UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE POSGRADO

MAESTRÍA EN COMPUTACIÓN CON MENCIÓN EN SEGURIDAD INFORMÁTICA

TEMA:

"IMPLEMENTACIÓN DE UNA METODOLOGÍA PARA LA MITIGACIÓN DE VULNERABILIDADES FRONTEND BASADA EN OWASP ASVS Y CONTROLES PROACTIVOS"

AUTOR:

Ing. Abraham Jacobo Vallejos Males

DIRECTOR:

Msc. Xavier Mauricio Rea Peñafiel

IBARRA – ECUADOR

2026

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN
A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1004098446		
APELLIDOS Y NOMBRES:	ABRAHAM JACOBO VALLEJOS MALES		
DIRECCIÓN:	Imbabura, Ibarra, La Florida		
EMAIL:	ajvallejosm@utn.edu.ec		
TELÉFONO FIJO:	062631589	**TELÉFONO MÓVIL:**	0979753001

DATOS DE LA OBRA	
TÍTULO:	IMPLEMENTACIÓN DE UNA METODOLOGÍA PARA LA MITIGACIÓN DE VULNERABILIDADES FRONTEND BASADA EN OWASP ASVS Y CONTROLES PROACTIVOS
AUTOR (ES):	ABRAHAM JACOBO VALLEJOS MALES
FECHA: DD/MM/AAAA	07/02/2026
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	☐ **PREGRADO** ■ **POSGRADO**
TITULO POR EL QUE OPTA:	Magíster en Computación con mención en Seguridad Informática
ASESOR /DIRECTOR:	PhD. Irving Marlon Reascos Paredes, Msc. Xavier Mauricio Rea Peñafiel

2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 07 días del mes de Febrero de 2026

EL AUTOR:

Firmado electrónicamente por:
ABRAHAM JACOBO VALLEJOS MALES
Validar únicamente con FirmaEC

(Firma)……
Nombre: Abraham Jacobo Vallejos Males

Ibarra,10 de noviembre de 2025

Dr. Jorge Gordón
Decano
Facultad de Posgrado

ASUNTO: Conformidad con el documento final

Señor Decano:

Nos permitimos informar a usted que revisado el Trabajo final de Grado "IMPLEMENTACIÓN DE UNA METODOLOGÍA PARA LA MITIGACIÓN DE VULNERABILIDADES FRONTEND BASADA EN OWASP ASVS Y CONTROLES PROACTIVOS" del maestrante Abraham Jacobo Vallejos Males, de la Maestría en Computación con Mención en Seguridad Informática, certificamos que han sido acogidas y satisfechas todas las observaciones realizadas.

Atentamente,

	Apellidos y Nombres	**Firma**
Director	Msc. Xavier Mauricio Rea Peñafiel	Firmado electrónicamente por: XAVIER MAURICIO REA PEÑAFIEL Validar únicamente con FirmaEC
Asesor	PhD. Irving Marlon Reascos Paredes	IRVING MARLON REASCOS PAREDES Firmado digitalmente por IRVING MARLON REASCOS PAREDES Fecha: 2025.11.12 11:04:32 -05'00'

DEDICATORIA

A mi hijo **Adriel Mark Vallejos**, quien se ha convertido en la mayor inspiración de mi vida. Cada uno de mis esfuerzos, mis días de estudio y mis noches de trabajo tienen un propósito más profundo desde tu llegada. Este logro es para ti, para enseñarte que todo sueño se alcanza con perseverancia, disciplina y fe. Que este camino que hoy concluyo sea un ejemplo del amor inmenso que siento por ti y del deseo constante de construir un futuro mejor a tu lado.

AGRADECIMIENTO

Agradezco con profundo respeto y sinceridad a todas las personas que formaron parte de este proceso académico y personal. Cada palabra de aliento, cada consejo y cada gesto de apoyo contribuyeron significativamente a la culminación de este trabajo.

Mi agradecimiento especial es para mis padres, quienes han estado presentes en todas las etapas de mi vida, acompañándome con amor, paciencia y fortaleza. Gracias por enseñarme el valor del esfuerzo, por impulsarme a continuar aun en los momentos más difíciles y por creer en mí incluso cuando las circunstancias no eran fáciles. Este logro también es suyo.

Extiendo igualmente mi gratitud a quienes me brindaron apoyo académico, profesional y emocional a lo largo de este proyecto. A cada docente, compañero y familiar que aportó con su comprensión y respaldo, mi sincero reconocimiento.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

La capa frontend de las aplicaciones web modernas representa una superficie de ataque altamente expuesta, especialmente en arquitecturas Single Page Application (SPA), donde gran parte de la lógica y las interacciones se ejecutan en el navegador. Aunque existen estándares como OWASP ASVS y los Controles Proactivos, su aplicación en el desarrollo frontend suele ser limitada, generando brechas entre las buenas prácticas recomendadas y su implementación real. Ante esta necesidad, la presente investigación propone una metodología para mitigar vulnerabilidades en el frontend, adaptada a proyectos construidos con frameworks modernos como Angular. El estudio se realizó sobre la aplicación institucional ISC.TimeReport.FE, desarrollada bajo el modelo SPA. Se empleó un enfoque tecnológico y cuasi-experimental que permitió comparar el estado inicial del sistema con los resultados obtenidos tras aplicar la metodología propuesta. El análisis se efectuó mediante SonarQube y OWASP ZAP, utilizando evaluaciones estáticas (SAST) y dinámicas (DAST) para identificar fallas en validación de entradas, controles de acceso, exposición de datos sensibles y gestión de sesiones. La metodología se organizó en cinco fases: planificación, diseño, implementación, verificación y monitoreo, alineadas con dominios relevantes de OWASP ASVS y los Controles Proactivos. Su aplicación permitió reducir hasta un 80% las alertas detectadas por OWASP ZAP y disminuir los problemas de severidad mayor de 84% a 28% en el análisis estático.

Los resultados evidencian que la metodología es efectiva y replicable, contribuyendo a fortalecer la seguridad en el desarrollo frontend y a disminuir la brecha entre las recomendaciones internacionales y su implementación práctica.

Palabras clave: seguridad web, frontend, OWASP ASVS, Controles Proactivos, SPA, SonarQube, OWASP ZAP, análisis estático, análisis dinámico.

ABSTRACT

The frontend layer of modern web applications represents a highly exposed attack surface, particularly in Single Page Application (SPA) architectures where much of the logic and user interactions occur in the browser. Although standards such as OWASP ASVS and the Proactive Controls exist, their practical adoption in frontend development is often limited, creating gaps between recommended best practices and their real implementation. In response to this need, this research proposes a methodology to mitigate frontend vulnerabilities, adapted to projects built with modern frameworks such as Angular. The study was conducted on the institutional application ISC.TimeReport.FE, developed under the SPA model. A technological, quasi-experimental approach was applied to compare the system's initial security state with the results obtained after implementing the proposed methodology. Static and dynamic analyses were performed using SonarQube and OWASP ZAP (SAST and DAST) to identify weaknesses in input validation, access control, sensitive data exposure, and session management. The methodology was structured into five phases: planning, design, implementation, verification, and monitoring, aligned with relevant domains of OWASP ASVS and the Proactive Controls. Its application reduced up to 80% of the alerts detected by OWASP ZAP and lowered major-severity issues from 84% to 28% in the static analysis.

The results demonstrate that the methodology is effective and replicable, strengthening security in frontend development and reducing the gap between international recommendations and their practical adoption.

Keywords: web security, frontend, OWASP ASVS, Proactive Controls, SPA, SonarQube, OWASP ZAP, SAST, DAST.

CAPÍTULO I

EL PROBLEMA

1.1 Planteamiento del problema

En el desarrollo de aplicaciones web modernas, la capa *frontend* representa una de las superficies más críticas de exposición a amenazas. Al ser la interfaz directa entre el usuario y el sistema, cualquier descuido en su implementación puede comprometer la seguridad de la información. A pesar de la creciente adopción de marcos de seguridad en el *backend*, muchas organizaciones aún subestiman la importancia de aplicar controles sólidos desde el lado del cliente.

Ataques como *Cross-Site Scripting* (XSS), inyección de scripts, falsificación de peticiones o la manipulación de rutas son recurrentes debido a una gestión inadecuada de la validación de datos, autenticación, autorización y el manejo de sesiones. Estos problemas no solo ponen en riesgo la integridad de la aplicación, sino que también abren la puerta a la exposición de información sensible y accesos no autorizados.

El Open Web Application Security Project (OWASP) ha propuesto el estándar Application Security Verification Standard (ASVS) y los Controles Proactivos como guías fundamentales para el desarrollo seguro de software. Ambos marcos ofrecen un conjunto estructurado de requisitos y buenas prácticas orientadas a mitigar vulnerabilidades de forma anticipada en el ciclo de vida del software (OWASP Foundation, 2021a; 2023b). Sin embargo, estos recursos carecen de una metodología específica y adaptada al *frontend*, lo que dificulta su aplicación efectiva por parte de los equipos de desarrollo.

Estudios recientes evidencian que el desconocimiento o la implementación incorrecta de buenas prácticas de seguridad en el desarrollo *frontend* persiste en entornos reales, incluso en

equipos que aplican metodologías ágiles. Lindberg (2023), en un estudio de adopción del estándar ASVS en equipos de desarrollo, demuestra que, aunque se reconoce su valor, existe una brecha significativa entre la teoría y su correcta implementación práctica, particularmente en el *frontend*.

Ante esta problemática, se identifica la necesidad de diseñar e implementar una metodología clara y práctica, basada en OWASP ASVS y los Controles Proactivos, que permita a los desarrolladores de *frontend* integrar de forma efectiva prácticas de seguridad desde la codificación inicial. Esta metodología debe facilitar la identificación, mitigación y prevención de vulnerabilidades comunes, reduciendo la brecha entre desarrollo funcional y desarrollo seguro.

1.2 Interrogantes de la investigación

A partir del planteamiento del problema, surgen las siguientes preguntas de investigación que guían el desarrollo del presente estudio:

- ¿Cuáles son las vulnerabilidades más comunes en el desarrollo de aplicaciones *frontend* relacionadas con la autenticación, autorización, validación de datos y gestión de estado?
- ¿Cómo se pueden mapear estas vulnerabilidades con los controles definidos en OWASP ASVS y los Controles Proactivos?
- ¿Qué estructura metodológica puede facilitar la implementación práctica de dichos controles durante el ciclo de desarrollo *frontend*?
- ¿Qué mejoras en términos de seguridad se evidencian al aplicar la metodología propuesta en un entorno real de desarrollo?

1.3 Objetivos de la investigación

1.3.1 Objetivo general

Implementar una metodología basada en OWASP ASVS y Controles Proactivos para el desarrollo de *frontend*, que permita mejorar la mitigación de vulnerabilidades relacionadas con la autenticación, autorización, validación de datos y gestión de estado.

1.3.2 Objetivos específicos

- Identificar las vulnerabilidades comunes en el desarrollo de *frontend* relacionadas con el control de acceso, validación de datos y exposición de información sensible.
- Establecer la correspondencia entre las vulnerabilidades identificadas y los requisitos de OWASP ASVS y Controles Proactivos en el desarrollo de *frontend* seguro.
- Elaborar una guía de recomendaciones para el desarrollo seguro de *frontend*, que permita la mejora de la autenticación, autorización, validación de datos y gestión de estado

1.4 Hipótesis de trabajo

La implementación de una metodología basada en OWASP ASVS y Controles Proactivos para el desarrollo *frontend* mejora significativamente la mitigación de vulnerabilidades relacionadas con la autenticación, autorización, validación de datos y gestión de estado.

1.5 Hipótesis de investigación (H_1)

La implementación de una metodología basada en OWASP ASVS y Controles Proactivos para el desarrollo *frontend* permite mejorar significativamente la mitigación de vulnerabilidades relacionadas con la autenticación, autorización, validación de datos y gestión de estado.

1.6 Hipótesis nula (H_0)

La implementación de una metodología basada en OWASP ASVS y Controles Proactivos para el desarrollo *frontend* no mejora significativamente la mitigación de vulnerabilidades relacionadas con la autenticación, autorización, validación de datos y gestión de estado.

1.6.1 Categorización de variables

Para evaluar la hipótesis de esta investigación, se definen las siguientes variables:

- **Variable independiente (VI):** Implementación de una metodología basada en OWASP ASVS y Controles Proactivos para el desarrollo *frontend*.
- **Variable dependiente (VD):** Mitigación de vulnerabilidades relacionadas con autenticación, autorización, validación de datos y gestión de estado.

1.7 Justificación

El desarrollo seguro de software es una necesidad crítica en el entorno digital actual, donde las amenazas cibernéticas evolucionan constantemente y afectan tanto a grandes corporaciones como a pequeñas empresas. En particular, la capa *frontend,* que actúa como una interfaz directa con el usuario, representa una de las superficies de ataque más explotadas. Vulnerabilidades originadas por errores en la validación de entradas, una gestión inadecuada de sesiones, almacenamiento inseguro de datos o fallos en la autorización de accesos son comúnmente detectadas y explotadas en aplicaciones web en producción (OWASP Foundation, 2023).

Diversos estudios internacionales han evidenciado que los errores en el *frontend* no solo persisten, sino que son una de las causas más frecuentes de compromisos de seguridad. Rindell, Mäntylä, Hyrynsalmi y Leppänen (2021) evidencian, a partir de una encuesta a profesionales de desarrollo ágil, que la seguridad tiende a recibir menor prioridad que la entrega de funcionalidades.

Aunque marcos de referencia como los propuestos por OWASP son conocidos, su adopción práctica resulta limitada debido a los tiempos ajustados de entrega, un fenómeno presente incluso en organizaciones con experiencia en metodologías ágiles y prácticas DevOps.

