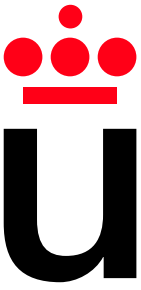


Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

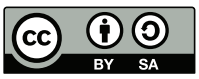
Grado en Ingeniería de la Ciberseguridad

Curso 2024-2025

Trabajo Fin de Grado

EVALUACIÓN DE LA EFICACIA DE MODELOS DE LENGUAJE DE GRAN TAMAÑO EN AUDITORÍAS DE SEGURIDAD WEB MEDIANTE UNA PLATAFORMA AUTOMATIZADA DE ESCANEO

Autor: Gabriel Izquierdo González
Tutor: Jesús María González Barahona
Cotutor: Michel Maes Bermejo



Agradecimientos

Quiero expresar mi más profundo agradecimiento a toda mi familia y amigos, por su apoyo incondicional y constante esfuerzo por hacer posible que hoy esté aquí cerrando esta etapa. En especial a mis padres y hermana, cuyo cariño ha sido pilar fundamentales en mi camino. Gracias también a María, mi compañera, por estar siempre a mi lado durante todo el proceso de realización de este trabajo. Quiero dar las gracias a esos profesores que no dejaron de creer en mí, especialmente a César, quien en mis momentos más difíciles, sobre todo de salud, nunca tiró la toalla y siempre tuvo palabras de ánimo y paciencia. Por último, quiero expresar mi gratitud a mis tutores, Jesús y Michel: a Jesús, por brindarme su confianza y compartir conmigo su sabiduría; a Michel, por su desempeño y dedicación durante todo este proyecto.

Gracias a todos por acompañarme en este recorrido y ayudarme a llegar hasta aquí.

Resumen

Este Trabajo Final de Grado se centra en el desarrollo y el diseño de una plataforma web cuyo principal objetivo es analizar y evaluar el comportamiento de los grandes modelos de lenguaje (LLM) en el ámbito de las auditorías de seguridad web. La herramienta integra tecnologías, como OWASP ZAP para el escaneo de vulnerabilidades, junto con un LLM que permite generar resúmenes y documentos personalizados, comparación de resultados y búsquedas de falsos positivos.

La plataforma está implementada principalmente en Python y Flask. Permite al usuario ejecutar y programar escáneres de seguridad, ser notificado una vez terminado el escaner, ver los resultados a través de gráficas y un chatbot inteligente y recibir explicaciones y recomendaciones sobre las vulnerabilidades detectadas. Además, se ha desarrollado un motor de generación automática de informes en formato DOCX, en el que el LLM se encarga de redactar descripciones, evaluar riesgos y proponer medidas de mitigación de forma contextualizada.

La arquitectura de la plataforma se ha desarrollado de tal manera que pueda ser modular y escalable. Se usa PostgreSQL para la persistencia de los datos, Docker para la contenedorización y el despliegue, y APScheduler para la programación de escaneres. Y además, se ha incluido la generación de informes a partir de archivos JSON generados por Owasp Zap.

Entre algunos de los logros del proyecto, se encuentran la capacidad del LLM para traducir información técnica en explicaciones mas sencillas, así como su papel en la generacion documentación, y el análisis de resultados. Sin embargo, también se han identificado limitaciones en la detección de falsos positivos, lo que abre la puerta a futuras mejoras y líneas de investigación.

Palabras clave:

- Python
- Ciberseguridad
- OWASP ZAP
- Flask

- Inteligencia Artificial
- Chatbot
- Seguridad web
- Automatización
- Docker
- PostgreSQL
- LangGraph
- SQLAlchemy
- DOCX
- LLM (Large Language Models)
- APScheduler
- API
- Portabilidad
- Interfaz
- Lenguaje Natural
- SendGrid

Índice de contenidos

Índice de figuras

1 Introducción

Durante los últimos años, el desarrollo de aplicaciones web ha crecido rápidamente, y con él han surgido nuevos desafíos relacionados con la seguridad informática. Como estudiante apasionado por la tecnología, he podido observar personalmente la importancia que tiene protegerse frente a vulnerabilidades, algo fundamental en cualquier proyecto digital, sin importar su tamaño o complejidad.

Este Trabajo de Fin de Grado nace precisamente de mi interés por descubrir formas más sencillas y eficaces para mejorar la seguridad web. No se trata únicamente de aplicar las herramientas ya establecidas en el sector, sino también de explorar cómo tecnologías emergentes, como la inteligencia artificial, pueden revolucionar nuestra manera de identificar y prevenir riesgos informáticos.

Mi objetivo es presentar una visión práctica e innovadora que pueda facilitar el trabajo diario tanto a profesionales del sector como a usuarios que no cuentan con conocimientos técnicos avanzados. De este modo, busco contribuir a crear una cultura de ciberseguridad más abierta y colaborativa, accesible para todos.

1.1. Contexto

En el actual contexto tecnológico, donde la seguridad informática se ha consolidado como un pilar fundamental en el desarrollo y mantenimiento de aplicaciones, la identificación y gestión eficiente de vulnerabilidades web resulta prioritaria para organizaciones y desarrolladores. Herramientas como OWASP Zed Attack Proxy (OWASP ZAP) destacan en la detección y evaluación de vulnerabilidades, siendo ampliamente reconocidas por su eficacia y versatilidad en el ámbito de la

ciberseguridad.

La reciente irrupción de la inteligencia artificial, y especialmente de los modelos de lenguaje de gran tamaño (LLM, por sus siglas en inglés, Large Language Model), abre nuevas posibilidades para transformar la interacción entre los usuarios y las herramientas de seguridad. Los LLM permiten automatizar tareas complejas y ofrecer una experiencia conversacional, intuitiva y adaptada a las necesidades de cada usuario. Gracias a estas capacidades, facilitan la comprensión y gestión de los resultados de los análisis de seguridad incluso para perfiles no especializados. Por ejemplo, un usuario puede consultar al chatbot sobre una vulnerabilidad detectada y recibir una explicación clara junto con recomendaciones personalizadas, sin necesidad de conocimientos técnicos avanzados.

Este Trabajo de Fin de Grado se sitúa en una línea exploratoria, proponiendo el desarrollo de una plataforma web que integra OWASP ZAP con un LLM avanzado, permitiendo al usuario interactuar con el sistema a través de un chatbot conversacional. Esta integración tiene como objetivos facilitar la ejecución y programación de escaneos de seguridad, permitir consultas directas sobre la base de datos de resultados, obtener resúmenes automáticos de los análisis realizados y recibir explicaciones detalladas sobre las vulnerabilidades detectadas, así como recomendaciones personalizadas de mitigación. Todo ello se realiza mediante una interfaz web accesible, orientada a democratizar el acceso a la ciberseguridad y reducir las barreras de entrada para todo tipo de usuarios.

