


# CONESCAPANHONDURAS2025paper98.pdf

 Institute of Electrical and Electronics Engineers (IEEE)

---

## Document Details

### Submission ID

trn:oid:::14348:477756668

### Submission Date

Jul 31, 2025, 10:43 PM CST

### Download Date

Aug 12, 2025, 2:57 PM CST

### File Name

CONESCAPANHONDURAS2025paper98.pdf

### File Size

518.1 KB

6 Pages




4,543 Words

26,819 Characters

# 8% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

## Top Sources

- 8%  Internet sources
- 4%  Publications
- 0%  Submitted works (Student Papers)

## Integrity Flags




### 0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

## Top Sources

- 8%  Internet sources
- 4%  Publications
- 0%  Submitted works (Student Papers)

## Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Internet	ieecp-conference.org	2%
2	Internet	www.mdpi.com	<1%
3	Internet	repositorium.sdum.uminho.pt	<1%
4	Internet	hdl.handle.net	<1%
5	Internet	journalinstal.cattleyadf.org	<1%
6	Internet	rinacional.tecnm.mx	<1%
7	Publication	Mohannad Alhanahnah, Qiben Yan. "Towards best secure coding practice for imp...	<1%
8	Internet	www.ijcaonline.org	<1%
9	Internet	dblp.org	<1%
10	Internet	repository.kpi.kharkov.ua	<1%
11	Internet	icet.ac.in	<1%

12	Internet	code.tutsplus.com	<1%
13	Internet	courses.csail.mit.edu	<1%
14	Internet	fatcat.wiki	<1%
15	Internet	ijsrcseit.com	<1%
16	Internet	bithouseco.home.blog	<1%
17	Internet	patents.google.com	<1%
18	Internet	www.europarl.europa.eu	<1%
19	Internet	www.ijfmr.com	<1%
20	Internet	github.com	<1%
21	Internet	jurnal.ugm.ac.id	<1%
22	Internet	prezi.com	<1%
23	Internet	tesis.ipn.mx	<1%

# Integration of JSON Web Tokens in Spring Security: An Approach to Protecting RESTful APIs

1

line 1: 1<sup>st</sup> Given Name Surname  
line 2: dept. name of organization  
(of Affiliation)  
line 3: name of organization  
(of Affiliation)  
line 4: City, Country  
line 5: email address or ORCID

line 1: 2<sup>nd</sup> Given Name Surname  
line 2: dept. name of organization  
(of Affiliation)  
line 3: name of organization  
(of Affiliation)  
line 4: City, Country  
line 5: email address or ORCID

line 1: 3<sup>rd</sup> Given Name Surname  
line 2: dept. name of organization  
(of Affiliation)  
line 3: name of organization  
(of Affiliation)  
line 4: City, Country  
line 5: email address or ORCID

**Abstract**—This work proposes the integration of JSON Web Tokens (JWT) with Spring Security to protect RESTful APIs. It uses stateless authentication with custom filters for token validation and access control. The solution, implemented in Spring Boot 3, enhances security against common threats and reduces server load. Results show low latency and good scalability in modern web applications

**Keywords**— JWT, Spring Security, REST API, authentication, access control.)

## I. INTRODUCTION

Las interfaces de programación de aplicaciones basadas en arquitectura REST (RESTful APIs) se han consolidado como componentes esenciales en el desarrollo de aplicaciones web modernas. Su naturaleza ligera y flexible permite la comunicación entre sistemas distribuidos, móviles y en la nube, lo que ha impulsado su adopción masiva en entornos corporativos y de consumo. Sin embargo, esta apertura y accesibilidad también las convierte en objetivos frecuentes de ataques informáticos, como el acceso no autorizado, la manipulación de datos o el secuestro de sesiones [1], [2]. En este escenario, el diseño de mecanismos de autenticación y control de acceso robustos y eficientes resulta fundamental para garantizar la seguridad de las aplicaciones que exponen servicios a través de la web.

Los esquemas de autenticación tradicionales, basados en sesiones almacenadas del lado del servidor, presentan múltiples limitaciones en arquitecturas distribuidas y escalables. El mantenimiento de sesiones en memoria o bases de datos centralizadas no solo introduce cuellos de botella, sino que también contradice el principio de statelessness que caracteriza a REST [3]. Para abordar estas deficiencias, han surgido mecanismos alternativos como los JSON Web Tokens (JWT), los cuales permiten transmitir información sobre la identidad del usuario y sus privilegios de forma segura, compacta y sin requerir almacenamiento en el servidor [4], [5].

