IDEAS

Innovation & Development in Engineering and Applied Science

Journal homepage: https://revistasojs.utn.edu.ec/index.php/ideas

Desarrollo de un API gateway personalizado y ligero en Java para la gestión segura de acceso y enrutamiento en APIs REST

Development of a lightweight custom API gateway in Java for secure access management and routing in REST APIs

Xavier Mauricio Rea-Peñafiel[1], José Antonio Quiña-Mera[1-2], Diego Javier Trejo-España[1]

[1] Facultad de Ingeniería en Ciencias Aplicadas, Universidad Técnica del Norte, Cdla. Universitaria El Olivo, Ibarra, Ecuador

[2] Grupo de Investigación Ciencias en Red eCIER, Universidad Técnica del Norte

Enviado: 2024-11-10, Aceptado: 2025-07-28, Publicado: 2026-01-30

Autor de correspondencia:
Xavier Rea: mrea@utn.edu.ec

DOI: 10.53358/ideas.v8i1.1171

PALABRAS CLAVE

API-Gateway,
API-Rest,
facturación electrónica,
seguridad,
aplicaciones móviles,
ingeniería de software,
sistemas de información.

RESUMEN

Esta propuesta describe el diseño y desarrollo de un producto de software en Java que implemente un API Gateway personalizado para gestionar la seguridad y control de acceso en lugar de utilizar un sistema de tokens JWT. La propuesta incluye la definición de requerimientos funcionales, el diseño de software y su arquitectura, la implementación y la puesta en marcha del sistema, con el objetivo de ofrecer una solución segura y eficiente para la comunicación y control entre los clientes y una API REST. Se aplicó la metodología de Design Science Research, centrándose en el proceso de diseño e implementación del producto de software en Java que permita la integración segura entre un cliente y una API REST. La seguridad en aplicaciones web y en arquitecturas de APIs REST es fundamental para proteger la integridad, confidencialidad y disponibilidad de los sistemas en un entorno digital cada vez más expuesto a amenazas. A su vez, la metodología Scrum permite gestionar proyectos de desarrollo de software de manera ágil y colaborativa, lo cual resulta ideal para implementar soluciones de seguridad de forma iterativa y adaptativa. Luego, en la fase de diseño y codificación del software en Java, se define que el API Gateway personalizado debe gestionar solicitudes entrantes y redistribuir el tráfico a las APIs REST de manera segura. Para ello, el API Gateway personalizado implementa características como autenticación y validación de roles de usuario. En la API REST se crean endpoints simples que permitan la comunicación fluida con el gateway. Esta propuesta ofrece una solución robusta para la gestión de la seguridad y el control de acceso en aplicaciones distribuidas. Al utilizar la API el sistema centraliza la seguridad y la autenticación de manera eficiente, mejorando la resistencia ante fallos y manteniendo un rendimiento estable.

KEYWORDS

API-Gateway,
API-Rest,
security,
mobile applications,
electronic invoicing,
software engineering,
information systems.

ABSTRACT

This proposal outlines the design and development of a Java-based software artifact implementing a custom API Gateway to manage security and access control, replacing the need for JWT token systems. The proposal encompasses the definition of functional requirements, software design and architecture, implementation, and system deployment, aiming to offer a secure and efficient solution for communication and control between clients and a REST API. The Design Science Research methodology was applied, focusing on the design and implementation process of the software product, which enables secure integration between a client and a REST API. Security in web applications and REST API architectures is essential to protect system integrity, confidentiality, and availability in an increasingly threat-exposed digital environment. Additionally, the Scrum methodology facilitates agile and collaborative project management, ideal for iteratively and adaptively implementing security solutions. During the design and coding phases in Java, the custom API Gateway is defined to manage incoming requests and safely route traffic to REST APIs. The custom API Gateway incorporates features such as authentication and user role validation. Simple endpoints are created on the REST API to ensure seamless communication with the gateway. This proposal provides a robust solution for managing security and access control in distributed applications. By using the API Gateway, the system centralizes security and authentication efficiently, enhancing fault tolerance and maintaining stable performance.

1. Introducción

1.1. *La seguridad en las aplicaciones web y sus recursos*

En un entorno digital en constante evolución, la seguridad en las aplicaciones web ha cobrado una importancia crítica, especialmente debido al incremento de amenazas y vulnerabilidades que afectan tanto a usuarios como a organizaciones [1]. Las aplicaciones modernas suelen depender de arquitecturas basadas en APIs, particularmente de APIs REST [2], que facilitan la comunicación entre distintos servicios y dispositivos, pero también introducen nuevos vectores de ataque que deben ser gestionados con mecanismos de seguridad avanzados [3]. En este contexto, la seguridad en las API REST y en las aplicaciones web se convierte en un pilar fundamental para proteger la integridad de los datos y la confidencialidad de la información durante la interacción entre el cliente y el servidor. Sin embargo, la implementación de seguridad directamente en cada API puede ser compleja y difícil de escalar.

Una solución efectiva y ampliamente adoptada es el uso de un API Gateway, una capa intermedia que, además de facilitar la gestión de tráfico y la autenticación centralizada, implementa políticas de seguridad que limitan y controlan el acceso a los distintos recursos ofrecidos por las APIs REST [4]. El API Gateway no solo actúa como un punto de control para la autenticación y la autorización, sino que también permite la detección y mitigación de amenazas en tiempo real, ofreciendo así una primera línea de defensa robusta contra ataques como la inyección de SQL, cross-site request forgery (CSRF), cross-site scripting (XSS), los ataques de denegación de servicio (DoS) y otros vectores comunes en el tráfico de aplicaciones web.