El carácter exploratorio del proyecto reside en la investigación y evaluación del potencial de los LLM para asistir y potenciar el trabajo de auditoría de seguridad web, valorando su capacidad para interpretar, resumir y explicar información técnica de forma comprensible y útil. Asimismo, se analiza cómo la interacción conversacional puede mejorar la experiencia de usuario, agilizar la toma de decisiones y fomentar una cultura de seguridad más proactiva y transversal en los equipos de desarrollo.

En definitiva, este TFG busca no solo aportar una solución práctica a la gestión y explotación de los resultados de OWASP ZAP, sino también explorar el impacto real que la inteligencia artificial puede tener en la evolución de las herramientas de ciberseguridad, sentando las bases para futuras investigaciones y desarrollos en este ámbito.

2

Estado del Arte

El desarrollo de aplicaciones para las auditorías de seguridad web ha experimentado una notable transformación en los últimos años, especialmente gracias a la aparición de la inteligencia artificial y los modelos de lenguaje grande . Tradicionalmente, herramientas como OWASP ZAP han sido siempre un punto de referencia para el análisis dinámico de vulnerabilidades web, permitiendo a los equipos tanto de desarrollo como de seguridad identificar los posibles riesgos en las aplicaciones web de forma automatizada.

2.1. Integración de la inteligencia artificial en OWASP ZAP.

En la comunidad ZAP ya se han comentando y se han propuesto ideas para combinar la potencia de modelos de lenguaje como GPT con la propia herramienta, con el objetivo de mejorar la calidad de las recomendaciones y reducir los falsos positivos [1]. Aunque estas iniciativas todavía se encuentran en una fase inicial, demuestran que ya existe un interés por automatizar y enriquecer la experiencia del usuario. De hecho, el propio equipo de ZAP anunció en octubre de 2024 avances en la integración de las API de LLM, con especial atención a la identificación automática de falsos positivos y la mejora de la interpretación de los resultados [2]. Sin embargo, aunque estos desarrollos están en curso aún no ofrecen una solución completa.

2.2. Chatbots de seguridad basados en inteligencia artificial.

En el campo de los asistentes conversacionales, proyectos como VulBot han explorado el uso de chatbots para escanear redes y aplicaciones en busca de vulnerabilidades, utilizando técnicas de procesamiento del lenguaje natural y API especializadas [3]. Aunque comparten la idea de facilitar la interacción, estas soluciones no se integran directamente con ZAP ni generan informes enriquecidos utilizando LLM. Por otro lado, iniciativas como LangGraph Chatbot Agent han demostrado la viabilidad de utilizar arquitecturas multiagente y flujos conversacionales estructurados para gestionar tareas complejas, aunque su aplicación directa a la ciberseguridad sigue siendo limitada.

2.3. Generación automatizada de informes con LLM

En cuanto a la generación de informes, existen herramientas como LLM Report Generator que utilizan ChatGPT para crear informes a partir de datos estructurados como CSV o imágenes [4]. Sin embargo, estas soluciones suelen estar orientadas a la presentación de resultados y carecen de integración con motores de escaneo como ZAP, así como de la capacidad de contextualizar las vulnerabilidades en lenguaje natural. Otras propuestas, como FuzzAI, han complementado ZAP con cargas útiles específicas para evaluar la robustez de los propios modelos de lenguaje frente a los ataques, pero no abordan la generación de informes explicativos para los usuarios finales [2].

2.4. Contribución y diferenciación del proyecto

En este contexto, la herramienta desarrollada en el TFG destaca por combinar la potencia de OWASP ZAP con un chatbot conversacional basado en LLM, capaz de contextualizar y explicar los resultados de los escaneos en lenguaje natural. Además, la capacidad de generar informes completos y personalizados en formato DOCX, junto con la posibilidad de programar auditorías y procesar resultados externos, sitúa a esta solución a la vanguardia de la accesibilidad y la automatización en ciberseguridad. En la actualidad, no existe ninguna herramienta pública que integre todas estas características y esté dirigida tanto a usuarios expertos como a perfiles menos técnicos.

3 Objetivos

El presente Trabajo de Fin de Grado tiene como propósito fundamental explorar y demostrar el potencial de la integración de modelos de lenguaje de gran tamaño (LLM) en el ámbito de la seguridad web, concretamente en la interacción, gestión y explotación de los resultados obtenidos mediante OWASP ZAP. Para ello, se plantea el desarrollo de una plataforma web que no solo facilite la ejecución y programación de escaneos de seguridad, sino que también permita al usuario interactuar de forma natural y conversacional con los datos y resultados, así como automatizar la generación de reportes técnicos personalizados, promoviendo así una experiencia más accesible, eficiente y enriquecedora.

3.1. Objetivo general

Desarrollar y evaluar una plataforma web que integre OWASP ZAP con un modelo de lenguaje avanzado (LLM), permitiendo la interacción conversacional, la consulta, el análisis automatizado de resultados y la generación asistida de reportes de seguridad web, con el fin de facilitar la gestión de vulnerabilidades y explorar el impacto de la inteligencia artificial en la experiencia de usuario y en la eficacia de las auditorías de seguridad.

3.2. Objetivos especificos

Para alcanzar el objetivo general, se han definido los siguientes objetivos específicos, que guían el desarrollo y la orientación exploratoria del proyecto:

- **Diseñar una interfaz web intuitiva**

  Crear una interfaz accesible y amigable que permita a usuarios de distintos perfiles ejecutar y programar escaneos de seguridad utilizando OWASP ZAP, eliminando barreras técnicas y promoviendo la adopción de buenas prácticas en ciberseguridad.

- **Integrar un modelo de lenguaje de gran tamaño (LLM)**.

  Incorporar un LLM que actuando como asistente conversacional sea capaz de contestar en lenguaje natural las preguntas realizadas por el usuario, generar resúmenes de forma automática de los resultados de los escaneos, aclarar vulnerabilidades y ofrecer recomendaciones individualizadas de mitigación.

- **Facilitar la gestión y consulta de resultados**.

  Permitir al usuario interactuar con la base de datos de resultados de manera conversacional, obteniendo la información más relevante de forma rápida y comprensible, así como realizar búsquedas y filtrados, para posteriormente poder obtener recomendaciones y explicaciones de las mismas ayudandote del LLM.

- **Automatizar la programación y seguimiento de auditorías**.

  Implementar un sistema de planificación de escáneres que permita programar auditorías periódicas y llevar un control histórico de las mismas, de esta manera se consigue un enfoque proactivo y sistemático en la gestión de la seguridad web.

- **Automatizar la generación de reportes personalizados**.

