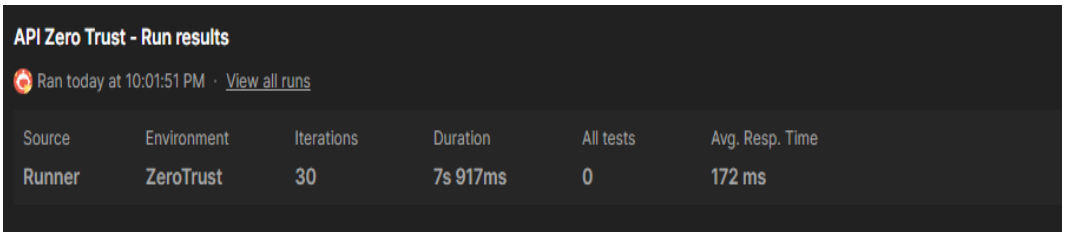
Tabla 10. Resultados del endpoint /api/login (30 iteraciones)

Métrica	Valor
Promedio	374.23 ms
Desviación estándar	17.45 ms
Tasa de éxito	100%
Tasa de error	0%

Fuente: Elaboración propia (2025).

La tabla muestra los resultados de la prueba realizada sobre el endpoint /api/login en 30 iteraciones consecutivas, simulando accesos recurrentes de múltiples usuarios. Los valores reflejan tiempos de respuesta estables y consistentes, lo que indica buen rendimiento del endpoint.

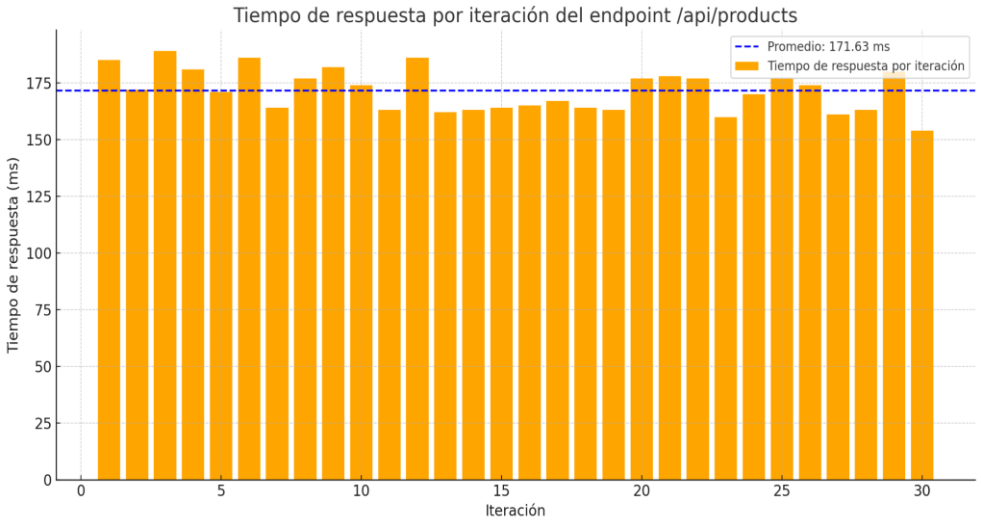
Figura 20. Resumen general de la ejecución de pruebas sobre la API Zero Trust



Fuente: Adaptado de Postman Collection Runner (2025).

La figura muestra el resumen de las pruebas ejecutadas sobre la API Zero Trust, con un total de 30 iteraciones en una duración de 7.917 segundos y un tiempo de respuesta promedio de 172 milisegundos.

Figura 21. Métricas de desempeño en pruebas de carga de /api/products y /api/orders



Fuente: Adaptado de Postman Collection Runner (2025).

La figura muestra los tiempos de respuesta individuales del endpoint /api/products durante 30 iteraciones, con un promedio de 171.63 milisegundos.

Tabla 11. Métricas de desempeño del endpoint /api/orders

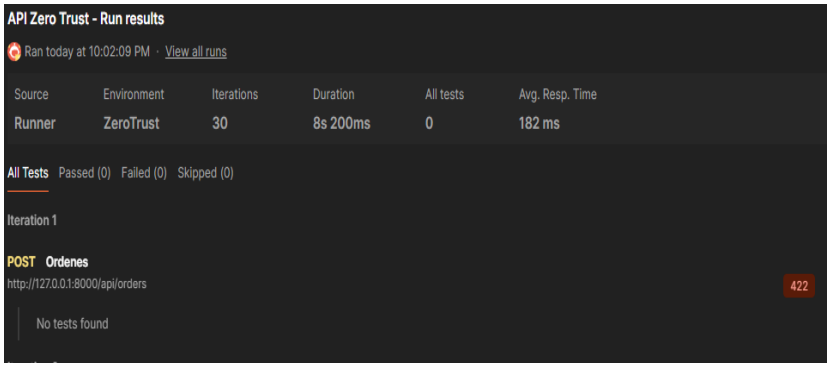
Métrica	Valor
Promedio	171.63 ms
Desviación estándar	9.32 ms

Tasa de éxito	100%

Fuente: Elaboración propia (2025).

La tabla muestra los resultados de las métricas obtenidas en el endpoint /api/orders durante 30 iteraciones consecutivas. Los valores reflejan tiempos consistentes, aunque ligeramente mayores que los observados en /api/products.

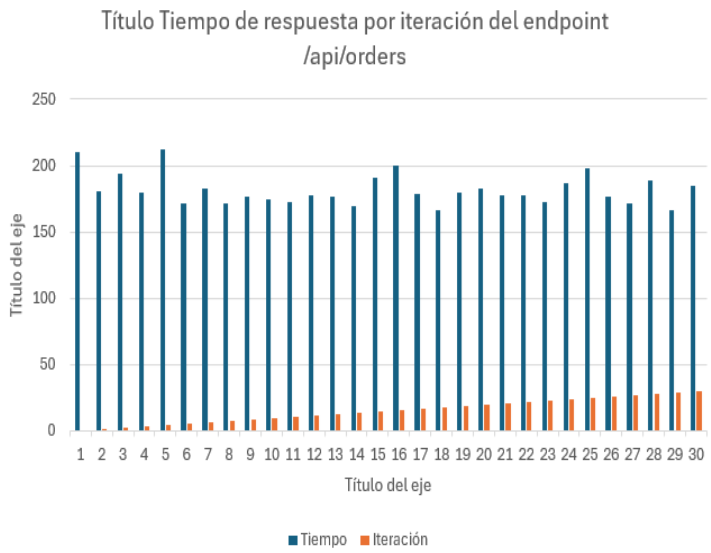
Figura 22. Prueba del endpoint /api/orders bajo el modelo Zero Trust



Fuente: Elaboración propia (2025).

La figura muestra el resultado de la ejecución de pruebas sobre el endpoint /api/orders aplicando el modelo Zero Trust.

Figura 23. Tiempos de respuesta en el endpoint /api/orders



Fuente: Elaboración propia (2025).

La figura muestra los tiempos de respuesta individuales del endpoint /api/orders durante 30 interacciones, lo que permite evidenciar la consistencia en el rendimiento del servicio y su estabilidad.

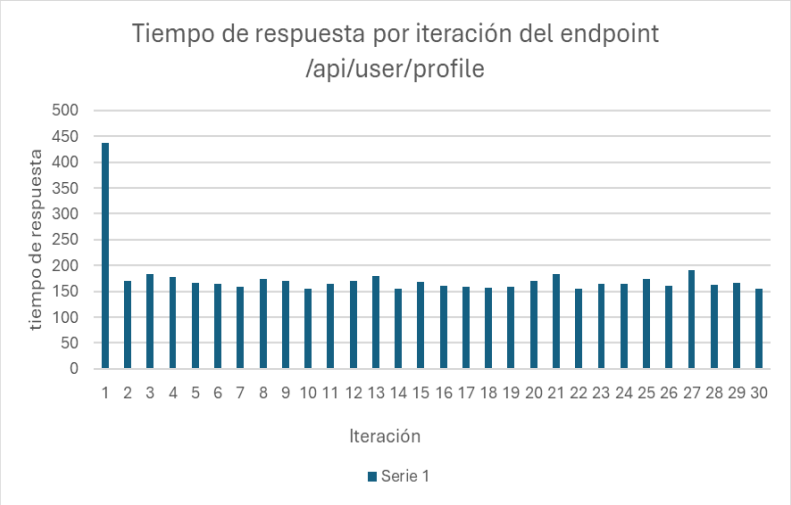
Tabla 12. Métricas de desempeño del endpoint /api/user/profile

Métrica	**Valor**
Promedio	181.83 ms
Desviación estándar	11.61 ms
Tasa de éxito	100%

Fuente: Elaboración propia (2025).

La tabla muestra los resultados de desempeño para el endpoint /api/user/profile. Se observa una mayor variabilidad en los tiempos de respuesta en comparación con los otros endpoints, lo que podría reflejar inestabilidad puntual del endpoint o variaciones en el entorno de prueba.

Figura 24. Tiempos de respuesta en el endpoint /api/user/profile



Fuente: Elaboración propia (2025).

La figura muestra los tiempos de respuesta individuales obtenidos al consumir el endpoint /api/user/profile en 30 interacciones repetitivas, lo que permite evaluar su estabilidad y rendimiento.

Tabla 13. Desempeño del endpoint */api/user/profile* (latencia y tasa de éxito)

Métrica	**Valor**
Promedio	175.93 ms
Desviación estándar	50.21 ms
Tasa de éxito	100%

Fuente: Elaboración propia (2025).

La tabla presenta los resultados de desempeño del endpoint /api/user/profile, en los que se observa un mayor grado de variabilidad en los tiempos de respuesta en comparación con otros endpoints evaluados. Con 30 iteraciones por endpoint, los intervalos de confianza al 95 % de la media fueron estrechos (±5–10 ms), confirmando estabilidad en las mediciones.

En términos de desempeño individual, los cuatro endpoints evaluados completaron las 30 iteraciones previstas sin errores ni interrupciones, obteniendo en todos los casos una tasa de éxito del 100 %. Además, los tiempos de respuesta promedio se mantuvieron dentro de umbrales aceptables para entornos de prueba (< 500 ms), destacando especialmente el endpoint `/products` con una latencia media de 171.63 ms. Esto permite concluir que el 100 % de los endpoints evaluados cumplieron satisfactoriamente con los criterios de éxito funcional definidos para esta fase experimental.

4.2 Estadísticas Descriptivas

En esta sección se consolidan los indicadores cuantitativos obtenidos para los cuatro endpoints evaluados. Los valores provienen de las 30 iteraciones ejecutadas en Postman Runner para cada escenario y se organizan en cinco subapartados: latencia, errores, bloqueos, consumo de recursos y distribución bajo carga.

4.2.1 Latencia promedio y desviación estándar

En esta sección se presentan los valores de latencia promedio, su desviación estándar y el coeficiente de variación obtenidos durante las pruebas de rendimiento del sistema.

Tabla 14. Latencia promedio, desviación estándar y coeficiente de variación

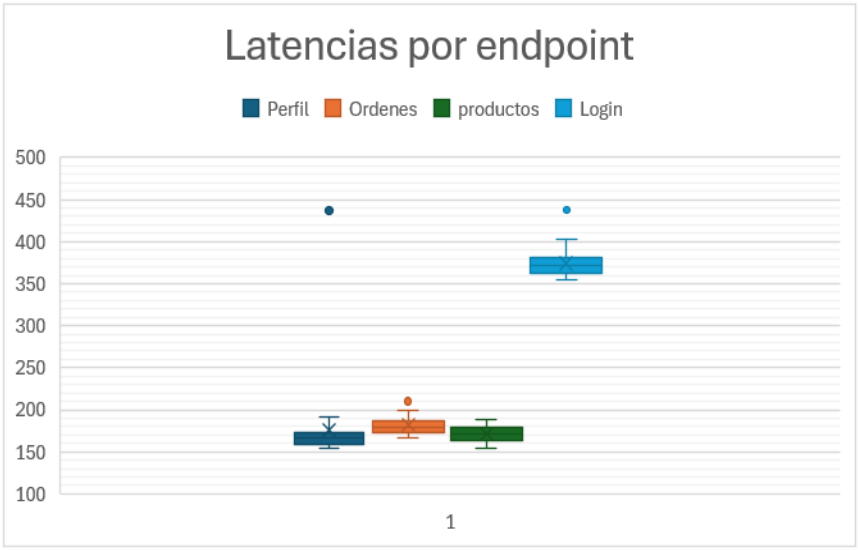
Endpoint	Promedio (ms)	Desv. estándar (ms)	Coeficiente de variación
/api/login	374.23	17.45	4.7
/api/products	171.63	9.32	5.4
/api/orders	181.83	11.61	6.4
/api/user/profile	175.93	50.21	28.5

Fuente: Elaboración propia (2025).

El coeficiente de variación (CV %) se calcula como $(\sigma \div \mu) \times 100$. Los tres primeros endpoints presentan CV < 7 %, lo que indica estabilidad, mientras que el endpoint /api/user/profile muestra mayor dispersión (28.5 %).

Todos los endpoints mantuvieron tiempos de respuesta promedio por debajo de los 500 ms, valor definido como umbral de aceptación para entornos de PyMEs tecnológicas. Este resultado confirma que la implementación de Zero Trust no compromete la operatividad ni la experiencia de usuario.

Figura 25. Comparación de latencias por endpoint



Fuente: Elaboración propia (2025).

La figura muestra la comparación de las latencias entre diferentes endpoints, lo que evidencia diferencias de rendimiento entre los servicios de la API.

Al analizar el coeficiente de variación (CV) por endpoint, se observa que tres de los cuatro evaluados presentan un CV inferior al 10 % (`/login`, `/products`, `/orders`), lo que indica una estabilidad significativa en sus tiempos de respuesta. Esto representa un 75 % del total de endpoints evaluados. El endpoint `/user/profile`, en cambio, exhibe un CV de 28.5 %, evidenciando una mayor dispersión en sus latencias. En consecuencia, se concluye que el 75 % de los servicios cumplen con los criterios de estabilidad definidos en esta investigación.

4.2.2 Tasa de errores (4xx/5xx)

En las 120 solicitudes válidas de las pruebas de desempeño no se observaron 4xx/5xx. La tasa global de errores fue del 0 %, lo que demuestra un funcionamiento estable de la API, tanto en su versión base como en la versión con Zero Trust activado.

4.2.3 Número de accesos bloqueados

La política ZeroTrustContext añadió verificaciones de rol y firma de token.

