PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

SEDE IBARRA

ESCUELA DE HÁBITAT INGENIO Y CREATIVIDAD

TRABAJO DE INTEGRACIÓN CURRICULAR

PREVIO A LA OBTENCIÓN DEL TÍTULO DE

INGENIERA EN TECNOLOGÍAS DE LA INFORMACIÓN

ANÁLISIS DE VULNERABILIDADES DE SOFTWARE EN EL MÓDULO ADMINISTRATIVO DEL SISTEMA DE GESTIÓN DOCUMENTAL DE LA PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR SEDE IBARRA

NATHALY GUADALUPE ANGAMARCA ANGAMARCA

TUTOR: MGS. DIEGO FERNANDO BAROJA LLANOS

IBARRA – ECUADOR

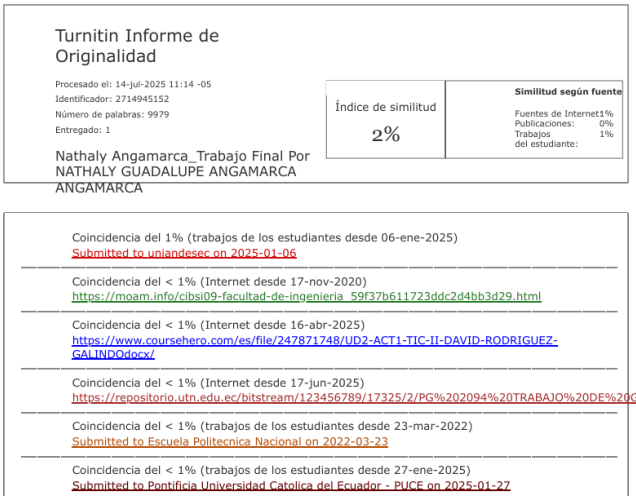
JULIO, 2025

Ibarra, Julio de 2025

CERTIFICACIÓN ASESOR

En mi calidad de Asesor del Trabajo de Titulación titulado **ANÁLISIS DE VULNERABILIDADES DE SOFTWARE EN EL MÓDULO ADMINISTRATIVO DEL SISTEMA DE GESTIÓN DOCUMENTAL DE LA PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR SEDE IBARRA**, presentado por la estudiante Nathaly Guadalupe Angamarca Angamarca con cédula de ciudadanía N° 1005193303, para obtener el Título de Ingeniera en Tecnologías de la Información. Certifico que el trabajo cumple con todos los parámetros establecidos, mediante el cual el estudiante demuestra el desarrollo de competencias en el campo de conocimiento de su profesión con un nivel de argumentación coherente, para ser sometido a la evaluación por parte de los lectores.

Adicionalmente, se adjunta el certificado de porcentaje de originalidad de TURNITIN.



Turnitin Informe de Originalidad

Procesado el: 14-jul-2025 11:14 -05
Identificador: 2714945152
Número de palabras: 9979
Entregado: 1

Nathaly Angamarca_Trabajo Final Por NATHALY GUADALUPE ANGAMARCA ANGAMARCA

Índice de similitud	Similitud según fuente
2%	Fuentes de Internet 1% Publicaciones: 0% Trabajos del estudiante: 1%

Coincidencia del 1% (trabajos de los estudiantes desde 06-ene-2025)
Submitted to uniandesec on 2025-01-06

Coincidencia del < 1% (Internet desde 17-nov-2020)
https://moam.info/cibsi09-facultad-de-ingenieria_59f37b611723ddc2d4bb3d29.html

Coincidencia del < 1% (Internet desde 16-abr-2025)
https://www.coursehero.com/es/file/247871748/UD2-ACT1-TIC-II-DAVID-RODRIGUEZ-GALINDOdocx/

Coincidencia del < 1% (Internet desde 17-jun-2025)
https://repositorio.utn.edu.ec/bitstream/123456789/17325/2/PG%202094%20TRABAJO%20DE%20GRADO

Coincidencia del < 1% (trabajos de los estudiantes desde 23-mar-2022)
Submitted to Escuela Politecnica Nacional on 2022-03-23

Coincidencia del < 1% (trabajos de los estudiantes desde 27-ene-2025)
Submitted to Pontificia Universidad Catolica del Ecuador - PUCE on 2025-01-27

Firmado digitalmente por1002402061 DIEGO FERNANDO BAROJA LLANOS
Fecha:2025-08-13 21:38-05:00

(f):

Mgs. Diego Fernando Baroja Llanos
ASESOR DE TRABAJO
C.C.: 1002402061

PÁGINA DE APROBACIÓN DEL TRIBUNAL

El tribunal examinador, aprueba el presente trabajo en nombre de la Pontificia Universidad Católica del Ecuador Ibarra:

Firmado digitalmente por1002402061 DIEGO FERNANDO BAROJA LLANOS
Fecha:2025-08-13 21:38-05:00

(f): ……

Mgs. Diego Fernando Baroja Llanos

ASESOR DE TRABAJO

C.C.: 1002402061

GALO HERNAN PUETATE HUERA
Firmado digitalmente por GALO HERNAN PUETATE HUERA
Fecha: 2025.08.14 16:11:31 -05'00'

(f):……

Msc. Galo Hernán Puetate Huera

C.C.: 0401375787

JOSÉ LUIS IBARRA ESTÉVEZ
Firmado digitalmente por JOSÉ LUIS IBARRA ESTÉVEZ
Fecha: 2025.08.14 17:51:24 -05'00'

(f):……

Msc. José Luis Ibarra Estévez

C.C.: 1002640728

ACTA DE CESIÓN DE DERECHOS

Yo, Nathaly Guadalupe Angamarca Angamarca, declaro conocer y aceptar la disposición del Art. 165 del Código Orgánico de Economía Social de los Conocimientos, Creatividad e Innovación, que manifiesta textualmente: "Se reconoce facultad de los autores y demás titulares de derechos de disponer de sus derechos o autorizar las utilizaciones de sus obras o prestaciones a título gratuito y oneroso, según las condiciones que determinen. Esta facultad podrá ejercerse mediante licencias libres, abiertas y otros modelos alternativos de licenciamiento o la renuncia".

Ibarra, julio del 2025

(f): Nathaly Guadalupe Angamarca Angamarca

Firmado digitalmente por Nathaly Guadalupe Angamarca Angamarca
Fecha: 2025.08.13 21:31:54 -05'00'

Nathaly Guadalupe Angamarca Angamarca

C.C.: 1005193303

AUTORIA

Yo, Nathaly Guadalupe Angamarca Angamarca, portadora de la cedula de ciudadanía N° 1005193303, declaro que el presente trabajo de investigación es de total responsabilidad de la autora, y eximo expresamente a la Pontificia Universidad Católica del Ecuador Ibarra de posibles reclamos o acciones legales.

(f): Nathaly Guadalupe Angamarca Angamarca

Firmado digitalmente por Nathaly Guadalupe Angamarca Angamarca
Fecha: 2025.08.13 21:32:12 -05'00'

Nathaly Guadalupe Angamarca Angamarca
C.C.: 1005193303

DEDICATORIA

Este trabajo va dedicado a mis padres, Nancy y Gerardo, quienes me han brindado un apoyo incondicional para lograrlo, realizando esfuerzos que, más que una obligación, han hecho con amor, permitiéndome alcanzar un logro más en mi vida.

A mis segundos padres, Carlos y Carmita, por estar siempre al pendiente de mí y tenderme su mano cuando lo necesito; y a Karlita, quien ha compartido conmigo risas y lágrimas, siendo una compañía invaluable en este camino.

A mis amigos, por los momentos de apoyo y motivación, por la paciencia, el cariño y por hacer de este camino una experiencia especial.

AGRADECIMIENTOS

Quiero agradecer a Dios por permitirme llegar a esta etapa de mi vida y por todas las bendiciones recibidas a lo largo de este trayecto universitario.

A mis padres, por su sacrificio, su amor incondicional y por confiar en mí incluso cuando yo dudaba. Ustedes son y serán siempre mi mayor bendición.

A mi asesor, por ser un excelente guía. Gracias por su profesionalismo, por brindarme su tiempo, su apoyo constante y por la amabilidad con la que siempre respondió a mis dudas. Su acompañamiento hizo posible la culminación de este trabajo.

Al director de la Unidad de Desarrollo de Software Empresarial de la PUCE-I, por brindarme la oportunidad de realizar este proyecto en un entorno real, lo cual enriqueció significativamente mi formación.

A mis docentes, gracias por compartir no solo sus conocimientos y experiencias académicas, sino también por aquellos consejos que trascienden el aula; palabras que invitan a reflexionar.

Finalmente, a todas las personas que, en su momento, estuvieron presentes y formaron parte de este camino universitario, gracias por su granito de arena.

ÍNDICE

CERTIFICACIÓN ASESOR ..... **II**
PÁGINA DE APROBACIÓN DEL TRIBUNAL ..... **III**
ACTA DE CESIÓN DE DERECHOS ..... **IV**
AUTORIA ..... **V**
DEDICATORIA ..... **VI**
AGRADECIMIENTOS ..... **VII**
ÍNDICE ..... **VIII**
ÍNDICE DE TABLAS ..... **XI**
ÍNDICE DE FIGURAS ..... **XIII**
RESUMEN ..... **XIV**
ABSTRACT ..... **XV**
INTRODUCCIÓN **1**
CAPÍTULO I **4**
ESTADO DEL ARTE **4**
1.1 Metodologías de análisis de vulnerabilidades 4
1.2 Normativas y estándares de seguridad 6
1.3 Amenazas en sistemas de gestión documental 8
1.3.1 Impacto de las vulnerabilidades en la gestión documental 8
1.4 Seguridad en el ciclo de vida del software 10
1.5 Tendencias actuales en seguridad de software 11
1.6 Herramientas utilizadas para el análisis de vulnerabilidades 11
1.6.1 SonarQube 12
1.6.2 Semgrep 13
1.7 Casos de estudio relevantes 14
1.7.1 Vulnerabilidades de ciberseguridad en universidades de Etiopía 14
1.7.2 Amenazas cibernéticas en instituciones educativas 14
1.7.3 Revisión de vulnerabilidades y soluciones en sistemas digitales 15
1.7.4 Brechas de seguridad en universidades de Estados Unidos 15
1.7.5 Análisis forense de código y procesos de la aplicación Ubidesk de la empresa

Uboratech 15
CAPÍTULO II 17
MATERIALES Y MÉTODOS 17
2..... 17
2.1 Entorno técnico del módulo administrativo 17
2.2 Metodología de análisis estático con SonarQube 20
2.2.1 Proceso de configuración de SonarQube 20
2.2.2 Proceso de definición de reglas y perfil de calidad 22
2.2.3 Proceso de análisis del código 24
2.2.4 Proceso de revisión de resultados 25
2.2.5 Proceso de documentación y generación de reportes 25
2.3 Metodología de análisis con Semgrep 26
2.3.1 Preparación del entorno 26
2.3.2 Configuración de reglas personalizadas 27
2.3.3 Ejecución del escaneo 29
2.3.4 Documentación y visualización de resultados 30
CAPÍTULO III 33
RESULTADOS Y DISCUSIÓN 33
2.4 Resultados del análisis estático con SonarQube 33
2.4.1 Métricas globales del proyecto 33
2.4.2 Hallazgos de seguridad, mantenibilidad y calidad del código 34
2.4.3 Observaciones sobre las incidencias detectadas 38
2.5 Resultados del análisis personalizado con Semgrep 39
2.5.1 Observaciones sobre las incidencias detectadas 41
2.5.2 Caso representativo de vulnerabilidad crítica 42
2.6 Análisis comparativo y propuesta técnica 42
2.6.1 Matriz de riesgos del módulo administrativo 43
2.6.2 Propuesta técnica para la mitigación de riesgos 46
2.7 Resultados posteriores a la implementación de mejoras y comparación con el análisis inicial 49
2.7.1 Resultados posteriores con SonarQube 49
2.7.2 Resultados posteriores con Semgrep 51
CONCLUSIONES 54

RECOMENDACIONES **55**
REFERENCIAS BIBLIOGRÁFICAS **56**
ANEXOS **60**

ÍNDICE DE TABLAS

Tabla 1 Comparación entre Análisis Estático y Análisis Dinámico 5

Tabla 2 Resumen de Categorías, Nombres y Alcance del OWASP Top 10 (2021) 6

Tabla 3 Herramientas de análisis de seguridad y sus principales características..... 11

Tabla 4 Tecnologías utilizadas en el desarrollo del módulo administrativo 18

Tabla 5 Listado de reglas establecidas en SonarQube 23

Tabla 6 Resumen de incidencias encontradas..... 25

Tabla 7 Reglas personalizadas implementadas en Semgrep para el análisis del módulo administrativo 28

Tabla 8 Resumen de resultados Semgrep 32

Tabla 9 Métricas generales del análisis estático con SonarQube..... 34

Tabla 10 Incidencias detectadas por SonarQube clasificadas por tipo, severidad y cantidad 35

Tabla 11 Incidencias detectadas por Semgrep según regla, severidad y ocurrencias 40

Tabla 12 Detalle técnico de vulnerabilidad crítica detectada 42

Tabla 13 Comparación de incidencias detectadas por SonarQube y Semgrep 43

Tabla 14 Matriz de riesgos detectados en el módulo administrativo 44

Tabla 15 Matriz priorización de riesgos del módulo administrativo 45

Tabla 16 Prioridad técnica de vulnerabilidades detectadas 46

Tabla 17 Plan de Mitigación por tipo de vulnerabilidad..... 46

Tabla 18 Recomendación por Capa de Seguridad 48

Tabla 19 Incidencias detectadas en el análisis final SonarQube..... 49

Tabla 20 Comparativa de métricas generales antes y después de las mejoras SonarQube 50

Tabla 21 Comparativa de hallazgos Semgrep inicial y final 51

ÍNDICE DE FIGURAS

Figura 1 Arquitectura general del sistema de gestión documental de la PUCE-I..... 19

Figura 2 Proceso metodológico aplicado con SonarQube para el análisis de código fuente 20

Figura 3 Configuración del archivo sonar-project.properties para el análisis en SonarQube 21

Figura 4 Configuración del entorno de pruebas y generación de cobertura 22

Figura 5 Acceso al servidor web de SonarQube tras iniciar sesión como administrador. 24

Figura 6 Entorno WSL utilizado para el análisis..... 26

Figura 7 Primeros comandos ejecutados en WSL 27

Figura 8 Archivo reglas-seguridad.yml 29

Figura 9 Resumen del análisis realizado con Semgrep desde la línea de comandos..... 30

Figura 10 Creación del archivo reporte-semgrep.txt 31

RESUMEN

La investigación tuvo como propósito analizar las vulnerabilidades del módulo administrativo del Sistema de Gestión Documental de la Pontificia Universidad Católica del Ecuador Sede Ibarra (PUCE-I), identificando debilidades técnicas que puedan afectar la seguridad, calidad y continuidad operativa. Para ello, se aplicó una metodología de tipo tecnológico basada en análisis estático del código fuente en su versión de desarrollo, utilizando las herramientas SAST SonarQube y Semgrep, configuradas con perfiles y reglas personalizadas adaptadas al contexto institucional.