  Desarrollar un mecanismo que, mediante el apoyo del LLM, sea capaz de generar automáticamente reportes técnicos y personalizados en base a las alertas detectadas por OWASP ZAP. Esta funcionalidad deberá ser capaz de completar los apartados de descripción, riesgo y solución para cada alerta, adaptando el contenido al contexto y permitiendo la exportación de informes en formatos más profesionales como DOCX.

- **Evaluar la experiencia de usuario y el impacto del LLM**.

  Analizar, mediante encuestas a usuarios, la funcionalidad y eficacia de la plataforma, así como el valor que aporta la integración del LLM en los procesos de auditoría, gestión de vulnerabilidades y generación de reportes.

- **Explorar nuevas formas de interacción y asistencia**.

  Investigar el potencial que tienen los LLM para transformar la forma en que los usuarios interactúan con los resultados de las auditorias de seguridad web, evaluando cómo la inteligencia artificial puede facilitar la interpretación de información técnica y la toma de decisiones informadas.

- **Sentar las bases para futuros desarrollos**.

  Documentar tanto los hallazgos como las limitaciones identificadas durante el desarrollo del proyecto, con el objetivo de aportar conocimiento y abrir nuevas líneas de investigación en la aplicación de inteligencia artificial en la ciberseguridad.

  Es por ello que los objetivos del TFG no solo se centran en la creación de una herramienta funcional, sino que también profundizan en el posible impacto que la inteligencia artificial puede tener en la comprensión y gestión de la seguridad web, especialmente en la automatización y la generación de reportes técnicos.

4

Tecnologías, Herramientas y Metodologías

4.1. Tecnologías

En este apartado se describen las principales tecnologías, herramientas y servicios que han sido fundamentales para el desarrollo del proyecto. Se explicará el papel que desempeña cada una de ellas dentro de la arquitectura de la aplicación, así como los motivos de su elección y las ventajas que han aportado al proceso de diseño e implementación. Esta visión global permitirá comprender cómo se ha construido la solución y por qué se han seleccionado estas tecnologías frente a otras alternativas disponibles en el mercado.

4.1.1. OWASP ZAP

OWASP ZAP (Zed Attack Proxy) es una herramienta de análisis de seguridad dinámica para aplicaciones web, desarrollada por la Fundación OWASP. Por su capacidad para detectar vulnerabilidades de forma automatizada y manual [5], esta herramienta de código abierto es muy utilizada tanto en entornos profesionales como académicos. ZAP es una de las herramientas más reconocidas dentro de la categoría DAST (Dynamic Application Security Testing) y se halla, gracias al soporte activo de la comunidad OWASP, en constante evolución [6]. Entre sus funcionalidades principales destacan:

- Escaneo automático de amenazas comunes: permite identificar vulnerabilidades como XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery) o inyecciones SQL [7].
- Interceptación y modificación de peticiones HTTP: funciona como un proxy que permite inspeccionar y modificar el tráfico entre el cliente y el servidor.
- Integración con entornos CI/CD: facilita la automatización de pruebas de seguridad en las primeras fases del ciclo de desarrollo.
- Generación de informes detallados: proporciona reportes comprensibles que destacan las vulnerabilidades detectadas, su nivel de riesgo y posibles soluciones.

OWASP ZAP fue utilizada en el presente proyecto como herramienta principal para llevar a cabo las auditorías de seguridad sobre la aplicación desarrollada en Flask. Su integración mediante la API y la librería oficial ZAPv2 permitió automatizar escaneos desde el backend en Python [8], mejorando significativamente la eficiencia en la detección de vulnerabilidades. Un valor añadido de esta herramienta es su capacidad para generar reportes detallados, lo que facilitó la comprensión de los hallazgos y contribuyó a la creación de una aplicación más robusta y alineada con buenas prácticas de seguridad.

4.1.2. Python

Python es un lenguaje de alto nivel de programación interpretado cuya ontología hace hincapié en la legibilidad de su código [9]. Es un lenguaje de programación multiparadigma, pues soporta la orientación a objetos, parcialmente; una programación imperativa y, en menor medida, una programación funcional. Es, consecuentemente, un lenguaje interpretado, dinámico y multiplataforma.

Uno de los motivos esenciales por los que se usó Python fue por la existencia de la librería ZAPv2, que permite una conexión directa con la API de OWASP ZAP. Dicha librería facilitó considerablemente, el lanzamiento automatizado de escáneres, así como la recopilación de resultados y la generación de informes desde la propia aplicación. También fue motivo importante el previo conocimiento tanto en este lenguaje como en el framework utilizado Flask.

Fue la facilidad para la integración de las APIs de los modelos de lenguaje, requisito imprescindible a la hora de optar por Python. Esta característica permitió interpretar y contextualizar los resultados de los escaneos, a la par que añadía un valor significativo para la generación de recomendaciones automatizadas así como para la mejora de la experiencia del usuario.

La elección de Python se sustentó y fundamentó en características como su simplicidad, productividad y versatilidad (comparado con otros lenguajes), sin

olvidar el hecho de que cuenta con una extensa documentación lo que facilitó un desarrollo más fácil y adaptable a las necesidades cambiantes del proyecto.

4.1.3. Flask

Flask [10] es un framework web ligero y flexible para Python que elegí para este proyecto porque se adapta perfectamente a las necesidades de desarrollo rápido y personalización. Una de las principales razones por las que decidí utilizar Flask es su núcleo minimalista, que me permitió diseñar la estructura de la aplicación de una forma que tuviera sentido para el proyecto, sin verme forzado a seguir un patrón rígido.

Desde el principio, se pudo apreciar lo fácil que era establecer rutas y organizar la aplicación en diferentes módulos. Flask también se integra muy bien con herramientas como SQLAlchemy y Flask-Migrate, óptimas las dos para la gestión y los cambios en la base de datos; ambas fueron esenciales para mantener los datos del proyecto coherentes y fáciles de actualizar a medida que la aplicación evolucionaba.

Otra gran ventaja fue la fuerte comunidad de Flask. Hay una enorme cantidad de documentación, tutoriales y extensiones disponibles, lo que hizo mucho más fácil encontrar soluciones a cualquier problema que me encontrara y añadir funciones como la autenticación, los trabajos en segundo plano o las notificaciones por correo electrónico.

Por tanto, fueron la sencillez de Flask, su flexibilidad y el apoyo de su comunidad lo que la convirtieron en la opción ideal para construir una aplicación web robusta y fácil de mantener que, además, pudiera crecer y adaptarse a medida que se desarrollara el proyecto.

4.1.4. PostgreSQL

PostgreSQL [11]es un sistema de gestión de bases de datos relacionales de código abierto, potente, robusto y escalable. Es ampliamente utilizado tanto en aplicaciones empresariales como en proyectos de código abierto.