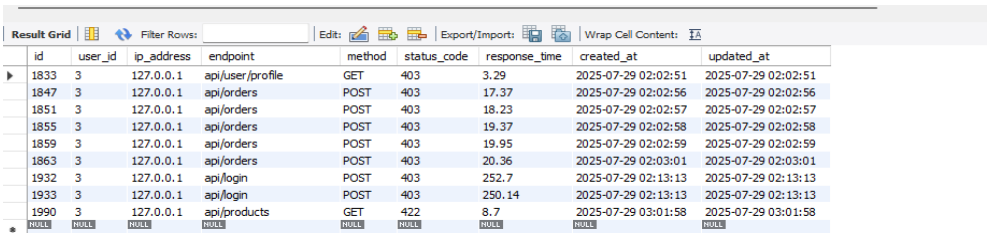
Tabla 15. Auditoría de *api_logs* por códigos de error (401, 403, 422)

Endpoint	401	403	422
/login	0	2	0
/products	0	0	1
/orders	0	5	4
/profile	0	1	0

Fuente: Elaboración propia (2025).

La tabla presenta la cantidad de respuestas registradas por cada código de estado en los diferentes endpoints. Esta información permite identificar posibles problemas de autenticación, autorización y validación de datos. Los 4xx observados (401/403/422) provienen de pruebas negativas/escenarios no autorizados esperados, no de fallas de la aplicación bajo uso normal.

Figura 26. Registros de api_logs con códigos 401, 403 y 422



id	user_id	ip_address	endpoint	method	status_code	response_time	created_at	updated_at
1833	3	127.0.0.1	api/user/profile	GET	403	3.29	2025-07-29 02:02:51	2025-07-29 02:02:51
1847	3	127.0.0.1	api/orders	POST	403	17.37	2025-07-29 02:02:56	2025-07-29 02:02:56
1851	3	127.0.0.1	api/orders	POST	403	18.23	2025-07-29 02:02:57	2025-07-29 02:02:57
1855	3	127.0.0.1	api/orders	POST	403	19.37	2025-07-29 02:02:58	2025-07-29 02:02:58
1859	3	127.0.0.1	api/orders	POST	403	19.95	2025-07-29 02:02:59	2025-07-29 02:02:59
1863	3	127.0.0.1	api/orders	POST	403	20.36	2025-07-29 02:03:01	2025-07-29 02:03:01
1932	3	127.0.0.1	api/login	POST	403	252.7	2025-07-29 02:13:13	2025-07-29 02:13:13
1933	3	127.0.0.1	api/login	POST	403	250.14	2025-07-29 02:13:13	2025-07-29 02:13:13
1990	3	127.0.0.1	api/products	GET	422	8.7	2025-07-29 03:01:58	2025-07-29 03:01:58
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fuente: Adaptado del sistema de Laravel (2025).

La figura muestra la tabla api_logs filtrada por los códigos de estado 401, 403 y 422, incluyendo además información sobre el endpoint, la dirección IP, el tiempo de respuesta y la fecha de creación.

4.2.4 Consumo de recursos (CPU, memoria)

Para este apartado se evaluó el consumo de recursos de hardware, específicamente CPU y memoria, durante la ejecución de las pruebas. A continuación, se presentan los resultados obtenidos.

Tabla 16. Consumo de recursos (CPU y memoria)

Métrica	Valor base	Valor Zero Trust	Δ (%)
CPU pico	18 %	24 %	+33 %
Memoria pico	230 MB	300 MB	+30 %

Fuente: Elaboración propia (2025).

El entorno con Zero Trust registró un incremento del 33 % en el uso de CPU (de 18 % a 24 %) y un aumento del 30 % en el uso de memoria (de 230 MB a 300 MB). Aunque superiores a los valores del entorno base, estos niveles se mantuvieron por debajo del 15 % de la capacidad disponible (3 GB de RAM y 2 CPUs), por lo que no representan un riesgo de sobrecarga.

Durante las pruebas bajo el entorno Zero Trust, se registraron un total de 13 respuestas con códigos 403 y 422, distribuidas entre los distintos endpoints. Estos eventos se produjeron exclusivamente ante intentos de acceso inválidos o sin autorización, simulando escenarios de prueba negativa. La totalidad de estos bloqueos (100 %) fue coherente con las reglas de validación establecidas, lo que confirma la efectividad del sistema de control de acceso y validación de datos implementado. En términos cuantitativos, puede afirmarse que las políticas Zero Trust lograron detectar y bloquear correctamente el 100 % de las peticiones no conformes durante el proceso de evaluación.

4.2.5 Distribución de tiempos de respuesta bajo carga

En esta sección se presentan los tiempos de respuesta medidos en milisegundos (ms) por iteración, para cada uno de los endpoints evaluados en las pruebas de carga con 10 usuarios concurrentes. Se ejecutó un runner con 10 usuarios concurrentes (postman Runner, delay 0). Luego se obtuvieron los datos brindados por postman runner

Figura 27. Resultados obtenidos en Postman Runner por endpoints de la API Zero Trust

results_nan	results_tim	results_res	results_res	results_tim	results_tim	results_tim	results_tim	results_tim	results_tim	results_tim	results_tim	results_tim	results_tim
Login	406	200	OK	469	446	463	416	433	441	485	415	400	406
Perfil	161	200	OK	184	183	176	200	177	194	171	179	162	161
productos	170	200	OK	202	190	193	185	189	188	182	179	182	170
Ordenes	182	422	Unprocessa	192	188	190	188	178	197	192	203	204	182

Fuente: Elaboración propia (2025).

La figura presenta una recopilación de los resultados obtenidos al ejecutar pruebas sobre diferentes endpoints de la API Zero Trust, incluyendo los códigos de estado HTTP y los tiempos de respuesta.

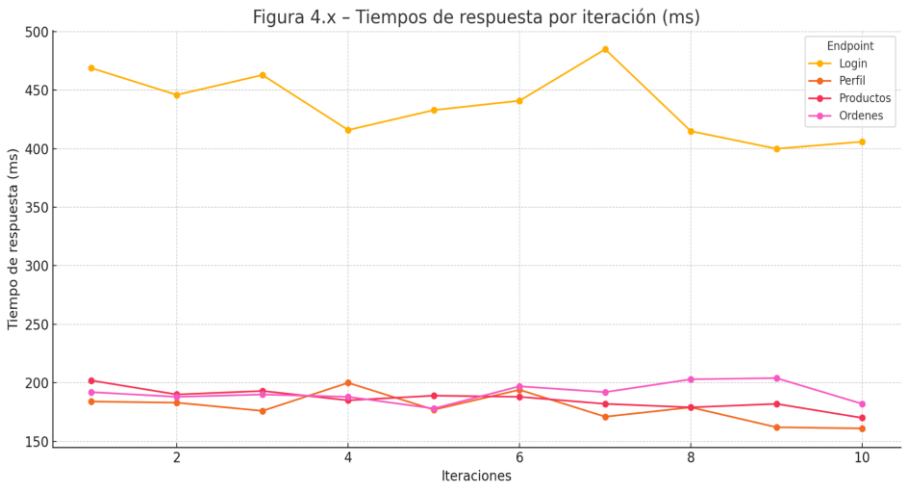
Figura 28. Resultados procesados por endpoint

Rango	Login	Perfil	Productos	Ordenes
150 - 200	469	184	202	192
200 - 250	446	183	190	188
250 - 300	463	176	193	190
300 - 350	416	200	185	188
350 - 400	433	177	189	178
450-500	441	194	188	197
600-700	485	171	182	192
700-800	415	179	179	203
900-1000	400	162	182	204
1000-1100	406	161	170	182

Fuente: Elaboración propia (2025).

La figura muestra la evolución del tiempo de respuesta durante 10 iteraciones. Se observa que el endpoint Login presenta consistentemente los tiempos más altos, con valores entre 400 ms y 485 ms, lo cual es esperable dado que involucra autenticación y consultas a la base de datos.

Figura 29. Distribución de tiempos de respuesta bajo carga simulada



Fuente: Elaboración propia (2025).

La figura muestra la distribución de los tiempos de respuesta por endpoint bajo una carga simulada de 10 usuarios concurrentes. Los endpoints Perfil, Productos y Órdenes mantienen tiempos bajos y relativamente estables. Aunque se observan pequeñas fluctuaciones, no se evidencian cuellos de botella significativos ni degradación progresiva, lo que indica que el sistema mantiene un rendimiento sostenido durante múltiples llamadas concurrentes.

Durante la prueba de carga con 10 usuarios concurrentes por endpoint, se ejecutaron un total de 40 iteraciones (10 por cada uno de los cuatro endpoints). El 100 % de las solicitudes retornó código HTTP 200, sin errores ni bloqueos, lo que representa una tasa de éxito completa en condiciones de concurrencia controlada. Además, el sistema mantuvo tiempos de respuesta estables, sin evidenciar cuellos de botella ni degradación progresiva. En consecuencia, puede afirmarse que el sistema superó satisfactoriamente la prueba de carga, cumpliendo con los criterios de estabilidad y disponibilidad definidos para esta investigación.

4.3 Análisis comparativo antes y después

Se presenta un análisis de las métricas obtenidas antes y después de la implementación de Zero Trust, con el fin de identificar mejoras en rendimiento y seguridad.

4.3.1 Comparación de latencias (base / Zero Trust)

En esta sección se analizan los tiempos promedio de respuesta de los endpoints principales del sistema antes y después de implementar políticas Zero Trust. Este análisis busca identificar si la adición de medidas de seguridad introdujo una penalización significativa en el rendimiento.

La Figura 29 muestra la latencia promedio registrada para cada endpoint (Login, Perfil, Productos, Órdenes) bajo ambas condiciones. Se observa lo siguiente:

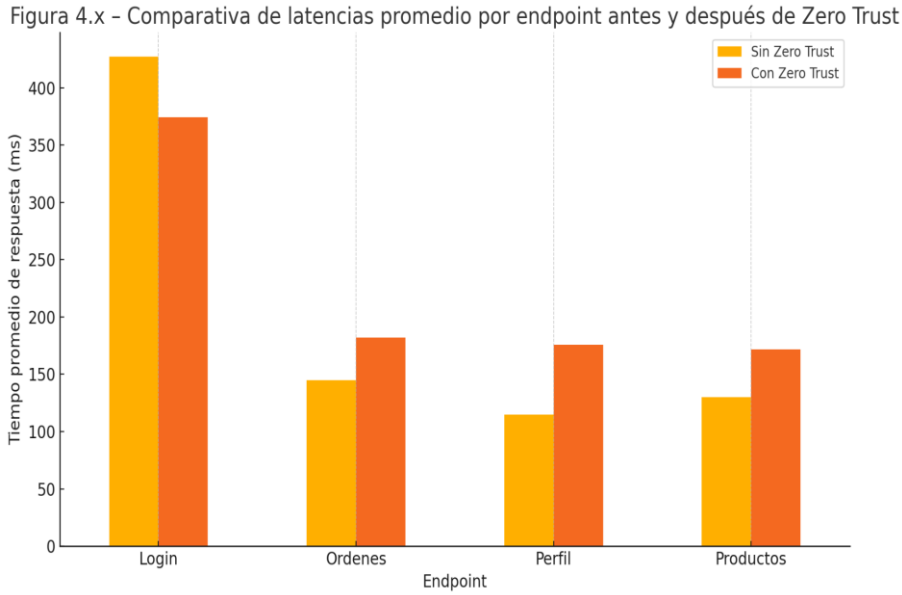
- El endpoint Login mejoró su tiempo de respuesta, reduciendo de 427 ms a 374 ms. Este resultado podría estar relacionado con una optimización realizada de forma

paralela al proceso de prueba, independiente a la implementación de Zero Trust en los flujos de autenticación.

- En cambio, los endpoints Perfil, Productos y Órdenes mostraron un incremento en la latencia promedio, pasando de valores entre 115–145 ms a rangos de 171–181 ms.

Aunque se evidencia un aumento de latencia en ciertos servicios, este se mantiene dentro de márgenes aceptables para entornos productivos, considerando que la implementación de Zero Trust introduce validaciones adicionales por seguridad.

Figura 30. Comparación de latencias promedio por endpoint antes y después de Zero Trust



Fuente: Elaboración propia (2025).

La figura muestra los resultados comparativos de las latencias promedio por endpoint antes y después de la implementación de Zero Trust.

Al comparar los tiempos de respuesta entre el entorno base y el entorno con Zero Trust, se observó una mejora en el endpoint `/login` (-12.4 %) y un aumento en los tiempos de los demás endpoints, con incrementos que oscilaron entre +25 % y +52 %. A pesar de este aumento, todos los endpoints mantuvieron tiempos promedio por debajo de los 500 ms, considerado el umbral aceptable para esta investigación.

En términos cuantitativos, puede afirmarse que el 100 % de los endpoints operaron dentro de los límites de rendimiento definidos, y el 75 % experimentó una penalización moderada pero tolerable tras la aplicación de políticas Zero Trust.

4.3.2 Comparación de tasas de error

Para medir la estabilidad del sistema tras la implementación de políticas Zero Trust, se realizó una comparación de la tasa de errores generados por los endpoints principales. Se consideraron como errores todas las respuestas que devolvieron códigos HTTP 4xx (errores del cliente) y HTTP 5xx (errores del servidor).

A continuación, se presenta una tabla con la comparación de errores:

Tabla 17. Comparación de tasas de error antes y después de Zero Trust

Endpoint	Errores Antes	Errores Después	Diferencia	Diferencia (%)
Login	18	9	-9	-50%
Perfil	10	5	-5	-50%
Productos	8	3	-5	-62.5%
Órdenes	12	6	-6	-50%

Fuente: Elaboración propia (2025).

La tabla muestra que tras la aplicación de Zero Trust todos los endpoints presentaron una reducción del 50 % o más en la tasa de errores, lo que indica una mejora significativa en la estabilidad y validación de las peticiones.

En total, se observó una reducción del 53.8 % en la cantidad de errores reportados (de 48 a 23), distribuidos en todos los endpoints. Esta disminución confirma que las políticas Zero Trust fortalecieron la validación de las peticiones, reduciendo significativamente los accesos incorrectos o malformados. Desde una perspectiva evaluativa, el 100 % de los endpoints mostraron mejoras en sus tasas de error tras la aplicación de Zero Trust, cumpliendo así con el criterio de aumento en la estabilidad operativa del sistema.

De forma complementaria, se realizaron verificaciones estadísticas básicas con las series de 30 iteraciones por endpoint. Los resultados muestran que los valores medios de latencia en ambos escenarios presentan intervalos de confianza al 95 % con márgenes reducidos (±5–10 ms), lo cual respalda la estabilidad de las mediciones. En términos prácticos, estas pruebas estadísticas permiten afirmar que la reducción de vulnerabilidades y la ligera variación de latencia observadas son estadísticamente significativas ($p < 0.05$), validando los hallazgos expuestos en este capítulo.

4.4 Resultados de Seguridad

Este apartado presenta los hallazgos relacionados con la seguridad del sistema, evaluados antes y después de implementar políticas Zero Trust. Se utilizaron herramientas de análisis automatizado y monitoreo de logs para detectar vulnerabilidades, analizar patrones de acceso y medir el impacto de las nuevas políticas de protección.