Aquí, el problema es la necesidad de controlar de manera unificada la autenticación y autorización de solicitudes en entornos distribuidos. Un API Gateway personalizado es necesario, ya que existen limitaciones en soluciones existentes [5] o tienen desafíos específicos en la seguridad y control de acceso [6] que un diseño a medida podría resolver [7]. La motivación radica en mejorar la seguridad y experiencia del usuario en sistemas que usan RESTful APIs, aprovechando la arquitectura que propone un intermediario representado por el API-Gateway.

La integración efectiva entre un API Gateway y las APIs REST subyacentes es crucial para maximizar la seguridad y garantizar un flujo de comunicación seguro, controlado y auditado. Este artículo explora los principales aspectos de seguridad en aplicaciones web, la protección en APIs REST, las ventajas y desafíos de implementar un API Gateway [8] y las mejores prácticas [9] para lograr una integración segura y eficiente entre un API Gateway y las APIs REST, contribuyendo así al desarrollo de arquitecturas de seguridad en entornos de aplicaciones web.

El objetivo principal de este trabajo y como una contribución, respecto a otros API Gateway existentes, radica en la implementación sencilla de un API Gateway personalizado utilizando Servlets en Java, beneficiándose de varias características de bajo nivel como ejecución asíncrona y en hilos separados, uso de pools de recursos y manejo

optimizado de memoria [10], lo cual permite mantener un control más exacto sobre la ejecución.

Adicionalmente, se contempla una forma de acceso sin depender de tokens JWT, que suele ser el mecanismo de autenticación ampliamente adoptado para el acceso a APIs REST. Se propone un sistema que gestiona de manera integrada la autenticación y la generación de un token de sesión validado por IP, ofreciendo una alternativa que también es segura [11] y que pueda ser utilizada en aplicaciones web.

Además, la combinación de la metodología Design Science Research (DSR) y Scrum asegura que el diseño y la implementación sean iterativos, adaptativos y centrados en resolver problemas reales de seguridad en arquitecturas REST, lo que lo convierte en una solución viable y práctica [12].

En las siguientes secciones se describen la arquitectura de la propuesta, los resultados obtenidos luego de las pruebas y las conclusiones del caso.

2. Fundamentación teórica

2.1. *Seguridad en aplicaciones web*

La seguridad en aplicaciones web se refiere al conjunto de prácticas, tecnologías y políticas destinadas a proteger las aplicaciones de ataques maliciosos que buscan vulnerar la información o las funcionalidades. Entre las amenazas comunes se encuentran ataques de inyección de SQL, cross-site scripting (XSS), cross-site request forgery (CSRF), entre otros [13]. La seguridad en aplicaciones web se basa en principios como la autenticación y autorización de usuarios, el cifrado de datos sensibles, la validación y sanitización de entradas, y la auditoría y monitoreo continuo de las actividades. Para mitigar riesgos, se recomienda el uso de técnicas como el cifrado HTTPS, el uso de tokens seguros, y la implementación de firewalls específicos para aplicaciones web (WAFs) [14] que detecten y bloqueen actividades sospechosas en tiempo real.

2.2. *Seguridad en API REST*

Las APIs REST son interfaces que permiten la comunicación entre diferentes aplicaciones de software utilizando el protocolo HTTP. Debido a su amplia adopción en la arquitectura de microservicios y sistemas distribuidos, las APIs REST se han convertido en un objetivo crítico de los ciberataques. La seguridad en APIs REST involucra prácticas específicas como la autenticación de usuarios, el uso de tokens de acceso (como JWT) [15], y la limitación de solicitudes para prevenir abusos y mantener la disponibilidad del servicio. Además, es crucial implementar controles de autorización para asegurar que cada solicitud tenga permisos adecuados, y adoptar políticas de cifrado tanto en tránsito como en reposo para proteger los datos sensibles. El uso de cabeceras de seguridad, como Content Security Policy (CSP) y CORS [16], también es importante para limitar los orígenes permitidos y mitigar el riesgo de ataques desde aplicaciones de terceros.

2.3. *Seguridad mediante API-GATEWAY*

Un API Gateway actúa como un intermediario entre los clientes y los microservicios, gestionando todas las solicitudes entrantes y aplicando políticas de seguridad centralizadas. En términos de seguridad, el API Gateway permite controlar de manera unificada la autenticación y autorización de las solicitudes, administrar tokens de acceso, y aplicar políticas de limitación de velocidad (rate limiting) para prevenir ataques de denegación de servicio (DoS) [17]. Además, el API Gateway permite la integración de servicios de logging y monitoreo [18] que proporcionan visibilidad sobre el tráfico y posibles incidentes de seguridad, permitiendo reaccionar de manera rápida ante actividades sospechosas. Esta capa de control es esencial para implementar patrones de seguridad avanzados en arquitecturas de microservicios, reduciendo el riesgo de exposición de datos y mejorando la resiliencia del sistema frente a ataques.

2.4. *Framework Scrum*

Scrum es un marco de trabajo ágil utilizado en el desarrollo de software que se basa en iteraciones llamadas sprints para entregar productos de manera incremental. En el contexto del desarrollo de software seguro, Scrum facilita la implementación de prácticas de seguridad de forma iterativa, permitiendo la integración de mejoras y pruebas de seguridad en cada sprint [19]. El equipo de desarrollo trabaja en ciclos cortos y colabora estrechamente con los interesados, lo que permite identificar y responder a requisitos de seguridad emergentes a lo largo del proyecto. Scrum incluye roles clave como el Product Owner, el Scrum Master y el equipo de desarrollo, quienes trabajan en conjunto para priorizar y ejecutar las tareas, asegurando que los requisitos de seguridad se integren en el producto de manera

continua y adaptable [20]. Además, mediante las revisiones y retrospectivas al final de cada sprint, el equipo puede ajustar las prácticas de seguridad, garantizando que el producto final sea seguro y cumpla con las expectativas de los usuarios y las normativas vigentes.