Una de las principales razones por las que se ha elegido PostgreSQL en este proyecto fue la capacidad para gestionar de forma eficiente grandes volúmenes de datos y consultas complejas. PostgreSQL ofrece soporte avanzado para transacciones ACID y control de concurrencia multiversión (MVCC) lo que garantiza la integridad y la disponibilidad de los datos incluso bajo cargas elevadas.

Además, PostgreSQL se integra perfectamente con Python, lenguaje el cual hemos elegido para el desarrollo de la plataforma. En particular, a través de

bibliotecas como SQLAlchemy y Flask-Migrate, facilitan la gestión y la migración de esquemas a lo largo del desarrollo. Su naturaleza de código abierto, junto con una comunidad activa, garantiza mejoras continuas y parches de seguridad.

En este proyecto, se ha seleccionado PostgreSQL como solución de almacenamiento principal, con el objetivo de proporcionar un entorno más escalable, robusto y concurrente. De este modo, ofrece una base sólida para aplicaciones de gran volumen de datos, lo que es fundamental para la escalabilidad y la alta disponibilidad.

4.1.5. Modelos de Lenguaje Grande (LLM)

Los modelos de lenguaje grande (LLM, por sus siglas en inglés, Large Language Model) son sistemas avanzados de inteligencia artificial diseñados para procesar, interpretar y generar texto en lenguaje natural [12]. Estos modelos, entre los que se encuentra ChatGPT de OpenAI [13], se entrenan utilizando grandes volúmenes de datos textuales y llegan, incluso, a realizar tareas complejas como responder preguntas, resumir información, extraer datos relevantes y generar contenido contextualizado.

En este trabajo de investigación, los LLM desempeñan un papel fundamental al proporcionar capacidades avanzadas de interpretación y contextualización de los resultados generados por OWASP ZAP. Su integración permitió:

- **Interpretación de resultados:** Los LLM analizaron las vulnerabilidades detectadas por OWASP ZAP, ofreciendo explicaciones comprensibles sobre su naturaleza, impacto y riesgos asociados.
- **Generación de recomendaciones:** A partir del análisis de los resultados, los LLM generaron sugerencias automatizadas para la mitigación de vulnerabilidades, facilitando la toma de decisiones informadas.
- **Interacción mediante chatbot**: Se integró un chatbot que utiliza un LLM para responder a consultas realizadas por los usuarios en lenguaje natural. Esto permitió una interacción más intuitiva y accesible con el sistema.

La elección de integrar un LLM como ChatGPT o Gemini [14] se basó en su facilidad de uso, documentación clara y capacidad para adaptarse a distintos contextos. Además, su API permitió una integración fluida con el backend desarrollado en Python, lo que facilitó la implementación de funcionalidades avanzadas como la extracción de parámetros desde texto libre o la generación de resúmenes detallados sobre reportes.

El uso de LLM representa uno de los aspectos más innovadores del proyecto, ya que combina inteligencia artificial con análisis de seguridad para ofrecer una

herramienta más completa y eficiente. Esta integración no solo optimizó el análisis técnico, sino que también mejoró significativamente la experiencia del usuario al hacer que los resultados fueran más comprensibles y accionables.

4.1.6. JQuery

jQuery [15]es una biblioteca de JavaScript rápida, ligera y ampliamente utilizada para el desarrollo web del lado del cliente. Desde su creación en 2006 por John Resig [16], se ha consolidado como una herramienta fundamental en el desarrollo frontend gracias a su capacidad para simplificar la manipulación del DOM, el manejo de eventos, la realización de animaciones y la ejecución de operaciones AJAX, todo ello mediante una sintaxis concisa y compatible con múltiples navegadores.

Entre sus características técnicas más destacadas se incluyen la selección y modificación eficiente de elementos HTML, un sistema unificado para gestionar eventos del usuario, métodos incorporados para efectos visuales y funciones simplificadas para realizar peticiones asíncronas a servidores y APIs externas.

En el presente proyecto, jQuery se utilizó principalmente para mejorar la interactividad del frontend. Se implementaron validaciones dinámicas de formularios, actualizaciones de contenido sin recarga de página mediante AJAX y pequeños efectos visuales que optimizan la experiencia del usuario. Gracias a su integración sencilla y directa, jQuery facilitó la creación de una interfaz más fluida, accesible y responsive, evitando la necesidad de incorporar frameworks más complejos.

4.1.7. SQLAlchemy

SQLAlchemy es una biblioteca de Python que actúa como un potente sistema de mapeo objeto-relacional (ORM) y un conjunto completo de herramientas SQL [17]. Esta biblioteca permite interactuar con diferentes tipos de bases de datos de forma abstracta, facilitando tanto la generación de consultas SQL dinámicas mediante su SQL Expression Language como la manipulación de datos a través de su ORM, el cual traduce las tablas de bases de datos en clases Python y las filas en instancias de objetos.

En este proyecto, SQLAlchemy se utilizó para modelar y gestionar la base de datos PostgreSQL de manera eficiente. Gracias a su integración con herramientas de migración como Flask-Migrate, se logró mantener y actualizar el esquema de la base de datos de forma estructurada, permitiendo que los cambios en el modelo se reflejen de manera automatizada en la base de datos sin perder la consistencia o escalabilidad del sistema.

La elección de SQLAlchemy se fundamentó en sus ventajas en términos de

abstracción, portabilidad y robustez, lo que permitió a los desarrolladores centrarse en la lógica de negocio sin preocuparse por los detalles del manejo directo de SQL. Esta flexibilidad y capacidad de adaptación lo convierten en una herramienta esencial para proyectos que requieren un manejo avanzado de datos en Python.

4.1.8. Langchain

LangChain es un framework de código abierto diseñado para facilitar la construcción de aplicaciones basadas en grandes modelos de lenguaje (LLM)[18]. Proporciona un conjunto de herramientas y abstracciones que permiten encadenar llamadas a LLM, incorporar lógica personalizada, gestionar la memoria y conectar con herramientas externas, lo que resulta esencial para la automatización y la ejecución de flujos complejos en aplicaciones inteligentes.

En este proyecto, LangChain fue fundamental para integrar capacidades avanzadas de procesamiento de lenguaje, como la interpretación de resultados y la generación automatizada de recomendaciones de seguridad. Su uso permitió manipular y extraer información relevante de los datos generados por las herramientas de análisis, mejorando así la toma de decisiones en tiempo real.

Además, se incorporó Langgraph, una extensión complementaria que se especializa en el manejo de flujos conversacionales [19]. Langgraph se utilizó específicamente en la parte del chatbot del sistema, posibilitando que el agente conversacional determinase de forma dinámica qué función o "tool" ejecutar en base a las consultas en lenguaje natural del usuario. Esta funcionalidad enriqueció la interacción, permitiendo no solo responder a preguntas de manera precisa, sino también coordinar acciones dentro del sistema para optimizar el proceso automatizado [20].