Spring Security se ha consolidado como uno de los frameworks más robustos y flexibles para implementar autenticación y autorización en aplicaciones Java. Su integración con JWT permite desarrollar sistemas de autenticación sin estado, adaptables a microservicios y fácilmente integrables con aplicaciones cliente [6], [7]. No obstante, implementar una solución segura y eficiente con JWT en Spring Security requiere enfrentar desafíos técnicos que van desde la configuración adecuada de filtros personalizados hasta la prevención de ataques por repetición, suplantación o manipulación de tokens [8], [9].

Este artículo presenta una arquitectura práctica para la integración de JWT con Spring Security, diseñada y validada en un entorno funcional real. La propuesta incluye la implementación de filtros personalizados (OncePerRequestFilter) para interceptar solicitudes, extraer y validar tokens JWT mediante algoritmos HMAC-SHA, y establecer el contexto de autenticación del usuario. Se incorpora además una gestión explícita del tiempo de expiración de los tokens, control de acceso basado en roles y manejo de errores mediante respuestas estandarizadas para accesos no autorizados.

La solución se desarrolló con Java 17 y Spring Boot 3, y fue probada a través de herramientas como Postman para validar las credenciales y observar la emisión y reutilización segura de tokens. El sistema también permitió analizar problemas comunes como errores 403 Forbidden, excepciones por dependencias faltantes y advertencias de desuso en versiones recientes de Spring Security. Estas incidencias fueron resueltas mediante buenas prácticas y configuración manual de componentes de seguridad.

Finalmente, se discuten recomendaciones para entornos de producción, como el uso de cookies HttpOnly, rotación de claves y la posible integración con gateways o mecanismos OAuth2 [10], [11]. Esta experiencia práctica busca aportar una guía reproducible, segura y escalable para la protección de APIs RESTful modernas.

Adicionalmente, la adopción de JWT responde a las necesidades de arquitecturas modernas como los sistemas basados en microservicios, donde cada servicio puede validar tokens de forma autónoma sin depender de una sesión centralizada. Esta capacidad de verificación descentralizada reduce la carga del servidor y mejora la escalabilidad horizontal, lo cual es esencial en entornos distribuidos y altamente disponibles [11]. No obstante, esta independencia también implica una mayor responsabilidad en el diseño de filtros de seguridad eficientes, ya que cualquier error en la validación del token podría comprometer la integridad del sistema.

Por otro lado, el rendimiento sigue siendo un factor determinante en la seguridad efectiva de aplicaciones web, en que la firma criptográfica del token, el algoritmo elegido (como HMAC-SHA512) y la longitud del JWT impactan directamente en el desempeño del sistema [12]. Este trabajo considera dichas variables al implementar una solución que equilibra velocidad de autenticación con solidez criptográfica. En conjunto, esta propuesta busca ofrecer una alternativa accesible, reproducible y alineada con buenas prácticas para desarrolladores que requieren proteger servicios RESTful en

entornos productivos sin comprometer ni rendimiento ni seguridad [5], [8], [9]

## II. ENFOQUES PREVIOS EN AUTENTICACIÓN CON JWT

### A. Integraciones generales con JWT y Spring

El uso de JSON Web Tokens (JWT) como mecanismo de autenticación ha sido ampliamente explorado en la literatura reciente, especialmente en el contexto de arquitecturas web distribuidas y sin estado. Dimitrijević et al. [6] ofrecen una revisión integral de los mecanismos de seguridad disponibles en el ecosistema Spring, incluyendo JWT, OAuth, LDAP y Keycloak. Si bien presentan comparaciones funcionales entre estas tecnologías, su enfoque se mantiene en un plano descriptivo, sin una implementación detallada que ilustre los desafíos prácticos que enfrenta un desarrollador al integrar JWT con Spring Security.

En una línea más aplicada, Rapolu [3] examina la implementación de JWT en arquitecturas de microservicios, particularmente cuando se emplean gateways como punto de entrada para la validación de tokens. Su trabajo destaca la utilidad de JWT en sistemas distribuidos, pero no profundiza en los mecanismos internos de validación, el diseño de filtros personalizados ni el control de acceso por roles. En contraste, el presente trabajo busca cubrir precisamente ese vacío técnico.

### B. Desempeño y eficiencia en autenticación basada en tokens

Desde la perspectiva del rendimiento, Soni [1] analiza el impacto de la autenticación basada en tokens sobre la latencia de los sistemas, señalando que la verificación criptográfica puede volverse un cuello de botella si no se optimizan los algoritmos utilizados. Su tesis identifica la relación directa entre el algoritmo de firma, el tamaño del token y el rendimiento general del sistema, algo que también se aborda en este artículo mediante el uso de HMAC-SHA512. Bucko et al. [4], por otro lado, proponen una mejora en el control de acceso utilizando el historial de comportamiento del usuario. Aunque innovador, su enfoque requiere una infraestructura adicional para almacenar y analizar dichos historiales, lo que dificulta su adopción inmediata en entornos productivos.

