Mecanismos de Seguridad para el Diseño de Desarrollo Web.

Buenas prácticas de Seguridad.

Nombre Estudiante
Sergio Molina González

Área de trabajo final
Seguridad Informática

Nombre Tutor/a de TF
Jorge Miguel Moneo

Profesor/a responsable de la asignatura
Jorge Miguel Moneo

Fecha Entrega
28/06/2024

B) GNU Free Documentation Licúense (GNU FDL)

C) Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Seguridad para el Diseño de Desarrollo Web.
Nombre del autor:	Sergio Molina González
Nombre del director/a:	Jorge Miguel Moneo
Nombre del PRA:	Jorge Miguel Moneo
Fecha de entrega (mm/aaaa):	28/06/2024
Titulación o programa:	Grado de Ingeniería Informática
Área del Trabajo Final:	Seguridad Informática
Idioma del trabajo:	Castellano
Palabras clave	Aplicaciones web, seguridad y vulnerabilidades.

Resumen del Trabajo

El crecimiento acelerado de la utilización de aplicaciones web ha generado un incremento significativo en los riesgos de seguridad, afectando la disponibilidad, integridad y confidencialidad de la información. A pesar de los avances tecnológicos, la complejidad, necesidad y conectividad del software incrementan la dificultad para mantener la seguridad en las aplicaciones web.

La velocidad en el desarrollo de aplicaciones web potencia el riesgo de no identificar vulnerabilidades a tiempo, lo que requiere de una gestión eficiente de incidentes de seguridad.

Es esencial la implementación de estrategias de seguridad desde el inicio del desarrollo de aplicaciones, incluyendo la detección, prevención y corrección de vulnerabilidades. Los especialistas en seguridad deben conocer metodologías reconocidas a nivel mundial para crear un enfoque adaptable y efectivo en la gestión de problemas de seguridad en aplicaciones web.

Existen numerosos recursos que ofrecen información detallada sobre seguridad en entornos web, incluyendo legislación relevante y estándares de seguridad.

Este proyecto busca proporcionar un enfoque equilibrado, resumiendo los aspectos críticos encontrados en la literatura existente, analizando la legislación y los estándares de seguridad, detallando las vulnerabilidades posibles de una manera comprensible y aplicando este conocimiento en una aplicación de entrenamiento para pruebas de penetración, ofreciendo así práctica a desarrolladores en la identificación y corrección de estas vulnerabilidades.

Abstract

The accelerated growth in the use of web applications has generated a significant increase in security risks, affecting the availability, integrity and confidentiality of information. Despite technological advances, the complexity, necessity, and connectivity of software increase the difficulty of maintaining security in web applications.

The speed in the development of web applications increases the risk of not identifying vulnerabilities in time, which requires efficient management of security incidents.

It is essential to implement security strategies from the beginning of application development, including the detection, prevention, and correction of vulnerabilities. Security specialists must be familiar with globally recognized methodologies to create an adaptable and effective approach to managing web application security issues.

There are numerous resources that offer detailed information on web security, including relevant legislation and security standards.

This project seeks to provide a balanced approach, summarizing the critical aspects found in existing literature, analysing security legislation and standards, detailing possible vulnerabilities in an understandable way, and applying this knowledge in a penetration testing training application, offering thus practicing developers in identifying and correcting these vulnerabilities.

Índice

Listas de Figuras

1. Introducción.

En el dinámico mundo de la informática, la **Ciberseguridad** se erige como un bastión crítico, evolucionando al ritmo frenético de las innovaciones tecnológicas. Cada nuevo avance desvela complejidades y retos de seguridad inéditos, **especialmente en el ámbito de las aplicaciones web**, que se han expandido vertiginosamente. En este contexto, los datos, su esencia y circulación, han ganado un valor inestimable, atrayendo la mirada de ciberdelincuentes. La magnificación de usuarios amplifica este riesgo, demandando un escrutinio y protección rigurosos. La adaptabilidad es clave; entender y mitigar amenazas emergentes es imperativo en este panorama tecnológico en constante metamorfosis.

"La adopción masiva de tecnologías de la información y telecomunicaciones está transformando nuestra sociedad y todos los sectores de nuestra economía. Está cambiando como nos relacionamos como sociedad, como interactuamos con las Administraciones Publicas y como las empresas desarrollan y entregan sus productos." (MARTÍNEZ, J.G., 2018).

El salto tecnológico en las corporaciones es innegable, donde la informática es ya piedra angular en la optimización empresarial. Las aplicaciones web, en particular, reformulan la interacción laboral, ofreciendo accesibilidad y eficiencia sin precedentes. **El proyecto descrito se sumerge en esta realidad, proponiendo una herramienta analítica para emprendedores, desmenuzando el potencial de nuevas iniciativas para profundizar en la seguridad del desarrollo de aplicaciones web.** A través de un enfoque dual, administrador y cliente, el estudio promete una evaluación detallada y estratégica, brindando luz en el intrincado camino del emprendimiento. Este documento, aspira a ser un faro de conocimiento y guía práctica en el universo del desarrollo web y la mitigación de los fallos en seguridad.

La protección de la información y los sistemas que la respaldan, incluyendo las redes y la infraestructura de la tecnología de informática (TI), es esencial para las organizaciones, necesitando una defensa efectiva contra amenazas que comprometan aspectos críticos como la **disponibilidad, integridad y confidencialidad de la información**, todos vitales para alcanzar las metas de nuestro proyecto.

Los datos se procesan, intercambian y almacenan mediante infraestructuras tecnológicas, estando sujetos a amenazas de seguridad multifacéticas, internas y externas. Además de riesgos físicos como accesos indebidos, es crucial prestar atención a riesgos digitales, incluyendo **malware y ataques de denegación de servicio**, que son cada vez más frecuentes y complejos.

Minimizar estos riesgos y el impacto de potenciales amenazas es factible sin grandes inversiones o equipos extensos, pero requiere una gestión de riesgos meticulosa y la implementación de controles de seguridad adecuados en el desarrollo de cualquier aplicación web.

Mejorar los mecanismos de seguridad se logra instaurando un conjunto de políticas, procesos, procedimientos y sistemas organizativos y técnicos, que deben ser constantemente supervisados y actualizados para cumplir con los requisitos tanto en el

desarrollo como en el mantenimiento e implantar las técnicas de testing necesarias para la correcta experiencia del usuario con la interfaz.

Es un error considerar la ciberseguridad como una responsabilidad exclusiva de especialistas en TI; en realidad, implica un compromiso de todos los implicados, partiendo de los desarrolladores en las buenas praxis para evitar cualquier brecha que pueda significar una vulneración de los sistemas en el despliegue de cualquier aplicación web.

Implementar medidas de seguridad uniformes sin analizar la relevancia y particularidades de cada área puede llevar a una protección inadecuada o excesiva. Una gestión eficaz involucra la integración de recursos humanos y técnicos, apoyada por medidas administrativas, estableciendo controles efectivos alineados con los objetivos. Este proyecto intenta brindar unas herramientas comprehensivas, facilitando una visión integral sobre la seguridad en el desarrollo y permitiendo tomar decisiones informadas sobre la estrategia de seguridad adoptada.

1.1. Contexto y justificación del Trabajo.

En el contexto actual donde la tecnología digital es omnipresente, las aplicaciones web se han convertido en herramientas fundamentales para el desarrollo empresarial, catalizando nuevas ideas de negocio. Este proyecto se justifica al reconocer la importancia crítica de integrar la ciberseguridad de manera intrínseca en el desarrollo web, abordando la complejidad creciente en ambos dominios, Frontend y Backend.

"Las amenazas a la seguridad de la información han existido siempre, pero ha sido en los últimos años cuando los riesgos asociados a las mismas han sufrido un crecimiento exponencial." (MARTÍNEZ, J.G., 2018).

Los aspectos más significativos lo podemos justificar en:

Necesidad de Seguridad en Aplicaciones Web: La integración creciente de las aplicaciones web en operaciones críticas de cualquier organización subraya la fundamental necesidad de asegurar su protección contra amenazas en constante evolución. La repercusión de las brechas de seguridad no solo incurre en pérdidas económicas significativas, sino que también deteriora la confianza y la imagen de las organizaciones afectadas.

"El uso de desarrollo web con una estrategia Headless, es decir una web sin back-end puede llevarnos a realizar malas prácticas que pueden poner en compromiso la seguridad del sitio web.

Pasar por alto la seguridad front-end puede dejar tus aplicaciones web vulnerables a una amplia gama de amenazas, incluidos ataques de cross- site scripting (XSS), ataques de falsificación de solicitudes entre sitios (CSRF) y otras vulnerabilidades de seguridad. Es por ello por lo que me gustaría que me acompañes a explorará las mejores prácticas esenciales de seguridad front-end para ayudarte a proteger tus aplicaciones web contra scripts maliciosos y posibles riesgos de seguridad." (VERGARA, S., 2023).

Evolución y Escalada de Amenazas de Seguridad: El panorama de las amenazas cibernéticas ha mostrado una evolución hacia técnicas más sofisticadas y ataques dirigidos, lo que exige una evolución paralela en las estrategias de seguridad. Esta dinámica complejiza la gestión de la seguridad informática y subraya la importancia de adoptar enfoques proactivos y basados en el conocimiento actualizado.

"En años anteriores, la protección de la información era más sencilla, ya que las grandes plataformas de datos actuaban de forma independiente sin conectividad alguna, las arquitecturas existentes eran totalmente centralizadas y las terminales tenían capacidades de procesamiento limitadas. Este desarrollo ha ocasionado un aumento en la exposición de las organizaciones y de los usuarios a violaciones de seguridad que pueden poner en riesgo la confidencialidad, integridad y reputación de estos." (Iván Coronel Suárez, 2022).

Importancia de la Seguridad Integrada desde el Diseño: Introducir la seguridad desde las etapas iniciales del diseño y desarrollo web no solo es una estrategia efectiva para mitigar riesgos sino también un enfoque costo-eficiente, alineándose con las prácticas recomendadas en la industria del desarrollo de software.

"Después de haber trabajado para muchos clientes pequeños, medianos y grandes he escuchado, de tanto personas de negocio como de técnicos "expertos", que el frontend solo la pinta y colorea, que solo importa la seguridad del backend, no hace falta incluir test de calidad, etc. Pero la realidad que nos encontramos cuando un usuario accede a nuestra aplicación, la interfaz móvil o web, será la primero con lo que trate.

Los dos conceptos que deberemos tener en cuenta a nivel de seguridad en nuestro frontal son:

- Huecos e información de nuestro sistema
- Ataques directos a nuestro frontal." (CUESTA, J., 2019).

Cumplimiento Normativo y Responsabilidad Corporativa: Las regulaciones y estándares de seguridad y privacidad de datos presionan a las organizaciones para adoptar medidas rigurosas, subrayando la responsabilidad corporativa en la protección de la información del usuario. Publicado en el BOE-A-2021-8806 Ley Orgánica 7/2021, de 26 de mayo, de protección de datos personales tratados para fines de prevención, detección, investigación y enjuiciamiento de infracciones penales y de ejecución de sanciones penales.

Este proyecto se justifica dada la aportación a la Seguridad Informática al identificar un nicho específico dentro del vasto campo de la seguridad informática.

Además, se propone aplicaciones prácticas de sus hallazgos, ofreciendo a los desarrolladores y organizaciones herramientas y metodologías para fortalecer la seguridad en los desarrollos web.

1.2. Objetivos del Trabajo.

Este proyecto busca establecerse como un referente en el ámbito del desarrollo de aplicaciones web seguras, proporcionando una guía exhaustiva que cubra tanto el análisis

de la ciberseguridad aplicada a los dominios del Frontend y Backend como la práctica a través de casos aplicados. A través de un enfoque sistemático y detallado, se pretende subrayar la importancia de adoptar estrategias de seguridad efectivas y reconocer las vulnerabilidades comunes en el desarrollo web para mejorar la seguridad general de las aplicaciones web.

Los objetivos específicos detallados en el texto se pueden resumir en los siguientes puntos:

- **Desarrollo de una guía de desarrollo seguro para aplicaciones web:** Se creará un manual completo que oriente a los desarrolladores en el proceso de creación de aplicaciones web seguras, abarcando aspectos tanto del Frontend como del Backend.

- **Identificar Vulnerabilidades Comunes y Estrategias de Mitigación:** Analizar y documentar las vulnerabilidades más comunes que afectan a las aplicaciones web, junto con las estrategias de mitigación recomendadas para prevenirlas, basadas en estándares de seguridad reconocidos como **OWASP**.

- **Identificar y proponer estrategias de mitigación de riesgos**: Proporcionar directrices para la implementación de controles de seguridad efectivos que anticipen y neutralicen posibles amenazas a la seguridad, fomentando un enfoque proactivo en lugar de reactivo en el desarrollo web.

- **Mejorar la Educación y Conciencia sobre Seguridad en Desarrollo Web:** Elevar la conciencia sobre la importancia de la seguridad en el desarrollo de aplicaciones web mediante la educación y la capacitación de desarrolladores y diseñadores web en prácticas de seguridad.

- **Demostrar la Aplicabilidad de las Estrategias de Seguridad mediante un Caso Práctico:** Implementar las estrategias de seguridad discutidas en un caso de estudio práctico, proporcionando evidencia concreta de su eficacia y aplicabilidad en escenarios de desarrollo web reales.

- **Promover una Herramienta de Análisis de Código Fuente mediante Inteligencia Artificial:** Desarrollar y promover el uso de una herramienta avanzada que emplea técnicas de inteligencia artificial para analizar y detectar vulnerabilidades en el código fuente, mejorando así la seguridad y la eficiencia del proceso de desarrollo de software.

- **Validación Práctica a través de un Caso de Estudio con Enfoque en Autenticación y Autorización:** Implementar un caso práctico para evaluar la efectividad y la aplicabilidad de la guía propuesta en un escenario real de desarrollo web, con un enfoque particular en los sistemas de autenticación y autorización. Este caso de estudio aplicará las estrategias y recomendaciones de seguridad delineadas en la guía, enfocándose especialmente en cómo las directrices pueden mejorar la seguridad en las áreas críticas de autenticación y autorización. Este enfoque práctico pretende demostrar cómo las prácticas de seguridad pueden ser implementadas efectivamente para proteger contra

vulnerabilidades críticas, facilitando así la toma de decisiones informadas y promoviendo una mejora continua en la seguridad del desarrollo web.

- **Desarrollo y Promoción de una Herramienta de Análisis de Código Fuente Mediante Inteligencia Artificial:** Desarrollar y promover el uso de una herramienta avanzada de análisis de código fuente que utilice técnicas de inteligencia artificial para detectar y analizar vulnerabilidades. Esta herramienta será integrada como parte del caso práctico para demostrar su capacidad en el entorno real de desarrollo web. Al hacerlo, se busca evaluar la precisión y eficacia de la inteligencia artificial en la identificación proactiva de problemas de seguridad, contribuyendo así a la optimización de los procesos de desarrollo y fortalecimiento de las aplicaciones web frente a amenazas de seguridad emergentes.

1.3. Impacto en sostenibilidad, ético-social y de diversidad.

Los impactos del Trabajo Final de Grado (TFG) en las dimensiones de la competencia transversal "Compromiso ético y global" de la UOC pueden evaluarse de la siguiente manera:

a) Sostenibilidad: las prácticas de seguridad implementadas en el desarrollo de aplicaciones web influyen de manera positiva en diferentes aspectos ambientales y de sostenibilidad:

Consumo y ahorro energético: La optimización de código en el desarrollo y la gestión eficiente de dependencias pueden contribuir a una reducción en el consumo de recursos computacionales, lo cual indirectamente afecta el consumo energético. Aunque esta influencia es relativamente pequeña a escala individual, cuando se considera a nivel global, la optimización del software puede contribuir significativamente al ahorro energético.

Producción de residuos: Aunque un proyecto de software no produce residuos tangibles directos como lo harían los sectores industriales, la gestión eficiente de las dependencias y el mantenimiento actualizado del software evita la obsolescencia tecnológica y, por lo tanto, reduce la necesidad de reemplazar hardware frecuentemente, lo cual puede ser considerado como una reducción indirecta en la generación de residuos electrónicos.

Contaminación y agotamiento de materias primas: El enfoque en la seguridad del software y la gestión de dependencias contribuye a prolongar la vida útil del hardware necesario para operar dichas aplicaciones, lo cual a su vez puede ayudar a disminuir la demanda de nuevas producciones de equipos y, por lo tanto, la explotación de materias primas y los procesos contaminantes asociados a la fabricación de componentes electrónicos.

Respecto a los Objetivos de Desarrollo Sostenible (ODS), estas prácticas podrían alinearse indirectamente con:

ODS 12 - Producción y consumo responsables: Al promover el uso eficiente de recursos y la reducción en la necesidad de hardware a través de software optimizado y seguro.

ODS 9 - Industria, innovación e infraestructura: A través de la promoción de una infraestructura tecnológica más sostenible y resiliente.

b) **Ética y responsabilidad social**: este Trabajo Final de Grado (TFG) refleja el compromiso de desarrollar aplicaciones web seguras, lo cual tiene implicaciones directas en la protección de la privacidad del usuario y la integridad de los datos. Los aspectos destacados en el documento reflejan una serie de buenas prácticas y consideraciones éticas en el desarrollo de software:

Protección de Datos Confidenciales: La seguridad en el desarrollo de aplicaciones web no solo es una cuestión técnica sino también ética. Proteger la información confidencial de los usuarios es fundamental para preservar su privacidad y evitar el mal uso de sus datos. Implementar medidas de seguridad sólidas y mantener actualizadas las dependencias y el software ayuda a proteger contra vulnerabilidades que podrían exponer datos sensibles.

Integridad de la Aplicación: Seguir pautas de codificación segura y realizar pruebas de penetración son prácticas que contribuyen a la integridad y fiabilidad de las aplicaciones web. Estas prácticas son cruciales para evitar que actores malintencionados exploten vulnerabilidades y comprometan tanto la aplicación como los datos de los usuarios.

Responsabilidad en la Gestión de Dependencias: Elegir y gestionar cuidadosamente las dependencias en el desarrollo del software minimiza el riesgo de incorporar componentes con vulnerabilidades conocidas. Esto no solo afecta la seguridad de la aplicación sino también demuestra un compromiso con la construcción de software responsable y confiable.

Uso de Prácticas de Desarrollo Seguro: Adoptar una metodología de desarrollo que enfatice la seguridad desde el diseño hasta la implementación y el mantenimiento refleja un compromiso con la responsabilidad ética. Educar y capacitar a los desarrolladores en estas prácticas es fundamental para fomentar un enfoque proactivo hacia la seguridad.

Transparencia y Confianza del Usuario: Informar a los usuarios sobre las medidas de seguridad implementadas y cómo se protegen sus datos no solo cumple con las expectativas éticas, sino que también construye confianza. Promover la transparencia y permitir que los usuarios tengan control sobre sus datos personales son aspectos clave de la responsabilidad social y ética en el desarrollo web.

c) **Diversidad y derechos humanos:** Podemos considerar varios aspectos clave que se alinean con las dimensiones mencionadas en la guía:

Respeto por la Diversidad: El proyecto garantiza que no discrimina ni excluye a ningún grupo basado en género, raza, religión, orientación sexual, etnia,

ideología, o cualquier otra característica. Esto incluye tanto el proceso de desarrollo como la interfaz y funcionalidad del producto final, asegurando que sea accesible y usable para una amplia gama de usuarios.

Promoción de la Igualdad: El desarrollo y los resultados contribuyen a la promoción de la igualdad, apoyando los esfuerzos para reducir las desigualdades. Esto podría reflejarse en cómo el producto aborda problemas específicos que afectan a comunidades marginadas o en la forma en que fomenta prácticas inclusivas.

Cumplimiento de Legislación y Normativas: Es fundamental cumplir con las legislaciones y normativas vigentes relacionadas con la privacidad de datos, los derechos laborales, la propiedad intelectual y la seguridad.

Impacto en los ODS: El proyecto incide positivamente en aspectos relacionados con la igualdad de género (ODS 5) o la reducción de las desigualdades (ODS 10).

1.4. Enfoque y método seguido.

En el desarrollo de este trabajo, se ha realizado una evaluación exhaustiva de metodologías estándar reconocidas en el ámbito de la seguridad informática, con el objetivo de estructurar el proyecto de una manera más organizada y efectiva.

Después de un análisis detallado, se ha optado por integrar principios de la metodología **OWASP** (Open Web Application Security Project), específicamente aquellos relacionados con el desarrollo seguro de aplicaciones web, adaptándolos a las necesidades y objetivos específicos de este proyecto.

La elección de integrar la metodología **OWASP** se fundamenta en su amplio reconocimiento como un estándar para la seguridad en aplicaciones web, proporcionando un marco sólido y probado para identificar, clasificar y mitigar riesgos de seguridad web. Este nos permite centrar los esfuerzos en áreas críticas de la seguridad de aplicaciones web, desde la fase de diseño hasta la implementación y las pruebas de penetración, garantizando así una cobertura integral de los aspectos de seguridad relevantes.

Además, **se ha desarrollado e integrado una herramienta de análisis de código fuente utilizando técnicas avanzadas de inteligencia artificial.** Este sistema está diseñado para complementar las directrices de **OWASP** mediante la identificación automática de vulnerabilidades en el código, que a menudo pueden pasarse por alto en revisiones manuales.

La combinación de metodologías estándar de seguridad con tecnologías de IA avanzadas proporciona un enfoque robusto para el desarrollo seguro de aplicaciones web, alineando la prevención de riesgos con las mejores prácticas y la innovación tecnológica. No solo mejora la eficiencia del proceso de desarrollo, sino que también eleva la calidad y la seguridad del software producido.

Fases del Proyecto Basadas en la Metodología Ágil:

- **Planificación y Definición de Alcance:** Se definirán los objetivos del proyecto, identificando los requisitos de seguridad clave y estableciendo el alcance del análisis. Durante esta fase, se realizará también la selección de las tecnologías y herramientas a utilizar, incluyendo la integración de la herramienta de IA para el análisis de código fuente, asegurando que se alinean con las metas del proyecto y los estándares de seguridad actuales.

- **Análisis y Diseño de Seguridad:** Se llevará a cabo un análisis de riesgos basado en los criterios establecidos por **OWASP**, identificando vulnerabilidades potenciales y diseñando estrategias de mitigación. En esta etapa, se configurará la herramienta de IA para que analice los patrones de diseño y código utilizados, identificando inconsistencias y vulnerabilidades que no se ajusten a las prácticas de seguridad recomendadas.

- **Desarrollo Seguro:** Se aplicarán prácticas de codificación segura recomendadas por **OWASP** durante el desarrollo de la aplicación de entrenamiento para pruebas de penetración. **La herramienta de IA** se utilizará para monitorizar continuamente el código en desarrollo, proporcionando retroalimentación instantánea y sugerencias de mejora en tiempo real para asegurar la adherencia a los principios de seguridad desde las primeras etapas de desarrollo.

- **Pruebas de Seguridad:** Se llevarán a cabo pruebas de penetración siguiendo la metodología de pruebas de **OWASP**, permitiendo identificar y corregir vulnerabilidades efectivamente. **La herramienta de IA** también desempeñará un papel crucial en esta fase, analizando los datos de las pruebas de penetración para aprender de las interacciones y mejorar su capacidad de detección de futuras vulnerabilidades.

- **Evaluación y Mejora Continua:** Basándose en los resultados de las pruebas y el análisis proporcionado por la **herramienta de IA**, se realizarán ajustes en la aplicación, fomentando un enfoque de mejora continua en la seguridad. Esta fase incluirá la revisión de todas las métricas de seguridad recogidas durante el proyecto para evaluar la eficacia de las estrategias implementadas y determinar las áreas que requieren mayor atención o ajuste.

1.5. Planificación del Trabajo.

Para optimizar la guía destinada al desarrollo de Mecanismos de Seguridad en el Diseño de Desarrollo Web, es crucial iniciar con una definición clara del problema, seguido de un análisis meticuloso y la evaluación de la solución propuesta. Utilizamos el **diagrama de Gantt** como una herramienta estratégica para monitorear detalladamente el avance del proyecto.

El diagrama será actualizado progresivamente conforme avance el proyecto, proporcionando inicialmente una perspectiva de las etapas iniciales y las tareas a ejecutar en la primera fase.

Incluimos una ilustración detallada del plan de trabajo a implementar en este proyecto, subrayando los hitos clave que se deben alcanzar a lo largo de su desarrollo.

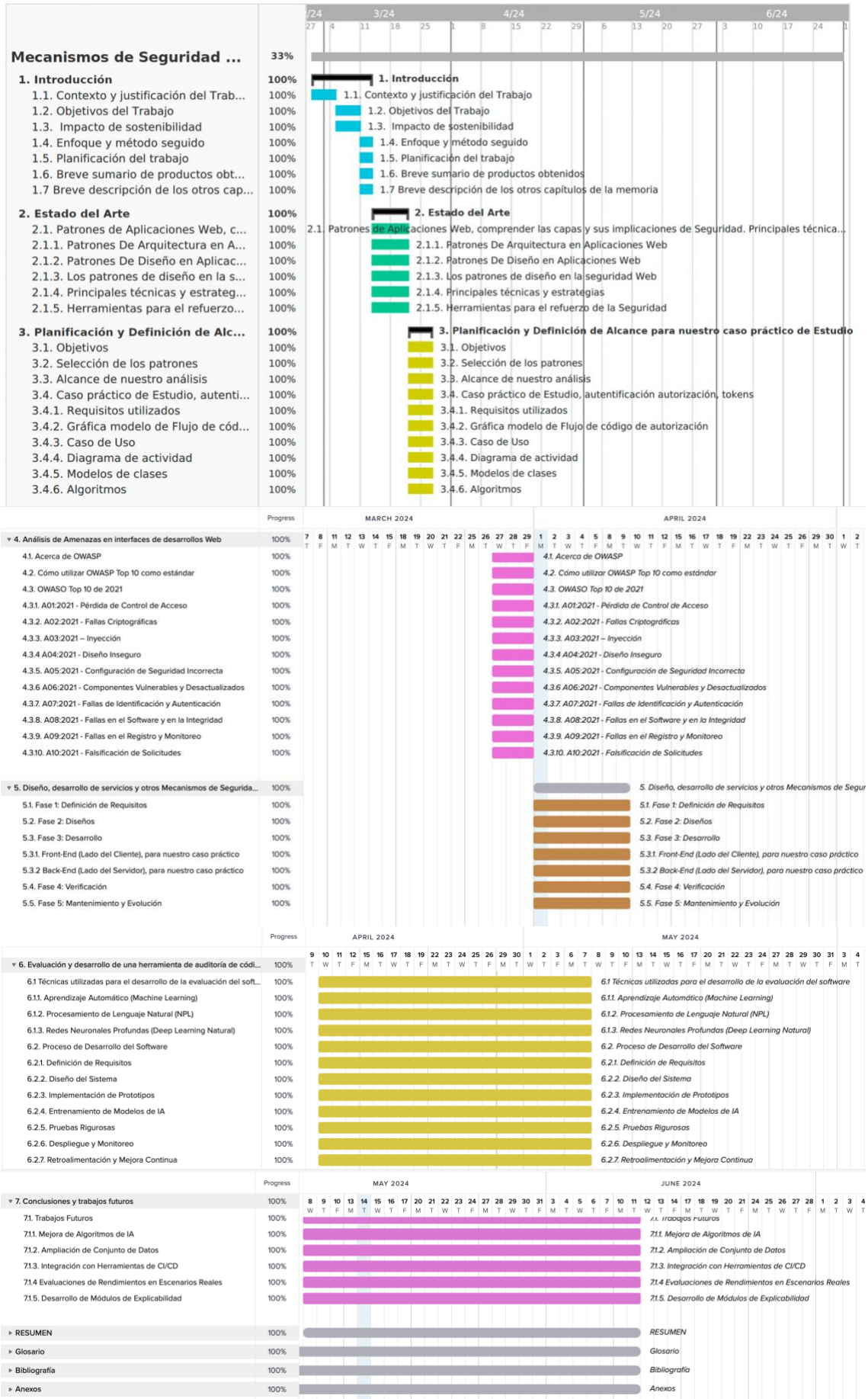

Ilustración 1: Planificación del Trabajo

Descripción Hito	Fecha Plan	Fecha Actual	Estado
Inicio de proyecto	28/02/2024	12/03/2024	Finalizado
PEC 2. Seguimiento del Trabajo - Entrega parcial	13/03/2924	09/04/2024	Finalizado
PEC 3. Seguimiento del trabajo - Entrega parcial.	10/04/2024	07/05/2024	Finalizado
PEC 4. Memoria final - Entrega definitiva.	10/05/2024	11/06/2024	Finalizado

Análisis de la evaluación de los riesgos identificados en la planificación inicial y su seguimiento y control.

Riesgo	Acción	Tipo	Riesgo	Fecha
Retrasos en la Investigación: La recopilación y el análisis de la vasta información normativa y metodológica pueden demorarse más de lo esperado.	Establecer un cronograma de trabajo realista con hitos y revisiones periódicas. En caso de retrasos, reasignar recursos o ajustar el cronograma sin comprometer la calidad del análisis.	Mitigadora	**Medio.**	**25/03/2024**
Cambios en Normativas o Tecnologías: Las regulaciones de seguridad informática y las tecnologías evolucionan rápidamente, lo que podría hacer que la información inicialmente recopilada quede obsoleta.	Mantenerse actualizado con las tendencias y cambios normativos durante todo el proyecto. Si se identifican cambios significativos, reevaluar y ajustar la estrategia de investigación y diseño acordemente.	No necesario.	**Bajo.**	**30/05/2024**
Descubrimiento de Vulnerabilidades Inesperadas: Durante el análisis de riesgos, pueden identificarse vulnerabilidades críticas no anticipadas**.**	Destinar recursos para una evaluación y respuesta inmediatas a cualquier vulnerabilidad crítica descubierta, incluyendo la revisión y adaptación de las prácticas de diseño propuestas.	Correctora	**Medio.**	**30/05/2024**
Limitaciones de Recursos: Restricciones en la disponibilidad de material cualificado pueden impactar el avance del proyecto.	Priorizar las actividades críticas y buscar eficiencias en el trabajo. Considerar la posibilidad de desarrollar las competencias necesarias o la reasignación de tareas para optimizar el uso de los recursos disponibles.	Mitigadora	**Bajo.**	**30/05/2024**

Falta de Cooperación o Compromiso: La falta de compromiso en el desarrollo, de las partes interesadas, puede obstaculizar el desarrollo del proyecto	Fomentar una cultura asegurando que todos los involucrados comprendan la importancia su papel en el proyecto. Establecer canales de comunicación efectivos y mecanismos de feedback.		**Muy Bajo.**	**Cierre**
Plazos ajustados. No alcanzar las fechas previstas en cada hito.	Ajustar los planes de seguimientos y cambiar la metodología seguida.	Correctora	**Alto.**	**30/05/2024**
Complejidad tecnológica del estudio. No alcanzar el entendimiento en el desarrollo de los mecanismos de seguridad, puede ralentizar el cumplimiento del cierre del proyecto	Ajustar los planes de seguimientos y cambiar la metodología seguida.	Correctora	**Alto.**	**30/05/2024**

1.6. Breve sumario de productos obtenidos.

El modelo de seguimiento a desarrollar en el proyecto será:

- **Casos de estudio:** Esta fase inicial, proporcionará los conocimientos esenciales para comprender las subsiguientes fases y plantea el problema central que orientará el enfoque del proyecto.

- **Evaluación de Riesgos:** En este apartado se examinan meticulosamente las amenazas prevalentes en el ámbito de las aplicaciones web, desde las más amplias hasta aquellas particularmente relevantes para el desarrollo.

- **Estrategias y Herramientas de Protección:** Tras identificar las amenazas, este capítulo aborda los servicios de protección críticos en aplicaciones web, describiendo los mecanismos de seguridad aplicables para implementar estos servicios.

- **Elecciones Cruciales en el Diseño**: Este capítulo se enfoca en las decisiones clave, desde la perspectiva de seguridad, que deben tomarse durante el diseño de una aplicación.

- **Síntesis y Reflexiones:** Después de presentar los problemas y sus posibles soluciones, se realiza una síntesis de los temas abordados, reflexionando sobre los puntos más significativos.

- **Conclusión y Asesoramiento:** Como colofón, se presentan las deducciones obtenidas de un análisis detallado sobre la seguridad en el desarrollo de aplicaciones, junto con una serie de recomendaciones definitivas.

1.7. Breve descripción de los otros capítulos de la memoria.

Resumen esclarecedor de los componentes clave que conforman nuestra memoria, estructurada para proporcionar una comprensión integral del proyecto y sus resultados:

- **Introducción:** Se detalla la concepción y el propósito del sistema desarrollado, subrayando la imperiosa necesidad de su implementación. Esta sección contextualiza el proyecto dentro de su campo de aplicación, ofreciendo una panorámica de su alcance y objetivos.

- **Recursos Utilizados:** Enumera y describe exhaustivamente los recursos tanto software como hardware movilizados para el adelanto del proyecto. Esta exposición ilustra las herramientas y tecnologías fundamentales para la exploración meticulosa de los mecanismos de seguridad críticos en el desarrollo de aplicaciones web.

- **Capítulo 1: Patrones de Aplicaciones Web, comprender las capas y sus implicaciones de Seguridad. Principales técnicas y estrategias para asegurar el desarrollo web. Herramientas para el refuerzo de la Seguridad en el Desarrollo Web:** Divide y expone claramente las responsabilidades entre Cliente (Frontend) y Servidor (Backend). Discute las arquitecturas de desarrollo preeminentes y sus patrones asociados, destacando cómo una estructuración apropiada tanto del Backend como del Frontend cataliza una interacción usuario-aplicación enriquecida y fluida.

- **Capítulo 2: Análisis Profundo de Amenazas en Interfaces de Desarrollo:** Profundiza en la identificación y evaluación de amenazas para asegurar la robustez y seguridad en interfaces accesibles a través de diversos dispositivos. Esta evaluación crítica conduce al reconocimiento y la integración de servicios y mecanismos de seguridad esenciales, delineando un marco para la mitigación efectiva de riesgos.

- **Capítulo 3: Diseño, desarrollo de servicios y otros Mecanismos de Seguridad:** Detalla servicios de seguridad imperativos, organizados en categorías esenciales: Autenticación, Autorización, Integridad de Datos, Confidencialidad y Disponibilidad. La integración de estas medidas en el diseño y desarrollo subraya el compromiso con una protección avanzada y una experiencia de usuario fiable.

- **Capítulo 4: Evaluación y desarrollo de una herramienta de auditoría de código fuente automatizada que utilice técnicas avanzadas de inteligencia artificial:** Para contribuir innovadoramente en el campo de la seguridad informática y diseño de desarrollo web, se propone explorar el desarrollo de una herramienta de auditoría de código fuente automatizada que utilice técnicas avanzadas de inteligencia artificial. Esta herramienta podría integrar aprendizaje profundo para identificar y aprender patrones de vulnerabilidades en el código de manera más efectiva que los sistemas tradicionales de detección de vulnerabilidades. Además, podría adaptarse continuamente a nuevas amenazas y técnicas de ataque emergentes mediante el aprendizaje automático continuo.