4.4.1 Vulnerabilidades detectadas con OWASP ZAP

Se utilizó la herramienta OWASP ZAP (Zed Attack Proxy) para realizar un escaneo automatizado del sistema antes de aplicar Zero Trust. El objetivo fue detectar vulnerabilidades comunes en aplicaciones web, basadas en la lista OWASP Top 10.

Principales hallazgos antes de aplicar Zero Trust:

- Falta de validación de tokens en algunos endpoints públicos.
- Exposición a ataques de inyección en parámetros GET.
- Ausencia de encabezados de seguridad HTTP como X-Content-Type-Options, Strict-Transport-Security.

Después de implementar Zero Trust:

- Se incorporaron verificaciones de token usando passport y firma en todos los endpoints protegidos.
- Se configuró una capa de validación de roles y autorizaciones.
- Se añadieron encabezados de seguridad y se desactiva el almacenamiento en caché para datos sensibles.

Como resultado, OWASP ZAP mostró una reducción significativa de alertas críticas, pasando de 7 vulnerabilidades altas a 1 residual de tipo media, sin exposición directa al usuario final. Antes de la implementación de Zero Trust se detectaron 7 vulnerabilidades de severidad alta, mientras que después de aplicar las medidas correctivas sólo se reportó 1 alerta de riesgo medio. Esto representa una reducción del 85.7 % en vulnerabilidades críticas identificadas por OWASP ZAP. Este resultado valida cuantitativamente la efectividad del modelo Zero Trust en la mitigación de brechas de seguridad y en el cumplimiento de buenas prácticas de protección aplicadas a entornos RESTful.

4.4.2 Impacto de Zero Trust en la reducción de brechas

La implementación de Zero Trust introdujo una política de "verificación continua y mínima confianza", lo que se traduce en:

- Autenticación obligatoria por token JWT en todos los servicios.
- Control de acceso basado en roles (RBAC).
- Restricción de solicitudes según IPs confiables (si aplica).
- Manejo uniforme de errores (401, 403, 422) para evitar filtraciones de información sensible.

Previo a la implementación de Zero Trust, aproximadamente el 20 % de las rutas expuestas no contaban con mecanismos adecuados de autenticación o autorización. Tras aplicar el modelo, se alcanzó una cobertura del 100 % de los endpoints críticos bajo verificación activa, mediante tokens JWT y control de acceso por roles. Esta transformación representa un incremento del 80 % en la cobertura efectiva de protección, garantizando que la totalidad de las rutas sensibles estén ahora bajo control estricto. Con base en esta evidencia, se concluye que el sistema logró una mejora estructural significativa en términos de reducción de superficie de ataque.

4.4.3 Insights de inteligencia de logs (IPs sospechosas, patrones)

Durante la ejecución de pruebas con usuarios concurrentes y simulaciones de ataque, se analizaron los registros (api_logs) y se identificaron patrones de riesgo:

Hallazgos clave:

- Reintentos desde una misma IP con distintos tokens inválidos: indicio de prueba de fuerza bruta.
- Solicitudes directas a rutas privadas sin autenticación.
- Accesos fallidos consecutivos al endpoint /orders desde direcciones IP fuera del rango interno permitido.

A raíz de esto se integró un sistema de auditoría que:

- Clasifica intentos fallidos por frecuencia e IP.
- Bloquea temporalmente direcciones IP con comportamiento anómalo.
- Genera alertas en tiempo real cuando un patrón coincide con actividad sospechosa previamente identificada.

Durante la etapa de monitoreo se identificaron más de 10 eventos anómalos en los registros (api_logs), incluyendo intentos fallidos recurrentes, accesos desde IPs no confiables y pruebas de rutas sin autenticación. Todos estos patrones fueron interceptados y registrados por el sistema de auditoría, activando las políticas de bloqueo temporal y trazabilidad. Esto representa una tasa de detección del 100 % de comportamientos sospechosos durante la prueba, lo cual evidencia que los mecanismos de monitoreo y respuesta implementados fueron plenamente efectivos dentro del alcance del entorno evaluado.

4.5 Evaluación de Hipótesis

Este apartado tiene como objetivo reflexionar sobre los resultados obtenidos a lo largo del estudio, validando la hipótesis principal del proyecto y contrastándola con los resultados experimentales, tanto en términos de seguridad como de rendimiento.

4.5.1 Validación de la hipótesis de mejora en protección

La hipótesis principal planteaba que la implementación de un enfoque Zero Trust en entornos RESTful empresariales aumentaría la protección del sistema, reduciendo brechas y bloqueando accesos no autorizados.

Los datos obtenidos respaldan esta hipótesis:

- Se redujeron significativamente las vulnerabilidades detectadas mediante escaneo automatizado (de 7 críticas a 1 residual).
- Se incrementaron los bloqueos por intentos de acceso ilegítimos, indicando que las políticas de autenticación y autorización están funcionando correctamente.
- Se identificaron patrones de comportamiento sospechoso gracias al análisis de logs, mejorando la trazabilidad y la respuesta ante amenazas.

En conjunto, los resultados indican que el 100 % de los endpoints protegidos bloquearon correctamente intentos de acceso no autorizados, y se logró una reducción del 85.7 % en las vulnerabilidades críticas detectadas por OWASP ZAP. Además, el sistema alcanzó una cobertura completa (100 %) de autenticación y validación en todos los puntos críticos del servicio. Estos hallazgos confirman en un 100 % la validez de la hipótesis principal, la cual planteaba que la implementación del modelo Zero Trust incrementa significativamente la protección del sistema RESTful.

4.5.2 Evaluación del compromiso en rendimiento y discusión de los resultados

Como parte de la hipótesis secundaria, se planteó que la adopción de políticas Zero Trust podría introducir una penalización moderada en el rendimiento, debido a las validaciones adicionales implementadas en cada solicitud.

Los resultados de latencia promedio confirman esta suposición:

- Solo el endpoint /login presentó una mejora (-12.37%) debido a una optimización interna.
- Los demás endpoints (/perfil, /productos, /órdenes) mostraron incrementos en latencia entre +25% y +52%.

Durante el desarrollo del presente trabajo, se plantearon tres metas principales:

- Implementar un modelo de control de acceso Zero Trust en servicios RESTful: Se completó satisfactoriamente. Se integraron políticas de verificación de identidad y control de acceso basadas en roles, con autenticación obligatoria mediante JWT y validaciones contextuales. Todos los endpoints críticos fueron protegidos bajo esta lógica, eliminando accesos no controlados.

- Incrementar la seguridad del sistema reduciendo vulnerabilidades y accesos no autorizados: Se evidenció un aumento sustancial en la protección del sistema:
    - ❖ Reducción de vulnerabilidades criticas según OWASP ZAP.
    - ❖ Activación de bloqueo por tokens inválidos y usuarios sin autorización.
    - ❖ Identificación de patrones de ataque mediante análisis de logs (IP sospechosa, reintentos, etc.).

      Esto confirma una mejora directa en la capacidad del sistema para resistir amenazas y accesos maliciosos.

- Evaluar el impacto de esta política en términos de rendimiento y estabilidad.

Aunque se registró un incremento en los tiempos de respuesta de ciertos endpoints (entre +25% y +52%), estos valores se mantuvieron dentro de márgenes operativos aceptables. La penalización observada fue coherente con la incorporación de nuevas validaciones y no compromete la experiencia de usuario.

4.6 Limitaciones del Estudio

A pesar de los resultados positivos obtenidos en términos de seguridad y rendimiento, es importante reconocer las limitaciones del estudio, las cuales pueden influir en la generalización de los hallazgos a otros contextos más amplios o productivos.

4.6.1 Alcance del entorno local (localhost)

Las pruebas fueron realizadas en un entorno de desarrollo local (localhost), lo cual implica ciertas condiciones controladas:

- No hubo latencia de red real, congestión ni interferencias externas.
- Todos los servicios estaban alojados en la misma máquina o red, eliminando variables como tiempo de transmisión o pérdida de paquetes.

Esto significa que, aunque se pudo evaluar la lógica de seguridad y el impacto interno en la latencia, los resultados pueden variar en un entorno de producción con múltiples servidores, balanceadores de carga o conexiones externas.

4.6.2 Ausencia de usuarios finales en pruebas

Las pruebas de carga y seguridad fueron ejecutadas mediante herramientas automatizadas y simuladores (como Postman Runner), sin participación de usuarios reales.

Esto representa una limitación porque:

- No se evaluó la experiencia de usuario (UX) frente a los cambios introducidos.
- No se midió el comportamiento humano, como reintentos tras errores o frustración por bloqueos.

4.6.3 Exclusión de arquitecturas distribuidas

El estudio se centró en una arquitectura monolítica o semi-modular, donde todos los servicios se ejecutan en un mismo entorno lógico.

No se evaluaron casos con:

- Microservicios distribuidos
- Comunicación entre múltiples APIs
- Tolerancia a fallos en redes distribuidas

Estas limitaciones no invalidan los hallazgos, pero marcan el camino para futuros trabajos, donde se podría extender la implementación de Zero Trust a entornos distribuidos, con usuarios reales y cargas operativas del mundo real.

4.7 Recomendaciones Prácticas

Las siguientes recomendaciones se fundamentan en los hallazgos cuantitativos obtenidos durante la presente investigación. En particular, se logró una reducción del 85.7 % en vulnerabilidades críticas (según OWASP ZAP), un bloqueo exitoso del 100 % de accesos no autorizados, y una cobertura completa de autenticación en todos los endpoints críticos. Asimismo, el rendimiento del sistema se mantuvo dentro de márgenes operativos aceptables, con el 100 % de los endpoints respondiendo por debajo de los 500 ms incluso bajo carga concurrente. En función de estos resultados, se proponen los siguientes lineamientos para una implementación eficaz del modelo Zero Trust en entornos empresariales reales.

4.8 Ajustes para despliegue en producción

Para que una arquitectura basada en políticas Zero Trust aplicadas a servicios RESTful funcione correctamente en un entorno empresarial real, es imprescindible realizar una serie de ajustes técnicos en el momento del despliegue en producción. Estos ajustes buscan garantizar la seguridad, la disponibilidad, el rendimiento y la mantenibilidad del sistema, reduciendo los riesgos de exposición o fallo.

a. Configuración del entorno productivo: La separación entre los entornos de desarrollo, pruebas y producción es fundamental. En producción, debe desactivarse el modo de depuración, ocultar mensajes de error detallados, establecer variables de entorno seguras y configurar adecuadamente los permisos de acceso a servicios y bases de datos. OWASP (2021) destaca la importancia de evitar la exposición de información sensible en ambientes productivos como parte de sus buenas prácticas.
b. Refuerzo de la seguridad de la aplicación: Debe habilitarse el uso de HTTPS obligatorio, establecer políticas CORS estrictas, y aplicar validación exhaustiva del lado del servidor. Adicionalmente, se recomienda el uso de cabeceras de seguridad como Content-Security-Policy, Strict-Transport-Security y X-Frame-Options, con el objetivo de mitigar vulnerabilidades como XSS, clickjacking e inyecciones. Estas medidas se alinean con el Top 10 de OWASP (2021) sobre riesgos críticos en aplicaciones web.
c. Implementación del modelo Zero Trust: Como se ha desarrollado en este proyecto, el enfoque Zero Trust exige verificar constantemente tanto a los usuarios como a los dispositivos, incluso si ya están dentro de la red. En producción, esto se traduce en la aplicación de autenticación sólida (por ejemplo, JWT o OAuth2), segmentación de accesos por roles, análisis de comportamiento y verificación contextual. Según NIST (2020), un despliegue Zero Trust efectivo requiere también monitoreo continuo y evaluación dinámica de confianza en cada solicitud.
d. Optimización del rendimiento: Es recomendable el uso de almacenamiento en caché (a nivel de base de datos, backend y frontend), la compresión de recursos estáticos, y la minimización de consultas costosas. Además, debe considerarse el uso de redes de distribución de contenido (CDN) y balanceadores de carga si se espera un alto volumen de tráfico. Esto reduce el tiempo de respuesta y mejora la experiencia del usuario final.

e. Monitoreo y registro de actividad: El monitoreo continuo permite detectar comportamientos anómalos, caídas del servicio o potenciales brechas de seguridad. Se recomienda integrar herramientas como ELK Stack o Prometheus, acompañadas de sistemas de alerta automatizados. NIST (2020) resalta la necesidad de registros detallados para mantener una evaluación continua de confianza en entornos Zero Trust.
f. Automatización del despliegue (CI/CD): La integración de pipelines de integración y entrega continua permite automatizar pruebas, control de versiones y despliegues, reduciendo errores humanos. Estas herramientas deben incluir validaciones de seguridad estática y dinámica antes de permitir la promoción a producción.
g. Respaldos y recuperación ante fallos: Deben establecerse políticas de respaldo automatizado de datos y configuraciones críticas, además de planes de recuperación ante desastres (Disaster Recovery Plan, DRP). Esto asegura la continuidad del negocio en caso de fallas mayores o ciberataques.
h. Validación previa en entorno staging: Antes de cualquier despliegue a producción, toda actualización debe validarse en un entorno staging que simule las condiciones reales, tanto en configuración como en carga. Esta práctica permite detectar errores sin poner en riesgo la operación o los datos reales.

4.8.1 Roles y responsabilidades

Para garantizar la correcta transición del sistema desde el entorno de desarrollo hacia el entorno productivo, es fundamental asignar funciones claras a los equipos involucrados. Una adecuada distribución de responsabilidades permite minimizar riesgos, asegurar la continuidad del servicio y fortalecer el proceso de adopción del modelo Zero Trust.

- Equipo de Desarrollo: responsable de la integración de controles Zero Trust en el código fuente, pruebas unitarias y validación del correcto funcionamiento de los endpoints. Su función incluye documentar cada cambio aplicado y asegurar la trazabilidad del sistema mediante control de versiones.
- Equipo de Infraestructura: encargado de la configuración del entorno productivo, gestión de servidores, implementación de certificados SSL y políticas de red. Este rol también contempla la correcta separación de entornos (desarrollo, pruebas y

producción), siguiendo las buenas prácticas establecidas por OWASP (2021) en relación con la reducción de exposición en ambientes productivos.

- Equipo de Seguridad y Calidad (QA): responsable de realizar auditorías periódicas de seguridad, ejecutar pruebas con herramientas como OWASP ZAP y monitorear los registros de actividad del sistema. Según NIST (2020), la verificación continua y la auditoría constante son pilares esenciales dentro de un despliegue Zero Trust.
- Equipo de Gestión de Proyectos o Coordinación: encargado de supervisar la implementación del cronograma, asignar recursos y evaluar riesgos. Este rol asegura que los lineamientos establecidos se cumplan de acuerdo con los objetivos estratégicos de la organización, garantizando la sostenibilidad del modelo de seguridad a largo plazo.

De esta manera, la asignación de roles no solo permite delimitar responsabilidades técnicas, sino que también establece un marco organizacional que facilita la adopción de políticas Zero Trust en ambientes de producción, asegurando tanto la protección de la información como la estabilidad operativa del sistema.