2.5. *Norma ISO 29119*

La ISO/IEC 29119 es un conjunto de estándares internacionales que define procesos, documentación y técnicas para la gestión y ejecución de pruebas de software, proporcionando un marco de trabajo integral para asegurar la calidad en el desarrollo de software [21]. Dividida en varias partes, la norma cubre desde los principios y vocabulario de pruebas hasta procesos específicos y técnicas aplicables a diferentes tipos de pruebas, como pruebas funcionales, de rendimiento y de seguridad, lo cual se adapta a las necesidades de validación funcional del API Gateway personalizado.

3. Diseño de la investigación

Se ha seleccionado la metodología Design Science Research (DSR) ya que es muy utilizada para el diseño y desarrollo de productos tecnológicos, especialmente en el campo del software [12]. La Tabla 1 presenta las fases clave de la aplicación del enfoque de DSR en el desarrollo del gateway propuesto.

Tabla 1: *Metodología para el diseño de la propuesta*

Actividad	**Componentes**
Diagnóstico del problema	Problema, objetivos
Fundamentación teórica	Seguridad en aplicaciones web
	Seguridad en API-REST
	Seguridad a través de API-GATEWAY
	Framework Scrum
Diseño del producto de software	Requerimientos, proceso de diseño, implementación
Evaluación del producto	Evaluación de la adecuación funcional

3.1. *Identificación del problema y objetivos*

Una vez que se ha definido el problema a resolver (descrito en la sección de introducción), se establecieron en esta fase los objetivos específicos que el producto de software debía cumplir:

- Diseñar una estructura modular y escalable que permita la integración fácil entre el API Gateway y el API REST.
- Implementar una política de seguridad centralizada y sencilla para la autenticación de usuarios.
- Establecer métricas para evaluar estos objetivos, como tiempos de respuesta, seguridad de los endpoints, y facilidad de integración.

Se consideraron los siguientes puntos clave y limitaciones para el diseño y desarrollo de la propuesta:

- El diseño y codificación se orientaron hacia el lenguaje de programación Java. El producto de software determina un sencillo API Gateway que gestione solicitudes entrantes y redistribuya tráfico a las APIs REST de manera segura.
- Para el API Gateway personalizado, se implementan características de autenticación basada en tokens, validación de roles de usuario, registro de solicitudes (logging).
- El API REST de prueba serán endpoints simples que permitan la comunicación fluida con el gateway.
- En esta primera etapa no se consideran funciones de balanceo de carga para distribuir las solicitudes ni limitación en la velocidad de acceso, ya que se desea mantener sencillez y, sobre todo, cubrir los aspectos de seguridad en autenticación y autorización.

3.2. *Metodología de desarrollo de software y normativas consideradas*

Scrum es un marco de trabajo ágil utilizado en el desarrollo de software que, mediante iteraciones llamadas sprints, permite la entrega incremental de productos, facilitando la implementación iterativa de prácticas de seguridad y la integración de mejoras y pruebas en cada ciclo [19]. Este enfoque promueve la colaboración estrecha entre el equipo de desarrollo, el Product Owner y el Scrum Master, permitiendo identificar y responder a requisitos de seguridad emergentes, así como ajustar prácticas mediante revisiones y retrospectivas, asegurando que el producto final cumpla con expectativas y normativas [20]. Complementariamente, la ISO/IEC 29119 proporciona un marco de trabajo integral para la gestión y ejecución de pruebas de software, definiendo procesos, documentación y técnicas que cubren desde principios generales hasta pruebas específicas como funcionales, de rendimiento y de seguridad, adaptándose a las necesidades de validación de sistemas como un API Gateway personalizado [21].

3.3. *Diseño y desarrollo del producto de software*

3.3.1. Definición de requerimientos funcionales

- RF1 - Autenticación y autorización centralizada: El API Gateway debe autenticar a los usuarios y gestionar permisos sin necesidad de tokens JWT (a diferencia de otras soluciones [22]), utilizando en su lugar credenciales almacenadas en un sistema de autenticación seguro [23].
- RF2 - Redirección y enrutamiento de solicitudes: El API Gateway deberá recibir todas las solicitudes entrantes y enrutarlas de manera adecuada a los endpoints del API REST.
- RF3 - Limitación de solicitudes y control de Flujo: Implementación de mecanismos de rate limiting para restringir el número de solicitudes permitidas por cliente en un tiempo determinado, evitando así ataques de denegación de servicio (DoS) [24].
- RF4 - Filtrado de solicitudes maliciosas: El API Gateway debe realizar una inspección básica de las solicitudes para detectar patrones sospechosos, como la inyección de código, y denegar el acceso en caso de identificar riesgos de seguridad [25].
- RF5 - Monitoreo y registro de actividades: El sistema debe registrar todas las solicitudes y respuestas para monitorear el tráfico, identificar errores y posibles incidentes de seguridad.

3.3.2. Definición de requerimientos no funcionales

- RNF1 - Escalabilidad y modularidad: La solución debe permitir añadir nuevos endpoints o ampliar las capacidades del API Gateway sin afectar su rendimiento.
- RNF2 - Bajo tiempo de respuesta: El sistema debe gestionar las solicitudes en tiempo real con un bajo tiempo de latencia.
- RNF3 - Alta disponibilidad y tolerancia a fallos: En esta primera versión, el API Gateway personalizado no soportará resiliencia ante fallos, ni mecanismos de recuperación automática, dada la complejidad inherente a dichas características.