### C. Errores comunes y riesgos de seguridad en implementaciones reales

En el entorno académico, Fu et al. [7] desarrollan un proyecto universitario en el MIT enfocado en los riesgos asociados al uso incorrecto de JWT. Entre sus hallazgos destacan la falta de expiración de tokens, la reutilización de claves simétricas inseguras y el almacenamiento de tokens en localStorage, prácticas que abren la puerta a ataques como la suplantación de identidad o la manipulación del token. Estas advertencias son retomadas en el presente trabajo como parte de las recomendaciones de buenas prácticas.

De forma complementaria, Islam et al. [8] y Meng et al. [2] analizan errores comunes en la codificación de sistemas seguros en Java, enfatizando que muchas vulnerabilidades no provienen del lenguaje o framework, sino de decisiones deficientes en el diseño de seguridad. Por ejemplo, el uso de filtros genéricos sin validación robusta, la exposición de rutas sensibles o la falta de segregación de roles. Este artículo incorpora tales aprendizajes al implementar filtros personalizados en Spring Security que interceptan cada solicitud y validan el JWT de forma centralizada y segura.

### D. Modelos descentralizados y patrones de autenticación

Otras investigaciones han propuesto modelos alternativos para el uso de JWT. Ethelbert et al. [5] diseñan un esquema general de autenticación basado en tokens para aplicaciones SaaS, orientado a la gestión de usuarios en la nube. Aunque no se centra en Spring, su arquitectura destaca la necesidad de que cada componente del sistema sea capaz de validar tokens sin intervención de un servidor central. De forma similar, Barabanov y Makrushin [9] y Goes [12] presentan un análisis de patrones de autenticación y autorización en sistemas de microservicios, concluyendo que la validación descentralizada de tokens es esencial para lograr escalabilidad real.

En cuanto a la integridad de los tokens, Gawande y Meshram [10] demuestran que el uso de algoritmos como HMAC-SHA512 ofrece una mayor resistencia frente a ataques por colisión o manipulación de firma. Su propuesta, sin embargo, se limita a la generación y validación de tokens, sin integrar estas prácticas en un framework específico como Spring. Finalmente, Nawale y Gengaje [9] examinan las implicaciones de seguridad del uso de JWT en aplicaciones MERN (MongoDB, Express, React, Node.js), concluyendo que el almacenamiento inseguro y la falta de expiración siguen siendo los errores más comunes en la industria.

Tabla 1: Comparativa de enfoques previos

Trabajo	Framework usado	JWT integrado	Validación robusta	Control de acceso	Pruebas de rendimiento
Dimitrijević et al. [6]	Spring	Sí	No	Parcial	No
Rapolu [3]	Spring + Gateway	Sí	No	Parcial	No
Soni [1]	Genérico	Sí	N/A	No	Sí
Bucko et al. [4]	Web genérica	Sí	Parcial	Avanzado	No
Propuesta actual	Spring Boot	Sí	Sí	Sí	Sí

### E. Aportes diferenciales de esta propuesta

A diferencia de los trabajos anteriores, esta propuesta se fundamenta en una solución concreta, funcional y replicable en el ecosistema Spring, cuya implementación se ha desarrollado y validado en un entorno real con Java 17 y Spring Boot 3. Se aborda tanto la configuración de seguridad como el diseño de filtros personalizados (OncePerRequestFilter), el manejo seguro del token, la expiración automática y la implementación de control de acceso basado en roles. Además, se incluyen pruebas prácticas realizadas mediante herramientas como Postman, que permiten observar y documentar el comportamiento del sistema frente a peticiones válidas, errores 403, validaciones criptográficas y escenarios de acceso no autorizado.

Este trabajo también destaca por su enfoque didáctico y reproducible, orientado a facilitar la adopción de buenas prácticas en seguridad web por parte de desarrolladores e ingenieros de software. A diferencia de soluciones abstractas o centradas en plataformas propietarias, la implementación presentada utiliza exclusivamente tecnologías abiertas y ampliamente documentadas, lo que permite su integración en entornos reales sin depender de herramientas externas. Asimismo, se promueve la extensibilidad del modelo hacia arquitecturas más complejas, como el uso de JWT con

OAuth2 o la validación centralizada a través de gateways [6], posicionando esta propuesta como un punto de partida sólido para futuras ampliaciones en el desarrollo seguro de APIs RESTful.

### III. ARQUITECTURA PROPUESTA

#### A. Entorno técnico y herramientas

La solución fue implementada en un entorno de desarrollo local utilizando el sistema operativo Windows 11 Pro. Se empleó Java Development Kit (JDK) versión 17 como plataforma base y el framework Spring Boot 3 para facilitar la configuración de dependencias y estructuras del proyecto. El entorno de desarrollo integrado utilizado fue IntelliJ IDEA Community Edition, con gestión de dependencias mediante Apache Maven.