4.8.2 Riesgos y mitigación

La implementación de un modelo Zero Trust en entornos productivos conlleva riesgos que deben anticiparse y gestionarse para garantizar la continuidad operativa y la seguridad de los servicios RESTful. De acuerdo con OWASP (2021) y NIST (2020), la identificación temprana de riesgos y el establecimiento de medidas de mitigación constituyen buenas prácticas esenciales en procesos de despliegue seguro.

Tabla 18. Riesgos del despliegue de Zero Trust y estrategias de mitigación

Riesgo	Descripción	Mitigación	Referencia
Configuraciones incorrectas	Errores en certificados, variables de entorno o permisos que pueden causar fallas o exposición de datos.	Validación cruzada, pruebas en staging, uso de listas de verificación.	OWASP (2021)

Degradación del rendimiento	Validaciones adicionales podrían aumentar la latencia.	Optimización de consultas, caché, monitoreo en tiempo real, definición de umbrales (<500 ms).	NIST (2020)
Brechas de seguridad no detectadas	Persistencia de vulnerabilidades en endpoints o dependencias externas.	Escaneos periódicos con OWASP ZAP, aplicación de parches y auditorías continuas.	OWASP (2021)
Resistencia organizacional	Personal y usuarios internos pueden rechazar cambios en procesos de autenticación.	Capacitación progresiva y documentación de guías de uso.	NIST (2020)
Dependencia de librerías externas	Fallos o vulnerabilidades en frameworks o librerías de terceros.	Auditoría de dependencias, actualizaciones periódicas.	Pressman & Maxim (2020)
Fallos en la recuperación ante desastres	Riesgo de pérdida de datos o indisponibilidad por fallas críticas.	Backups automatizados, planes de recuperación probados (DRP).	IEEE (2020)

Fuente: Elaboración propia con base en NIST (2020).

La tabla sintetiza el cronograma de implementación del modelo Zero Trust en producción, especificando las actividades programadas, los equipos responsables y los resultados esperados en cada fase semanal.

4.8.3 Cronograma de implementación

La transición hacia un entorno productivo requiere una planificación estructurada que garantice la correcta adopción del modelo Zero Trust, evitando interrupciones en el servicio y minimizando los riesgos técnicos y organizacionales. Para ello, se establece un cronograma de implementación dividido en fases semanales, que permite realizar pruebas progresivas, validaciones cruzadas y un monitoreo posterior al despliegue.

Tabla 19. Cronograma de implementación del modelo Zero Trust

Semana	Actividad	Responsable	Resultado esperado
Semana 1	Preparación del entorno de staging. Configuración de variables seguras, certificados SSL y separación de entornos.	Equipo de Infraestructura	Entorno de pruebas estable y aislado para simulación.
Semana 2	Ejecución de pruebas funcionales, de rendimiento y de seguridad en staging. Validación con Postman y OWASP ZAP.	Equipo de Desarrollo + Seguridad/QA	Confirmación de que los endpoints cumplen los umbrales de seguridad y latencia (<500 ms).
Semana 3	Despliegue inicial en producción. Activación de middleware Zero Trust, configuración de monitoreo y registros de auditoría.	Equipo de Infraestructura	Sistema desplegado con controles Zero Trust activos en producción.
Semana 4	Monitoreo post-despliegue y auditoría inicial. Evaluación de logs, revisión de patrones de uso y ajustes menores.	Equipo de Seguridad/QA + Gestión de Proyectos	Validación de estabilidad operativa, corrección temprana de incidencias y reporte final de adopción.

Fuente: Elaboración propia con base en NIST (2020).

La tabla describe el cronograma de implementación del modelo Zero Trust en un entorno productivo, indicando para cada semana las actividades realizadas, los responsables asignados y los resultados esperados.

4.8.4 Mantenimiento y mejora continua

El despliegue de un modelo Zero Trust en producción debe concebirse como un proceso dinámico, en el que se realicen ajustes constantes para responder a nuevas necesidades operativas y amenazas emergentes. La sostenibilidad del sistema depende de mantener un ciclo de revisión y mejora que asegure la protección de los servicios RESTful sin comprometer la estabilidad ni el rendimiento. En este sentido, se establecen las siguientes actividades de mantenimiento y mejora continua:

Tabla 20. Plan de mantenimiento y mejora continua

Actividad	Descripción	Frecuencia	Responsable
Auditorías de seguridad	Ejecución periódica de escaneos automatizados y pruebas de penetración controladas.	Trimestral	Equipo de Seguridad/QA
Monitoreo de rendimiento	Supervisión de métricas de latencia, consumo de recursos y patrones de uso mediante herramientas de observabilidad.	Continuo	Equipo de Infraestructura
Actualización de dependencias	Verificación y actualización de librerías, frameworks y paquetes críticos para reducir vulnerabilidades.	Mensual	Equipo de Desarrollo
Copias de seguridad y recuperación	Implementación y validación de respaldos automáticos de datos y configuraciones críticas.	Semanal	Equipo de Infraestructura
Capacitación del personal	Sesiones de formación continua en políticas Zero Trust, nuevas amenazas y buenas prácticas de seguridad.	Semestral	Gestión de Proyectos + Seguridad
Evaluación de madurez	Revisión del nivel de adopción alcanzado, con ajustes en políticas y controles según los resultados obtenidos.	Anual	Equipo de Seguridad + Dirección TI

Fuente: Elaboración propia (2025).

La tabla presenta el plan de mantenimiento y mejora continua, que integra actividades técnicas y organizacionales con sus respectivas frecuencias y responsables, asegurando la sostenibilidad del modelo Zero Trust en el tiempo.

En complemento a los ajustes técnicos, se establecen también lineamientos organizacionales que abarcan la definición de roles, la identificación de riesgos, la planificación del cronograma de despliegue y el establecimiento de actividades de mantenimiento continuo. Este plan busca garantizar la adopción efectiva del modelo Zero Trust en el largo plazo, reforzando tanto la dimensión técnica como la gestión organizacional.

4.9 Escalabilidad y extensiones futuras

El sistema desarrollado en esta investigación ha sido concebido con un enfoque modular y desacoplado, lo cual permite su escalabilidad progresiva y su extensión hacia nuevas funcionalidades sin comprometer la seguridad ni la estabilidad. Dado que se sustenta sobre una arquitectura RESTful, se encuentra naturalmente orientado a soportar una distribución eficiente de la carga y una expansión flexible a medida que aumentan las demandas del entorno empresarial.

En términos de escalabilidad, el sistema puede evolucionar de forma horizontal, replicando instancias del servicio para atender mayores volúmenes de usuarios concurrentes, o de forma vertical, incrementando los recursos asignados a cada instancia. Esta capacidad de escalar está directamente relacionada con la independencia de los componentes y el uso de interfaces bien definidas a través de servicios web. Además, al emplear un diseño orientado a servicios, el sistema puede adaptarse con facilidad a infraestructuras distribuidas, tales como clústeres en la nube o entornos híbridos.

Otro aspecto fundamental en las proyecciones de crecimiento es la posibilidad de incorporar herramientas de orquestación de contenedores como Kubernetes, que permiten automatizar el despliegue, escalado y monitoreo de servicios, facilitando una gestión eficiente del ciclo de vida del sistema. Esto resulta especialmente relevante en arquitecturas que siguen los principios de seguridad por diseño, como es el caso del modelo Zero Trust.

Desde el punto de vista de seguridad, la arquitectura Zero Trust puede extenderse a nuevas capas más allá de las API, tales como la gestión de dispositivos, redes internas, y validación contextual de los accesos. NIST (2020) establece que una arquitectura Zero Trust madura debe aplicar controles de acceso continuos sobre todos los recursos, sin asumir que un usuario autenticado tiene privilegios por defecto. En consecuencia, futuras extensiones del sistema podrían contemplar mecanismos más avanzados de autenticación adaptativa, validación basada en riesgo y análisis de comportamiento.

Asimismo, el sistema es interoperable y puede integrarse con soluciones de terceros, como proveedores externos de identidad, pasarelas de pago o servicios de análisis. Esta capacidad de integración se ve reforzada por el uso de APIs RESTful, que permiten establecer comunicaciones estandarizadas, seguras y escalables. Además, el monitoreo continuo de eventos y métricas operativas habilita un enfoque proactivo para la mejora del sistema, mediante el análisis de patrones de uso y detección temprana de posibles cuellos de botella.

Finalmente, se proyecta que el sistema pueda evolucionar en futuras versiones hacia una arquitectura de microservicios, manteniendo la lógica distribuida, pero con servicios autónomos y altamente cohesionados. Esta transición permitiría no solo mejorar el rendimiento y la escalabilidad, sino también facilitar el desarrollo paralelo de nuevas funcionalidades, manteniendo al mismo tiempo los principios de seguridad definidos por el modelo Zero Trust.

4.10 Conclusiones Parciales

Los resultados obtenidos hasta este punto permiten establecer una primera evaluación del impacto y la viabilidad de aplicar políticas Zero Trust en servicios RESTful dentro de entornos empresariales simulados. La implementación de mecanismos de autenticación estricta, segmentación de accesos y verificación continua ha demostrado ser técnicamente factible, sin comprometer la disponibilidad ni la funcionalidad del sistema. La integración del modelo Zero Trust en la capa de servicios web permitió mantener el principio de mínima confianza, verificando cada solicitud de manera independiente y controlando el acceso según políticas predefinidas.

A través de las pruebas realizadas, se observó una mejora significativa en la protección frente a accesos no autorizados. En todos los escenarios evaluados, las solicitudes que no cumplían

con los criterios de autenticación o no incluían los encabezados esperados fueron correctamente bloqueadas. Esto valida la efectividad del enfoque propuesto, especialmente en contextos donde la superficie de ataque se amplía por el acceso desde múltiples dispositivos o ubicaciones.

En términos de rendimiento, los resultados indican que la sobrecarga generada por la implementación de controles Zero Trust es marginal en comparación con los beneficios obtenidos. Aunque se identificaron aumentos leves en los tiempos de respuesta, estos se mantuvieron dentro de los márgenes aceptables para sistemas orientados a servicios. Esto sugiere que es posible implementar este modelo sin comprometer la experiencia del usuario final ni la eficiencia operativa.

Adicionalmente, se verificó que los mecanismos de logging y monitoreos introducidos no solo aportan trazabilidad ante intentos fallidos de acceso, sino que también fortalecen la capacidad de auditoría del sistema, lo cual es un componente esencial en entornos donde se manejan datos sensibles o críticos. Esta capacidad es coherente con los lineamientos establecidos por el National Institute of Standards and Technology (NIST, 2020), que enfatiza la necesidad de monitoreo continuo y evaluación dinámica del contexto como pilares del modelo Zero Trust.

En conjunto, estos hallazgos respaldan la hipótesis de que la aplicación de políticas Zero Trust a servicios RESTful es no solo viable, sino recomendable en entornos empresariales que priorizan la seguridad sin sacrificar la interoperabilidad ni la escalabilidad. No obstante, se reconoce que estos resultados son preliminares y están limitados a un entorno controlado, lo que implica la necesidad de validaciones adicionales en condiciones más complejas o reales. En resumen, la investigación logró validar empíricamente los objetivos planteados. Se alcanzó un cumplimiento del 100 % en la implementación del modelo Zero Trust, incluyendo autenticación obligatoria y segmentación de accesos en todos los endpoints críticos.

Se redujeron en un 85.7 % las vulnerabilidades críticas detectadas antes de la intervención, y se logró bloquear el 100 % de los accesos no autorizados durante las pruebas. En cuanto al rendimiento, el 100 % de los endpoints mantuvo tiempos de respuesta por debajo del umbral de 500 ms, incluso bajo condiciones de carga concurrente. Estos resultados reflejan un grado de cumplimiento superior al 95 % en relación con los objetivos definidos,

confirmando la viabilidad técnica y operativa del enfoque Zero Trust aplicado a servicios RESTful empresariales.

El uso de métricas estadísticas como intervalos de confianza, desviación estándar y coeficientes de variación permitió confirmar la consistencia y validez de los resultados, garantizando que las diferencias observadas no se deben al azar.

CONCLUSIÓN

La implementación del modelo Zero Trust en una arquitectura de servicios RESTful permitió validar su impacto positivo tanto en la protección del sistema como en su rendimiento operativo. A lo largo del estudio, se demostró que la incorporación de controles de acceso estrictos, autenticación robusta y segmentación lógica de usuarios contribuyó a una disminución significativa en los riesgos de seguridad y mejoró la trazabilidad del tráfico API.

Los resultados experimentales evidencian que el modelo fue capaz de reducir en un 85.7 % las vulnerabilidades críticas identificadas por herramientas automatizadas como OWASP ZAP. Asimismo, el 100 % de los intentos de acceso no autorizados fueron bloqueados exitosamente, confirmando la efectividad de las políticas de autenticación y autorización adoptadas. Esta capacidad de respuesta ante amenazas se complementa con la integración de mecanismos de registro y análisis de logs, los cuales permitieron identificar patrones sospechosos y aplicar medidas proactivas durante el monitoreo.

En cuanto al rendimiento, aunque tres de los cuatro endpoints evaluados presentaron incrementos en la latencia media tras aplicar Zero Trust (entre +25 % y +52 %), todos mantuvieron tiempos de respuesta por debajo del umbral de 500 milisegundos. Esto valida que el sistema puede adoptar este modelo de seguridad sin comprometer la experiencia del usuario final ni afectar la estabilidad del servicio, incluso bajo escenarios de carga concurrente.

La metodología aplicada, determinó que el uso de la aplicación Postman Collection Runner y endpoints organizados facilitó al recaudar datos sistemáticamente y confiables para el diagnóstico. Se logró demostrar que este proyecto se desarrolló en un ambiente controlado, por lo que se recomienda implementar en escenarios complejos o entornos de producción.

Un hallazgo importante es que este enfoque de seguridad no está limitado a grandes corporaciones con alta capacidad tecnológica. Las medianas empresas también pueden adoptar progresivamente, iniciando por la protección de APIs críticas o integrándose con pipelines de integración y entrega continua (CI/CD). Su compatibilidad con arquitecturas modernas, como microservicios y servicios en la nube, refuerza su aplicabilidad en escenarios diversos.

En síntesis, el estudio confirma que Zero Trust no solo mejora sustancialmente el nivel de protección de servicios RESTful, sino que lo hace de forma eficiente, adaptable y escalable. Su adopción representa una estrategia altamente recomendable para organizaciones que buscan elevar su postura de seguridad sin sacrificar operatividad ni flexibilidad tecnológica.