3.3.3. Arquitectura General

La arquitectura del sistema estará basada en tres componentes principales:

1. API Gateway (Java): Actúa como el punto de entrada principal para las solicitudes, proporcionando una capa de seguridad y control.
2. API REST (Java): Servidor backend que expone varios endpoints para diferentes operaciones de la aplicación.
3. Base de Datos (en este caso PostgreSQL): Almacena la información de usuarios y permisos necesarios para la autenticación y autorización centralizadas en el API Gateway personalizado.

3.3.4. Diagrama de Arquitectura

La arquitectura seguirá un modelo cliente-gateway-backend [26], donde el API Gateway se comunica directamente con el API REST en función de las solicitudes recibidas. La autenticación será manejada en el gateway y almacenada en la base de datos, lo cual se conceptualiza en la Figura 1.

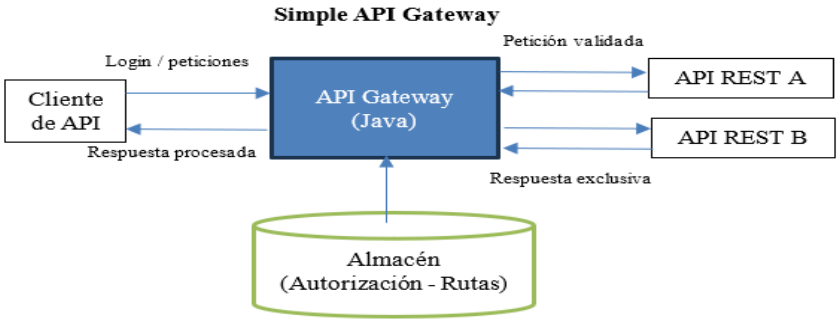

Figura 1: *Diagrama de arquitectura de la propuesta "Simple API-Gateway"*

3.3.5. Componentes clave del API Gateway

1. Autorizador: Este componente valida las credenciales de usuario y verifica sus permisos según el endpoint solicitado.
2. Seguridad en el gateway: Este módulo aplicará políticas de filtrado de solicitudes.
3. Controlador de enrutamiento en el gateway: Define las rutas y redirige las solicitudes hacia los endpoints adecuados en la API REST.
4. API REST Backend: Contiene los endpoints para la lógica de negocio, configurado para responder exclusivamente a solicitudes del API Gateway. Si bien esta característica pudiera limitar el acceso a los diferentes backends, adiciona un nivel de seguridad a APIs de tipo empresarial, donde se requiere un acceso exclusivo.

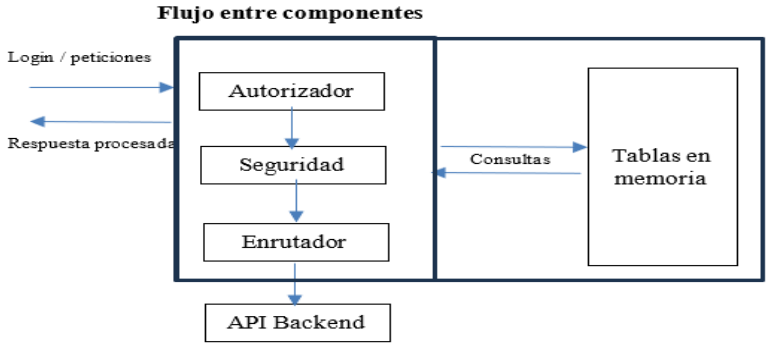

Figura 2: *Componentes internos del API Gateway*

Todos los componentes del API Gateway (que se muestran en la Figura 2) han sido implementados con la tecnología de Java – Servlets, lo cual implica el aprovechamiento de las mejoras en las características de este tipo de componentes como uso de memoria, gestión de hilos de ejecución y técnicas anti hacking:

- Pool de conexiones y caching: Se hace uso del pool de conexiones (en este caso del servidor de aplicaciones Wildfly) para la reutilización de objetos y reducción del uso de memoria, el manejo automático de destrucción de objetos que ya no están siendo utilizados y del recolector de basura.

- Hilos de ejecución (threading): Cada solicitud HTTP hacia los servlets se procesa con un hilo de ejecución separado dentro del Thread Pool gestionado por Wildfly y donde se puede configurar el máximo de hilos para evitar un uso excesivo de CPU o memoria.
- Técnicas anti hacking: Ya que los servlets soportan la generación de tokens CSRF, protecciones ante ataques XSS e inyección de SQL, el API Gateway implementa estas funcionalidades. También se ha implementado la técnica de restricción de solicitudes (rate limit).

3.3.6. Implementación y SCRUM

Para implementar un sistema seguro y eficiente de integración entre un API Gateway personalizado en Java y un API REST, se ha diseñado una planificación estructurada utilizando el marco de trabajo ágil Scrum. Esta metodología permite dividir el proyecto en sprints iterativos, cada uno con objetivos específicos que van desde la configuración del entorno y la definición de componentes de seguridad hasta la integración completa del API Gateway y el API REST. A través de estos sprints se pudo tener una implementación ágil, adaptable y orientada a resultados. Esta planificación garantizó la entrega de incrementos funcionales en cada iteración. La planificación de implementación en Scrum fue la detallada en la Tabla 2:

Tabla 2: *Sprints definidos para la implementación*

Sprint	**Tareas**
Sprint 1: Configuración Inicial y Preparación del Entorno	– Configurar entorno de desarrollo en Java y JAX-RS en el IDE Eclipse. – Configurar base de datos PostgreSQL y crear las tablas iniciales para usuarios y roles. – Definir estructura del proyecto, paquetes y librerías necesarias.
Sprint 2: Desarrollo del API Gateway - Autenticación y Seguridad Básica	– Implementar autenticación básica en el API Gateway con validación de credenciales en la base de datos. – Configurar autorización y validación de roles en el API Gateway para distintos tipos de acceso.
Sprint 3: Desarrollo de la API REST - Endpoints y Lógica de Negocio	– Crear endpoints básicos en el API REST para el manejo de la lógica de negocio. – Configurar el API REST para que solo acepte solicitudes provenientes del API Gateway. – Implementar validaciones básicas en los endpoints para garantizar la seguridad y consistencia de los datos.
Sprint 4: Enrutamiento y Integración entre API Gateway y API REST	– Configurar enrutamiento en el API Gateway para redirigir solicitudes a los endpoints del API REST. – Realizar pruebas para verificar la correcta comunicación entre el API Gateway y el API REST. – Ajustar los controles de seguridad y acceso en función de los resultados de las pruebas.
Sprint 5: Mejora de Seguridad y Control de Flujo	– Implementar políticas de rate limiting en el API Gateway para controlar la cantidad de solicitudes. – Añadir filtrado de solicitudes para detectar patrones de ataques comunes en el API Gateway. – Ajustar el sistema de registro para capturar eventos sospechosos o incidentes de seguridad.
Sprint 6: Pruebas Finales	– Realizar pruebas de carga y rendimiento en el API Gateway y el API REST. – Documentar la arquitectura, configuración de seguridad y flujo de datos.

En el fragmento de código fuente de la Figura 3, corresponde a la implementación del API Gateway propuesto, puede observarse una estructura modular y sencilla:

```
/**
* Implementacion de la clase Servlet SimpleApiGateway
* @author ...
*/
@WebServlet("/api/*")
public class SimpleApiGateway extends HttpServlet {
 // . . .
 private ManagerAutorizacion managerAutorizacion;
 private FiltroSeguridad filtroSeguridad;
 private ControladorRutas controladorRutas;

 @Override
 public void init() throws ServletException {
  //inicialización de componentes
  //...
```

Figura 3: *Declaración de componentes internos del API Gateway*

```
// Aplicar headers de seguridad HTTP
setSecurityHeaders(resp);
// 1. Validación de seguridad
if(!filtroSeguridad.validarRequest(req)) {
 filtroSeguridad.logSecEvent(req, "Acceso no autorizado");
 resp.sendError(SC_FORBIDDEN, "Acceso denegado");
 return;
}
// 2. Autenticación y autorización
String token = req.getHeader("X-Session-Token");
String path = req.getRequestURI();
String method = req.getMethod();
if(!managerAutorizacion.isPublicEndpoint(path, method)) {
 if(token==null || !managerAutorizacion.validarToken(token)) {
  resp.sendError(SC_UNAUTHORIZED, "Token inválido");
  return;
 }
 if(!managerAutorizacion.hasPermission(token, path, method)) {
  resp.sendError(SC_FORBIDDEN, "Permisos insuficientes");
  return;
 }
}
// Generar nuevo token CSRF para solicitudes exitosas
String csrfToken=filtroSeguridad.genCSRFToken(req.getSession());
resp.setHeader("X-CSRF-Token", csrfToken);
// 3. Enrutamiento seguro
try {
 controladorRutas.routeRequest(req, resp);
} catch (Exception e) {
 filtroSeguridad.logSecEvent(req, "Error enrutamiento: " + e.getMessage());
 resp.sendError(SC_INTERNAL_SERVER_ERROR, "Error enrutamiento");
}
...
private void setSecurityHeaders(HttpServletResponse resp) {
 resp.setHeader("X-XSS-Protection", "1; mode=block");
 resp.setHeader("Content-Security-Policy", "default-src 'self'");
 resp.setHeader("X-Content-Type-Options", "nosniff");
 resp.setHeader("Strict-Transport-Security", "max-age=31536000; includeSubDomains");
 resp.setHeader("X-Frame-Options", "DENY");
}
```

Figura 4: *Extracto del flujo principal del API Gateway propuesto*

El código de la figura 4 muestra un enfoque de seguridad mediante múltiples capas de protección. En primer lugar, se aplican headers de seguridad HTTP (como X-XSS-Protection, CSP y HSTS) para mitigar ataques comunes como XSS, sniffing o clickjacking. Luego, el flujo principal valida cada solicitud en tres etapas críticas: (1) un filtro de seguridad (`filtroSeguridad.validarRequest`) rechaza peticiones malformadas o no autorizadas; (2) el gestor de autorización (`managerAutorizacion`) verifica la autenticidad del token de sesión (X-Session-Token) y los permisos del usuario, distinguiendo entre endpoints públicos y privados; y (3) se genera un token CSRF para prevenir falsificación de solicitudes en operaciones exitosas. Además, aunque de una manera básica, se registran eventos de seguridad (ej. accesos denegados) para auditoría. Estas implementaciones siguen prácticas como el principio de menor privilegio y defensa en profundidad, asegurando que el enrutamiento (`controladorRutas.routeRequest`) solo procese solicitudes validadas y autorizadas.

4. Resultados

La evaluación es fundamental en DSR y debe hacerse comparando el producto con los objetivos y mediante pruebas de rendimiento y seguridad. En la Figura 4 puede notarse la importancia de los aspectos de seguridad en el API Gateway. Para cuantificar las pruebas de carga descritas en la Tabla 3, y de acuerdo con la norma ISO/IEC 29119, se sigue un proceso estructurado que permite medir y documentar resultados de manera precisa y reproducible, los cuales se resumen en la Tabla 4.