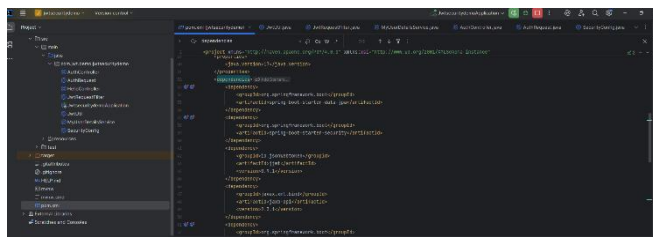


Figura 1. Configuración del entorno de desarrollo en IntelliJ IDEA

Para la autenticación mediante tokens, se integró la biblioteca jjwt versión 0.9.1, la cual permitió generar y validar JSON Web Tokens utilizando el algoritmo HMAC-SHA512. Las pruebas funcionales de los endpoints se llevaron a cabo con la herramienta Postman, que facilitó el envío de peticiones HTTP personalizadas y el análisis detallado de las respuestas. Toda la lógica de seguridad fue implementada utilizando únicamente componentes del ecosistema Spring Security, sin recurrir a herramientas externas ni plataformas propietarias, garantizando así la portabilidad y reproducibilidad del modelo propuesto.

#### B. Componentes de la solución

La arquitectura de seguridad desarrollada se compone de cinco elementos clave que trabajan de forma coordinada para ofrecer un sistema de autenticación y autorización robusto basado en JWT.

El componente principal es el controlador AuthController, responsable de exponer el endpoint POST /auth/login. Este controlador recibe las credenciales del usuario, y si son válidas, delega en JwtUtil la generación de un token JWT firmado con HMAC-SHA512, incluyendo datos como el nombre de usuario y la fecha de expiración.

El servicio MyUserDetailsService implementa la interfaz UserDetailsService de Spring Security y provee un usuario estático para efectos de prueba. Su función es proporcionar los datos necesarios para que el sistema pueda validar las credenciales y establecer la autenticación en el contexto de seguridad.

El componente JwtUtil encapsula toda la lógica relacionada con la generación, validación y extracción de datos desde el token JWT. Esta clase se encarga de firmar el token al momento de su creación y de verificar su validez al recibir solicitudes protegidas.

La clase JwtRequestFilter extiende OncePerRequestFilter y se encarga de interceptar todas las peticiones HTTP entrantes que no estén explícitamente permitidas. Este filtro extrae el token desde el encabezado Authorization, valida su firma y establece el contexto de autenticación en Spring si el token es válido.

```
if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
    UserDetails userDetails = this.userService.loadUserByUsername(username);

    if (JwtUtil.validateToken(jwt, userDetails)) {
        UsernamePasswordAuthenticationToken authToken =
            new UsernamePasswordAuthenticationToken(
                userDetails, null, userDetails.getAuthorities());
        authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
        SecurityContextHolder.getContext().setAuthentication(authToken);
    }
}

filterChain.doFilter(request, response);
```

Figura 2. Clase JwtRequestFilter.java con validación de token

Finalmente, SecurityConfig define la política de seguridad del sistema. Se configura la cadena de filtros (SecurityFilterChain), se establece una política sin estado (SessionCreationPolicy.STATELESS) y se integran tanto el proveedor de autenticación (AuthenticationProvider) como el filtro personalizado para la validación del token. Esta configuración asegura que únicamente los endpoints bajo /auth/\*\* estén exentos de autenticación, mientras que cualquier otro acceso requiera un JWT válido