- **Capítulo 5: Reflexiones y Conclusiones:** Articula una reflexión detallada sobre las conclusiones del proyecto, evaluando la eficacia de las decisiones y estrategias implementadas. Esta evaluación valida la selección de tecnologías y la arquitectura empleada, culminando en un reconocimiento del éxito del proyecto al cumplir con los requerimientos y los plazos, subrayando su eficacia y relevancia.

2. Estado del arte.

En la actual era digital, donde la innovación tecnológica avanza a un ritmo vertiginoso, la informática emerge como un soporte esencial de nuestra sociedad moderna, impulsando transformaciones en cada aspecto de nuestras vidas.

El estudio que abordaremos proporciona una perspectiva profunda sobre el impacto y la evolución de las tecnologías informáticas, ilustrando la importancia crítica de la integración entre hardware, software y la interfaz con el usuario en la construcción de sistemas informáticos robustos, eficientes y adaptativos. Mientras que el hardware constituye la base tangible de estos sistemas, el software introduce la inteligencia y flexibilidad requeridas para ejecutar operaciones complejas, y la interfaz del usuario garantiza que la interacción hombre-máquina sea intuitiva, efectiva y enriquecedora.

Este apartado se adentra en las diversas dimensiones de los sistemas informáticos modernos, explorando tanto los lenguajes de programación que actúan como el alma de cualquier aplicación, como los entornos de desarrollo que proporcionan el marco necesario para su creación y mantenimiento. Desde Visual Basic .NET hasta Java, cada lenguaje y entorno de desarrollo tiene sus particularidades, aplicaciones preferentes y comunidades de desarrolladores, conformando un ecosistema tecnológico diverso y dinámico. La elección de un lenguaje de programación depende de sus conocimientos del lenguaje y del ámbito de la aplicación que esta generado. Las aplicaciones de pequeño tamaño se suelen crear utilizando un único lenguaje, y es frecuente implementar aplicaciones grandes utilizando varios lenguajes.

Antes de adentrarnos en el estudio de investigación de nuestro proyecto, es necesario tener claro los siguientes conceptos, de manera simplificada, con el fin poder centrarnos en los casos de usos que se mostrarán a lo largo del mismo.

Sistemas Informáticos: El proyecto detalla cómo un sistema informático integra hardware, software y **humanware**, subrayando la relevancia de cada uno para la eficacia operativa. Resalta la importancia de los sistemas informáticos en las empresas, indicando que facilitan el acceso a la información, mejoran la atención al usuario, fomentan la colaboración y aumentan la productividad y expone la FACULTAD DE CIENCIAS Y TECNOLOGÍA Universidad Isabel I, (Año 2023).

Ilustración 2: Sistemas Informáticos

Ilustración 3: Lenguajes de Programación.

"Un sistema informático (SI) es un sistema que permite almacenar y procesar información; es el conjunto de partes interrelacionadas: hardware, software y personal informático. El hardware incluye computadoras o cualquier tipo de dispositivo electrónico, que consisten en procesadores, memoria, sistemas de almacenamiento externo, etc. El software incluye al sistema operativo, firmware y aplicaciones, siendo especialmente importante los sistemas de gestión de bases de datos. Por último, el componente humano incluye al personal técnico que apoya y mantienen el sistema (analistas, programadores, operarios, etc.) y a los usuarios que lo utilizan." (Wikipedia.org).

Lenguajes de Programación: El lenguaje es esencial para la comunicación humana, usando palabras y gestos para compartir ideas. Análogamente, en una computadora, el hardware y software interactúan mediante código binario, un lenguaje compuesto de unos y ceros, para ejecutar tareas. Esto permite que la computadora funcione correctamente, similar a cómo el lenguaje facilita la interacción entre personas. (MENDOZA, M.L., 2020).

"Un lenguaje de programación es un lenguaje formal (o artificial, es decir, un lenguaje con reglas gramaticales bien definidas) que proporciona a una persona, en este caso el programador, la capacidad y habilidad de escribir (o programar) una serie de instrucciones o secuencias de órdenes en forma de algoritmos con el fin de controlar el comportamiento físico o lógico de un sistema informático, para que de esa manera se puedan obtener diversas clases de datos o ejecutar determinadas tareas. A todo este conjunto de órdenes escritas mediante un lenguaje de programación se le denomina programa informático." (Wikipedia.org).

A continuación, destacamos algunos de los lenguajes de programación más usados en la actualidad:

Visual Basic .NET: Descrito como una evolución del Visual Basic tradicional, facilita la creación de aplicaciones .NET, incluyendo servicios web y aplicaciones web ASP.NET, aprovechando las ventajas del .NET framework.

Visual C# .NET: Lenguaje moderno y eficiente, diseñado para el desarrollo rápido de aplicaciones .NET, que se beneficia de las características del Common Language Runtime y el .NET Framework.

Visual C++.NET: Continuación del legado de Visual C++, se centra en el desarrollo de aplicaciones de alto rendimiento, con acceso a una amplia biblioteca de recursos.

Transact-SQL: Lenguaje específico para el manejo de bases de datos en SQL Server, esencial para la gestión de datos relacionales.

JAVA: Lenguaje orientado a objetos y plataforma independiente, destacado por su flexibilidad y adaptabilidad en diversos entornos de programación.

Entornos de Programación: Se refiere al conjunto de métodos y herramientas usadas para crear programas o código fuente, conocido también como Entorno de Desarrollo Integrado (IDE). Estos entornos facilitan la creación de software al integrar múltiples servicios en una sola plataforma, mejorando la eficiencia y reduciendo costos en proyectos de desarrollo. Permiten programar tanto el Frontend como el Backend de aplicaciones, siendo esenciales para la productividad del desarrollador por su intuitivita y facilidad de uso. Los entornos de desarrollo incluyen etapas como desarrollo, integración y producción, y deben ser multiplataforma, contar con interfaces atractivas y ofrecer asistencia y recursos comunitarios para optimizar la experiencia del usuario.

"Un entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés integrated development environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitar al desarrollador o programador el desarrollo de software." (Wikipedia.org).

Ilustración 4: Entornos de Programación.

A continuación, destacamos algunos de los entornos de programación más usados en la actualidad:

Visual Studio: IDE versátil para Windows y Mac que soporta numerosos lenguajes de programación, permitiendo el desarrollo de una amplia gama de aplicaciones y servicios.

NetBeans: Entorno centrado en Java, pero extensible a otros lenguajes mediante módulos, promoviendo un desarrollo colaborativo y modular.

2.1 Patrones de Aplicaciones Web, comprender las capas y sus implicaciones de Seguridad. Principales técnicas y estrategias para asegurar el desarrollo web. Herramientas para el refuerzo de la Seguridad en el Desarrollo Web.

Desarrollo Web: La creación de sitios web se refiere al proceso de edificar y mantener programas informáticos accesibles mediante un navegador web, conocidos como aplicaciones web. Este proceso implica una serie de pasos variados, los cuales, en términos generales, se pueden clasificar en dos categorías principales:

- **Lado del cliente (front-end):** Incluye todas aquellas actividades relacionadas con el aspecto de la aplicación que los usuarios finales ven y con el cual interactúan en sus navegadores web. Esta parte se encarga de la interacción con los usuarios y debe comunicarse adecuadamente con los servicios del lado del servidor para operar de manera efectiva.

- **Lado del servidor (back-end):** Engloba las tareas involucradas en construir el servicio o servicios web que ejecutan la lógica necesaria de la aplicación, proporcionando soporte a los clientes basándose en las interacciones de los usuarios.

2.1.1. Patrones De Arquitectura en Aplicaciones Web.

Los patrones de arquitectura son soluciones generalizadas y reutilizables para problemas comunes en el diseño de sistemas de software, representando las mejores prácticas que han surgido a partir de la experiencia colectiva en el campo. Estos patrones ofrecen un marco estructurado para diseñar y construir sistemas que sean flexibles, mantenibles y escalables, enfocándose en los aspectos de alto nivel del diseño, como la distribución de responsabilidades, comunicación entre componentes, gestión de datos, y la estructura general del sistema. (MURILLO, R., 2023).

"Un patrón arquitectónico es una solución general y reutilizable a un problema común en la arquitectura de software dentro de un contexto dado. Los patrones arquitectónicos son similares al patrón de diseño de software, pero tienen un alcance más amplio." (Wilber Ccori huaman, 2018).

Las ventajas de utilizar patrones de arquitectura incluyen la identificación de características básicas de una aplicación, garantía de calidad y eficiencia, fomento de la agilidad en el desarrollo, resolución de problemas recurrentes, y aumento de la productividad. Estos beneficios derivan de la aplicación de soluciones probadas que han

sido mejoradas a lo largo del tiempo, permitiendo a los equipos de desarrollo evitar errores comunes y tomar decisiones informadas.

Patrones de Arquitecturas más conocidas hoy en día.

- **Patrón de Capas:** Divide el sistema en capas lógicas con responsabilidades específicas, promoviendo la separación de preocupaciones y facilitando el mantenimiento.

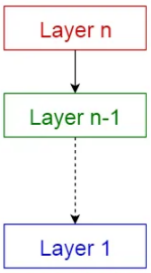

Ilustración 5: Patrón de Capas. SOURCE: https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b

- **Patrón Cliente-Servidor**: Divide el sistema en dos partes, cliente y servidor, permitiendo una clara distribución de responsabilidades y facilitando la escalabilidad.

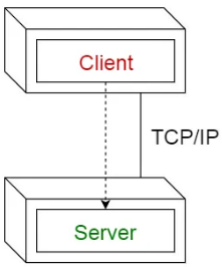

Ilustración 6: Patrón Cliente-Servidor. SOURCE: https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b

- **Patrón maestro-esclavo:** Este diseño se basa en una estructura de dos niveles, denominados maestro y esclavos. El elemento maestro tiene la tarea de asignar labores a varios componentes esclavos, los cuales son copias exactas entre sí y trabajan las asignaciones de manera simultánea. Una vez que los esclavos completan sus respectivas tareas, envían sus resultados al maestro, quien entonces se encarga de compilar estos resultados individuales para producir el resultado global o final. Este método permite una eficiente distribución y procesamiento paralelo de las tareas, optimizando el rendimiento y la gestión de los recursos disponibles.

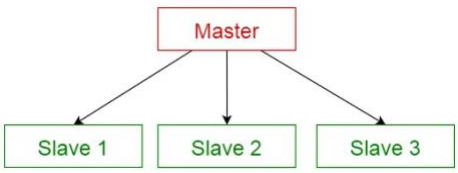

Ilustración 7: Patrón Maestro-esclavo. SOURCE: https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b

- **Patrón de filtro de tubería:** Este diseño se aplica en la organización de sistemas que generan y manipulan flujos de datos. Se organiza cada etapa del procesamiento en módulos conocidos como filtros. Los datos por procesar se transmiten a través de conexiones denominadas tuberías, que pueden servir tanto para almacenar temporalmente datos como para sincronizar las diferentes etapas de procesamiento.

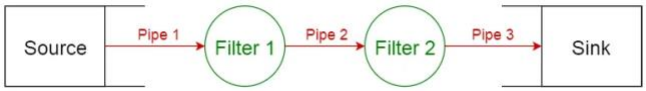

Ilustración 8: Patrón Filtro de tubería. SOURCE: https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b

- **Patrón de agente:** Este diseño se emplea para organizar sistemas distribuidos compuestos por elementos independientes entre sí. Estos elementos tienen la capacidad de comunicarse mediante el uso de llamadas a servicios ubicados en diferentes puntos de la red. Existe un componente central, actuando como mediador, cuya función es facilitar y coordinar esta interacción entre los diversos componentes del sistema.

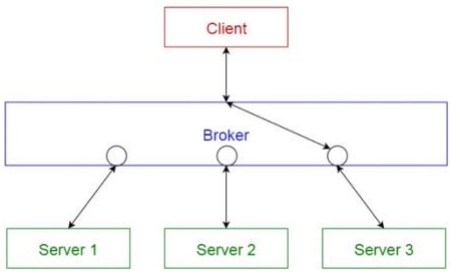

Ilustración 9: Patrón Agente. SOURCE: https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b

- **Patrón de igual a igual:** En este modelo, cada elemento del sistema se denomina par, y tiene la capacidad de desempeñar dualmente roles: puede ser cliente, solicitando servicios de otros elementos, o servidor, ofreciendo servicios a sus congéneres. Los pares tienen la flexibilidad de alternar entre actuar como cliente, servidor, o ambos simultáneamente, y pueden ajustar su función de manera dinámica conforme evolucionan las necesidades y el contexto.

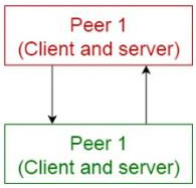

Ilustración 10: Patrón Igual a Igual. SOURCE: https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b

- **Patrón de bus de evento:** Este diseño se centra en la gestión de eventos y se compone de cuatro elementos clave: el origen del evento, el receptor del evento, el canal, y el bus de eventos. Los orígenes generan y envían mensajes a través de canales específicos dentro de un bus de eventos. Los receptores, por su parte, se inscriben en estos canales específicos para recibir notificaciones. Cuando un mensaje se difunde en un canal al que un receptor está suscrito, este último recibe una alerta del mensaje publicado.

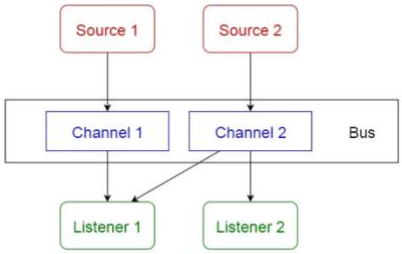

Ilustración 11: Patrón Bus de Evento. SOURCE: https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b

- **Modelo-Vista-Controlador (MVC):** Separa la lógica de negocio, la interfaz de usuario y la interacción entre ambos, facilitando la modularidad y mantenimiento del sistema.

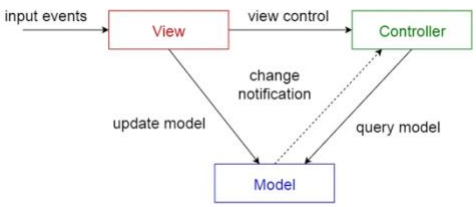

Ilustración 12: Patrón Modelo-Vista-Controlador. SOURCE: https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b

- **Modelo-Vista-Vista-Modelo (MVVM):** El patrón Model View View Model (MVVM) es una evolución del patrón Modelo-Vista-Controlador (MVC) que busca optimizar la relación entre la interfaz gráfica de usuario (la vista) y los datos o lógica de la aplicación (el modelo). En MVVM, la vista y el modelo están desacoplados mediante una capa intermedia conocida como ViewModel. Esta capa actúa como un puente que sincroniza los datos del modelo con los componentes visuales, de tal manera que solo las partes relevantes del modelo se vinculan con elementos específicos de la vista.

  Una característica distintiva de MVVM es el enlace de datos bidireccional (two-way data binding) entre la vista y el ViewModel. Este mecanismo permite que los cambios en el estado del modelo se reflejen automáticamente en la vista, y viceversa, facilitando así una actualización dinámica y fluida de la interfaz de usuario en respuesta a las interacciones del usuario o cambios en los datos. Gracias a esta automatización en la actualización de la interfaz, el desarrollo y mantenimiento de aplicaciones se vuelve más eficiente, ya que reduce la necesidad de manipular manualmente los elementos de la vista para reflejar los cambios en los datos o en el estado de la aplicación.

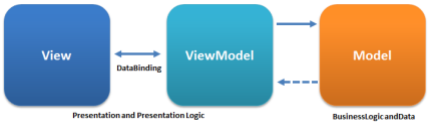

Ilustración 13: Patrón Modelo-Vista-Vista-Modelo. SOURCE: https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-

2.1.2. Patrones De Diseño en Aplicaciones Web.

Los patrones de diseño son aplicables a cualquier lenguaje de programación y pueden ser implementados en cualquier tipo de proyecto, ofreciendo un esquema general en lugar de soluciones específicas. El libro "Design Patterns - Elements of Reusable Object-Oriented Software" identifica 23 patrones oficiales y es considerado uno de los textos más influyentes en el ámbito del desarrollo orientado a objetos y la programación en general. A través de la aplicación de estos patrones, los desarrolladores pueden facilitar la comunicación dentro del equipo, optimizar el proceso de desarrollo y asegurar un enfoque coherente y efectivo para la resolución de problemas. (GOMEZ, E.E.P., 2023).

Patrones de Diseño más conocidas hoy en día:

- **El patrón de diseño Singleton:** El patrón Singleton se asegura de que solo exista una única instancia de una clase u objeto, manteniendo dicha instancia en una variable global. Este enfoque permite la creación de la instancia únicamente cuando es necesaria, conocido como carga perezosa, evitando así la existencia simultánea de múltiples instancias que podrían conducir a comportamientos erráticos inesperados.

  Comúnmente, esta singularidad se garantiza mediante la configuración apropiada del constructor de la clase. El propósito principal del patrón Singleton es gestionar y controlar el estado global de una aplicación.

  Un caso práctico del uso del patrón Singleton es el sistema de registro (logging) que se utiliza de manera habitual. En el contexto de frameworks de desarrollo frontend como React o Angular, manejar los registros generados por múltiples componentes puede ser complicado. Aquí es donde el patrón Singleton brilla, al asegurar que solo exista una instancia del objeto de registro. De esta forma, independientemente de dónde se genere un registro dentro del código, este se dirigirá a la misma instancia del sistema de registro. Esto evita la pérdida de información de registro que podría ocurrir si se creasen nuevas instancias del objeto de registro, manteniendo la integridad y centralización de los datos de seguimiento de errores.

- **El patrón de diseño estrategia:** El patrón de estrategia es esencialmente una forma sofisticada de elegir entre diferentes comportamientos o algoritmos mediante el uso de una estructura condicional, similar a cómo se usaría una sentencia if-else, pero a un nivel más abstracto y flexible. Este patrón implica crear una interfaz que actúa como un contrato para un método específico dentro de una clase base. Dicha interfaz se utiliza después para seleccionar la implementación adecuada de dicho método que será ejecutada en una clase

derivada, siendo esta selección realizada en tiempo de ejecución y dependiendo de las necesidades del cliente.

Esta técnica es particularmente valiosa en escenarios donde una clase puede tener métodos que son tanto obligatorios como opcionales. En algunos casos, ciertas instancias de la clase quizás no requieran de los métodos opcionales, presentando un desafío para el modelo de herencia tradicional. Aunque el uso de interfaces para estos métodos opcionales es una opción, esto implicaría tener que reescribir la implementación cada vez que la clase se utilice, dado que no se contaría con una implementación predeterminada.

El patrón de estrategia ofrece una solución elegante a este problema, permitiendo que sea la propia interfaz de estrategia la que se encargue de delegar y seleccionar la implementación correcta, eliminando la necesidad de que el cliente directamente especifique cuál implementación utilizar. Un ejemplo práctico de su aplicación se encuentra en los sistemas de procesamiento de pagos. Considerando un carrito de compras que inicialmente solo acepta pagos con tarjeta de crédito, la inclusión de este patrón posibilita la incorporación de nuevos métodos de pago sin la necesidad de alterar el código existente del carrito de compras o del proceso de pago, ofreciendo así una mayor flexibilidad y adaptabilidad en la gestión de diferentes opciones de pago para los clientes.

- **El patrón de diseño observador:** Si alguna vez has trabajado con el modelo MVC (Modelo-Vista-Controlador), en realidad has aplicado el patrón observador sin darte cuenta. En este contexto, el Modelo actúa como el sujeto que alberga todos los datos y el estado actual de esos datos. Por otro lado, la Vista funciona como un observador que se mantiene al tanto de los cambios en el sujeto (el Modelo). Los observadores, o componentes de la Vista, son notificados y se actualizan con los nuevos datos cada vez que hay un cambio en el Modelo.

  El propósito principal del patrón observador es establecer una dinámica de comunicación uno-a-muchos entre el sujeto y sus múltiples observadores. Esta relación asegura que, ante cualquier modificación en el estado del sujeto, todos los observadores asociados reciban una actualización de forma inmediata, manteniéndolos sincronizados con el estado actual del sujeto.

  Este patrón es especialmente útil en situaciones como el envío de notificaciones a usuarios, la actualización de interfaces basadas en cambios de estado, la implementación de filtros dinámicos, o la gestión de suscripciones a ciertos tipos de información.

  Un escenario práctico donde el patrón observador es invaluable ocurre en aplicaciones web de página única, particularmente en aquellas que ofrecen características interactivas como listas desplegables anidadas, donde la selección en un menú desplegable de nivel superior afecta las opciones disponibles en los menús subsecuentes. Este comportamiento es común en sitios de comercio electrónico, donde los filtros de productos pueden ajustarse dinámicamente en función de selecciones previas, mejorando así la experiencia de navegación del usuario al proporcionar opciones relevantes y personalizadas basadas en sus intereses.

- **El patrón de diseño decorador:** El patrón de diseño decorador ofrece una solución elegante y flexible para extender la funcionalidad de los objetos sin necesidad de modificar la clase base o recurrir a la herencia, que puede resultar engorrosa o ineficaz en ciertos contextos. Imagina que tienes una clase base con un conjunto de métodos y propiedades. Eventualmente, puedes encontrarte en la situación de requerir funcionalidades adicionales para ciertas instancias de esta clase, las cuales no están presentes inicialmente.

  Añadir directamente estas nuevas funcionalidades a la clase base podría complicar el código o afectar otras instancias que no necesitan estas extensiones. La creación de subclases específicas es una alternativa, pero puede llevar a una inflación del código y a una jerarquía de clases difícil de manejar.

  Aquí es donde brilla el patrón decorador. Permite "decorar" o añadir nuevas propiedades y métodos específicamente a las instancias que lo requieran, sin alterar la clase base ni impactar a otras instancias. Por ejemplo, si necesitas añadir una propiedad para manejar el precio a una instancia específica de un objeto, puedes hacerlo mediante el decorador, adaptando esa instancia a tus necesidades sin afectar al resto.

  Un ejemplo cotidiano de este patrón se puede ver en los servicios de pedido de comida en línea. Al personalizar un sándwich con ingredientes extra, el sistema no modifica la receta base del sándwich para todos los pedidos; en cambio, añade tus selecciones específicas únicamente a tu pedido, demostrando cómo el patrón decorador permite la personalización individual sin alterar el objeto base o afectar las elecciones de otros usuarios.

2.1.3. Los patrones de diseño en la seguridad Web.

Los patrones de diseño en la seguridad web constituyen un conjunto de estrategias y metodologías basadas en principios de ingeniería de software y seguridad informática, orientadas a fortalecer la protección de aplicaciones y sistemas web contra amenazas cibernéticas. Estos patrones se derivan de prácticas consolidadas y lecciones aprendidas en el ámbito de la seguridad, proporcionando un marco estructurado para abordar problemas específicos de seguridad en entornos digitales.

Cada patrón se centra en un desafío particular de seguridad, describiendo el contexto del problema, el mecanismo de ataque potencial y, más importante aún, la solución recomendada para mitigarlo. Estas soluciones pueden incluir, entre otros, la validación de entrada, la autenticación y autorización robustas, la gestión segura de sesiones, y el cifrado de datos. Al aplicar estos patrones, los desarrolladores y arquitectos de seguridad pueden implementar defensas proactivas, minimizando las vulnerabilidades explotables por actores maliciosos.

La importancia de los patrones de diseño en seguridad web se manifiesta en su capacidad para estandarizar las mejores prácticas de seguridad, facilitando así su adopción en el desarrollo y mantenimiento de proyectos web. Estos patrones ayudan a prevenir errores comunes de seguridad, ofrecen un lenguaje común entre profesionales

para discutir estrategias de protección y permiten la reutilización de soluciones probadas en diferentes contextos.

En el panorama actual, donde las amenazas cibernéticas evolucionan rápidamente, la implementación de patrones de diseño en seguridad web se convierte en un componente fundamental de cualquier estrategia de defensa cibernética. Su aplicación no solo mejora la postura de seguridad de las aplicaciones y sistemas web, sino que también contribuye a una cultura de seguridad más informada y resiliente dentro de las organizaciones. (Huasteca Network, sin fecha).

Patrones de Diseño en la seguridad Web más conocidas hoy en día:

- **Inicio de sesión único:** El acceso unificado (SSO) es una estrategia de seguridad diseñada para permitir a los usuarios entrar a diversas aplicaciones o servicios utilizando solo un juego de credenciales. Esto elimina la necesidad de memorizar y teclear contraseñas distintas para cada servicio, habilitando a los usuarios a autenticarse en varias plataformas con un único inicio de sesión. Este método mejora la usabilidad para el usuario, minimiza la posibilidad de infracciones de seguridad relacionadas con contraseñas y facilita la gestión de identidades. El SSO puede ser habilitado a través de diferentes estándares y protocolos, tales como OAuth, OpenID Connect y SAML.

- **Confianza cero:** La Confianza Cero es un enfoque en seguridad que opera bajo la premisa de no asumir confianza inherente hacia ninguna entidad, ya sea red, dispositivo o usuario. Contrario a las estrategias de seguridad perimetral convencionales, que se apoyan en firewalls o VPNs, este modelo exige una validación rigurosa de cada solicitud de acceso y transacción. Implementa el concepto de privilegio mínimo, otorgando a usuarios y sistemas únicamente el acceso esencial para sus funciones específicas. La adopción del modelo de Confianza Cero eleva el nivel de seguridad, mejora el control y la visibilidad sobre las operaciones de TI, y fortalece la capacidad de respuesta en ambientes tanto híbridos como basados en la nube.

- **Patrón de diseño: Control de acceso basado en roles:** Este enfoque de seguridad implica limitar la entrada a los activos digitales de una aplicación en línea según las responsabilidades asignadas a los usuarios.

  A cada usuario se le otorga un conjunto de permisos específicos basados en su rol, lo que restringe su capacidad para ver o manipular datos y funcionalidades no permitidas. Así, se impide que los usuarios interactúen con contenido o ejecuten tareas para las cuales no están autorizados. Este método es crucial para asegurar que la información sensible y los datos permanezcan confidenciales y no alterados.

- **Patrón de diseño: Validación de entrada:** Este modelo estructural se centra en verificar la corrección de los datos introducidos en una aplicación web. Hay que asegurar que las entradas cumplan con normas específicas es vital para bloquear amenazas como la inserción de SQL o códigos malintencionados. Mediante la implementación de controles rigurosos de validación, se garantiza que los datos

ingresados respeten parámetros establecidos, previniendo de esta forma vulnerabilidades de seguridad.

- **Patrón de diseño: Encriptación de datos:** El cifrado de información es fundamental para preservar la privacidad de los datos. Este enfoque utiliza técnicas de cifrado para transformar la información en un formato indescifrable para aquellos sin la llave correcta para revertir el proceso. El objetivo del cifrado es evitar que individuos no autorizados accedan a datos confidenciales, ya sea mientras estos se trasladan por la red o cuando están almacenados.

Estas estrategias son ejemplos de cómo los patrones de diseño en seguridad web afrontan problemas específicos, ofreciendo soluciones robustas para la seguridad de aplicaciones y plataformas online.

2.1.4. Principales técnicas y estrategias para asegurar el desarrollo Web.

Para asegurar el desarrollo web de forma efectiva, se deben adoptar diversas estrategias y técnicas que ayuden a prevenir riesgos y fortalecer la protección de las aplicaciones web. A continuación, se presentan algunas de las prácticas fundamentales en este ámbito, descritas por MSuiteAgency (Año 2024):

- **Evaluación Proactiva de la Seguridad en Aplicaciones Web:** Consiste en una revisión exhaustiva de las aplicaciones web para identificar y solucionar posibles brechas de seguridad. Esta evaluación incluye la revisión del código fuente, la ejecución de pruebas de intrusión y la verificación de la adhesión a las prácticas recomendadas de seguridad.

- **Adopción de Protocolos de Seguridad como HTTPS:** HTTPS (Protocolo de Transferencia de Hipertexto Seguro) es la versión protegida de HTTP que encripta la comunicación entre el servidor y el cliente. Su implementación es crucial para la protección de datos sensibles transmitidos en línea, incluyendo contraseñas e información financiera.

- **Verificación Rigurosa de los Datos Ingresados por los Usuarios:** Esta técnica implica comprobar la validez de los datos proporcionados por los usuarios para asegurar que cumplan con los criterios establecidos y estén libres de contenido malicioso. La validación previene la inserción de código perjudicial, como inyecciones SQL o scripts malintencionados.

- **Implementación de Sistemas de Autenticación y Autorización:** Estas prácticas se refieren a la implementación de sistemas para controlar el acceso a recursos y datos de la aplicación web. La autenticación verifica la identidad del usuario, mientras que la autorización determina los permisos de acceso del usuario autenticado.

- **Gestión Segura de las Sesiones de Usuario:** Implica implementar medidas para administrar las sesiones de forma segura, protegiéndolas contra ataques como la suplantación de identidad o el secuestro de sesión. Utilizar tokens de sesión

seguros y establecer tiempos de expiración adecuados son prácticas recomendadas en esta área.

2.1.5. Herramientas para el refuerzo de la Seguridad en el Desarrollo Web.

- **Burp Suite:** Es una herramienta integral para la prueba de penetración de aplicaciones web, utilizada para identificar vulnerabilidades como inyecciones SQL y XSS.

- **OWASP ZAP**: Herramienta de código abierto para la seguridad de aplicaciones web, permite identificar una amplia gama de vulnerabilidades.

- **Nikto:** Utilidad de código abierto que escanea aplicaciones web en busca de vulnerabilidades comunes y configuraciones inseguras en servidores web.

- **Acunetix:** Herramienta automatizada que escanea y detecta vulnerabilidades en aplicaciones web, incluyendo inyecciones SQL y problemas de seguridad en servidores web.

- **Qualys:** Ofrece soluciones de seguridad en la nube para escanear aplicaciones web y detectar vulnerabilidades, ayudando a mantener la seguridad y cumplimiento.

- **NIST, "Digital Identity Guidelines", NIST Special Publication 800-63B:** ofrece directrices detalladas sobre cómo implementar mecanismos de autenticación segura.

- **Cisco, "Annual Cybersecurity Report", Cisco Systems:** Este reporte destaca cómo las actualizaciones regulares pueden mitigar vulnerabilidades.

- **ISO/IEC, "Information technology — Security techniques — Information security incident management", ISO/IEC 27035:** proporciona un marco para el monitoreo y la respuesta a incidentes de seguridad informática.

"La importancia de utilizar herramientas para garantizar la calidad y seguridad de las aplicaciones web radica en la necesidad de prevenir ataques maliciosos y proteger la información confidencial de los usuarios. Además, el uso de herramientas de seguridad en el desarrollo web garantiza que las aplicaciones estén actualizadas y cumplan con los estándares de seguridad, lo que contribuye a la protección de los datos personales y sensibles de los usuarios." (PRACTICAS, 2023).

3. Planificación y Definición de Alcance para nuestro caso práctico de Estudio.

En línea con las fases del proyecto establecidas por la Metodología Ágil, tal y como se expuso en el apartado 1.4, avanzaremos hacia la definición detallada de las características de nuestro caso de estudio. Este proceso abarcará la clarificación de los objetivos, enfatizando los criterios de selección aplicados y precisando el alcance de

nuestro análisis. Además, procederemos a identificar meticulosamente los requisitos necesarios y a desarrollar unos algoritmos propios específicamente diseñado para reforzar la robustez de nuestro caso práctico.

3.1. Objetivos.

En esta etapa inicial, el objetivo se dirige hacia el diseño de algoritmos robustos para aplicaciones web, destinados a guiar a los desarrolladores en la implementación segura a lo largo de todas las capas, tanto en el Frontend como en el Backend. El objetivo es garantizar que las aplicaciones no solo satisfagan los requisitos funcionales establecidos, sino que también integren desde sus cimientos las prácticas de seguridad más avanzadas y recomendadas, asegurando así su fortaleza y confiabilidad desde el principio de su desarrollo.

3.2. Selección de los patrones.

La diferencia principal entre un patrón de arquitectura de software y un patrón de diseño radica en su enfoque y nivel de abstracción; mientras que los patrones de arquitectura se centran en la estructura general del sistema y la distribución de responsabilidades a un alto nivel, los patrones de diseño abordan problemas más específicos a nivel de componentes individuales y están más orientados a los detalles de implementación.

De todos ellos, para nuestro caso práctico, **nos hemos basado en el patrón que separa el servidor del cliente, además del patrón MVC**, siendo el principal motivo de separar el frontend del Backend como práctica fundamental en la arquitectura de aplicaciones web por varias razones relacionadas con la seguridad, mantenibilidad y escalabilidad.

Esta separación de responsabilidades mejora la seguridad de las aplicaciones de varias maneras:

- **Minimiza la Superficie de Ataque**: Al separar el frontend del backend, se reduce la superficie de ataque de la aplicación. Esto se debe a que los atacantes tienen menos oportunidades de explotar vulnerabilidades que podrían surgir de una integración demasiado estrecha de la lógica de presentación y la lógica de negocio.

  “Integre verificaciones de viabilidad en cada capa de su aplicación (desde el frontend al backend)." (OWASP Top 10, 2021).

- **Permite la Implementación de Políticas de Seguridad Específicas:** La separación facilita la implementación de políticas de seguridad específicas para cada capa. Por ejemplo, se pueden aplicar políticas de **CORS** (Cross-Origin Resource Sharing) más estrictas en el backend para controlar desde qué orígenes se permiten las solicitudes, protegiendo así contra ataques de tipo CSRF (Cross-Site Request Forgery) o XSS (Cross-Site Scripting).

"El intercambio de recursos de origen cruzado (CORS, por sus siglas en inglés), es un mecanismo basado en cabeceras HTTP que permite a un servidor indicar cualquier dominio, esquema o puerto con un origen (en-US) distinto del suyo desde el que un navegador debería permitir la carga de recursos. CORS también se basa en un mecanismo por el cual los navegadores realizan una solicitud de "verificación previa" al servidor que aloja el recurso de origen cruzado, con el fin de comprobar que el servidor permitirá la solicitud real. En esa comprobación previa, el navegador envía cabeceras que indican el método HTTP y las cabeceras que se utilizarán en la solicitud real." (MDN Web Docs, 2023)

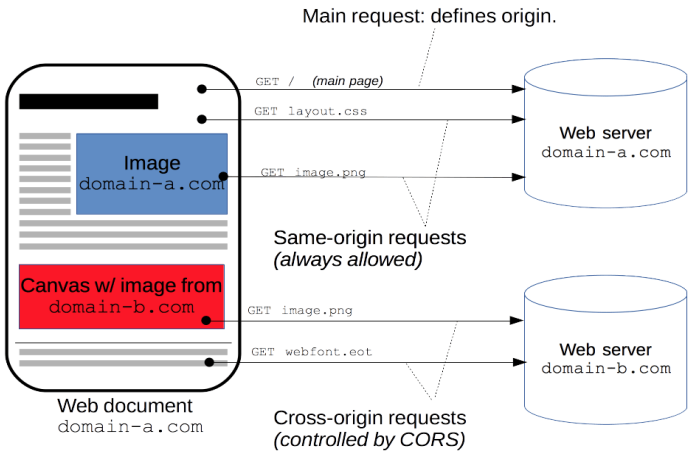

Ilustración 14: Intercambio de recursos de origen cruzado (CORS). SOURCE: https://developer.mozilla.org/es/docs/Web/HTTP/CORS047edf0b

- **Facilita la Autenticación y Autorización Seguras:** La separación clara entre el frontend y el backend permite implementar sistemas de autenticación y autorización más robustos. El backend puede manejar la autenticación y generar tokens de seguridad (como JWT - JSON Web Tokens) que el frontend utiliza para acceder a recursos protegidos.