Por otra parte, organismos como el NIST (National Institute of Standards and Technology) a través del marco SP 800-53 y la misma OWASP Foundation han impulsado el principio de "*Security by Design*", en el cual se establece que los controles de seguridad deben integrarse desde las fases más tempranas del ciclo de desarrollo. Sin embargo, estos marcos se presentan de forma general, sin directrices específicas aplicables a tecnologías *frontend* modernas como Angular, React o Vue.js (NIST, 2020; OWASP Foundation, 2021). Como resultado, los desarrolladores carecen de rutas prácticas para trasladar los principios de seguridad a su flujo de trabajo diario.

Adicionalmente, informes como los presentados por Veracode (2023) y Checkmarx (2022) han mostrado que más del 60% de las aplicaciones web evaluadas presentan al menos una vulnerabilidad de severidad alta relacionada con la interfaz del usuario. Esto se debe, en gran medida, a la subestimación del *frontend* como punto crítico de entrada, y a la dependencia excesiva de medidas de seguridad implementadas en el *backend*, lo cual es insuficiente en modelos modernos de arquitectura desacoplada (*frontend-backend* distribuido).

Desde un enfoque organizacional, los fallos de seguridad en el *frontend* pueden tener consecuencias operativas y reputacionales significativas, afectando directamente la experiencia del usuario, la confianza del cliente y el cumplimiento de normativas de protección de datos como el Reglamento General de Protección de Datos dentro de la Unión Europea o la Ley Orgánica de Protección de Datos Personales en Ecuador. La carencia de controles adecuados puede derivar en fugas de información confidencial, fraudes por suplantación de identidad, ataques de tipo *session hijacking*, o pérdida del control de los flujos de navegación de una aplicación.

Además, el *frontend* moderno ya no se limita a una interfaz visual estática, sino que contiene lógica de negocio relevante: validaciones en tiempo real, decisiones condicionales, control de formularios, y almacenamiento de tokens de sesión o credenciales. La seguridad de una aplicación no depende únicamente del *backend*, ya que el *frontend* también gestiona datos y flujos que requieren mecanismos de protección. Por ello, el desarrollo de *frontend* incluye responsabilidades explícitas en la implementación de medidas de seguridad.

Ante esta situación, se justifica la presente investigación que propone el diseño y validación de una metodología específica para el desarrollo seguro del *frontend*, tomando como base los estándares internacionales OWASP ASVS y Controles Proactivos. Esta metodología busca traducir dichos marcos en pasos prácticos, adaptados al flujo de trabajo real de los equipos de desarrollo, permitiendo así su implementación efectiva desde la codificación inicial.

El objetivo es doble: por un lado, reducir el riesgo técnico mediante la mitigación sistemática de vulnerabilidades comunes como XSS, *cross-site request forgery* **(**CSRF), inyecciones en rutas, fallos en autenticación y mal manejo de tokens; por otro lado, fomentar una cultura de seguridad entre desarrolladores, integrando la seguridad como parte del diseño funcional y no como una etapa posterior o correctiva.

Finalmente, esta propuesta tiene como propósito generar un efecto multiplicador, en el sentido de que la estandarización de una metodología replicable puede facilitar su adopción por equipos con distintos niveles de madurez tecnológica, contribuyendo a cerrar una brecha dentro de la industria del software. Se espera que sus resultados puedan ser utilizados como guía por organizaciones que buscan fortalecer su postura de seguridad, sin comprometer la productividad ni la agilidad en el desarrollo.

CAPÍTULO II

MARCO REFERENCIAL

2.1 Antecedentes

Se realizó una revisión de literatura preliminar con el fin de identificar estudios relevantes sobre la seguridad en aplicaciones web, con enfoque específico en el desarrollo *frontend* y metodologías aplicadas en el análisis de vulnerabilidades.

Para ello, se definió una cadena de búsqueda que permita responder a la interrogante:

- ¿Cómo integrar prácticas de seguridad en el desarrollo *frontend* utilizando estándares internacionales como OWASP ASVS y Controles Proactivos?

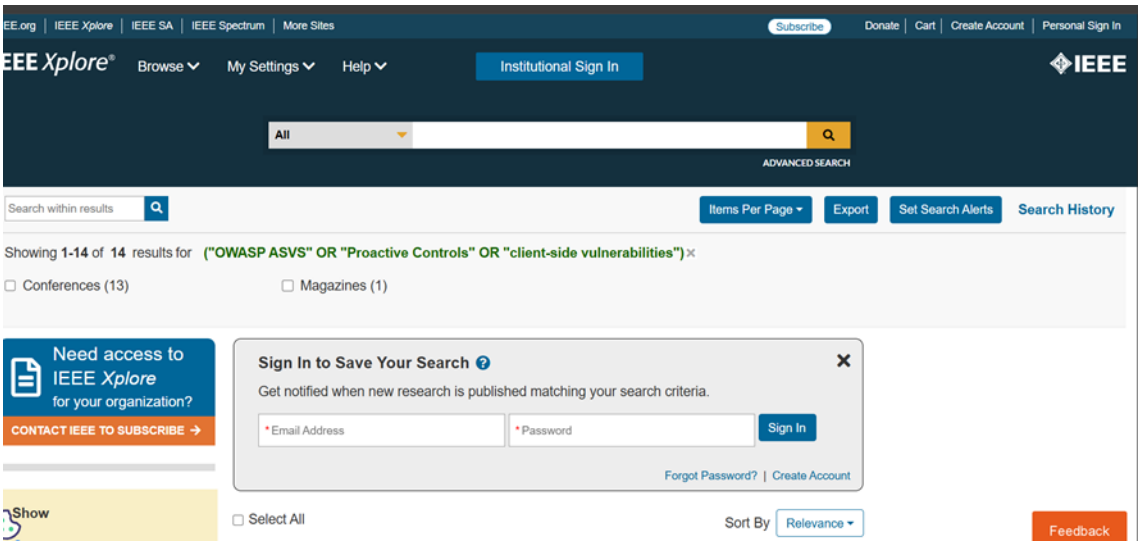
Cadena de búsqueda utilizada:

("OWASP ASVS" OR "Proactive Controls" OR "client-side vulnerabilities")

La Figura 1 presenta el resultado de la búsqueda realizada en la base de datos IEEE Xplore, utilizando los términos relacionados con OWASP ASVS y vulnerabilidades del lado del cliente.

Figura 1

Resultado de la búsqueda en la base de datos IEEE Xplore



Nota. Elaboración propia basada en IEEE Xplore (2025).

La aplicación de esta cadena permitió identificar tanto estudios conceptuales como propuestas metodológicas orientadas a la integración de seguridad en el ciclo de vida del software. Estos trabajos se diferencian en su alcance, ya que algunos enfatizan marcos de referencia generales y otros abordan técnicas específicas aplicables al desarrollo *frontend*.

La seguridad en aplicaciones web ha sido un tema ampliamente abordado en los últimos años, en especial ante la creciente sofisticación de los ciberataques. Uno de los enfoques emergentes ha sido integrar la seguridad directamente en el ciclo de vida del software, con especial atención al desarrollo *frontend*. Diversos estudios destacan la necesidad de adoptar estándares de seguridad desde las primeras etapas del desarrollo para prevenir vulnerabilidades comunes como XSS, CSRF y fallos en la gestión de sesiones (Brown et al., 2022).

Ekpobimi, Kandekere y Fasanmade (2024) proponen un marco conceptual para integrar la seguridad en el desarrollo *frontend* con el fin de mitigar vulnerabilidades comunes como XSS y

CSRF. El modelo plantea la incorporación de prácticas de codificación segura, modelado de amenazas, automatización de seguridad, integración en pipelines de integración y entrega continuas (CI/CD) y monitoreo continuo. Su enfoque resalta que la seguridad debe estar presente en cada etapa del desarrollo, lo que permite reducir de forma significativa la superficie de ataque en aplicaciones web y fortalecer la infraestructura digital.

El planteamiento de Ekpobimi et al. (2024) se complementa con los lineamientos de la OWASP Foundation (2023), que ofrecen controles proactivos de aplicación inmediata desde las etapas de diseño y codificación. Sin embargo, la literatura coincide en que la adopción de estas prácticas sigue siendo baja, principalmente en entornos de desarrollo ágil donde se prioriza la entrega rápida de funcionalidades (OWASP Foundation, 2023).

En contextos más amplios, como el planteado por el NIST (2020), se establece que la gestión proactiva de riesgos de software debe considerar todas las capas del sistema, incluyendo el *frontend*, como parte integral de la postura de seguridad. No obstante, gran parte de las guías disponibles carecen de aplicaciones prácticas específicas a entornos *frontend* modernos como Angular, React o Vue.js.

Sion, Yskout, Van Landuyt y Joosen (2018) proponen un análisis de seguridad basado en el diseño del sistema, conocido como *Risk-Based Design Security Analysis*, que permite evaluar amenazas desde la etapa de arquitectura. Su enfoque está pensado para sistemas críticos y entornos corporativos complejos.

Una línea emergente de investigación se enfoca directamente en el cliente, destacando los navegadores como componente crítico en la superficie de ataque. Cohen (2025) propone un marco de análisis de postura de seguridad en navegadores denominado *Browser Security Posture Analysis*

Framework, el cual opera completamente en el cliente mediante JavaScript y WebAssembly. Este enfoque permite ejecutar más de ciento veinte pruebas de seguridad directamente en el navegador, evaluando políticas como *Same-Origin*, intercambio de recursos de origen cruzado **(**CORS) y Content Security Policy (CSP), así como mecanismos de *sandboxing*, validación de certificados Secure Sockets Layer **(**SSL), uso de APIs criptográficas y comportamiento de memoria. El modelo también contempla riesgos asociados a extensiones y configuraciones inseguras, proporcionando así un diagnóstico integral del estado de seguridad del entorno cliente.

Entre sus principales aportes destaca la capacidad de observar decisiones internas del navegador que no pueden ser detectadas por herramientas de red o del sistema operativo, lo que resulta especialmente útil para identificar desconfiguraciones, extensiones maliciosas y vulnerabilidades que facilitan la exfiltración de datos. Además, el marco contribuye a la adopción de estrategias de *Zero Trust* y al cumplimiento de normativas como Health Insurance Portability and Accountability Act **(**HIPAA), el Estándar de Seguridad de Datos para la Industria de Tarjeta de Pago (PCI DSS) o por parte de la Organización Internacional de Normalización (ISO) el estándar ISO-27001, reforzando la consideración del navegador como un componente crítico dentro de la superficie de ataque empresarial.

De manera paralela, se han propuesto modelos cuantitativos que buscan objetivar la evaluación de seguridad, transformando controles cualitativos en métricas comparables. Liu et al. (2023) presentan un modelo cuantitativo para la evaluación de seguridad en aplicaciones web basado en el estándar ASVS. La propuesta busca transformar criterios de seguridad cualitativos en métricas numéricas, con el fin de calcular un puntaje global que refleje el nivel de protección de una aplicación. El modelo asigna ponderaciones a los diferentes controles de ASVS y permite

establecer comparaciones objetivas entre aplicaciones, además de identificar cuáles medidas tienen mayor impacto en la seguridad general.

Los resultados evidencian que la metodología facilita la priorización de esfuerzos de mitigación, dado que otorga una visión más estructurada de los riesgos en función de su peso cuantitativo. Asimismo, el modelo puede integrarse dentro del ciclo de desarrollo seguro como herramienta de auditoría, promoviendo la mejora continua de los sistemas evaluados. Entre las limitaciones señaladas, se encuentra la dependencia de la asignación subjetiva de ponderaciones y la dificultad de cuantificar con precisión algunos controles, lo que exige adaptar el enfoque al contexto particular de cada aplicación. Aunque estos enfoques ofrecen herramientas de evaluación, persisten propuestas centradas en la valoración de activos, cuyo aporte resulta útil, pero demanda ajustes para escenarios interactivos propios del *frontend*.

Tatar y Karabacak (2012) presentan un método jerárquico para la valoración de activos de información, el cual permite priorizar recursos y definir salvaguardas con base en la criticidad de cada activo. No obstante, al analizarse desde la perspectiva del desarrollo *frontend* moderno, se identifican limitaciones en cuanto a la consideración de dinámicas interactivas y asincrónicas, lo cual demanda ajustes metodológicos.

Finalmente, Kudriavtseva y Gadyatskaya (2022) realizan una revisión multivocal de la literatura sobre metodologías de desarrollo seguro de software, abarcando 28 propuestas provenientes de la industria, el gobierno y la academia. El estudio organiza las prácticas de seguridad identificadas según las fases del ciclo de vida del software, incluyendo tanto prácticas técnicas, siendo el modelado de amenazas o análisis estático, como prácticas auxiliares de carácter organizacional, legal o de gobernanza.

El trabajo evidencia que, aunque existen múltiples metodologías orientadas a la seguridad, persisten brechas en su validación empírica y en la integración de aspectos sociotécnicos. Asimismo, se resalta la necesidad de mayor investigación que evalúe la efectividad real de estas metodologías y promueva la producción de software con menos vulnerabilidades. La sistematización presentada ofrece a investigadores y organizaciones una base clara para comprender, comparar y mejorar las prácticas de seguridad en el desarrollo de software.

En conjunto, los estudios revisados muestran aportes relevantes y limitaciones específicas. Para resumir estos hallazgos de manera clara, a continuación, se presenta la Tabla 1 en comparativa con autores, puntos importantes y puntos críticos.

Tabla 1

Comparación de estudios relevantes sobre seguridad en aplicaciones web y desarrollo frontend

Autor/es	Puntos importantes	Puntos críticos
Brown et al. (2022)	Relevancia de integrar seguridad desde etapas tempranas del ciclo de vida; prevención de vulnerabilidades comunes como XSS, CSRF y gestión de sesiones.	Visión general sin un enfoque específico en *frontend* moderno.
Ekpobimi, Kandekere y Fasanmade (2024)	Marco conceptual para seguridad en *frontend*; prácticas de codificación segura, modelado de amenazas, CI/CD y monitoreo.	Requiere validación práctica y evidencia empírica de su efectividad.