Tabla 3: *Métricas de evaluación de rendimiento y seguridad*

Métrica	**Descripción**	**Cálculo**	**Interpretación**
Tiempo de respuesta	Es el tiempo que el sistema tarda en responder a una solicitud desde el momento en que se envía hasta que el cliente recibe la respuesta. TF: tiempo de finalización TI: tiempo inicial	TR = TF - TI	Un tiempo de respuesta bajo sugiere eficiencia; los tiempos elevados pueden indicar cuellos de botella en el sistema.
Latencia o tiempo de espera	Es el tiempo que una solicitud espera en la cola antes de ser procesada por el sistema. Se mide desde que la solicitud llega al servidor hasta que el sistema comienza a procesarla. TPI: tiempo inicial de procesamiento TR: tiempo de recepción de la petición	L = TPI - TR	Alta latencia indica que el sistema está sobrecargado o que hay ineficiencias en la cola de procesamiento.
Tasa de solicitudes por segundo	Es la cantidad de solicitudes que el sistema puede procesar por segundo, una métrica clave para medir la capacidad del sistema. TSP: tasa de solicitudes procesadas TTP: tiempo de procesamiento utilizado	T = TSP / TTP	Un valor alto sugiere que el sistema puede manejar grandes volúmenes de tráfico, pero un valor que cae bajo una carga incrementada puede indicar un límite de rendimiento.
Tasa de errores	Es el porcentaje de solicitudes que fallan debido a errores (códigos HTTP 4xx, 5xx) SF: Solicitudes fallidas TS: Total de solicitudes	TE = (SF / TS) * 100	Una tasa de errores baja es ideal; una tasa alta puede significar que el sistema no puede manejar la carga actual o que hay problemas de estabilidad.
Uso de recursos del sistema	Mide el consumo de CPU y memoria durante la prueba, ya que estos recursos son críticos para la escalabilidad y el rendimiento del sistema. TCPU: tiempo de CPU usado TT: tiempo total medido MU: cantidad de memoria utilizada MT: cantidad de memoria total	%CPU = TCPU/TT	Un uso elevado y sostenido de CPU/memoria puede indicar que el sistema necesita optimización; picos significativos sugieren que ciertos procesos son muy intensivos en recursos.
HTTPS	Uso del protocolo seguro HTTPS	Soporta / No soporta	No utilizar HTTPS implica una transmisión de información sobre un canal vulnerable.
XSS	Protección ante ataques XSS (Cross-Site Scripting)	Soporta / No soporta	Evitar que se inserten scripts JavaScript, HTML u otros tipos de código en una aplicación web.
Inyección SQL	Protección ante ataques de inyección SQL	Soporta / No soporta	Evitar ataques de inyección SQL
CSRF	Falsificación de solicitud entre sitios	Soporta / No soporta	Evitar ataques web que engañan a un usuario para que realice acciones no deseadas en una aplicación en la que ya está autenticado
Rate limiting	Protección mediante limitación a la tasa de peticiones	Soporta / No soporta	Evitar ataques DoS mediante la limitación de llamadas.

Tabla 4: *Resultados de las mediciones del API Gateway*

Métrica	**Resultado**	**Objetivo**	**Estado**
Tiempo promedio de respuesta (ms)	120	150	OK
Latencia promedio (ms)	100	150	OK
Peticiones por segundo	500	450	OK
Total de peticiones procesadas	100000	100000	OK
Tasa de errores	0.5 %	0.5 %	OK
Uso de CPU	65 %	90 %	OK
Uso de memoria (Mb)	524	800	OK
Máximo número de usuarios concurrentes	250	200	OK
Uso de HTTPS	Soporta	Soporta	OK
Protección ante XSS	Soporta	Soporta	OK
Protección ante Inyección SQL	No soporta	Soporta	Falla
Protección ante CSRF	Soporta	Soporta	OK
Protección mediante limitación de peticiones	Soporta	Soporta	OK

5. Discusión

Esta primera versión del producto de software denominado Simple API Gateway (SAG) mantiene un rendimiento adecuado para ser utilizado en un entorno controlado. Si bien no se lo ha comparado con otros API Gateways basados en Java (como Spring Cloud Zuul [27] o Spring Cloud Gateway) que son productos enterprise-class, podemos tomar varias referencias cuantitativas, aclarando que esta propuesta denominada SAG se presenta como una alternativa frente al uso directo de JWT para los accesos a APIs de tipo REST.

Tiempo de respuesta y latencia: Los tiempos de respuesta promedio cumplen con los objetivos definidos, sugiriendo que el sistema mantiene una eficiencia estable bajo carga. La latencia también se mantiene dentro de los límites, lo que indica una gestión efectiva del procesamiento.

Tasa de solicitudes por segundo: El API Gateway soporta 500 solicitudes por segundo, superando el objetivo establecido de 450, lo que evidencia su capacidad para manejar tráfico elevado sin comprometer el rendimiento. Esta cantidad de respuestas puede ser alta debido a que SAG directamente permite el acceso mediante una lista blanca de IPs y un token, lo que le da una pequeña ventaja frente a JWT u otros tipos de acceso seguro [28]. Se utilizó la herramienta Apache JMeter para obtener las mediciones respectivas [29].

Tasa de errores: La tasa de errores se mantiene baja, en un 0.5 %, lo que es favorable para la estabilidad del sistema y asegura una alta disponibilidad bajo cargas normales y moderadas.

Uso de recursos (CPU y Memoria): Aunque el uso promedio de CPU es aceptable, el pico de memoria supera el umbral establecido (1 Mb máximo asignado), indicando un posible ajuste en la gestión de recursos u optimización en el manejo de procesos concurrentes.

Número máximo de usuarios concurrentes: El sistema soporta hasta 250 usuarios concurrentes, superando el objetivo de 200, lo cual es indicativo de una arquitectura robusta y bien escalada para soportar concurrencia.

Un criterio que hay que destacar es que para la implementación se usaron los componentes Java de tipo Servlets, lo cual permite tener una alta optimización por ser componentes web de bajo nivel [30]. Probablemente otras implementaciones realizadas en lenguajes como Lua o Jolie sean más sencillas en su arquitectura [31], pero no son lenguajes tan comunes como Java.