  "Un JWT es un mecanismo para verificar el propietario de algunos datos JSON. Es una cadena codificada y segura para URL que puede contener una cantidad ilimitada de datos (a diferencia de una cookie) y está firmada criptográficamente.

  Cuando un servidor recibe un JWT, puede garantizar que los datos que contiene sean confiables porque están firmados por la fuente. Ningún intermediario puede modificar un JWT una vez enviado." (Ciberseguridad, 2021)

- **Mejora la Gestión de Dependencias y Actualizaciones:** Al mantener separados el frontend y el backend, es más sencillo gestionar dependencias y actualizar cada parte de la aplicación de manera independiente. Esto es crucial para la seguridad, ya que permite aplicar rápidamente parches y actualizaciones de seguridad sin afectar el resto de la aplicación.

  "Puedes realizar cambios en el frontend sin afectar el backend y viceversa. Esto facilita la adaptación a nuevas tecnologías y tendencias sin rehacer todo el sistema." (Arizbé Ken, 2023)

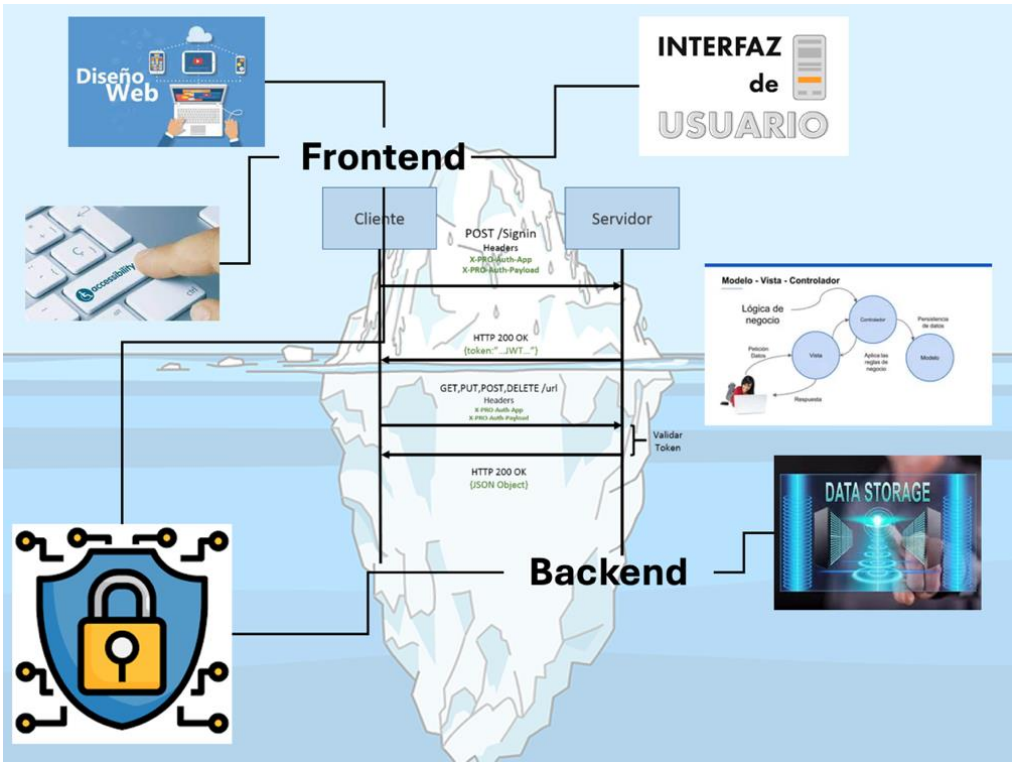

Ilustración 15: Caso de Estudio.

3.3. Alcance de nuestro análisis.

Para determinar el alcance de nuestro análisis de manera precisa, enfocaremos nuestra atención en los aspectos clave que componen la seguridad en el desarrollo de aplicaciones web. Esto implica una revisión exhaustiva de las legislaciones y normativas aplicables, asegurando el cumplimiento con las regulaciones de privacidad y seguridad. También incluye la evaluación de metodologías de seguridad informática reconocidas, **con un enfoque particular en la integración de los principios de la metodología OWASP**. Nuestro análisis se desarrollará siguiendo un enfoque estructurado, basado en la Metodología Ágil, que abarca desde la planificación inicial y la definición de los requisitos de seguridad, pasando por la implementación y las pruebas, hasta llegar a la evaluación y mejora continua.

En el siguiente apartado, el análisis profundizará en la identificación de riesgos de seguridad y en el diseño de estrategias de mitigación adecuadas, considerando tanto el Frontend como el Backend.

Se implementarán prácticas de codificación segura y se llevarán a cabo pruebas de seguridad, como pruebas de penetración, para identificar y corregir vulnerabilidades. Este proceso no solo cumple con los requisitos funcionales y de seguridad, sino que también promueve la responsabilidad social y ética, en consonancia con los objetivos de desarrollo sostenible, asegurando así una aplicación segura, confiable y legalmente cumplidora.

3.4. Caso práctico de Estudio, autentificación autorización, tokens.

3.4.1. Requisitos utilizados.

Para el desarrollo del **Backend**, se seleccionaron las siguientes tecnologías y herramientas:

- **Sistema Operativo:** Mac OS X, versión 14.4.1, operando en arquitectura x86_64.
- **Lenguaje de Programación:** Java, versión 21.0.2.
- **Entorno de Desarrollo Integrado (IDE):** Apache NetBeans, versión 21.

En cuanto al desarrollo del **Frontend**, se emplearon:

- **Sistema Operativo:** Mac OS X, versión 14.4.1, sobre arquitectura x86_64.
- **Framework:** React Native, optimizado para el desarrollo de aplicaciones móviles multiplataforma.
- **Entorno de Desarrollo Integrado (IDE):** Visual Studio Code, versión 1.87.2 (Universal), destacado por su flexibilidad y amplio soporte de extensiones para desarrollo web.

3.4.2. Gráfica modelo de Flujo de código de autorización.

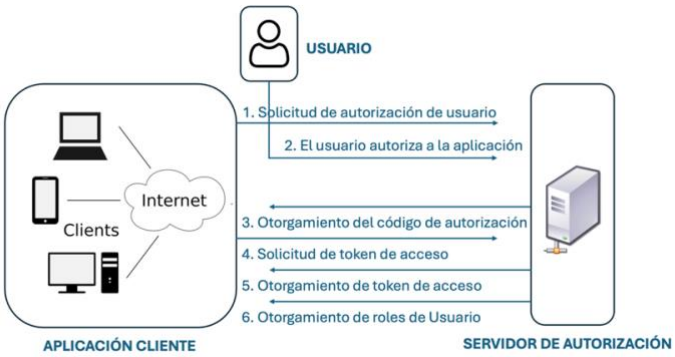

Ilustración 16: Flujo de código de autorización.

3.4.3. Caso de Uso.

Identificador caso de uso : Acceso Usuario
Actor principal : Usuario
Actor secundario : Aplicación Cliente
Nivel : General
Ámbito : Sistema

Escenario principal de éxito:

1. El Usuario envía sus credenciales.
2. La Aplicación encripta sus credenciales.
3. La Aplicación envía las credenciales al Servidor de Autorización.
4. La Aplicación solicita autentificarse al Servidor de Autorización.
5. El Servidor de Autorización autentifica al Usuario.
6. El Servidor de Autorización autoriza al Usuario.
7. El Usuario solicita token de acceso al Servidor de Autorización.
8. El Servidor de Autorización envía token de acceso al Usuario.
9. El Servidor de Autorización otorga roles de Usuario a la Aplicación.

10. El Usuario accede a la Aplicación.

Escenarios alternativos:
4.a. El Usuario no está registrado en el sistema.
4.a.1 Volver al paso1. Repetir hasta tres veces.
4.a.2 El Servidor de Autorización bloquea la URL.
4.a.3 El caso de uso termina.
4.b. El Usuario está registrado pero las credenciales no son válidas.
4.b.1 Volver al paso1. Repetir hasta tres veces.
4.b.2 El Servidor de Autorización bloquea la URL.
4.b.3 El Servidor de Autorización bloquea al Usuario.
4.b.4 El caso de uso termina.

3.4.4. Diagrama de actividad.

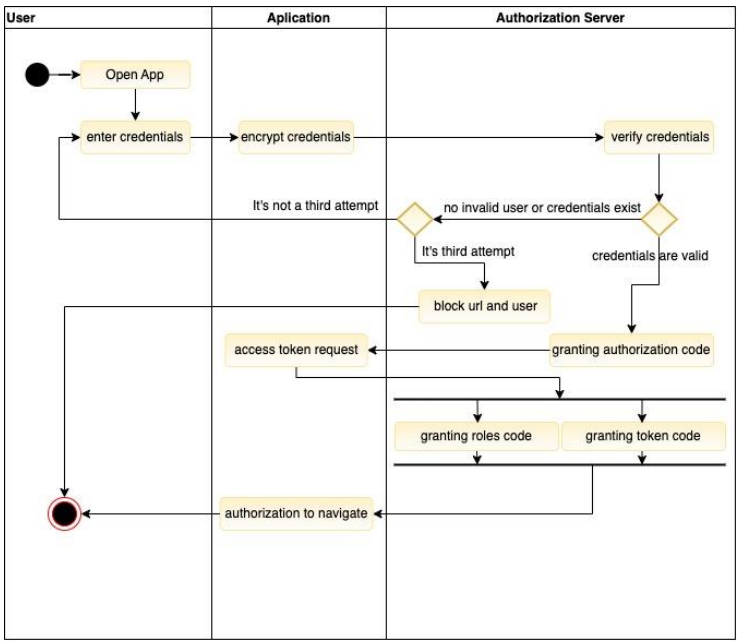

Ilustración 16: Diagrama de actividad.

3.4.5. Modelo de clases.

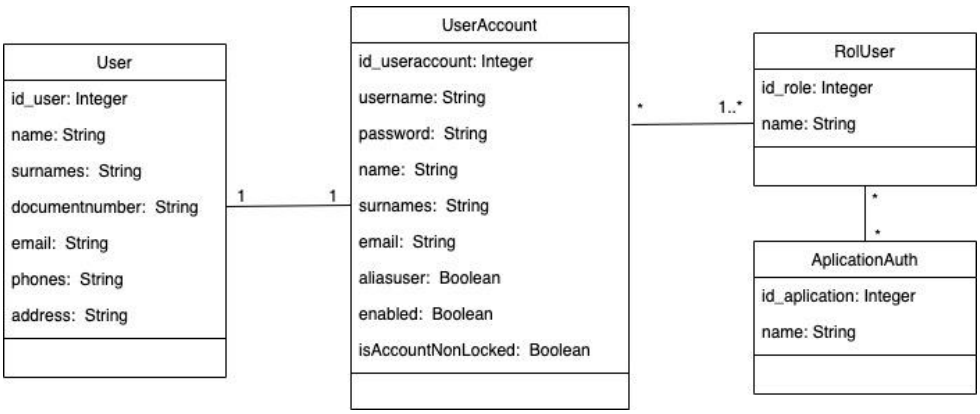

Ilustración 17: Modelo de clases.

3.4.6. Algoritmos Encriptación de datos e integración Spring Security para manejar la autenticación y autorización, además de utilizar JSON Web Tokens (JWT) para la gestión de sesiones sin estado.

El código presentado es una clase de utilidad en Java llamada PasswordUtil (Anexo 1: Clase "PasswordUtil"), que proporciona métodos estáticos para generar claves **AES** y vectores de inicialización (**IV**), así como para cifrar y descifrar datos utilizando el algoritmo **AES** en modo **CBC** con **padding PKCS5Padding**. Aquí se describe cada componente y su funcionalidad:

Clave AES y Vector de Inicialización (IV):

- AES (Advanced Encryption Standard) es un algoritmo de cifrado simétrico ampliamente utilizado. En este caso, se utiliza una clave de 256 bits, lo que ofrece un alto nivel de seguridad.
- IV (Vector de Inicialización) es un bloque de datos utilizado para introducir aleatoriedad en el cifrado. Tiene un tamaño de 16 bytes. Es importante que el IV sea único y aleatorio para cada operación de cifrado para garantizar la seguridad.

Métodos:

- **generateAESStrKey():** Genera una clave AES segura de 256 bits y la devuelve codificada en Base64. Esto facilita su almacenamiento y transmisión.
- **generateIv():** Genera un IV de forma segura y aleatoria y lo devuelve codificado en Base64.

Cifrado y Descifrado:

- **encryptAES(byte[] content, String secretKeyStr, String ivStr):** Cifra los datos proporcionados (content) utilizando AES en modo CBC con padding PKCS5Padding. Tanto la clave (secretKeyStr) como el IV (ivStr) se pasan como parámetros codificados en Base64. El método devuelve los datos cifrados codificados en Base64.
- **decryptAES(String content, String secretKeyStr, String ivStr):** Descifra los datos proporcionados (content), que deben estar cifrados y codificados en Base64, utilizando AES en modo CBC con padding PKCS5Padding. Al igual que en el método de cifrado, la clave y el IV se proporcionan codificados en Base64. Devuelve los datos descifrados en forma de String.

Consideraciones de Seguridad:

- El uso de AES con una clave de 256 bits en modo CBC es una práctica segura para el cifrado simétrico, ofreciendo un fuerte nivel de protección para los datos.
- La generación de IV aleatorios y el uso de claves seguras son fundamentales para la seguridad del cifrado.
- Es importante manejar y almacenar de manera segura las claves AES y los IV para evitar comprometer la seguridad de los datos cifrados.
- Este método proporciona una forma efectiva de cifrar y descifrar datos sensibles, como contraseñas o información personal, asegurando que solo aquellos con la clave AES correspondiente puedan acceder a los datos originales.

El diseño presentado es una configuración de seguridad para una aplicación **Spring Boot**, que integra Spring Security para manejar la autenticación y autorización, además de utilizar **JSON Web Tokens** (**JWT**) para la gestión de sesiones sin estado. A

continuación, se detalla la clase de utilidad en Java llamada MainSecurity (Anexo 2: Clase "MainSecurity")con el propósito y funcionamiento de sus componentes principales:

Configuración General:

- **@Configuration:** Indica que la clase es una fuente de definiciones de beans para el contexto de aplicación.
- **@EnableWebSecurity:** Habilita la seguridad web en Spring. Activa el soporte para Spring Security y proporciona la configuración de seguridad predeterminada.
- **@EnableGlobalMethodSecurity(prePostEnabled = true):** Permite la seguridad a nivel de método en Spring. Con prePostEnabled activado, se pueden usar anotaciones para controlar el acceso a métodos basado en la expresión.

Componentes Principales:

- **UserDetailsServiceImpl:** Un servicio personalizado que implementa UserDetailsService para cargar los datos del usuario (como la contraseña, roles, etc.) por su nombre de usuario.
- **JwtEntryPoint:** Un componente que se utiliza para manejar el error de autenticación como respuestas no autorizadas (HTTP 401) para proteger los puntos finales REST.
- **JwtTokenFilter:** Un filtro que se ejecuta por cada solicitud HTTP, se encarga de verificar la presencia y validez del JWT para permitir el acceso a los recursos protegidos.

Métodos y Configuraciones:

- **jwtTokenFilter():** Define un bean para el filtro de tokens JWT, que se encarga de la validación de estos tokens en cada solicitud.
- **passwordEncoder():** Proporciona un bean para el codificador de contraseñas, utilizando BCryptPasswordEncoder, que se utiliza para codificar las contraseñas en la base de datos y compararlas con las contraseñas proporcionadas durante el inicio de sesión.
- **configure(AuthenticationManagerBuilder auth):** Configura el AuthenticationManagerBuilder para usar el servicio userDetailsService personalizado para cargar los datos del usuario y el codificador de contraseñas para la autenticación.
- **authenticationManagerBean():** Sobrescribe el método para exponer el AuthenticationManager como un bean de Spring, permitiendo su uso en otras partes de la aplicación.
- **configure(HttpSecurity http):** Configura la seguridad a nivel de HTTP. En este método se desactivan CORS y CSRF por razones de compatibilidad y se configura el manejo de sesiones como sin estado (SessionCreationPolicy.STATELESS) debido al uso de JWT, que no requiere sesiones. Se especifica el manejo de excepciones para puntos de entrada no autenticados y se configura el filtro JWT para ejecutarse antes del filtro de autenticación de nombre de usuario y contraseña.

Consideraciones de Seguridad:

- Esta configuración establece las bases para una seguridad robusta en una aplicación web, utilizando JWT para la autenticación sin estado. Al no mantener estado, es ideal para aplicaciones escalables y distribuidas. La seguridad en el nivel de método permite un gran control sobre quién puede acceder a qué funcionalidades, mientras que la encriptación de contraseñas y la verificación de

tokens aseguran que solo los usuarios autorizados puedan acceder a recursos protegidos.

El método siguiente pertenece a una clase llamada JwtEntryPoint (Anexo 3: Clase "JwtEntryPoint") que implementa la interfaz AuthenticationEntryPoint de Spring Security. El propósito principal de este método es personalizar la respuesta que se envía al cliente cuando ocurre un error de autenticación, es decir, cuando un usuario no autorizado intenta acceder a un recurso protegido.

Configuración General:

- **Registro de Errores:** Utiliza el logger para registrar dos mensajes de error. El primero registra el mensaje de error de la excepción de autenticación (AuthenticationException) que se ha lanzado, y el segundo registra un mensaje indicando el intento no autorizado de acceso, incluyendo el método HTTP (GET, POST, etc.) y la URI solicitada.
- **Preparación de la Respuesta JSON:** Crea un mapa para estructurar la respuesta que se enviará al cliente. Este mapa incluye:
- **timestamp:** La marca de tiempo actual.
- **status:** El código de estado HTTP para no autorizado, que es 401.
- **error:** Una cadena que indica que la solicitud es "No autorizada".
- **message:** El mensaje de la excepción de autenticación.
- **path:** La URI que se intentó acceder.
- **Configuración de la Respuesta:** Establece el tipo de contenido de la respuesta a application/json;charset=UTF-8 y el estado de la respuesta a no autorizado (401). Además, añade encabezados de seguridad adicionales para mejorar la seguridad de la comunicación, tales como:
- **X-Content-Type-Options:** Con valor "nosniff" para evitar que el navegador intente inferir el MIME type que no coincide con el declarado en el encabezado Content-Type.
- **X-Frame-Options**: Con valor "DENY" para evitar que la página sea incluida en un <iframe>, lo cual puede ayudar a proteger contra ataques de clickjacking.
- **X-XSS-Protection:** Con valor "1; mode=block" para activar el filtro de protección XSS del navegador, el cual intentará sanitizar las páginas si detecta ataques de scripting entre sitios.
- **Envío de la Respuesta:** Finalmente, utiliza un objeto ObjectMapper para convertir el mapa de la respuesta en una cadena JSON y enviarla al cliente a través del OutputStream de la respuesta HTTP.

Este método personaliza la respuesta enviada a los clientes cuando intentan acceder a recursos protegidos sin estar autenticados, proporcionando información detallada sobre el error y mejorando la seguridad de la respuesta.

El método **generateToken**, **getNameUserFromToken** y **validateToken** forman parte de una clase llamada JwtProvider (Anexo 4: Clase "JwtProvider") dentro de un paquete que parece estar diseñado para la gestión de JSON Web Tokens (JWT) en una aplicación, para autenticación y autorización. Cada uno de estos métodos tiene un propósito específico en el ciclo de vida de un token JWT.

Se explica qué hace cada uno:

- **generateToken(Authentication authentication)** Este método genera un nuevo token JWT para un usuario autenticado. Utiliza la autenticación proporcionada por Spring Security (que representa al usuario autenticado) para construir y firmar el token.

- **getNameUserFromToken(String token)** Este método extrae y devuelve el nombre de usuario del sujeto del token JWT proporcionado:

- **validateToken(String token)** Este método valida el token JWT proporcionado para asegurarse de que es válido y no ha sido manipulado:

El método **doFilterInternal** en la clase JwtTokenFilter(Anexo 5: Clase "JwtTokenFilter"), que extiende **OncePerRequestFilter** de Spring Security, implementa un filtro que se ejecuta una vez por cada solicitud HTTP para realizar operaciones de autenticación basadas en tokens JWT (JSON Web Tokens).

Configuración General:

- **Extracción del Token:** El método primero intenta obtener el token JWT de la cabecera Authorization de la solicitud HTTP. Si la cabecera está presente y bien formada (comenzando con "Bearer "), extrae el token eliminando el prefijo "Bearer ".
- **Validación y Autenticación:** Si se encontró un token y es válido (jwtProvider.validateToken(token) retorna true), el método procede a extraer el nombre de usuario del token mediante jwtProvider.getNameUserFromToken(token). Con el nombre de usuario extraído del token, utiliza userDetailsService.loadUserByUsername(nombreUsuario) para cargar los detalles del usuario, que incluyen su nombre, contraseña (que no se usa en este contexto) y autoridades/roles. Luego, crea un UsernamePasswordAuthenticationToken con los detalles del usuario y sus autoridades. Este token de autenticación no tiene una credencial (contraseña) porque la autenticación ya fue verificada mediante el token JWT. Finalmente, establece este token de autenticación en el contexto de seguridad de Spring Security (SecurityContextHolder.getContext().setAuthentication(auth)), lo que efectivamente autentica al usuario para el contexto de la solicitud actual. Continuar con la Cadena de Filtros: Independientemente de si el token es válido o no, o si se encontró un token, el filtro pasa la solicitud y la respuesta al siguiente filtro en la cadena de filtros de Spring Security mediante filterChain.doFilter(req, res).

4. Análisis de Amenazas en interfaces de desarrollos Web.

Para estudiar los elementos y métodos de protección esenciales en la creación de desarrollos web, utilizables en diversos dispositivos como computadoras, smartphones o tablets, comenzaremos identificando las principales vulnerabilidades según el Top 10 de 2024 de la organización OWASP.

Ilustración 18: OWASP Top 10. SOURCE: https://www.owasptopten.org/the-release-of-the-owasp-top-10-2021

4.1. Acerca de OWASP.

El Proyecto Abierto de Seguridad en Aplicaciones Web (**OWASP**) es una comunidad abierta enfocada en permitir que las organizaciones creen, obtengan y mantengan aplicaciones y API seguras. Ofrece una variedad de recursos gratuitos y abiertos, incluyendo herramientas y estándares de seguridad, investigaciones de vanguardia, controles de seguridad estándar, bibliotecas, guías de revisión de seguridad en aplicaciones, materiales de capacitación como presentaciones y videos, hojas de ayuda sobre temas comunes, y organiza reuniones, eventos, entrenamientos y conferencias. Todos los recursos de OWASP están disponibles sin costo para quienes estén interesados en mejorar la seguridad de las aplicaciones. La organización aboga por abordar la seguridad de las aplicaciones como una combinación de problemas humanos, procesos y tecnología, promoviendo soluciones que mejoren estos tres aspectos. OWASP, al ser independiente de presiones comerciales, proporciona información imparcial y práctica sobre la seguridad de las aplicaciones. No está afiliada a ninguna empresa tecnológica, aunque apoya el uso adecuado de tecnologías de seguridad comerciales. Sus materiales se producen de manera colaborativa, transparente y abierta. La fundación OWASP, una entidad sin fines de lucro asegura el éxito continuo del proyecto, apoyando la investigación en seguridad con becas e infraestructura, con la mayoría de sus asociados actuando como voluntarios, (OWASP Top 10, 2021).

4.2. Cómo utilizar OWASP Top 10 como estándar.

El OWASP Top 10 es un documento diseñado para concienciar sobre los riesgos de seguridad en aplicaciones, y aunque se ha usado como un estándar de la industria AppSec desde 2003, es importante reconocer que sirve como un mínimo y un punto de partida, no como una solución completa. La aplicación del OWASP Top 10 como un

estándar enfrenta desafíos porque se centra en riesgos de seguridad más que en problemas concretos y fácilmente verificables. Por ejemplo, aspectos como el Diseño Inseguro o el monitoreo y registro efectivo, son difíciles de evaluar mediante pruebas convencionales, y su revisión podría requerir métodos como entrevistas o análisis de respuestas a incidentes, que no siempre son directamente verificables.

Estas son las recomendaciones sobre cuándo es apropiado utilizar el OWASP Top 10:

Caso de uso	OWASP Top 10 2021	Estándar de Verificación de Seguridad en Aplicaciones de OWASP (ASVS)
Concientización	Sí	
Capacitación	Nivel Introductorio	Completo
Diseño y arquitectura	Ocasionalmente	Sí
Estándar de codificación	Apenas Mínimo	Sí
Revisión de código seguro	Apenas Mínimo	Sí
Lista verificación para la revisión por pares	Apenas Mínimo	Sí
Pruebas unitarias	Ocasionalmente	Sí
Pruebas de integración	Ocasionalmente	Sí
Pruebas de penetración	Apenas Mínimo	Sí
Soporte de herramientas	Apenas Mínimo	Sí
Cadena de suministro segura	Ocasionalmente	Sí

Ilustración 19: ¿Cuándo es apropiado usar OWASP? SOURCE: https://owasp.org/Top10/es/A00_2021_How_to_use_the_OWASP_Top_10_as_a_standard/

Para aquellos interesados en adoptar un estándar de seguridad de aplicaciones más robusto y verificable, OWASP recomienda el Estándar de Verificación de Seguridad de Aplicaciones (ASVS), que está diseñado para ser utilizado en todas las fases del desarrollo de software y es adecuado para una verificación y prueba completas. El ASVS se presenta como la opción preferida para los proveedores de herramientas, ya que aborda de manera integral los riesgos que el OWASP Top 10 no puede cubrir completamente, desaconsejando cualquier afirmación de cobertura total del OWASP Top 10 debido a sus limitaciones inherentes.

4.3. OWASP Top 10 de 2021.

Este Top 10 fue creado hace dos décadas con el objetivo de examinar las principales amenazas en el desarrollo de aplicaciones web seguras. La versión más reciente se publicó en 2017 y no ha recibido actualizaciones desde entonces. Sin embargo, a partir del año 2021, se ha iniciado el trabajo en una nueva edición de este Top 10. Las modificaciones con respecto a la versión anterior se ilustran en la imagen siguiente:

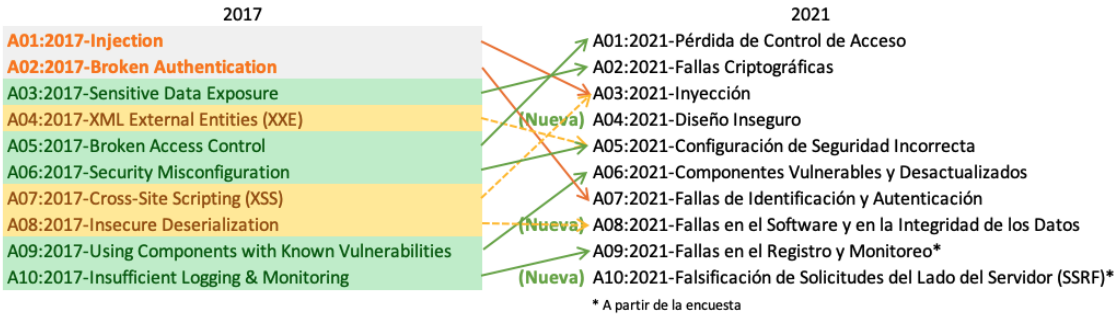

Ilustración 20: OWASP Top 10 de 2021. SOURCE: https://owasp.org/Top10/es/

Estos cambios en el OWASP Top 10 de 2021 subrayan un enfoque creciente en la prevención proactiva de vulnerabilidades, el diseño seguro, y la importancia de considerar la seguridad en todas las fases del desarrollo de aplicaciones.

4.3.1. A01:2021 - Pérdida de Control de Acceso.

Ahora ocupa el primer lugar, ascendiendo desde la quinta posición. Un promedio del 3,81% de las aplicaciones examinadas presentaban vulnerabilidades relacionadas, con más de 318,000 ocurrencias de problemas específicos, convirtiéndose en la categoría con mayor número de incidencias.

La implementación del control de acceso asegura que los usuarios actúen solo dentro de los límites de sus permisos asignados, previniendo así la divulgación no autorizada, modificación o destrucción de datos, o la ejecución indebida de funciones empresariales. Las vulnerabilidades típicas en el control de acceso incluyen:

- Infracción del principio de mínimo privilegio, que estipula que solo se debe otorgar acceso a aquellos con necesidades específicas, manteniendo el resto restringido.
- Omisión de controles de acceso mediante cambios en la URL, el estado de la aplicación o el código HTML, o utilizando herramientas para alterar solicitudes a APIs.
- Acceso a cuentas de terceros mediante el uso de identificadores únicos vulnerables.
- Uso de APIs sin controles de acceso adecuados para operaciones críticas.
- Elevación de privilegios, ya sea accediendo sin autenticación o asumiendo roles no correspondientes.
- Manipulación de metadatos, como alterar o reutilizar tokens de acceso para ganar privilegios no concedidos.
- Configuraciones erróneas de CORS que exponen APIs a orígenes no seguros.
- Navegación forzada hacia páginas restringidas sin la autenticación o autorización necesaria.

Para mitigar estos riesgos, es crucial:

- Aplicar controles de acceso en el servidor o a nivel de API, fuera del alcance de manipulaciones externas.
- Adoptar una política de denegación por defecto para recursos no públicos.

- Establecer mecanismos de control de acceso reutilizables y limitar el uso de CORS.
- Garantizar que los controles de acceso se apliquen a nivel de datos, restringiendo las operaciones que los usuarios pueden realizar.
- Hay que asegurar que los modelos de dominio reflejen las necesidades de negocio de la aplicación.
- Desactivar el listado de directorios en servidores web y proteger el acceso a archivos sensibles.
- Monitorear y registrar intentos fallidos de acceso, alertando a los administradores cuando sea pertinente.
- Limitar el número de solicitudes a APIs y controladores para reducir el impacto de ataques automatizados.
- Invalidar identificadores de sesión tras el cierre de esta y preferir el uso de tokens JWT de corta duración.
- Incluir pruebas de control de acceso en los procesos de desarrollo y aseguramiento de la calidad.

"**Ejemplos de escenarios de ataque**

Escenario #1: La aplicación utiliza datos no verificados en una llamada SQL que accede a información de una cuenta:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

Un atacante simplemente modifica el parámetro 'acct' en el navegador para enviar el número de cuenta que desee. Si no es verificado correctamente, el atacante puede acceder a la cuenta de cualquier usuario.

```
https://example.com/app/accountInfo?acct=notmyacct
```

Escenario #2: Un atacante simplemente navega a una URL específica. Se deberían requerir derechos de administrador para acceder a la página de administración.

```
https://example.com/app/getappInfo
https://example.com/app/admin_getappInfo
```

Si un usuario no autenticado puede acceder a cualquiera de las páginas, es una falla. Si una persona que no es administrador puede acceder a la página de administración, esto es también una falla." (OWASP Top 10, 2021).

4.3.2. A02:2021 - Fallas Criptográficas.

Sube a la segunda posición, anteriormente enfocada en la exposición de datos sensibles. Este cambio refleja un enfoque en las fallas de criptografía como causa raíz de exposiciones, en lugar de meramente características de vulnerabilidades.

Para asegurar la protección de datos sensibles tanto en tránsito como en reposo, es crucial identificar y clasificar estos datos según su necesidad de protección, especialmente si están regulados por leyes de privacidad como el GDPR o normativas como el PCI DSS. Las preocupaciones clave incluyen:

- La transmisión de datos en texto claro a través de protocolos no seguros.
- El uso de algoritmos criptográficos, claves o protocolos obsoletos o débiles.
- La inadecuada gestión o rotación de claves criptográficas.
- Fallos en forzar el cifrado o validar certificados y cadenas de confianza adecuadamente.
- Uso inseguro de vectores de inicialización o modos de operación criptográficos.
- Aplicación incorrecta de funciones hash o generadores de aleatoriedad.

Para prevenir vulnerabilidades y asegurar la protección de datos, se recomienda:

- Clasificar los datos para identificar aquellos que requieren protección especial.
- Evitar el almacenamiento innecesario de datos sensibles, optando por su eliminación rápida o el uso de técnicas como el tokenización o el truncamiento.
- Cifrar datos sensibles en reposo y en tránsito, empleando estándares y protocolos seguros y actualizados.
- Implementar una gestión de claves adecuada y asegurar la no utilización de protocolos antiguos para la transmisión de datos sensibles.
- Utilizar funciones de hashing robustas para almacenar contraseñas, aplicar cifrado autenticado y asegurar el uso adecuado de vectores de inicialización.
- Adoptar prácticas de seguridad criptográfica moderna, evitando funciones obsoletas y asegurando la aleatoriedad adecuada para la generación de claves.
- Siguiendo estas prácticas y consultando recursos como ASVS Crypto, Data Protection, y SSL/TLS, se puede fortalecer significativamente la protección de los datos y minimizar el riesgo de compromiso.

“Ejemplos de escenarios de ataque

Escenario #1: Una aplicación cifra los números de tarjetas de crédito en una base de datos mediante el cifrado automático de la base de datos. Sin embargo, estos datos se descifran automáticamente cuando se recuperan, lo que permite que por una falla de inyección SQL se recuperen números de tarjetas de crédito en texto sin cifrar.

Escenario #2: Un sitio no utiliza ni aplica TLS para todas sus páginas o admite un cifrado débil. Un atacante monitorea el tráfico de la red (por ejemplo, en una red inalámbrica insegura), degrada las conexiones de HTTPS a HTTP, intercepta solicitudes y roba la cookie de sesión del usuario. El atacante luego reutiliza esta cookie y secuestra la sesión (autenticada) del usuario, accediendo o modificando los datos privados del usuario. En lugar de lo anterior, podrían alterar todos los datos transportados, por ejemplo, el destinatario de una transferencia de dinero.

Escenario #3: La base de datos de contraseñas utiliza hashes simples o sin un valor inicial aleatorio único(salt) para almacenar todas las contraseñas. Una falla en la carga de archivos permite a un atacante recuperar la base de datos de contraseñas. Todos los hashes sin salt se pueden calcular a partir de una rainbow table de hashes pre calculados. Los hashes generados por funciones hash simples o rápidas pueden ser descifrados a través de cálculos intensivos provistos por una o más GPUs, incluso si utilizan un salt.” (OWASP Top 10, 2021).

4.3.3. A03:2021 – Inyección.

Incluyendo Cross-Site Scripting, baja al tercer lugar. A pesar de que el 94% de las aplicaciones fueron probadas para algún tipo de inyección, esta categoría sigue siendo altamente prevalente con el segundo mayor número de ocurrencias.

La inyección, ahora en el tercer lugar en términos de riesgos de seguridad, fue analizada en el 94% de las aplicaciones, mostrando una tasa de incidencia máxima del 19%, un promedio del 3% y 274.000 ocurrencias. Las categorías relevantes incluyen XSS (CWE-79), inyección SQL (CWE-89) y control externo de archivos (CWE-73). Las vulnerabilidades surgen cuando los datos del usuario no son adecuadamente validados o sanitizados, y se ejecutan consultas dinámicas sin la debida codificación de parámetros. Esto también se aplica al uso de datos en parámetros ORM para extracción indebida de registros o cuando se concatenan datos peligrosos en consultas o comandos.