Los resultados mostraron que, aunque el proyecto superó el Quality Gate general en SonarQube, se detectaron 45 incidencias relevantes, entre ellas duplicación de literales y alta complejidad cognitiva, que afectan la mantenibilidad. El análisis con Semgrep identificó 168 hallazgos adicionales, destacando el uso de `@ts-ignore` sin justificación, código no utilizado y configuraciones críticas como la desactivación de la validación TLS, lo que expone a ataques de hombre-en-el-medio. Con base en estos hallazgos, se elaboró una matriz integral de riesgos priorizados por severidad técnica y se diseñó un plan de mitigación con acciones específicas, responsables y plazos, además de una propuesta técnica segmentada por capas de seguridad, desde la presentación hasta la infraestructura.

Se concluye que el uso conjunto de SonarQube y Semgrep es efectivo para detectar fallos estructurales y vulnerabilidades en entornos universitarios. Se recomienda mantener revisiones periódicas de código, eliminar configuraciones inseguras y cumplir estándares como TLS, fortaleciendo así la protección de activos digitales y la confianza institucional.

Palabras clave: análisis estático, SonarQube, Semgrep, vulnerabilidades, gestión documental, seguridad de software.

ABSTRACT

The purpose of this research was to analyze the vulnerabilities of the administrative module of the Document Management System at the Pontifical Catholic University of Ecuador, Ibarra campus (PUCE-I), identifying technical weaknesses that could affect security, quality, and operational continuity. A technological methodology was applied, based on static analysis of the development version of the source code, using SAST tools SonarQube and Semgrep configured with customized profiles and rules adapted to the institutional context.

The results showed that, although the project passed the overall Quality Gate in SonarQube, 45 relevant issues were detected, including duplicated string literals and high cognitive complexity, which impact maintainability. The Semgrep analysis identified 168 additional findings, highlighting the use of `@ts-ignore` without justification, unused code, and critical configurations such as disabling TLS validation, which exposes the system to man-in-the-middle attacks.

Based on these findings, a comprehensive risk matrix was developed, prioritizing vulnerabilities by technical severity, and a mitigation plan was designed with specific actions, assigned responsibilities, and timelines, along with a technical proposal segmented by security layers, from the presentation to the infrastructure level.

It is concluded that the combined use of SonarQube and Semgrep is effective for detecting structural flaws and security vulnerabilities in university environments. It is recommended to maintain periodic code reviews, remove insecure configurations, and comply with standards such as TLS, thus strengthening the protection of digital assets and institutional trust.

Keywords: static analysis, SonarQube, Semgrep, vulnerabilities, document management, software security.

INTRODUCCIÓN

En la era digital actual, la seguridad de la información se ha consolidado como un pilar fundamental para organizaciones e individuos que gestionan datos sensibles. La creciente dependencia de infraestructuras digitales ha expuesto a las instituciones a un espectro cada vez más sofisticado de amenazas cibernéticas, incluyendo accesos no autorizados, vulnerabilidades en aplicaciones y filtración de datos. La materialización de estas amenazas puede desencadenar consecuencias devastadoras, desde la pérdida de confianza de los usuarios hasta sanciones legales y daño permanente a la reputación institucional (MetaCompliance, 2021).

En la actualidad existen herramientas especializadas de análisis estático como SonarQube y Semgrep, que permiten examinar el código fuente en busca de debilidades de seguridad. Estas debilidades suelen originarse por errores de desarrollo, configuraciones inadecuadas o malas prácticas de codificación, y pueden facilitar el acceso no autorizado, la alteración o incluso la destrucción de la información gestionada por las aplicaciones (ENISA, 2021).

Los sistemas de gestión documental constituyen el núcleo operativo para la administración de información crítica dentro del ecosistema organizacional. Estos sistemas no solo facilitan el almacenamiento y recuperación eficiente de datos, sino que también deben garantizar su protección integral. Para ello, se requiere implementar medidas que aseguren la disponibilidad, la confidencialidad y la integridad de los documentos digitales, especialmente cuando se trata de información institucional sensible.

La Pontificia Universidad Católica del Ecuador Sede Ibarra (PUCE-I), fundada en 1976, ha desarrollado una trayectoria académica distinguida ofreciendo diversos programas de formación superior y participando activamente en iniciativas de investigación (Pontificia

Universidad Católica del Ecuador, 2025). Como custodio de información académica, administrativa y personal de su comunidad universitaria, la PUCE-I tiene la responsabilidad fundamental de proteger estos activos contra amenazas potenciales.

En este contexto institucional, se consideró fundamental evaluar la seguridad del Módulo Administrativo del Sistema de Gestión Documental, identificando y mitigando las posibles vulnerabilidades existentes. En este sentido, el objetivo general de esta investigación se estableció de la siguiente manera:

El objetivo general de esta investigación es analizar las vulnerabilidades del código fuente en el Módulo Administrativo del Sistema de Gestión Documental de la PUCE-I, con el fin de garantizar la seguridad de la aplicación. Para alcanzar este propósito, se plantean como objetivos específicos: analizar la literatura académica sobre vulnerabilidades de software para el desarrollo del estado del arte; identificar las vulnerabilidades técnicas en el módulo administrativo mediante la configuración del servidor de SonarQube y Semgrep, estableciendo los parámetros necesarios para el análisis de código; evaluar los resultados obtenidos del análisis realizado con estas herramientas, determinando la exposición del sistema a amenazas externas e internas; y proponer recomendaciones basadas en dichos resultados para fortalecer la seguridad del sistema.

El presente trabajo se estructura en tres capítulos fundamentales. En el primer capítulo, se desarrolla un marco teórico exhaustivo que examina el estado del arte, incluyendo antecedentes históricos, conceptos fundamentales y estudios previos relacionados con el análisis de vulnerabilidades en sistemas de gestión documental. En el segundo capítulo, se detalla la metodología aplicada, describiendo minuciosamente los procedimientos técnicos, configuraciones específicas y herramientas implementadas en la evaluación integral del sistema. Finalmente, en el tercer capítulo, se presentan los resultados obtenidos mediante visualizaciones analíticas,

acompañados de un análisis crítico de los hallazgos y un conjunto estructurado de recomendaciones técnicas y organizativas para fortalecer la arquitectura de seguridad del sistema evaluado.

CAPÍTULO I

ESTADO DEL ARTE

Este capítulo presenta los fundamentos conceptuales que sustentan la investigación, abordando temas como las vulnerabilidades de software, los principios de la seguridad de la información y el análisis estático de código. Además, se revisan estudios previos y marcos normativos relacionados con la seguridad en entornos similares.

1.1 Metodologías de análisis de vulnerabilidades

El análisis de vulnerabilidades en aplicaciones de software se fundamenta en técnicas que permiten identificar debilidades de seguridad que podrían ser aprovechadas por un atacante. Entre los enfoques más utilizados se encuentran el análisis estático y el análisis dinámico, los cuales ofrecen distintos niveles de profundidad y cobertura.

El análisis estático de código fuente (SAST) examina la estructura del programa sin ejecutarlo, permitiendo detectar fallos desde etapas tempranas del desarrollo. Este enfoque ayuda a identificar errores como inyecciones, uso de funciones inseguras o configuraciones incorrectas. Según Jit (2025), el SAST resulta útil en metodologías ágiles al integrarse en entornos de integración continua (CI) y facilitar correcciones antes del despliegue.

Por su parte, el análisis dinámico (DAST) evalúa la aplicación en tiempo de ejecución, simulando interacciones reales para detectar vulnerabilidades como validaciones débiles, errores de configuración o comportamientos inesperados. Este tipo de análisis es especialmente útil para identificar problemas que solo se manifiestan en ambientes activos (Codacy, 2023).

A continuación, en la Tabla 1 se presenta una comparación entre las características del análisis estático y dinámico, con el fin de destacar sus diferencias funcionales y aplicaciones prácticas dentro del análisis de seguridad.

Tabla 1

Comparación entre Análisis Estático y Análisis Dinámico

Característica	Análisis estático	Análisis dinámico
Definición	Examina el código fuente sin ejecutarlo.	Examina el código durante su ejecución en un entorno controlado.
Aplicación	Se realiza antes de la ejecución del programa (en tiempo de compilación o diseño).	Se realiza durante la ejecución del programa.
Ventajas	- Detecta errores tempranos. - Cubre todo el código. - No requiere ejecución.	- Identifica errores en tiempo real. - Evalúa el comportamiento dinámico.
Limitaciones	- Puede generar falsos positivos. - No detecta errores de tiempo de ejecución.	- Requiere un entorno de ejecución. - No cubre todo el código.
Uso típico	Revisión de código, cumplimiento de estándares, seguridad estática.	Pruebas de rendimiento, seguridad dinámica, validación de funcionalidades.

Nota. Adaptado de *Análisis Estático y Dinámico*, por R. Camacho, 4 de abril de 2023, Parasoft. https://es.parasoft.com/blog/static-analysis-and-dynamic-analysis/

Estas metodologías se aplican mediante herramientas especializadas. Para el análisis estático, se utilizan soluciones como SonarQube, Semgrep o Checkmarx. En cuanto al análisis dinámico, destacan herramientas como OWASP ZAP, Burp Suite y Nessus. Estas tecnologías permiten automatizar el proceso de detección de vulnerabilidades, adaptándose al tipo de sistema,

lenguaje de programación y profundidad requerida (OWASP, 2021; Semgrep, 2025a; SonarSource, 2025).

1.2 Normativas y estándares de seguridad

El análisis de seguridad en sistemas informáticos se apoya en marcos normativos internacionales que definen directrices técnicas, administrativas y legales para proteger la información. Entre los más relevantes se encuentra la ISO/IEC 27001, que establece un sistema de gestión de seguridad de la información (SGSI), abarcando controles sobre accesos, políticas de cifrado, identificación de riesgos y continuidad operativa (Ramos, 2025).

En el ámbito de ciberseguridad web, el OWASP Top 10 es una guía ampliamente adoptada que clasifica los riesgos más frecuentes que afectan a las aplicaciones en producción. No constituye un estándar para análisis de código fuente, sino una referencia para categorizar vulnerabilidades como inyecciones, fallos de autenticación, uso de componentes obsoletos y errores de configuración (OWASP, 2021). Su principal utilidad radica en orientar los esfuerzos de mitigación y establecer criterios de evaluación en pruebas dinámicas y herramientas de seguridad. La tabla 2 sirve como referencia global para comprender y mitigar las vulnerabilidades críticas:

Tabla 2

Resumen de Categorías, Nombres y Alcance del OWASP Top 10 (2021)

Categoría	Nombre	Alcance
A01:2021	Broken Access Control	Fallas en restricciones de acceso de usuarios, permitiendo acciones no autorizadas.
A02:2021	Cryptographic Failures	Fallos relacionados con protección de datos en tránsito o almacenamiento, incluyendo cifrado débil o ausente.

A03:2021	Injection	Vulnerabilidades que permiten que datos no confiables sean enviados a un intérprete, como SQL, NoSQL, OS y LDAP injection.
A04:2021	Insecure Design	Fallas en los patrones y principios de diseño de seguridad, más allá de simples errores de implementación.
A05:2021	Security Misconfiguration	Configuraciones inseguras o por defecto en software, plataformas o frameworks, que exponen vulnerabilidades.
A06:2021	Vulnerable and Outdated Components	Uso de componentes obsoletos o vulnerables (bibliotecas, frameworks, etc.) que facilitan ataques.
A07:2021	Identification and Authentication Failures	Fallos en los mecanismos de identificación y autenticación de usuarios, incluyendo gestión de sesiones.
A08:2021	Software and Data Integrity Failures	Compromiso de integridad en el software y datos, como cargas de actualizaciones sin verificación o pipelines de CI/CD inseguros.
A09:2021	Security Logging and Monitoring Failures	Insuficiencia de registros y monitoreo que dificultan la detección y respuesta a incidentes de seguridad.
A10:2021	Server-Side Request Forgery (SSRF)	Vulnerabilidad donde el servidor puede ser manipulado para realizar solicitudes a recursos internos o externos maliciosos.

Nota. Elaboración propia basada en *OWASP Top 10: 2021*, por OWASP Foundation, 2021, OWASP. https://owasp.org/Top10/es/

Esta tabla orienta a desarrolladores, analistas y testers a focalizar sus esfuerzos de análisis y mejora en los puntos de mayor riesgo.

Por otro lado, el marco estadounidense NIST SP 800-53 define un conjunto de controles para proteger la confidencialidad, integridad y disponibilidad de los sistemas de información. Este documento contempla buenas prácticas como autenticación multifactor, monitoreo de eventos, gestión de incidentes y control de configuraciones (NIST, 2020).

1.3 Amenazas en sistemas de gestión documental

Un sistema de gestión documental (SGD) es una herramienta tecnológica que permite almacenar, organizar y controlar documentos digitales dentro de una institución, facilitando la trazabilidad, el acceso controlado y la automatización de flujos de trabajo (Ayerdi, 2024). Al centralizar información crítica, estos sistemas se convierten en un objetivo frecuente para actores maliciosos.