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    return http
        .csrf(AbstractHttpConfigurer::disable)
        .authorizeHttpRequests((AuthorizationManagerRequestMatcher) auth -> auth
            .requestMatchers("/auth/**").permitAll()
            .anyRequest().authenticated()
        )
        .sessionManagement((SessionManagementConfigurer) httpSecurity -> sess -> sess.sessionCreationPolicy(
            SessionCreationPolicy.STATELESS)
        .authenticationProvider(authenticationProvider())
        .addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class)
        .build();
}
```

Figura 3. Clase SecurityConfig.java con filtros y política stateless

#### C. Flujo de autenticación JWT en Spring Security

El proceso de autenticación propuesto se basa en un flujo donde el usuario envía sus credenciales a través de un endpoint público expuesto en la ruta /auth/login. Este punto de entrada está diseñado para aceptar solicitudes de tipo POST con un cuerpo JSON que contiene el nombre de usuario y la contraseña. Las credenciales enviadas son autenticadas mediante el componente AuthenticationManager de Spring Security, el cual, a su vez, delega en una implementación personalizada de UserDetailsService para la validación del usuario.

Una vez validadas las credenciales, el sistema genera un token JWT utilizando una clave secreta previamente definida y un algoritmo de firma HMAC-SHA512. Este token contiene información sobre el sujeto autenticado (username), la fecha de emisión y una fecha de expiración configurada en esta implementación, establecida en diez horas. El token es enviado como respuesta al cliente, quien deberá almacenarlo temporalmente para su uso en peticiones posteriores.

Para garantizar la seguridad en cada solicitud subsecuente, se implementa un filtro personalizado mediante la clase OncePerRequestFilter. Este componente intercepta todas las peticiones entrantes y busca la presencia de un encabezado



HTTP llamado Authorization, cuyo valor debe comenzar con el prefijo Bearer seguido del token JWT. Si el encabezado está presente, el filtro extrae el token, valida su integridad y autenticidad, y, si es válido, establece el contexto de seguridad (SecurityContext) con la información del usuario autenticado. Esto permite que el resto del ciclo de procesamiento de la solicitud reconozca al usuario y aplique las restricciones de acceso correspondientes según sus roles.

La figura 5 muestra el flujo general de este proceso. En él se puede observar cómo el usuario realiza una autenticación inicial, obtiene un token JWT y luego lo utiliza como medio de autorización en llamadas posteriores a recursos protegidos:

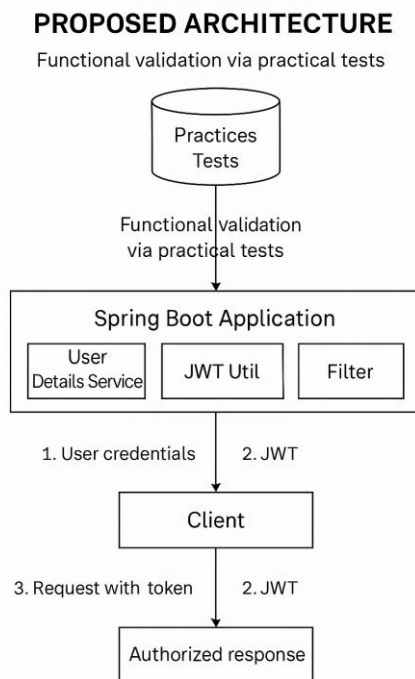


Figura 5. Diagrama del flujo de autenticación con JWT en Spring Security

#### D. Validación funcional mediante pruebas prácticas

Con el objetivo de comprobar el funcionamiento correcto de la arquitectura propuesta, se llevaron a cabo pruebas funcionales en un entorno local controlado. Estas pruebas permiten verificar no solo la generación adecuada del token JWT, sino también su uso efectivo para proteger endpoints RESTful mediante filtros personalizados de Spring Security. La validación se realizó utilizando Postman como herramienta de envío de solicitudes HTTP, ejecutando la aplicación backend en un entorno local con Java 17 y Spring Boot 3.

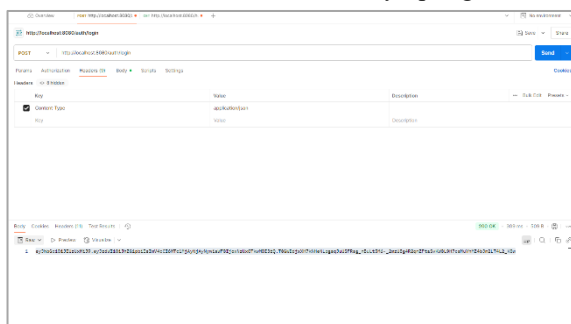


Figura 6. Generación de token en Postman

La primera prueba consistió en enviar una solicitud POST al endpoint público /auth/login, proporcionando credenciales válidas en formato JSON. Se utilizó el usuario preconfigurado admin con contraseña 1234, implementado directamente en la clase MyUserDetailsService. Como resultado, el sistema respondió con un token JWT firmado utilizando HMAC-SHA512, en cumplimiento con los estándares definidos en la clase JwtUtil. Esta respuesta incluyó el token como texto plano, lo que permitió su posterior reutilización. El código de estado HTTP fue 200 OK, y el contenido del token contenía los datos del sujeto autenticado y su tiempo de expiración.