Las inyecciones comunes abarcan SQL, NoSQL, comandos del sistema operativo, ORM, LDAP, y OGNL, siendo la revisión de código fuente el método más eficaz para detectar vulnerabilidades. Se recomienda la implementación de pruebas automatizadas en parámetros, encabezados, URL, y otros formatos de entrada, complementadas con herramientas de análisis estático (SAST), dinámico (DAST), o interactivo (IAST) integradas en los pipelines de CI/CD para identificar y rectificar fallos antes del despliegue.

Para prevenir inyecciones, es crucial separar los datos de comandos y consultas, preferentemente mediante el uso de APIs seguras que eviten intérpretes, interfaces parametrizadas, o herramientas ORM. Es importante tener en cuenta que los procedimientos almacenados pueden ser vulnerables si concatenan datos y consultas. Además, la validación de datos en el servidor mediante listas blancas y el escape de caracteres especiales en consultas dinámicas son prácticas recomendadas, aunque no infalibles, especialmente cuando los nombres de estructuras de SQL provienen del usuario. Finalmente, se sugiere el uso de LIMIT y otros controles SQL para minimizar el riesgo de exposición de datos en caso de inyección SQL.

"**Ejemplos de escenarios de ataque**

Escenario #1: Una aplicación usa datos no confiables en la construcción de la siguiente sentencia SQL vulnerable:

String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";

Escenario #2: Del mismo modo, la confianza total de una aplicación en frameworks puede resultar en consultas que siguen siendo vulnerables a inyecciones, (por ejemplo: Hibernate Query Language (HQL)):

Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");

En ambos casos, el atacante modifica el valor del parámetro "id" en su navegador y enviar por ejemplo: ' UNION SLEEP(10);--.

http://example.com/app/accountView?id=' UNION SELECT SLEEP(10);--

Esto modifica el significado de ambas consultas, retornando todos los registros de la tabla “accounts. Ataques más peligrosos podrían modificar datos o incluso invocar procedimientos almacenados.” (OWASP Top 10, 2021).

4.3.4. A04:2021 - Diseño Inseguro.

Es una nueva categoría, destacando la importancia de incorporar la seguridad desde las etapas iniciales del desarrollo. Subraya la necesidad de enfoques proactivos como la modelización de amenazas y el uso de patrones de diseño seguro.

El diseño inseguro abarca diversas debilidades relacionadas con la falta o ineficacia de controles de seguridad en el diseño de software, distinguiéndose claramente de las fallas de implementación. Estas debilidades no pueden rectificarse con una implementación impecable, dado que los controles necesarios para contrarrestar ataques específicos no existen desde la etapa de diseño. Contribuye a esta problemática la omisión de perfiles de riesgo empresarial durante el desarrollo del software, lo que lleva a una inadecuada planificación de los requerimientos de seguridad.

- **Gestión de Requerimientos y Recursos:** Es fundamental recopilar y negociar tanto los requerimientos de negocio como los técnicos, incluyendo aspectos de seguridad que afectan la confidencialidad, integridad, disponibilidad, y autenticidad de los datos y la lógica de negocio.

- **Diseño Seguro:** Adoptar una cultura y metodología de diseño seguro implica la evaluación constante de amenazas y asegurar que el software esté diseñado y probado para prevenir ataques conocidos. Esto incluye integrar el modelado de amenazas en el proceso de desarrollo y aprender de los errores para promover mejoras continuas.

- **Ciclo de Desarrollo Seguro (S-SDLC):** Implementar un S-SDLC que incluya patrones de diseño seguro, metodologías, bibliotecas de componentes seguros y modelado de amenazas. La colaboración con especialistas en seguridad es esencial a lo largo de todo el proyecto, incluida la fase de mantenimiento.

Prevención:

- Establecer y seguir un ciclo de desarrollo seguro con el apoyo de profesionales en seguridad de aplicaciones.
- Adoptar un catálogo de patrones de diseño seguros y componentes de "camino pavimentado".
- Aplicar modelado de amenazas a procesos críticos como autenticación, control de acceso y lógica de negocio.
- Integrar consideraciones y controles de seguridad desde la planificación hasta las pruebas, incluyendo pruebas unitarias y de integración para validar la resistencia frente a amenazas.

- Implementar una separación robusta de tenants y capas del sistema según las necesidades de exposición y protección, y limitar el consumo de recursos por usuario o servicio.
- Esta aproximación proactiva al diseño seguro busca prevenir vulnerabilidades desde el inicio, integrando la seguridad en cada etapa del ciclo de desarrollo del software.

"**Ejemplos de escenarios de ataque**

Escenario #1: Un flujo de trabajo de recuperación de credenciales puede incluir "preguntas y respuestas", lo cual está prohibido por NIST 800-63b, OWASP ASVS y OWASP Top 10. No se puede confiar en preguntas y respuestas como evidencia de identidad ya que más de una persona puede conocer las respuestas. Dicho código debe eliminarse y reemplazarse por un diseño más seguro.

Escenario #2: Una cadena de cines permite descuentos en la reserva de grupos y tiene un máximo de quince asistentes antes de solicitar un depósito. Los atacantes podrían modelar este flujo y probar si podían reservar seiscientos asientos en todos los cines a la vez utilizando unos pocos pedidos, lo que provocaría grandes pérdidas de ingresos.

Escenario #3: El sitio web de comercio electrónico de una cadena minorista no tiene protección contra bots administrados por revendedores que compran tarjetas de video de alta gama para revender sitios web de subastas. Esto crea una publicidad terrible para los fabricantes de tarjetas de video y los propietarios de cadenas minoristas y una mala sangre duradera con entusiastas que no pueden obtener estas tarjetas a ningún precio. El diseño cuidadoso de anti-automatización y las reglas de lógica de negocio, como compras realizadas a los pocos segundos de disponibilidad, pueden identificar compras no auténticas y rechazar dichas transacciones." (OWASP Top 10, 2021).

4.3.5. A05:2021 - Configuración de Seguridad Incorrecta.

Avanza al quinto lugar, subrayando los riesgos asociados con el software altamente configurable. La categoría resalta la importancia de una configuración segura y adecuada para prevenir vulnerabilidades.

Una aplicación puede ser vulnerable por varios motivos, como la falta de medidas de seguridad (**hardening**), funciones innecesarias activas, cuentas predeterminadas sin modificar, revelación de información a través del manejo de errores, deshabilitación de funciones de seguridad actualizadas, configuraciones inseguras en servidores o software, y la falta de un proceso de configuración de seguridad organizado y repetitivo, lo que incrementa el riesgo de ataques.

Para prevenir estas vulnerabilidades, es esencial implementar prácticas de instalación y configuración seguras que incluyan:

- **Hardening Seguro y Repetible:** Establecer un proceso de hardening que sea fácil de replicar para nuevos entornos, asegurando que el desarrollo, control de calidad y producción estén configurados de manera idéntica, pero con credenciales

distintas. Este proceso debe ser automatizado para reducir el esfuerzo de configurar nuevos entornos seguros.

- **Plataforma Mínima:** Mantener el sistema libre de funciones, componentes y documentación innecesarios, eliminando o no instalando características y marcos de trabajo no utilizados.
- **Gestión de Parches y Actualizaciones:** Incluir revisiones y actualizaciones de seguridad en el proceso de gestión de parches, asegurándose de revisar y actualizar configuraciones según sea necesario, incluyendo permisos en servicios de almacenamiento en la nube.
- **Arquitectura de Aplicación Segmentada:** Diseñar la arquitectura de la aplicación para que haya una separación efectiva y segura entre componentes o instancias, utilizando segmentación, contenedores o grupos de seguridad en la nube.
- **Directivas de Seguridad para Clientes:** Enviar directivas de seguridad, como encabezados de seguridad, a los clientes para fortalecer la protección.
- **Verificación Automatizada:** Implementar un proceso automatizado que verifique la efectividad de las configuraciones y ajustes de seguridad en todos los entornos.

"**Ejemplos de escenarios de ataque**

Escenario #1: El servidor de aplicaciones contiene aplicaciones de ejemplo que no se eliminan del servidor de producción. Estas aplicaciones de ejemplo poseen fallas de seguridad conocidas que los atacantes utilizan para comprometer el servidor. Supongamos que una de estas aplicaciones es la consola de administración y no se modificaron las cuentas predeterminadas. En ese caso, el atacante inicia sesión con las contraseñas predeterminadas y toma el control.

Escenario #2: El listado de directorios no se encuentra deshabilitado en el servidor. Un atacante descubre que simplemente puede enumerar directorios. El atacante detecta y descarga las clases Java compiladas, que decompila y aplica ingeniería inversa para ver el código. El atacante luego encuentra una falla severa de control de acceso en la aplicación.

Escenario #3: La configuración del servidor de aplicaciones permite que se retornen a los usuarios mensajes de error detallados, por ejemplo, trazas de pila(stack traces). Esto potencialmente expone información confidencial o fallas subyacentes, como versiones de componentes que se sabe son vulnerables.

Escenario #4: Un proveedor de servicios en la nube (CSP) posee permisos de uso compartido predeterminados abiertos a Internet a otros usuarios. Esto permite acceder a los datos confidenciales almacenados en el almacenamiento en la nube." (OWASP Top 10, 2021).

4.3.6. A06:2021 - Componentes Vulnerables y Desactualizados.

Se mueve desde la novena a una posición más prominente, enfatizando los desafíos en la evaluación de riesgos y la prueba de componentes con vulnerabilidades conocidas que proporciona métodos estáticos para generar claves AES.

Es probable que su aplicación sea vulnerable si:

- **Desconoce las Versiones de los Componentes:** Esto incluye tanto los componentes usados directamente como las dependencias anidadas, en el cliente y el servidor.
- **Software Desactualizado o Vulnerable:** Si los componentes como el sistema operativo, servidores web/aplicaciones, DBMS, aplicaciones, APIs, entornos de ejecución y bibliotecas están desactualizados o no tienen soporte.
- **Falta de Análisis de Vulnerabilidades:** Si no realiza análisis regulares en busca de vulnerabilidades ni sigue boletines de seguridad de los componentes que usa.
- **Retrasos en la Aplicación de Parches:** Si no actualiza o repara plataformas, frameworks y dependencias de manera oportuna, lo cual puede dejar a las organizaciones expuestas a vulnerabilidades por días o meses.
- **Falta de Pruebas de Compatibilidad:** Si los desarrolladores no prueban la compatibilidad con bibliotecas actualizadas o parcheadas.
- **Configuraciones de Componentes Inseguras:** Referencia a la sección A05:2021 sobre configuraciones de seguridad incorrectas.
  Prevención:

Para mitigar estas vulnerabilidades, se recomienda implementar un proceso de gestión de parches que:

- **Elimine Componentes Innecesarios:** Descarte dependencias, funcionalidades, componentes, archivos y documentación que no se utilizan.
- **Inventario Continuo**: Use herramientas para mantener un registro actualizado de todas las versiones de los componentes y sus dependencias. Monitoree fuentes como CVE y NVD para detectar vulnerabilidades.
- **Fuentes Seguras:** Asegúrese de obtener componentes solo de fuentes oficiales y preferiblemente paquetes firmados para evitar la inclusión de componentes maliciosos.
- **Vigilancia de Mantenimiento:** Preste atención a las bibliotecas y componentes que no se mantienen o no ofrecen parches de seguridad para versiones antiguas. Utilice parches virtuales si es necesario.
- **Plan de Actualizaciones Continuas:** Establezca un plan organizacional para el monitoreo, clasificación y aplicación de actualizaciones o cambios de configuración a lo largo del ciclo de vida de la aplicación.

"**Ejemplos de escenarios de ataque**

Escenario #1: Los componentes normalmente se ejecutan con los mismos privilegios que la propia aplicación, por lo que las fallas en cualquier componente pueden tener un impacto grave. Tales fallas pueden ser accidentales (por ejemplo, error de codificación) o intencionales (por ejemplo, una puerta trasera en un componente). Algunos ejemplos de vulnerabilidades de componentes explotables descubiertos son:

CVE-2017-5638, una vulnerabilidad de ejecución remota de código de Struts 2 que permite la ejecución arbitraria de código en el servidor, ha sido culpada de brechas importantes.

Si bien el Internet de las Cosas (IoT) es con frecuencia difícil o imposible de parchear, la importancia de parchearlo puede ser grande (por ejemplo, dispositivos biomédicos).” (OWASP Top 10, 2021).

4.3.7. A07:2021 - Fallas de Identificación y Autenticación.

Baja a la séptima posición, reflejando posiblemente una mejoría en la adopción de frameworks estandarizados que ayudan a mitigar estas vulnerabilidades.

La confirmación de la identidad, autenticación, y gestión de sesiones son críticas para la seguridad y para prevenir ataques de autenticación. Las vulnerabilidades en esta área pueden surgir si la aplicación:

- Permite ataques automatizados, incluyendo la reutilización de credenciales ya comprometidas.
- Facilita ataques de fuerza bruta o similares.
- Admite el uso de contraseñas predeterminadas, débiles o comunes.
- Tiene procesos ineficaces para la recuperación de contraseñas, como preguntas de seguridad predecibles.
- Almacena contraseñas en texto claro o con cifrado/hash débil.
- No implementa adecuadamente la autenticación multifactor.
- Expone el identificador de sesión en la URL.
- No renueva los identificadores de sesión tras el inicio de sesión.
- No invalida correctamente los identificadores de sesión tras el cierre de sesión o después de un periodo de inactividad.

Prevención:

- **Implementar Autenticación Multifactor:** Protege contra ataques de reutilización de credenciales, fuerza bruta, y robo de credenciales.
- **Eliminar Credenciales Predeterminadas:** Especialmente para cuentas de administrador.
- **Control de Contraseñas Débiles:** Verifique que las contraseñas nuevas no estén entre las más vulnerables o comunes.
- **Seguir Políticas de Contraseñas Basadas en Evidencia:** Como las recomendadas en la guía NIST 800-63b, ajustando la longitud, complejidad, y rotación de contraseñas.
- **Evitar la Enumeración de Usuarios:** Usando mensajes genéricos para evitar revelar información a través de las respuestas de registro o recuperación de credenciales.
- **Limitar Intentos de Inicio de Sesión:** Incrementar el tiempo de espera después de intentos fallidos, registrando y alertando sobre actividades sospechosas.
- **Gestión Segura de Sesiones:** Utilizar un gestor que genere nuevos ID de sesión con alta entropía tras el inicio de sesión, que no exponga los ID en URL, y que invalide las sesiones de forma segura después del cierre de sesión o inactividad.

“**Ejemplos de escenarios de ataque**

Escenario #1: Relleno de credenciales, el uso de listas de contraseñas conocidas es un ataque común. Supongamos que una aplicación no se implementa protección

automatizada de relleno de credenciales. En ese caso, la aplicación puede usarse como oráculo de contraseñas para determinar si las credenciales son válidas.

Escenario #2: La mayoría de los ataques de autenticación ocurren debido al uso de contraseñas como único factor. Las consideradas mejores prácticas de requerir de una rotación y complejidad de las contraseñas son vistos como alentadoras del uso y reúso de contraseñas débiles por parte de los usuarios. Se le recomienda a las organizaciones que detengan dichas prácticas y utilicen las prácticas recomendadas en la guía NIST 800-63 y utilicen autenticación multi-factor.

Escenario #3: Los tiempos de espera (timeouts) de las sesiones de aplicación no están configurados correctamente. Un usuario utiliza una computadora pública para acceder a una aplicación. En lugar de seleccionar "cerrar sesión", el usuario simplemente cierra la pestaña del navegador y se aleja. Un atacante usa el mismo navegador una hora más tarde, y el usuario continúa autenticado." (OWASP Top 10, 2021).

4.3.8. A08:2021 - Fallas en el Software y en la Integridad de los Datos.

Es otra nueva categoría que pone el foco en la seguridad de los procesos de actualización de software y la verificación de la integridad de los datos críticos.

Los fallos de integridad del software y de los datos surgen cuando el código y la infraestructura no están protegidos contra modificaciones no autorizadas. Estos fallos pueden ocurrir al depender de plugins, bibliotecas o módulos de fuentes no confiables, tener un pipeline CI/CD inseguro que permite accesos no autorizados o la inclusión de código malicioso, implementar actualizaciones automáticas sin verificaciones de integridad adecuadas, o al manejar objetos o datos de manera que puedan ser vistos y modificados por atacantes, llevando a una deserialización insegura.

Prevención:

- **Verificación de Fuentes a través de Firmas Digitales:** Use firmas digitales o mecanismos similares para asegurar que el software o los datos provienen de la fuente esperada y no han sido alterados.
- **Uso de Repositorios Confiados para Dependencias:** Asegúrese de que las bibliotecas y dependencias provengan de repositorios confiables, y considere la posibilidad de alojarlas en un repositorio interno después de un análisis previo si el riesgo es alto.
- **Herramientas de Análisis de Componentes:** Emplee herramientas como OWASP Dependency Check o OWASP CycloneDX para detectar vulnerabilidades conocidas en los componentes de terceros.
- **Revisión de Código y Configuraciones:** Implemente un proceso de revisión de cambios en el código y las configuraciones para evitar la introducción de elementos maliciosos en su pipeline.
- **Controles en el Pipeline CI/CD:** Garantice que su pipeline CI/CD tenga controles de acceso adecuados, segregación de funciones y configuraciones de seguridad para mantener la integridad del código durante el proceso de compilación y despliegue.
- **Protección de Datos Enviados a Clientes No Confiables:** No envíe datos sin cifrar o firmar a clientes no confiables sin mecanismos de verificación de

integridad o firmas electrónicas que puedan detectar modificaciones o reutilización de datos serializados.

"**Ejemplos de escenarios de ataque**

Escenario #1: Actualizaciones no firmadas: Muchos routers domésticos, decodificadores de televisión, firmware de dispositivos, entre otros, no verifican las firmas de sus actualizaciones de firmware. El firmware sin firmar es un objetivo creciente para los atacantes y se espera que empeore. Esto es una gran preocupación, ya que muchas veces no existe otro mecanismo para remediarlo que corregirlo en una versión futura y esperar a que las versiones anteriores caduquen.

Escenario #2: Actualización maliciosa de SolarWinds: Se sabe que los Estados-Naciones utilizan como vector de ataque los mecanismos de actualización, siendo un caso reciente de pública notoriedad el sufrido por SolarWinds Orion. La compañía que desarrolla el software poseía procesos seguros de construcción y mecanismos de integridad en sus actualizaciones. Sin embargo, estos fueron comprometidos y, durante varios meses, la firma distribuyó una actualización maliciosa a más de 18.000 organizaciones, de las cuales alrededor de un centenar se vieron afectadas. Se trata de una de las brechas de este tipo de mayor alcance y más importantes de la historia.

Escenario #3: Deserialización insegura: Una aplicación React utiliza un conjunto de microservicios implementados en Spring Boot. Tratándose de programadores funcionales, intentaron asegurarse de que su código fuera inmutable. La solución implementada consistió en serializar el estado de la sesión para el usuario y enviarlo entre los componentes con cada solicitud. Un atacante advierte el uso de un objeto Java serializado y codificado en base64 (identifica un string que comienza con "rO0") y utiliza la herramienta Java Serial Killer para obtener una ejecución remota de código en el servidor de aplicación." (OWASP Top 10, 2021).

4.3.9. A09:2021 - Fallas en el Registro y Monitoreo.

Escala posiciones, destacando la importancia de una adecuada visibilidad y respuesta ante incidentes de seguridad para el análisis forense y las alertas de incidentes.

El regreso del OWASP Top 10 en 2021 enfatiza la importancia de la detección, escalado, y respuesta ante brechas de seguridad, subrayando el papel crítico de los registros y el monitoreo. La falta de un monitoreo adecuado y respuesta activa se manifiesta en varios aspectos, como la no registro de eventos auditables, registros inadecuados o confusos de advertencias y errores, falta de monitoreo de las aplicaciones y APIs para detectar actividades sospechosas, almacenamiento local exclusivo de registros, implementación deficiente de umbrales de alerta y procesos de escalado, y la incapacidad para detectar y alertar sobre ataques en tiempo real.

Para contrarrestar estas deficiencias, se recomienda implementar una serie de controles, adecuados al riesgo de la aplicación:

- **Registro de Eventos Críticos:** Asegurar que los errores de inicio de sesión, control de acceso y validación de entradas del servidor se registren con el contexto

necesario para identificar actividades sospechosas, manteniendo estos registros el tiempo suficiente para análisis forenses.

- **Formato de Registros:** Generar registros en formatos que las herramientas de gestión de registros puedan procesar fácilmente.
- **Codificación de Datos de Registros:** Codificar correctamente los datos de registros para prevenir inyecciones o ataques en el sistema de monitoreo.
- **Trazabilidad de Transacciones de Alto Valor:** Mantener una traza de auditoría con controles de integridad que prevengan la modificación o eliminación de registros.
- **Monitoreo y Alertas por DevSecOps:** Los equipos de DevSecOps deben establecer sistemas de alerta y monitoreo para detectar y responder a actividades sospechosas de manera rápida.
- **Plan de Respuesta y Recuperación:** Adoptar un plan de respuesta ante incidentes, como el NIST 800-61r2 o más reciente.
- **Uso de Frameworks de Protección:** Implementar soluciones de protección de aplicaciones, tanto comerciales como de código abierto, como las reglas de ModSecurity de OWASP y sistemas de correlación de registros como ELK (Elasticsearch, Logstash, Kibana) con paneles personalizados y alertas.

"**Ejemplos de escenarios de ataque**

Escenario #1: El sitio web de un prestador de salud que provee un plan para niños no pudo detectar una brecha debido a la falta de monitoreo y registro. Alguien externo informó al prestador que un atacante había accedido y modificados registros médicos sensibles de más de 3,5 millones de niños. Una revisión post incidente detectó que los desarrolladores del sitio web no habían encontrado vulnerabilidades significativas. Como no hubo ni registro ni monitores del sistema, la brecha de datos pudo haber estado en proceso desde el 2013, por un período de más de 7 años.

Escenario #2: Una gran aerolínea India tuvo una brecha de seguridad que involucró a la pérdida de datos personales de millones de pasajeros por más de 10 años, incluyendo pasaportes y tarjetas de crédito. La brecha se produjo por un proveedor de servicios de almacenamiento en la nube, quien notificó a la aerolínea después de un tiempo.

Escenario #3: Una gran aerolínea Europea sufrió un incumplimiento de la GRPD que debe reportar. La causa de la brecha se debió a que un atacante explotó una vulnerabilidad en una aplicación de pago, obteniendo más de 400,000 registros de pagos de usuarios. La aerolínea fue multada con 20 millones de libras como resultado del regulador de privacidad." (OWASP Top 10, 2021).

4.3.10. A10:2021 - Falsificación de Solicitudes del Lado del Servidor.

Se añade al Top 10, reflejando su importancia según la comunidad de seguridad, a pesar de tener una baja tasa de incidencia en las pruebas realizadas.

Las fallas de SSRF (Server-Side Request Forgery) se producen cuando una aplicación web accede a un recurso remoto sin validar adecuadamente la URL suministrada por el usuario. Esto permite a un atacante forzar a la aplicación a enviar una solicitud falsificada a un destino no previsto, inclusive sitios protegidos detrás de

firewalls, VPNs o listas de control de acceso a la red (ACLs). Con las aplicaciones web modernas ofreciendo funciones que implican la búsqueda de URL por parte de los usuarios finales, la incidencia y gravedad de los ataques SSRF ha ido en aumento, especialmente por la adopción de servicios en la nube y la complejidad de las arquitecturas de sistemas.

Para mitigar los riesgos asociados con SSRF, se pueden implementar múltiples controles de defensa en profundidad:

Desde la Capa de Red:

- **Segmentación de Redes:** Aislar la funcionalidad que accede a recursos remotos en redes separadas para minimizar el impacto de un ataque SSRF.
- **Políticas de Firewall:** Implementar políticas de "denegar por defecto" en el firewall y reglas de ACL para restringir el tráfico a lo estrictamente necesario.

Desde la Capa de Aplicación:

- **Sanitización y Validación de Entradas:** Asegurar que todos los datos proporcionados por el usuario sean debidamente sanitizados y validados.
- **Restricción de URL, Puerto y Destino:** Utilizar listas positivas para permitir únicamente esquemas de URL, puertos y destinos autorizados.
- **Evitar Respuestas "Crudas":** No enviar a los clientes respuestas en formato "crudo" que puedan contener datos sensibles.
- **Deshabilitar Redirecciones HTTP:** Prevenir la posibilidad de que la aplicación realice redirecciones automáticas a URL no seguras.
- **Consistencia de URL:** Estar alerta a técnicas como el enlace de DNS y condiciones de carrera TOCTOU para evitar ataques que exploten estas vulnerabilidades.

Consejos Adicionales:

- Evitar mitigar SSRF solo con listas de denegación o expresiones regulares, ya que los atacantes pueden eludirlas fácilmente.
- No colocar servicios críticos para la seguridad en sistemas frontales y controlar el tráfico local para evitar abusos.
- Para frontends con requisitos de seguridad elevados y grupos de usuarios manejables, considerar el uso de cifrado de red (como VPNs) en sistemas dedicados.

“**Ejemplos de escenarios de ataque**

Los atacantes pueden usar SSRF para atacar sistemas protegidos detrás de firewalls de aplicaciones web, firewalls o ACLs de red, utilizando escenarios tales como:

Escenario #1: Escaneo de puertos de servidores internos – Si la arquitectura de red no se encuentra segmentada, los atacantes pueden trazar un mapa de las redes internas y determinar si los puertos están abiertos o cerrados en los servidores internos a partir

de los resultados de la conexión o del tiempo transcurrido para conectar o rechazar las conexiones de payload SSRF.

Escenario #2: Exposición de datos sensibles: los atacantes pueden acceder a archivos locales como servicios internos para obtener información confidencial como file:///etc/passwd y http://localhost:28017/.

Escenario #3: Acceso al almacenamiento de metadatos de los servicios en la nube: la mayoría de los proveedores de la nube tienen almacenamiento de metadatos como http://169.254.169.254/. Un atacante puede leer los metadatos para obtener información confidencial.

Escenario #4: Exposición de los servicios internos: el atacante puede abusar de los servicios internos para realizar más ataques, como la ejecución remota de código (RCE) o la denegación de servicio (DoS).” (OWASP Top 10, 2021).

5. Diseño, desarrollo de servicios y otros Mecanismos de Seguridad. Secure Software Development Lifecycle (SSDLC).

Con el tiempo, hemos comprendido que invertir en el control de calidad del software es más beneficioso que costoso. Sin embargo, la responsabilidad de esta tarea a menudo no se asigna claramente, lo que lleva a problemas persistentes. Aún más descuidado es el aspecto de la seguridad del software, un tema crítico que este texto busca resaltar y abordar.

“El criminal que se beneficie de la ausencia de seguridad no te va a dar feedback, no va a estar en el Daily Scrum, ni te va a avisar sobre qué falta para que tu producto esté completo. Va a preferir darte una sorpresa.” (Grupo Oesía, 2022).

La calidad y la seguridad son fundamentales en el software, pero difieren en su enfoque: mientras la calidad se centra en el rendimiento y la satisfacción del usuario, la seguridad protege datos críticos. La calidad se mide a través del feedback del usuario, pero la seguridad, siendo menos visible para el cliente, a menudo se pasa por alto. A pesar de esto, es vital no subestimar su importancia y gestionarla con la misma diligencia que cualquier otro aspecto del desarrollo de software, implementándola de manera ágil y progresiva.

¿Cómo empezamos?

Igual que la calidad, la seguridad es fundamental en cualquier producto. Invertir en ella desde el principio, siguiendo normas básicas, es beneficioso y nos introduce a este aspecto crucial.

Ilustración 21: Desarrollo de software seguro (SSDLC). SOURCE: https://grupooesia.com/insight/desarrollo-de-software-seguro-ese-gran-desconocido/

Integrar medidas de seguridad a lo largo del Ciclo de Vida del Desarrollo de Software Seguro (SSDLC) implica adoptar un enfoque proactivo y multifacético para proteger las aplicaciones.

5.1. Fase 1: Definición de Requisitos.

En este inicio, se detallan las necesidades para nuevas funcionalidades, recopilando perspectivas de todas las partes interesadas. Es crucial aquí esbozar los requerimientos de seguridad para garantizar que solo se acceda a la información pertinente por los usuarios autorizados.

Los casos de uso describen la funcionalidad de una aplicación y sus interacciones, ayudando a identificar actores, relaciones y secuencias de acciones. Los casos de mal uso o abuso complementan esto al revelar cómo podría explotarse maliciosamente la aplicación. Analizar estos escenarios permite identificar y documentar vulnerabilidades potenciales, así como las medidas necesarias para mitigar los riesgos de seguridad asociados, facilitando la definición de requisitos de seguridad críticos.

Ejemplo Práctico: Asegurar un sistema de autenticación:

- **Escenario Funcional:** Descripción del proceso de autenticación estándar, donde la aplicación valida las credenciales de los usuarios y muestra mensajes de error específicos si la autenticación falla.

- **Escenario Negativo:** Análisis de potenciales ataques, como los de fuerza bruta, destacando cómo los mensajes de error pueden revelar información útil para un atacante.

- **Casos de Uso y Abuso:** Comparación de los procesos de autenticación normales frente a los ataques para identificar medidas preventivas.

- **Obtención de Requerimientos de Seguridad:** Definición de criterios específicos de seguridad basados en los análisis anteriores, incluyendo:
    - Contraseñas alfanuméricas con mayúsculas, minúsculas y una longitud mínima de siete caracteres.
    - Bloqueo de cuentas tras tres intentos fallidos de acceso.
    - Uso de mensajes de error genéricos para no revelar información útil a los atacantes

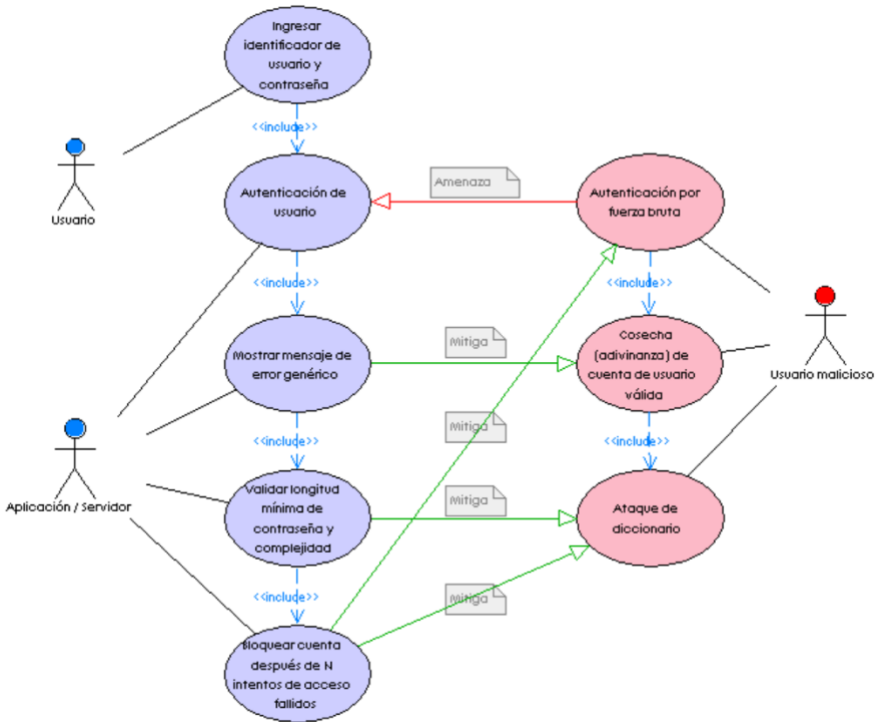

Ilustración 22: Definición de Requisitos (SSDLC). SOURCE: https://owasp.org/www-pdf-archive/Guía_de_pruebas_de_OWASP_ver_3.0.pdf

5.2. Fase 2: Diseños.

Esta etapa convierte los requisitos en un plan detallado para la aplicación. Mientras que los requisitos funcionales delinean las acciones deseadas, los requisitos de seguridad se enfocan en prevenir acciones no deseadas.

Durante esta etapa, se toman los requisitos tanto funcionales como no funcionales (incluyendo los de seguridad) y se elabora un plan detallado que guiará la construcción de la aplicación. Este plan incluye especificaciones de arquitectura, modelos de datos, diagramas de flujo, y otros documentos de diseño que ofrecen una vista completa de cómo deberá construirse el software para cumplir con esos requisitos.

Para el ejemplo práctico: **"Diseño Funcional: Mostrar en pantalla inicio de sesión"**, vamos a desglosarlo en términos de diseño, incluyendo aspectos funcionales y de seguridad:

Diseño Funcional:

El objetivo es crear una interfaz de usuario que permita a los usuarios iniciar sesión en una aplicación. Esto incluye varios elementos:

- **Interfaz de Usuario (UI):** Diseñar una pantalla de inicio de sesión que sea intuitiva y amigable para el usuario. Esto incluye campos para ingresar el nombre de usuario y la contraseña, y, un botón para realizar la acción de inicio de sesión, así como un enlace para recuperar la contraseña y una casilla para recordar las credenciales.

- **Flujo de Usuario:** Definir el proceso que sigue el usuario desde que ingresa sus credenciales hasta que se le concede o deniega el acceso basado en la verificación de sus credenciales. Esto incluye qué pasa si el usuario ingresa credenciales incorrectas (por ejemplo, mostrar un mensaje de error y permitir reintentar).

Requisitos de Seguridad:

En el contexto del inicio de sesión, los requisitos de seguridad son fundamentales para proteger la información de los usuarios y prevenir accesos no autorizados. Esto podría incluir:

- **Encriptación:** Asegurar que las contraseñas y otra información sensible se almacenan y transmiten de manera segura, utilizando encriptación.
- **Limitación de Intentos de Inicio de Sesión:** Para prevenir ataques de fuerza bruta, podría ser prudente limitar el número de intentos fallidos de inicio de sesión antes de bloquear temporalmente al usuario o requerir acciones adicionales (como verificación por correo electrónico o SMS).
- **Validación de Entradas:** Prevenir inyecciones SQL y otros ataques de inyección verificando las entradas de usuario antes de procesarlas.
- **Autenticación de Dos Factores (2FA):** Ofrecer o incluso requerir 2FA para añadir una capa adicional de seguridad al proceso de inicio de sesión.

Ejemplo del Caso Práctico Detallado:

UI/UX: Una pantalla de inicio de sesión con campos para usuario y contraseña, un botón de inicio de sesión, un enlace para "Olvidé mi contraseña" que dirige a los usuarios a un proceso de recuperación de contraseña, y una casilla para recordar credenciales. La interfaz debe ser limpia y fácil de usar en dispositivos móviles y de escritorio.

Seguridad: Implementar encriptación de extremo a extremo para las credenciales de inicio de sesión. Utilizar un mecanismo de tasa de límite para prevenir ataques de fuerza bruta, requiriendo CAPTCHA después de 3 intentos fallidos de inicio de sesión. Almacenar contraseñas utilizando un algoritmo de hashing seguro como el que vimos en apartados anteriores. Ofrecer 2FA mediante una aplicación de autenticación o SMS.

Flujo de Usuario: Después de ingresar las credenciales, el usuario presiona el botón de inicio de sesión. Si las credenciales son correctas y el usuario ha pasado las verificaciones de seguridad, se le redirige al Dashboard principal de la página de

desarrollo. Si las credenciales son incorrectas, se muestra un mensaje de error y se le permite intentar nuevamente hasta un límite de intentos, después del cual se requiere verificación adicional.