Entre las amenazas más comunes se encuentran el acceso no autorizado, la fuga de información, los errores en la configuración de permisos y el almacenamiento sin cifrado adecuado (Canteli, 2024). En entornos académicos, Alghamdi y Alghamdi (2024) destacan la prevalencia de ataques como phishing, ransomware y denegación de servicio (DoS), los cuales afectan la continuidad operativa de manera directa.

Ayerdi (2024) también advierte que la falta de autenticación fuerte, la ausencia de controles por rol y la escasa validación de registros pueden facilitar el uso indebido de los documentos o su alteración sin trazabilidad. A esto se suman los errores humanos, como cargas mal ejecutadas o la eliminación involuntaria de archivos, que representan riesgos latentes incluso en entornos técnicamente seguros.

1.3.1 Impacto de las vulnerabilidades en la gestión documental

Las vulnerabilidades presentes en un sistema de gestión documental pueden generar consecuencias críticas para la organización, especialmente en entornos donde se manejan documentos administrativos, legales o académicos.

Cuando no existen mecanismos de control adecuados, fallos técnicos derivados de una implementación insegura o mal gestionada pueden permitir la alteración o eliminación de documentos clave. Esto afecta la trazabilidad de la información, el cumplimiento de normas legales y la continuidad de los procesos institucionales (Canteli, 2024).

Además, según Ayerdi (2024), la exposición de archivos sensibles —como contratos, actas, datos personales o reportes financieros— puede derivar en sanciones legales, pérdida de reputación institucional o uso indebido de la información por terceros.

Estas fallas impactan directamente sobre los principios de seguridad de la información previamente descritos. En el contexto de los sistemas de gestión documental, la confidencialidad se ve comprometida cuando se permite el acceso no autorizado a expedientes académicos, administrativos o personales. La integridad se afecta si los documentos son alterados, eliminados o sobrescritos sin control, lo cual debilita la trazabilidad y validez de los registros institucionales. Finalmente, la disponibilidad se vulnera cuando el sistema sufre caídas o bloqueos, impidiendo el acceso oportuno a información crítica. Según Fortinet (2025), estos tres pilares deben mantenerse equilibrados para asegurar la continuidad, fiabilidad y protección de los activos digitales en cualquier entorno organizacional.

La interrupción del servicio también representa un riesgo significativo, ya que puede paralizar temporalmente los procesos de gestión documental y limitar el acceso a información institucional clave. Esto no solo afecta la operatividad diaria, sino que también puede generar

retrasos en trámites, pérdida de productividad y dificultad para dar cumplimiento a obligaciones administrativas o legales (Alghamdi & Alghamdi, 2024).

Por tanto, mitigar las vulnerabilidades no solo es una medida técnica, sino una estrategia crítica para garantizar la seguridad jurídica, la eficiencia operativa y la confianza institucional.

1.4 Seguridad en el ciclo de vida del software

Incorporar la seguridad durante todo el ciclo de vida del desarrollo de software permite identificar y mitigar vulnerabilidades desde etapas tempranas. Este enfoque, conocido como *Secure Software Development Lifecycle* (SSDLC), promueve la integración de controles técnicos y buenas prácticas en cada fase del desarrollo: desde el análisis de requisitos hasta el mantenimiento posterior a la implementación (Gartner, 2022).

En la etapa de planificación, se establecen políticas de acceso, objetivos de seguridad y lineamientos normativos. Durante el diseño, se definen arquitecturas resistentes a ataques, considerando aspectos como autenticación, cifrado y validación de entradas. En la fase de codificación, se aplican herramientas de análisis estático para identificar errores en el código fuente, mientras que en la etapa de pruebas se utilizan escáneres automáticos y casos de prueba orientados a detectar fallos de seguridad funcional. Finalmente, en el despliegue y mantenimiento, se refuerzan las configuraciones del entorno, se gestionan parches de seguridad y se monitorean eventos relevantes (IBM, 2021).

Este modelo puede complementarse con enfoques recientes que buscan automatizar e integrar la seguridad de forma continua a lo largo del desarrollo.

1.5 Tendencias actuales en seguridad de software

Una de las prácticas más adoptadas actualmente es el modelo *shift-left*, que consiste en ejecutar actividades de seguridad —como el análisis estático de código— desde las primeras etapas del ciclo de vida del software. Según Jit (2025), esta estrategia permite detectar vulnerabilidades antes del despliegue, reduciendo los costos de corrección y mejorando la calidad del producto.

También se ha incrementado el uso de inteligencia artificial (IA) aplicada a la seguridad del software, orientada tanto a la detección temprana de patrones anómalos en el código como a la automatización de controles de validación y monitoreo. Conforme señala Legit Security (2025), las soluciones de IA en ciberseguridad ayudan a inspeccionar grandes volúmenes de datos, adaptarse a nuevas amenazas y mejorar la eficiencia en la detección y respuesta ante incidentes.

1.6 Herramientas utilizadas para el análisis de vulnerabilidades

En el contexto del análisis de seguridad, diversas herramientas permiten automatizar la detección de vulnerabilidades técnicas, errores de configuración o malas prácticas de desarrollo. Estas plataformas aplican enfoques estáticos o dinámicos según su propósito, y se integran en diferentes fases del ciclo de vida del software. Su correcta aplicación contribuye a reducir los riesgos de explotación y a reforzar la calidad del código evaluado (OWASP, 2021; Tenable, s.f.; Acunetix, 2025). A continuación, en la Tabla 3 se presenta una comparación de las herramientas más utilizadas.

Tabla 3

Herramientas de análisis de seguridad y sus principales características

Herramienta	Descripción	Características

OWASP ZAP	Herramienta gratuita de análisis dinámico para aplicaciones web.	Intercepta, modifica y escanea tráfico HTTP/HTTPS; incluye escaneo pasivo y activo, *spidering*, *fuzzing* y API REST para automatización.
Burp Suite	Suite profesional de pruebas de penetración web.	Interceptores de tráfico, escaneo activo, análisis del árbol del sitio, manipulación de peticiones en tiempo real y módulos de fuerza bruta.
Acunetix	Escáner comercial para vulnerabilidades web y APIs.	Escaneo de REST y GraphQL, detección de más de 7,000 vulnerabilidades, reportes OWASP/ISO/PCI-DSS y alertas en tiempo real.
Nessus	Herramienta de evaluación de infraestructura y redes.	Escaneo de servidores y bases de datos, detección de configuraciones débiles y generación de informes de riesgo priorizado.

Nota. Elaboración propia basada en *OWASP Top 10: 2021*, por OWASP Foundation, 2021, OWASP (https://owasp.org/Top10/es/); *Burp Suite*, por PortSwigger, 2025, PortSwigger (https://portswigger.net/burp); *Acunetix Web Vulnerability Scanner*, por Acunetix, 2025, Acunetix (https://www.acunetix.com/); y *Nessus*, por Tenable, s.f., Tenable (https://www.tenable.com/products/nessus).

1.6.1 SonarQube

SonarQube es una plataforma de análisis estático que permite examinar el código fuente con el fin de identificar errores de calidad, vulnerabilidades y patrones de desarrollo inadecuados. Su funcionalidad se centra en aplicar reglas predefinidas o personalizadas para evaluar aspectos como seguridad, legibilidad y mantenibilidad del código, sin necesidad de ejecutarlo.

Entre sus capacidades técnicas destacan la compatibilidad con múltiples lenguajes de programación, el análisis incremental sobre los cambios recientes, y la integración con sistemas de control de versiones y entornos de integración continua. Además, permite estructurar

compuertas de calidad (*quality gates*) y perfiles de reglas alineados con estándares como OWASP Top 10 y CWE.

La versión comunitaria de la herramienta proporciona funciones esenciales para análisis de calidad, aunque no incluye opciones avanzadas como revisión de archivos binarios, análisis de secretos expuestos o escaneo en entornos corporativos más amplios (Security Boulevard, 2025; MegaInterview, 2024; SonarSource, 2025).

1.6.2 Semgrep

Semgrep es una herramienta de análisis estático orientada a la detección de vulnerabilidades y patrones inseguros mediante reglas personalizables escritas en YAML. A diferencia de otras plataformas, su motor de análisis semántico permite identificar coincidencias específicas en el código fuente, considerando tanto la sintaxis como el contexto en el que se aplican las instrucciones.

Esta herramienta soporta múltiples lenguajes, incluyendo JavaScript, TypeScript, Python y Go. Sus funcionalidades permiten aplicar reglas alineadas con marcos como OWASP, PCI DSS o HIPAA, así como definir políticas internas adaptadas al entorno del proyecto. Es compatible con flujos de integración y entrega continua (CI/CD), y puede integrarse en repositorios para escaneos automáticos.

Semgrep está disponible en una versión gratuita que incluye la ejecución de reglas personalizadas y acceso a bibliotecas públicas, mientras que su versión empresarial incorpora paneles de auditoría, monitoreo centralizado y controles de cumplimiento normativo (Semgrep, 2025a; Semgrep, 2025b).

1.7 Casos de estudio relevantes

A continuación, se presentan estudios relevantes que han abordado el análisis de vulnerabilidades en entornos similares, cuyos enfoques metodológicos y resultados contribuyen al marco contextual de esta investigación.

1.7.1 Vulnerabilidades de ciberseguridad en universidades de Etiopía

En un estudio realizado por Eshetu, Mohammed y Salau (2024), se identificaron múltiples debilidades en los sitios web de universidades etíopes, tales como el uso de software obsoleto, falta de cifrado y contraseñas débiles. Se aplicaron herramientas de escaneo automático que revelaron importantes fallos de seguridad, lo cual pone en riesgo la confidencialidad y disponibilidad de la información académica. Como solución, se propusieron medidas correctivas alineadas con estándares internacionales de ciberseguridad.

1.7.2 Amenazas cibernéticas en instituciones educativas

Alghamdi y Alghamdi (2024) examinaron amenazas comunes en universidades, incluyendo ataques de denegación de servicio, phishing y ransomware. Concluyeron que las instituciones requieren estrategias proactivas, formación continua y políticas robustas de seguridad para mitigar estas amenazas y proteger sus entornos digitales.

1.7.3 Revisión de vulnerabilidades y soluciones en sistemas digitales

Kumar, Sahu y Maheshwari (2023) realizaron una revisión sistemática sobre vulnerabilidades de ciberseguridad, tales como XSS, DoS y fallos de configuración. El estudio enfatiza la importancia de aplicar contramedidas como autenticación multifactor, cifrado y herramientas de análisis preventivo para reducir la exposición al riesgo.

1.7.4 Brechas de seguridad en universidades de Estados Unidos

Wilson (2022) analizó cómo las universidades estadounidenses enfrentan desafíos en ciberseguridad debido a limitaciones presupuestarias y escasa concientización institucional. El autor recomienda establecer planes estratégicos, invertir en tecnologías de protección y capacitar al personal como pasos esenciales para una gestión segura de la información.

1.7.5 Análisis forense de código y procesos de la aplicación Ubidesk de la empresa Uboratech

En el trabajo de titulación realizado por Vilañez Ordoñez (2024), se llevó a cabo un análisis forense del código y los procesos de la aplicación Ubidesk, desarrollados por la empresa Uboratech, con el objetivo de identificar y mitigar vulnerabilidades de seguridad que podrían comprometer la integridad, confidencialidad y disponibilidad de los datos manejados por la plataforma.

La investigación incluyó fases de recolección de información, análisis estático y dinámico del código fuente, revisión de registros y pruebas de penetración, utilizando herramientas como SonarQube para detectar vulnerabilidades tales como inyecciones de código, errores de autenticación y configuraciones inseguras. El análisis dinámico permitió observar comportamientos anómalos y tráfico sospechoso en un entorno controlado, mientras que la evaluación de políticas y procedimientos de seguridad reveló áreas críticas para mejorar.

Como resultado, se identifican vulnerabilidades críticas y fallos en la implementación que podrían facilitar los accesos no autorizados o la pérdida de información sensible. Se propusieron recomendaciones para fortalecer la seguridad, incluyendo validaciones de entrada más estrictas, uso de cifrado robusto y monitoreo continuo de registros. Este análisis contribuye a mejorar la seguridad y eficiencia de Ubidesk, garantizando la protección de los datos de los usuarios y la continuidad del negocio.

CAPÍTULO II

MATERIALES Y MÉTODOS

En este capítulo se describen los procedimientos, técnicas y herramientas utilizadas para analizar las vulnerabilidades en el Módulo Administrativo del Sistema de Gestión Documental de la PUCE-I.

2.1 Entorno técnico del módulo administrativo

El análisis de seguridad se enfoca en el módulo administrativo del Sistema de Gestión Documental de la PUCE-I, correspondiente al backend de la aplicación. Para ello, se utilizó el código fuente en su versión de desarrollo, proporcionado por la Unidad de Desarrollo de Software Empresarial, el cual se encuentra estructurado en carpetas funcionales que contienen controladores, rutas, modelos de datos, configuraciones y servicios auxiliares.

Los elementos evaluados se limitan a:

Código fuente backend: desarrollado en TypeScript sobre el entorno de ejecución Node.js. El sistema emplea una arquitectura modular con controladores (controllers), rutas (routes), modelos (models) y middlewares. El framework identificado en el entorno es Express.

Base de datos: el sistema integra MongoDB junto con Mongoose como ODM, lo cual permite definir esquemas y aplicar validaciones en los modelos. El análisis se centró en la estructura declarada de estos modelos, sin acceso a datos en producción.

Configuración general: el backend expone servicios RESTful, documentados mediante Swagger, y se gestiona bajo control de versiones con Git. El despliegue se apoya en prácticas automatizadas mediante Jenkins, y el sistema aplica autenticación mediante JWT.

La Tabla 4 presenta un resumen de las tecnologías utilizadas tanto en el backend como en el frontend, aunque este último no fue objeto de análisis en esta investigación.