OWASP (2023)	Controles Proactivos: lista actualizada de prácticas esenciales para desarrolladores.	Baja adopción en entornos de desarrollo ágil donde prima la rapidez.
NIST (2020)	Lineamientos de gestión proactiva de riesgos considerando todas las capas del sistema.	Carece de aplicaciones prácticas específicas para *frontend* en *frameworks* modernos.
Sion, Yskout, Van y Joosen (2018)	*Risk-Based Design Security Analysis*: evaluación de amenazas desde el diseño y arquitectura.	Orientado a entornos corporativos complejos; requiere adaptación para *frontend* ágil.
Cohen (2025)	Browser Security Posture Analysis: más de 120 pruebas *cliente-side*; diagnóstico integral de seguridad del navegador; soporte a Zero Trust y normativas.	Dependencia de tecnologías modernas de navegador; no sustituye otras capas de defensa.
Liu et al. (2023)	Modelo cuantitativo basado en OWASP ASVS; convierte controles cualitativos en métricas; priorización de riesgos.	Subjetividad en ponderaciones y dificultad de cuantificación en algunos controles.

Tatar y Karabacak (2012)	Método jerárquico para valoración de activos; priorización según criticidad.	No considera dinámicas asincrónicas propias del *frontend* moderno.
Kudriavtseva y Gadyatskaya (2022)	Revisión multivocal de 28 SSDMs; clasificación por fases del SDLC; integración de prácticas técnicas y organizacionales.	Persisten brechas en validación empírica e integración sociotécnica.

Nota. Elaboración propia (2025).

2.2 Marco Teórico

2.2.1 Seguridad en el Ciclo de Desarrollo de Software

El desarrollo de software seguro busca integrar controles de seguridad desde las primeras fases del ciclo de vida del software: análisis de requisitos, diseño, implementación, pruebas y mantenimiento (McGraw, 2006). Tradicionalmente, la seguridad era tratada como una etapa posterior o un añadido al final del desarrollo, lo cual ha demostrado ser ineficaz y costoso ante la presencia de vulnerabilidades en producción.

En este sentido, el enfoque *security by design* ha cobrado fuerza, ya que promueve que cada componente del sistema, incluyendo la capa cliente, incorpore mecanismos de protección desde su concepción. El NIST (2020) respalda este enfoque en sus marcos normativos, afirmando que la seguridad debe ser transversal a toda la arquitectura tecnológica, anticipándose a posibles amenazas en lugar de reaccionar ante ellas. Esta visión es importante en entornos modernos donde la interacción del usuario, la asincronía y el consumo de APIs externas son elementos constantes.

2.2.2 El *frontend* en aplicaciones web modernas

El *frontend* se define como la capa de presentación de una aplicación web, es decir, la parte visible e interactiva que el usuario percibe directamente en el navegador. Incluye todos los elementos visuales y dinámicos que permiten la experiencia de usuario, mientras que el *backend* corresponde a la lógica de negocio y al manejo de datos en el servidor, encargándose de procesar solicitudes y responder con la información necesaria para que el *frontend* funcione (Amazon Web Services, 2025).

En la Tabla 2 se presenta un resumen comparativo entre *frontend* y *backend*, donde se evidencian sus funciones, lenguajes y entornos de ejecución.

Tabla 2
Resumen de diferencias entre frontend y backend

	Frontend	***Backend***
Tecnologías	Utiliza HTML, CSS, JavaScript y marcos de *frontend*.	Utiliza lenguajes de programación como Java, Python, Ruby, API y sistemas de administración de bases de datos.
Simultaneidad	Cada usuario tiene su propia copia de una aplicación, por lo que el *frontend* no tiene que gestionar los problemas de concurrencia.	El *backend* utiliza varias estrategias para gestionar miles de solicitudes de usuarios al mismo tiempo.
Almacenamiento en caché	Los navegadores o las aplicaciones cliente almacenan en caché los archivos de la aplicación y los utilizan para mejorar el rendimiento.	Los sistemas de *backend* almacenan en caché los archivos en diferentes servidores o en una CDN.
Seguridad	La seguridad en el desarrollo del *frontend* recae en gran medida en los desarrolladores. Su responsabilidad principal es implementar flujos de trabajo seguros para la validación de datos ingresados por los usuarios y los procesos de autenticación.	La seguridad en el desarrollo del *backend* es más completa para proteger las bases de datos, los servicios de *backend* y la propia aplicación. Se logra mediante el cifrado, los sistemas de autenticación seguros y las prácticas de codificación seguras.

Objetivos de desarrollo	El desarrollo del *frontend* se centra en crear interfaces de usuario totalmente funcionales, con buena capacidad de respuesta y bien diseñadas.	El desarrollo del *backend* implica la creación de una arquitectura fiable que respalde el desarrollo del *frontend*.
Habilidades de desarrollo	Los desarrolladores de *frontend* conocen HTML, CSS y JavaScript. Pueden utilizar marcos de *frontend* y crear páginas visualmente atractivas. Estos marcos desentrañan los puntos débiles de los usuarios a medida que interactúan con la aplicación.	Los desarrolladores de *backend* cuentan con habilidades de codificación y administración de bases de datos. También entienden la seguridad del código y cómo utilizar las herramientas, plataformas y marcos de desarrollo de aplicaciones.

Nota. Adaptado de Amazon Web Services (2025).

Para el desarrollo del *frontend* se emplean principalmente HTML, CSS y JavaScript, considerados los pilares fundamentales de la web. Estos lenguajes trabajan de manera conjunta: HTML estructura el contenido, CSS define el diseño visual y JavaScript aporta interactividad. En entornos modernos, su uso se complementa con *frameworks* y librerías como Angular, React o Vue. Los *frameworks* proporcionan una estructura completa para desarrollar aplicaciones, mientras que las librerías son conjuntos de funciones reutilizables que los desarrolladores pueden integrar de manera flexible (MDN Web Docs, 2025). En la Tabla 3 se presentan sus principales características.

Tabla 3
Comparación entre frameworks y librerías de frontend

Aspecto técnico	Frameworks	Librerías
Estructura	Proveen un andamiaje completo que organiza la aplicación bajo patrones de arquitectura definidos (MVC, MVVM).	Ofrecen módulos o funciones específicas que se integran en el código existente.
Control del flujo	El *framework* establece la lógica central y dicta la forma en que se deben implementar los componentes (*inversión de control*).	El desarrollador mantiene el control principal y decide cómo y dónde utilizarlas.

Escalabilidad	Diseñados para proyectos de mediana y gran escala, facilitando la creación de aplicaciones de una sola página.	Adecuados para añadir funcionalidades concretas sin necesidad de reestructurar todo el proyecto.
Curva de aprendizaje	Mayor complejidad inicial debido a la cantidad de conceptos, herramientas y convenciones que integran.	Curva más corta, ya que se enfocan en resolver problemas específicos.
Aplicaciones típicas	Desarrollo de interfaces completas con gestión de rutas, estados y comunicación con APIs.	Incorporación de interactividad, manipulación del DOM o componentes aislados.

Nota. Adaptado de MDN Web Docs (2025).

2.2.3 Seguridad en aplicaciones de página única

Las aplicaciones de página única (SPA por sus siglas en inglés) son aplicaciones web que cargan un único documento y actualizan su contenido dinámicamente mediante JavaScript, por ejemplo, a través de la Fetch API, evitando recargas completas de página. Esto mejora la experiencia y el rendimiento percibido, aunque introduce compromisos adicionales como mayores esfuerzos para gestionar el estado, la navegación y el monitoreo del rendimiento (MDN contributors, 2025).

A diferencia de las aplicaciones tradicionales, en una SPA la carga inicial incluye los recursos base (HTML, CSS y JavaScript) y, a partir de ahí, la interfaz se reescribe en el cliente conforme el usuario interactúa, solicitando al servidor solo los datos y fragmentos necesarios en lugar de páginas completas. Este enfoque produce una navegación fluida, con menos interrupciones y tiempos de respuesta más rápidos al evitar renderizados completos en cada interacción (Lawson, 2025).

En términos de seguridad, su naturaleza dinámica plantea retos específicos de análisis: gran parte del contenido y del flujo se genera en el navegador, lo que dificulta el *spidering* y puede

dejar fuera elementos construidos por JavaScript si se usan métodos de escaneo tradicionales. Dado que las SPAs se emplean en procesos críticos del negocio, es fundamental someterlas a pruebas y escaneos de vulnerabilidades adecuados a su arquitectura (Fleetham, 2025).

2.2.4 Vulnerabilidades comunes en el frontend

El *frontend* constituye una de las capas más expuestas en aplicaciones web modernas, ya que su ejecución en el navegador lo convierte en un objetivo directo para atacantes. Entre las vulnerabilidades más relevantes, identificadas en el Top 10 por la OWASP Foundation (2021), se encuentran:

- **A01:2021 – Broken Access Control (*Pérdida de Control de Acceso*)**: fallas en los controles de acceso permiten que usuarios no autorizados accedan o modifiquen recursos.
- **A02:2021 – Cryptographic Failures (*Fallas Criptográficas*)**: uso inadecuado de algoritmos o configuraciones criptográficas que exponen datos sensibles.
- **A03:2021 – Injection (*Inyección*)**: incluye ataques como XSS e inyecciones SQL, donde entradas mal validadas ejecutan comandos maliciosos.
- **A04:2021 – Insecure Design (*Diseño Inseguro*):** ausencia de modelos de amenazas, patrones de diseño seguro o arquitecturas de referencia que prevengan riesgos desde el inicio.
- **A05:2021 – Security Misconfiguration (*Configuración de Seguridad Incorrecta*):** configuraciones por defecto, cabeceras HTTP inadecuadas o exposición de información sensible en el modelo de objetos del documento (DOM, por sus siglas en inglés).

- **A06:2021 – Vulnerable and Outdated Components (*Componentes Vulnerables y Desactualizados*)**: uso de librerías o dependencias con vulnerabilidades conocidas en *frameworks frontend*.
- **A07:2021 – Identification and Authentication Failures (*Fallas de Identificación y Autenticación*)**: errores en la gestión de sesiones o tokens que facilitan el secuestro de cuentas.
- **A08:2021 – Software and Data Integrity Failures (*Fallas en el Software y en la Integridad de los Datos*)**: ausencia de verificación en actualizaciones, pipelines CI/CD o datos críticos, lo que habilita manipulaciones.
- **A09:2021 – Security Logging and Monitoring Failures (*Fallas en el Registro y Monitoreo*)**: deficiencias en el registro y monitoreo que dificultan la detección temprana de ataques.
- **A10:2021 – Server-Side Request Forgery (*Falsificación de Solicitudes del Lado del Servidor*):** manipulación de peticiones desde el cliente hacia el servidor para acceder a recursos internos.

Estas categorías demuestran que muchas de las amenazas tradicionales, como XSS, CSRF o fallas en la autenticación, siguen vigentes y afectan directamente al *frontend*. Por ello, aplicar medidas de mitigación alineadas al OWASP Top 10 es un componente básico para reducir la superficie de ataque en aplicaciones modernas.

2.2.5 OWASP ASVS

El ASVS, desarrollado por OWASP, constituye un marco de referencia internacionalmente aceptado para evaluar la seguridad de aplicaciones web mediante requisitos verificables. El

estándar está organizado en tres niveles progresivos de rigurosidad: Nivel 1, que cubre controles esenciales aplicables a cualquier aplicación; Nivel 2, recomendado para sistemas que procesan información sensible o con funcionalidades críticas de negocio; y Nivel 3, destinado a aplicaciones de misión crítica y entornos de alto riesgo (OWASP Foundation, 2023).

El ASVS contempla catorce dominios de seguridad que abarcan desde autenticación y gestión de sesiones hasta criptografía, lógica de negocio y validación de entradas. No obstante, para esta investigación se priorizan aquellos apartados con mayor relevancia en el frontend, tales como el control de autenticación (V2), la gestión de sesiones (V3), la validación de entradas (V5) y la protección de datos sensibles (V10). Estos criterios permitirán orientar la verificación de seguridad hacia prácticas directamente aplicables a arquitecturas modernas de aplicaciones web.

2.2.6 OWASP Controles Proactivos

Los Controles Proactivos de OWASP constituyen una guía práctica dirigida a desarrolladores, cuyo objetivo es incorporar medidas de seguridad desde las etapas iniciales de codificación y diseño. A diferencia del Top 10 de vulnerabilidades ya explotadas, los Controles Proactivos buscan prevenirlas mediante buenas prácticas de desarrollo seguro (OWASP Foundation, 2023). La versión más reciente, publicada en el 2023 por la OWASP Foundation, presenta diez controles que cubren aspectos importantes en la construcción de software seguro, los cuales se presentan a continuación.

- **C1 - Implement Access Control (*Implementar Control de Acceso*):** Define y aplica mecanismos de autorización y políticas de acceso (por ejemplo, principios de mínimo privilegio y comprobaciones en los puntos de ejecución).

- **C2 - Use Cryptography the proper way (*Usar Criptografía de la manera correcta*):** Uso adecuado de cifrado y gestión de claves para proteger datos en tránsito y en reposo, evitando criptografía casera o configuraciones inseguras.
- **C3 - Validate all Input & Handle Exceptions (*Validar todas las Entradas y Manejar Excepciones*):** Validación y saneamiento de entradas (listas blancas, reglas por tipo) y gestión segura de errores/excepciones para evitar fugas de información y vulnerabilidades por entrada maliciosa.
- **C4 - Address Security from the Start (*Encargarse de la Seguridad desde el Inicio*):** Integrar requisitos de seguridad, modelado de amenazas y prácticas "shift-left" desde las primeras fases del ciclo de desarrollo.
- **C5 - Secure By Default Configurations (*Configuraciones Seguras por Defecto*):** Asegurar que las configuraciones por defecto sean seguras, minimizar superficie de ataque y eliminar elementos de desarrollo/depuración en producción.
- **C6 - Keep your Components Secure (*Mantener sus Componentes Seguros*):** Gestión de dependencias y componentes terceros (actualizaciones, verificación de vulnerabilidades, SCA) para reducir riesgos por componentes desactualizados o vulnerables.
- **C7 - Secure Digital Identities (*Implementar Identidad Digital):*** Prácticas robustas de autenticación, gestión de credenciales, sesiones y ciclo de vida de identidades digitales.
- **C8 - Leverage Browser Security Features (*Aprovechar las Características de Seguridad del Navegador*):** Uso de controles del navegador (CSP, SameSite, *headers* de seguridad, políticas de contenido) para reducir vectores de ataque en el cliente.