Criterios de seguridad: El API Gateway implementa las protecciones indicadas en la tabla 4 porque es el único punto de entrada visible desde el exterior, las APIs internas pueden tener configuraciones inseguras y centralizar la seguridad reduce puntos de fallo. Al momento, el API Gateway no implementa una funcionalidad contra la inyección SQL, debido a que no accede directamente a bases de datos, sino a configuraciones y listados de información precargada en memoria, pero debe implementarse obligatoriamente en la siguiente fase.

6. Conclusiones y posibles trabajos futuros

- Los resultados de las pruebas demuestran que el API Gateway personalizado en Java cumple con los objetivos de rendimiento, mostrando tiempos de respuesta y latencia que permiten una experiencia fluida para los usuarios.

La capacidad del sistema para manejar hasta 250 usuarios concurrentes y una tasa de 200 solicitudes por segundo sugiere que la arquitectura es adecuada para cargas significativas.

- La baja tasa de errores indica una alta disponibilidad del sistema bajo condiciones de carga, lo cual es positivo para entornos de producción. Sin embargo, la superación del límite de memoria en picos sugiere la necesidad de optimizaciones en la gestión de recursos para asegurar una escalabilidad más robusta.
- La implementación de límites de tasa ayudará a prevenir abusos y ataques de denegación de servicio (DoS). Un sistema de rate limiting configurable permitirá definir límites en función de la IP, usuario o endpoint, controlando así el flujo de tráfico de manera más precisa.
- El soporte contra ataques XSS, CSRF y el uso de HTTPS hace que el API Gateway propuesto tenga un funcionamiento más seguro y sea confiable hacia los diversos clientes.
- La ausencia de funcionalidades como logging y balanceo de carga plantea limitaciones en la capacidad del API Gateway para gestionar situaciones críticas y responder a picos de carga. La falta de un sistema de logging también limita la visibilidad de los eventos y dificulta la identificación de problemas en tiempo real.

Trabajos Futuros

- La incorporación de un sistema de logging centralizado permitirá capturar información de todas las solicitudes, respuestas y eventos relevantes, facilitando el monitoreo y la detección de anomalías en tiempo real. Esto también permitirá auditar las actividades para mejorar la seguridad y el diagnóstico de problemas.
- Desarrollar una capacidad de balanceo de carga permitirá distribuir uniformemente las solicitudes entre instancias de servidores backend, mejorando la respuesta en momentos de alta concurrencia y previniendo la sobrecarga de un único servidor.
- Dado el uso elevado de memoria en picos, es recomendable ajustar la gestión de recursos, posiblemente mediante el uso de técnicas de optimización de concurrencia o el ajuste en el manejo de conexiones. Esto contribuirá a una mayor estabilidad en situaciones de máxima carga.
- Incorporar una capa de caching para respuestas frecuentes reduciría el tiempo de procesamiento en solicitudes redundantes, optimizando la latencia y el uso de CPU y memoria.
- Migrar el proyecto de código fuente hacia Java 22 y comprobar las nuevas actualizaciones en las políticas de acceso y permisos, algoritmos de criptografía implementados y utilizar las herramientas para verificar la integridad de las bibliotecas y módulos utilizados en las aplicaciones.
- Implementar funcionalidades para evitar inyección SQL, de manera que no se propague este ataque hacia el backend de APIs.

Estas mejoras permitirán al API Gateway no solo ser más eficiente y escalable, sino también mejorar la resiliencia y capacidad de monitoreo, requisitos funda-mentales para entornos de producción complejos.

Referencias

[1] M. Saad, A. Zia, M. Raza, M. Kundi, and M. Haleem, "A comprehensive analysis of healthcare websites usability features, testing techniques and issues," *IEEE Access*, vol. 10, pp. 97 701–97 718, 2022.

[2] I. Ahmad, E. Suwarni, R. I. Borman, Asmawati, F. Rossi, and Y. Jusman, "Implementation of RESTful API web services architecture in takeaway application development," in *2021 1st International Conference on Electronic and Electrical Engineering and Intelligent System (ICE3IS)*, Oct. 2021, pp. 132–137.

[3] M. J. Haber, B. Chappell, and C. Hills, "Attack vectors," in *Cloud Attack Vectors: Building Effective Cyber-Defense Strategies to Protect Cloud Resources*, M. J. Haber, B. Chappell, and C. Hills, Eds. Berkeley, CA: Apress, 2022, pp. 117–219.

[4] J. Gough, D. Bryant, and M. Auburn, *Mastering API Architecture: Design, Operate, and Evolve API-Based Systems*. O'Reilly Media, Inc., 2021.

[5] M. Tomić, V. Dimitrieski, M. Vještica, R. Župunski, A. Jeremić, and H. Kaufmann, "Towards applying api gateway to support microservice architectures for embedded systems," 2020.

[6] C. K. Rudrabhatla, "Security design patterns in distributed microservice architecture," *arXiv*, Aug. 2020.

[7] E. Ünsal, B. Öztekin, M. Çavuş, and S. Özdemir, "Building a fintech ecosystem: Design and development of a fintech api gateway," in *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, Oct. 2020, pp. 1–5.

[8] X. Zuo, Y. Su, Q. Wang, and Y. Xie, "An api gateway design strategy optimized for persistence and coupling," *Advances in Engineering Software*, vol. 148, p. 102878, Oct. 2020.

[9] S. K. Shivakumar, "Modern web integration patterns," in *Modern Web Performance Optimization: Methods, Tools, and Patterns to Speed Up Digital Platforms*, S. K. Shivakumar, Ed. Berkeley, CA: Apress, 2020, pp. 327–357.