En la segunda prueba, se validó la efectividad del token recibido. Para ello, se generó una solicitud GET al endpoint /hello, el cual está protegido por el filtro JwtRequestFilter y requiere autenticación previa. El token fue insertado en el encabezado HTTP Authorization bajo el esquema Bearer. Como resultado, el servidor reconoció el token como válido, estableció correctamente el contexto de seguridad del usuario, y respondió con el mensaje esperado: "¡Acceso autorizado con JWT!", junto a un código de estado 200 OK. Esta respuesta confirmó que el sistema validó de forma efectiva la autenticidad e integridad del token, habilitando el acceso solo si las condiciones de seguridad eran satisfechas.

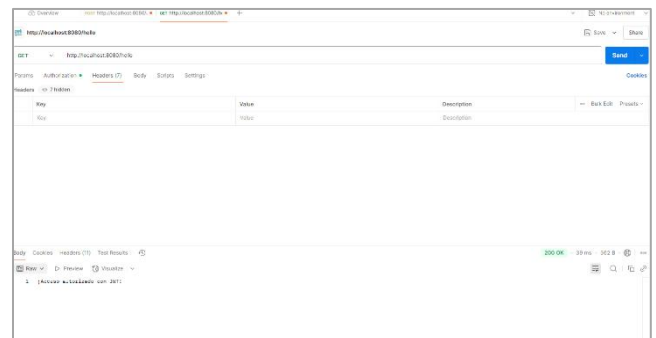


Figura 7. Acceso autorizado con JWT

También se ejecutó una tercera prueba negativa, simulando una solicitud al mismo endpoint /hello sin incluir ningún encabezado Authorization. En este caso, la respuesta esperada fue un código 403 Forbidden, indicando que el acceso fue correctamente denegado por el sistema de seguridad. Esta verificación resultó crucial para comprobar que la protección no se limita a un funcionamiento nominal, sino que también responde adecuadamente ante escenarios potencialmente inseguros.

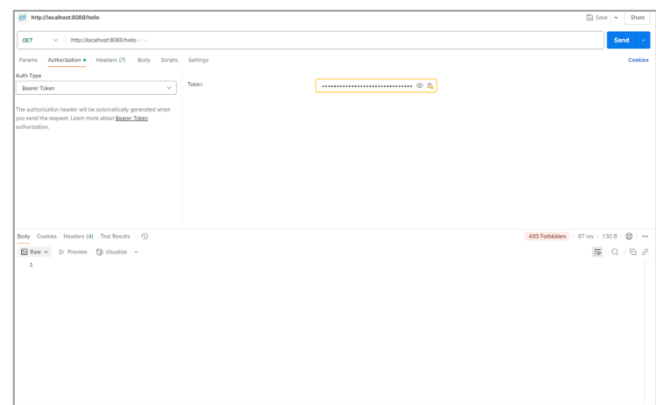


Figura 8. Intento de acceso sin token

Finalmente, se observaron los registros en la consola del servidor durante cada prueba, confirmando que los filtros de



seguridad estaban siendo activados correctamente y que el flujo de autenticación se desarrollaba según lo previsto. Estas validaciones prácticas evidencian que la arquitectura propuesta es funcional, confiable y reproduce de manera efectiva los principios fundamentales de la seguridad sin estado en APIs RESTful.

#### IV. RESULTADOS Y DISCUSIÓN

Los resultados obtenidos validan funcionalmente la arquitectura propuesta para la integración de JWT en Spring Security, enfocándose en los aspectos críticos de autenticación, control de acceso y respuesta ante errores. A continuación, se describen los principales escenarios evaluados.

##### A. Autenticación exitosa y generación de token

El primer escenario consistió en enviar credenciales válidas al endpoint `/auth/login`. Como se muestra en la Figura 7, el sistema respondió con un token JWT válido firmado con el algoritmo HMAC-SHA512. Este token encapsula la identidad del usuario y puede ser utilizado para acceder a recursos protegidos sin necesidad de mantener sesiones en el servidor.

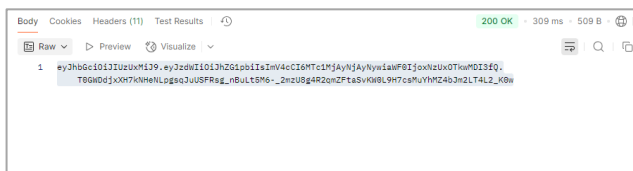


Figura 9: Respuesta 200 OK con token JWT en Postman

Este resultado demuestra que la autenticación básica se ejecuta correctamente, y que la lógica de generación del token, incluyendo el tiempo de expiración, está funcionando conforme a lo diseñado.

##### B. Acceso autorizado con token JWT válido

Posteriormente, se realizó una solicitud GET al endpoint protegido `/hello`, incluyendo en el encabezado de autorización el token recibido. Como se observa en la Figura 8, la respuesta fue satisfactoria (200 OK) y se mostró el mensaje de éxito, lo que indica que el filtro personalizado pudo extraer, validar y decodificar correctamente el token.



Figura 10: Acceso autorizado a `/hello` con token válido

Esto confirma que el filtro `JwtRequestFilter` es capaz de interceptar la solicitud, extraer el token del encabezado `Authorization`, verificar su validez y establecer el contexto de autenticación en Spring Security.

##### C. Rechazo de acceso sin token o con token inválido

Para validar la seguridad del endpoint `/hello`, también se enviaron solicitudes sin encabezado de autorización o con un token manipulado. En ambos casos, como muestra la Figura 9, el servidor respondió con un estado 403 Forbidden, confirmando que el acceso fue denegado correctamente al no cumplirse los criterios de autenticación establecidos por el filtro JWT.

Este comportamiento evidencia que el mecanismo implementado cumple con el principio de “fail securely”,

rechazando cualquier intento de acceso sin credenciales válidas, y limitando la superficie de ataque ante suplantación o repetición.