- **C9 - Implement Security Logging and Monitoring (*Implementar Registro y Monitoreo de Seguridad*):** Registro de auditoría y monitoreo continuo que permitan detectar, investigar y responder a incidentes de seguridad.
- **C10 - Stop Server Side Request Forgery (*Detener la Falsificación de Solicitudes del Lado del Servidor*):** Controles para evitar que el servidor realice solicitudes no deseadas (validación/whitelisting de URLs, protección de metadatos internos, límites de red).

La aplicación conjunta de estos controles en proyectos de desarrollo *frontend* es considerada una medida para la reducción de riesgos frecuentes, entre ellos XSS, CSRF y la exposición de información sensible. De igual forma, su integración ha sido planteada como un mecanismo para la incorporación de prácticas de seguridad en los equipos de desarrollo.

2.2.7 Herramientas de evaluación: ZAP y SonarQube

Para evaluar la seguridad antes y después de aplicar la metodología propuesta, se emplearán dos herramientas ampliamente utilizadas:

2.2.7.1 OWASP ZAP

El *Zed Attack Proxy (ZAP)* es una herramienta de seguridad de código abierto desarrollada por OWASP que funciona como un proxy de interceptación entre el navegador y la aplicación, permitiendo analizar e identificar vulnerabilidades durante la ejecución. ZAP ofrece funciones de escaneo activo y pasivo, manipulación de solicitudes y automatización de pruebas, lo que la convierte en una opción accesible y eficaz para evaluar aplicaciones web (OWASP Foundation, 2024).

2.2.7.2 SonarQube

SonarQube es una plataforma de análisis de código estático que examina el código fuente sin necesidad de ejecutarlo, con el objetivo de identificar errores, vulnerabilidades y problemas de calidad en etapas tempranas del ciclo de desarrollo. Mediante reglas predefinidas y análisis basados en árboles de sintaxis abstracta, la herramienta permite a los equipos detectar inconsistencias de seguridad, estilos de codificación deficientes y fallos lógicos antes de que lleguen a producción (SonarSource, 2023).

2.2.8 Técnicas de análisis estático y dinámico

El análisis estático de seguridad de aplicaciones (SAST por sus siglas en inglés) es una técnica de verificación *white-box* que examina el código fuente, los binarios o el *bytecode* de una aplicación antes de su ejecución. Su objetivo es identificar vulnerabilidades desde etapas tempranas del ciclo de desarrollo, facilitando correcciones inmediatas y reduciendo costos asociados a fallas descubiertas en producción (OpenText, 2025).

En contraste, el análisis dinámico de seguridad de aplicaciones (DAST por sus siglas en inglés) corresponde a un enfoque *black-box* que evalúa aplicaciones en ejecución, simulando ataques reales para detectar vulnerabilidades como inyecciones, fallas de autenticación o errores de configuración. A diferencia del SAST, no requiere acceso al código, sino que analiza la aplicación desde la perspectiva del atacante, ofreciendo una visión práctica de su nivel de exposición (OpenText, 2025).

La aplicación de estas técnicas en SPAs presenta desafíos particulares. Debido a que gran parte del contenido y las interacciones se generan dinámicamente en el navegador, los escáneres

tradicionales suelen pasar por alto componentes críticos. En este contexto, la adaptación de DAST resulta clave para evaluar de forma efectiva las rutas, llamadas a APIs y flujos asincrónicos de las SPA, asegurando que la validación de seguridad cubra todo el ciclo de interacción del usuario (Astra Security, 2023).

CAPÍTULO III

MARCO METODOLÓGICO

3.1 Descripción del área de estudio

El presente estudio se desarrolla sobre la capa frontend del sistema *ISC.TimeReport.FE*, una aplicación web institucional de tipo *Single Page Application* (SPA) orientada a la gestión de tiempos y actividades. Dada su naturaleza dinámica y su interacción directa con el usuario, esta capa constituye una parte de valor significativo para la evaluación de seguridad, al representar el punto de contacto más expuesto dentro del entorno del sistema.

En este contexto, la investigación aplica una metodología de verificación técnica fundamentada en los lineamientos del Application Security Verification Standard (ASVS) y los Controles Proactivos de OWASP, enfocada en los dominios de autenticación, gestión de sesiones, validación de entradas y protección de datos sensibles. Dicho enfoque permite evaluar la conformidad del sistema frente a buenas prácticas de desarrollo seguro desde la perspectiva del cliente web.

Las herramientas SonarQube y OWASP ZAP se incorporan de manera complementaria al proceso metodológico, con el propósito de corroborar los hallazgos y verificar la eficacia de las medidas aplicadas, sin constituir el eje central de la investigación. De esta manera, se preserva el carácter analítico del estudio dentro de un marco de validación sustentado en estándares internacionales.

La Tabla 4 resume las principales tecnologías empleadas en la capa *frontend* del sistema *ISC.TimeReport.FE*, describiendo su función dentro del entorno de desarrollo y validación.

Tabla 4

Principales tecnologías y herramientas del proyecto ISC.TimeReport.FE

Tecnología / Herramienta	Tipo / Versión	Función dentro del proyecto
Angular	Framework (TypeScript)	Base principal del *frontend*. Gestiona componentes, servicios, enrutamiento y el patrón SPA.
TypeScript	Lenguaje de programación	Aporta tipado estático y mejora la mantenibilidad del código.
HTML5 / SCSS (CSS3)	Lenguajes de estructura y estilo	Definen la interfaz visual y el diseño adaptable.
RxJS	Librería reactiva	Permite manejar flujos de datos asíncronos mediante observables.
Bootstrap	Biblioteca de estilos	Facilita el diseño *responsive* de las vistas.
Node.js / npm	Entorno de ejecución y gestor de paquetes	Permiten la compilación y gestión de dependencias del proyecto.
Angular CLI	Herramienta de línea de comandos	Automatiza tareas de construcción, pruebas y despliegue.
Docker / Docker Compose	Contenedorización	Permiten ejecutar el entorno del *frontend* en contenedores aislados junto con Nginx.
Nginx	Servidor web	Sirve los archivos compilados y gestiona las peticiones HTTP.
SonarQube	Herramienta SAST	Evalúa la calidad y seguridad del código fuente.
OWASP ZAP	Herramienta DAST	Verifica vulnerabilidades a nivel de ejecución y respuesta del servidor.
Git / GitHub	Control de versiones	Gestiona la trazabilidad y colaboración del código.
Visual Studio Code	Entorno de desarrollo	Editor utilizado para la codificación y depuración.

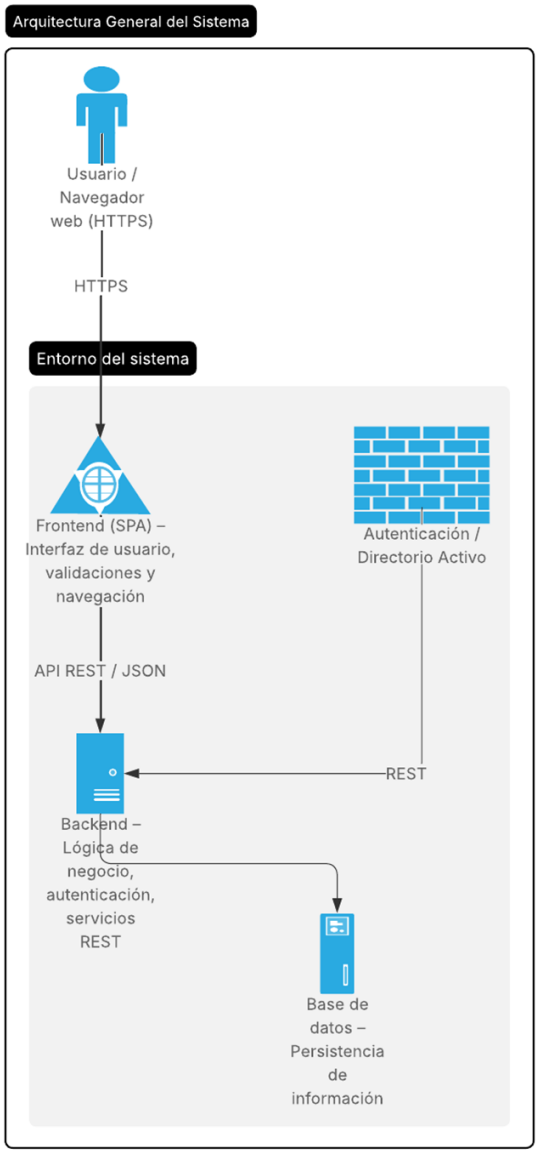
Nota. Elaboración propia (2025).

La Figura 2 presenta la arquitectura general del sistema *ISC.TimeReport.FE*, en la cual se observa la interacción entre la capa *frontend* de tipo SPA, el *backend* que gestiona la lógica de negocio y la autenticación, así como los servicios de persistencia de datos y directorio activo. Esta organización refleja un entorno típico de aplicaciones web institucionales, donde el cliente y el

servidor intercambian información mediante servicios REST sobre HTTPS, manteniendo la separación de responsabilidades y facilitando la evaluación de seguridad desde el lado del cliente.

Figura 2

Arquitectura del sistema evaluado



Nota. Elaboración propia (2025).

3.2 Enfoque y tipo de investigación

La investigación adopta un enfoque tecnológico y aplicado, al fundamentarse en la utilización de estándares reconocidos de seguridad del software dentro de un entorno real de desarrollo. Esta perspectiva permite trasladar los lineamientos de ASVS y los Controles Proactivos a la práctica, evaluando su aplicabilidad y efectividad en la verificación de un sistema existente.

Por su naturaleza, el estudio es descriptivo y analítico, ya que documenta las condiciones observadas en la capa *frontend*, identifica las vulnerabilidades presentes y examina su correspondencia con los criterios establecidos por OWASP. Asimismo, incorpora un alcance comparativo limitado, que permite contrastar el estado inicial del sistema con los resultados obtenidos tras la aplicación de ajustes enfocados en la mejora de seguridad, sin desbordar el alcance metodológico ni convertir las herramientas de apoyo en objeto principal de investigación.

3.3 Diseño metodológico

Esta investigación tiene un diseño cuasi-experimental basado en la comparación del estado inicial y final de la aplicación evaluada. La unidad de análisis será una aplicación web de mediana complejidad, desarrollada en un entorno moderno (por ejemplo, Angular, React o Vue), lo cual permite evaluar con precisión las vulnerabilidades típicas del *frontend* y aplicar las recomendaciones propuestas.

Para medir el impacto de la metodología, se utilizarán herramientas de análisis de seguridad como:

- OWASP ZAP: Para ejecutar pruebas dinámicas (DAST) que simulen ataques reales sobre la aplicación.

- SonarQube: Para realizar análisis estático del código fuente (SAST) y detectar fallas comunes de codificación insegura.

Se compararán los resultados de seguridad antes y después de la intervención, mediante indicadores como número de vulnerabilidades, nivel de severidad y cumplimiento de controles definidos por OWASP.

3.4 Procedimiento de investigación

El proceso metodológico se organizó en cinco fases secuenciales, conforme se indica a continuación:

Fase 1. Revisión documental y normativa: Se recopilaró información de fuentes académicas y técnicas relacionadas con OWASP ASVS, los Controles Proactivos, NIST SP 800-53 y otras buenas prácticas de desarrollo seguro. Esta etapa fundamentó el marco teórico y la estructura metodológica.

Fase 2. Diseño de la metodología: Se elaboró una metodología basada en los requisitos de OWASP ASVS y los Controles Proactivos, específicamente adaptada al contexto *frontend*.

Fase 3. Identificación de vulnerabilidades comunes: Mediante herramientas de análisis estático (SonarQube) y dinámico (OWASP ZAP), se evaluó un proyecto *frontend* para detectar fallas como validaciones deficientes, controles de acceso débiles y exposición de datos sensibles.

Fase 4. Implementación en un proyecto real: La metodología fue aplicada en una aplicación web en desarrollo, permitiendo evaluar la efectividad práctica de los controles propuestos.

Fase 5. Evaluación de resultados: Nuevamente se aplicaron las pruebas de seguridad empleadas en la Fase 2 para realizar un análisis comparativo entre estos resultados para medir el impacto de la metodología en términos de reducción de vulnerabilidades.

3.5 Consideraciones bioéticas

La presente investigación no involucra la participación de personas ni la manipulación de datos sensibles. Sin embargo, se aplicarán medidas estrictas para garantizar la confidencialidad del código fuente, entornos de prueba y documentación técnica utilizada.

Se observarán los principios de ética profesional: honestidad, respeto por los derechos de autor, uso responsable de herramientas de seguridad y transparencia en la presentación de resultados. Los hallazgos se difundirán con fines académicos, promoviendo el desarrollo de software seguro como responsabilidad social en el campo tecnológico.

CAPÍTULO IV

RESULTADOS

4.1 Diseño de la metodología para la mitigación de vulnerabilidades frontend basada en OWASP ASVS y Controles Proactivos

Dentro del ámbito del desarrollo de software coexisten dos paradigmas metodológicos: aquellas tradicionales, caracterizadas por su impulso hacia la rigurosidad procesal y la secuencialidad lineal de sus fases; y las ágiles, que priorizan las entregas mediante ciclos iterativos de retroalimentación continua con el cliente (Crêspo Boaventura, Peña Herrera, Verdecia Vicet, & Fustiel Alvarez, 2016).