Tabla 4

Tecnologías utilizadas en el desarrollo del módulo administrativo

Categoría	Backend	Frontend
Framework	NestJS / Node.js	NuxtJS / Vue.js
Estilos	—	TailwindCSS
Documentación	Swagger	—
Seguridad	JWT	—
Despliegue continuo	—	Jenkins
Base de datos	PostgreSQL	—
Control de versiones	Git	Git
Ambientes	Desarrollo / QA / Producción	Desarrollo / QA / Producción

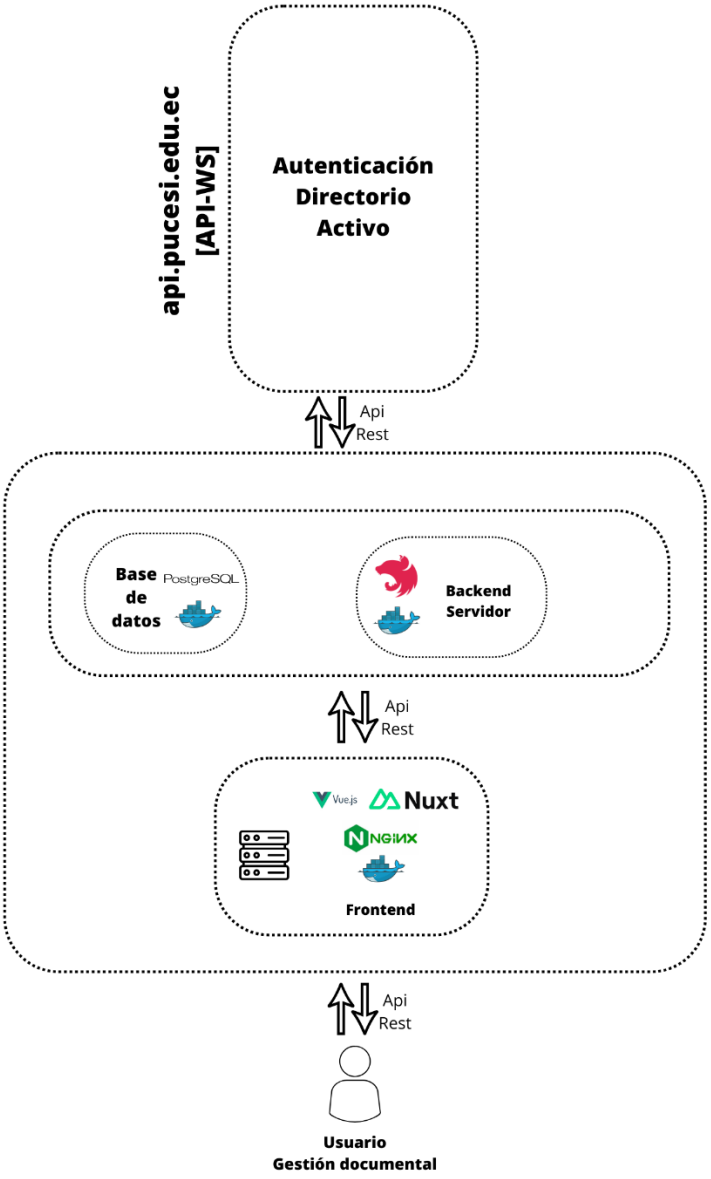
En cuanto a la arquitectura, el sistema responde a un modelo orientado a servicios, donde los distintos módulos se comunican a través de interfaces bien definidas. El módulo administrativo actúa como proveedor de servicios backend, exponiendo su funcionalidad a través de una API REST.

De acuerdo con la información oficial proporcionada, el sistema opera bajo tres ambientes diferenciados: desarrollo, pruebas (QA) y producción. Esta segmentación garantiza un manejo controlado de versiones y facilita la ejecución de pruebas técnicas sin afectar la operación en línea.

A continuación, en la Figura 1 se presenta un esquema representativo de la arquitectura general del sistema de gestión documental, donde se evidencian los principales componentes involucrados en el flujo de información y servicios.

Figura 1

Arquitectura general del sistema de gestión documental de la PUCE-I



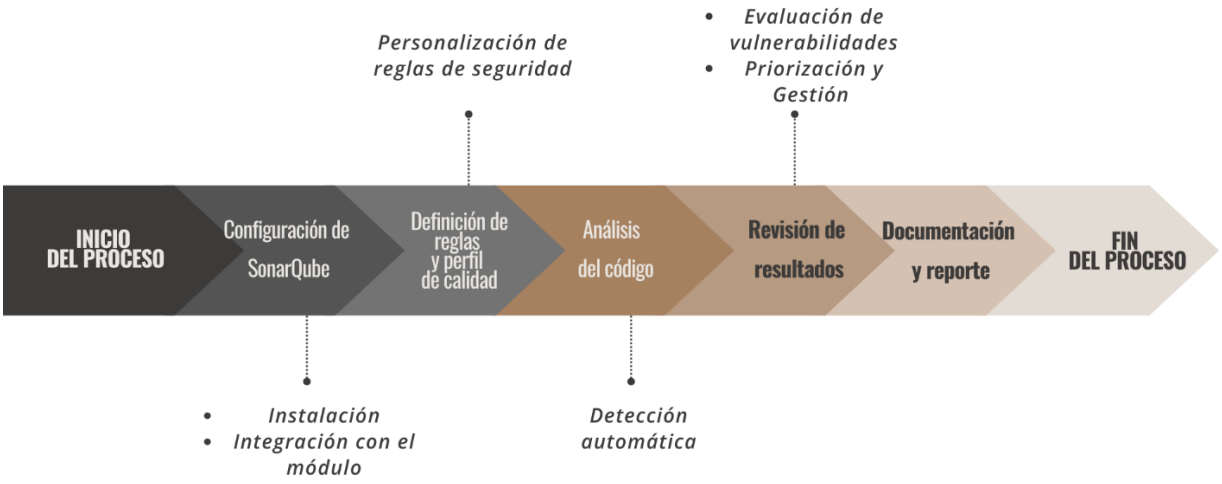
Nota. Imagen elaborada por la Unidad de Desarrollo de Software Empresarial – PUCE-I.

2.2 Metodología de análisis estático con SonarQube

El análisis del código fuente se llevó a cabo mediante una inspección estática utilizando la herramienta SonarQube. Este procedimiento se aplicó directamente sobre el Módulo Administrativo del Sistema de Gestión Documental de la PUCE-I, en un entorno controlado de pruebas. La Figura 2 ilustra el entorno de análisis y algunos de los resultados obtenidos, como punto de partida para detallar las fases del proceso: configuración inicial, definición de reglas de seguridad, ejecución del escaneo, evaluación de los resultados y generación de reportes técnicos.

Figura 2

Proceso metodológico aplicado con SonarQube para el análisis de código fuente



2.2.1 Proceso de configuración de SonarQube

Instalación y entorno:

Se instaló SonarQube versión 9.9.4 LTS en un entorno local con Windows 11. El servidor fue iniciado mediante el script `StartSonar.bat`, ubicado en `C:\sonarqube\bin\windows-x86-`

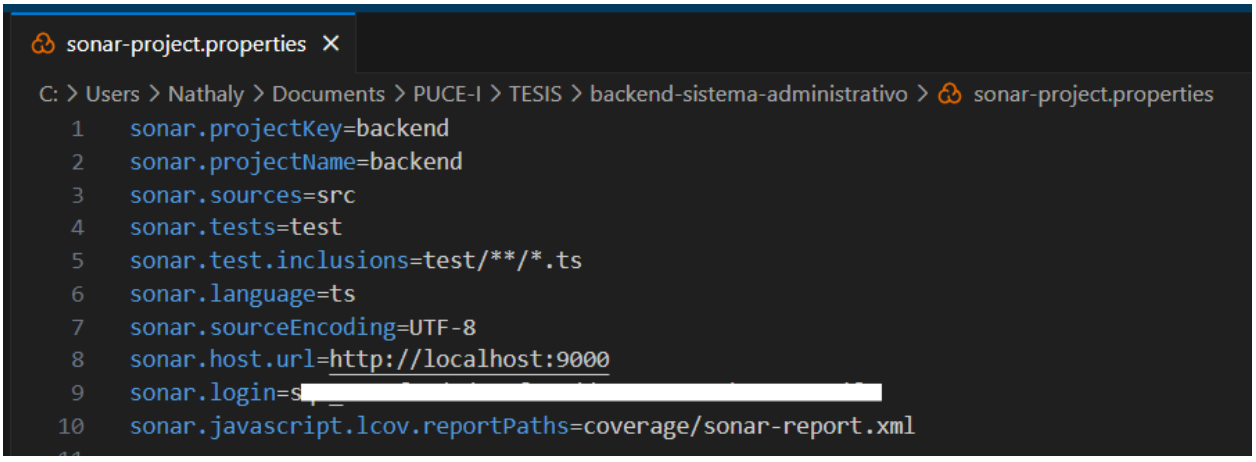
64. Para el análisis se utilizó SonarScanner CLI versión 7.1.0.4889, autenticado mediante un token generado desde la consola web del servidor.

Integración con el módulo administrativo:

El análisis se aplicó al proyecto backend-sistema-administrativo, ubicado en C:\Users\Nathaly\Documents\PUCE-I\TESIS. Al no encontrarse bajo un sistema de control de versiones (SCM), la ejecución del escaneo se realizó manualmente desde la raíz del repositorio. La configuración del análisis se definió en el archivo sonar-project.properties, cuya estructura se presenta en la Figura 3, donde se especificaron parámetros como la codificación (UTF-8), rutas del código fuente (src), pruebas (test), exclusiones e inclusión del token de autenticación.

Figura 3

Configuración del archivo sonar-project.properties para el análisis en SonarQube



Cobertura de código:

Se habilitó la cobertura mediante `jest` junto con el generador `jest-sonar-reporter`. Aunque el proyecto contiene pruebas E2E localizadas en la carpeta `test/`, la cobertura obtenida fue limitada debido al enfoque funcional de dichas pruebas. No obstante, el archivo `sonar-`

`report.xml` fue generado correctamente en el directorio `coverage/` y reconocido automáticamente por SonarQube, como se observa en la Figura 4.

Figura 4

Configuración del entorno de pruebas y generación de cobertura

2.2.2 Proceso de definición de reglas y perfil de calidad

Se estableció un perfil de calidad personalizado denominado “Administrador”, orientado específicamente al análisis del código TypeScript del módulo evaluado. Este perfil fue configurado a partir del estándar “Sonar way”, el cual se adaptó tras un análisis inicial que permitió descartar reglas genéricas poco relevantes para el contexto del proyecto.

Como resultado, se activaron 13 reglas clave, priorizando aspectos relacionados con la seguridad, la mantenibilidad y la legibilidad del código. Estas reglas enfatizan el manejo adecuado

de errores, la eliminación de estructuras redundantes, la validación de entradas y la prevención de patrones de codificación que puedan representar riesgos.

La Tabla 5 presenta el conjunto de reglas configuradas dentro del perfil de calidad "Administrador".

Tabla 5

Listado de reglas establecidas en SonarQube

ID de Regla	Descripción
typescript:S1523	Evitar el uso de `console.log()` en producción
typescript:S4326	Reducir la complejidad cognitiva excesiva
typescript:S1192	Evitar cadenas duplicadas
typescript:S2688	Eliminar asignaciones no utilizadas
typescript:S6329	Evitar estructuras condicionales anidadas innecesarias
typescript:S1871	Condiciones siempre verdaderas o falsas
typescript:S2583	Verificación de condiciones redundantes
typescript:S3504	Comparaciones innecesarias con valores booleanos
typescript:S3656	No capturar excepciones sin realizar acciones
typescript:S4790	Validar correctamente el uso de expresiones regulares
typescript:S5304	Usar `strict` y `noImplicitAny` en la configuración del compilador
typescript:S5247	Evitar ciclos `for-in` en objetos sin validación
typescript:S6073	Evitar promesas no manejadas

2.2.3 Proceso de análisis del código

El análisis del código fuente del proyecto `backend-sistema-administrativo` fue ejecutado de forma manual desde el entorno local, empleando el escáner de SonarQube junto con el perfil de calidad "Administrador" previamente configurado.

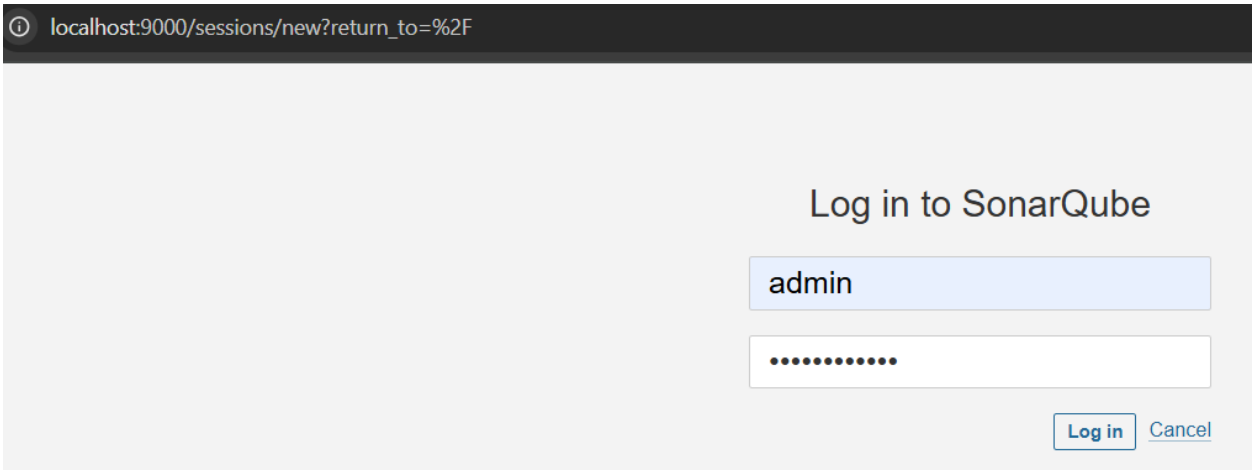
Previo a su ejecución, se levantó el servidor local de SonarQube mediante el comando:

C:\sonarqube\bin\windows-x86-64> StartSonar.bat

Una vez iniciado el servicio, se accedió al panel web de administración, como se muestra en la Figura 5.

Figura 5

Acceso al servidor web de SonarQube tras iniciar sesión como administrador.