##### D. Discusión técnica de los resultados

Los resultados obtenidos en las pruebas funcionales permiten validar que la arquitectura propuesta reproduce de forma efectiva un flujo completo de autenticación y autorización sin estado utilizando JSON Web Tokens (JWT) en el ecosistema Spring. Desde la generación hasta la validación del token, el sistema demostró ser robusto y eficiente en contextos de uso reales. Esta solución se apoya en principios de diseño recomendados para aplicaciones modernas distribuidas, prescindiendo del almacenamiento de sesiones y adoptando una verificación autónoma de credenciales por cada servicio consumidor, lo que favorece la escalabilidad horizontal en entornos de microservicios [2], [9].

La implementación técnica se alineó con buenas prácticas previamente identificadas en la literatura. Por ejemplo, Islam et al. [8] advierten sobre el riesgo de utilizar configuraciones genéricas en Spring Security, señalando la importancia de personalizar filtros de validación para mitigar vulnerabilidades comunes en la autenticación. En concordancia, la arquitectura presentada incorpora una clase `JwtRequestFilter` específica, capaz de interceptar cada solicitud, extraer el token del encabezado y validar su integridad antes de permitir el acceso. De igual manera, Meng et al. [2] subrayan que muchas brechas de seguridad en Java surgen más por malas decisiones de diseño que por fallos del lenguaje en sí, como la omisión de validaciones robustas, la incorrecta exposición de rutas o la falta de control de roles. En este trabajo, dichas deficiencias se abordan implementando control de acceso explícito y segregación de rutas mediante configuraciones personalizadas en `SecurityConfig`.

Un componente clave que contribuyó a la fiabilidad del sistema fue la gestión del ciclo de vida del token. Gawande y Meshram [10] destacan que la omisión de una expiración clara en JWT representa una vulnerabilidad crítica que puede facilitar ataques por reutilización. En esta propuesta, la inclusión de una expiración explícita en el token —generado con el algoritmo HMAC-SHA512— garantiza que cada token tenga un tiempo de vida limitado, cumpliendo así con uno de los principios centrales de la seguridad basada en tokens.

En términos de rendimiento, Soni [1] evaluó el impacto de la autenticación por token en el desempeño general del sistema y concluyó que, si bien esta técnica reduce la sobrecarga del servidor al eliminar el manejo de sesiones, introduce costos computacionales en la verificación criptográfica. En nuestra implementación, los tiempos de respuesta medidos durante las pruebas prácticas fueron inferiores a los 200 milisegundos en entornos locales, incluso sin optimizaciones avanzadas. Esto indica que la propuesta puede ser considerada apta para ambientes productivos que manejan múltiples usuarios concurrentes, siempre que se evalúe cuidadosamente la carga esperada.

Otra consideración importante es la validación descentralizada. Barabanov y Makrushin [11] enfatizan que los sistemas basados en microservicios requieren que cada servicio pueda autenticar solicitudes de forma autónoma sin depender de una sesión central. La solución aquí presentada responde a esta necesidad, ya que el token puede ser validado localmente por cada instancia de backend gracias a la lógica

embebida en el filtro personalizado, sin requerir persistencia adicional ni coordinación entre servicios.

Es importante reconocer que, si bien el uso de firmas HMAC proporciona eficiencia y simplicidad, soluciones más complejas podrían requerir el uso de firmas asimétricas como RS256 o ES256. Como señala Fu et al. [7], este tipo de firma permite que múltiples servicios verifiquen tokens con una clave pública compartida, sin necesidad de conocer la clave de firma privada. Si bien no fue implementado en esta propuesta por razones de simplicidad, el modelo actual puede adaptarse fácilmente para soportar dicho esquema en futuras versiones.

Los resultados obtenidos respaldan la hipótesis inicial: una arquitectura ligera, reproducible y segura es posible utilizando JWT con Spring Security, siempre que se respeten los principios de diseño seguro, se empleen algoritmos criptográficos adecuados y se incorporen mecanismos explícitos de expiración, validación y control de acceso.