La metodología propuesta en la presente investigación se fundamenta en los principios de las metodologías ágiles, decisión sustentada por su notable tasa de adopción que alcanza el 94% en organizaciones a nivel global, según datos del 15th Annual State Of Agile Report (Akiwatkar, 2025). Este éxito sobre las metodologías tradicionales se debe a la implementación de una comunicación y coordinación efectiva por parte del equipo de desarrollo, utilizando mecanismos como seguimiento del progreso del proyecto y una apropiada documentación (Binboga & Altin Gumussoy, 2024). De manera complementaria, la presente investigación adoptó el enfoque de Design Science Research (Hevner, March, Park, & Ram, 2004) como marco metodológico para el diseño de la metodología propuesta, con la finalidad de validar su utilidad y eficacia mediante criterios de rigor académico.

Dado que las metodologías ágiles suelen presentar fases comunes en su estructura operativa, esta investigación presenta una metodología la cual ha sido diseñada para integrarse sobre las etapas del ciclo de vida de cualquier metodología ágil existente. Las fases establecidas

corresponden a: planificación, diseño, implementación, verificación y monitoreo, componiendo una estructura análoga a las etapas de las metodologías ágiles convencionales. La definición de estas fases se fundamenta en los Controles Proactivos y ASVS de OWASP, permitiendo determinar las instancias específicas del ciclo de desarrollo donde dichos controles de seguridad deben ser implementados. En la Tabla 5 se presenta una comparativa que establece la correspondencia entre las verificaciones del estándar ASVS (V), los controles proactivos (C), y las fases de la metodología propuesta.

Tabla 5

Comparación entre las fases de la metodología propuesta con los Controles Proactivos y ASVS

Fase de la metodología	Control Proactivo	Verificación ASVS
4.1.1Planificación	C1, C7	V1, V4
4.1.2 Diseño	C1, C2, C3, C6, C7, C8	V1, V2, V3, V4, V8, V9, V14
4.1.3 Implementación	C2, C3, C4, C5, C6 C8, C9, C10	V2, V3, V5, V6, V7, V8, V9, V12, V14
4.1.4 Verificación	C5, C7, C10	V4, V5, V7, V9, V14
4.1.5 Monitoreo	C8, C9	V7, V8, V14

Nota. Elaboración propia (2025).

4.1.1. Planificación

La primera fase busca materializar el concepto de *security by design* para que las consideraciones ante los riesgos de seguridad constituyan requisitos fundacionales del proyecto desde su creación, en lugar de adiciones retroactivas (Sethi, 2024). Para esto, es primordial realizar

una especificación funcional de la aplicación, un proceso análogo al diseño de casos de uso en ingeniería de software tradicional, donde se describan todas las interacciones necesarias que tendrá el usuario con el sistema, ya que esta documentación permitirá identificar qué componentes expondrán funcionalidades sensibles que requerirán controles de seguridad específicos.

A partir de la clasificación de roles de usuario, diferenciando entre usuarios invitados o autenticados, se podrá definir niveles de acceso conforme al principio de mínimo privilegio, facilitando posteriormente la implementación de controles de autorización basados en roles y estableciendo así una línea base que oriente las decisiones arquitectónicas de las fases subsecuentes (Sanders & Yue, 2019). Simultáneamente, al ejecutar un mapeo del flujo de datos dentro del *frontend* se conocerá qué tipo de información circulará, se procesará o se almacenará temporalmente en el lado del cliente. La elaboración de un catálogo de datos sensibles específico del dominio de la aplicación permitirá determinar los requisitos definidos por legislaciones de protección de datos y derechos al consumidor que sean aplicables.

4.1.2. Diseño

Tras la clasificación de roles de usuario en la fase anterior, será posible definir una división de componentes y los niveles de acceso que tendrán los usuarios de acuerdo con sus requisitos de autenticación y autorización. Esta clasificación permitirá el diseño de una aplicación de página única donde se aplique una estrategia de controles de autorización basados en roles que especifique los mecanismos de visualización condicional de elementos de interfaz y sobre el sistema de enrutamiento. No obstante, es imperativo reconocer desde la perspectiva arquitectónica que estos controles en el *frontend* constituyen únicamente medidas de usabilidad y experiencia de usuario, no controles de seguridad efectivos, dado que el código JavaScript ejecutándose en el navegador

puede ser arbitrariamente manipulado, inspeccionado o modificado por actores maliciosos (CWE Community, 2025).

Como establece Microsoft (2025) sobre el paradigma de la confianza cero, dependiendo de los datos que maneje una aplicación, es necesario un modelo donde todas las decisiones de seguridad sean manejadas completamente por el *backend*, donde se opera en un entorno de ejecución controlado y no manipulable por el usuario final. Esta separación de responsabilidades, incluso en escenarios de compromiso total del *frontend*, significa que los activos críticos del sistema permanecerán protegidos por las verificaciones del servidor con autoridad, sin depender de la integridad del código cliente.

Por estas razones, durante el diseño de la aplicación, procesos como la autenticación o sistemas de pagos deben ser delegados al *backend*, a proveedores especializados o mediante protocolos estándar de la industria, evitando implementaciones personalizadas que frecuentemente contienen vulnerabilidades (Graham, 2022). Es importante definir una arquitectura que especifique que la información sensible transitará únicamente a través de canales seguros hacia el *backend* sin persistencia en el cliente, y que estos mecanismos mantendrán el estado de tal forma que se minimice la exposición a código JavaScript ejecutándose en el navegador (Tal, 2020).

4.1.3. Implementación

En correspondencia a los controles tomados a consideración previamente, esta tercera fase aborda la adopción de prácticas de codificación segura para resguardar la información que será manejada por la SPA. La implementación de sistemas de tipado fuerte proporciona una primera capa de seguridad al detectar en tiempo de compilación errores que podrían derivar en vulnerabilidades durante la ejecución de la aplicación, de esta manera se contará con un

mecanismo de verificación temprana que previene categorías enteras de fallos relacionados con el manejo inadecuado de datos antes de que el código alcance el entorno de producción (Equipo de Imagina, 2025).

Complementariamente, como establece Casero (2024), se promueve la validación de todas las entradas de usuario como manera preventiva, aprovechando las herramientas de protección incorporadas en los *frameworks* modernos. En cuanto al manejo de errores derivados de estas validaciones o de operaciones del sistema, se deben presentar mensajes genéricos y comprensibles al usuario final, evitando exponer detalles técnicos o rastros en consola (Villa, 2024).

La gestión de secretos requiere atención particularmente rigurosa, ya que el código fuente de las SPA es inherentemente accesible desde el navegador; por tanto, no debe incluirse información sensible directamente en el código, optando en su lugar por variables de entorno inyectadas durante el proceso de construcción o mediante servicios intermediarios que oculten estos detalles al cliente (The Editor, 2022). Como establece la propia OWASP Foundation (2025), al implementar una gestión de estado global mediante bibliotecas especializadas se debe limitar su uso exclusivamente a información no sensible, evitando almacenar datos que puedan ser inspeccionados trivialmente a través de las herramientas de desarrollo del navegador.

4.1.4. Verificación

Una vez implementados los controles de seguridad, es necesario verificar su eficiencia. Según Qadir, Waheed, Khanum y Jehan (2025), el análisis automatizado de seguridad comprende dos enfoques complementarios que examinan la aplicación desde perspectivas distintas. Herramientas de análisis estático evalúan el código fuente sin ejecutarlo, identificando patrones inseguros y violaciones de mejores prácticas mediante *linters* especializados integrados en el

entorno de desarrollo. Mientras que, por su parte, las herramientas para análisis dinámico evalúan la aplicación en ejecución, detectando vulnerabilidades como inyecciones de código, configuraciones inseguras de intercambio de recursos entre orígenes, encabezados de seguridad ausentes y exposiciones inadvertidas de información sensible.

Posteriormente, como indica Benites (2023), es necesario diseñar y ejecutar pruebas funcionales específicas que validen la efectividad real de los controles implementados, ya que este tipo de pruebas personalizadas superan las limitaciones de las herramientas automatizadas que no pueden interpretar la lógica de negocio particular de cada aplicación. Estos casos de prueba, documentados formal e idealmente automatizados mediante *frameworks* de pruebas *end-to-end*, pueden ejecutarse repetidamente en cada iteración del desarrollo para garantizar la consistencia de los controles de seguridad. Finalmente, el *testing* basado en roles y estados combina diferentes perfiles de usuario con diversos estados de la aplicación para identificar exposiciones accidentales de funcionalidades o datos que podrían no manifestarse en escenarios de uso convencional.

Cabe recalcar que los resultados de esta fase pueden requerir iteraciones hacia las fases de diseño o implementación para remediar las vulnerabilidades identificadas, lo cual se encuentra en línea con la naturaleza iterativa de las metodologías ágiles, donde cada ciclo incorpora desarrollo, validación y refinamiento.

4.1.5. Monitoreo

Esta última fase reconoce que la seguridad no constituye un estado final alcanzable sino un proceso dinámico y continuo que se extiende indefinidamente más allá del despliegue inicial. Tal como explica Drmunozcl (2025), el monitoreo constante de una aplicación requiere la integración de herramientas especializadas de seguimiento de errores y rendimiento que proporcionan

visibilidad en tiempo real sobre el comportamiento y la seguridad en producción. Por medio de estas plataformas es posible detectar y alertar sobre comportamientos anómalos y patrones indicativos de posibles incidentes de seguridad. Adicionalmente, se deben ejecutar revisiones periódicas de los registros generados para verificar la ausencia de exposiciones accidentales de información sensible, considerando que los sistemas de *logging* frecuentemente tienen controles de acceso más laxos que los sistemas productivos, son almacenados por períodos prolongados y son accesibles por múltiples equipos.

Un pilar importante dentro de la etapa de monitoreo es la gestión segura de dependencias y componentes de terceros, considerando que la cadena de suministro de software representa un vector de ataque cada vez más explotado, ejemplificado por incidentes de alto impacto como el compromiso de event-stream en 2018 o las vulnerabilidades en UAParser.js en 2021 (Arvanitis, Ntousakis, Ioannidis, & Vasilakis, 2022; Olsson, 2023). Por ello, es necesario mantener actualizadas las bibliotecas, llevar una eliminación periódica de dependencias no utilizadas mediante análisis, y optar, de preferencia, por bibliotecas oficiales y aquellas que sean reconocidas por la comunidad.

Finalmente, como indica la Universidad Nacional Abierta y a Distancia en su *Guía para la gestión y clasificación de incidentes de ciberseguridad* (2024), es necesario establecer un protocolo formal de respuesta ante un incidente de seguridad, que puede ser descubierto internamente o reportado externamente. Este protocolo debe tomar en cuenta un proceso de evaluación del impacto donde se determine el alcance de la vulnerabilidad, considerando tanto severidad técnica como impacto en el negocio. En el caso de equipos de desarrollo grandes, se debe especificar la coordinación con los equipos relevantes al caso y la comunicación transparente pero responsable con usuarios afectados cuando corresponda, cumpliendo obligaciones legales de

notificación establecidas en regulaciones de protección de datos y balanceando transparencia con responsabilidad al no divulgar detalles técnicos de explotación hasta que la mayoría de los usuarios hayan aplicado las correcciones necesarias.

4.2 Identificación de vulnerabilidades comunes dentro de un proyecto *frontend*

Antes de la aplicación de la metodología propuesta, se realizó un análisis de diagnóstico inicial sobre el repositorio *ISC.TimeReport.FE*, un proyecto *frontend* creado con el *framework* Angular para generar reportes de actividades dentro de la empresa Integrity Solutions Cía. Ltda. Esta revisión, con el objetivo de identificar las principales debilidades presentes en la capa *frontend,* permitió establecer una línea base que sirviera de referencia para la posterior verificación de mejoras.

4.2.1 Proceso de configuración de SonarQube

El análisis fue ejecutado mediante la herramienta SonarQube para la modalidad SAST, la cual efectuó una inspección del código fuente en busca de vulnerabilidades, errores de estilo y prácticas de desarrollo que puedan afectar la mantenibilidad o la seguridad del sistema. Los resultados obtenidos se clasificaron según el nivel de severidad definido por la propia plataforma.

Instalación y entorno

Se instaló SonarQube versión 9.9.4 LTS en un entorno local con sistema operativo Windows 11. El servidor fue iniciado mediante el script `StartSonar.bat`, ubicado en la ruta `C:\sonarqube\bin\windows-x86-64`.

Para el análisis de código se empleó SonarScanner CLI versión 7.1.0.4889, autenticado mediante un token de acceso generado desde la consola web del servidor.

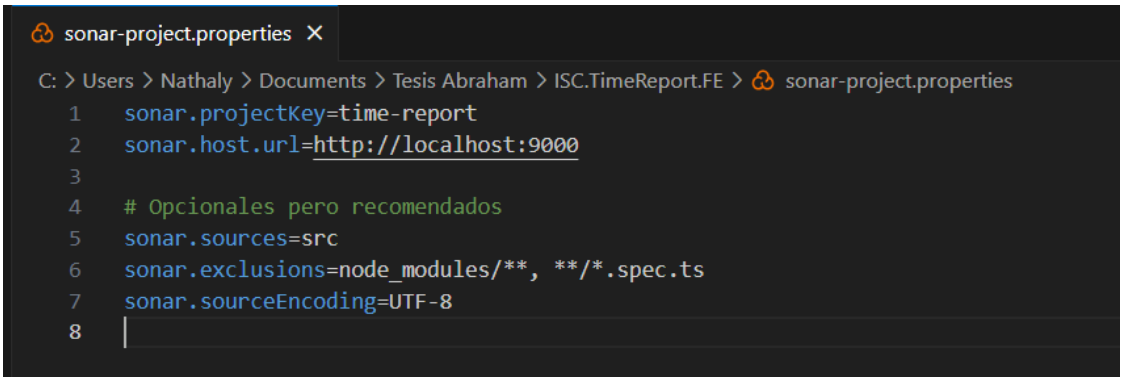
Para respaldar los resultados del análisis estático, se empleó el complemento SonarQube CNES Report (v. 4.5.2).