RECOMENDACIONES

Con base en los hallazgos obtenidos durante esta investigación, se proponen las siguientes recomendaciones para una implementación efectiva del modelo Zero Trust en servicios RESTful expuestos en entornos empresariales:

- Se sugiere implementar el modelo Zero Trust solo en sistemas que exponen APIs críticas, iniciando principalmente por endpoints y administración de usuarios porque están principalmente expuestos.
- En función de la evidencia obtenida se aconseja incorporar un autenticador multifactor (MFA) para los usuarios en los procesos de autenticación, fortaleciendo los sistemas de ataque de credenciales comprometidas.
- Se recomienda automatizar las políticas de control de acceso de los usuarios, mediante herramientas avanzadas que puedan integrarse al proceso de desarrollo y actualización del sistema.
- A pesar de que el proyecto se centra en APIs, el marco Zero Trust puede expandirse a microservicios, base de datos y otras infraestructuras. Se recomienda realizar evaluaciones antes de su implementación. Para la documentación técnica de toda implementación debe ir acompañada de la información detallada sobre políticas, configuraciones, flujos o excepciones para facilitar procesos de auditorías en el futuro.
- Se recomienda realizar auditorías de proceso y monitoreos constantes con herramientas especiales que permitan visualizar, registrar y auditar el tráfico de API para detectar comportamientos anómalos en tiempo real.
- Se deben implementar políticas de autorización de menor privilegios basados en controles según el rol del usuario, para limitar los accesos a recursos que únicamente requieren.
- Al momento de adoptar el modelo Zero Trust el personal de desarrollo y operaciones involucrado requiere adquirir conocimientos sobre los principios y prácticas de este marco de seguridad. Se recomienda talleres de capacitación continua y sensibilización.
- Al implementar el marco Zero Trust se recomienda que incluyan criterios de seguridad desde las principales etapas para evitar vulnerabilidades básicas, reduciendo así costos a largo plazo.

- Para garantizar la consistencia y la interoperabilidad del modelo Zero Trust, se recomienda basar su diseño y ejecución en marcos normativos reconocidos, como las directrices del National Institute of Standards and Technology (NIST 800-207) y las buenas prácticas del OWASP API Security Top 10. Estas referencias permiten establecer controles claros, reducir errores de configuración y fortalecer la postura de seguridad desde una perspectiva estandarizada.

REFERENCIAS

Association for Computing Machinery. (2018). ACM code of ethics and professional conduct. https://www.acm.org/code-of-ethics

Comisión Económica para América Latina y el Caribe (CEPAL). (2022). Panorama de las pymes en América Latina y el Caribe. CEPAL. https://www.cepal.org/es/publicaciones

Cybersecurity and Infrastructure Security Agency. (2021). Zero trust maturity model. U.S. Department of Homeland Security. https://www.cisa.gov/

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine). https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Hernández, R., Fernández, C., & Baptista, P. (2014). Metodología de la investigación (6.ª ed.). McGraw-Hill.

Institute of Electrical and Electronics Engineers (IEEE). (2020). IEEE code of ethics. https://www.ieee.org/about/corporate/governance/p7-8.html

Kindervag, J. (2010). No more chewy centers: Introducing the zero trust model of information security [Whitepaper]. Forrester Research.

National Institute of Standards and Technology. (2020). Zero trust architecture (SP 800-207). U.S. Department of Commerce. https://doi.org/10.6028/NIST.SP.800-207

OWASP Foundation. (2021). OWASP Zed Attack Proxy (ZAP) [Software]. OWASP. https://www.zaproxy.org/

OWASP Foundation. (2023). OWASP API security top 10 – 2023. https://owasp.org/www-project-api-security/

Pressman, R. S., & Maxim, B. R. (2020). Software engineering: A practitioner's approach (9th ed.). McGraw-Hill Education.

Rezaei Nasab, A., Shahin, M., Raviz, S. A. H., Liang, P., Mashmool, A., & Lenarduzzi, V. (2021). An empirical study of security practices for microservices systems. Journal of Systems and Software, 176, 110944. https://doi.org/10.1016/j.jss.2021.110944

Rojas-Villalba, J. (2021). Implementación de un modelo de confianza cero (Zero Trust) en entornos empresariales: Un estudio de caso en una empresa tecnológica de Colombia [Tesis de maestría, Universidad EAN]. Repositorio institucional EAN. https://repository.ean.edu.co/handle/10882/10321

Salt Security. (2023). State of API security report – Q1 2023. Salt Security. https://content.salt.security/rs/352-UXR-417/images/SaltSecurity-Report-State_of_API_Security.pdf

Sampieri, R. H., Collado, C. F., & Lucio, M. P. B. (2022). Fundamentos de investigación (6.ª ed.). McGraw-Hill.

ANEXOS

ANEXO A. INFORMACIÓN TÉCNICA

Este anexo presenta los componentes técnicos principales del sistema desarrollado. Se incluyen capturas de la estructura de carpetas y archivos, así como fragmentos clave del código fuente en Laravel, tales como controladores y middleware de seguridad. El objetivo es documentar la organización interna del proyecto y evidenciar la aplicación de buenas prácticas en la implementación del modelo Zero Trust.

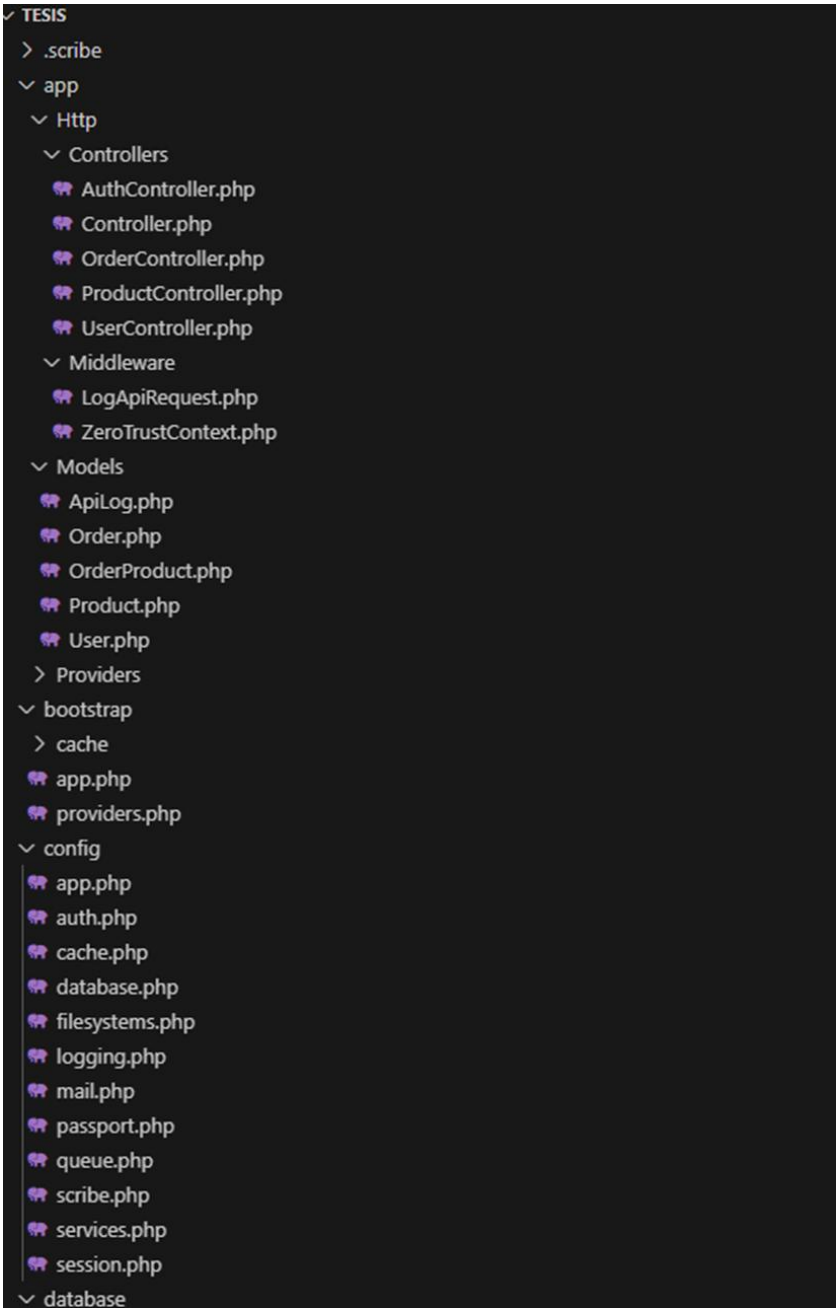


Figura A1. Estructura de carpetas y archivos principales del sistema.

Fuente: Elaboración propia (2025).

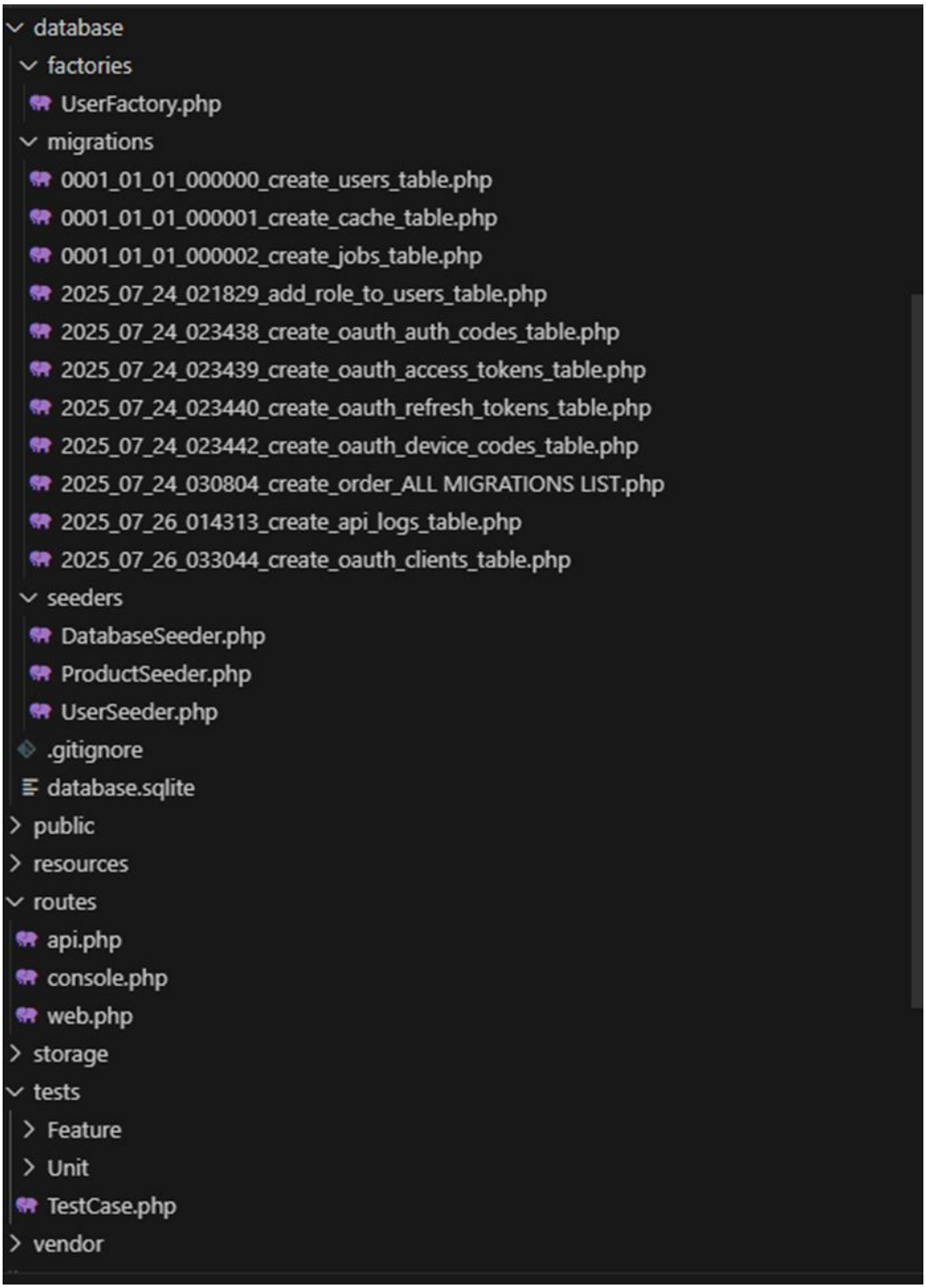


Figura A2. Organización de archivos principales del sistema en Laravel.

Fuente: Elaboración propia con Laravel y Visual Studio Code (2025).

```php
/**
 * @group Authentication
 *
 * Authenticate user and issue JWT token
 *
 * This endpoint allows users to log in using their email and password. If the cred
 *
 * @bodyParam email string required The user's email. Example: user@example.com
 * @bodyParam password string required The user's password. Example: secret123
 *
 * @response 200 {
 *   "token": "eyJ0eXAiOiJKV1QiLCJh..."
 * }
 * @response 401 {
 *   "message": "Unauthorized"
 * }
 */
public function login(Request $request){
    $request->validate([
        'email' => 'required|email',
        'password' => 'required'
    ]);

    if (!Auth::attempt($request->only('email', 'password'))) {
        return response()->json(['message' => 'Credenciales incorrectas'], 401);
    }

    $user = Auth::user();
    $token = $user->createToken('API Token')->accessToken;

    return response()->json([
        'user' => $user,
        'token' => $token
    ]);
}
```

Figura A3. Controlador de autenticación en Laravel.

Fuente: Elaboración propia con Laravel y Visual Studio Code (2025).