La combinación de LangChain y Langgraph aportó un componente innovador al proyecto, al unir la capacidad de procesamiento inteligente de textos con una gestión avanzada de la conversación. Esto facilitó la integración de la inteligencia artificial en el flujo de trabajo del chatbot, haciendo que la herramienta sea más intuitiva, proactiva y adaptativa a las necesidades del usuario.

4.1.9. APScheduler

APScheduler (Advanced Python Scheduler) es una librería para Python que ofrece una solución flexible y robusta para la programación de tareas [21]. Permite ejecutar trabajos a intervalos fijos, en horarios determinados o siguiendo patrones de cron, de forma similar a las tareas programadas de sistemas operativos, pero directamente integradas en un entorno Python [22]. Esta característica

resulta especialmente útil para automatizar procesos críticos, ya que posibilita la ejecución asíncrona y programada de funciones sin intervención manual.

Entre sus principales beneficios destacan:

- **Configuración flexible de disparadores:** APScheduler admite disparadores basados en intervalos, fechas específicas o patrones cron, lo que facilita el diseño de tareas que se ejecuten de acuerdo a los requerimientos del proyecto.
- **Ejecución paralela:** La librería permite la ejecución de trabajos en hilos separados o mediante otros mecanismos de concurrencia, lo que es ideal para no bloquear la aplicación principal.
- **Integración sencilla con aplicaciones web:** APScheduler se integra de forma fluida con frameworks como Flask, permitiendo programar tareas en el contexto de aplicaciones web sin añadir una sobrecarga significativa en la configuración.
- **Adaptabilidad a flujos de trabajo dinámicos:** Su flexibilidad permite reprogramar tareas en tiempo real y adaptar el comportamiento de la aplicación a las necesidades cambiantes, lo que es crucial en entornos de integración continua y despliegue automático.

En este proyecto, APScheduler se ha utilizado para programar escaneos de seguridad automatizados con OWASP ZAP, asegurando que dichos escaneos se ejecuten en intervalos específicos sin requerir intervención manual. Esto ha contribuido a incrementar la eficiencia y robustez del sistema, integrando de manera efectiva la automatización en el flujo de trabajo del desarrollo.

4.1.10. Python-docx

Python-docx es una biblioteca de Python que permite crear y modificar documentos en formato DOCX, el estándar de Microsoft Word [23]. Esta herramienta facilita la generación automatizada de reportes a partir de plantillas predefinidas, lo que permite estructurar el contenido en secciones, encabezados, párrafos, tablas y otros elementos de formato de forma sencilla. En este proyecto, python-docx se ha utilizado en combinación con el modelo de lenguaje (LLM) para rellenar dinámicamente la información en una plantilla de reporte. Esta integración posibilita que los resultados obtenidos en los procesos de análisis y verificación de seguridad se conviertan en un documento bien estructurado y profesional, listo para su revisión o distribución [24].

El uso de Python-docx, junto con la capacidad del LLM para interpretar y generar texto, ha permitido automatizar la creación de reportes complejos de

manera eficiente. Esta solución es especialmente valiosa en entornos donde la generación de documentación debe adaptarse a cambios constantes y donde la claridad y profesionalidad del reporte final es crucial. La combinación de una plantilla fija y la inserción dinámica de datos garantiza que cada reporte comunique de forma precisa y estandarizada los hallazgos obtenidos durante el análisis de seguridad.

4.2. Herramientas

4.2.1. Docker

Docker es una plataforma de virtualización ligera que permite empaquetar aplicaciones junto con todas sus dependencias en contenedores portátiles y aislados [25]. Desde su lanzamiento en 2013 [26], Docker se ha consolidado como una herramienta clave en la entrega continua y la automatización de despliegues debido a su capacidad para garantizar que las aplicaciones se ejecuten de manera consistente en diferentes entornos, ya sean de desarrollo, prueba o producción.

Entre sus características más relevantes destacan:

- **Portabilidad entre sistemas**: Los contenedores Docker son independientes del sistema operativo subyacente, lo que asegura que las aplicaciones funcionen sin problemas en cualquier entorno compatible.
- **Aislamiento de procesos**: Cada contenedor opera de forma aislada, evitando conflictos entre aplicaciones y garantizando un entorno seguro.
- **Eficiencia en la gestión de recursos**: Docker optimiza el uso de recursos del sistema, permitiendo ejecutar múltiples contenedores simultáneamente con un impacto mínimo.
- **Facilidad de configuración mediante Dockerfiles**: Permite definir entornos complejos de forma reproducible y controlada, facilitando la gestión y despliegue automatizado.

En este proyecto, Docker se utilizó para mantener un entorno homogéneo entre desarrollo y producción, evitando conflictos relacionados con versiones o dependencias. Gracias a su uso, se garantizó una mayor estabilidad durante las pruebas y se facilitó el despliegue automatizado y reproducible de la aplicación en distintos sistemas. Además, Docker permitió simplificar la integración de OWASP ZAP y otros componentes del sistema, asegurando que todos los módulos funcionaran correctamente en un entorno controlado.

4.2.2. SendGrid

SendGrid [27] es una plataforma de envío de correo electrónico basada en la nube. Una de las principales razones por las que se eligió SendGrid para este proyecto es su capacidad para enviar correos electrónicos de forma fácil y eficiente, sin necesidad de configurar y mantener nuestro propio servidor de correo.

Gracias a su API y a la biblioteca «requests» de Python, la integración de SendGrid en el proyecto fue muy sencilla. Esto nos permitió automatizar el envío de informes a los usuarios una vez completado cada escaneo, mejorando la experiencia del usuario y la entrega de resultados y documentos. Además, SendGrid ofrece garantías de entrega, seguimiento de mensajes y gestión de errores.

La elección de SendGrid simplificó considerablemente la gestión y envíos de correo electrónico en la plataforma, lo que permitió centrarnos en el desarrollo de otras funciones más importantes sin preocuparnos por el envío de correos electrónicos.

4.2.3. Visual Studio Code

Visual Studio Code (VSCode) es un editor de código fuente desarrollado por Microsoft que se ha consolidado como una herramienta esencial en el desarrollo de software gracias a su ligereza, potencia y versatilidad [28]. Ofrece una experiencia de edición altamente personalizable, con soporte integrado para control de versiones mediante Git, funciones avanzadas de autocompletado (IntelliSense), depuración en tiempo real y la capacidad de trabajar con múltiples lenguajes y frameworks. Su extenso ecosistema de extensiones permite a los desarrolladores adaptar el entorno a las necesidades específicas del proyecto; por ejemplo, en este trabajo se emplearon extensiones como GitLens para gestionar de manera avanzada los repositorios y Bracket Pair Colorizer para mejorar la legibilidad del código [29].