Posteriormente, desde la raíz del proyecto, se ejecutó el siguiente comando:

C:\Users\Nathaly\Documents\PUCE-I\TESIS\backend-sistema-administrativo> sonar-scanner

2.2.4 Proceso de revisión de resultados

Al concluir el análisis con el perfil personalizado y el *Quality Gate* configurado, se obtuvieron resultados que evidencian el estado del código en términos de seguridad, mantenibilidad y duplicaciones. La Tabla 6 muestra un resumen de las principales incidencias identificadas, categorizadas por severidad y cantidad de ocurrencias.

Tabla 6

Resumen de incidencias encontradas

Nombre	Severidad	Cantidad
Literales de texto duplicados	Crítica	8
Alta complejidad cognitiva	Crítica	1
Uso de console.log()	Mayor	33
Asignaciones no utilizadas	Mayor	1
Uso innecesario de *await*	Menor	2

Estos hallazgos permitieron identificar con claridad los puntos críticos del código que deben ser atendidos, sirviendo como base para futuras mejoras.

2.2.5 Proceso de documentación y generación de reportes

Para respaldar los resultados del análisis estático, se empleó el complemento SonarQube CNES Report (v. 4.5.2), que facilita la exportación de informes en formatos DOCX, XLSX, CSV y Markdown. El archivo `.jar` fue descargado desde el repositorio oficial de GitHub e instalado en la carpeta `extensions/plugins` de SonarQube, versión 9.9.4 LTS. Tras reiniciar el servidor, el plugin quedó operativo.

Desde el panel de administración del proyecto se generó el informe en formato Word (`.docx`), con el objetivo de documentar los hallazgos de manera estructurada y legible dentro del trabajo de titulación. Este documento incluyó una descripción general del análisis, métricas clave del proyecto (como complejidad, duplicación, cobertura y líneas de código), el detalle de reglas incumplidas clasificadas por severidad y el historial de resultados por archivo, sirviendo como respaldo técnico para sustentar la evaluación realizada.

2.3 Metodología de análisis con Semgrep

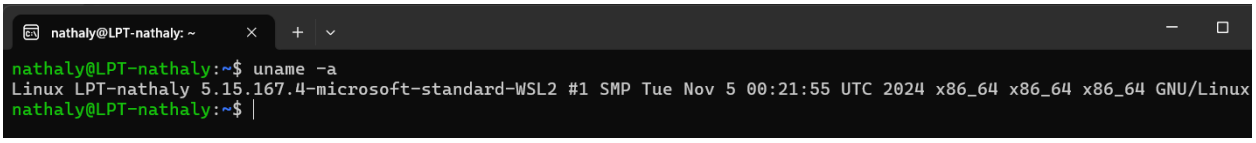
El análisis mediante Semgrep se realizó como una inspección estática personalizada, orientada a identificar vulnerabilidades y prácticas de codificación inadecuadas en el proyecto desarrollado con TypeScript. Esta herramienta fue seleccionada para complementar el análisis efectuado con SonarQube, dada su capacidad para aplicar reglas flexibles definidas en archivos YAML, lo que permitió adaptar los patrones de revisión al contexto específico del módulo administrativo del Sistema de Gestión Documental de la PUCE-I.

2.3.1 Preparación del entorno

El análisis con Semgrep se ejecutó en un entorno WSL (*Windows Subsystem for Linux*) sobre Windows 11, con el objetivo de garantizar compatibilidad con herramientas basadas en Unix y facilitar la ejecución de comandos en línea, como se observa en la Figura 6.

Figura 6

Entorno WSL utilizado para el análisis



El proyecto se localizó en la ruta `/mnt/c/Users/Nathaly/Documents/PUCE-I/TESIS/backend-sistema-administrativo`. La instalación de Semgrep se realizó mediante el comando:

```
pip install --user semgrep --break-system-packages
```

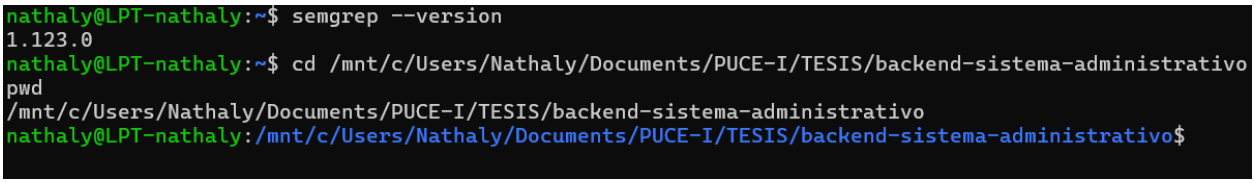
y su correcta configuración fue verificada con:

```
semgrep --version
```

La Figura 7 presenta un resumen de los comandos ejecutados durante la preparación del entorno.

Figura 7

Primeros comandos ejecutados en WSL



Para excluir directorios irrelevantes del análisis, se creó el archivo `.semgrepignore` con el siguiente contenido:

```
node_modules/

dist/
coverage/
test/
```

2.3.2 Configuración de reglas personalizadas

Se configuró un conjunto de 14 reglas personalizadas en formato YAML, desarrolladas a partir del perfil de calidad previamente establecido en SonarQube para el entorno de la PUCE-I. Estas reglas fueron adaptadas de forma específica al contexto técnico del módulo administrativo, manteniendo coherencia con los lineamientos institucionales en materia de seguridad, estilo y mantenibilidad del código.

El objetivo principal fue trasladar los criterios definidos en SonarQube al motor semántico de Semgrep, con el fin de identificar de manera más precisa prácticas inseguras susceptibles de manifestarse en tiempo de ejecución, errores lógicos frecuentes en proyectos TypeScript y patrones que evidencian un nivel reducido de calidad técnica.

La Tabla 7 presenta el conjunto de reglas implementadas, organizadas según el tipo de análisis que abordan.

Tabla 7

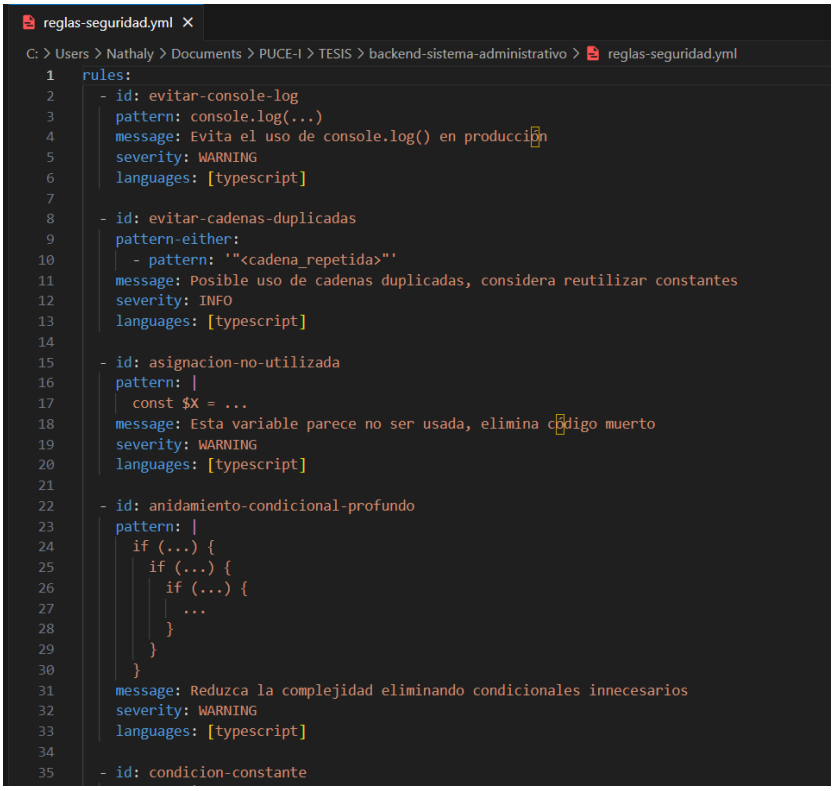
Reglas personalizadas implementadas en Semgrep para el análisis del módulo administrativo

ID de regla	Descripción
evitar-console-log	Evita el uso de `console.log()` en producción
evitar-cadenas-duplicadas	Detecta repetición de literales de texto
asignacion-no-utilizada	Identifica variables declaradas pero no utilizadas
anidamiento-condicional-profundo	Señala estructuras `if` anidadas innecesariamente
condicion-constante	Detecta condiciones siempre verdaderas o falsas
condicion-redundante	Marca verificaciones redundantes (`if (x === true)`)
comparar-con-booleano	Recomienda evaluar booleanos sin comparación explícita
catch-vacio	Evita bloques `catch` sin lógica de manejo
regex-insegura	Marca expresiones regulares potencialmente vulnerables (ReDoS)
evitar-any	Desaconseja el uso del tipo `any`
evitar-ts-ignore	Detecta uso de `@ts-ignore` sin justificación
for-in-sin-validacion	Identifica bucles `for-in` sin validación con `hasOwnProperty`
promesa-no-manejada	Detecta promesas no retornadas ni manejadas
reject-unauthorized-false	Reporta uso de `rejectUnauthorized: false` en agentes HTTPS

Nota. Elaboración propia basada en el perfil de calidad personalizado previamente configurado en SonarQube para el entorno PUCE-I.

Las reglas personalizadas fueron consolidadas en el archivo `reglas-seguridad.yml`, estructurado según la sintaxis de Semgrep en formato YAML. Este archivo centraliza la definición de los patrones utilizados en el análisis, facilitando su mantenimiento y futura reutilización. La Figura 8 muestra un fragmento representativo del archivo, evidenciando la alineación de las reglas con el perfil de calidad previamente configurado en SonarQube para el entorno de la PUCE-I.

Figura 8
Archivo reglas-seguridad.yml



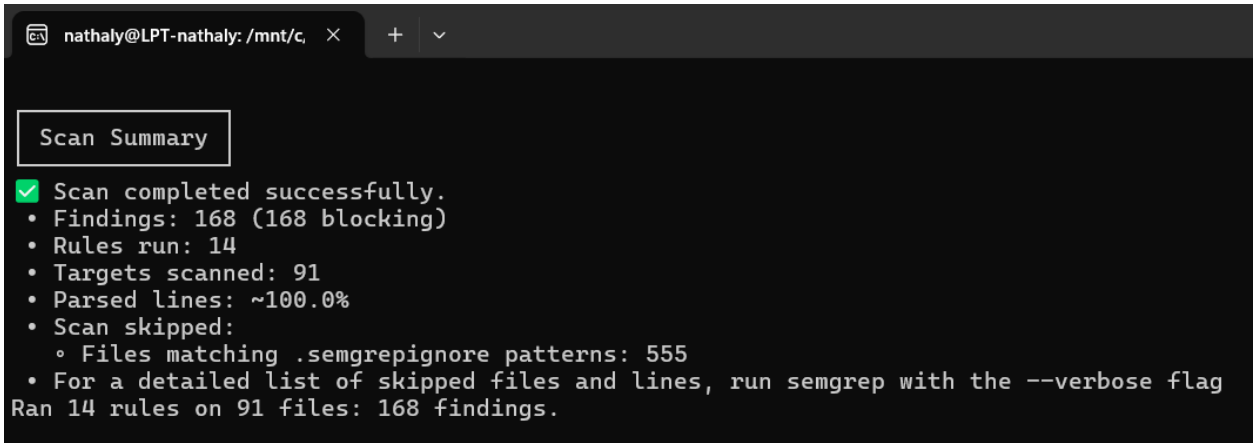
2.3.3 Ejecución del escaneo

El análisis se ejecutó desde la raíz del proyecto utilizando el comando `semgrep --config reglas-seguridad.yml .`, lo que permitió aplicar las 14 reglas personalizadas sobre el conjunto

de archivos fuente. En la Figura 9 se presenta el resumen del proceso, que confirma la correcta ejecución del escaneo sobre 91 archivos, generando un total de 168 hallazgos bloqueantes.

Figura 9

Resumen del análisis realizado con Semgrep desde la línea de comandos



2.3.4 Documentación y visualización de resultados

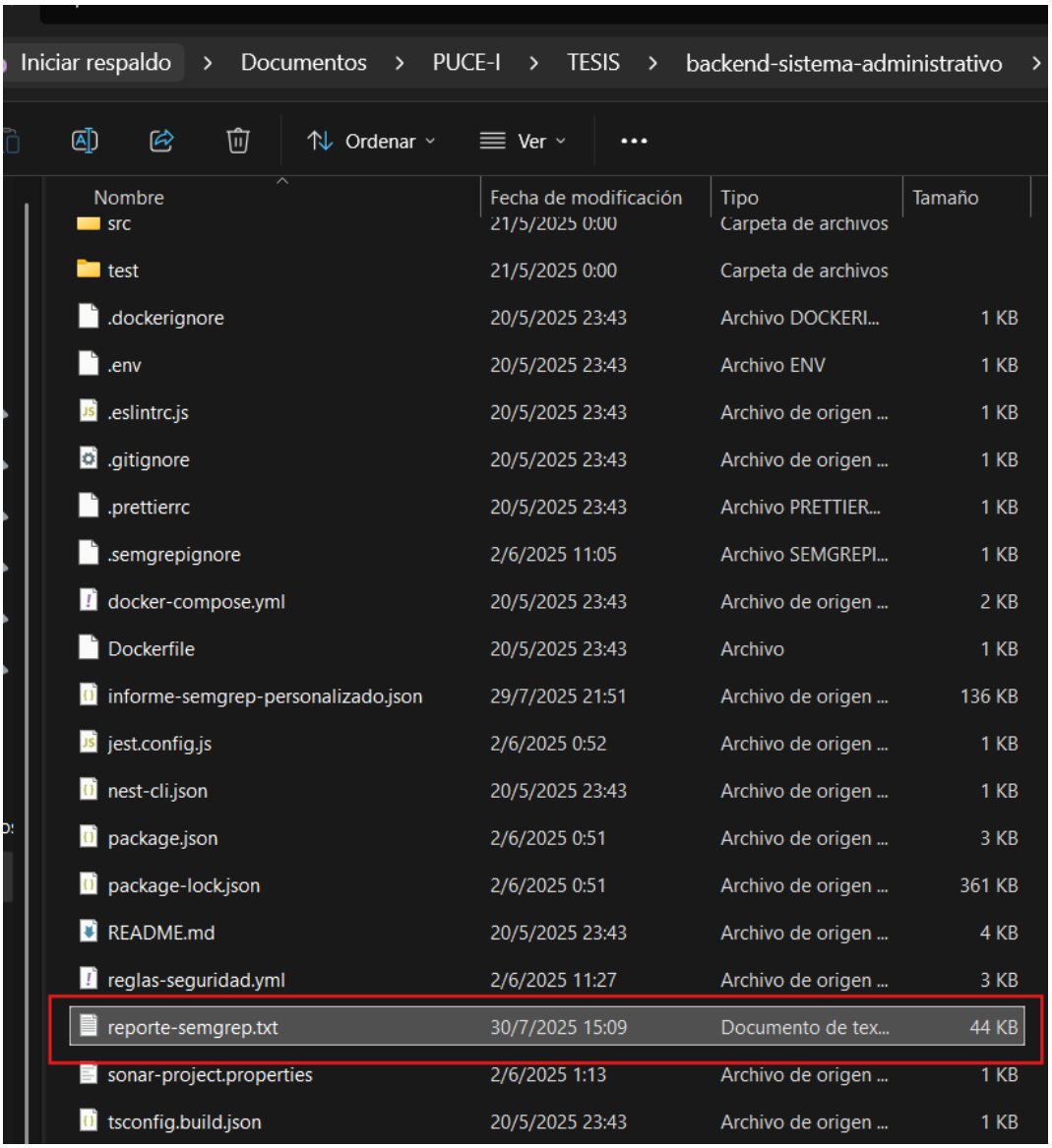
Con el objetivo de respaldar los hallazgos obtenidos, se ejecutó el comando:

```
semgrep --config reglas-seguridad.yml .
```