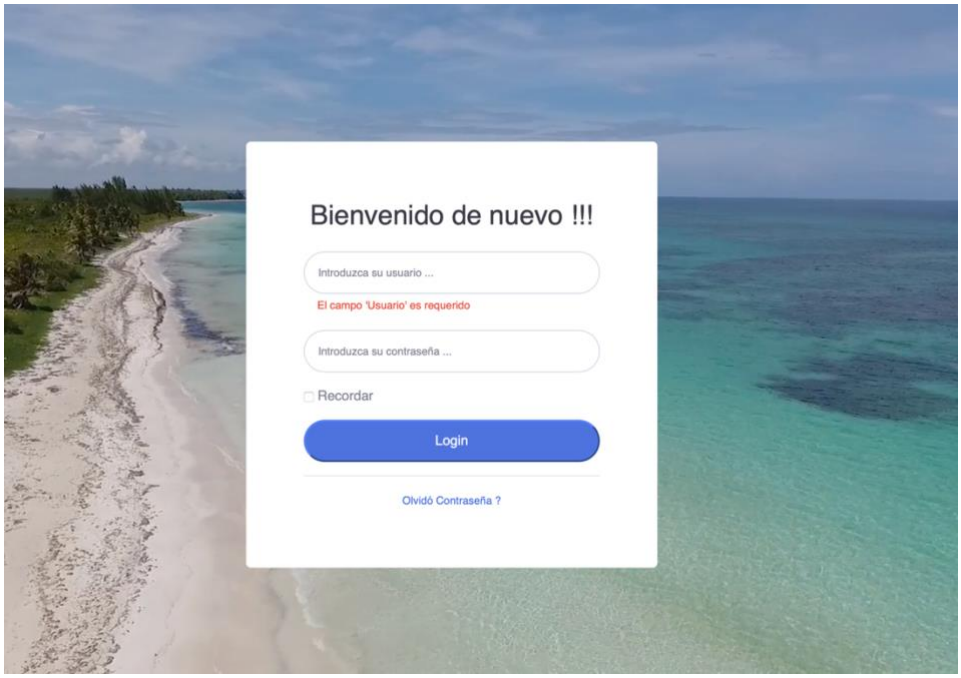

Ilustración 23: Diseños (SSDLC).

5.3. Fase 3: Desarrollo.

En el núcleo del desarrollo, la atención se centra en adherirse a prácticas de codificación seguras y realizar revisiones de código que aseguren el cumplimiento de estas prácticas. Dado que muchas aplicaciones modernas se construyen sobre componentes de código abierto, es vital examinar estas dependencias para vulnerabilidades.

Es recomendable que los desarrolladores creen casos de prueba específicos para la seguridad, integrándolos dentro del marco existente de pruebas unitarias como parte de un conjunto más amplio de evaluaciones de seguridad. Estos casos de prueba pueden basarse en escenarios previamente identificados de uso legítimo e ilegítimo, dirigidos a examinar la robustez de las funciones, métodos y clases frente a amenazas de seguridad. Dentro de este conjunto amplio de pruebas de seguridad, se deben incluir pruebas diseñadas para comprobar la eficacia de varios controles de seguridad, abarcando aspectos como:

- El control de acceso y los mecanismos de autenticación.
- La codificación y verificación de los datos de entrada.
- Los procesos de cifrado utilizados.
- La administración de sesiones y la gestión de usuarios.
- El manejo adecuado de errores y excepciones.
- Los procedimientos de auditoría y registro.

RECOMENDACIONES GENERALES:

- Utilizar el método GET de HTTP únicamente para consultar información y el método POST para enviar o intercambiar información.
- No confiar en las cabeceras HTTP para la validación o el envío de información sensible, ya que pueden ser manipuladas por atacantes.
- Hay que asegurar que todas las interacciones entre los componentes del entorno Web (servidores Web, de aplicación y bases de datos) se realicen de manera segura, con comunicaciones cifradas, autenticadas y con integridad garantizada.
- Almacenar toda información sensible, incluyendo la lógica de la aplicación y las credenciales de acceso, de forma cifrada en todos los servidores, especialmente en los servidores de bases de datos, para proteger contra accesos no autorizados.

FILTRADO DE DATOS DE ENTRADA DEL USUARIO:

Para mantener la seguridad web, es esencial ser constante en la implementación de medidas de seguridad, como el filtrado de datos proporcionados por usuarios. Esto ayuda a mitigar ataques como XSS e inyección SQL, al filtrar contenido malicioso tanto al recibir como al enviar datos. Existen dos enfoques principales para el filtrado: eliminar lo malicioso o permitir solo lo seguro, siendo este último el más seguro, pero más difícil de implementar sin un conocimiento completo de los datos válidos.

Se recomienda usar una librería centralizada de código para el filtrado de todos los datos de entrada, lo que asegura una aplicación uniforme de las medidas de seguridad y simplifica la gestión y actualización de estas medidas. Esto es crucial para prevenir una amplia gama de ataques web y mantener la integridad de la aplicación.

La biblioteca debe ofrecer métodos específicos para validar la entrada del usuario, considerando:

a) Tipo de Datos: Incluir funciones dedicadas a validar solo letras, números, combinaciones alfanuméricas, y datos más específicos como fechas, documentos de identidad, números telefónicos, entre otros.

b) Protección contra XSS: Filtrar cualquier entrada que incluya elementos HTML como <script>, <img>, <a href=...>, <object>, <iframe>, etc. Además, se recomienda eliminar o restringir caracteres utilizados en HTML como <, >, y sus diversas codificaciones (<, >, %3C, %3E, etc.) para prevenir la inyección de elementos HTML.

c) Prevención de Inyección SQL: Adaptar el filtro según el tipo de base de datos, excluyendo caracteres especiales y términos reservados de SQL como comentarios (--, #, /* */), punto y coma (;), comodines (*, %, _), operadores (OR, TRUE, SELECT, etc.), y sus representaciones codificadas (por ejemplo, %27 para ', %3D para =).

d) Evitar el Desplazamiento por Directorios: Impedir referencias a directorios superiores (..) y sus codificaciones, así como rutas absolutas usando / o \.

e) **Control de Referencias Directas a Archivos:** Restringir el acceso a archivos, evitando tratar URLs como rutas de archivo local.

f) **Limitación de Ejecución de Comandos del Sistema:** Prevenir el uso de caracteres especiales que puedan ejecutar comandos en el sistema (;, >, <, |, etc.).

g) **Protección contra HTTP Response Splitting:** Filtrar caracteres de salto de línea (\r, \n) que podrían permitir la inyección de cabeceras HTTP adicionales o datos en las cabeceras.

h) **Uso de Expresiones Regulares:** Cuando no existan funciones específicas, se pueden aplicar filtros mediante expresiones regulares para una validación adicional.

i) **Defensa contra Ataques Tradicionales:** Incluir la verificación de la longitud de los datos de entrada para evitar desbordamientos de buffer y otros ataques comunes.

El filtrado de datos de usuario debe hacerse tanto en el cliente como en el servidor para una seguridad óptima. Si solo se puede escoger uno, siempre debe ser en el servidor, debido a que las validaciones del lado del cliente pueden ser alteradas por atacantes. Dada la variedad de formas en que los datos pueden ser codificados (ASCII, codificación URL en hexadecimal, Unicode, etc.), y los intentos de evasión mediante distintas técnicas de codificación, es crucial normalizar primero los datos a un formato estándar antes de filtrarlos, asegurando que la validación sea efectiva contra diversas tácticas de ataque.

El uso de **PreparedStatement** en Java es una práctica recomendada para prevenir ataques de inyección SQL, ya que permite que la base de datos distinga claramente entre el código SQL y los datos proporcionados por el usuario.

Además de los algoritmos de Encriptación de datos e integración Spring Security para manejar la autenticación y autorización, además de utilizar JSON Web Tokens (JWT) para la gestión de sesiones sin estado, que vimos en el apartado 3.4.6 se añaden los siguientes algoritmos de seguridad para nuestro caso práctico:

5.3.1. Front-End (Lado del Cliente), para nuestro caso práctico.

Ejemplo de implementación de utilidades de cifrado y descifrado AES usando las bibliotecas crypto-js y uuid en React-Native (Anexo 6: “Cifrado y Descifrado AES con React-Native”).

Importaciones y Dependencias:

- Se importan **AESKEY y IV** (Vector de Inicialización) desde "../../types/Types", asumiendo que estos son constantes predefinidas en un módulo separado que proporciona la **clave de cifrado AES y el vector de inicialización que coinciden con el lado del servidor.**
- La biblioteca **crypto-js** es necesaria para realizar operaciones criptográficas como el cifrado y descifrado AES.

- La biblioteca **uuid** se utiliza para generar identificadores únicos, que aquí se usan para generar dinámicamente claves **AES y IVs.**
- **Objeto AESUtil:**
  - Este objeto encapsula las utilidades de cifrado y descifrado AES.
- **Función encrypt:**
  - Esta función cifra un contenido dado (_content) utilizando cifrado AES (en modo CBC con padding PKCS7).
  - El contenido, la clave AES (AESKEY) y el vector de inicialización (IV) primero se convierten al formato codificado UTF-8.
  - El contenido cifrado luego se convierte a una cadena codificada en Base64 antes de ser devuelto.
- **Función decrypt:**
  - Esta función descifra un contenido dado que ha sido cifrado con AES y codificado en Base64 (_content) de vuelta a su forma original.
  - Similar al cifrado, utiliza la clave AES (AESKEY) y el IV, convirtiéndolos al formato UTF-8.
  - El contenido descifrado se convierte de nuevo a una cadena UTF-8.
- **Función getAesKey:**
  - Genera una clave AES de 32 bytes usando un UUID (generado con UUID.v1()) como base.
  - El UUID primero se convierte al formato codificado UTF-8, luego a Base64, y finalmente, se toma una subcadena para asegurar que la clave tenga 32 bytes de longitud.
- **Función getIv:**
  - Similar a getAesKey, genera un vector de inicialización de 16 bytes usando un UUID.
  - Después de convertir el UUID a UTF-8 y luego a Base64, se toma una subcadena de la longitud requerida (16 bytes).

Esta utilidad proporciona una interfaz simple para cifrar y descifrar datos utilizando AES con generación dinámica de clave e IV. Está estructurada para ser fácilmente integrada en proyectos que requieran cifrado de datos para seguridad.

Ejemplo de cómo implementar una capa adicional de seguridad en aplicaciones web, cifrando las credenciales del usuario antes de transmitirlas, lo cual es particularmente útil en escenarios donde la integridad y la confidencialidad de los datos son críticos.

La función loginServiceEncrypted (Anexo 6A: Función "loginServiceEncrypted") es un servicio de autenticación que utiliza el método de cifrado AES para asegurar las credenciales del usuario antes de enviarlas a un servidor.

Parámetros de la Función:

- **ENDPOINT:** La URL base del servidor o API con la que se está comunicando. Este parámetro permite que la función se use con diferentes **endpoints** o puntos de acceso.
- **username:** El nombre de usuario que se está autenticando.
- **password:** La contraseña del usuario.
- **Proceso de Cifrado:**

- La función concatena el username y el password con un separador //. Este formato asegura que ambos elementos puedan ser claramente identificados y separados después del descifrado en el servidor.
- Utiliza la función AESUtil.encrypt (descrita en el código anterior) para cifrar la cadena resultante. Esto transforma las credenciales en un formato seguro que puede transmitirse a través de Internet sin exponer la información sensible del usuario.

- **Solicitud HTTP POST:**
  - La función realiza una solicitud HTTP POST al servidor, concatenando el ENDPOINT proporcionado con un path específico para el login (/auth/loginhead/) y el encrypted texto cifrado como parte de la URL.
  - Se establece el método de la solicitud como 'POST' y se configuran los encabezados para indicar que el contenido es de tipo 'application/json' y el charset es 'UTF-8'. Estos encabezados aseguran que el servidor interprete correctamente el formato y la codificación de la solicitud.
- **Envío y Respuesta:**
  - La función fetch es utilizada para enviar la solicitud al servidor. fetch es una API del navegador para realizar solicitudes HTTP de manera asincrónica, lo que significa que la función puede enviar la solicitud y manejar la respuesta (como la confirmación del login o un mensaje de error) sin bloquear la ejecución del código.
  - La función devuelve el resultado de la llamada fetch, que es una promesa que se resuelve con la respuesta del servidor. Esto permite que quien llame a loginServiceEncrypted maneje la respuesta (por ejemplo, procesando un **token** de acceso o mostrando un mensaje de error) utilizando .then() y .catch() o async/await.

5.3.2. Back-End (Lado del Servidor), para nuestro caso práctico.

Autenticación y generación de tokens JWT en aplicaciones web, proporcionando una manera segura y eficiente de gestionar sesiones de usuario y permisos sin necesidad de mantener un estado de sesión en el servidor.

La función descrita es parte de un controlador REST denominado AuthController (Anexo 7: Clase "AuthController"), diseñado para gestionar la autenticación de usuarios dentro de una aplicación. Este controlador se caracteriza por:

- **@RestController**, lo que indica que sus métodos manejan solicitudes web y devuelven datos en lugar de una vista.
- **@RequestMapping(value = "/auth")** para mapear todas las solicitudes dirigidas a /auth hacia los métodos definidos dentro de esta clase.
- Permitir solicitudes CORS de cualquier origen y cabecera mediante **@CrossOrigin**(origins = "*", allowedHeaders = "*"), facilitando la integración con otros servicios web que puedan estar alojados en dominios diferentes.
- **El método específico loginhead** se encarga de autenticar usuarios y generar un token JWT (JSON Web Token) para aquellos que logren autenticarse exitosamente. El proceso detallado es el siguiente:

- Recibe una cadena cifrada (param**)** a través de una solicitud **GET**, donde esta cadena contiene el nombre de usuario y la contraseña.
- La cadena cifrada es descifrada usando una clave **AES** y un vector de inicialización (**IV**), ambos obtenidos del entorno de la aplicación. Si el proceso de descifrada falla por cualquier razón (por ejemplo, argumentos inválidos), se devuelve una respuesta con el estado **HTTP UNAUTHORIZED**.
- La cadena descifrada se divide para obtener el nombre de usuario y la contraseña. Estos datos se utilizan para intentar autenticar al usuario mediante **AuthenticationManager**.
- Si la autenticación es exitosa, se genera un token **JWT** utilizando **JwtProvider**. Además, se obtiene información detallada del usuario autenticado (**UserDetails**).
- Se verifica si el token generado está presente y es válido. Si hay algún problema con el token, se devuelve un mensaje de error con el estado **HTTP BAD_REQUEST.**
- Se realiza una búsqueda adicional para obtener detalles completos del usuario a través de **userAccountAction**, basándose en el nombre de usuario.
- Si los detalles del usuario están disponibles, se crea y devuelve un objeto **JwtUserAccountResponseDTO** que incluye tanto **los detalles del usuario como el token JWT y los permisos del usuario, encapsulado en un ResponseEntity con estado HTTP OK.**
- En caso de errores durante la autenticación, se devuelve un mensaje de error con estado **UNAUTHORIZED**.

El código proporcionado modifica la configuración de seguridad para una aplicación Spring Boot, específicamente orientada a configurar aspectos relacionados con la autenticación y autorización mediante el uso de JSON Web Tokens (JWT).

La función descrita es parte de un controlador REST denominada MainSecurity (Anexo 8: Clase "MainSecurity"), diseñado para gestionar aspectos relacionados con la autenticación y autorización. Este controlador se caracteriza por:

- **Anotaciones:**
  - **@Configuration:** Indica que la clase tiene métodos que proporcionan instancias de beans que deberían ser gestionadas por el contenedor Spring.
  - **@EnableWebSecurity:** Habilita la seguridad web en Spring. Es parte de Spring Security y permite tener control sobre la seguridad a nivel de las solicitudes HTTP.
  - **@EnableGlobalMethodSecurity(prePostEnabled = true):** Habilita la seguridad a nivel de métodos con soporte para anotaciones de seguridad previas y posteriores a la invocación de un método. Por ejemplo, puede restringir el acceso a métodos específicos basado en la autoridad del usuario.
- **Componentes Autowired**
  - **UserDetailsServiceImpl:** Un servicio que implementa UserDetailsService, proporcionando la lógica para cargar los datos del usuario (como las autoridades) para la autenticación.
  - **JwtEntryPoint:** Un componente que maneja el punto de entrada de autenticación, usado para devolver una respuesta de error a las solicitudes no autenticadas.

- **Métodos Bean**
  - **jwtTokenFilter():** Proporciona una instancia del filtro JWT que se utiliza para interceptar y examinar las solicitudes HTTP para validar los tokens JWT.
  - **passwordEncoder():** Define el codificador de contraseñas a utilizar en la aplicación, en este caso, BCryptPasswordEncoder, que es una forma fuerte de hash para contraseñas.
- **Configuración de Autenticación**
  - En el método **configure(AuthenticationManagerBuilder auth)**, se especifica que la autenticación se realizará utilizando el servicio userDetailsService con un codificador de contraseñas especificado (passwordEncoder()).
- **Gestor de Autenticación**
  - **El método authenticationManagerBean()** sobreescrito permite exponer el AuthenticationManager como un Bean de Spring, facilitando su inyección y uso en otras partes de la aplicación.

- **Configuración HTTP:**
- En **configure(HttpSecurity http)**, se establece la configuración de seguridad a nivel de HTTP:
  - **Se habilita CORS y se deshabilita CSRF**, lo que es común en APIs que son consumidas por clientes de diferentes dominios.
  - Se configura el manejo de autorizaciones de solicitudes, permitiendo **acceso sin autenticación a ciertos endpoints** (por ejemplo, /auth/**) y requiriendo autenticación para el resto.
  - Se establece un manejador de excepciones para los puntos de entrada de autenticación, usando el **JwtEntryPoint**.
  - Se define la política de creación de sesiones como **STATELESS**, lo que es adecuado para APIs REST donde no se desea mantener el estado del usuario en sesión.
  - Finalmente, se agrega el filtro **JWT** antes del filtro de autenticación de nombre de usuario y contraseña (UsernamePasswordAuthenticationFilter.class), asegurando que el token JWT se procese en cada solicitud.

5.4. Fase 4: Verificación.

Tras la fase de desarrollo, de lleva a cabo una revisión minuciosa y detallada para garantizar la conformidad de la aplicación con todos los requisitos funcionales y no funcionales previamente definidos. Esta revisión abarca exhaustivamente tanto la lógica de negocio como los aspectos de seguridad, empleando una suite de pruebas automatizadas diseñada para validar cada componente crítico del sistema.

Los aspectos clave de esta estrategia incluyen:

- **Implementación de Pruebas Automatizadas para Rutas Críticas:** Se desarrolla y mantiene un conjunto de pruebas de integración y de sistema para evaluar las rutas críticas de la aplicación. Esto asegura no solo la funcionalidad esperada bajo condiciones normales de operación, sino también la robustez frente a casos de uso inesperados o malintencionados.

- **Verificación de la Exactitud de la Aplicación Mediante Pruebas Unitarias:** Se utiliza un enfoque de desarrollo guiado por pruebas (**TDD**, por sus siglas en

inglés) para construir pruebas unitarias que cubren cada unidad de código. Esto facilita la detección temprana de errores, promueve un diseño de software más limpio y mejora la mantenibilidad a largo plazo.

- **Evaluación Continua de la Seguridad de la Aplicación:** Se integran herramientas de análisis estático de código y pruebas de penetración automatizadas en el proceso de **CI/CD** para identificar vulnerabilidades de seguridad de manera proactiva. Estas herramientas examinan el código en busca de patrones conocidos de vulnerabilidades y simulan ataques contra la aplicación para detectar debilidades.

- **Monitorización y Registro en Tiempo Real:** Se establece un sistema de monitorización y registro en tiempo real que permite rastrear el comportamiento de la aplicación en producción, facilitando la detección temprana de cualquier anomalía o comportamiento inesperado que pueda indicar problemas de seguridad o errores de funcionamiento.

5.5. Fase 5: Mantenimiento y Evolución.

Tras el lanzamiento, el equipo mantiene un estado constante de alerta y vigilancia frente a vulnerabilidades emergentes, abarcando tanto el código desarrollado internamente como los componentes de terceros basados en código abierto.

Este enfoque proactivo se centra en dos frentes principales:

- **Monitoreo Continuo de Vulnerabilidades:** Se implementa un proceso sistemático para el monitoreo continuo de bases de datos de vulnerabilidades reconocidas, listas de distribución de seguridad, y fuentes de inteligencia sobre amenazas. Este proceso asegura que cualquier nueva vulnerabilidad identificada sea evaluada en relación con su aplicabilidad y potencial impacto en la aplicación.

- **Respuesta Rápida y Gestión de Parches de Seguridad:** Al detectarse vulnerabilidades que afecten directa o indirectamente al software, se prioriza la implementación de parches de seguridad o mitigaciones apropiadas. Las correcciones de seguridad se planifican meticulosamente para su integración en las próximas actualizaciones de mantenimiento o versiones mayores, dependiendo de la severidad y la urgencia de la vulnerabilidad. Además, se establece un canal de comunicación directo con los usuarios para informar sobre los riesgos de seguridad relevantes y las medidas correctivas adoptadas.

- **Evaluación y Actualización de Dependencias:** Se realiza una revisión periódica de las dependencias de código abierto para identificar aquellas que ya no se mantienen o que han quedado obsoletas. La actualización o reemplazo de estas dependencias es parte integral del mantenimiento continuo de la seguridad y la integridad del sistema.

- **Educación y Concienciación del Equipo:** Se promueve un programa de formación continua para el equipo de desarrollo, enfocado en las mejores prácticas de seguridad, las últimas técnicas de defensa, y la sensibilización sobre

las amenazas emergentes. Esto fomenta una cultura de seguridad que contribuye significativamente a la resiliencia general del software.

- **Revisiones de Seguridad Regulares:** Además de la respuesta a vulnerabilidades emergentes, se planifican revisiones de seguridad regulares del código y la arquitectura. Esto permite identificar y corregir proactivamente debilidades antes de que sean explotadas.

"La importancia de la seguridad en el SDLC

Uno de los problemas más comunes en el desarrollo de software es que la seguridad se aborda en una etapa demasiado avanzada del proceso: la de pruebas, después de haber completado las tareas más importantes de diseño e implementación. En muchos casos, los controles de seguridad que se ejecutan en esa etapa son superficiales, es decir, se limitan al análisis y las pruebas de intrusión. Por eso es posible que se pasen por alto problemas de seguridad más complejos que, de detectarse, podrían retrasar la llegada del sistema a la producción. Además, la resolución de los problemas lleva mucho tiempo y es más costosa, ya que puede requerir que se vuelva a desarrollar y probar todo el software." (Redhat, 2022).

6. Evaluación y desarrollo de una herramienta de auditoría de código fuente automatizada que utilice técnicas avanzadas de inteligencia artificial.

En la era digital actual, la seguridad y la eficiencia del código fuente son cruciales para el éxito de cualquier software. Con la proliferación de amenazas cibernéticas y la creciente complejidad de los sistemas de software, es más importante que nunca garantizar que el código no solo sea funcional, sino también seguro y optimizado. Este capítulo se dedica al desarrollo y evaluación de una herramienta de auditoría de código fuente automatizada, que incorpora técnicas avanzadas de inteligencia artificial (IA) para mejorar la detección de vulnerabilidades y errores en el código.

La necesidad de tal herramienta surge de la limitación inherente a las metodologías de auditoría manual, que son tanto tiempo-intensivas como propensas a errores humanos. Al integrar soluciones de IA, como el aprendizaje automático y el procesamiento del lenguaje natural, nuestra herramienta busca superar estas limitaciones, proporcionando un análisis más profundo y exhaustivo del código, de manera eficiente y precisa.

En este apartado, exploraremos la fundamentación teórica detrás de las técnicas de IA empleadas, seguido por el desarrollo de la herramienta propiamente dicha. Posteriormente, evaluaremos su eficacia mediante una serie de pruebas rigurosas, comparando su rendimiento con las soluciones de auditoría convencionales. Este análisis no solo destacará la superioridad de la automatización impulsada por IA en términos de precisión y eficiencia, sino que también demostrará su potencial para transformar las prácticas de seguridad del software en la industria.

La incorporación de la inteligencia artificial en la auditoría de código fuente automatizada abre un abanico de posibilidades para mejorar la detección y corrección de errores y vulnerabilidades. En esta sección, exploraremos las técnicas específicas de IA que se implementarán en nuestra herramienta, describiendo su funcionamiento y su importancia en el contexto de la auditoría de código. Consideraremos los fundamentos establecidos en los apartados anteriores para justificar su implementación.

6.1. Técnicas utilizadas para el desarrollo de la evaluación del software.

6.1.1. Aprendizaje Automático (Machine Learning).

El aprendizaje automático es una de las piedras angulares de nuestra herramienta, permitiendo que el sistema aprenda de los datos históricos de código para identificar patrones que podrían indicar errores o vulnerabilidades.

Utilizaremos dos enfoques principales:

Aprendizaje Supervisado: Mediante modelos entrenados con ejemplos previamente etiquetados de código bueno y malo, nuestra herramienta puede clasificar nuevas instancias de código y predecir su calidad o seguridad.

El libro de **Bishop** es una introducción exhaustiva a los campos del reconocimiento de patrones y el aprendizaje automático. La obra destaca por su enfoque en métodos bayesianos y modelos gráficos, que han adquirido importancia y se han desarrollado notablemente en la última década.

Uno de los puntos fuertes del texto es su tratamiento de los métodos de inferencia aproximada, como Bayes variacional y propagación de expectativas, que han ampliado la aplicabilidad práctica de los métodos bayesianos. Además, se abordan modelos basados en **Kernels**, que han impactado significativamente tanto en algoritmos como en aplicaciones prácticas.

"La comparación de modelos bayesianos implica el uso de probabilidades para representar la incertidumbre en la elección de un modelo, junto con una aplicación consistente de las reglas de suma y producto de la probabilidad" (Christopher M. Bishop, 2006).

Aprendizaje No Supervisado: Este enfoque se emplea para detectar anomalías en el código que difieren significativamente del estándar aceptado, lo que a menudo puede indicar errores no documentados o nuevas vulnerabilidades.

El libro "**Outlier Analysis**" explora a fondo el concepto y las técnicas de detección de anomalías, que es una aplicación crucial del aprendizaje no supervisado. En el aprendizaje no supervisado, no se tienen etiquetas predefinidas para los datos, y el objetivo es inferir la estructura subyacente de los datos a partir de los datos en sí. Este libro se enfoca específicamente en cómo identificar las observaciones que no se ajustan a un patrón esperado o típico, conocidas como **Outliers** o anomalías.

El autor detalla diversos métodos y algoritmos para la detección de **Outliers**, incluyendo enfoques estadísticos, basados en proximidad, y basados en densidad. Un

aspecto importante discutido en el libro es cómo los diferentes enfoques pueden ser aplicables dependiendo del tipo de datos y del contexto específico, ya sea para detección en bases de datos grandes, series temporales, o redes complejas.

"En muchas formas de aprendizaje predictivo, como la clasificación y la recomendación, existe una dicotomía natural entre los métodos de aprendizaje basados en instancias y los métodos de generalización explícita. Dado que los métodos de detección de valores atípicos requieren el diseño de un modelo de datos normales para poder hacer predicciones, esta dicotomía también se aplica al dominio no supervisado." (Charu C. Aggarwal, 2017).

6.1.2. Procesamiento de Lenguaje Natural (NPL).

El libro **"Natural Language Processing with Python**" proporciona una introducción completa al procesamiento de lenguaje natural (NLP) mediante el uso del Natural Language Toolkit (NLTK). Este enfoque práctico no solo enseña programación y técnicas fundamentales de NLP, sino que también subraya la importancia de integrar la teoría y la práctica. Las aplicaciones de NLP cubiertas van desde la **Tokenización** y el etiquetado hasta análisis más complejos como el **Parsing** y el manejo de grandes conjuntos de datos.

El libro enfatiza la manipulación y análisis de datos lingüísticos, presentando conceptos clave de **NLP** y lingüística para describir y analizar el lenguaje. Además, se profundiza en cómo se almacenan los datos de lenguaje en formatos estándar y cómo estos pueden ser utilizados para evaluar el rendimiento de las técnicas de NLP. Este enfoque práctico permite implementar ideas y verificar su utilidad práctica, facilitando una comprensión más profunda de los fundamentos teóricos del campo.

"En muchas formas de aprendizaje predictivo, como la clasificación y la recomendación, existe una dicotomía natural entre los métodos de aprendizaje basados en instancias y los métodos de generalización explícita. Dado que los métodos de detección de anomalías requieren el diseño de un modelo de los datos normales para hacer predicciones, esta dicotomía también se aplica al dominio no supervisado." (BIRD, S., KLEIN, E. y LOPER, E., 2009).

El procesamiento del lenguaje natural se utiliza para interpretar y analizar el código fuente, que es, en sí mismo, una forma de lenguaje. Utilizaremos técnicas de NLP para:

Análisis Sintáctico: Descomponer el código en sus componentes fundamentales para evaluar su estructura y lógica.

Análisis Semántico: Comprender el significado del código para asegurarnos de que las acciones que realiza son seguras y están dentro de las expectativas.

6.1.3. Redes Neuronales Profundas (Deep Learning Natural).

El libro "**Deep Learning. MIT Press**" proporciona una visión integral sobre las redes neuronales profundas, enfocándose en su capacidad para aprender representaciones de datos a múltiples niveles de abstracción. Los autores discuten cómo las redes

profundas, especialmente las redes neuronales de alimentación directa, redes convolucionales y redes neuronales recurrentes han revolucionado campos como la visión por computadora, el procesamiento del lenguaje natural y el reconocimiento de patrones.

Un tema central del libro es la capacidad de las redes neuronales profundas para realizar aprendizaje de representación, lo cual implica que estas redes aprenden automáticamente las características necesarias para la clasificación o predicción desde los datos brutos, eliminando la necesidad de ingeniería manual de características. Este aspecto es crucial para lograr altos niveles de rendimiento en tareas de aprendizaje automático complejas.

El libro también cubre técnicas avanzadas para entrenar estas redes, incluyendo aspectos como la regularización, inicialización de parámetros y optimización, que son esenciales para construir modelos de **Deep Learning** que no solo aprenden eficientemente, sino que también generalizan bien fuera de las muestras de entrenamiento.

Esta síntesis aborda desde los fundamentos teóricos hasta las aplicaciones prácticas, proporcionando así una base sólida para comprender cómo funcionan las redes neuronales profundas y cómo pueden ser aplicadas para resolver problemas del mundo real.

“Dropout, un algoritmo bastante simple de implementar proporciona una forma de combinar aproximadamente de manera eficiente una cantidad exponencialmente grande de arquitecturas de redes neuronales diferentes. El término 'dropout' se refiere a dejar fuera unidades (ocultas y visibles) en una red neuronal.” (Ian Goodfellow, Yoshua Bengio y Aaron Courville, 2023).

Las redes neuronales profundas son particularmente útiles para modelar problemas complejos en los que las relaciones entre los elementos son intrincadas y profundamente enlazadas. En el contexto de la auditoría de código fuente, utilizaremos:

Redes Neuronales Convolucionales (CNNs): Tradicionalmente utilizadas en el procesamiento de imágenes, las aplicamos para analizar patrones en bloques de código, identificando irregularidades estructurales.

Redes Neuronales Recurrentes (RNNs): Efectivas para trabajar con secuencias, como código, donde la comprensión del contexto anterior y posterior es crucial para la interpretación adecuada de su funcionalidad.

6.2. Proceso de Desarrollo del Software.

6.2.1. Definición de Requisitos.

Antes de iniciar el desarrollo, es crucial comprender completamente las necesidades y objetivos que la herramienta debe cumplir. Esto incluye la identificación de los tipos de vulnerabilidades y errores más comunes y críticos que la herramienta deberá detectar, así como las funcionalidades específicas deseadas por los usuarios finales. La colaboración con expertos en seguridad cibernética y desarrolladores de

software durante esta fase es esencial para asegurar una cobertura exhaustiva de los requisitos.

A) Identificación de Necesidades y Objetivos.

El primer paso en la definición de requisitos es realizar un análisis exhaustivo para comprender las necesidades específicas que la herramienta debe satisfacer. Esto implica:

Entrevistas con Stakeholders: Involucrar a desarrolladores de software, expertos en seguridad cibernética, y potenciales usuarios finales para obtener una visión completa de las expectativas y necesidades.

Análisis de Mercado: Examinar herramientas existentes para identificar deficiencias y oportunidades de mejora, asegurando que nuestra herramienta ofrezca soluciones novedosas y más eficientes.

B) Especificación de Requisitos Funcionales.

Una vez identificadas las necesidades, se procede a especificar los requisitos funcionales del sistema:

Detección de Vulnerabilidades: La herramienta debe ser capaz de identificar una amplia gama de vulnerabilidades conocidas y potenciales mediante técnicas de análisis estático y dinámico del código. La lista específica de las vulnerabilidades que la herramienta debe ser capaz de identificar, como inyecciones **SQL, cross-site scripting (XSS), desbordamientos de buffer, etc.**

Reporte de Errores y Sugerencias de Corrección: No solo debe detectar fallos o problemas, sino también proporcionar recomendaciones detalladas para la corrección de estos. Se debe definir si se utilizará análisis estático, dinámico o una combinación de ambos, y cómo estas técnicas se aplicarán en la herramienta.

Interfaz de Usuario: Especificaciones sobre cómo los usuarios interactuarán con la herramienta, incluyendo **Dashboards,** notificaciones, y opciones de configuración.

Integración con Entornos de Desarrollo (IDEs): Ser compatible con los principales entornos de desarrollo para facilitar su uso en el flujo de trabajo diario de los programadores. Detalles sobre cómo la herramienta se integrará con IDEs y otros sistemas de gestión de código fuente.

C) Requisitos No Funcionales.

Los requisitos no funcionales también son críticos para el éxito de la herramienta:

Rendimiento: Alta eficiencia en el procesamiento del código para minimizar el impacto en el tiempo de desarrollo. Parámetros claros sobre los tiempos de respuesta aceptables y la eficiencia en el procesamiento del código.

Escalabilidad: Capacidad para manejar proyectos de diferentes tamaños, desde pequeñas aplicaciones hasta grandes sistemas empresariales. Directrices sobre cómo la

herramienta debe escalarse en relación con el tamaño del proyecto y la cantidad de usuarios concurrentes.

Seguridad: Asegurar que la herramienta misma no introduce nuevas vulnerabilidades en el entorno en el que se implementa. Normativas específicas para asegurar que la herramienta opera de manera segura y no introduce vulnerabilidades adicionales.

Compatibilidad y Soporte de Plataformas: Especificaciones sobre las plataformas y entornos operativos soportados.

D) Documentación de Requisitos.

Finalmente, todos los requisitos recopilados se documentan de manera formal. Esto incluye la creación de:

Documentos de Especificación de Requisitos del Software (SRS): Que ofrecen una descripción detallada de las funcionalidades y restricciones de la herramienta. Este documento debe estar revisado y aprobado, conteniendo toda la información necesaria sobre funcionalidades y limitaciones técnicas.

Historias de Usuario y Casos de Uso: Para representar de manera clara y concisa lo que los usuarios esperan de la herramienta en escenarios de uso real. Detallar escenarios específicos en los que la herramienta será utilizada, incluyendo flujos de trabajo típicos y casos de uso extremos para asegurar que el diseño los contempla.