En el presente proyecto, Visual Studio Code se utilizó como el entorno de desarrollo principal para la codificación en Python y la integración de múltiples herramientas. Su interfaz intuitiva y la facilidad para integrar otras aplicaciones (como terminales embebidos y herramientas de formateo de código) facilitaron la gestión del proyecto, contribuyendo a un desarrollo más ágil y colaborativo. Gracias a su soporte multiplataforma y a la activa comunidad de desarrolladores, VSCode permitió optimizar la productividad y mantener una coherencia en el desarrollo a lo largo de todo el ciclo de vida del proyecto.

4.2.4. Git

Git es un sistema de control de versiones distribuido de código abierto, diseñado para gestionar de manera eficiente los cambios en el código fuente a lo largo del ciclo de desarrollo del software [30]. Creado por Linus Torvalds en 2005, Git se ha convertido en la herramienta de referencia para equipos de desarrollo debido a su capacidad para manejar proyectos de cualquier escala, facilitando la colaboración y el seguimiento de cada modificación realizada en el repositorio [31].

Entre sus características más destacadas se encuentran la gestión de ramas y fusiones (merge), lo que permite un flujo de trabajo colaborativo basado en modelos como Gitflow, la posibilidad de trabajar de forma descentralizada sin depender de un servidor central, así como un sistema robusto de seguimiento del historial de cambios, que posibilita la identificación rápida de errores y la reversión a versiones anteriores en caso de ser necesario.

En este proyecto, Git se utilizó como núcleo para el control de versiones, integrándose de manera fluida con plataformas de alojamientos de código como GitHub y siendo fundamental para implementar un flujo de trabajo basado en Gitflow [29]. Esto permitió gestionar eficientemente las modificaciones en el código durante las distintas fases del desarrollo, garantizando que los cambios fueran revisados, documentados y fusionados de forma organizada, contribuyendo a la estabilidad y calidad del software.

4.2.5. GitHub

GitHub es una plataforma basada en la web que permite alojar y gestionar repositorios Git, promoviendo la colaboración efectiva entre desarrolladores [29]. Desde su lanzamiento en 2008, se ha consolidado como la principal herramienta para proyectos de código abierto y privado, integrando funcionalidades clave como revisión de código, gestión de incidencias y automatización de flujos de trabajo.

Entre sus características más destacadas se encuentran el alojamiento remoto de repositorios, el uso de Pull Requests para revisión colaborativa, el seguimiento de tareas mediante Issues y la automatización de procesos mediante GitHub Actions, lo que permite implementar prácticas de integración y entrega continua (CI/CD). Durante el desarrollo del proyecto, GitHub se utilizó como plataforma central de coordinación, facilitando el control del ciclo de vida del código y la colaboración entre los miembros del equipo, lo cual contribuyó directamente a la mejora del flujo de trabajo y de la calidad del producto final.

4.2.6. SonarQube

SonarQube es una plataforma de inspección continua de la calidad del código que realiza análisis estático para detectar errores, problemas de mantenibilidad, olor a código (code smells) y vulnerabilidades de seguridad en múltiples lenguajes de programación [32]. Esta herramienta se ha convertido en un componente esencial en los procesos de integración continua, proporcionando retroalimentación inmediata sobre el estado del código y facilitando la adopción de buenas prácticas de desarrollo seguro [33].

En este proyecto, SonarQube se integró en el pipeline de CI/CD para monitorear y mejorar la calidad del código fuente de la aplicación. Cada vez que se realizaba un commit y se ejecutaba una compilación, SonarQube analizaba automáticamente el código, generando informes detallados que permitían identificar áreas de mejora y potenciales riesgos de seguridad. Esta integración no solo permitió detectar incidentes de calidad de manera temprana, sino que también ayudó a garantizar que el código cumpliera con los estándares de seguridad y mantenibilidad establecidos, lo que se tradujo en un producto final más robusto y confiable.

4.2.7. Codeium

Codeium AI es una herramienta de inteligencia artificial diseñada para potenciar la productividad de los desarrolladores mediante la integración de capacidades avanzadas de autocompletado, generación de código y asistencia en tiempo real dentro del entorno de desarrollo integrado (IDE). Su enfoque principal es mantener a los desarrolladores en un flujo continuo de trabajo, reduciendo interrupciones y facilitando la creación de soluciones complejas con mayor eficiencia [34].

En este proyecto, Codeium AI se utilizó para optimizar el proceso de desarrollo del sistema, especialmente en tareas repetitivas o complejas relacionadas con la escritura de código. La herramienta fue una aliada clave para incrementar la velocidad y precisión al implementar funciones críticas, como la integración con APIs externas, la gestión de bases de datos y la configuración avanzada del backend. Además, su capacidad para entender el contexto del código permitió sugerencias precisas que aceleraron el desarrollo iterativo e incremental.

Una característica destacada fue su integración en el IDE utilizado durante el proyecto, lo que permitió que Codeium AI proporcionara asistencia directamente en el entorno de trabajo sin necesidad de cambiar entre herramientas.

4.3. Metodologías

El desarrollo de este proyecto se ha sustentado en un enfoque incremental e iterativo, permitiendo incorporar y validar de forma progresiva cada uno de los componentes de la plataforma y adaptarse a los retos y oportunidades surgidos durante el proceso. Un aspecto fundamental a lo largo de todas las fases ha sido el constante seguimiento y acompañamiento por parte del tutor del Trabajo de Fin de Grado. Las reuniones periódicas con el tutor han sido esenciales para debatir problemas técnicos y conceptuales, poner en común posibles mejoras y soluciones, y orientar el proyecto hacia la consecución de sus objetivos. Este apoyo ha resultado clave tanto en la toma de decisiones como en el enriquecimiento de la experiencia investigadora.