La salida generada por consola fue copiada manualmente y almacenada en un archivo de texto `reporte-semgrep.txt` para su posterior análisis. A partir de esta salida, se organizaron los resultados en una tabla estructurada según la ruta del archivo, número de línea, regla activada y observación correspondiente, como se muestra en la Figura 10.

Figura 10

Creación del archivo reporte-semgrep.txt



A partir del reporte consolidado se elaboró la Tabla 8, la cual presenta un resumen general de las reglas detectadas, indicando la descripción y la cantidad de incidencias asociadas a cada una. Los detalles individuales de cada hallazgo serán desarrollados posteriormente en el Capítulo III.

Tabla 8

Resumen de resultados Semgrep

Regla	Descripción	Cantidad
evitar-console-log	Salidas estándar en producción	12
evitar-ts-ignore	Uso indebido de @ts-ignore	14
asignacion-no-utilizada	Código muerto o no utilizado	109
reject-unauthorized-false	Desactivación de validación TLS (riesgo crítico)	2
Resto de reglas	Diversas prácticas inseguras o redundantes	31
Total		**168**

CAPÍTULO III

RESULTADOS Y DISCUSIÓN

Este capítulo presenta el análisis detallado de los resultados obtenidos durante la inspección estática realizada sobre el módulo administrativo del Sistema de Gestión Documental de la PUCE-I. Para ello se utilizaron las herramientas SonarQube y Semgrep, aplicando perfiles y reglas personalizadas adaptadas al contexto del proyecto. Se exponen los principales hallazgos en términos de vulnerabilidades, calidad y mantenibilidad del código, así como un análisis comparativo entre las incidencias detectadas por cada herramienta. Finalmente, se plantean propuestas técnicas orientadas a mitigar los riesgos identificados y fortalecer la seguridad del módulo evaluado.

2.4 Resultados del análisis estático con SonarQube

2.4.1 Métricas globales del proyecto

El análisis general del módulo administrativo realizado con SonarQube 9.9.4 LTS permitió cuantificar indicadores clave relacionados con el tamaño, la complejidad y la calidad técnica del código. La Tabla 9 resume las principales métricas obtenidas durante el escaneo estático.

Estas métricas muestran el volumen total del proyecto en líneas de código, el porcentaje de duplicación presente y los niveles de complejidad ciclomática y cognitiva, que influyen directamente en el esfuerzo necesario para entender y mantener el sistema. Asimismo, se reporta la densidad de comentarios como medida del soporte documental incorporado al código. Finalmente, los índices globales de fiabilidad, seguridad, mantenibilidad y cobertura de pruebas

proporcionan una visión consolidada del estado actual del proyecto frente a estándares de buenas prácticas.

Tabla 9

Métricas generales del análisis estático con SonarQube

Tamaño del Proyecto	Valor
Líneas de Código	2.892
Lenguaje	TypeScript
Comentarios	
Densidad de líneas comentadas	4,4%
Duplicaciones	
Porcentaje de duplicación	2,6%
Cobertura de pruebas	
Cobertura de código	0,0%
Complejidad	
Complejidad Ciclomática (máxima)	199
Complejidad Cognitiva (máxima)	103
Problemas por gravedad	
Problemas Críticos (Critical)	9
Problemas Mayores (Major)	34
Problemas Menores (Minor)	2
Bloqueadores, Vulnerabilidades y Bugs	0
Total de problemas detectados	**45**

2.4.2 Hallazgos de seguridad, mantenibilidad y calidad del código

El análisis estático del módulo administrativo se realizó empleando el perfil de calidad personalizado “Administrador”, configurado con 13 reglas clave adaptadas al contexto del proyecto. El Quality Gate reportó un estado general “OK”, lo que evidencia un cumplimiento aceptable de los umbrales definidos para fiabilidad, mantenibilidad y seguridad. No obstante, el escaneo identificó incidencias específicas que deben ser atendidas para fortalecer la robustez del

sistema. Los niveles de severidad fueron determinados conforme a la clasificación estándar de SonarQube, la cual categoriza las incidencias en críticos (rojo), mayores (amarillo) y menores (verde), en función del impacto potencial sobre la seguridad, calidad del código y su mantenibilidad. La Tabla 10 muestra un consolidado de los principales hallazgos detectados, organizados por tipo, severidad y cantidad de ocurrencias. Esta representación utiliza un código de color para destacar el nivel de criticidad de cada incidencia, facilitando su priorización.

Tabla 10

Incidencias detectadas por SonarQube clasificadas por tipo, severidad y cantidad

Nombre	Descripción	Tipo	Severidad	Cantidad
No se deben duplicar literales de texto	La duplicación de literales de texto hace que el proceso de refactorización sea propenso a errores, ya que se debe actualizar cada ocurrencia. En cambio, las constantes pueden ser referenciadas desde varios lugares y solo necesitan ser actualizadas en un único sitio. **Excepciones:** se excluyen los literales con menos de 10 caracteres, los que coincidan con `/^\w*$/`, los que estén dentro de sentencias `import/export` o atributos JSX,	CODE_SMELL	CRÍTICO	8

	así como los del estilo `'use strict';`.			
La Complejidad Cognitiva de las funciones no debe ser demasiado alta	La Complejidad Cognitiva mide qué tan difícil es entender el flujo de control de una función. Las funciones con alta complejidad cognitiva serán difíciles de mantener.	CODE_SMELL	CRÍTICO	1
No se deben usar directamente salidas estándar para registrar información	Las sentencias de depuración son útiles en desarrollo, pero si se incluyen en producción, especialmente en el lado del cliente, se corre el riesgo de exponer información sensible, ralentizar el navegador o incluso provocar errores. **Ejemplo no conforme:** `console.log(password_entered);` **Referencias:** OWASP Top 10 2021 - Fallos en el monitoreo,	CODE_SMELL	MAYOR	33

	OWASP Top 10 2017 - Exposición de datos sensibles.			
Se deben eliminar asignaciones no utilizadas	Una "asignación muerta" ocurre cuando una variable local recibe un valor que luego no se utiliza. Esto puede ser un error serio o, en el mejor de los casos, un desperdicio de recursos. **Excepciones:** Se ignoran asignaciones a `-1`, `0`, `1`, `undefined`, `[]`, `{}`, `true`, `false`, `""`; variables que inicien con `_`; asignación de `null`; expresiones de incremento o decremento; y casos con destructuración usando `rest`. **Referencias:** MITRE CWE-563.	CODE_SMELL	MAYOR	1
No se debe usar "await" de forma redundante	Una función `async` siempre envuelve el valor de retorno en una `Promise`. Usar `return await` no es necesario y tiene un costo extra en rendimiento. No obstante, a veces se mantiene para conservar el rastro	CODE_SMELL	MENOR	2

	de llamadas en caso de errores asíncronos. **Ejemplo no conforme:** `return await foo();` **Solución conforme:** `return foo();`		
TOTAL			**45**

2.4.3 Observaciones sobre las incidencias detectadas

El análisis estático permitió identificar un total de 45 incidencias, clasificadas por SonarQube en función de su severidad en críticas, mayores y menores. Entre los hallazgos más relevantes se destacan los siguientes:

Ocho casos críticos de duplicación de literales de texto, los cuales incrementan el riesgo de inconsistencias futuras, al obligar a replicar manualmente los cambios en múltiples ubicaciones del código. Este patrón dificulta la mantenibilidad y aumenta la posibilidad de defectos funcionales si alguna ocurrencia queda sin actualizar.

Una función con complejidad cognitiva excesiva, también clasificada como incidencia crítica. Este hallazgo implica que el flujo lógico del procedimiento es difícil de seguir, lo que eleva la probabilidad de introducir errores durante modificaciones o revisiones. Además, complica la tarea de pruebas unitarias exhaustivas.

33 usos de console.log() en ambiente productivo, categorizados como incidencias mayores. Estas llamadas directas a la salida estándar pueden exponer datos internos o información sensible,

entorpecer la depuración en entornos de producción y contravenir las recomendaciones del OWASP Top 10 respecto a un manejo adecuado del registro y monitoreo.

Finalmente, se detectó una asignación no utilizada y dos usos redundantes del operador await, clasificados como incidencias mayor y menor respectivamente. Aunque su impacto inmediato es bajo, reflejan un manejo ineficiente que afecta la claridad y puede incrementar innecesariamente el consumo de recursos.

Estos resultados evidencian que, aunque el módulo analizado cumple con el Quality Gate general configurado, existen aspectos internos que requieren atención prioritaria para asegurar la mantenibilidad, eficiencia y seguridad del código a largo plazo.

2.5 Resultados del análisis personalizado con Semgrep

El análisis realizado con Semgrep permitió identificar incidencias concretas vinculadas a llamadas de depuración en producción, directivas que omiten validaciones del compilador, estructuras redundantes y configuraciones críticas que podrían comprometer la verificación TLS. Las 14 reglas personalizadas aplicadas sobre el módulo administrativo facilitaron un examen puntual de estos aspectos, ampliando el alcance respecto a los hallazgos obtenidos previamente con SonarQube.

La Tabla 11 muestra el consolidado de los resultados, detallando cada regla implementada, su impacto estimado y la cantidad exacta de ocurrencias detectadas durante el escaneo del código.

Tabla 11

Incidencias detectadas por Semgrep según regla, severidad y ocurrencias

Regla	Descripción técnica	Severidad estimada	Ocurrencias detectadas
evitar-console-log	Detección del uso de `console.log` en entornos de producción, que puede exponer datos sensibles y contraviene buenas prácticas de monitoreo seguro.	Media	12
evitar-ts-ignore	Identificación de directivas `@ts-ignore` que omiten comprobaciones del compilador TypeScript, incrementando el riesgo de errores en tiempo de ejecución.	Media	14
asignacion-no-utilizada	Variables o asignaciones declaradas sin uso posterior, lo que genera código muerto y complica la mantenibilidad.	Baja	109
reject-unauthorized-false	Configuraciones HTTPS que desactivan la validación TLS (`rejectUnauthorized: false`), constituyendo un riesgo crítico al aceptar certificados no confiables.	Crítica	2
anidamiento-condicional-profundo	Estructuras `if` excesivamente anidadas que reducen la legibilidad y aumentan la complejidad cognitiva del flujo de control.	Media	4
condicion-constante	Validaciones con resultados siempre verdaderos o falsos, que pueden indicar errores lógicos no detectados.	Baja	6
condicion-redundante	Comparaciones redundantes (`if (x === true)`) que simplifican la lectura si se ajustan.	Baja	3
comparar-con-booleano	Comparaciones innecesarias con valores booleanos explícitos, afectando la claridad del código.	Baja	5
catch-vacio	Bloques `catch` sin lógica de manejo ni trazabilidad, lo que dificulta la detección de fallos en tiempo de ejecución.	Media	6
regex-insegura	Uso de expresiones regulares potencialmente vulnerables a ataques ReDoS (Regular expression Denial of Service).	Crítica	1
evitar-any	Declaraciones que utilizan el tipo `any`, debilitando el tipado estático y la detección temprana de errores.	Media	2
evitar-ts-ignore sin justificación	Usos de `@ts-ignore` sin comentarios aclaratorios que justifiquen su empleo.	Baja	2
for-in-sin-validacion	Iteraciones `for-in` sin comprobación `hasOwnProperty`, lo que puede procesar propiedades heredadas inesperadamente.	Media	1
promesa-no-manejada	Promesas que no son retornadas ni controladas, elevando el riesgo de errores asíncronos silenciosos.	Media	1

Total	**168**

2.5.1 Observaciones sobre las incidencias detectadas

El escaneo realizado con Semgrep detectó un total de 168 incidencias distribuidas en 14 reglas específicas. Entre ellas, sobresalen los 109 casos de asignaciones no utilizadas, que generan fragmentos de código muerto y dificultan la mantenibilidad, incrementando el riesgo de introducir errores durante modificaciones futuras.

Se identificaron además 14 directivas `@ts-ignore` que omiten validaciones del compilador TypeScript sin justificación, debilitando el control de tipado estricto y posibilitando fallos silenciosos en tiempo de ejecución. Junto a esto, los 12 registros mediante `console.log` expuestos en producción contradicen prácticas recomendadas de registro seguro, ya que podrían revelar información del flujo interno o datos sensibles en entornos operativos.