Integración con el proyecto

El análisis se aplicó al proyecto ISC.TimeReport.FE, ubicado en la ruta `C:\Users\Nathaly\Documents\TesisAbraham\ISC.TimeReport.FE`.

Debido a que el repositorio no se encontraba bajo un sistema de control de versiones (SCM), la ejecución del escaneo se realizó manualmente desde la raíz del proyecto. La configuración del análisis fue establecida en el archivo **`sonar-project.properties`**, cuya estructura se presenta en la Figura 3.

Figura 3
Configuración del archivo sonar-project.properties SonarQube



Definición de reglas y perfil de calidad

Durante la fase de configuración del análisis, se emplearon los perfiles de calidad por defecto de SonarQube, asociados a los lenguajes presentes en el proyecto ISC.TimeReport.FE, sin realizar personalizaciones adicionales. El propósito fue obtener una evaluación objetiva y comparativa del código fuente en su estado real, utilizando las métricas estándar de calidad que ofrece la herramienta.

Los perfiles de calidad (Quality Profiles) aplicados fueron los siguientes:

Sonar way [CSS] – archivo: `AZcuR-peDcbCilU1yVLm.json`

Sonar way [TypeScript] – archivo: `AZcuR_tZDcbCilU1yWqt.json`

Sonar way [HTML] – archivo: `AZcuR_SyDcbCilU1yWNQ.json`

Asimismo, el Quality Gate empleado fue el predeterminado denominado "Sonar way", definido en el archivo `Sonar way.xml`. Este conjunto de configuraciones permitió medir de forma uniforme la calidad del código en cuanto a fiabilidad, seguridad y mantenibilidad.

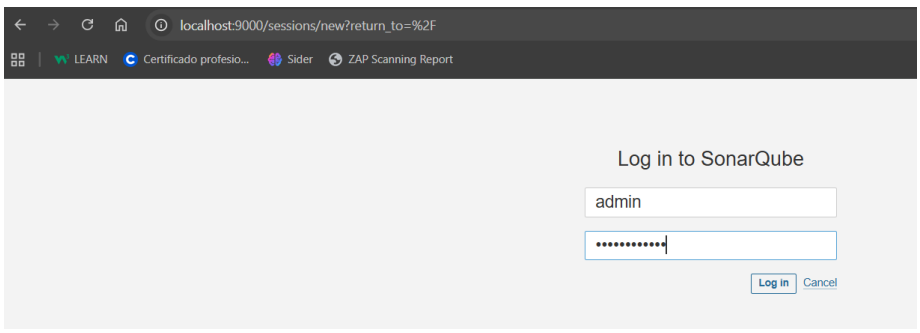
Análisis del código

El análisis del código fuente del proyecto ISC.TimeReport.FE **se** ejecutó de forma manual desde el entorno local, utilizando el escáner de SonarQube junto con los perfiles de calidad "Sonar way" de CSS, TypeScript y HTML. 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 4, que corresponde al acceso al servidor tras iniciar sesión como administrador.

Figura 4
Acceso al servidor web de SonarQube



Posteriormente, desde la raíz del proyecto, como se muestra en la Figura 5, se ejecutó los siguientes comandos.

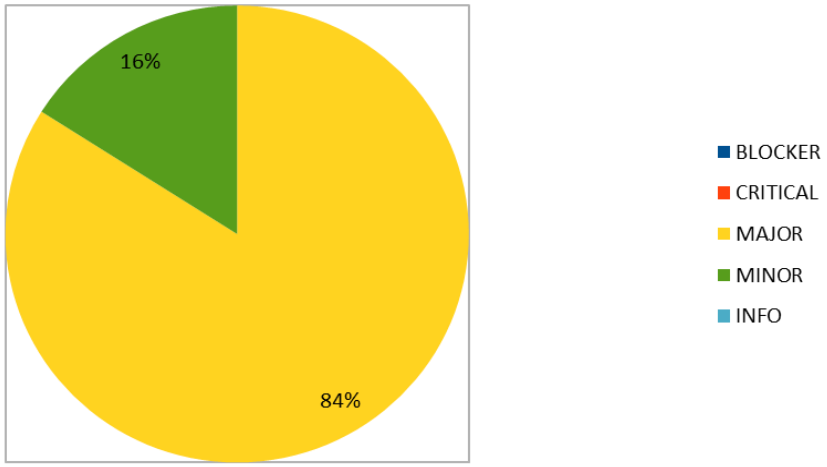
Figura 5
Ejecución del análisis en SonarScanner con autenticación mediante token

```
PS C:\Users\Nathaly\Documents\Tesis Abraham\ISC.TimeReport.FE> $env:SONAR_TOKEN = "
"
PS C:\Users\Nathaly\Documents\Tesis Abraham\ISC.TimeReport.FE> sonar-scanner "-Dsonar.login=$env:SONAR_TOKEN"
```

Al finalizar la ejecución, se obtuvieron los resultados del análisis estático**,** los cuales permitieron identificar las siguientes vulnerabilidades y el nivel de calidad del código fuente, de acuerdo con los parámetros del perfil aplicado.

La Figura 6 presenta la distribución porcentual de los hallazgos identificados. Como se observa, el 84% corresponde a incidencias de tipo mayor, relacionadas principalmente con configuraciones o prácticas que, si bien no comprometen directamente la seguridad, pueden derivar en vulnerabilidades si no se corrigen. El 16% restante corresponde a problemas menores, asociados a aspectos de legibilidad y cumplimiento de estándares de codificación. No se registraron hallazgos de criticidad alta o bloqueante en esta fase.

Figura 6
Número de problemas por gravedad en SonarQube



A continuación, en la Tabla 6, se resumen los principales hallazgos reportados, su descripción técnica y el tipo de severidad asignado por SonarQube.

Tabla 6
Resultados del análisis inicial de seguridad SonarQube

	Descripción	Tipo	Severidad	Número
Las declaraciones de fuente deben contener al menos una familia de fuente genérica	Si ninguno de los nombres de fuente definidos en una declaración de `font` o `font-family` está disponible en el navegador del usuario, el navegador mostrará el texto usando su fuente predeterminada. Se recomienda definir siempre una familia de fuente genérica para cada declaración de `font` o `font-family` con el fin de obtener una situación menos degradada que depender de la fuente predeterminada del navegador. Todos los navegadores deberían implementar una lista de fuentes genéricas que coincidan con estas familias: Serif, Sans-serif, cursive, fantasy, Monospace. Ejemplo de código no conforme `css a { font-family: Helvetica, Arial, Verdana, Tahoma; /* No conforme; no hay una fuente genérica en la lista */ }` Solución conforme `css a { font-family: Helvetica, Arial, Verdana, Tahoma, sans-serif; }` Ver: CSS Specification - Generic font families	BUG	MAYOR	36
Las etiquetas "" deben tener una descripción	Para que las tablas sean accesibles a usuarios con discapacidad visual, es importante que proporcionen una descripción de su contenido antes de acceder a los datos. La forma más simple, y también la recomendada por la versión 2 de las Web Content Accessibility Guidelines, es agregar un elemento `<caption>` dentro de la `<table>`. Otras técnicas aceptadas por esta regla son: - Agregar una descripción concisa mediante atributos `aria-label` o `aria-labelledby` en la	BUG	MENOR	16

	<table>. - Referenciar un elemento de descripción con un atributo aria-describedby en la <table>. - Incrustar la <table> dentro de un <figure> que también contenga un <figcaption>. - Agregar un atributo summary a la etiqueta <table>. **Nota:** este atributo está en desuso en HTML5. Ejemplo no conforme `html <table> <!-- No conforme --> ... </table>` Soluciones conformes Agregar un <caption>: `html <table> <caption>Resultados del Maratón de la Ciudad de Nueva York 2013</caption> ... </table>` Agregar un aria-describedby: `html <p id="mydesc">Resultados del Maratón de la Ciudad de Nueva York 2013</p> <table aria-describedby="mydesc"> ... </table>` Incrustar en <figure> con <figcaption>: `html <figure> <figcaption>Resultados del Maratón de la Ciudad de Nueva York 2013</figcaption> <table> ... </table> </figure>` Agregar un atributo summary (obsoleto): `html <table summary="Resultados del Maratón de la Ciudad de Nueva York 2013"> ... </table>` **Excepciones**: No se genera problema en <table> usada para maquetación (con role="presentation" o "none") o si tiene aria-hidden="true". Ver: WCAG2 1.3.1, WCAG2 H39			
No se deben comentar secciones de código	Los programadores no deben comentar código ya que esto infla los programas y reduce la legibilidad. El código no utilizado debe eliminarse y puede recuperarse desde el historial de control de versiones si es necesario.	CODE_ SMELL	MAYOR	7
No se deben duplicar los selectores	La duplicación de selectores puede indicar un error de copia-pega. La regla detecta: - duplicados dentro de una lista de selectores en un único conjunto de reglas - duplicados en diferentes conjuntos de reglas dentro de una misma hoja de estilo. Ejemplo no	CODE_ SMELL	MAYOR	19

	conforme `css .foo, .bar, .foo { ... } /* No conforme */ .class1 { ... } .class1 { ... } /* No conforme */` Solución conforme `css .foo, .bar { ... } .class1 { ... } .class2 { ... }`			
Los archivos CSS no deben estar vacíos	Esta regla genera un problema cuando un archivo CSS está vacío (es decir, contiene solo espacios).	CODE_ SMELL	MAYOR	22
TOTAL				100

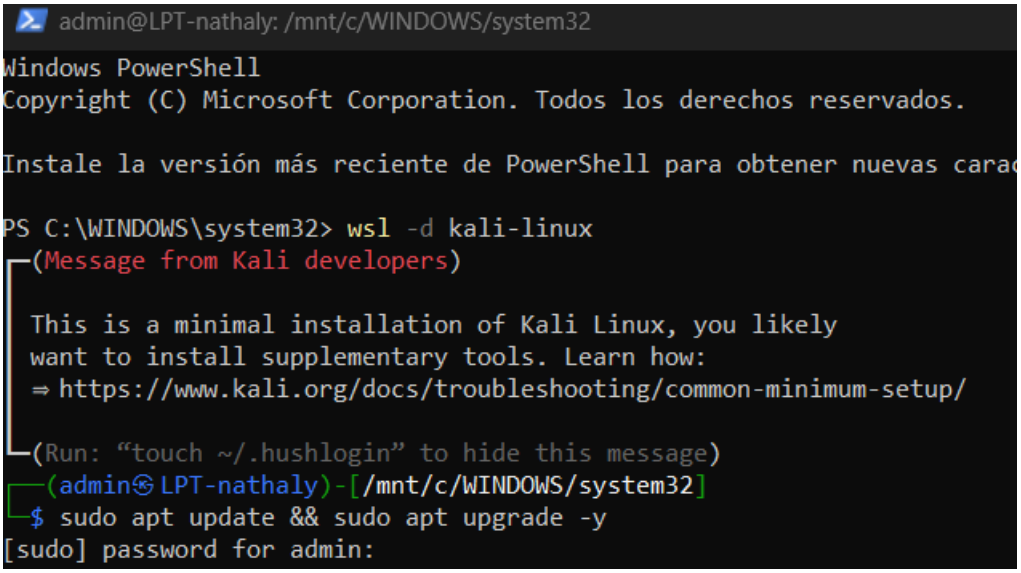
4.2.1 Proceso de configuración de OWASP ZAP

Para realizar el análisis de seguridad del módulo frontend del sistema *ISC.TimeReport.FE*, se empleó la herramienta OWASP ZAP bajo entorno Kali Linux en WSL (Windows Subsystem for Linux)**.** Este análisis se enfocó únicamente en las carpetas src y public, que representan los recursos públicos del proyecto. Antes de la ejecución del escaneo, se realizó la instalación y configuración de los componentes necesarios.

Instalación y configuración del entorno

Se actualizó el sistema y sus repositorios, como se observa en la Figura 7, con los siguientes comandos

Figura 7
Inicialización de Kali-Linux en WSL



OWASP ZAP requiere un entorno de ejecución Java. Para ello, se instaló *OpenJDK 11* con los siguientes comandos:

```
sudo apt install -y openjdk-11-jre-headless
```

```
java -version
```

Posteriormente, se instaló desde los repositorios oficiales de Kali con el comando `sudo apt install -y zaproxy`.

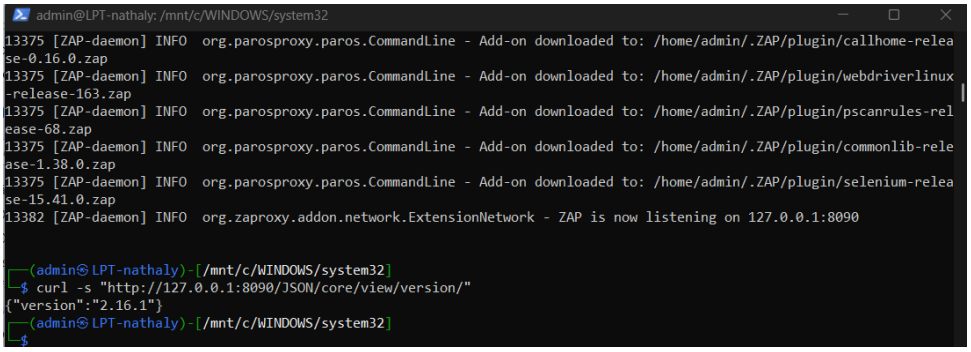
Configuración del entorno de análisis

Se inició OWASP ZAP en modo Daemon para habilitar la comunicación con el servidor API de ZAP con el siguiente comando:

```
/usr/share/zaproxy/zap.sh -daemon -host 127.0.0.1 -port 8090 -config
api.disablekey=true &
```

Posteriormente, se verificó la conexión como se muestra en la Figura 8.