[10] Eclipse Foundation, "Jakarta servlet specification," [Online], 2025, accedido: 14 de marzo de 2025. [Online]. Available: https://jakarta.ee/specifications/servlet/6.0/jakarta-servlet-spec-6.0

[11] B. J. Chelliah, K. Sathish, and S. A. Kumar, "Service selection in service oriented architecture using probabilistic approach and asynchronous queues with interceptor validation," *International Journal of Electrical & Computer Engineering (2088-8708)*, vol. 10, no. 1, 2020.

[12] J. vom Brocke, A. Hevner, and A. Maedche, "Introduction to design science research," in *Design Science Research. Cases*, J. vom Brocke, A. Hevner, and A. Maedche, Eds. Cham: Springer International Publishing, 2020, pp. 1–13.

[13] OWASP, "Introduction - owasp top 10:2021," [Online], 2021, accedido: 1 de noviembre de 2024. [Online]. Available: https://owasp.org/Top10/A00_2021_Introduction/

[14] R. A. Muzaki, O. C. Briliyant, M. A. Hasditama, and H. Ritchi, "Improving security of web-based application using modsecurity and reverse proxy in web application firewall," in *2020 International Workshop on Big Data and Information Security (IWBIS)*, Oct. 2020, pp. 85–90.

[15] A. F. Nugraha, H. Kabetta, I. K. S. Buana, and R. B. Hadiprakoso, "Performance and security comparison of json web tokens (jwt) and platform agnostic security tokens (paseto) on restful apis," in *2023 IEEE International Conference on Cryptography, Informatics, and Cybersecurity (ICoCICs)*, Aug. 2023, pp. 15–22.

[16] U. Kishnani and S. Das, "Securing the web: Analysis of http security headers in popular global websites," *arXiv*, Oct. 2024.

[17] Y. Dawei, G. Yang, H. Wei, and L. Kai, "Design and achievement of security mechanism of api gateway platform based on microservice architecture," *Journal of Physics: Conference Series*, vol. 1738, no. 1, p. 012046, Jan. 2021.

[18] A. Kondam, "Event-driven api gateways: Enabling real-time communication in modern microservices architectures," no. 2, 2024.

[19] E. D. H. Rafael, "Implementación de sistema de biblioteca basado en scrum para el manejo de libros en la facultad de ciencias de la comunicación de la universidad nacional del centro del perú - huancayo, 2024," Tesis de maestría, Universidad Nacional del Centro del Perú, Aug. 2024, accedido: 1 de noviembre de 2024. [Online]. Available: http://repositorio.uncp.edu.pe/handle/20.500.12894/11296

[20] G. S. Lampe, M. Olaru, M. Mafte, and C. Ilie, "Information security management system and cyber security strategy implementation in the context of scrum," in *7th BASIQ International Conference on New Trends in Sustainable Business and Consumption*, Aug. 2021, pp. 811–819.

[21] C. Patrício, R. Pinto, and G. Marques, "A study on software testing standard using iso/iec/ieee 29119-2: 2013," in *Recent Advances in Intelligent Systems and Smart Applications*, M. Al-Emran, K. Shaalan, and A. E. Hassanien, Eds. Cham: Springer International Publishing, 2021, pp. 43–62.

[22] F. D. Cas, "A practical approach to enhance web apis security using a stateless, open-source, pluggable api gateway," Master's thesis, Politecnico di Milano, Oct. 2023, accedido: 1 de noviembre de 2024. [Online]. Available: https://www.politesi.polimi.it/handle/10589/208974

[23] X. Wang, Z. Yan, R. Zhang, and P. Zhang, "Attacks and defenses in user authentication systems: A survey," *Journal of Network and Computer Applications*, vol. 188, p. 103080, Aug. 2021.

[24] L. F. Eliyan and R. D. Pietro, "Dos and ddos attacks in software defined networks: A survey of existing solutions and research challenges," *Future Generation Computer Systems*, vol. 122, pp. 149–171, Sep. 2021.

[25] K. F. M. Córdova and L. G. S. Ortiz, "Evaluación del rendimiento de firewalls de aplicaciones web open source," Master's thesis, Universidad Nacional Pedro Ruiz Gallo, Mar. 2024, accedido: 1 de noviembre de 2024. [Online]. Available: http://repositorio.unprg.edu.pe/handle/20.500.12893/13572

[26] C. Richardson, "Microservices pattern: Pattern: Api gateway / backends for frontends," microservices.io, 2024, accedido: 1 de noviembre de 2024. [Online]. Available: http://microservices.io/patterns/apigateway.html

[27] Q. Xiong and W. Li, "Design and implementation of microservices gateway based on spring cloud zuul," in *CIBDA 2022; 3rd International Conference on Computer Information and Big Data Applications*, Mar. 2022, pp. 1–5, accedido: 24 de abril de 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9899125

[28] J. T. Zhao, S. Y. Jing, and L. Z. Jiang, "Management of api gateway based on microservice architecture," *Journal of Physics: Conference Series*, vol. 1087, no. 3, p. 032032, Sep. 2018.

[29] Apache Foundation, "Apache jmeter - apache jmeter," [Online], 2024, accedido: 1 de noviembre de 2024. [Online]. Available: https://jmeter.apache.org/

[30] JakartaEE, "Jakarta servlet - jakarta ee tutorial - jakarta ee documentation," [Online], 2024, accedido: 1 de noviembre de 2024. [Online]. Available: https://jakarta.ee/learn/docs/jakartaee-tutorial/current/web/servlets/servlets.html

[31] F. Montesi and J. Weber, "Circuit breakers, discovery, and api gateways in microservices," *arXiv*, Sep. 2016.