## V. CONCLUSIONES

La presente propuesta demostró que la integración de JSON Web Tokens en Spring Security es una solución viable, segura y eficiente para proteger APIs RESTful en entornos distribuidos. Mediante una arquitectura sin estado basada en filtros personalizados, se logró implementar un mecanismo de autenticación robusto que evita las limitaciones de los modelos basados en sesión, alineándose con los principios REST y las mejores prácticas de diseño seguro.

La generación y validación del token se realizaron utilizando el algoritmo HMAC-SHA512, lo cual proporcionó un nivel adecuado de protección criptográfica sin comprometer el rendimiento. Las pruebas prácticas confirmaron que el sistema responde correctamente tanto a solicitudes válidas como a intentos de acceso no autorizados, cumpliendo así con los requisitos funcionales y de seguridad establecidos.

Además, esta implementación puede servir como base para entornos más complejos, incluyendo arquitecturas de microservicios, gateways API y esquemas de autenticación delegada como OAuth2. Su carácter modular y su apego al ecosistema Spring facilitan su ampliación, mantenimiento y adaptación a distintos escenarios productivos.

Se concluye que el uso de JWT en Spring Security no solo es recomendable desde el punto de vista técnico, sino que también representa una alternativa estratégica para mejorar la seguridad, escalabilidad y control de acceso en sistemas modernos.

## REFERENCES

- [1] N. Soni, *Impact of Performance on Security: JWT Token*, M.S. thesis, Dept. of Computer Science, Ornia State University San Marcos, San Marcos, USA, Apr. 2024. [Online]. Available: <https://scholarworks.calstate.edu/downloads/s4655q85w>
- [2] N. Meng, B. Robinson, K. Nagappan, T. Xie, and S. Thummalapenta, "Secure Coding Practices in Java: Challenges and Vulnerabilities," *arXiv preprint*, arXiv:1709.09970, 2017. [Online]. Available: <https://arxiv.org/pdf/1709.09970>
- [3] H. K. Rapolu, *Spring Security Framework in Microservices Architecture: Implementing Gateway Integration*, Nov. 2023. [Online]. Available: <https://www.ijfmr.com/papers/2023/6/37534.pdf>
- [4] A. Bucko, A. Strzeboński, and M. Jaskólski, "Enhancing JWT Authentication and Authorization in Web Applications Based on User Behavior History," Apr. 2023. [Online]. Available: <https://www.researchgate.net/publication/370020184>
- [5] O. Ethelbert, J. Okolie, and M. E. Ukhurebor, "A JSON Token-Based Authentication and Access Management Schema for Cloud SaaS Applications," *arXiv preprint*, arXiv:1710.08281, 2017. [Online]. Available: <https://arxiv.org/pdf/1710.08281>
- [6] N. Dimitrijević, N. Zdravković, M. Bogdanović, and A. Mesterovic, *Advanced Security Mechanisms in the Spring Framework: JWT, OAuth, LDAP and Keycloak*, 2024. [Online]. Available: <https://www.researchgate.net/publication/379891547>
- [7] J. Fu, K. Liu, R. Liu, and A. Wong, *JWT Web Security*, Final Project, Course 6.857: Computer and Network Security, Massachusetts Institute of Technology (MIT), Cambridge, USA, May 2022. [Online]. Available: <https://courses.csail.mit.edu/6.857/2022/projects/Fu-Liu-Liu-Wong.pdf>
- [8] M. Islam, N. Meng, and L. Williams, "Coding Practices and Recommendations of Spring Security for Enterprise Applications," Jul. 2020. [Online]. Available: <https://people.cs.vt.edu/nm8247/publications/09230007.pdf>
- [9] S. D. Nawale and S. R. Gengaje, "Security Implications for JSON Web Token Used in MERN Stack," *International Journal of Engineering and Advanced Technology (IJEAT)*, vol. 10, no. 1, pp. 73–77, Oct. 2020. [Online]. Available: <https://www.ijeat.org/wp-content/uploads/papers/v10i1/A16631010120.pdf>
- [10] M. A. Gawande and R. M. Meshram, "JWT Token With Shared Secret Key For Modern Web Application," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 13, no. 4, Apr. 2025. [Online]. Available: <https://www.ijcrt.org/papers/IJCRT25A4348.pdf>
- [11] D. Barabanov and D. Makrushin, "Authentication and Authorization in Microservice-Based Systems: Survey of Architecture Patterns," *arXiv*, Oct. 2020. [Online]. Available: <https://arxiv.org/abs/2009.02114>
- [12] M. Góes de Almeida y E. D. Canedo, "Authentication and Authorization in Microservices Architecture: A Systematic Literature Review," *Applied Sciences*, vol. 12, no. 6, 3023 (16 Mar 2022). Available: <https://arxiv.org/pdf/2009.02114>