Resulta crítico destacar los dos casos de **`rejectUnauthorized: false`**, donde se desactiva la validación de certificados TLS en conexiones HTTPS, exponiendo directamente a ataques de intermediario (MITM), capaces de comprometer la confidencialidad e integridad de los datos transmitidos.

Adicionalmente, se detectaron incidencias relacionadas con estructuras condicionales anidadas innecesariamente, comparaciones redundantes, validaciones constantes y uso del tipo `any`, todas ellas afectando la legibilidad del código y aumentando la complejidad cognitiva requerida para su revisión o depuración.

2.5.2 Caso representativo de vulnerabilidad crítica

Durante el análisis personalizado con Semgrep se detectó una vulnerabilidad de alto impacto asociada a la desactivación explícita de la verificación de certificados TLS mediante el parámetro `rejectUnauthorized: false`. Esta configuración elimina el control que garantiza la validez de los certificados en conexiones HTTPS, exponiendo el canal de comunicación a posibles ataques de tipo hombre-en-el-medio (MITM), con el consiguiente riesgo de interceptación o manipulación de datos sensibles.

La Tabla 12 documenta los aspectos técnicos relevantes de esta incidencia, destacando su origen en el código, el impacto potencial sobre la confidencialidad e integridad de la información, y la recomendación inmediata para su remediación. Este hallazgo evidencia la importancia de mantener controles criptográficos robustos, conforme a las mejores prácticas descritas en el OWASP Top 10 (A02:2021 - Cryptographic Failures).

Tabla 12

Detalle técnico de vulnerabilidad crítica detectada

Elemento	Descripción
Vulnerabilidad	Desactivación de validación TLS (rejectUnauthorized: false)
Origen	Código desactiva verificación de certificados en conexiones HTTPS
Impacto	Expone a ataques de hombre-en-el-medio, robo de datos
Recomendación	Nunca usar esta configuración en producción; siempre usar certificados válidos y verificados

2.6 Análisis comparativo y propuesta técnica

El análisis estático aplicado con SonarQube y Semgrep evidenció coincidencias significativas respecto a la calidad técnica y las prácticas inseguras detectadas en el módulo administrativo. Si bien SonarQube identificó principalmente incidencias relacionadas con

duplicación de literales, alta complejidad cognitiva y uso excesivo de `console.log()`, Semgrep permitió profundizar en reglas personalizadas que identificaron variables no utilizadas, condiciones redundantes, estructuras condicionales anidadas y desactivación explícita de verificaciones TLS.

La Tabla 13 muestra una comparación consolidada de los tipos de incidencias detectadas por cada herramienta, resaltando cómo Semgrep complementa el alcance de SonarQube al abordar reglas adaptadas al contexto del sistema.

Tabla 13

Comparación de incidencias detectadas por SonarQube y Semgrep

Tipo de incidencia detectada	SonarQube	Semgrep
Literales duplicados	Detectado	No detectado
Complejidad cognitiva elevada	Detectado	No detectado
Uso de `console.log()` en producción	Detectado	Detectado
Variables asignadas pero no utilizadas	Detectado (mínimo)	Detectado
Uso innecesario del operador `await`	Detectado	No detectado
Directivas `@ts-ignore` sin justificación	No detectado	Detectado
Validación TLS deshabilitada (`rejectUnauthorized`)	No detectado	Detectado
Validaciones redundantes o condicionales complejas	No detectado	Detectado
Promesas no manejadas explícitamente	No detectado	Detectado

2.6.1 Matriz de riesgos del módulo administrativo

A partir del análisis realizado con SonarQube y Semgrep, se elaboró una matriz consolidada que clasifica los principales riesgos detectados en el módulo administrativo, considerando su severidad técnica, el tipo de vulnerabilidad, el impacto potencial sobre el sistema y las recomendaciones específicas para su mitigación. Este enfoque permite priorizar acciones

correctivas de manera fundamentada, orientando los esfuerzos hacia los elementos críticos que comprometen la seguridad, la mantenibilidad y la eficiencia operativa del software.

La Tabla 14 resume estos riesgos, alineándolos con las categorías establecidas por el OWASP Top 10, lo que facilita su interpretación bajo estándares reconocidos en la industria.

Tabla 14

Matriz de riesgos detectados en el módulo administrativo

Nº	Vulnerabilidad / Regla Detectada	Herramienta	Severidad	Cantidad / Frecuencia	Categoría / OWASP	Impacto Potencial	Recomendación
1	Literales de texto duplicados	SonarQube	Crítica	8	A04, A05	Mantenibilidad baja, errores de configuración	Usar constantes, refactorizar
2	Alta complejidad cognitiva	SonarQube	Crítica	1	A04	Difícil de mantener, mayor probabilidad de errores	Re-estructurar, dividir funciones complejas
3	Uso de console.log() en producción	SonarQube, Semgrep	Mayor	45 aprox.	A09	Información confidencial en logs, fuga de datos	Deshabilitar o eliminar logs en producción, usar sistemas de log seguros
4	Código muerto/asignaciones no utilizadas	SonarQube, Semgrep	Mayor	110 aprox.	A04	Carga innecesaria, posible ocultamiento de vulnerabilidades	Limpiar código regularmente, eliminar bloques y variables no usados
5	Uso innecesario de await	SonarQube	Menor	2	A04	Bajo desempeño, bloqueos innecesarios	Optimizar o eliminar llamadas asíncronas innecesarias
6	Uso indebido de @ts-ignore	Semgrep	Warning	14	A05	Oculta errores críticos en producción	Revisar y eliminar donde no sea estrictamente necesario
7	Desactivación de validación TLS (rejectUnauthorized: false)	Semgrep	Crítica	2	A02	Exposición a ataques MITM, pérdida de confidencialidad	Nunca desactivar validaciones TLS, usar certificados válidos
8	Prácticas inseguras varias: bucles sin validación, promesas no manejadas, regex insegura, anidamiento excesivo, condiciones redundantes	Semgrep	Warning/ Menor	31	A04, A05, A07, A08, A09	Comportamiento inesperado, ocultamiento de errores de lógica	Implementar lógica defensiva, revisar y corregir patrones inseguros

En complemento al análisis detallado, la Tabla 15 organiza los principales hallazgos en una matriz semafórica, relacionando el impacto técnico estimado —calificado en una escala de 1 a 5— con la frecuencia aproximada detectada durante el análisis del código. Esta representación permite visualizar de manera inmediata las áreas que requieren atención prioritaria para mitigar riesgos sobre la seguridad, mantenibilidad y rendimiento del módulo.

Los valores numéricos asignados responden a una escala técnica inspirada en criterios de evaluación de riesgo como el CVSS, donde:

1 indica un impacto leve, generalmente sin afectación funcional relevante;

4 corresponde a un impacto medio que podría comprometer la eficiencia o generar inconsistencias menores;

5 refleja un impacto crítico con potencial de comprometer la seguridad o estabilidad del sistema.

Esta calificación, combinada con la frecuencia observada (Muy Alta, Alta, Media-Baja), orienta la priorización de las acciones correctivas.

Tabla 15

Matriz priorización de riesgos del módulo administrativo

Prob./Impacto	**Leve (1)**	**Bajo (2)**	**Medio (3)**	**Alto (4)**	**Catastrófico (5)**
Muy Alta (5)	Uso de console.log	Código muerto	Literales duplicados	-	TLS validation OFF
Alta (4)	Await innecesario	Prácticas inseguras	@ts-ignore	Complejidad cognitiva	-
Media a baja (3-1)	-	-	-	-	-

A continuación, la Tabla 16 resume estos hallazgos organizándolos únicamente por nivel de criticidad técnica, lo que permite identificar de forma directa cuáles son las vulnerabilidades que demandan atención prioritaria.

Tabla 16

Prioridad técnica de vulnerabilidades detectadas

Severidad	Vulnerabilidad Principal
Crítica	Desactivación TLS, Alta complejidad, Literales duplicados
Mayor/Alta	Uso de console.log, código muerto/asignaciones no utilizadas
Media-Menor	Uso de await innecesario, @ts-ignore, prácticas inseguras varias

2.6.2 Propuesta técnica para la mitigación de riesgos

Para atender los riesgos identificados durante el análisis, se estableció un plan de mejora técnica estructurado inicialmente por tipo de vulnerabilidad, el cual prioriza acciones correctivas en función de su criticidad y facilita asignar responsables y plazos claros. Este plan se presenta en la Tabla 17.

Tabla 17

Plan de Mitigación por tipo de vulnerabilidad

Nº	Vulnerabilidad / Riesgo	Acción de Mitigación	Responsable	Prioridad	Plazo sugerido
1	Desactivación de validación TLS (rejectUnauthorized: false)	Eliminar la desactivación de TLS, usar siempre certificados válidos en servidores y clientes.	Equipo de desarrollo	Crítica	Inmediato
2	Uso de console.log() en producción	Remover o condicionar logs, usar sistema seguro de logging (con niveles y sin datos sensibles).	Desarrollo/QA	Alta	1 semana

3	Código muerto o asignaciones no utilizadas	Refactorizar y eliminar código innecesario mediante revisiones regulares y control de ramas.	Desarrollo	Alta	2 semanas
4	Uso indebido de @ts-ignore	Revisar cada uso y justificar documentalmente, eliminar donde no sea necesario.	Desarrollo	Media	2 semanas
5	Complejidad cognitiva alta	Rediseñar métodos críticos, dividir en funciones más pequeñas y documentar.	Desarrollo	Alta	2 semanas
6	Literales de texto duplicados	Mantener constantes globales, evitar duplicidad y actualizar documentación.	Desarrollo	Media	1 mes
7	Prácticas inseguras generales (e.g., bucles, regex, validaciones)	Aplicar linters, revisión de pares, reforzar validaciones y control de errores.	Todo el equipo	Media	Continuo
8	Uso innecesario de await	Optimizar llamadas asíncronas, revisar buenas prácticas de asincronía.	Desarrollo	Baja	1 mes

Para complementar este plan por tipo de riesgo, se elaboró una segunda propuesta técnica organizada por capas de la arquitectura del sistema, con el objetivo de abordar de manera integral la superficie de ataque y mejorar la calidad del software en todos los niveles. La Tabla 18 sintetiza estas recomendaciones específicas alineadas con las vulnerabilidades detectadas y el contexto tecnológico del módulo administrativo.

Tabla 18

Recomendación por Capa de Seguridad

Capa	Recomendaciones específicas	Herramientas / Acciones Sugeridas
Presentación	- Deshabilitar logs en producción - Validar exhaustivamente toda entrada del usuario - Evitar exposición de mensajes de error	Linter, frameworks front-end, manejo de excepciones
Aplicación	- Remover código muerto y duplicado - Estandarizar uso de constantes - Documentar código crítico - Limitar el uso de @ts-ignore	Code review, SAST (SonarQube, Semgrep)
Lógica de Negocio	- Validar permisos ante operaciones críticas - Controlar flujo de errores - Añadir pruebas unitarias y de integración	Revisiones de lógica, unit tests
Datos	- Evitar exposición de datos sensibles en logs - Uso estricto de ORM con validación - Enmascarar/anonimizar datos cuando sea posible	Sistemas de logging, enmascarado
Comunicación/Transporte	- Encriptar todo el tráfico usando TLS - Nunca desactivar validación de certificados - Hacer seguimiento de configuraciones inseguras	Configuración HTTPS, test de penetración
Infraestructura	- Revisar permisos de archivos y bases de datos - Implementar control de accesos y auditorías	Herramientas de DevOps, controles de acceso

	- Configurar backups y restauración

2.7 Resultados posteriores a la implementación de mejoras y comparación con el análisis inicial

2.7.1 Resultados posteriores con SonarQube

Tras la implementación de las mejoras derivadas del análisis inicial, se realizó un nuevo escaneo con SonarQube utilizando el perfil de calidad "Administrador". Los resultados muestran una disminución significativa en el número de incidencias y la eliminación total de las vulnerabilidades críticas y mayores previamente identificadas, manteniendo un estado general "OK" en todos los indicadores evaluados. La Tabla 19 presenta las incidencias detectadas en el análisis final, todas clasificadas como *code smells.*

Tabla 19

Incidencias detectadas en el análisis final SonarQube

Nombre	Descripción	Tipo	Severidad	Ocurrencias
Complejidad cognitiva de las funciones no debe ser demasiado alta	La complejidad cognitiva mide qué tan difícil es entender el flujo de control de una función. Funciones con alta complejidad cognitiva son difíciles de mantener.	Code Smell	Crítica	1
No se debe usar directamente la salida estándar para registrar información	Las sentencias de depuración son útiles durante el desarrollo, pero incluirlas en código de producción, especialmente en el lado del cliente, puede exponer información sensible, ralentizar el navegador o incluso provocar errores. Ejemplo no conforme: `console.log(password_entered);` (OWASP Top 10 2021 A9, 2017 A3).	Code Smell	Mayor	1
No se debe usar `await`	Una función `async` siempre envuelve el valor retornado en una promesa. Usar	Code Smell	Menor	1

de forma redundante	`return await` no es necesario y genera un costo extra de rendimiento, aunque puede conservar el rastro de llamadas en caso de errores asíncronos. Ejemplo no conforme: `return await foo();`.	
TOTAL		3

A continuación, la Tabla 20 presenta la comparación entre el análisis inicial y el análisis final, evidenciando la mejora en las métricas y la reducción de incidencias.

Tabla 20

Comparativa de métricas generales antes y después de las mejoras SonarQube

Métrica	**Análisis inicial**	**Análisis final**	**Variación**
Líneas de código	2.892	2.677	-215
% Duplicación	2,6 %	1,5 %	-1,1 %
Complejidad cognitiva máxima	103	110	+7
Complejidad ciclomática máxima	199	217	+18
Cobertura de código	0 %	0 %	=
Densidad de comentarios	4,4 %	3,6 %	-0,8 %
Severidad	**Análisis inicial**	**Análisis final**	**Variación**
Críticas	9	1	-8
Mayores	34	1	-33
Menores	2	1	-1
Vulnerabilidades	0	0	=
Bugs	0	0	=

Se evidenció una reducción considerable en las incidencias por severidad. Las críticas pasaron de 9 a 1, eliminando el 88,9 % de este tipo de hallazgos, mientras que las mayores disminuyeron de 34 a 1, lo que representa una reducción del 97,1 %. Las menores se redujeron de 2 a 1, y no se registraron cambios en vulnerabilidades o *bugs*, manteniéndose ambos en cero. Estos resultados reflejan que las correcciones implementadas no solo eliminaron las incidencias más graves, sino que dejaron únicamente tres *code smells* sin impacto directo en la seguridad, confirmando la efectividad del proceso de mejora y la solidez del estado actual del código.