6.2.2. Diseño del Sistema.

Con los requisitos definidos, se procede al diseño de la arquitectura del sistema. Esta fase involucra la creación de un modelo de software que integra las diferentes técnicas de IA discutidas anteriormente, como el aprendizaje automático y el procesamiento del lenguaje natural. Se debe prestar atención especial a la escalabilidad y el modularidad del diseño, permitiendo futuras expansiones o modificaciones según evolucionen las necesidades de auditoría.

Avanzando al Diseño del Sistema para nuestra herramienta de auditoría de código fuente automatizada que utiliza técnicas de inteligencia artificial, esta fase es crucial para definir cómo se integrarán y se implementarán los componentes y las funcionalidades del sistema. Aquí detallo cómo podríamos proceder:

A) Diseño Arquitectónico.

El diseño arquitectónico de la herramienta es el primer paso para visualizar la estructura del sistema. Este diseño incluirá:

Modelo de Capas: Organizar la herramienta en capas lógicas (por ejemplo, presentación, lógica de negocio, acceso a datos), facilitando el modularidad y la escalabilidad.

- **Capa de Presentación:** Responsable de la interfaz de usuario y la interacción con el usuario final. Incluye la interfaz gráfica de usuario (GUI) y cualquier API o servicio web utilizado para la comunicación con clientes externos.

- **Capa de Lógica de Aplicación:** Contiene la lógica central de la aplicación. Se encarga de procesar las solicitudes del usuario, coordinar las acciones entre los diferentes componentes del sistema y aplicar las reglas de negocio. Incluye los algoritmos de análisis de código, la lógica de detección de vulnerabilidades y la generación de informes.

- **Capa de Acceso a Datos:** Encargada de interactuar con la base de datos y otros sistemas de almacenamiento de datos. Realiza operaciones de lectura y escritura en la base de datos, recuperando y almacenando la información necesaria para el análisis del código fuente y la generación de informes. Puede incluir una capa de acceso a datos específica para el almacenamiento y recuperación de modelos de inteligencia artificial entrenados.

- **Capa de Integración Externa:** Facilita la integración con otros sistemas externos, como sistemas de gestión de versiones (**VCS**), sistemas de seguimiento de problemas (**issue tracking systems**) o servicios de terceros para análisis complementarios. Permite la importación y exportación de datos, así como la sincronización con sistemas externos para mantener la coherencia y la integridad de la información.

- **Capa de Infraestructura:** Proporciona los servicios y recursos subyacentes necesarios para que el sistema funcione correctamente. Incluye la configuración del servidor, la gestión de la red, la gestión de la seguridad y cualquier otro componente de infraestructura necesario para garantizar el funcionamiento del sistema.

Ilustración 24: Diseño del Sistema una herramienta de auditoría de código fuente automatizada con IA

Diagramas de Arquitectura: Crear diagramas UML (Unified Modeling Language) para representar los componentes del sistema y sus interacciones. Esto incluye diagramas de componentes, diagramas de clases y diagramas de secuencia que clarifican las relaciones y dependencias.

- **Diagrama de Componentes:** El diagrama de componentes muestra los componentes del sistema y las relaciones entre ellos.

  Interfaz de Usuario (UI): Este componente representa la interfaz gráfica de usuario a través de la cual los usuarios interactúan con la herramienta de auditoría de código fuente.

  Lógica de Aplicación: Este componente contiene la lógica central de la aplicación, incluyendo los algoritmos de análisis de código y detección de vulnerabilidades.

  Base de Datos (DB): Este componente almacena y gestiona la información relevante sobre proyectos, análisis de código y resultados de auditoría.

  Integración Externa: Este componente facilita la integración con sistemas externos, como sistemas de control de versiones (VCS) y servicios externos de análisis de código.

  Infraestructura: Este componente proporciona los servicios y recursos subyacentes necesarios para el funcionamiento del sistema, como servidores, redes y medidas de seguridad.

Ilustración 25: : Diagrama de Componentes de una herramienta de auditoría de código fuente automatizada con IA

- **Diagrama de Clases:** El diagrama de clases muestra las clases del sistema y las relaciones entre ellas.

  RoleUser: Almacena información sobre roles de acceso.
  UserAccount: Almacena información sobre los usuarios del sistema, incluyendo autenticación y roles de acceso.
  Developer: Almacena información sobre los desarrolladores de proyectos, incluyendo el tipo de acceso de usuario.
  Project: Contiene datos de los proyectos de código fuente que se analizan.
  ***Resource*:** Contiene datos de las clases o de los paquetes de los proyectos de código fuente que se analizan.
  Package: Contiene datos de los paquetes de los proyectos de código fuente que se analizan. Un paquete puede contener otros paquetes, pero no puede contenerse a sí mismo.
  Class: Contiene datos de las clases de los proyectos de código fuente que se analizan.

Análisis: Registra cada sesión de análisis realizada sobre un proyecto, incluyendo referencias a resultados específicos y configuraciones de análisis.
DataModel: Guarda información sobre los modelos de aprendizaje automático utilizados o generados durante los análisis. Utilizar técnicas de particionamiento y sharding para manejar el volumen y la velocidad de los datos ingresados y consultados.
Vulnerability: Contiene la información sobre las vulnerabilidades que se pueden producir en cualquier código fuente.
DataModel: Registra las vulnerabilidades que se han generado en el código fuente según el modelo de aprendizaje automático.

- **Diagrama de Secuencia:** El diagrama de secuencia muestra cómo interactúan los objetos del sistema en un escenario específico.

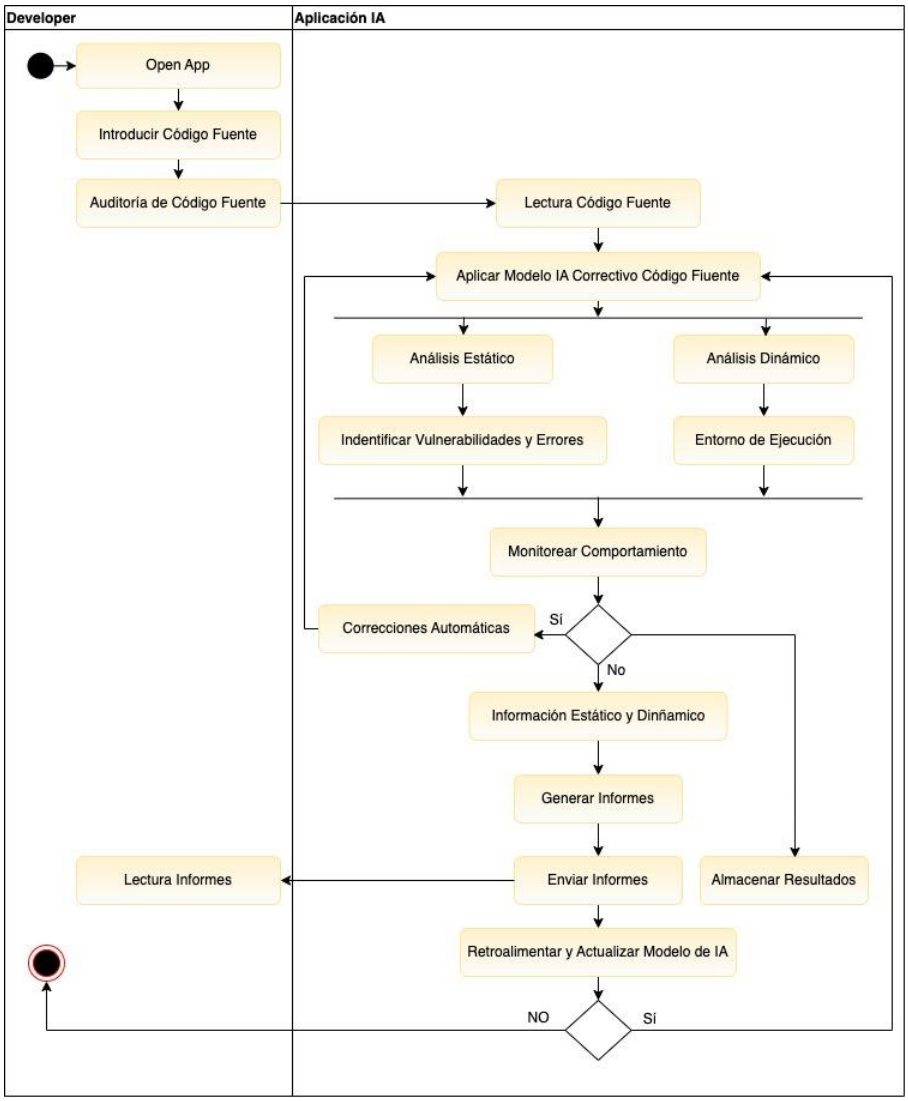

Ilustración 26: Diagrama de Secuencias de una herramienta de auditoría de código fuente automatizada con IA

Definición de APIs: Especificar las interfaces de programación de aplicaciones (APIs) para la comunicación entre los distintos módulos y servicios externos, asegurando la flexibilidad y la integración con otros sistemas.

Para el diseño de nuestras APIs, hemos optado por adoptar el enfoque RESTful, que se destaca por su simplicidad y por integrarse de manera fluida con la web y los protocolos HTTP existentes. Este modelo nos permite aprovechar métodos HTTP estándar (como GET, POST, PUT y DELETE) para facilitar la comunicación y la manipulación de recursos, lo que resulta en interfaces limpias y fáciles de usar.

En cuanto a las especificaciones de estas APIs, nos basaremos en los mecanismos de comunicación y autenticación detallados en el apartado anterior. Esto incluye la implementación de protocolos seguros para la autenticación de usuarios y la autorización de accesos, garantizando así la seguridad en el intercambio de información entre los diferentes módulos del sistema y con entidades externas.

B) Diseño de Base de Datos.

Modelo de Datos: Diseñar un esquema de base de datos que sea eficiente para las operaciones de consulta y actualización requeridas, y que soporte la estructura de datos necesaria para el aprendizaje automático y el almacenamiento de los resultados del análisis.

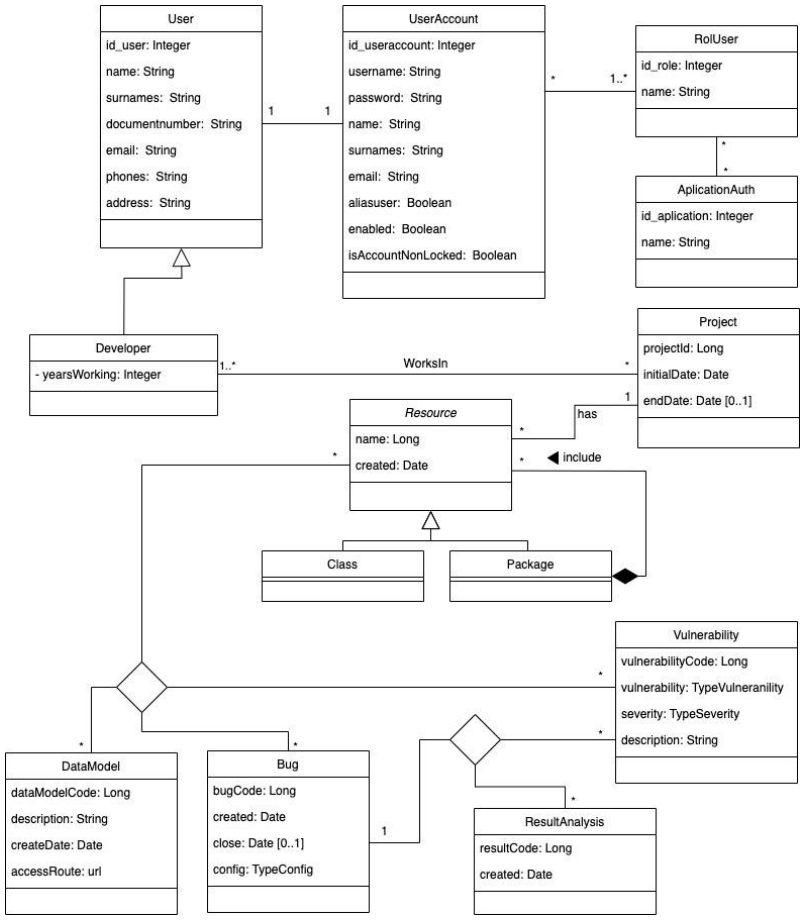

Ilustración 27: Diagrama de Clases UML de una herramienta de auditoría de código fuente automatizada con IA

C) Diseño de la Interfaz de Usuario.

La interfaz de usuario (UI) debe ser intuitiva y fácil de usar, permitiendo a los usuarios interactuar eficazmente con la herramienta:

Prototipos de UI: Desarrollar prototipos de la interfaz gráfica, que serán validados con usuarios para asegurar su usabilidad.

Diseño Responsivo: Asegurar que la UI sea accesible y eficiente en diferentes dispositivos y resoluciones.

D) Diseño de Seguridad.

Incorporar consideraciones de seguridad desde el inicio del diseño para garantizar la robustez del sistema:

Auditorías de Seguridad: Planificar auditorías regulares y pruebas de penetración para identificar y mitigar potenciales vulnerabilidades.

Mecanismos de Autenticación y Autorización: Establecer métodos seguros para la autenticación de usuarios y la autorización de acceso a diferentes partes del sistema, vistos con anterioridad.

E) Planificación de la Integración y Pruebas.

Establecer un plan para la integración de los componentes del sistema y las pruebas subsiguientes:

Estrategia de Pruebas: Definir un enfoque detallado para las pruebas unitarias, de integración, de sistema y de aceptación de usuario.

Herramientas de Integración Continua: Seleccionar y configurar herramientas para la integración y despliegue continuos (CI/CD) que facilitarán la integración regular de nuevas partes del software y su validación inmediata.

6.2.3. Implementación de Prototipos.

La fase de implementación comienza con la creación de prototipos. Estos prototipos son versiones preliminares del software que incorporan las funciones críticas. Durante esta etapa, se utilizan métodos de desarrollo ágil para iterar rápidamente basándose en el feedback del usuario y los resultados de las pruebas iniciales. Las herramientas de integración y despliegue continuos (CI/CD) son fundamentales para agilizar este proceso.

Selección de Características Críticas: Identificamos las funciones esenciales que deben ser probadas primero, como la capacidad de la herramienta para analizar y detectar vulnerabilidades en el código fuente, integrar técnicas de inteligencia artificial para la mejora de los procesos de detección y la interfaz de usuario que permitirá a los desarrolladores interactuar con el sistema.

Adoptaremos metodologías ágiles para la implementación de los prototipos. A través de **Sprints** cortos, cada ciclo de desarrollo produce una versión mejorada del prototipo, incorporando nuevas funcionalidades y refinamientos basados en las pruebas anteriores y el feedback de los usuarios. Se facilitará sesiones regulares de revisión con

usuarios potenciales y partes interesadas para recoger sus impresiones y sugerencias, asegurando que el producto final cumpla con las necesidades y expectativas del usuario.

Las herramientas de CI/CD juegan un papel crucial en el proceso de implementación de prototipos. Automatizaremos la integración de código de los prototipos, permitiendo que cada cambio en el código sea automáticamente compilado, probado y reportado, lo que ayuda a detectar problemas de forma temprana y reduce los tiempos de integración. Implementaremos los cambios validados directamente en el entorno de prueba o producción, lo que facilitará pruebas continuas y la posibilidad de recibir feedback inmediato. Esto permitirá una iteración rápida sobre el prototipo basada en uso real y datos operativos.

Un ejemplo simple de un prototipo que podría ser parte de una herramienta de auditoría de código fuente automatizada, centrada en cargar y analizar archivos de código para buscar patrones específicos que podrían indicar vulnerabilidades sería:

Clase CodeAnalyzer: Esta clase se encargará de analizar el código fuente.

```
public class CodeAnalyzer {
    public void analyzeCode (String code) {
        if (code.contains("System.out.println")) {
            System.out.println("Alerta de seguridad: Uso de System.out.println encontrado.");
        } else {
            System.out.println("No se encontraron problemas de seguridad específicos.");
        }
    }
}
```

Clase Main: Esta clase cargará los archivos y utilizará CodeAnalyzer para analizarlos.

```
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.stream.Stream;

public class Main {
    public static void main(String[] args) {
        CodeAnalyzer analyzer = new CodeAnalyzer();
        // Asegúrate de cambiar la ruta del archivo según corresponda.
        String pathToFile = "src/exampleCode.txt";
        try (Stream<String> stream = Files.lines(Paths.get(pathToFile))) {
            stream.forEach(analyzer::analyzeCode);
        } catch (Exception e) {
            System.err.println("Error al leer el archivo: " + e.getMessage());
        }
    }
}
```

Creamos un archivo de texto simple llamado exampleCode.txt en el directorio src con el siguiente contenido para probar:

```
System.out.println("Hello, world!");
int a = 5;
int b = 10;
System.out.println(a + b);
```

Ejecutamos la clase Main desde el IDE. Debería analizar el archivo exampleCode.txt y mostrar alertas si encuentra usos de System.out.println.

6.2.4. Entrenamiento de Modelos de IA.

El entrenamiento de modelos de inteligencia artificial es un componente crucial en el desarrollo de nuestra herramienta de auditoría de código fuente automatizada. Este proceso permite al sistema aprender de grandes volúmenes de datos y mejorar su capacidad para identificar patrones de código que podrían indicar la presencia de vulnerabilidades.

Antes de proceder con el entrenamiento de los modelos, es fundamental preparar y preprocesar los datos de código fuente:

Recopilación de Datos: Compilar una amplia base de datos de muestras de código fuente, que incluye tanto código seguro como código que contiene vulnerabilidades conocidas.

- Plataformas como GitHub, GitLab, y Bitbucket son recursos invaluables para obtener muestras de código fuente. Puedes buscar proyectos en lenguajes de programación específicos.

- Utilizar bases de datos de vulnerabilidades como el National Vulnerability Database (NVD), Common Vulnerabilities and Exposures (CVE), y otros registros similares para identificar vulnerabilidades conocidas en el software. Estas bases de datos a menudo enlazan a los commits en los repositorios de código que corrigen las vulnerabilidades, proporcionando ejemplos directos de código vulnerable y su correspondiente corrección.

> “El programa de Vulnerabilidades y Exposiciones Comunes (CVE) es un diccionario o glosario de vulnerabilidades que se han identificado para bases de código específicas, como aplicaciones de software o bibliotecas abiertas. Esta lista permite a las partes interesadas adquirir los detalles de las vulnerabilidades haciendo referencia a un identificador único conocido como ID de CVE. Ha aumentado su conciencia en los últimos años, por lo que es importante que los participantes y los usuarios entiendan los elementos fundamentales del programa.
>
> Fundado en 1999, el programa CVE es mantenido por la corporación MITRE y patrocinado por los EE. UU. Departamento de Seguridad Nacional (DHS) y la Agencia de Ciberseguridad y Seguridad de la Infraestructura (CISA). Las identificaciones de CVE son asignadas principalmente por MITRE, así como por organizaciones autorizadas conocidas como Autoridades de Numeración de CVE (CNA), un grupo internacional de proveedores e investigadores de numerosos países. El proyecto tiene una junta asesora compuesta por actores importantes en la investigación de la ciberseguridad, el mundo académico y las comunidades de desarrollo de software.
>
> El programa CVE se creó con la visión de convertirse en el estándar de la industria en el establecimiento de una línea de base para las vulnerabilidades, y toda la

información contenida en el proyecto está disponible públicamente para cualquier parte interesada. Esto permite a las partes interesadas un medio común para discutir e investigar hazañas específicas y únicas. Los ID de CVE también son utilizados por los proveedores y el personal de ciberseguridad para la investigación y la identificación de nuevas vulnerabilidades. (MITRE y los CNA no ayudan a mitigar o parchear las vulnerabilidades en la lista de CVE).” (NATIONAL VULNERABILITY DATABASE, 2022).

- Sitios como LeetCode, HackerRank, y otros ofrecen una gran cantidad de código escrito por una amplia gama de programadores. Aunque estos no siempre se centran en la seguridad, pueden ser útiles para entrenar modelos en estilos de codificación variados y lógica general.

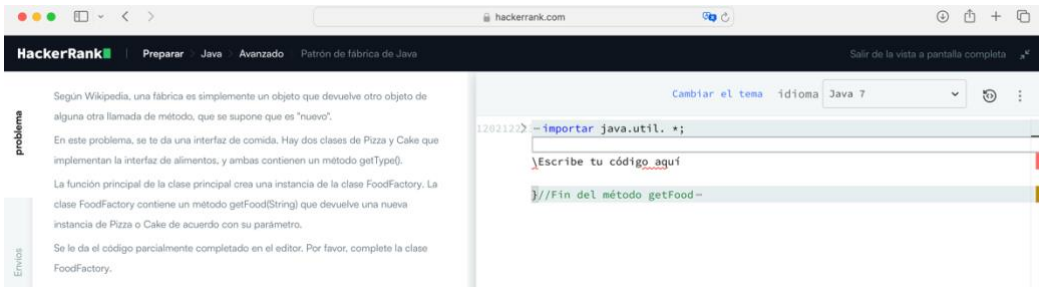

Ilustración 28: Ejemplo acceso al sitio HackerRank SOURCE: https://www.hackerrank.com/challenges/java-factory/problem?isFullScreen=true

- Generar datos sintéticos utilizando herramientas que automáticamente insertan vulnerabilidades conocidas en el código seguro para crear ejemplos de entrenamiento. Esto puede ser útil para garantizar que el modelo aprenda a identificar vulnerabilidades específicas.

- Automatizar la recolección de ejemplos de código de foros, blogs de tecnología, y sitios de preguntas y respuestas como Stack Overflow.

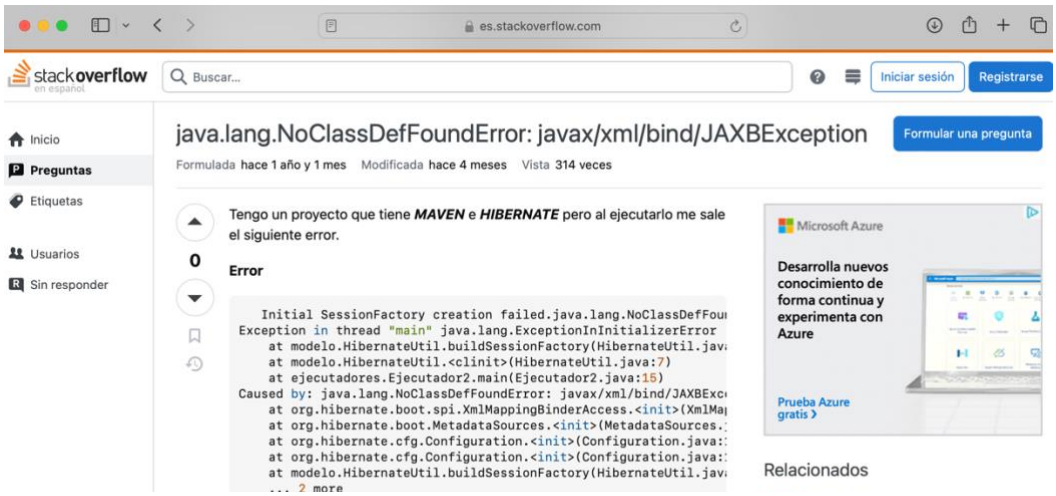

Ilustración 29: Ejemplo acceso al sitio Stack Overflow SOURCE: https://es.stackoverflow.com/questions/585691/java-lang-noclassdeffounderror-javax-xml-bind-jaxbexception

Etiquetado de Datos: Cada muestra de código fuente debe ser etiquetada adecuadamente como **'segura' o 'vulnerable'**, según corresponda. Este etiquetado es esencial para el aprendizaje supervisado.

Limpieza y Normalización: Eliminar cualquier dato irrelevante o redundante y normalizar el formato del código para asegurar la consistencia en el entrenamiento.

División de Datos: Dividir el conjunto de datos en sets de entrenamiento, validación y prueba. Esto permite evaluar la efectividad del modelo de manera objetiva.

Seleccionaremos **una red neuronal profunda** para manejar las complejidades del análisis de código fuente debido a su capacidad para aprender representaciones de alto nivel en datos secuenciales y su eficacia demostrada en tareas de procesamiento de lenguaje natural **(NLP)**:

Modelo de Red Neuronal Recurrente (RNN): Ideal para manejar secuencias de datos, como lo son las líneas de código fuente. Dentro de las RNN, las unidades LSTM (Long Short-Term Memory) o GRU (Gated Recurrent Units) pueden ser especialmente efectivas al manejar dependencias de largo plazo, comunes en estructuras de código. Herramienta con **MathWorks** son de gran utilidad para la creación de redes neuronales recurrentes.

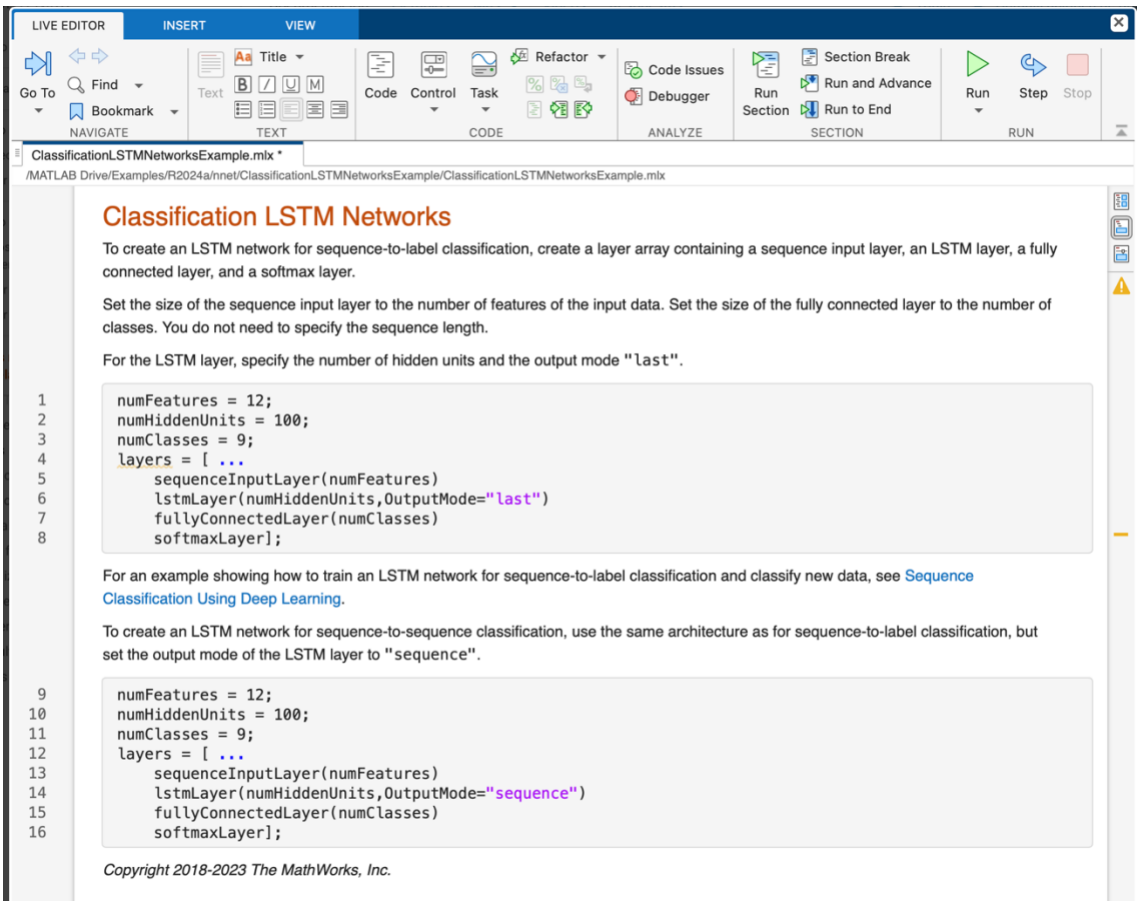

Ilustración 30: Ejemplo uso herramienta MathWorks SOURCE:
https://es.mathworks.com/help/deeplearning/ug/long-short-term-memory-networks.html

Embeddings de Código: Transformar cada línea o bloque de código en vectores numéricos que representan semánticamente el contenido del código. Esto puede hacerse utilizando técnicas como Word2Vec o GloVe, adaptadas al dominio específico del código fuente.

Capas Ocultas y Función de Activación: Configurar varias capas ocultas con funciones de activación como **ReLU** o **tanh** para introducir no linealidades en el modelo, lo cual es crucial para aprender patrones complejos en los datos.

Optimización: Utilizar algoritmos como **Adam o SGD (Stochastic Gradient Descent)** para optimizar la función de pérdida, que típicamente será una función de pérdida de entropía cruzada en clasificaciones binarias de **'seguro' vs 'vulnerable'**.

Entrenamiento Iterativo: Dada la naturaleza cambiante de las prácticas de programación y la aparición de nuevas vulnerabilidades, el modelo debe ser entrenado iterativamente en nuevos conjuntos de datos que reflejen las tendencias actuales en desarrollo de software.

6.2.5. Pruebas Rigurosas.

La herramienta es sometida a pruebas exhaustivas para validar su eficacia y eficiencia. Esto incluye pruebas unitarias, pruebas de integración y pruebas de sistema completas. Además, se realizan pruebas de penetración para asegurar que la herramienta no solo detecta las vulnerabilidades, sino que también resiste ataques dirigidos que podrían explotar sus propias debilidades.

6.2.6. Despliegue y Monitoreo.

Una vez que la herramienta ha pasado todas las pruebas de calidad, se procede a su despliegue. El despliegue inicial puede ser en un entorno limitado como una beta cerrada para monitorear su comportamiento en escenarios de uso real. El monitoreo continuo es crucial para identificar cualquier problema operativo o de rendimiento, y para recopilar datos que podrían usarse para mejorar futuras versiones del software.

6.2.7. Retroalimentación y Mejora Continua.

El último paso es un proceso cíclico de recopilación de feedback de los usuarios y la mejora continua del software. Utilizando técnicas de machine Learning, la herramienta puede ajustarse y optimizarse continuamente para mejorar su precisión y eficiencia.

7. Conclusiones y trabajos futuros.

En el ámbito del desarrollo de software, la seguridad se ha convertido en una prioridad esencial debido a la creciente sofisticación de las amenazas cibernéticas y la dependencia crítica de las aplicaciones web en cualquier ámbito. En el trabajo fin de grado se han abordado estos desafíos mediante el desarrollo de una herramienta de auditoría de código fuente automatizada, utilizando técnicas avanzadas de inteligencia artificial para mejorar la detección y mitigación de vulnerabilidades. Además, se han obtenido resultados significativos que subrayan la eficacia y eficiencia de las soluciones

propuestas, destacando la relevancia de adoptar enfoques de seguridad integrales y metodologías reconocidas en el ciclo de vida del desarrollo de software.

La herramienta de auditoría de código fuente automatizada mostrada demuestra una capacidad notable para mejorar la detección de vulnerabilidades y errores, superando las limitaciones de las auditorías manuales gracias a la integración de técnicas de inteligencia artificial. La implementación de algoritmos de aprendizaje automático y procesamiento de lenguaje natural permite un análisis más profundo y preciso del código fuente, lo que supone una mejora significativa en la identificación de patrones de vulnerabilidad en comparación con métodos tradicionales.

La automatización de la auditoría de código pretende reducir considerablemente el tiempo necesario para realizar análisis exhaustivos y disminuye la probabilidad de errores humanos, resultando en un proceso más eficiente y confiable. Esta automatización de complemente con la necesidad de integrar prácticas de seguridad en todas las etapas del ciclo de vida del desarrollo de software (SDLC), desde el diseño inicial hasta el despliegue y mantenimiento continuo, para garantizar aplicaciones más seguras y resilientes ante amenazas.

Estas pruebas validan la eficacia de la herramienta en entornos de producción, demostrando que supera a los métodos convencionales en términos de precisión y capacidad para adaptarse a nuevas amenazas y vulnerabilidades. Además, el uso de metodologías estándar como OWASP proporciona un marco sólido para el desarrollo seguro de aplicaciones web, complementado por la herramienta de IA para ofrecer una cobertura integral y robusta de los aspectos de seguridad relevantes.

El impacto positivo del proyecto en el campo de la seguridad informática subraya la importancia de las innovaciones tecnológicas para mejorar la calidad y seguridad del software.

Las conclusiones derivadas de este estudio no solo reflejan el éxito de la implementación técnica, sino que también destacan la relevancia de adoptar enfoques de seguridad integrales y metodologías reconocidas en el ciclo de vida del desarrollo de software.

Podemos resumir las conclusiones en:

Eficiencia y Eficacia de la Herramienta: La herramienta de auditoría de código fuente automatizada desarrollada demuestra una capacidad notable para mejorar la detección de vulnerabilidades y errores, superando las limitaciones de las auditorías manuales gracias a la integración de técnicas de inteligencia artificial (IA).

Integración de IA en Seguridad: La implementación de algoritmos de aprendizaje automático y procesamiento de lenguaje natural permite un análisis más profundo y preciso del código fuente, mostrando una mejora significativa en la identificación de patrones de vulnerabilidad en comparación con métodos tradicionales.

Ahorro de Tiempo y Reducción de Errores Humanos: La automatización de la auditoría de código reduce considerablemente el tiempo necesario para realizar análisis

exhaustivos y disminuye la probabilidad de errores humanos, lo que resulta en un proceso más eficiente y confiable.

Importancia del Desarrollo Seguro: Se subraya la necesidad de integrar prácticas de seguridad en todas las etapas del ciclo de vida del desarrollo de software (SDLC), desde el diseño inicial hasta el despliegue y mantenimiento continuo, para garantizar aplicaciones más seguras y resilientes ante amenazas.

Validación y Comparación con Métodos Tradicionales: Las pruebas realizadas validan la eficacia de la herramienta en entornos de producción, demostrando que supera a los métodos convencionales en términos de precisión y capacidad para adaptarse a nuevas amenazas y vulnerabilidades.

Adopción de Estándares y Metodologías Reconocidas: El uso de metodologías estándar como OWASP proporciona un marco sólido para el desarrollo seguro de aplicaciones web, complementado por la herramienta de IA para ofrecer una cobertura integral y robusta de los aspectos de seguridad relevantes.

7.1. Trabajos Futuros.

7.1.1. Mejora de Algoritmos de IA.

Continuar perfeccionando los algoritmos de aprendizaje automático y procesamiento del lenguaje natural es esencial para aumentar aún más la precisión y eficiencia de la herramienta de auditoría de código fuente. Existen varias líneas de investigación y desarrollo que pueden explorarse para lograr este objetivo:

a) **Incorporación de Técnicas de Aprendizaje Profundo (Deep Learning):**

Redes Neuronales Convolucionales (CNNs): Aunque las CNNs son más conocidas por su aplicación en el reconocimiento de imágenes, también pueden ser útiles en la detección de patrones complejos en secuencias de código fuente. Su capacidad para identificar características jerárquicas podría mejorar la identificación de vulnerabilidades que dependen de interacciones no triviales entre diferentes partes del código.

Redes Neuronales Recurrentes (RNNs) y LSTM: Las RNNs, y en particular las redes de memoria a largo corto plazo (LSTM), son adecuadas para analizar secuencias de datos y podrían ser muy efectivas para entender la lógica y el flujo del código fuente, identificando vulnerabilidades que dependen del contexto y del estado.