Figura 8
Verificación de conexión ZAP

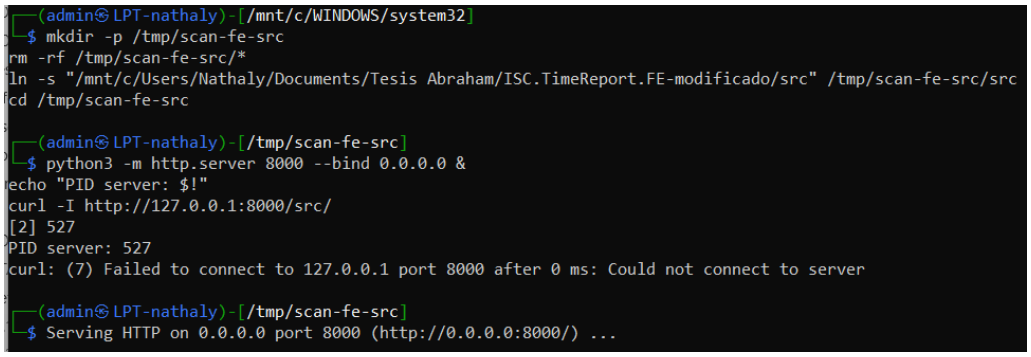


Preparación del entorno de análisis

Se creó un directorio temporal con acceso únicamente a las carpetas *src* **y** *public*, de forma que el análisis se enfoque solo en los recursos a evaluar. Y se utilizó el servidor HTTP integrado de Python para exponer las carpetas seleccionadas como se ilustra en la Figura 9.

Figura 9

Levantamiento del servidor local

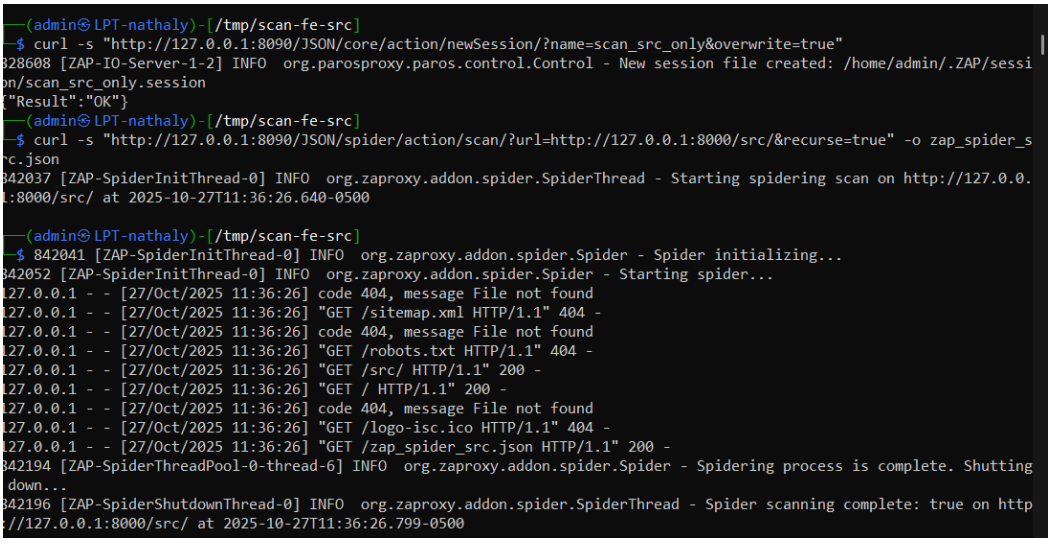


Ejecución del análisis

Para iniciar la evaluación, se creó una sesión en OWASP ZAP llamada scan_src_only con el fin de registrar exclusivamente los resultados del análisis de la carpeta src. Esta acción, que se realizó mediante un comando cURL, sobrescribe cualquier configuración previa. A continuación, se ejecutó un Spider Scan (rastreo de araña) para explorar el contenido público del proyecto e identificar los recursos accesibles desde el cliente, demostrado en la Figura 10.

Figura 10

Creación de la sesión 'scan_src_only' y ejecución del Spider Scan en OWASP ZAP



Una vez finalizada la fase de descubrimiento, se ejecutó el escaneo activo sobre las rutas detectadas donde se identificó un total de 10 alertas de seguridad distribuidas en diferentes niveles de severidad. Como se muestra en la Figura 11, no se registraron vulnerabilidades de nivel alto, mientras que se reportaron siete de nivel medio, dos de nivel bajo y una de carácter informativo. Esta distribución evidencia que, si bien el proyecto no presenta fallos críticos que comprometan directamente su integridad, sí existen configuraciones inseguras y cabeceras ausentes que podrían facilitar ataques en entornos de producción.

Figura 11

Alertas identificadas en el análisis con OWASP ZAP

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	7
Low	2
Informational	1
False Positives:	0

En la Figura 12 se detallan las vulnerabilidades detectadas por OWASP ZAP en el módulo *ISC.TimeReport.FE*. Las alertas de nivel medio corresponden principalmente a la ausencia de cabeceras de seguridad críticas como CSP, *Anti-clickjacking* y la exposición de errores de aplicación, factores que pueden facilitar ataques de inyección o manipulación del contenido mostrado al usuario. Las vulnerabilidades de nivel bajo e informativo se relacionan con la divulgación de información del servidor, cabeceras HTTP no configuradas y comentarios visibles en el código, los cuales, aunque no representan un riesgo inmediato, reflejan configuraciones que deben corregirse para fortalecer la seguridad del *frontend*.

Figura 12

Detalle de vulnerabilidades detectadas en el módulo ISC.TimeReport.FE mediante OWASP ZAP

Alerts

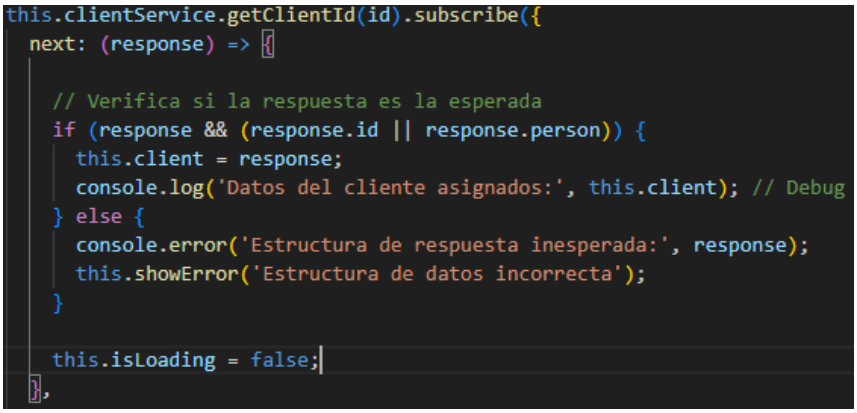
Name	Risk Level	Number of Instances
CSP: Meta Policy Invalid Directive	Medium	1
CSP: Wildcard Directive	Medium	1
CSP: script-src unsafe-eval	Medium	1
CSP: script-src unsafe-inline	Medium	1
CSP: style-src unsafe-inline	Medium	1
Content Security Policy (CSP) Header Not Set	Medium	4
Missing Anti-clickjacking Header	Medium	2
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	6
X-Content-Type-Options Header Missing	Low	3
User Agent Fuzzer	Informational	12

4.2. Aplicación de la metodología

En respuesta a los hallazgos de la herramienta SonarQube, se implementaron correcciones en el código fuente del *frontend* orientadas a subsanar deficiencias en la calidad y el cumplimiento de buenas prácticas de programación, según lo establecido en la metodología propuesta. Este proceso, ejemplificado en la Figura 13, incluyó la optimización de la base del código mediante la eliminación de secciones comentadas, la adición de fuentes de *fallback* y registros en la consola del navegador que exponían información sensible del usuario y *payloads* de la aplicación, con el fin de prevenir fugas de información.

Figura 13

Fragmento de código con exposición de datos sensibles en el frontend



```
this.clientService.getClientId(id).subscribe({
  next: (response) => {

    // Verifica si la respuesta es la esperada
    if (response && (response.id || response.person)) {
      this.client = response;
      console.log('Datos del cliente asignados:', this.client); // Debug
    } else {
      console.error('Estructura de respuesta inesperada:', response);
      this.showError('Estructura de datos incorrecta');
    }

    this.isLoading = false;
  },
```

Por otro lado, el análisis efectuado con ZAP reveló alertas concernientes a la configuración de las cabeceras de seguridad. Estas fueron mitigadas mediante el ajuste de la configuración del servidor Nginx, implementando directivas de CSP y cabeceras anti-*clickjacking* a través de X-Frame y X-Content, como evidenciado en la Figura 14.

Figura 14

Configuración del servidor de la aplicación en Angular

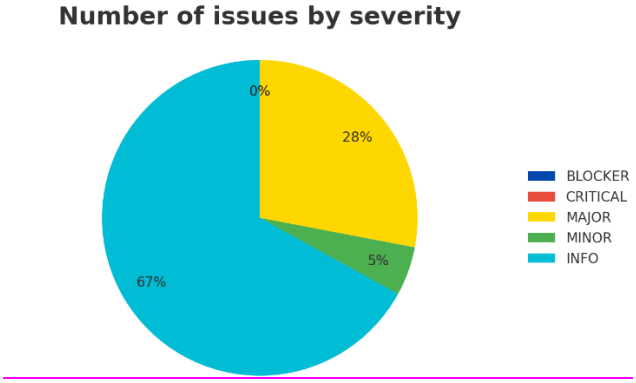
```
nginx.conf
server {
    listen 80;
    server_name localhost;
    server_tokens off;
    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
        # Encabezados de seguridad
        add_header X-Frame-Options "SAMEORIGIN" always;
        add_header X-Content-Type-Options "nosniff" always;
        add_header Referrer-Policy "strict-origin-when-cross-origin" always;
        add_header Content-Security-Policy "default-src 'self'; style-src 'self'
        'unsafe-inline' https://fonts.googleapis.com; font-src 'self' https://fonts.gstatic.com;
        script-src 'self'; img-src 'self' data:; frame-ancestors 'none';" always;
    }
}
```

4.3. Evaluación posterior

Tras la implementación, se volvieron a ejecutar las pruebas con las mismas herramientas, donde se evidenció una disminución significativa en el número de vulnerabilidades y en su nivel de severidad. En la Figura 15 se presentan los resultados con la herramienta SonarQube, donde, en comparación con los resultados presentados en la Figura 6, se observó una reducción del 84% a un 28% de alertas con severidad mayor, y del 16% a un 5% en alertas de severidad menor.

Figura 15

Resultados de SonarQube tras la implementación de la metodología



Por otro lado, los resultados del análisis por ZAP, presentados en la Figura 16, demuestran una reducción de diez tipos de alertas identificadas en la Figura 11, a dos tipos de alertas, representando una disminución del 80% después de la implementación de la metodología en el código *frontend*.

Figura 16

Resultados de ZAP tras la implementación de la metodología

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	1
Low	0
Informational	1
False Positives:	0

En base a la propuesta metodológica, los cambios implementados integraron el concepto de *security by design*, utilizando las verificaciones V1: Codificación y sanitización como una base para mejorar la calidad del código. Se dieron cambios para incrementar la accesibilidad y eliminación de código innecesario tomando como referencia las verificaciones V5: Manejo de archivos, mientras que la mitigación de vulnerabilidades de configuración detectadas por ZAP (relacionadas con CSP y cabeceras *anti-clickjacking*) abordó directamente las verificaciones V14: Protección de datos y el control proactivo C4: Encargarse de la seguridad desde el inicio. Finalmente, a partir de las verificaciones V10: OAuth y OIDC y el control proactivo C9: Implementar registro y monitoreo de seguridad, se cumplió con el manejo de errores y *logging* al eliminar registros en consola que exponían información sensible y *payloads*, para la prevención de fugas de datos.

CONCLUSIONES

Las vulnerabilidades identificadas en el proyecto ISC.TimeReport.FE comprenden la exposición de datos sensibles en la consola del navegador, configuraciones inseguras del servidor *frontend* derivadas de la ausencia de cabeceras CSP y controles *anti-clickjacking*, validación inadecuada de entradas, así como una gestión deficiente del código evidenciada por la presencia de segmentos comentados, dependencias carentes de fuentes genéricas y ausencia de descripciones en elementos de tabla. Estas vulnerabilidades se corresponden directamente con las categorías A05:2021 – Configuración de Seguridad Incorrecta y A09:2021 – Fallas en el Registro y Monitoreo del OWASP Top 10.

Mediante la presente investigación fue posible establecer una correspondencia entre las vulnerabilidades detectadas y los controles proactivos y verificaciones del estándar OWASP. Las vulnerabilidades de exposición de datos se asocian con la verificación V10: Protección de Datos Sensibles y el Control Proactivo C9: Registro y Monitoreo de Seguridad. Las configuraciones inseguras del servidor se vinculan con la V14: Configuración Segura y C8: Características de Seguridad del Navegador. Los problemas de accesibilidad y validación corresponden al V5: Validación de Entradas y C3: Validar Todas las Entradas. Finalmente, las vulnerabilidades relacionadas con la gestión de sesiones y autenticación hacen referencia a la verificación V3: Gestión de Sesiones y el C7: Implementar Identidad Digital Segura.

La metodología propuesta demostró ser efectiva para integrar la seguridad en el ciclo de desarrollo *frontend*, evidenciándose en su aplicación sobre el sistema ISC.TimeReport.FE mediante los siguientes resultados cuantitativos: una reducción del 80% en las alertas de seguridad detectadas por ZAP, una disminución de problemas de severidad mayor del 84% al 28% según el análisis de SonarQube, la eliminación de vulnerabilidades relacionadas con la exposición de datos sensibles en consola, y la implementación exitosa de cabeceras de seguridad dentro del sistema.