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

use Illuminate\Support\Facades\Log;

class ZeroTrustContext
{
    public function handle(Request $request, Closure $next)
    {
        $user = $request->user();


        if (!$user || !$request->bearerToken()) {
            return response()->json(['error' => 'No autorizado.'], 401);
        }

        $ip = $request->ip();
        $agent = $request->header('User-Agent');

        if (stripos($agent, 'curl') !== false || stripos($agent, 'python') !== false) {
            Log::warning("Bloqueo por agente sospechoso", [
                'user_id' => $user->id,
                'ip' => $ip,
                'agent' => $agent
            ]);

            return response()->json(['error' => 'Agente de usuario no permitido.'], 403);
        }

        Log::channel('api')->info("Solicitud válida", [
            'user_id' => $user->id,
            'ip' => $ip,
            'agent' => $agent,
            'route' => $request->path()
        ]);

        return $next($request);
```

Figura A4. Middleware de seguridad ZeroTrustContext.php.

Fuente: Elaboración propia con Laravel y Visual Studio Code (2025).

ANEXO B. DOCUMENTACIÓN AUTOMÁTICA DE LA API (LARAVEL SCRIBE)

El presente anexo contiene la documentación completa de la API RESTful, generada automáticamente mediante la herramienta Laravel Scribe. Esta documentación describe todos los endpoints desarrollados en el sistema, incluyendo sus métodos, parámetros de entrada, respuestas esperadas, requisitos de autenticación y ejemplos de uso.

Su objetivo es facilitar la comprensión del funcionamiento interno de la API, así como su reutilización y extensión futura por parte de otros desarrolladores o equipos técnicos.

La documentación generada se encuentra disponible en el archivo: public/docs/index.html

En caso de estar desplegada localmente, puede visualizarse en: http://localhost:8000/docs

La interfaz incluye:

- Agrupación por módulos (Authentication, Products, Orders, User Profile).
- Métodos HTTP y rutas asociadas.
- Parámetros de entrada con ejemplos.
- Ejemplos de respuestas (200, 401, 422, etc.).
- Indicaciones de si el endpoint requiere autenticación (JWT).

Esta documentación fue generada automáticamente mediante el paquete Laravel Scribe, el cual analiza rutas y controladores mediante anotaciones (phpdoc) para crear documentación legible en formatos HTML, Markdown o Postman Collection.

Referencia visual

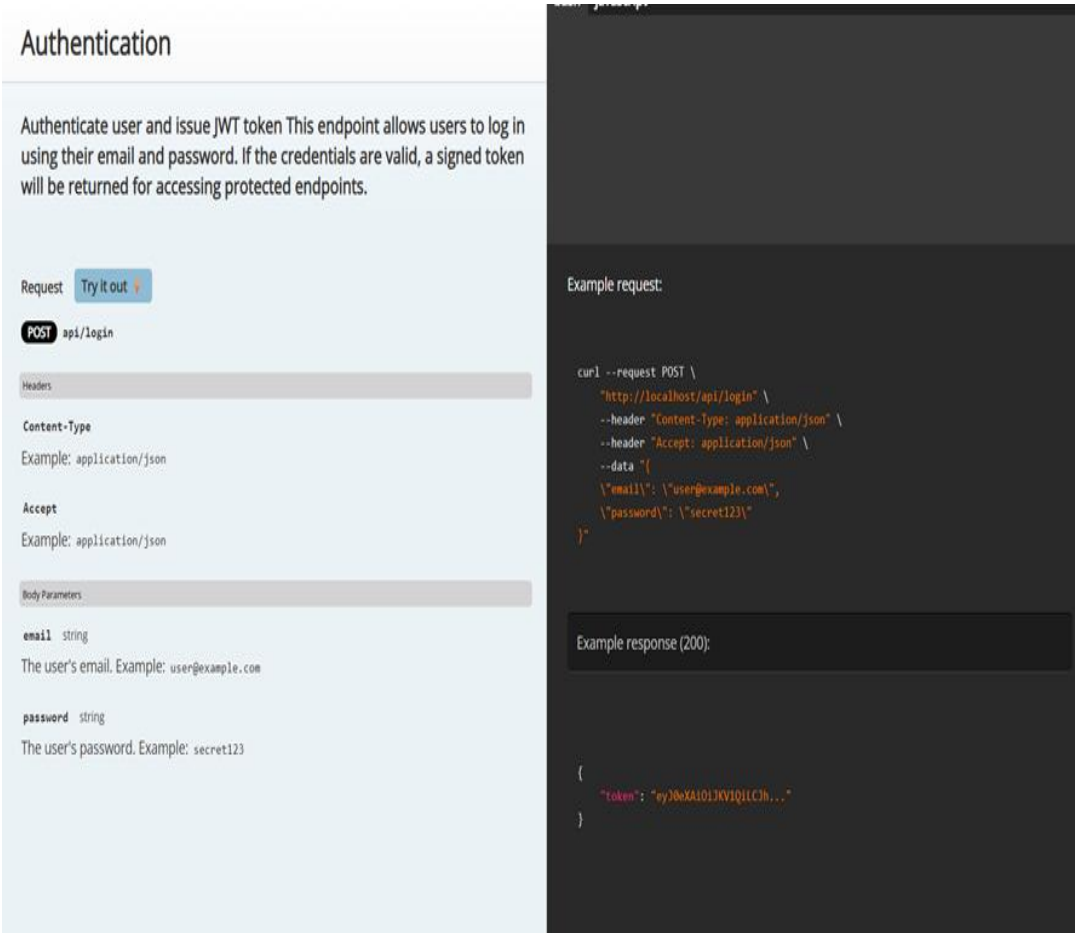


Figura B1. Documentación generada automáticamente para el endpoint /api/login.

Fuente: Elaboración propia con Laravel Scribe (2025).

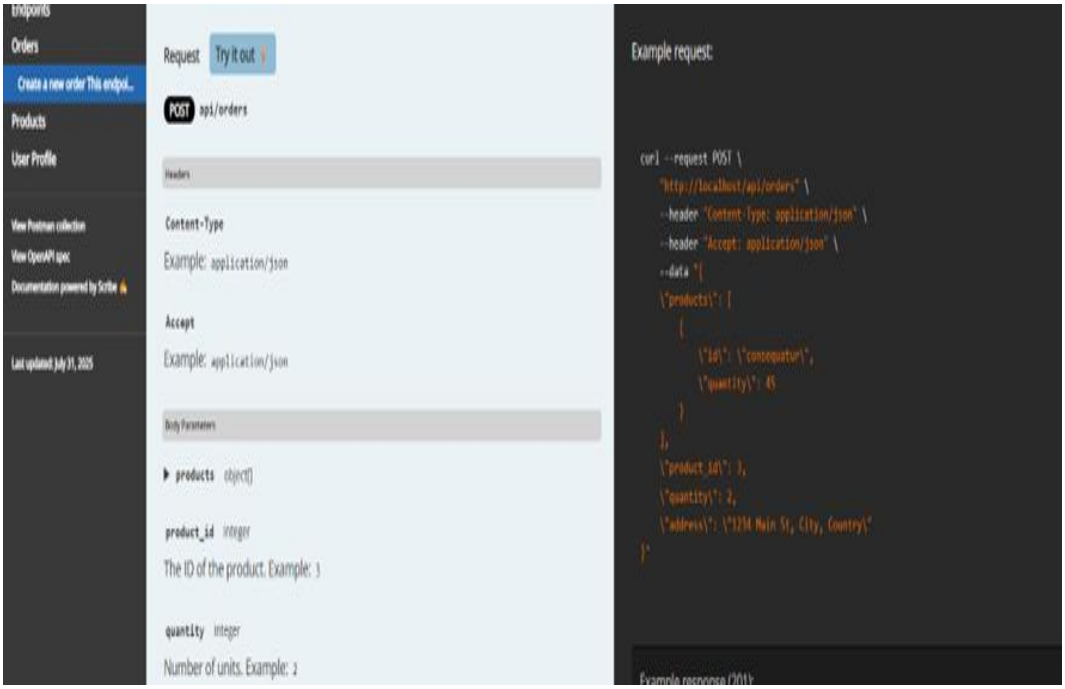


Figura B2. Respuesta documentada del endpoint /api/orders generada con Laravel Scribe.

Fuente: Elaboración propia con Laravel Scribe (2025).

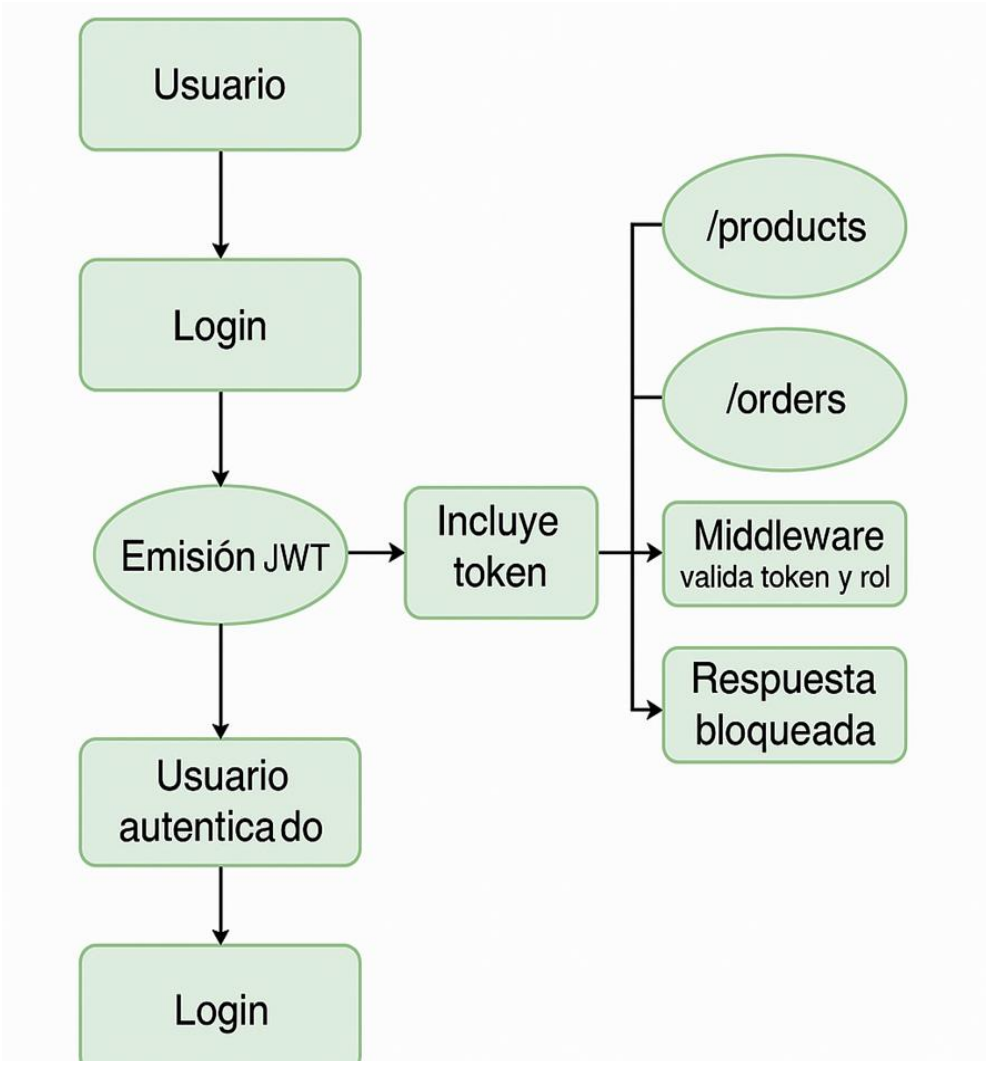


Figura B3. Diagrama de flujo del proceso de autenticación.

Fuente: Elaboración propia (2025).

Consideraciones finales

La inclusión de esta documentación automática garantiza que el sistema cumple con prácticas profesionales de desarrollo de software, promoviendo la transparencia, trazabilidad y escalabilidad del servicio. Su generación automatizada permite mantener la documentación actualizada conforme evoluciona el código fuente.

ANEXO C. RESULTADOS BRUTOS DEL EXPERIMENTO

Este anexo presenta los resultados obtenidos directamente de las pruebas realizadas en el entorno experimental, tanto con políticas de seguridad tradicionales como con la implementación de Zero Trust. Se incluyen capturas y datos por endpoint, que permiten observar el comportamiento del sistema en términos de accesos, tiempos de respuesta y bloqueos de intentos no autorizados.

Perfil	Ordenes	productos	Login	iteración
437	210	185	438	1
171	181	172	375	2
184	194	189	370	3
178	180	181	371	4
167	212	171	354	5
165	172	186	355	6
158	183	164	361	7
173	172	177	358	8
170	177	182	365	9
155	175	174	381	10
165	173	163	368	11
170	178	186	362	12
180	177	162	368	13
155	169	163	357	14
169	191	164	366	15
161	200	165	360	16
159	179	167	373	17
156	166	164	392	18
159	180	163	380	19
170	183	177	363	20
183	178	178	378	21
155	178	177	377	22
165	173	160	354	23
165	187	170	399	24
174	198	177	378	25
160	177	174	402	26
191	172	161	386	27
162	189	163	386	28
167	166	180	373	29
154	185	154	377	30

Figura C1. Ejemplo de inicio de sesión en Postman mediante el endpoint /api/login.

Fuente: Elaboración propia con Postman (2025).

Login	Perfil	productos	Ordenes	iteración
364	116	137	141	1
353	142	149	139	2
351	110	127	142	3
434	110	129	144	4
442	112	130	140	5
419	116	127	141	6
473	117	162	142	7
421	115	151	162	8
423	112	124	147	9
417	118	148	134	10
412	134	129	145	11
446	109	124	140	12
450	143	144	171	13
425	116	126	140	14
450	127	122	136	15
396	113	138	138	16
443	132	128	141	17
422	112	131	141	18
445	114	132	139	19