La implementación del proyecto se ha estructurado en cinco etapas principales, que se describen a continuación:

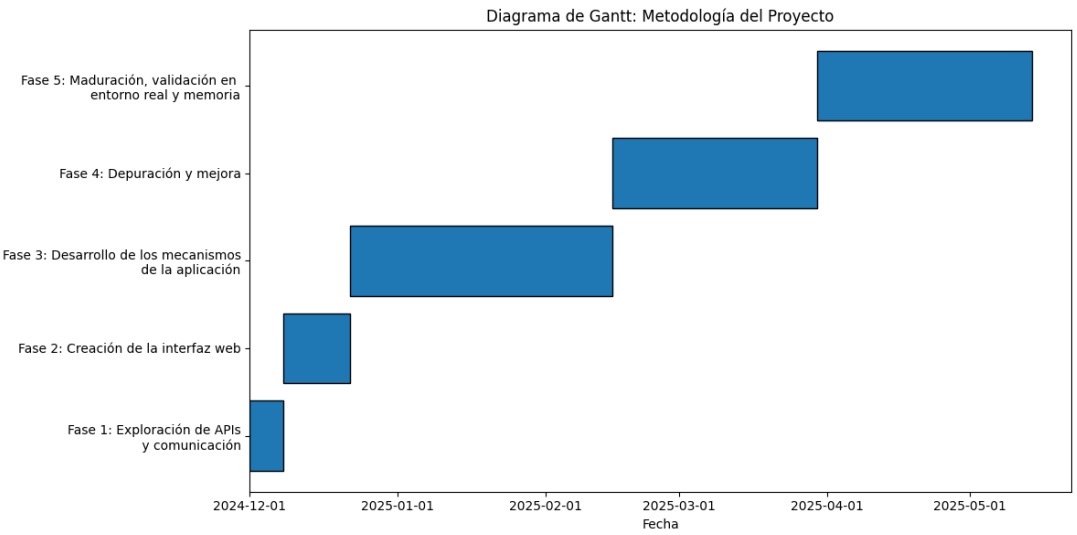


Figura 4.1: Diagrama Gantt

- **Fase 1: Exploración de APIs y comunicación**

  En esta etapa inicial se realizó un estudio profundo para comprender el funcionamiento de las APIs involucradas en el proyecto. Se investigó el acceso y manipulación de la API de OWASP ZAP, descubriendo la utilidad de la librería ZAPv2 de Python, lo que facilitó la integración de las funcionalidades de escaneo en la plataforma. Además, se efectuaron pruebas exploratorias con las API de distintos LLM, realizando consultas simples y aleatorias para evaluar su capacidad de respuesta, comprender los métodos de autenticación y el formato de sus respuestas. Se llevaron a cabo integraciones básicas del LLM para ejecutar funciones elementales, verificando la viabilidad de su futuro despliegue en el proyecto.

- **Fase 2: Creación de la interfaz web**

  Se inició el desarrollo de una interfaz web básica y funcional que sirviera como punto de interacción para el usuario. Esta interfaz no solo permitió una navegación sencilla, sino que también facilitó la ejecución de escaneos de seguridad de manera muy intuitiva, haciendo posible activar los procesos de análisis de forma directa y sin complicaciones. Posteriormente, se implementaron técnicas de AJAX para mejorar la interactividad y dinamismo de la interfaz, consolidando así una experiencia de usuario más fluida y accesible.

- **Fase 3: Desarrollo de los mecanismos de la aplicación**

  En esta etapa se desarrollaron los componentes centrales de la aplicación. Se integró un chatbot conversacional que permitía interactuar con la base de datos de resultados, generando resúmenes y proporcionando explicaciones sobre las vulnerabilidades detectadas. Adicionalmente, se implementó un sistema de programación basado en AppScheduler, se estableció la conexión con la base de datos y se configuró el envío automático de correos electrónicos. Durante este período se recopiló feedback de usuarios, lo que ayudó a identificar áreas de mejora en la interfaz y en el comportamiento del chatbot.

- **Fase 4: Depuración y mejora**

  Posteriormente, se destinó un esfuerzo a la optimización y depuración de los módulos desarrollados. Se refinaron y mejoraron los componentes existentes, destacándose la actualización del chatbot –ahora utilizando Langraph– para optimizar su capacidad de respuesta y precisión. Además, se implementó un mecanismo automatizado que, apoyado en el LLM, genera reportes personalizados en formato DOCX, completando automáticamente secciones referentes a la descripción, evaluación de riesgo y recomendaciones de mitigación para cada alerta de seguridad.

- **Fase 5: Maduración y validación en entorno real** La etapa final se concentró en consolidar la herramienta, corrigiendo errores remanentes y perfeccionando tanto la estética como la funcionalidad general de la plataforma. Se crearon contenedores Docker, lo que facilitó la realización de pruebas en entornos reales y garantizó la portabilidad del sistema. Asimismo, se desarrolló un endpoint específico que permite la generación de reportes a partir de la subida de archivos JSON, ampliando así la versatilidad de la aplicación para distintos escenarios de uso.

Este enfoque incremental e iterativo, sumado al valioso seguimiento y apoyo del tutor, ha sido determinante en la evolución progresiva del proyecto. Esta metodología ha permitido superar obstáculos, implementar mejoras continuas y asegurar la calidad, flexibilidad y escalabilidad de la plataforma desarrollada.

5

Descripción Informática

A continuación se presenta una visión general del proyecto que sirve de punto de partida para los apartados posteriores (requisitos, arquitectura, diagramas, diseño e implementación y fases de desarrollo). En este Trabajo de Fin de Grado se ha desarrollado una plataforma web de auditoría de seguridad que integra OWASP ZAP con un Modelo de Lenguaje Grande, ofreciendo al usuario una experiencia conversacional a través de un chatbot. La plataforma permite ejecutar y programar escaneos de vulnerabilidades, consultar y almacenar sus resultados en una base de datos, generar reportes detallados en formato DOCX y recibir recomendaciones de mitigación personalizadas. Se puede ver un diagrama general de la aplicación en la Figura 5.1.

En la sección de requisitos se definen las necesidades funcionales y no funcionales que guían el desarrollo. A continuación, en la arquitectura y descripción general se muestra cómo encajan los componentes principales -interfaz web, motor de escaneo, chatbot, motor de persistencia y generador de reportes- formando un ecosistema coherente y escalable. Los diagramas de clases de análisis y diseño, junto con el diagrama de casos de uso, ofrecen una representación gráfica de las entidades y las interacciones clave, facilitando la comprensión del sistema.

El apartado de diseño e implementación desglosa las decisiones técnicas, herramientas empleadas (Flask, SQLAlchemy, LangGraph, Docker, etc.) y la evolución del código a través de cinco fases iterativas, en las que se exploraron APIs, se construyó la interfaz, se implementaron los módulos centrales, se optimizó el chatbot y se validó el sistema en entornos reales. Esta estructura permite seguir paso a paso el proceso creativo y técnico que ha convertido una idea inicial en

una solución práctica, robusta y lista para su despliegue.