Las herramientas SonarQube y ZAP demostraron una alta complementariedad al proporcionar una visión amplia que abarca tanto el análisis estático del código fuente como la evaluación dinámica en tiempo de ejecución, lo cual resulta necesario para arquitecturas SPA.

La metodología propuesta es compatible con entornos de desarrollo ágil, al estructurarse en fases que pueden integrarse naturalmente en los ciclos iterativos característicos de metodologías como Scrum o Kanban, sin comprometer la velocidad de entrega.

RECOMENDACIONES

Incorporar la metodología propuesta como parte de los procesos institucionales de desarrollo de software, garantizando su adopción temprana en proyectos frontend.

Mantener la ejecución periódica de análisis SAST y DAST, utilizando herramientas actualizadas para detectar nuevas vulnerabilidades derivadas de dependencias o configuraciones.

Capacitar de forma continua a los desarrolladores en estándares OWASP ASVS y buenas prácticas de seguridad, con énfasis en el principio de *security by design*.

Extender la metodología a otras capas del sistema (*backend* y servicios) para lograr un enfoque integral de seguridad en toda la arquitectura de la aplicación.

A partir de esta investigación se pudo reconocer la necesidad de implementar varias medidas de seguridad a nivel de código entre las cuales destaca evitar el almacenamiento de tokens de acceso en localStorage o sessionStorage debido a su vulnerabilidad ante ataques XSS, optando por cookies httpOnly y secure o delegar esta responsabilidad al *backend*. Se debe configurar CSP con directivas estrictas, habilitar cabeceras de seguridad (X-Frame-Options, X-Content-Type-Options, Strict-Transport-Security), implementar tokens CSRF en formularios, y establecer SameSite=Strict en cookies de sesión. La validación de entradas debe realizarse tanto en cliente como en servidor, aplicando sanitización con bibliotecas especializadas como DOMPurify, y evitar la exposición de información sensible en mensajes de consola. Adicionalmente, se recomienda ejecutar auditorías de dependencias periódicamente mediante npm audit, y mantener revisiones de código con pruebas de penetración.

Finalmente, se propone extender la validación de la metodología propuesta en este trabajo mediante investigaciones futuras que evalúen su efectividad en proyectos *frontend* desarrollados

con diversos *frameworks* y librerías, como React, Vue.js, Svelte o Next.js, así como en arquitecturas emergentes, tales como *microfrontends* y aplicaciones web progresivas. A través de esta validación cruzada se podrán identificar las adaptaciones necesarias según cada tecnología, lo que permitirá refinar la metodología y demostrar su utilidad práctica en diferentes contextos de desarrollo *frontend*.

REFERENCIAS BIBLIOGRÁFICAS

Akiwatkar, R. (2025, 28 enero). Agile Adoption Statistics: How is Software Development changing? https://www.simform.com/blog/state-of-agile-adoption/

Amazon Web Services. (2025). *Diferencias entre frontend y backend*. AWS. https://aws.amazon.com/es/compare/the-difference-between-frontend-and-backend/

Arvanitis, I., Ntousakis, G., Ioannidis, S., & Vasilakis, N. (2022). A systematic analysis of the event-stream incident. EuroSec '22: Proceedings Of The 15th European Workshop On Systems Security, 22-28. https://doi.org/10.1145/3517208.3523753

Astra Security. (2023, September 21). A Guide to DAST for Single Page Applications (SPAs). GetAstra. https://www.getastra.com/blog/dast/dast-for-single-page-applications/

Benites, S. F. (2023). Universidad Católica Sedes Sapientiae Facultad de Ingeniería Implementación de un Framework de Automatización para Mejorar el Proceso de Pruebas Funcionales de una OSE, Lima, 2023 (Tesis de grado). Universidad Católica Sedes Sapientiae. https://hdl.handle.net/20.500.14095/1840

Binboga, B., & Altin Gumussoy, C. (2024). Factors affecting agile software project success. IEEE Access, 12, 95613-95633. https://doi.org/10.1109/ACCESS.2024.3384410

Brown, L., et al. (2022). Challenges and best practices for secure software development in financial institutions. *International Journal of Banking and Finance Security, 8*(1), 78–95.

Casero, A. (2024, 15 marzo). Técnicas de validación de entradas en programación. https://keepcoding.io/blog/validacion-de-entradas-en-programacion/

Cohen, A. (2025). Browser Security Posture Analysis: A Client-Side Security Assessment Framework. *arXiv.* https://arxiv.org/abs/2505.08050

Crêspo Boaventura, J., Peña Herrera, E., Verdecia Vicet, P., & Fustiel Alvarez, Y. (2016). Elección entre una metodología ágil y tradicional basado en técnicas de soft computing. *Revista Cubana de Ciencias Informáticas, 10*(Especial UCIENCIA), 145-158. http://www.scielo.sld.cu/scielo.php?script=sci_arttext&pid=S2227-18992016000500011

CWE Community. (2025, 9 septiembre). CWE-602: Client-Side Enforcement of Server-Side Security (4.18). https://cwe.mitre.org/data/definitions/602.html

Drmunozcl. (2025, 22 agosto). Seguridad como proceso continuo, no como destino. https://www.infoproteccion.com/desarrollo-software-seguro/seguridad-como-proceso-continuo-no-como-destino/

Ekpobimi, S., Kandekere, F., & Fasanmade, O. (2024). Front-end development and cybersecurity: A conceptual approach to building secure web applications. *ResearchGate.* https://www.researchgate.net/publication/383833448_Front-end_development_and_cybersecurity_A_conceptual_approach_to_building_secure_web_applications

Equipo de Imagina. (2025, 15 octubre). ¿Qué es TypeScript y Por qué Usarlo? https://imaginaformacion.com/tutoriales/que-es-typescript

Fleetham, C. (2025, April 25). Single page applications: Why do you need to scan them? *Intruder.* https://www.intruder.io/blog/single-page-applications-why-do-you-need-to-scan-them

Graham, C. (2022, 14 marzo). Computer scientist identifies JavaScript vulnerability in thousands of websites. https://hub.jhu.edu/2022/03/14/computer-scientist-identifies-javascript-vulnerability/

Hevner, N., March, N., Park, N., & Ram, N. (2004). Design Science in Information Systems Research. *MIS Quarterly, 28*(1), 75. https://doi.org/10.2307/25148625

Kudriavtseva, A., & Gadyatskaya, O. (2022). Secure Software Development Methodologies: A Multivocal Literature Review. *arXiv.* https://arxiv.org/abs/2211.16987v2

Lawson, K. (2025, August 28). What Are Single Page Applications and Why Do People Like Them So Much? *Bloomreach.* https://www.bloomreach.com/en/blog/what-is-a-single-page-application

Liu, F., Fang, Z., Shao, S., Xu, Z., Zhou, M., & Yu, J. (2023). A quantitative security evaluation and analysis model for web applications based on OWASP Application Security Verification Standard. *Computers & Security, 136,* 103579. https://doi.org/10.1016/j.cose.2023.103579

McGraw, G. (2006). *Software security: Building security in.* Addison-Wesley Professional.

MDN contributors. (2025, July 11). SPA (Single-page application) – Glossary. *Mozilla Developer Network.* https://developer.mozilla.org/en-US/docs/Glossary/SPA

MDN Web Docs. (2025). *Introduction to client-side frameworks*. Mozilla Developer Network.https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/Introduction

Microsoft. (2025, 5 marzo). ¿Qué es la confianza cero? https://learn.microsoft.com/es-es/security/zero-trust/zero-trust-overview

National Institute of Standards and Technology. (2020). *Security and privacy controls for information systems and organizations (SP 800-53 Rev. 5).* U.S. Department of Commerce. https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final

Olsson, S. (2023, 21 diciembre). UAParser.js npm Package Supply Chain Attack: Impact and Response. https://www.truesec.com/hub/blog/uaparser-js-npm-package-supply-chain-attack-impact-and-response

OpenText. (2025). What is DAST (Dynamic Application Security Testing)? *OpenText.* https://www.opentext.com/what-is/dast

OpenText. (2025). What is SAST (Static Application Security Testing)? *OpenText.* https://www.opentext.com/what-is/sast

OWASP. (2023). *OWASP Application Security Verification Standard (ASVS).* OWASP Foundation. https://owasp.org/www-project-application-security-verification-standard/

OWASP Foundation. (2021). *OWASP Top 10:2021 – The Ten Most Critical Web Application Security Risks.* https://owasp.org/www-project-top-ten/

OWASP Foundation. (2023a). *OWASP Top 10 Proactive Controls.* https://top10proactive.owasp.org/the-top-10/#security-controls

OWASP Foundation. (2023b). *OWASP Proactive Controls Cheat Sheet Series.* https://cheatsheetseries.owasp.org/IndexProactiveControls.html

OWASP Foundation. (2024). *ZAP – Getting started.* OWASP ZAP. https://www.zaproxy.org/getting-started/

OWASP Foundation. (2025). OWASP Top 10 Client-Side Security Risks. https://owasp.org/www-project-top-10-client-side-security-risks

Rindell, K., Mäntylä, M. V., Hyrynsalmi, S., & Leppänen, V. (2021). Security in agile software development: A practitioner survey. *Journal of Systems and Software, 181*, 111050. https://doi.org/10.1016/j.jss.2021.111050

Sethi, R. (2024, 13 agosto). What is Security by Design / Secure by Design? https://www.securitycompass.com/blog/what-is-secure-by-design/

Sion, L., Yskout, K., Van Landuyt, D., & Joosen, W. (2018). Risk-based design security analysis. *Proceedings of the 1st International Workshop on Security Awareness from Design to Deployment (SEAD'18),* 18–25. ACM. https://doi.org/10.1145/3194707.3194710

SonarSource. (2023). *Static code analysis: Developer's guide.* SonarSource. https://www.sonarsource.com/resources/library/static-code-analysis/

Tal, L. (2020, 3 junio). JavaScript security. https://snyk.io/articles/javascript-security/

Tatar, Ü., & Karabacak, B. (2012). An hierarchical asset valuation method for information security risk analysis. *Proceedings of the International Conference on Information Society (i-Society 2012),* 47–52. IEEE.

Universidad Nacional Abierta y a Distancia. (2024). Guía Para la Gestión y Clasificación de Incidentes de Ciberseguridad.

https://selloeditorial.unad.edu.co/images/Documentos/ciberseguridad/Gu%C3%ADa_para_la_Gesti%C3%B3n_y_Clasificaci%C3%B3n_de_un_Incidentes_de_Ciberseguridad.pdf

Villa, A. A. (2024, 26 noviembre). Herramientas esenciales para controlar errores en el código. https://profile.es/blog/herramientas-esenciales-para-controlar-errores-en-el-codigo/

ANEXOS

ACTA DE PROPUESTA DE METODOLOGÍA

Lugar: Guayaquil – Ecuador **Fecha:** 15 de octubre de 2025

Asunto:

Propuesta de aplicación de metodología de desarrollo seguro.

Antecedentes:

En el marco del desarrollo del proyecto **ISC.TimeReport.FE**, liderado por la **Ing. Kerly Jiménez**, se ha identificado la oportunidad de aplicar prácticas avanzadas de **seguridad en el ciclo de vida del software (Secure SDLC)** con el propósito de fortalecer la calidad, trazabilidad y mitigación de vulnerabilidades en los componentes **Frontend**.

El suscrito, Abraham Jacobo Vallejos Males, en mi calidad de estudiante, me encuentro desarrollando la tesis titulada:

"Implementación de una Metodología para la Mitigación de Vulnerabilidades Frontend Basada en OWASP ASVS y Controles Proactivos"

La cual tiene como objetivo establecer un marco metodológico práctico y medible que permita reducir los riesgos de seguridad en aplicaciones web, mediante la integración de herramientas automatizadas de análisis estático y dinámico, como **SonarQube** y **OWASP ZAP**, junto con lineamientos de las normas **OWASP ASVS** y **Controles Proactivos (OWASP Proactive Controls)**.

Propuesta:

Se solicita a la empresa **Integrity Solutions Cía. Ltda.**, en su calidad de organización responsable del proyecto, autorizar la **aplicación de la metodología de investigación propuesta** dentro del entorno de desarrollo del sistema **ISC.TimeReport.FE**, específicamente en el **Frontend (Angular)**, con fines de validación académica y de mejora continua del proceso de desarrollo seguro de software.

La metodología incluirá las siguientes fases:

1. **Diagnóstico inicial de vulnerabilidades Frontend.**
   Uso de herramientas OWASP ZAP y SonarQube para evaluación base.
2. **Definición de controles basados en OWASP ASVS y Controles Proactivos.**
3. **Implementación de medidas correctivas y ajustes en el código fuente.**
4. **Evaluación de la efectividad de la metodología y generación de evidencias.**
5. **Documentación de resultados.**

Compromisos:

- Toda la información analizada y los resultados obtenidos serán manejados bajo confidencialidad, respetando las políticas de seguridad de **Integrity Solutions**.

- No se alterará el entorno productivo ni se divulgarán datos sensibles del proyecto.
- Los resultados se usarán únicamente con fines académicos y de mejora interna de la práctica de desarrollo seguro.

Solicitud formal:

Por lo expuesto, se solicita la **autorización y apertura institucional** para la implementación de la metodología antes descrita dentro del marco del proyecto **ISC.TimeReport.FE**, con el acompañamiento y supervisión de la **Ing. Kerly Jiménez** como líder de desarrollo.

Firma del Solicitante:

_____

Ing. Abraham Jacobo Vallejos Males
Estudiante de la UNIVERSIDAD TÉCNICA DEL NORTE DEL PAÍS

Firma de Aprobación:

_____

Ing. Kerly Jiménez
Líder de Desarrollo
Integrity Solutions Cía. Ltda.

Firma de Aprobación:

_____

Tanya Elizabeth Cumbicos
Gerente de Proyectos
Integrity Solutions Cía. Ltda.