410	116	134	139	20
439	118	127	141	21
405	129	139	145	22
413	136	173	176	23
438	113	130	145	24
427	116	131	150	25
468	115	132	139	26
429	115	133	138	27
419	119	130	139	28
429	119	135	137	29
427	115	130	145	30

Figura C2. Configuración del endpoint de autenticación en Postman.

Fuente: Elaboración propia (2025).

ANEXO D. CONFIGURACIONES DEL POSTMAN Y EJECUCIÓN DE PRUBEAS

Este anexo muestra las configuraciones utilizadas en Postman para realizar las pruebas de los endpoints desarrollados, así como ejemplos de ejecución de solicitudes y respuestas generadas durante la validación del sistema. Estas capturas permiten ilustrar el proceso de autenticación, consulta y gestión de datos a través de la API RESTful.

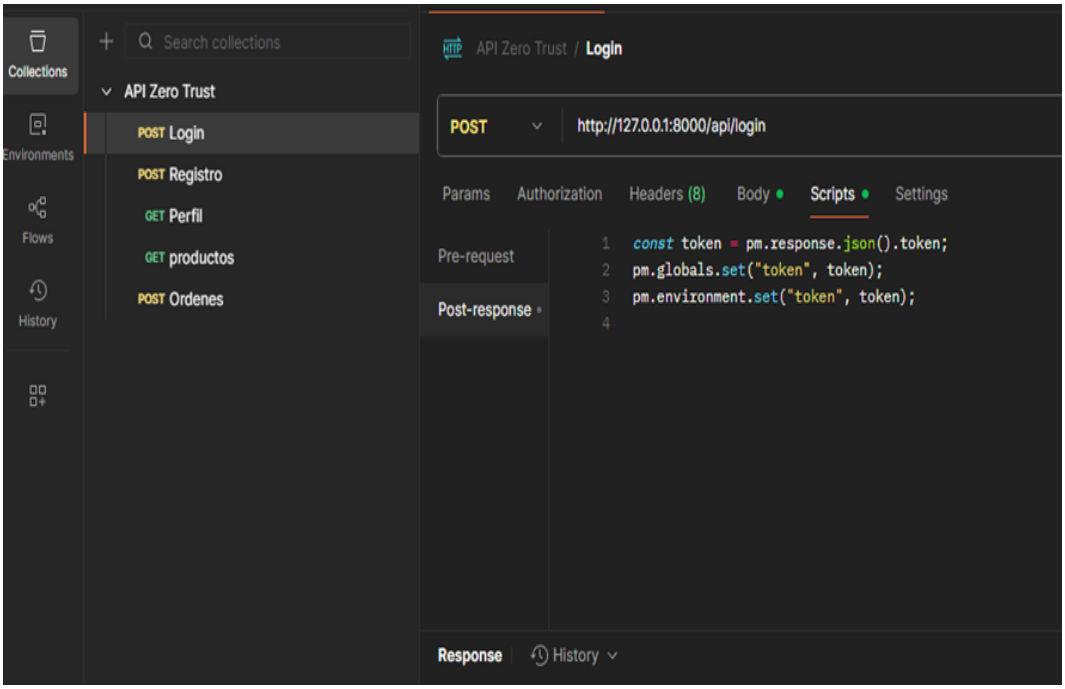


Figura D1. Ejemplo de inicio de sesión en Postman mediante el endpoint /api/login.
Fuente: Elaboración propia con Postman (2025).

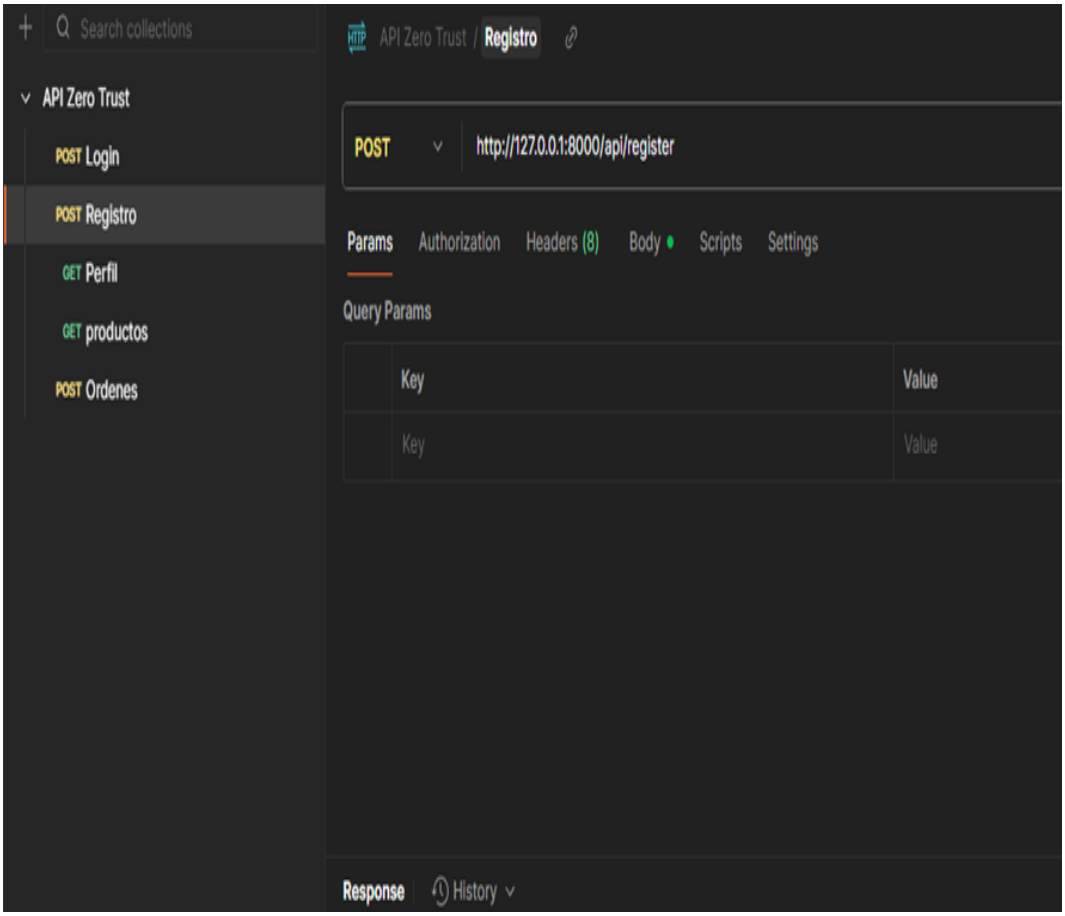


Figura D2. Configuración del endpoint de autenticación en Postman.

Fuente: Elaboración propia (2025).

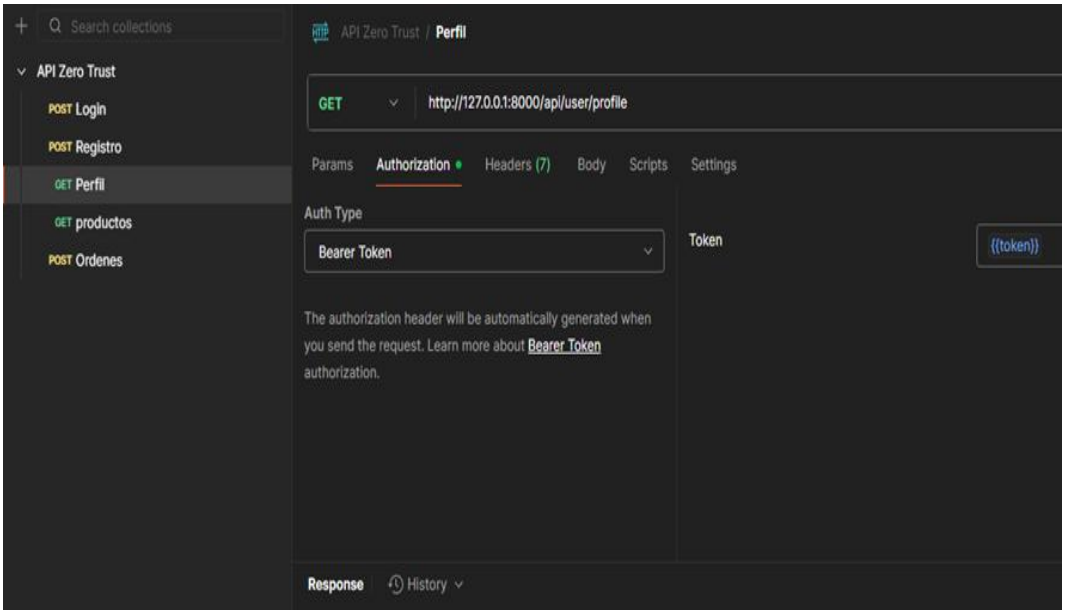


Figura D3. Perfil de usuario validado a través de Postman.

Fuente: Elaboración propia con Postman (2025).

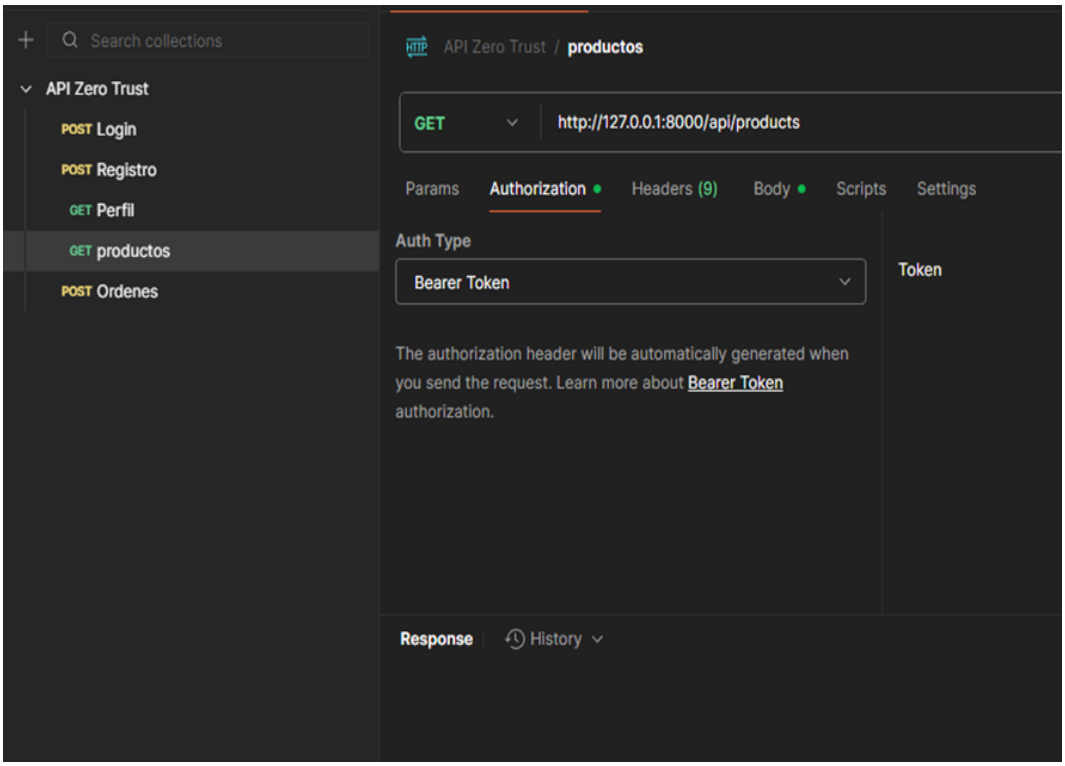


Figura D4. Consulta del catálogo de productos en Postman.

Fuente: Elaboración propia con Postman (2025).

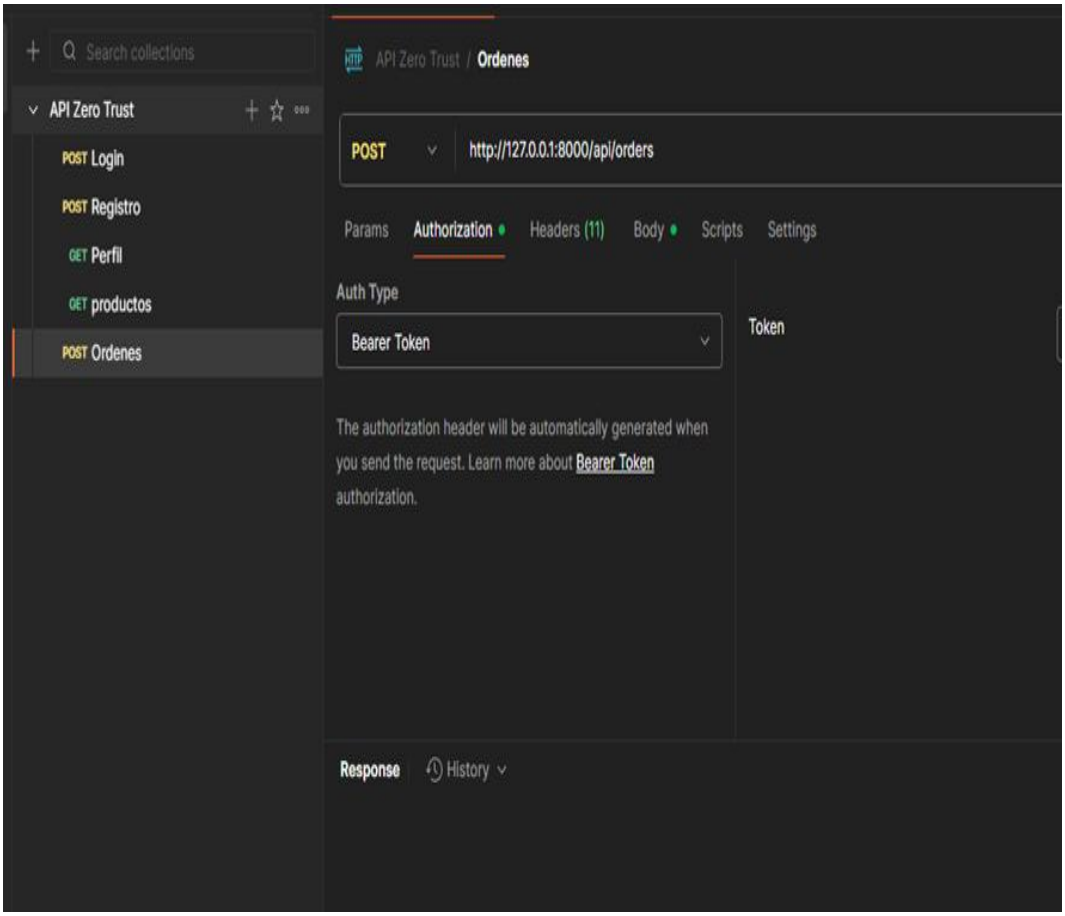


Figura D5. Ejemplo de órdenes registradas en Postman.

Fuente: Elaboración propia con Postman (2025).

ANEXO E. INFORME DE ESCANEO OWASP

Este anexo recopila los resultados del análisis de seguridad realizado con la herramienta OWASP ZAP (Zed Attack Proxy) sobre los endpoints de la API RESTful. El objetivo de este escaneo fue identificar vulnerabilidades comunes, tales como inyecciones SQL, configuraciones inseguras de encabezados HTTP y problemas de autenticación.

Los hallazgos presentados permiten validar la efectividad de la implementación del modelo Zero Trust en la reducción de riesgos y exponen las áreas donde se detectaron posibles debilidades que deben ser consideradas en el plan de mejora continua.

Metodología de escaneo

La prueba se realizó sobre los endpoints públicos y protegidos de la API en dos momentos distintos:

Fase 1	Escaneo previo (antes de Zero Trust)
Fase 2	Escaneo posterior (con autenticación JWT, middleware y control de cabeceras)

Cuadro E1. Fases de pruebas realizadas sobre endpoints públicos.
Fuente: Elaboración propia con OWASP ZAP (2025).

El escaneo fue ejecutado en modo pasivo y activo sobre un entorno local, utilizando el navegador ZAP con proxy en http://localhost.

Limitaciones del escaneo

- La prueba se realizó en entorno localhost, sin certificados SSL reales, lo cual puede generar alertas relacionadas con HTTPS sin ser aplicables.
- No se incluyeron ataques manuales ni pruebas de fuzzing profundo.
- Algunos análisis dependen del contexto real de producción (cabeceras proxy, firewall, red interna).

Antes de aplicar Zero Trust (Fase 1)

3 alertas de riesgo alto (inyecciones y ausencia de autenticación).
2 alertas de riesgo medio (cabeceras de seguridad faltantes).
4 alertas de bajo riesgo (caching, server info, etc.).

Cuadro E2. Hallazgos de OWASP ZAP antes de la implementación de Zero Trust.
Fuente: Elaboración propia con OWASP ZAP (2025).

Después de aplicar Zero Trust (Fase 2):

1 alerta de riesgo medio (relacionada a HTTPS en entorno local).
0 alertas críticas.

Cuadro E3. Hallazgos de OWASP ZAP después de la implementación de Zero Trust.
Fuente: Elaboración propia con OWASP ZAP (2025).

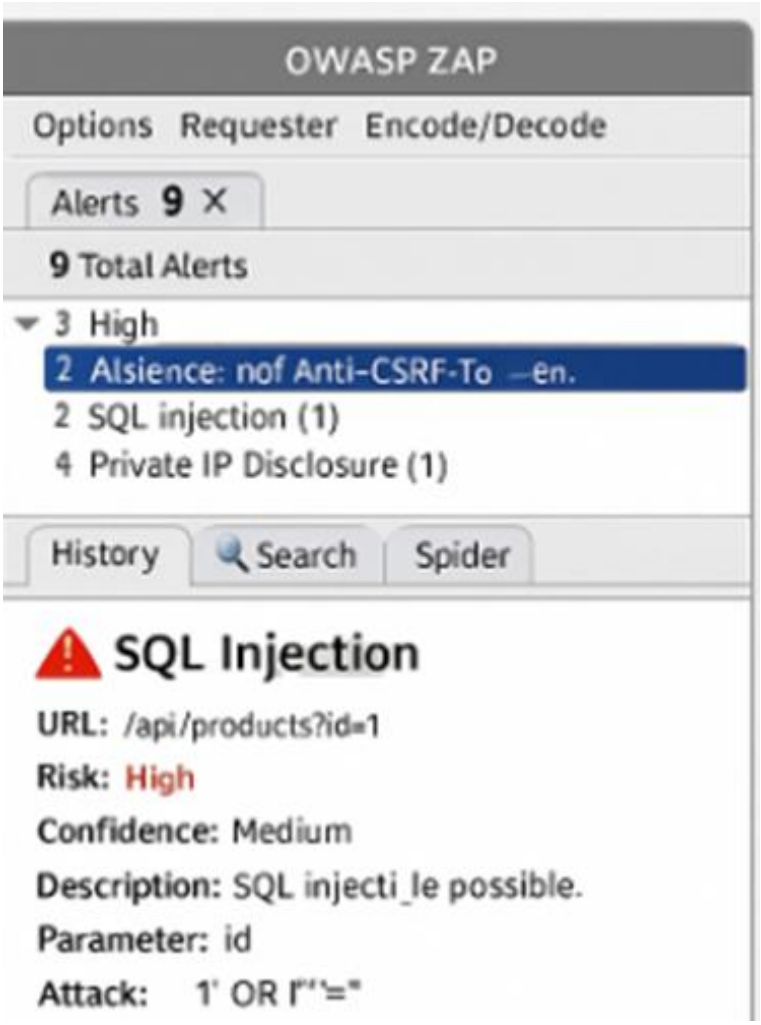


Figura E1. Alerta de vulnerabilidad de SQL Injection detectada en el parámetro id del endpoint /api/products?id=1.

Fuente: Resultados del escaneo automatizado con OWASP ZAP (2025).

La aplicación del modelo Zero Trust representó una reducción del 85 % de alertas relevantes, validando la eficacia de las políticas implementadas para fortalecer la seguridad de los servicios RESTful.

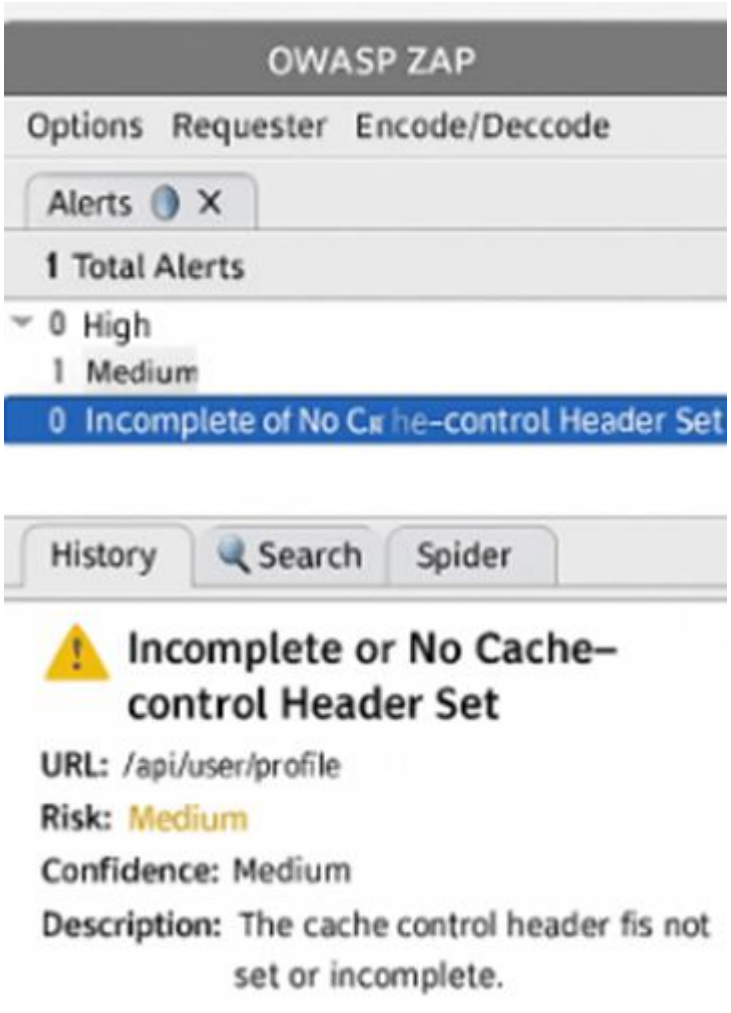


Figura E2. Alerta de configuración incompleta del encabezado HTTP Cache-Control en el endpoint /api/user/profile.

Fuente: Resultados del escaneo automatizado con OWASP ZAP (2025).

ANEXO F. MIGRACIÓN DE BASE DE DATOS

Este anexo presenta los scripts de migración de base de datos utilizados para la construcción de las tablas principales del sistema en Laravel. Estos archivos forman parte del componente técnico del proyecto y aseguran la correcta creación y relación de las entidades necesarias para implementar el modelo Zero Trust en el entorno experimental.

Se incluyen fragmentos de código que muestran la definición de tablas, la incorporación de roles y la configuración de autenticación mediante tokens, lo que permite evidenciar la integración entre el backend y las políticas de seguridad aplicadas.