Transformers: Modelos basados en transformers, como BERT y GPT, han revolucionado el procesamiento del lenguaje natural y podrían aplicarse para entender y analizar el código fuente de manera más contextual, capturando relaciones a largo plazo entre las diferentes partes del código.

b) **Mejoras en el Procesamiento del Lenguaje Natural (NLP):**

Modelos de Lenguaje Contextual: Utilizar modelos de lenguaje contextualizados que entiendan mejor el contexto en el que aparece el código, permitiendo una detección más precisa de vulnerabilidades y errores específicos del dominio.

Análisis Semántico Avanzado: Implementar técnicas de análisis semántico que puedan interpretar y entender el significado del código más allá de su sintaxis, lo cual es crucial para identificar vulnerabilidades que dependen de la lógica del negocio y las interacciones complejas entre componentes.

c) Aprendizaje Transferido (Transfer Learning):

Transferencia de Conocimiento: Aplicar técnicas de aprendizaje transferido para aprovechar modelos preentrenados en grandes conjuntos de datos relacionados, adaptándolos a la detección de vulnerabilidades específicas del dominio del software auditado.

Fine-Tuning en Dominios Específicos: Ajustar modelos preentrenados en grandes conjuntos de datos generales para mejorar su rendimiento en dominios específicos, como aplicaciones web, sistemas embebidos, o software de seguridad crítica.

d) Automatización y Optimización del Proceso de Entrenamiento:

AutoML: Utilizar técnicas de AutoML (Automated Machine Learning) para automatizar la selección, entrenamiento y optimización de modelos de IA, asegurando que se utilizan los algoritmos más efectivos sin necesidad de intervención manual constante.

Generación de Datos Sintéticos: Crear conjuntos de datos sintéticos que reflejen escenarios de vulnerabilidades reales para entrenar y evaluar modelos de IA, especialmente en casos donde los datos etiquetados son limitados o difíciles de obtener.

e) Mejora de la Interpretabilidad y Explicabilidad de los Modelos:

Modelos Explicables: Desarrollar modelos de IA que no solo sean precisos, sino también explicables, proporcionando justificaciones claras y comprensibles para las detecciones de vulnerabilidades y errores. Esto es crucial para ganar la confianza de los desarrolladores y facilitar la adopción de la herramienta.

Herramientas de Visualización: Implementar herramientas de visualización que permitan a los desarrolladores entender mejor cómo la IA está tomando decisiones, facilitando la identificación de falsas alarmas y la mejora continua del modelo.

f) Evaluaciones de Rendimiento y Escalabilidad:

Pruebas en Entornos Reales: Realizar evaluaciones extensivas de la herramienta en una variedad de entornos de producción para validar su rendimiento y escalabilidad. Esto incluye probar la herramienta en diferentes lenguajes de programación, arquitecturas de software y tamaños de proyectos.

Optimización de Recursos: Investigar y desarrollar técnicas para optimizar el uso de recursos computacionales durante el análisis del código, asegurando que la herramienta pueda ser utilizada eficientemente incluso en entornos con limitaciones de hardware.

7.1.2. Ampliación del Conjunto de Datos.

Expandir y diversificar los conjuntos de datos utilizados para entrenar los modelos de inteligencia artificial (IA) es crucial para mejorar la eficacia y precisión de la herramienta de auditoría de código fuente automatizada. Un conjunto de datos más amplio y diverso permitirá que la herramienta identifique una gama más amplia de vulnerabilidades y se adapte mejor a diferentes lenguajes de programación y entornos de desarrollo.

La recopilación de datos de diversas fuentes es fundamental. Repositorios de código abierto como **GitHub y GitLab** contienen proyectos de diferentes tamaños, lenguajes y estilos de codificación, proporcionando una rica fuente de ejemplos reales de código. Es esencial seleccionar proyectos que sean representativos de diversos dominios de aplicación, como aplicaciones web, móviles, sistemas embebidos y software empresarial. Además, utilizar bases de datos públicas de vulnerabilidades conocidas, como **CVE** y **NVD**, proporciona ejemplos concretos de vulnerabilidades explotadas en diferentes entornos. Establecer colaboraciones con empresas para obtener acceso a sus bases de código (anónimas y des identificadas) también puede enriquecer los conjuntos de datos con ejemplos de vulnerabilidades y patrones de seguridad relevantes en entornos de producción.

La generación de datos sintéticos también es una estrategia eficaz. Crear herramientas que generen automáticamente ejemplos de código con vulnerabilidades comunes y desarrollar técnicas de mutación de código que modifiquen ejemplos existentes para introducir variaciones y nuevas vulnerabilidades son pasos importantes. Estas técnicas aseguran que las vulnerabilidades generadas cubran una amplia gama de escenarios y contextos, simulando diferentes formas en las que las vulnerabilidades pueden manifestarse en el código.

El etiquetado y la anotación de datos son cruciales para asegurar la calidad de los conjuntos de datos. Involucrar a expertos en seguridad informática para anotar y etiquetar manualmente conjuntos de datos con ejemplos de código vulnerables y seguros garantiza que los modelos de IA reciban entrenamiento con datos de alta calidad y relevancia. Además, desarrollar herramientas que puedan etiquetar automáticamente grandes volúmenes de código utilizando reglas y heurísticas basadas en patrones de vulnerabilidad conocidos es una práctica efectiva. Es importante implementar un proceso de validación donde un subconjunto de las etiquetas generadas automáticamente sea revisado por expertos humanos para asegurar la precisión del etiquetado.

La evaluación y validación de los datos es un paso continuo. Hay que asegurar que los conjuntos de datos incluyan ejemplos de una amplia gama de lenguajes de programación y frameworks permitirá que la herramienta se entrene en diferentes sintaxis y paradigmas de programación. Implementar un sistema de pruebas cruzadas donde los modelos entrenados se evalúen continuamente en diferentes subconjuntos de datos medirá su rendimiento y capacidad de generalización. Utilizar los resultados de estas evaluaciones para ajustar y mejorar continuamente los modelos garantizará que se mantengan actualizados con las últimas amenazas y técnicas de seguridad.

La ampliación y diversificación de los conjuntos de datos utilizados para entrenar los modelos de IA es fundamental para mejorar la precisión y la eficiencia de la herramienta de auditoría de código fuente. Al recopilar datos de fuentes diversas, generar datos sintéticos, etiquetar y anotar de manera precisa, y evaluar continuamente el rendimiento, se puede crear una herramienta robusta y adaptable que identifique una gama más amplia de vulnerabilidades y se adapte mejor a diferentes lenguajes de programación y entornos de desarrollo. Estos esfuerzos no solo mejorarán la capacidad de la herramienta para detectar vulnerabilidades, sino que también fomentarán una cultura de desarrollo seguro y resiliente en la industria del software.

7.1.3. Integración con Herramientas de CI/CD.

Fortalecer la integración de la herramienta de auditoría de código fuente automatizada con pipelines de integración y despliegue continuo **(CI/CD)** es vital para detectar vulnerabilidades de manera temprana y permitir una respuesta rápida a cualquier problema de seguridad identificado. Esta integración mejorará significativamente el flujo de trabajo de desarrollo y la seguridad del software producido.

La integración de la herramienta con **CI/CD** puede comenzar con la automatización del análisis de seguridad en cada etapa del pipeline. Configurar la herramienta para que realice auditorías de seguridad automáticamente en cada **commit o pull request** garantiza que el código nuevo o modificado se evalúe en busca de vulnerabilidades antes de ser fusionado con la rama principal. Esto asegura que cualquier problema de seguridad se detecte y aborde en las primeras etapas del desarrollo, minimizando el riesgo de introducir vulnerabilidades en el código base.

Además, es esencial personalizar la configuración de la herramienta para adaptarse a las necesidades específicas del proyecto y del equipo de desarrollo. Permitir ajustes en las reglas y políticas de seguridad según los requisitos del proyecto garantiza que la herramienta se enfoque en las vulnerabilidades más relevantes. Por ejemplo, se pueden configurar diferentes niveles de severidad para alertas, priorizando las vulnerabilidades críticas que deben abordarse de inmediato y diferenciándolas de las de menor riesgo.

La notificación y reportes automáticos son componentes clave de una integración efectiva con **CI/CD**. Implementar un sistema de notificaciones que informe automáticamente a los desarrolladores sobre los problemas de seguridad detectados, ya sea a través de correos electrónicos, mensajes en plataformas de colaboración como **Slack** o directamente en las herramientas de gestión de proyectos, facilita una respuesta rápida y coordinada. Además, generar reportes detallados y accesibles en cada ejecución del pipeline proporciona a los desarrolladores información clara y accionable sobre las

vulnerabilidades encontradas, las áreas afectadas y las recomendaciones para su corrección.

Para asegurar que la herramienta se mantenga eficaz y relevante, es fundamental integrar un ciclo de retroalimentación continua. Los resultados de las auditorías de seguridad deben revisarse regularmente y utilizarse para mejorar las reglas y los algoritmos de detección de la herramienta. Esto puede incluir ajustes basados en el feedback de los desarrolladores sobre falsos positivos o negativos, así como la incorporación de nuevas reglas a medida que surgen nuevas amenazas y vulnerabilidades en el ecosistema de desarrollo.

La compatibilidad con múltiples herramientas y plataformas **CI/CD** es otro aspecto importante. La herramienta debe ser flexible y capaz de integrarse con una variedad de sistemas **CI/CD, como Jenkins, Travis CI, GitLab CI, y Azure DevOps, entre otros**. Esto asegura que equipos con diferentes configuraciones de desarrollo puedan aprovechar los beneficios de la auditoría de seguridad automatizada sin necesidad de realizar cambios significativos en sus pipelines existentes.

Finalmente, la educación y capacitación continua del equipo de desarrollo en el uso de la herramienta y la interpretación de sus resultados es crucial para maximizar su efectividad. Organizar sesiones de formación y proporcionar recursos educativos ayuda a los desarrolladores a comprender mejor las vulnerabilidades identificadas y las mejores prácticas para abordarlas. Esto no solo mejora la seguridad del código, sino que también contribuye a una cultura de seguridad dentro del equipo.

7.1.4. Evaluaciones de Rendimiento en Escenarios Reales.

Realizar pruebas y evaluaciones extensivas en entornos de producción reales es crucial para validar el rendimiento y la escalabilidad de la herramienta de auditoría de código fuente automatizada. Estas evaluaciones aseguran que la herramienta funcione eficazmente bajo diversas cargas de trabajo y configuraciones, proporcionando información valiosa sobre su comportamiento en escenarios del mundo real.

Evaluación del Rendimiento: Implementar una serie de pruebas en entornos de producción que reflejen situaciones reales de uso es esencial. Esto incluye la ejecución de auditorías de seguridad en proyectos de diferentes tamaños y complejidades, midiendo el tiempo de análisis y los recursos utilizados. Las pruebas de carga y estrés ayudan a determinar cómo se comporta la herramienta bajo condiciones normales y extremas, identificando posibles cuellos de botella.

Validación de la Escalabilidad: Evaluar la escalabilidad de la herramienta implica probar su capacidad para manejar un número creciente de auditorías simultáneas y su desempeño en diferentes configuraciones de infraestructura. Implementar pruebas de escalabilidad horizontal (añadiendo más instancias) y vertical (aumentando los recursos de una sola instancia) proporciona información sobre su adaptabilidad. Probar en infraestructuras de **nube y on-premise** valida su desempeño en diferentes entornos.

Análisis de Compatibilidad: Probar la herramienta con diversos lenguajes de programación, frameworks y arquitecturas de software asegura su adaptabilidad. **Realizar auditorías en proyectos con lenguajes como Java, Python, JavaScript, C++,**

y frameworks como React, Angular, Django, y Spring, garantiza que la herramienta puede manejar una amplia gama de tecnologías. Evaluar su desempeño en diferentes sistemas operativos (**Windows, Linux, macOS**) es también crucial.

Monitorización y Análisis de Resultados: Implementar sistemas de monitorización y registro detallados durante las pruebas es esencial para recopilar datos sobre el comportamiento de la herramienta. Utilizar herramientas como **Prometheus, Grafana o ELK Stack** permite visualizar métricas clave y realizar un seguimiento continuo del rendimiento y la utilización de recursos. Analizar estos datos ayuda a optimizar los algoritmos de detección de vulnerabilidades y mejorar la eficiencia del procesamiento.

Colaboración y Feedback: Colaborar con organizaciones que proporcionen acceso a sus entornos de producción y ofrezcan feedback sobre la herramienta permite probarla en escenarios reales y recibir comentarios directos de los usuarios finales. Participar en programas de pruebas beta y pilotos controlados en diferentes industrias proporciona una visión más amplia de su comportamiento en diversos contextos, ayudando a ajustar la herramienta para satisfacer mejor las demandas del mercado.

Realizar pruebas y evaluaciones extensivas en entornos de producción reales es fundamental para validar el rendimiento y la escalabilidad de la herramienta de auditoría de código fuente automatizada. Estas evaluaciones deben incluir pruebas de carga y estrés, validación de escalabilidad, análisis de compatibilidad, y monitorización detallada del rendimiento. Colaborar con organizaciones y recopilar feedback de usuarios reales permitirá optimizar la herramienta y asegurar que cumple con los estándares de calidad y seguridad necesarios para su implementación en diversos entornos de producción.

7.1.5. Desarrollo de Módulos de Explicabilidad.

Implementar funcionalidades que permitan explicar las decisiones tomadas por los modelos de inteligencia artificial (IA) es crucial para aumentar la transparencia y la confianza en la herramienta de auditoría de código fuente automatizada. La explicabilidad es especialmente importante en contextos donde se requieren auditorías de seguridad exhaustivas y justificables.

La explicabilidad en IA se refiere a la capacidad de los modelos para proporcionar interpretaciones comprensibles de sus predicciones y decisiones. En el contexto de la auditoría de código fuente, esto significa que los desarrolladores y los equipos de seguridad pueden entender por qué se ha identificado una vulnerabilidad o error específico, lo cual es esencial para la confianza en la herramienta y para tomar acciones correctivas informadas.

La visualización de decisiones es otra área importante. Utilizar diagramas de influencia para mostrar cómo diferentes partes del código afectan la decisión del modelo y mapas de calor que indiquen las áreas de mayor riesgo basados en la evaluación del modelo, pueden ayudar a los desarrolladores a identificar rápidamente las secciones del código que requieren atención. Estas visualizaciones hacen que las explicaciones sean más accesibles y comprensibles.

Generar reportes detallados que expliquen cada vulnerabilidad detectada, incluyendo el razonamiento del modelo y las características clave que llevaron a la decisión, es esencial para la documentación y la auditoría. Estos reportes deben ser comprensibles para los desarrolladores y los equipos de seguridad. Además, crear historias de usuario que ilustren cómo la herramienta llega a sus conclusiones puede proporcionar ejemplos prácticos y contextuales que muestren el proceso de decisión del modelo.

Desarrollar interfaces de usuario interactivas permite a los usuarios explorar cómo las decisiones del modelo cambian cuando se modifican ciertos parámetros o se ajustan partes del código. Esto no solo ayuda a entender mejor el modelo, sino que también permite una forma de aprendizaje activo. Implementar sistemas donde los usuarios puedan proporcionar feedback sobre las decisiones del modelo, indicando si consideran que las explicaciones son útiles y precisas, es también fundamental. Este feedback puede ser utilizado para mejorar continuamente los algoritmos de explicabilidad.

Utilizar técnicas avanzadas de análisis como el análisis contrafactual y la descomposición de modelos puede proporcionar explicaciones adicionales. El análisis contrafactual muestra ejemplos de "qué pasaría si", indicando cómo diferentes cambios en el código afectarían la decisión del modelo, mientras que la descomposición de modelos desglosa el modelo en partes más pequeñas y comprensibles que explican cómo las subcomponentes del modelo contribuyen a la decisión final.

Integrar módulos de explicabilidad en el ciclo de desarrollo de la herramienta requiere un enfoque metódico. Iniciar con el diseño de prototipos de las funcionalidades de explicabilidad, realizando pruebas piloto para evaluar su eficacia, y luego incorporar estas funcionalidades en el pipeline de integración continua **(CI/CD)** para asegurar que las explicaciones se generen automáticamente junto con los resultados de las auditorías de seguridad. Evaluar regularmente la calidad y la utilidad de las explicaciones proporcionadas, basándose en el feedback de los usuarios y en estudios de usabilidad, y mejorar continuamente los módulos de explicabilidad para asegurar que se mantengan alineados con las necesidades de los desarrolladores y los requisitos de seguridad.

Desarrollar módulos de explicabilidad para la herramienta de auditoría de código fuente automatizada es fundamental para aumentar la transparencia y la confianza en el sistema. Implementar técnicas de interpretación local, proporcionar visualizaciones claras, generar reportes detallados, desarrollar interfaces interactivas y utilizar técnicas avanzadas de análisis garantizará que los usuarios comprendan y confíen en las decisiones tomadas por los modelos de IA. Esto es especialmente importante en contextos de auditorías de seguridad exhaustivas, donde se requiere una justificación clara y comprensible de cada decisión.

7.2. Dificultades, Problemas y Limitaciones encontradas.

Durante el desarrollo de esta herramienta de auditoría de código fuente automatizada, se enfrentaron diversas dificultades y limitaciones que merecen ser destacadas. Uno de los principales retos fue la rápida evolución de las regulaciones de seguridad informática y las tecnologías, lo que demandó una constante actualización y adaptación de la herramienta para mantener su relevancia y eficacia. Además, durante el análisis de riesgos, surgieron vulnerabilidades críticas no anticipadas, lo que requirió

recursos adicionales para su evaluación y respuesta inmediata, incluyendo la revisión y adaptación de las prácticas de diseño propuestas.

Las restricciones en la disponibilidad de materiales cualificados también impactaron el avance del proyecto, obligando a priorizar actividades críticas y buscar eficiencias en el trabajo. Fue necesario desarrollar competencias adicionales o reasignar tareas para optimizar el uso de los recursos disponibles.

Por otra parte, la falta de compromiso de las partes interesadas en ciertos momentos obstaculizó el progreso del proyecto. Para mitigar este problema, se fomentó una cultura de seguridad mediante la educación continua y se establecieron canales de comunicación efectivos para mejorar la colaboración.

Los plazos ajustados y la complejidad del estudio tecnológico fueron otros desafíos significativos. No siempre se alcanzaron las fechas previstas para cada hito, lo que obligó a ajustar los planes de seguimiento y cambiar la metodología utilizada. La complejidad inherente en el entendimiento y desarrollo de los mecanismos de seguridad también ralentizó el cumplimiento del cierre del proyecto, requiriendo ajustes continuos en los planes de seguimiento y la adopción de metodologías diferentes para abordar estos desafíos.

RESUMEN

En el ámbito de la ciberseguridad, donde cada línea de código puede determinar la seguridad de los sistemas, este Trabajo Fin de Grado pretende presentar una innovación crucial. Titulado "Mecanismos de Seguridad para el Diseño de Desarrollo Web," propone una herramienta de auditoría de código fuente automatizada que revoluciona la manera en que se abordan las vulnerabilidades de seguridad en aplicaciones web.

Esta herramienta, impulsada por técnicas avanzadas de inteligencia artificial, logra identificar vulnerabilidades y errores con una precisión excepcional, superando las limitaciones de las auditorías manuales. La integración de algoritmos de aprendizaje automático y procesamiento de lenguaje natural permite un análisis profundo y eficiente del código, detectando patrones complejos y amenazas emergentes.

Una característica destacada es la integración de la herramienta en los pipelines de integración y despliegue continuo **(CI/CD)**. Esta integración asegura que **cada nuevo commit o pull request** se someta automáticamente a una auditoría de seguridad, detectando y abordando vulnerabilidades en tiempo real. Esto no solo mejora la seguridad del código desde las etapas iniciales del desarrollo, sino que también facilita una respuesta rápida y coordinada a cualquier problema identificado.

También se enfoca en la ampliación y diversificación de los conjuntos de datos utilizados para entrenar los modelos de IA. Al incorporar datos de repositorios de código abierto, bases de datos de vulnerabilidades conocidas y colaboraciones con empresas, la herramienta se adapta eficazmente a diferentes lenguajes de programación y entornos de desarrollo. Esto asegura que puede manejar una amplia gama de tecnologías y escenarios.

Además, se ha desarrollado módulos de explicabilidad que desmitifican las decisiones tomadas por los modelos de IA. Utilizando técnicas como visualizaciones y reportes detallados, estos módulos proporcionan interpretaciones comprensibles de las predicciones del modelo

Finalmente, se incluye pruebas y evaluaciones extensivas en entornos de producción reales. Estas pruebas validan el rendimiento y la escalabilidad de la herramienta bajo diversas cargas de trabajo y configuraciones, asegurando su robustez y adaptabilidad. La colaboración con organizaciones para implementar proyectos piloto y recopilar feedback de usuarios reales fortalece aún más la eficacia de la herramienta.

Este Trabajo Fin de Grado no solo introduce una herramienta innovadora para la auditoría de código fuente, sino que establece un nuevo estándar en la ciberseguridad del desarrollo web. Al integrar prácticas de seguridad desde el inicio del desarrollo y emplear técnicas avanzadas de IA, este proyecto enfrenta eficazmente las amenazas actuales y está preparado para las futuras, marcando un hito significativo en la protección de aplicaciones web.

Glosario de término

AES[52]: "El Estándar de Cifrado Avanzado (AES), también conocido por su nombre original Rijndael (pronunciación holandesa: ['reinda:l]), es una especificación para el cifrado de datos electrónicos establecida por los EE. UU. Instituto Nacional de Estándares y Tecnología (NIST) en 2001."
Ataques de Denegación De Servicio[17]: "Un ataque de denegación de servicio **(DoS)** es un tipo de ciberataque en el que un actor malicioso tiene como objetivo que un ordenador u otro dispositivo no esté disponible para los usuarios a los que va dirigido, interrumpiendo el funcionamiento normal del mismo."
Backend[1]: "Se encarga de la lógica detrás del funcionamiento de la aplicación. Es la parte invisible de la aplicación que maneja los datos y realiza las operaciones necesarias para que todo funcione correctamente. Aquí se encuentran los servidores, las bases de datos y cualquier otro componente necesario para procesar y almacenar información."
BCrypt[53]: "es una función de hash de contraseñas diseñada por Niels Provos y David Mazières, basada en el cifrado Blowfish y presentada en USENIX en 1999.[1]Además de incorporar una sal para proteger contra los ataques de la mesa del arco iris, bcrypt es una función adaptativa: con el tiempo, el recuento de iteraciones se puede aumentar para hacerlo más lento, por lo que sigue siendo resistente a los ataques de búsqueda por fuerza bruta incluso con un mayor poder de cálculo."
Bootstrap[2]: "Es un framework de CSS que ofrece una serie de estilos y componentes predefinidos, lo que facilita la creación de páginas web responsivas y con un diseño atractivo."
C#[3]: "Conocido como C-sharp, es uno de los lenguajes de programación de Backend más famosos y preferido para la automatización en el entorno de Windows. C# también se utiliza para el desarrollo web en el marco ASP.net. Es uno de los lenguajes de programación más antiguos y una extensión de C++."
C++[3]: "Es una versión extendida del lenguaje C. C++ se introdujo con clases. Este concepto de clases o programación orientada a objetos no estaba presente en el lenguaje C. La idea de la programación orientada a objetos es vital para que cualquier lenguaje de programación en el mundo moderno escriba código estructurado usando clases y definiendo sus relaciones."
CI/CD[54]: "La integración continua (continuous integration en inglés) es una práctica de ingeniería de software que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. Entendemos por integración la compilación y ejecución de pruebas de todo un proyecto."
CORS[55]: "El intercambio de recursos de origen cruzado o CORS (Cross-origin resource sharing, en sus siglas en inglés) es un mecanismo que permite que se puedan solicitar recursos restringidos (como, por ejemplo, las tipografías) en una página web desde un dominio diferente del dominio que sirvió el primer recurso."
CSRF[56]: "El CSRF (del inglés Cross-site request forgery o falsificación de petición en sitios cruzados) es un tipo de exploit malicioso de un sitio web en el que comandos no autorizados son transmitidos por un usuario en el cual el sitio web confía. Esta vulnerabilidad es conocida también por otros nombres como XSRF, enlace hostil, ataque de un clic, secuestro de sesión, y ataque automático."
CSS[2] (Cascading Style Sheets): "Es el lenguaje utilizado para dar estilo y diseño a una página web. Con CSS, puedes controlar el color, la tipografía, el tamaño y la disposición de los elementos en la página."
CVE[57]: "Common Vulnerabilities and Exposures (CVE, traducción: «Vulnerabilidades y exposiciones comunes»), es una lista de información registrada sobre vulnerabilidades de seguridad conocidas, en la que cada referencia tiene un número de identificación CVE-ID, descripción de la vulnerabilidad, que versiones del software están afectadas, posible solución al fallo (si existe) o como configurar para mitigar la vulnerabilidad y referencias a publicaciones o entradas de foros o blog donde se ha hecho pública la vulnerabilidad o se demuestra su explotación."
DBM[58]: "En informática, una DBM es una biblioteca y un formato de archivo que proporciona acceso rápido y con una sola tecla a los datos. Siendo una base de datos de valores clave del Unix original, dbm es uno de los primeros ejemplos de un sistema NoSQL."
Deep Learning Natural[59]: "Aprendizaje profundo (en inglés, deep learning) es un conjunto de algoritmos de aprendizaje automático (en inglés, machine learning) que intenta modelar abstracciones de alto nivel en datos usando arquitecturas computacionales que admiten transformaciones no lineales múltiples e iterativas de datos expresados en forma matricial o tensorial."
DevSecOps[60]. "DevOps (acrónimo inglés de development -desarrollo- y operations -operaciones-) es un conjunto de prácticas que agrupan el desarrollo de software (Dev) y las operaciones de TI (Ops). Su objetivo es hacer más rápido el ciclo de vida del desarrollo de software y proporcionar una entrega continua de alta calidad. DevOps es una práctica complementaria al desarrollo de software ágil ; esto debido a que varias de las características de DevOps provienen de la metodología Agile (término en inglés para la metodología de desarrollo ágil)."
Firewalls[61]: "En informática, un cortafuegos (del término original en inglés firewall) es la parte de un sistema o una red informáticos que está diseñada para bloquear el acceso no autorizado, permitiendo al mismo tiempo comunicaciones autorizadas."
Frameworks de JavaScript[2]: "Existen varios frameworks populares de JavaScript que facilitan el desarrollo front end, como React, Angular y Vue.js. Estos frameworks proporcionan herramientas y estructuras para construir aplicaciones web de manera más eficiente y organizada."
Frontend[1]: "Se refiere a la parte de una aplicación web con la que los usuarios interactúan directamente. Es la cara visible de la aplicación, lo que los usuarios ven y con lo que interactúan. Esto incluye el diseño, la presentación visual y la interacción con el usuario. El Frontend se compone de HTML, CSS y JavaScript principalmente, aunque también puede incluir otras tecnologías como frameworks de Frontend, preprocesadores CSS o bibliotecas de JavaScript."
GDPR[62]: "El Reglamento General de Protección de Datos (RGPD) o Reglamento (UE) 2016/679, es una ley europea relativa a la protección de las personas físicas (independientemente de si son ciudadanos de la Unión) en lo que respecta al tratamiento de sus datos personales y a la libre circulación de estos datos en la Unión Europea y el Espacio Económico Europeo (EEE)."
HTML[2] (HyperText Markup Language): "Es el lenguaje de marcado utilizado para estructurar y organizar el contenido de una página web. Con HTML, puedes crear encabezados, párrafos, enlaces, imágenes y otros elementos básicos. "
HTTP[63]: "El protocolo de transferencia de hipertexto (en inglés: Hypertext Transfer Protocol, abreviado HTTP) es el protocolo de comunicación que permite las transferencias de información a través de archivos (XML, HTML…) en la World Wide Web."
HTTPS[64]: "El Protocolo seguro de transferencia de hipertexto (en inglés: Hypertext Transfer Protocol Secure o https) es un protocolo de aplicación basado en el protocolo http, destinado a la transferencia segura de datos de hipertexto, es decir, es la versión segura de http."
Humanware[9]: "Es un término informático que se utiliza para definir los recursos humanos de un sistema informático o el hardware y el software que es diseñado pensando en la experiencia y la interfaz que le dará el usuario final."
infraestructura de la tecnología de informática[15] (TI): "La tecnología de la información (TI o IT, Information Technology) es la aplicación de ordenadores y equipos de telecomunicación para almacenar, recuperar, transmitir y manipular datos, con frecuencia utilizado en el de los negocios u otras empresas. El término se utiliza como sinónimo para los computadores y las redes de

computadoras, pero también abarca otras tecnologías de distribución de información, tales como la televisión y los teléfonos. Múltiples industrias están asociadas con las tecnologías de la información: hardware y software de computadoras, electrónica, semiconductores, internet, equipos de telecomunicación, el comercio electrónico y los servicios computacionales."

Java[3]: "Es una de las tecnologías Backend más poderosas, que ocupa el segundo lugar, según el índice TIOBE 2021. James Gosling desarrolló originalmente esta tecnología de programación en 1991, pero fue publicada en 1995 por Sun Microsystems."

JavaScript[2] : "Es un lenguaje de programación que se utiliza para agregar interactividad y dinamismo a una página web. Con JavaScript, puedes crear efectos visuales, validar formularios, realizar peticiones a servidores y mucho más."

JWT[65]: "SON Web Token (abreviado JWT) es un estándar abierto basado en JSON propuesto por IETF (RFC 7519) para la creación de tokens de acceso que permiten la propagación de identidad y privilegios o claims en inglés."

Kotlin[3]: "Es un lenguaje de programación de Backend que se utiliza para el desarrollo de aplicaciones Android. Se está apoderando de Java para el desarrollo de aplicaciones Android, y su demanda aumenta día a día. Más del 60% de los desarrolladores de aplicaciones de Android utilizan Kotlin en el backend. Kotlin interopera completamente con Java y JVM."

Machine Learning[66]: "El aprendizaje automático (AA); también llamado automatizado, computacional o de máquinas (del inglés machine learning, ML), es el subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan".

Malware[16] o "software malicioso": "Es un término amplio que describe cualquier programa o código malicioso que es dañino para los sistemas."

NLP[67]: "El procesamiento de(l) lenguaje natural o de lengua(je)s naturales,12 abreviado PLN34 (o NLP por sus siglas en inglés), es un campo de las ciencias de la computación, de la inteligencia artificial y de la lingüística que estudia las interacciones entre las computadoras y el lenguaje humano, así como los detalles computacionales de las lenguas naturales."

NVD[68]: "La Base de Datos Nacional de Vulnerabilidad (NVD) es el repositorio del gobierno de los Estados Unidos de datos de gestión de vulnerabilidades basados en estándares representados mediante el Protocolo de Automatización de Contenido de Seguridad (SCAP). Estos datos permiten la automatización de la gestión de vulnerabilidades, la medición de la seguridad y el cumplimiento."

OAuth (abreviatura de "Autorización abierta")[69]: "es un estándar abierto para la delegación de acceso, comúnmente utilizado como una forma para que los usuarios de Internet otorguen a sitios web o aplicaciones acceso a su información en otros sitios web, pero sin darles las contraseñas."

OpenID[70]: "es un estándar de identificación digital descentralizado, con el que un usuario puede identificarse en una página web a través de una URL (o un XRI en la versión actual) y puede ser verificado por cualquier servidor que soporte el protocolo."

ORM[71]: "El mapeo relacional de objetos u ORM (también O/RM, sigla en inglés de object-relational mapping), es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia.".

OWASP[72]: "es un documento de los diez riesgos de seguridad más importantes en aplicaciones web según la Fundación OWASP (en inglés, Open Web Application Security Project; en español "Proyecto Abierto de Seguridad de Aplicaciones Web")"

PCI DSS[73]: "El Estándar de Seguridad de Datos para la Industria de Tarjeta de Pago (Payment Card Industry Data Security Standard) o PCI DSS fue desarrollado por un comité conformado por las compañías de tarjetas (débito y crédito) más importantes, comité denominado PCI SSC (Payment Card Industry Security Standards Council) como una guía que ayude a las organizaciones que procesan, almacenan y/o transmiten datos de tarjetahabientes (o titulares de tarjeta), a asegurar dichos datos, con el fin de evitar los fraudes que involucran tarjetas de pago débito y crédito"

Perl[3]: "Es otra tecnología de Backend de uso general que Larry Wall desarrolló hace 32 años. Según la encuesta para desarrolladores de Stack Overflow de 2020, Perl es la tecnología de pago más importante a nivel mundial. Por cierto, es importante saber que la misma encuesta también clasifica a Perl entre los tres primeros lenguajes más temidos."

PHP[3]: "Es una tecnología líder de scripting del lado del servidor que fue introducida por Rasmus Lerdorf en 1994. Esta tecnología de Backend de código abierto se usa comúnmente para sitios web. Alrededor del 79,1% de los sitios web en Internet utilizan PHP como tecnología del lado del servidor, según una encuesta reciente de W3Techs."

Pipeline[74]: "En informática, una tubería (pipeline o cauce) consiste en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo. Permiten la comunicación y sincronización entre procesos. Es común el uso de búfer de datos entre elementos consecutivos."

PKCS5Padding[75]: "EN criptografía, el relleno es cualquiera de una serie de prácticas distintas que incluyen la adición de datos al principio, el medio o el final de un mensaje antes del cifrado. En la criptografía clásica, el relleno puede incluir la adición de frases sin sentido a un mensaje para ocultar el hecho de que muchos mensajes terminan de manera predecible, por ejemplo, sinceramente la tuya."

Python[3]: "Es un lenguaje de programación multiuso líder creado por un programador holandés llamado Guido van Rossum en 1991. Con una competencia central de códigos concisos y legibles, Python apoya a los programadores de Backend para escribir líneas de código racionales y explícitas."

Redes Neuronales Convolucionales (CNNs) [76]: "Una red neuronal convolucional es un tipo de red neuronal artificial donde las neuronas artificiales, corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria (V1) de un cerebro biológico."

Redes Neuronales Recurrentes (RNNs) [77]: "Las redes neuronales recurrentes (RNNs) son una clase de algoritmos de aprendizaje profundo que se utilizan para procesar datos secuenciales, como el lenguaje natural, la música y las series temporales."

ReLU[78]: "En el contexto de las redes neuronales artificiales, la función de activación del rectificador o ReLU (unidad lineal rectificada es una función de activación definida como la parte positiva de su argumento."

Ruby[3]: "Es otra tecnología de Backend de código abierto que fue diseñada por el científico informático japonés Yukihiro Matsumoto en la década de 1990. Este lenguaje de programación tiene similitudes con Python, Java y Perl. Ruby se usa generalmente para el desarrollo de aplicaciones web y se considera el favorito para la creación de prototipos."

SAML[79]: "el Lenguaje de Marcado para Confirmaciones de Seguridad, conocido como SAML, es un estándar abierto que define un esquema XML para el intercambio de datos de autenticación y autorización."

Scala[3]: "Es un lenguaje de alto nivel que combina programación funcional y orientada a objetos para hacerlo más conciso. Puede crear sistemas de alto rendimiento accediendo a amplias bibliotecas de sus tiempos de ejecución de JVM y JavaScript. Scala es interoperable con Java porque se ejecuta en JVM y puede mezclar el código de ambos lenguajes para brindar una solución definitiva."