2.7.2 Resultados posteriores con Semgrep

Tras implementar las mejoras derivadas del análisis inicial, se ejecutó un nuevo escaneo con Semgrep usando el mismo conjunto de reglas personalizadas. La tabla 21 presenta la comparación directa entre el análisis inicial y el análisis final:

Tabla 21

Comparativa de hallazgos Semgrep inicial y final

Regla	Descripción técnica	Severidad estimada	Ocurrencias iniciales	Ocurrencias finales	Variación
evitar-console-log	Uso de `console.log` en producción, posible exposición de datos sensibles.	Media	12	0	-12
evitar-ts-ignore	Uso de `@ts-ignore` que omite comprobaciones del compilador.	Media	14	0	-14
asignacion-no-utilizada	Variables o asignaciones sin uso posterior, generando código muerto.	Baja	109	109	=
reject-unauthorized-false	Configuración HTTPS que desactiva la validación TLS.	Crítica	2	0	-2
anidamiento-condicional-profundo	`if` anidados que reducen la legibilidad.	Media	4	0	-4
condicion-constante	Condiciones siempre verdaderas o falsas, posible error lógico.	Baja	6	0	-6
condicion-redundante	Comparaciones redundantes como `(x === true)`.	Baja	3	0	-3
comparar-con-booleano	Comparaciones innecesarias con	Baja	5	0	-5

	booleanos explícitos.				
catch-vacio	Bloques `catch` sin manejo de errores ni trazabilidad.	Media	6	0	-6
regex-insegura	Expresiones regulares vulnerables a ataques ReDoS.	Crítica	1	0	-1
evitar-any	Uso del tipo `any`, debilitando el tipado estático.	Media	2	2	=
evitar-ts-ignore sin justificación	`@ts-ignore` sin comentarios aclaratorios.	Baja	2	0	-2
for-in-sin-validacion	Uso de `for-in` sin comprobación `hasOwnProperty`.	Media	1	0	-1
promesa-no-manejada	Promesas sin control ni retorno, riesgo de errores asíncronos silenciosos.	Media	1	0	-1
Total		—	**168**	**143**	**-25**

En el análisis final con Semgrep se evidenció una mejora respecto al escaneo inicial, con una reducción total de 25 hallazgos, pasando de 168 a 143. Se eliminaron por completo todas las incidencias críticas, incluyendo *reject-unauthorized-false* y *regex-insegura*, lo que elimina riesgos directos de ataques de intermediario y de denegación de servicio por expresiones regulares inseguras. Asimismo, se resolvieron de forma total problemas de severidad media como *evitar-console-log*, *evitar-ts-ignore*, *anidamiento-condicional-profundo* y *catch-vacio*, así como varios hallazgos de baja severidad relacionados con condiciones redundantes o constantes.

La métrica *asignación-no-utilizada* se mantuvo estable en 109 ocurrencias, lo que indica que se controló el crecimiento de código muerto y se preservó la mantenibilidad sin modificar

lógicas esenciales. Otros hallazgos, como *evitar-any* y parámetros sin tipado estricto, permanecen sin cambios, representando áreas de mejora futura para fortalecer la robustez del código.

CONCLUSIONES

La aplicación combinada de SonarQube y Semgrep permitió identificar y priorizar vulnerabilidades críticas, malas prácticas de codificación y problemas de mantenibilidad en el módulo administrativo del Sistema de Gestión Documental de la PUCE-I, abarcando tanto incidencias genéricas como patrones inseguros específicos del contexto institucional.

El análisis inicial reveló 45 incidencias relevantes en SonarQube y 168 hallazgos en Semgrep, incluyendo riesgos críticos como la desactivación de la validación TLS, duplicación de literales y uso de `console.log()` en producción, los cuales podían comprometer la confidencialidad, integridad y trazabilidad del sistema.

Las mejoras implementadas redujeron en un 88,9 % las incidencias críticas y en un 97,1 % las de severidad mayor en SonarQube, eliminando por completo las vulnerabilidades críticas detectadas por Semgrep, lo que demuestra la efectividad del plan de mitigación propuesto.

La persistencia de métricas como “asignación-no-utilizada” y el uso de tipado débil (`any`) evidencia áreas de mejora futura orientadas a optimizar la mantenibilidad y robustez del código sin afectar su funcionalidad actual.

El alineamiento de los hallazgos con las categorías OWASP Top 10 permitió contextualizar los riesgos bajo un estándar reconocido, fortaleciendo la base técnica para futuras auditorías y asegurando la adopción de prácticas de desarrollo seguro.

RECOMENDACIONES

Mantener revisiones periódicas del código fuente con herramientas SAST como SonarQube y Semgrep, aplicando perfiles y reglas personalizadas alineadas con el contexto del sistema y estándares reconocidos como OWASP Top 10.

Eliminar configuraciones inseguras, en particular la desactivación de validaciones TLS (`rejectUnauthorized: false`), asegurando que todas las comunicaciones utilicen certificados válidos y verificados.

Sustituir el uso de `console.log()` en producción por sistemas de logging estructurado con niveles de severidad, evitando la exposición de datos sensibles y mejorando la trazabilidad de eventos.

Implementar procesos de limpieza de código para eliminar asignaciones y variables no utilizadas, reduciendo la complejidad y facilitando el mantenimiento.

Optimizar funciones con alta complejidad cognitiva mediante refactorización y división en métodos más simples, priorizando la legibilidad y facilidad de prueba.

Estandarizar el uso de constantes para evitar la duplicación de literales, centralizando su definición para facilitar futuras actualizaciones.

Reforzar la validación de entradas y el manejo de errores en todas las capas del sistema, complementando con pruebas unitarias y de integración que permitan prevenir fallos antes de la puesta en producción.

REFERENCIAS BIBLIOGRÁFICAS

3digits. (2024). *Seguridad en la gestión documental*. https://www.3digits.es/blog/seguridad-en-la-gestion-documental.html

Alghamdi, A. A. (2024). *Análisis de ciberataques y vulnerabilidades de ciberseguridad en el sector universitario*. *Computadoras, 14*(2), 49. https://doi.org/10.3390/computers14020049

Asociación Andaluza de Documentación. (2024). *Errores comunes en la gestión de documentos que afectan a la seguridad*. https://www.andaluzadedocumentacion.es/errores-comunes-en-la-gestion-de-documentos-que-afectan-a-la-seguridad/

Ayerdi, A. (21 de junio de 2024). *¿Qué es la gestión documental?* https://start.docuware.com/es/blog/que-es-la-gestion-documental

Bitegarden. (2025). *Análisis Estático del Código con SonarQube*. https://www.bitegarden.com/analisis-estatico-del-codigo-con-sonarqube

Canteli, A. (4 de noviembre de 2024). *La importancia de la Seguridad en la Gestión Documental: Consejos Prácticos*. OpenKM. https://www.openkm.com/es/blog/la-importancia-de-la-seguridad-en-la-gestion-documental-consejos-practicos.html

Camacho, R. (4 de abril de 2023). *Análisis estático y análisis dinámico.* https://es.parasoft.com/blog/static-analysis-and-dynamic-analysis/

Codacy. (25 de marzo de 2025). *Pruebas de seguridad de aplicaciones dinámicas (DAST)*. https://blog.codacy.com/what-is-dast

DataSunrise. (2024). *Mejores prácticas de seguridad para aplicaciones empresariales*. https://www.datasunrise.com/es/centro-de-conocimiento/mejores-practicas-seguridad-aplicaciones/

Dewhurst, R. (s.f.). *Análisis de código estático*. OWASP. https://owasp.org/www-

community/controls/Static_Code_Analysis?

ENISA. (29 de julio de 2021). *Threat Landscape for Supply Chain Attacks*. European Union Agency for Cybersecurity. https://www.enisa.europa.eu/publications/threat-landscape-for-supply-chain-attacks

Eshetu, A. M. (2024). *Vulnerabilidades y soluciones de ciberseguridad en sitios web de universidades etíopes*. *Revista de Big Data, 11*(1), 118. https://doi.org/10.1186/s40537-024-00980-z

Excentia. (2024). *Calidad de código y seguridad en tu flujo de trabajo*. https://www.excentia.es/calidad-de-codigo-y-seguridad-en-tu-flujo-de-trabajo

Fortinet. (2025). *¿Qué es la tríada CIA o CID?* https://www.fortinet.com/lat/resources/cyberglossary/cia-triad

Gartner. (13 de enero de 2022). *¿Qué es la seguridad del software?* https://www.gartner.com/en/articles/software-security

IBM. (14 de abril de 2021). *Política y objetivos de seguridad*. https://www.ibm.com/docs/es/i/7.3.0?topic=security-policy-objectives

IEEE Computer Society. (s.f.). *¿Qué es la seguridad del software y por qué es importante?* https://www.computer.org/resources/software-security

Jit. (9 de febrero de 2025). *Pruebas de seguridad de aplicaciones estáticas (SAST)*. https://www.jit.io/resources/appsec-tools/static-application-security-testing-sast-what-you-need-to-know

Kumar, A. S. (2023). *Una revisión exhaustiva de las vulnerabilidades, amenazas, ataques y soluciones de ciberseguridad*. *Electrónica, 12*(6), 1333. https://doi.org/10.3390/electronics12061333

Legit Security. (20 de marzo de 2025). *Understanding the Role of AI in Cybersecurity*. https://www.legitsecurity.com/aspm-knowledge-base/role-of-ai-in-cybersecurity

Limones, E. (23 de septiembre de 2022). *Análisis de vulnerabilidades informáticas: tipos y cómo prevenirlas*. OpenWebinars. https://openwebinars.net/blog/analisis-de-vulnerabilidades-informaticas/

MetaCompliance. (2021). *5 damaging consequences of a data breach*. https://www.metacompliance.com/es/blog/data-breaches/5-damaging-consequences-of-a-data-breach

NIST. (2020). *Security and Privacy Controls for Information Systems and Organizations.* Gaithersburg, MD: National Institute of Standards and Technology. https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final

OWASP. (2021). *OWASP Top 10: 2021*. https://owasp.org/Top10/es/

Pontificia Universidad Católica del Ecuador. (2025). *PUCE Ibarra*. https://puceinvestiga.puce.edu.ec/es/organisations/puce-ibarra

Ramos, L. (30 de enero de 2025). *Certificaciones en seguridad para la Gestión Documental: garantizar la protección y el cumplimiento normativo*. Athento. https://www.athento.com/es/certificaciones-en-seguridad-para-la-gestion-documental-garantizar-la-proteccion-y-el-cumplimiento-normativo/

Sancho, V. G. (2023). *Top 10 de vulnerabilidades en aplicaciones web*. https://vicentesg.com/top-10-vulnerabilidades-aplicaciones-web/

Securityium. (15 de octubre de 2024). *Comprensión de las pruebas de seguridad de aplicaciones*

estáticas. https://www.securityium.com/understanding-static-application-security-testing-key-benefits-and-trends/

Servernet. (s.f.). *Ciclo PDCA en seguridad informática: mejora continua*. https://servernet.com.ar/ciclo-pdca-seguridad-informatica/

Smowl. (6 de febrero de 2024). *¿Qué es una vulnerabilidad en la seguridad informática?* https://smowl.net/es/blog/vulnerabilidad-en-la-seguridad-informatica/

SonarSource. (2025). *SonarQube - Clean as You Code*. https://www.sonarsource.com/es/products/sonarqube/

Techesi. (14 de octubre de 2021). *22 vulnerabilidades comunes de aplicaciones web que debe conocer*. https://www.techesi.com/es/web-application-vulnerabilities.html

Vilañez Ordóñez, B. (2024). *Análisis forense de código y procesos de la aplicación Ubidesk de la empresa Uboratech*. Pontificia Universidad Católica del Ecuador. Ibarra: Trabajo de titulación. https://repositorio.puce.edu.ec/server/api/core/bitstreams/2b4b0c39-0724-4a13-a2a1-a096f900e9db/content

Wilson, S. V. (2022). *Ciberseguridad y educación superior: una revisión de las vulnerabilidades cibernéticas y su impacto en colegios y universidades*. ProQuest Dissertations Publishing. https://www.proquest.com/openview/073e3e48f60f8bad83b720d2d9684313/

ANEXOS

Pontificia Universidad Católica del Ecuador
Seréis mis testigos
IBARRA
DIRECCIÓN DE INVESTIGACIÓN, VINCULACIÓN Y EMPRENDIMIENTO

Ibarra, 13 de agosto de 2025

CARTA DE ACEPTACIÓN

Por medio de la presente, la Pontificia Universidad Católica del Ecuador Sede Ibarra (PUCE-I), a través de la Unidad de Desarrollo de Software Empresarial, tiene el agrado de informar que la Srta. **Nathaly Guadalupe Angamarca Angamarca**, estudiante de la carrera de Ingeniería en Tecnologías de la Información, realizó la entrega del proyecto de titulación titulado:

"Análisis de vulnerabilidades de software del módulo administrativo del sistema de gestión documental de la Pontificia Universidad Católica del Ecuador Sede Ibarra"

El proyecto fue desarrollado en el entorno de desarrollo proporcionado, utilizando herramientas de análisis estático de código para identificar vulnerabilidades y proponer mejoras de seguridad.

Se deja constancia de que el proyecto fue entregado correctamente y cumple con los requerimientos establecidos por la Unidad.

Atentamente,

JOSE LUIS IBARRA ESTEVEZ

Mgs. José Luis Ibarra Estévez
DIRECTOR - UNIDAD DE DESARROLLO DE SOFTWARE EMPRESARIAL PUCE-I