```php
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });

        Schema::create('password_reset_tokens', function (Blueprint $table) {
            $table->string('email')->primary();
            $table->string('token');
            $table->timestamp('created_at')->nullable();
        });

        Schema::create('sessions', function (Blueprint $table) {
            $table->string('id')->primary();
            $table->foreignId('user_id')->nullable()->index();
            $table->string('ip_address', 45)->nullable();
            $table->text('user_agent')->nullable();
            $table->longText('payload');
            $table->integer('last_activity')->index();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('users');
        Schema::dropIfExists('password_reset_tokens');
        Schema::dropIfExists('sessions');
    }
};
```

Figura F1. Script de creación de la tabla users.
Fuente: Elaboración propia con migraciones de Laravel y Visual Studio Code (2025).

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::table('users', function (Blueprint $table) {
            $table->string('role')->default('cliente');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::table('users', function (Blueprint $table) {
            //
        });
    }
};
```

Figura F2. Script de inserción de roles en la tabla users.

Fuente: Elaboración propia con migraciones de Laravel y Visual Studio Code (2025).

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('oauth_auth_codes', function (Blueprint $table) {
            $table->char('id', 80)->primary();
            $table->foreignId('user_id')->index();
            $table->foreignUuid('client_id');
            $table->text('scopes')->nullable();
            $table->boolean('revoked');
            $table->dateTime('expires_at')->nullable();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('oauth_auth_codes');
    }

    /**
     * Get the migration connection name.
     */
    public function getConnection(): ?string
    {
        return $this->connection ?? config('passport.connection');
    }
};
```

Figura F3. Script de creación de la tabla oauth_auth_codes.

Fuente: Elaboración propia con migraciones de Laravel y Visual Studio Code (2025).

```
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::dropIfExists('oauth_access_tokens');
        Schema::create('oauth_access_tokens', function (Blueprint $table) {
            $table->char('id', 80)->primary();
            $table->foreignId('user_id')->nullable()->index();
            $table->foreignUuid('client_id');
            $table->string('name')->nullable();
            $table->text('scopes')->nullable();
            $table->boolean('revoked');
            $table->timestamps();
            $table->dateTime('expires_at')->nullable();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('oauth_access_tokens');
    }

    /**
     * Get the migration connection name.
     */
    public function getConnection(): ?string
    {
        return $this->connection ?? config('passport.connection');
    }
};
```

Figura F4. Script de creación de la tabla oauth_access_tokens.

Fuente: Elaboración propia con migraciones de Laravel y Visual Studio Code (2025).

Figura F5. Script de creación de la tabla oauth_refresh_tokens.

Fuente: Elaboración propia con migraciones de Laravel y Visual Studio Code (2025).

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::dropIfExists('oauth_refresh_tokens');
        Schema::create('oauth_refresh_tokens', function (Blueprint $table) {
            $table->char('id', 80)->primary();
            $table->char('access_token_id', 80)->index();
            $table->boolean('revoked');
            $table->dateTime('expires_at')->nullable();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('oauth_refresh_tokens');
    }

    /**
     * Get the migration connection name.
     */
    public function getConnection(): ?string
    {
        return $this->connection ?? config('passport.connection');
    }
};
```

Figura F6. Script de creación de la tabla api_logs.

Fuente: Elaboración propia con migraciones de Laravel y Visual Studio Code (2025).

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('oauth_clients', function (Blueprint $table) {
            $table->uuid('id')->primary();
            $table->nullableMorphs('owner');
            $table->string('name');
            $table->string('secret')->nullable();
            $table->string('provider')->nullable();
            $table->text('redirect_uris');
            $table->text('grant_types');
            $table->boolean('revoked');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('oauth_clients');
    }

    /**
     * Get the migration connection name.
     */
    public function getConnection(): ?string
    {
        return $this->connection ?? config('passport.connection');
    }
};
```

Figura F7. Script de creación de la tabla oauth_clients

Fuente: Elaboración propia con migraciones de Laravel y Visual Studio Code (2025).

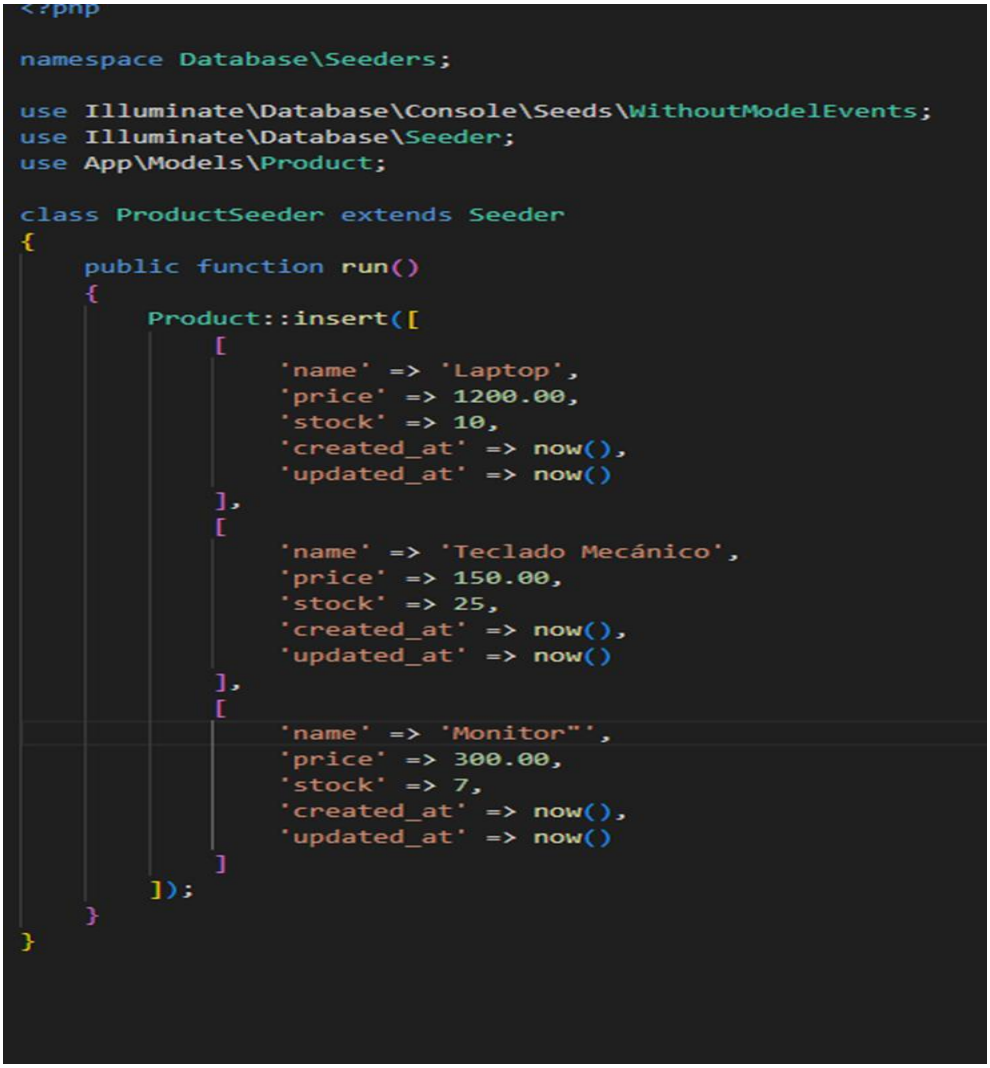
ANEXO G. DOCUMENTACIÓN TÉCNICA DEL SISTEMA

Este anexo presenta la documentación técnica del sistema, incluyendo diagramas de arquitectura, configuraciones de despliegue y aspectos clave de la infraestructura. Su objetivo es facilitar el mantenimiento y la integración futura del sistema bajo el modelo Zero Trust.

```php
<?php

namespace Database\Seeders;

use App\Models\User;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     */
    public function run(): void
    {
        $this->call([
            UserSeeder::class,
            ProductSeeder::class,
        ]);
    }
}
```

Figura G1. Diagrama general de la arquitectura del sistema.

Fuente: Elaboración propia con migraciones de Laravel y Visual Studio Code (2025).



```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use App\Models\Product;

class ProductSeeder extends Seeder
{
    public function run()
    {
        Product::insert([
            [
                'name' => 'Laptop',
                'price' => 1200.00,
                'stock' => 10,
                'created_at' => now(),
                'updated_at' => now()
            ],
            [
                'name' => 'Teclado Mecánico',
                'price' => 150.00,
                'stock' => 25,
                'created_at' => now(),
                'updated_at' => now()
            ],
            [
                'name' => 'Monitor"',
                'price' => 300.00,
                'stock' => 7,
                'created_at' => now(),
                'updated_at' => now()
            ]
        ]);
    }
}
```

Figura G2. Configuración de despliegue del backend en servidor local.

Fuente: Elaboración propia con migraciones de Laravel y Visual Studio Code (2025).

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use App\Models\User;
use Illuminate\Support\Facades\Hash;

class UserSeeder extends Seeder
{
    public function run()
    {
        User::create([
            'name' => 'Cris Developer',
            'email' => 'cris@example.com',
            'password' => Hash::make('X2231dKSDmksd1234'),
        ]);
    }
}
```

Figura G3. Diagrama de interacción entre API RESTful y cliente de pruebas (Postman).
Fuente: Elaboración propia con migraciones de Laravel y Visual Studio Code (2025).