SQL[80]: "Por sus siglas en inglés Structured Query Language; en español lenguaje de consulta estructurada) es un lenguaje específico de dominio, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales."

SRS[81]: "La especificación de requisitos de software (ERS) es una descripción completa del comportamiento del sistema que se va a desarrollar. Incluye un conjunto de casos de uso que describe todas las interacciones que tendrán los usuarios con el software."

SSO (single sign-on, SSO) [82]: "El inicio de sesión único o inicio de sesión unificado es un procedimiento de autenticación que habilita a un usuario determinado para acceder a varios sistemas con una sola instancia de identificación. Su traducción literal es «autenticación única» o «validación única».“

SSRF[83]: "La falsación de solicitudes del lado del servidor (SSRF) es un tipo de exploit de seguridad informática en el que un atacante abusa de la funcionalidad de un servidor, lo que le hace acceder o manipular información en el ámbito de ese servidor que de otro modo no sería directamente accesible para el atacante."

TDD[84]: "Desarrollo guiado por pruebas de software, o Test-driven development (TDD) es una práctica de ingeniería de software que involucra otras dos prácticas: Escribir las pruebas primero (Test First Development) y Refactorización (Refactoring). Para escribir las pruebas generalmente se utilizan las pruebas unitarias (unit test en inglés)."

VPNs[86]: "Una red privada virtual (RPV) (en inglés, virtual private network, VPN) es una tecnología de red de ordenadores que permite una extensión segura de la red de área local (LAN) sobre una red pública o no controlada como Internet."

XSS[86]: "Una secuencia de comandos en sitios cruzados o Cross-site scripting (XSS por sus siglas en idioma inglés) es un tipo de ataque informático que puede permitir a una tercera persona inyectar scripts maliciosos en páginas web visitadas por un usuario."

Bibliografía

[1] MARTÍNEZ, J.G. INNOVACIÓN EN CIBERSEGURIDAD. ESTRATEGIA Y TENDENCIAS. *Gob.es* [en línea]. [consulta: 6 marzo 2024]. Disponible en: https://www.mintur.gob.es/Publicaciones/Publicacionesperiodicas/EconomiaIndustrial/RevistaEconomiaIndustrial/410/JUAN%20GONZÁLEZ%20MARTÍNEZ.pdf.

[2] VERGARA, S., 2023. Buenas prácticas de seguridad para front-end. Blog ITDO - *Agencia de desarrollo Web, APPs y Marketing en Barcelona* [en línea]. [consulta: 6 marzo 2024]. Disponible en: https://www.itdo.com/blog/buenas-practicas-de-seguridad-para-front-end/

[3] *Researchgate.net* [en línea]. [consulta: 6 marzo 2024]. Disponible en: https://www.researchgate.net/publication/366623170_Seguridad_informatica_metodologias_estandares_y_marco_de_gestion_en_un_enfoque_hacia_las_aplicaciones_web

[4] CUESTA, J., 2019. La primera barrera de seguridad comienza en el frontend. OpenExpo Europe 2024 [en línea]. [consulta: 6 marzo 2024]. Disponible en: https://openexpoeurope.com/es/la-primera-barrera-de-seguridad-comienza-en-el-frontend/

[5] BOE-A-2021-8806 Ley Orgánica 7/2021, de 26 de mayo, de protección de datos personales tratados para fines de prevención, detección, investigación y enjuiciamiento de infracciones penales y de ejecución de sanciones penales. Boe.es [en línea]. [consulta: 6 marzo 2024]. Disponible en: https://www.boe.es/buscar/act.php?id=BOE-A-2021-8806.

[6] *Mediummultimedia.com* [en línea], [sin fecha]. [consulta: 6 marzo 2024 b]. Disponible en: https://www.mediummultimedia.com/web/por-que-separar-frontend-y-backend/#google_vignette.

[7] JIMÉNEZ, A., 2023. Importancia de separar front end y back end en desarrollo web. El Blog de Python [en línea]. [consulta: 7 marzo 2024]. Disponible en: https://elblogpython.com/desarrollo-web/importancia-de-separar-front-end-y-back-end-en-desarrollo-web/.

[8] PRESTA, M., 2021. Las 10 mejores tecnologías de backend. Back4App Blog [en línea]. [consulta: 7 marzo 2024]. Disponible en: https://blog.back4app.com/es/las-10-mejores-tecnologias-de-backend/.

[9] WIKIPEDIA CONTRIBUTORS, [sin fecha]. Humanware. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Humanware&oldid=153942039.

[10] *Wikipedia.org* [en línea]. [consulta: 8 marzo 2024]. Disponible en: https://es.wikipedia.org/wiki/Sistema_informático.

[11] Wikipedia.org [en línea]. [consulta: 8 marzo 2024]. Disponible en: https://es.wikipedia.org/wiki/Lenguaje_de_programación.

[12] ¿Qué son los sistemas informáticos? Ui1.es [en línea]. [consulta: 8 marzo 2024]. Disponible en: https://www.ui1.es/blog-ui1/sistemas-informaticos-si-que-son-caracteristicas-y-tipos.

[13] MENDOZA, M.L., 2020. Qué es un lenguaje de programación. *Openwebinars.net [*en línea]. [consulta: 9 marzo 2024]. Disponible en: https://openwebinars.net/blog/que-es-un-lenguaje-de-programacion.

[14] WIKIPEDIA CONTRIBUTORS. Entorno de desarrollo integrado. Wikipedia, *The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Entorno_de_desarrollo_integrado&oldid=157623377.

[15] INFANTE, D.C.H., 2022. Seguridad en aplicaciones web: Qué es, cómo funciona y los mejores servicios. *Tutoriales Hostinger* [en línea]. [consulta: 10 marzo 2024]. Disponible en: https://www.hostinger.es/tutoriales/seguridad-en-aplicaciones-web.

[16] *Wikipedia.org* [en línea]. [consulta: 12 marzo 2024]. Disponible en: https://es.wikipedia.org/wiki/Tecnología_de_la_información.

[17] ¿Qué es el malware? Definición y cómo saber si está infectado. *Malwarebytes* [en línea], 2018. [consulta: 11 marzo 2024]. Disponible en: https://es.malwarebytes.com/malware/.

[18] *Cloudflare.com* [en línea]. [consulta: 12 marzo 2024]. Disponible en: https://www.cloudflare.com/es-es/learning/ddos/glossary/denial-of-service/.

[19] MURILLO, R., 2023. Patrones de arquitectura de software. Yapiko [en línea]. [consulta: 19 marzo 2024]. Disponible en: https://yapiko.com/es/blog/patrones-arquitectura-software/.

[20] Medium. *Medium* [en línea]. [consulta: 19 marzo 2024]. Disponible en: https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b.

[21] GOMEZ, E.E.P., 2023. 4 patrones de diseño que deberías saber para desarrollo web: Observador, Singleton, estrategia, y decorador. *freecodecamp.org* [en línea]. [consulta: 20 marzo 2024]. Disponible en: https://www.freecodecamp.org/espanol/news/4-patrones-de-diseno-que-deberias-saber-para-desarrollo-web-observador-singleton-estrategia-y-decorador/.

[22] *Huastecanetwork.com* [en línea]. [consulta: 20 marzo 2024]. Disponible en: https://huastecanetwork.com/los-patrones-de-diseno-en-la-seguridad-web/.

[23] IT OPERATIONS, 2024. ¿Cuáles son los patrones de diseño de seguridad más fáciles de usar para las operaciones de TI? *Linkedin.com* [en línea]. [consulta: 22 marzo 2024]. Disponible en: https://es.linkedin.com/advice/1/what-most-user-friendly-security-design-patterns-knntc?lang=es.

[24] PRACTICAS, 2023. Introducción a la seguridad en el desarrollo web. *MSuiteAgency* [en línea]. [consulta: 22 marzo 2024]. Disponible en: https://www.msuiteagency.com/introduccion-a-la-seguridad-en-el-desarrollo-web/.

[25] A04 Diseño Inseguro - OWASP Top 10:2021. *Owasp.org [*en línea]. [consulta: 24 marzo 2024]. Disponible en: https://owasp.org/Top10/es/A04_2021-Insecure_Design/.

[26] Intercambio de recursos de origen cruzado (CORS). *MDN Web Docs* [en línea]. [consulta: 24 marzo 2024]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/CORS.

[27] Autenticación JWT, qué es y cuándo usarla. *Ciberseguridad* [en línea], 2021. [consulta: 25 marzo 2024]. Disponible en: https://ciberseguridad.com/guias/prevencion-proteccion/autenticacion-jwt/.

[28] Frontend y backend: ¿Qué son y cuáles son sus diferencias? *Gluo [*en línea]. [consulta: 25 marzo 2024]. Disponible en: https://www.gluo.mx/blog/frontend-y-backend-que-son-y-cuales-son-sus-diferencias.

[29] GLAS, B., 2021. The release of the OWASP Top 10:2021. *Owasptopten.org* [en línea]. [consulta: 28 marzo 2024]. Disponible en: https://www.owasptopten.org/the-release-of-the-owasp-top-10-2021.

[30] Acerca de OWASP. *Owasp.org* [en línea]. [consulta: 28 marzo 2024]. Disponible en: https://owasp.org/Top10/es/A00-about-owasp/.

[31] Cómo utilizar el OWASP Top 10 como un estándar - OWASP Top 10:2021. *Owasp.org* [en línea]. [consulta: 28 marzo 2024]. Disponible en: https://owasp.org/Top10/es/A00_2021_How_to_use_the_OWASP_Top_10_as_a_standard/.

[32] Inicio - OWASP Top 10:2021. *Owasp.org* [en línea]. [consulta: 28 marzo 2024]. Disponible en: https://owasp.org/Top10/es/.

[33] A01 Pérdida de Control de Acceso - OWASP Top 10:2021. *Owasp.org* [en línea]. [consulta: 28 marzo 2024]. Disponible en: https://owasp.org/Top10/es/A01_2021-Broken_Access_Control/.

[34] A02 Fallas Criptográficas - OWASP Top 10:2021. Owasp.org [en línea]. [consulta: 28 marzo 2024]. Disponible en: https://owasp.org/Top10/es/A02_2021-Cryptographic_Failures/.

[35] A03 Inyección - OWASP Top 10:2021. Owasp.org [en línea]. [consulta: 28 marzo 2024]. Disponible en: https://owasp.org/Top10/es/A03_2021-Injection/.

[36] A04 Diseño Inseguro - OWASP Top 10:2021. Owasp.org [en línea]. [consulta: 28 marzo 2024]. Disponible en: https://owasp.org/Top10/es/A04_2021-Insecure_Design/.

[37] A05 Configuración de Seguridad Incorrecta - OWASP Top 10:2021. *Owasp.org* [en línea]. [consulta: 28 marzo 2024]. Disponible en: https://owasp.org/Top10/es/A05_2021-Security_Misconfiguration/.

[38] A06 Componentes Vulnerables y Desactualizados - OWASP Top 10:2021. Owasp.org [en línea] [consulta: 28 marzo 2024]. Disponible en: https://owasp.org/Top10/es/A06_2021-Vulnerable_and_Outdated_Components/.

[39] A07 Fallas de Identificación y Autenticación - OWASP Top 10:2021. *Owasp.org* [en línea]. [consulta: 28 marzo 2024]. Disponible en: https://owasp.org/Top10/es/A07_2021-Identification_and_Authentication_Failures/.

[40] A08 Fallas en el Software y en la Integridad de los Datos - OWASP Top 10:2021. Owasp.org [en línea]. [consulta: 28 marzo 2024]. Disponible en: https://owasp.org/Top10/es/A08_2021-Software_and_Data_Integrity_Failures/.

[41] A09 Fallas en el Registro y Monitoreo - OWASP Top 10:2021. Owasp.org [en línea]. [consulta: 28 marzo 2024]. Disponible en: https://owasp.org/Top10/es/A09_2021-Security_Logging_and_Monitoring_Failures/.

[42] A10 Falsificación de Solicitud del Lado del Servidor (SSRF) - OWASP Top 10:2021. Owasp.org [en línea]. [consulta: 28 marzo 2024]. Disponible en: https://owasp.org/Top10/es/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/.

[43] Desarrollo de software seguro, ese gran desconocido. Grupo Oesía [en línea], 2022. [consulta: 31 marzo 2024]. Disponible en: https://grupooesia.com/insight/desarrollo-de-software-seguro-ese-gran-desconocido/.

[44] *Owasp.org* [en línea]. [consulta: 30 marzo 2024]. Disponible en: https://owasp.org/www-pdf-archive/Guía_de_pruebas_de_OWASP_ver_3.0.pdf.

[45] *Cni.es* [en línea]. [consulta: 30 marzo 2024 a]. Disponible en: https://www.ccn-cert.cni.es/es/series-ccn-stic/800-guia-esquema-nacional-de-seguridad/522-ccn-stic-812-seguridad-en-entornos-y-app-web/file?format=html.

[46] Seguridad en el ciclo de vida de desarrollo del software. *Redhat.com* [en línea], [sin fecha]. [consulta: 1 abril 2024]. Disponible en: https://www.redhat.com/es/topics/security/software-development-lifecycle-security.

[47] *Microsoft.com* [en línea]. [consulta: 4 abril 2024]. Disponible en: https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

[48] ANALYSIS, O., [sin fecha]. Charu C. Aggarwal. Sadbhavnapublications.org [en línea]. [consulta: 6 abril 2024]. Disponible en: https://sadbhavnapublications.org/research-enrichment-material/2-Statistical-Books/Outlier-Analysis.pdf.

[49] BIRD, S., KLEIN, E. y LOPER, E., 2009. Natural Language Processing with Python: *Analyzing text with the Natural Language Toolkit. S.l.:* «O'Reilly Media, Inc.» ISBN 9780596555719.

[50] Alvarestech.com [en línea]. [consulta: 12 abril 2024]. Disponible en: http://alvarestech.com/temp/deep/Deep%20Learning%20by%20Ian%20Goodfellow,%20Yoshua%20Bengio,%20Aaron%20Courville%20(z-lib.org).pdf

[51] NVD - CVEs and the NVD process. Nist.gov [en línea]. [consulta: 17 abril 2024]. Disponible en: https://nvd.nist.gov/general/cve-process.

[52] WIKIPEDIA CONTRIBUTORS, 2024. Advanced Encryption Standard. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Advanced_Encryption_Standard&oldid=1223003853.

[53] WIKIPEDIA CONTRIBUTORS, 2024. bcrypt. Wikipedia, *The Free Encyclopedia* [en línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Bcrypt&oldid=1222801891.

[54] WIKIPEDIA CONTRIBUTORS, [sin fecha]. CI/CD. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=CI/CD&oldid=152668077.

[55] WIKIPEDIA CONTRIBUTORS. Intercambio de recursos de origen cruzado. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Intercambio_de_recursos_de_origen_cruzado&oldid=136555200.

[56] WIKIPEDIA CONTRIBUTORS. Cross-site request forgery. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Cross-site_request_forgery&oldid=157832693.

[57] WIKIPEDIA CONTRIBUTORS. Common Vulnerabilities and Exposures. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Common_Vulnerabilities_and_Exposures&oldid=154159952.

[58] *Wikipedia.org* [en línea]. Disponible en: https://es.wikipedia.org/wiki/DBM_(informática).

[59] WIKIPEDIA CONTRIBUTORS. Aprendizaje profundo. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Aprendizaje_profundo&oldid=159924406.

[60] WIKIPEDIA CONTRIBUTORS, [sin fecha]. DevOps. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=DevOps&oldid=159568356.

[61] *Wikipedia.org* [en línea]. Disponible en: https://es.wikipedia.org/wiki/Cortafuegos_(informática).

[62] *Wikipedia.org* [en línea]. Disponible en: https://es.wikipedia.org/wiki/Reglamento_General_de_Protección_de_Datos#:~:text=El%20RGPD%20(GDPR%20por%20su,Fundamentales%20de%20la%20Unión%20Europea.

[63] WIKIPEDIA CONTRIBUTORS. Protocolo de transferencia de hipertexto. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Protocolo_de_transferencia_de_hipertexto&oldid=159523514.

[64] WIKIPEDIA CONTRIBUTORS. Protocolo seguro de transferencia de hipertexto. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Protocolo_seguro_de_transferencia_de_hipertexto&oldid=158297099.

[65] WIKIPEDIA CONTRIBUTORS. JSON Web Token. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=JSON_Web_Token&oldid=159811805.

[66] Wikipedia.org [en línea]. Disponible en: https://es.wikipedia.org/wiki/Aprendizaje_automático.

[67] WIKIPEDIA CONTRIBUTORS. Procesamiento de lenguajes naturales. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Procesamiento_de_lenguajes_naturales&oldid=159774248.

[68] WIKIPEDIA CONTRIBUTORS, 2024. National Vulnerability Database. Wikipedia, *The Free Encyclopedia* [en línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=National_Vulnerability_Database&oldid=1222884090.

[69] WIKIPEDIA CONTRIBUTORS, 2024. OAuth. *Wikipedia*, The Free Encyclopedia [en línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=OAuth&oldid=1219544559.

[70] WIKIPEDIA CONTRIBUTORS. OpenID. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=OpenID&oldid=157562093.

[71] WIKIPEDIA CONTRIBUTORS. Mapeo relacional de objetos. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Mapeo_relacional_de_objetos&oldid=159504674.

[72] WIKIPEDIA CONTRIBUTOR. OWASP Top 10. *Wikipedia*, The Free Encyclopedia [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=OWASP_Top_10&oldid=155122864.

[73] WIKIPEDIA CONTRIBUTORS. PCI DSS. Wikipedia, *The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=PCI_DSS&oldid=157900536.

[74] *Wikipedia.org* [en línea] Disponible en: https://es.wikipedia.org/wiki/Tubería_(informática).

[75] WIKIPEDIA CONTRIBUTORS, 2024. Padding (cryptography). Wikipedia, *The Free Encyclopedia* [en línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Padding_(cryptography)&oldid=1207389258.

[76] WIKIPEDIA CONTRIBUTORS, [sin fecha]. Red neuronal convolucional. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Red_neuronal_convolucional&oldid=158743241.

[77] WIKIPEDIA CONTRIBUTORS. Redes neuronales recurrentes. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Redes_neuronales_recurrentes&oldid=159512598.

[78] WIKIPEDIA CONTRIBUTORS, [sin fecha]. Rectificador (redes neuronales). *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Rectificador_(redes_neuronales)&oldid=150909789.

[79] WIKIPEDIA CONTRIBUTORS. Security assertion markup language. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Security_Assertion_Markup_Language&oldid=157839264.

[80] WIKIPEDIA CONTRIBUTORS. SQL. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=SQL&oldid=160021288.

[81] *Wikipedia.org* [en línea]. Disponible en: https://es.wikipedia.org/wiki/Especificación_de_requisitos_de_software

[82] WIKIPEDIA CONTRIBUTORS. Single Sign-On. Wikipedia, *The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Single_Sign-On&oldid=156580672.

[83] WIKIPEDIA CONTRIBUTORS, 2024. Server-side request forgery. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Server-side_request_forgery&oldid=1222473938.

[84] WIKIPEDIA CONTRIBUTORS. Desarrollo guiado por pruebas. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Desarrollo_guiado_por_pruebas&oldid=126589567.

[85] WIKIPEDIA CONTRIBUTORS. Red neuronal convolucional. Wikipedia, The Free Encyclopedia [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Red_neuronal_convolucional&oldid=158743241.

[86] WIKIPEDIA CONTRIBUTORS. Cross-site scripting. *Wikipedia, The Free Encyclopedia* [en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Cross-site_scripting&oldid=159770516.

Anexos

Anexo 1: Clase "**PasswordUtil**".

```
package com.market.utils.security;
import javax.crypto.*;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.security.SecureRandom;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
public class PasswordUtil {
    private static final int KEY_SIZE = 256; // Tamaño de la clave AES en bits
    private static final int IV_SIZE = 16; // Tamaño del IV en bytes
    /**
     * Genera una clave AES de forma segura.
     * @return Clave AES codificada en Base64.
     * @throws NoSuchAlgorithmException si el algoritmo AES no está disponible.
     */
    public static String generateAESStrKey() throws NoSuchAlgorithmException {
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(KEY_SIZE);
        SecretKey secretKey = keyGen.generateKey();
        return Base64.getEncoder().encodeToString(secretKey.getEncoded());
    }
    /**
     * Genera un vector de inicialización (IV) de forma segura.
     * @return IV codificado en Base64.
     */
    public static String generateIv() {
        byte[] iv = new byte[IV_SIZE];
        new SecureRandom().nextBytes(iv);
        return Base64.getEncoder().encodeToString(iv);
    }
    /**
     * Cifra los datos utilizando AES en modo CBC con padding PKCS5Padding.
     * @param content Datos a cifrar.
     * @param secretKeyStr Clave AES codificada en Base64.
     * @param ivStr IV codificado en Base64.
     * @return Datos cifrados codificados en Base64.
     * @throws Exception Si ocurre un error durante el cifrado.
     */
    public static String encryptAES(byte[] content, String secretKeyStr, String ivStr) throws Exception {
        SecretKeySpec secretKeySpec = new SecretKeySpec(Base64.getDecoder().decode(secretKeyStr), "AES");
        IvParameterSpec ivParameterSpec = new IvParameterSpec(Base64.getDecoder().decode(ivStr));
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, ivParameterSpec);
        return Base64.getEncoder().encodeToString(cipher.doFinal(content));
    }
    /**
     * Descifra los datos cifrados utilizando AES en modo CBC con padding PKCS5Padding.
     * @param content Datos cifrados codificados en Base64.
     * @param secretKeyStr Clave AES codificada en Base64.
     * @param ivStr IV codificado en Base64.
     * @return Datos descifrados.
     * @throws Exception Si ocurre un error durante el descifrado.
     */
    public static String decryptAES(String content, String secretKeyStr, String ivStr) throws Exception {
        byte[] contentDecByBase64 = Base64.getDecoder().decode(content);
        SecretKeySpec secretKeySpec = new SecretKeySpec(Base64.getDecoder().decode(secretKeyStr), "AES");
        IvParameterSpec ivParameterSpec = new IvParameterSpec(Base64.getDecoder().decode(ivStr));
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, ivParameterSpec);
        return new String(cipher.doFinal(contentDecByBase64));
    }
}
```

Anexo 2: Clase "**MainSecurity**".

```
package com.market.utils.security;
import com.market.controller.action.useraccount.UserDetailsServiceImpl;
import com.market.utils.security.jwt.JwtEntryPoint;
import com.market.utils.security.jwt.JwtTokenFilter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class MainSecurity extends WebSecurityConfigurerAdapter {
    @Autowired
    UserDetailsServiceImpl userDetailsService;
    @Autowired
    JwtEntryPoint jwtEntryPoint;
    @Bean
    public JwtTokenFilter jwtTokenFilter(){
        return new JwtTokenFilter();
    }
    @Bean
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
    }
    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
    @Override
    protected AuthenticationManager authenticationManager() throws Exception {
        return super.authenticationManager();
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().and().csrf().disable()
                .authorizeRequests()
                .antMatchers().permitAll()
                .anyRequest().permitAll()
                .and()
                .exceptionHandling().authenticationEntryPoint(jwtEntryPoint)
                .and()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        http.addFilterBefore(jwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);
    }
}
```

Anexo 3: Clase "**JwtEntryPoint**".

```
package com.market.utils.security.jwt;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

@Component
public class JwtEntryPoint implements AuthenticationEntryPoint {

    private final static Logger logger = LoggerFactory.getLogger(JwtEntryPoint.class);
    @Override
    public void commence(HttpServletRequest req, HttpServletResponse res, AuthenticationException e) throws
IOException, ServletException {
        logger.error("Unauthorized error: {}", e.getMessage());
        logger.error("Unauthorized attempt to access {} {}", req.getMethod(), req.getRequestURI());
        // Prepare JSON response
        Map<String, Object> response = new HashMap<>();
        response.put("timestamp", System.currentTimeMillis());
        response.put("status", HttpServletResponse.SC_UNAUTHORIZED);
        response.put("error", "Unauthorized");
        response.put("message", e.getMessage());
        response.put("path", req.getRequestURI());
        res.setContentType("application/json;charset=UTF-8");
        res.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
        // Add some additional security headers
        res.addHeader("X-Content-Type-Options", "nosniff");
        res.addHeader("X-Frame-Options", "DENY");
        res.addHeader("X-XSS-Protection", "1; mode=block");
        // Send the error response in JSON format
        new ObjectMapper().writeValue(res.getOutputStream(), response);
    }
}
```

Anexo 4: Clase "**JwtProvider**".

```
package com.market.utils.security.jwt;
import com.market.model.entity.useraccount.UserMain;
import io.jsonwebtoken.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Component;
import java.util.Date;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.env.Environment;

@Component
public class JwtProvider {

    @Autowired
    Environment env;
    public String generateToken(Authentication authentication){
        String secret = env.getProperty("jwt.secret");
        int expiration = Integer.parseInt(env.getProperty("jwt.expiration"));
        UserMain usuarioPrincipal = (UserMain) authentication.getPrincipal();
        return Jwts.builder().setSubject(usuarioPrincipal.getUsername())
                .setIssuedAt(new Date())
                .setExpiration(new Date(new Date().getTime() + expiration * 1000))
                .signWith(SignatureAlgorithm.HS512, secret)
                .compact();
    }
    public String getNameUserFromToken(String token){
        String secret = env.getProperty("jwt.secret");
        return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody().getSubject();
    }
    public boolean validateToken(String token){
        Logger logger = LoggerFactory.getLogger(JwtProvider.class);
        String secret = env.getProperty("jwt.secret");
        try {
            Jwts.parser().setSigningKey(secret).parseClaimsJws(token);
            return true;
        }catch (MalformedJwtException e){
            logger.error("Malformed Token");
        }catch (UnsupportedJwtException e){
            logger.error("Token not supported");
        }catch (ExpiredJwtException e){
            logger.error("Expired token");
        }catch (IllegalArgumentException e){
            logger.error("Empty token");
        }catch (SignatureException e){
            logger.error("Token signing failure");
        }
        return false;
    }
}
```

Anexo 5: Clase "**JwtTokenFilter**".

```
package com.market.utils.security.jwt;
import com.market.controller.action.useraccount.UserDetailsServiceImpl;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.filter.OncePerRequestFilter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import org.springframework.security.core.userdetails.UsernameNotFoundException;

public class JwtTokenFilter extends OncePerRequestFilter {
   private final static Logger logger1 = LoggerFactory.getLogger(JwtTokenFilter.class);
   @Autowired
   JwtProvider jwtProvider;
   @Autowired
   UserDetailsServiceImpl userDetailsService;
   @Override
   protected void doFilterInternal(HttpServletRequest req, HttpServletResponse res, FilterChain filterChain)
throws ServletException, IOException {
      try {
         String token = getToken(req);
         if(token != null && jwtProvider.validateToken(token)){
            String nombreUsuario = jwtProvider.getNameUserFromToken(token);
            UserDetails userDetails = userDetailsService.loadUserByUsername(nombreUsuario);
            UsernamePasswordAuthenticationToken auth =
                  new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
            SecurityContextHolder.getContext().setAuthentication(auth);
         }
      } catch (UsernameNotFoundException e){
         logger1.error("Token filter failure " + e.getMessage());
      }
      filterChain.doFilter(req, res);
   }
   private String getToken(HttpServletRequest request){
      String header = request.getHeader("Authorization");
      if(header != null && header.startsWith("Bearer"))
         return header.replace("Bearer ", "");
      return null;
   }
}
```

Anexo 6: “Cifrado y Descifrado AES con React- Native”.

```
import {
    AESKEY,
    IV
} from "../../types/Types";
let CryptJS = require("crypto-js");
let UUID = require("uuid");
export var AESUtil = {
    /**
     * Cifrado AES
     * @param _contenido contenido a cifrar
     * @param _key aesKey,
     * @param _iv vector de inicialización
     * @return devuelve texto sin formato después del procesamiento BASE64
     */
    encrypt: function (_content) {
        let content = CryptJS.enc.Utf8.parse(_content);
        let aesKey = CryptJS.enc.Utf8.parse(AESKEY);
        let iv = CryptJS.enc.Utf8.parse(IV);
        // cifrado
        let encrypted = CryptJS.AES.encrypt(content, aesKey, {
            iv: iv,
            mode: CryptJS.mode.CBC,
            padding: CryptJS.pad.Pkcs7
        })
        return CryptJS.enc.Base64.stringify(encrypted.ciphertext);
    },
    /**
     * Descifrado AES
     * @param: contenido a descifrar
     * @param: clave AES para descifrar
     * @param: vector de inicialización
     * @return devuelve texto sin formato después del procesamiento UTF-8
     */
    decrypt: function (_content) {
        let aesKey = CryptJS.enc.Utf8.parse(AESKEY);
        let iv = CryptJS.enc.Utf8.parse(IV);
        // descifrar
        let decrypted = CryptJS.AES.decrypt(_content, aesKey, {
            iv: iv,
```

```
/**
     * Obtener clave AES
     * @ devuelve la clave AES de 32 bytes
     */
    getAesKey: function () {
        let uuid = UUID.v1();
        let aeskey = CryptJS.enc.Utf8.parse(uuid);
        aeskey = CryptJS.enc.Base64.stringify(aeskey).substring(2, 34);
        return aeskey;
    }
    ,
    /**
     * Obtener vector de inicialización
     * @ devuelve el vector de inicialización de 16 bytes
     */
    getIv: function () {
        let uuid = UUID.v1();
```

Anexo 7: Clase "**AuthController**".

```
package com.market.crmmarket.controller.service.useraccount;
import com.market.common.security.jwt.JwtProvider;
import com.market.common.security.jwt.PasswordUtil;
import com.market.crmmarket.controller.DTO.security.JwtDto;
import com.market.crmmarket.controller.DTO.security.JwtUserAccountResponseDTO;
import com.market.crmmarket.controller.DTO.security.Message;
import com.market.crmmarket.controller.DTO.useraccount.UserAccountResponseDTO;
import com.market.crmmarket.controller.action.useraccount.UserAccountAction;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.env.Environment;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.*;
/**
 * Controlador REST para la autenticación de usuarios.
 * Este controlador maneja las solicitudes de autenticación y genera tokens JWT para los usuarios autenticados.
* @RestController Anota la clase como un controlador donde cada método devuelve un objeto de dominio
* @RequestMapping(value = "/auth") Mapea las solicitudes a /auth a los métodos de este controlador.
 * @CrossOrigin(origins = "*", allowedHeaders = "*") Permite solicitudes CORS de cualquier origen y cualquier
cabecera.
 */
@RestController
@RequestMapping(value = "/auth")
@CrossOrigin(origins = "*", allowedHeaders = "*")
public class AuthController {
@Autowired
AuthenticationManager authenticationManager;
@Autowired
UserAccountAction userAccountAction;
```

```
@Autowired
Environment env;
@Autowired
JwtProvider jwtProvider;
/**
* Autentica a un usuario y genera un token JWT
* Este método recibe una cadena cifrada que contiene el nombre de usuario y la contraseña,
* la descifra y utiliza estos detalles para autenticar al usuario. Si la autenticación es exitosa,
* se genera un token JWT y se devuelve junto con algunos detalles del usuario en un objeto JwtDto.
* @param param La cadena cifrada con el nombre de usuario y la contraseña.
* @return ResponseEntity<JwtDto> Un objeto ResponseEntity que contiene un objeto JwtDto con el token JWT
y los
* detalles del usuario, o un mensaje de error si la autenticación falla.
* @throws Exception Si ocurre un error durante el proceso de descifrado o autenticación.
*/
@GetMapping("/loginhead/{param}")
public ResponseEntity<JwtDto> loginhead(@PathVariable("param") String param) throws Exception {
        String username="";
        String password="";
        String decryptedParam;
        try {
            decryptedParam = PasswordUtil.decryptAES(param, env.getProperty("mailGun.aeskey"),
env.getProperty("mailGun.iv"));
        } catch (IllegalArgumentException ex) {
            return new ResponseEntity(new Message("permisos denegados"), HttpStatus.UNAUTHORIZED);
        }
        String[] paramSeparated = decryptedParam.split("//");
        if (paramSeparated.length==2){
            username=paramSeparated[0];
            password=paramSeparated[1];
        } else{ if (paramSeparated.length==1){username=paramSeparated[0];} }
        try {
        Authentication authentication =
        authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));
        SecurityContextHolder.getContext().setAuthentication(authentication);
        String jwt = jwtProvider.generateToken(authentication);
        UserDetails userDetails = (UserDetails)authentication.getPrincipal();
        JwtDto jwtDto = new JwtDto(jwt, userDetails.getUsername(), userDetails.getAuthorities());
        if (jwtDto.getToken().isEmpty() || jwtDto.getToken().isBlank()){
            return new ResponseEntity(new Message("Server connection error"), HttpStatus.BAD_REQUEST);
        } else {
            UserAccountResponseDTO userAccountResponseDTO =
            userAccountAction.getUserAccountFindbyUsername(username);
        if (userAccountResponseDTO==null){
            return new ResponseEntity(new Message("Server connection error"), HttpStatus.BAD_REQUEST);
        } else {
            JwtUserAccountResponseDTO jwtUserAccountResponseDTO =
                        new JwtUserAccountResponseDTO(
                          userAccountResponseDTO.getId(),
                          userAccountResponseDTO.getUsername(),
                          userAccountResponseDTO.getPassword(),
                          userAccountResponseDTO.getFirstName(),
                          userAccountResponseDTO.getLastName1(),
                          userAccountResponseDTO.getLastName2(),
                          userAccountResponseDTO.getEmail(),
                          userAccountResponseDTO.getEnabled(),
                          userAccountResponseDTO.getIsAccountNonLocked(),
                          userAccountResponseDTO.getAliasuser(),
                          userAccountResponseDTO.getCreatedate(),
                          userAccountResponseDTO.getRoles(),
                          userAccountResponseDTO.getResponsefails(),
                          jwtDto.getToken(),
                          jwtDto.getAuthorities());
            return new ResponseEntity(jwtUserAccountResponseDTO, HttpStatus.OK);
            }}
        } catch (AuthenticationException e) {
        return new ResponseEntity(new Message("Permissions denied"), HttpStatus.UNAUTHORIZED);
        }}
}
```

Anexo 8: Clase “MainSecurity”.

```
package com.market.common.security;
import com.market.common.security.jwt.JwtEntryPoint;
import com.market.common.security.jwt.JwtTokenFilter;
import com.market.crmmarket.controller.service.useraccount.UserDetailsServiceImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class MainSecurity extends WebSecurityConfigurerAdapter {
@Autowired
UserDetailsServiceImpl userDetailsService;
@Autowired
JwtEntryPoint jwtEntryPoint;
@Bean
public JwtTokenFilter jwtTokenFilter(){
    return new JwtTokenFilter();
}
@Bean
public PasswordEncoder passwordEncoder(){
    return new BCryptPasswordEncoder();
}
@Bean
@Override
public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
}
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
}
@Override
protected AuthenticationManager authenticationManager() throws Exception {
    return super.authenticationManager();
}
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.cors().and().csrf().disable()
            .authorizeRequests()
            .antMatchers("/auth/**").permitAll()
            .anyRequest().authenticated()
            .and()
            .exceptionHandling().authenticationEntryPoint(jwtEntryPoint)
            .and()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
             http.addFilterBefore(jwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);
    }
}
```