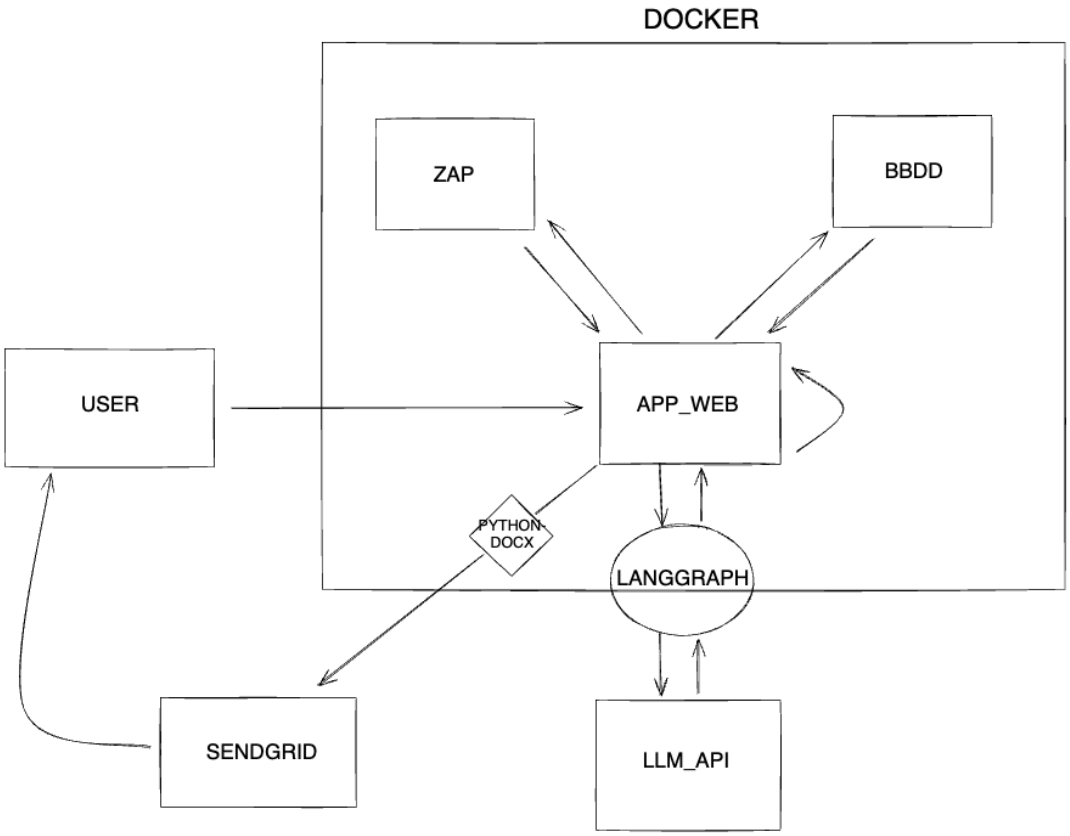


Figura 5.1: Diagrama General

5.1. Requisitos

En esta sección se profundiza en los diversos requisitos planteados para la aplicación, abarcando tanto aquellos definidos desde el inicio del proyecto como los que surgieron o se ajustaron durante el proceso de desarrollo. Estos requisitos se establecieron previamente, sirviendo de guía conceptual en la construcción del sistema, y fueron evolucionando en función de los desafíos y descubrimientos surgidos en cada iteración.

El objetivo es ofrecer una visión completa del conjunto de necesidades que impulsó la elaboración del proyecto, mostrando cómo estas se reconfiguraron ante los retos encontrados y las pruebas iterativas que llevaron a descartar o modificar ciertas funcionalidades. Para lograr una comprensión clara y cercana de las motivaciones, se ha optado por presentar los requisitos en un formato narrativo, lo que facilita la comunicación tanto de las metas como del proceso de toma de decisiones.

Más allá de los requisitos funcionales, se abordarán aspectos cruciales para asegurar la eficacia y la calidad del producto, tales como la estabilidad, usabilidad, seguridad y rendimiento. A medida que avanzaba el desarrollo, diversos experimentos y pruebas revelaron limitaciones en algunos enfoques, lo que propició la búsqueda de alternativas tecnológicas y la reformulación de ciertos objetivos. Estas experiencias de ensayo y error fueron determinantes para que el proyecto evolucionara de manera orgánica, ajustándose mejor a las necesidades reales de los usuarios y cimentando una base sólida para garantizar la coherencia y calidad de la solución final.

Asimismo, se documentarán situaciones en las que dificultades técnicas o cambios en la visión inicial motivaron ajustes en los objetivos propuestos. Esta documentación no solo justifica las decisiones adoptadas, sino que también ejemplifica cómo la adaptabilidad y la iteración constante fueron claves para alcanzar el nivel de madurez deseado en el sistema. En definitiva, se pretende ofrecer una visión integral y transparente del proceso de crecimiento y consolidación que dio forma al proyecto.

El establecimiento de los requisitos se realizó con el objetivo de definir de manera clara el conjunto de funcionalidades que debía incorporar la aplicación, así como establecer los parámetros de calidad que asegurarían su efectividad en la gestión de vulnerabilidades y la interacción con el usuario. Desde el inicio se plantearon las necesidades esenciales que servirían de guía conceptual durante el desarrollo, siendo estos requisitos sometidos a revisión y adaptación en función de los desafíos encontrados y de las pruebas iterativas realizadas.

5.1.1. Requisitos funcionales

Se identificaron y definieron los siguientes requisitos funcionales, que describen las principales operaciones que la aplicación debía implementar:

- La plataforma permitirá ejecutar escaneos de seguridad de manera directa mediante una interfaz web intuitiva, facilitando, sin requerir conocimientos técnicos avanzados, la activación de procesos de análisis.
- El sistema permitirá programar escaneos de seguridad en momentos específicos mediante un sistema de agendamiento (AppScheduler), garantizando una auditoría regular y proactiva.
- El sistema permitirá a los usuarios interactuar en lenguaje natural con la plataforma a través de un chatbot conversacional basado en un modelo de lenguaje avanzado (LLM), consultando la base de datos de resultados, obteniendo resúmenes automáticos de los escaneos y recibiendo explicaciones detalladas sobre las vulnerabilidades identificadas.

- El sistema permitirá la generación automática de reportes individualizados en formato DOCX, donde el LLM completará de forma asistida secciones relacionadas con la descripción de cada alerta, su estimación y valoración en términos de riesgo y, finalmente, recomendaciones de mitigación.
- El sistema permitirá la carga de archivos JSON a través de un endpoint habilitado, posibilitando la generación de reportes idénticos a los obtenidos tras un escaneo en tiempo real y ampliando las posibilidades de integración con otros flujos de trabajo.
- Cuando un reporte esté disponible, el sistema enviará de forma automática un correo electrónico con el resumen de la auditoría y, al mismo tiempo, permitirá a los usuarios acceder al documento completo desde la interfaz web.

5.1.2. Requisitos no funcionales

Amén de los requisitos funcionales, se establecieron criterios no funcionales que afianzan la calidad y robustez del producto final. Entre estos se incluyen:

- **Estabilidad y robustez**. La aplicación debe ser capaz de soportar un uso intensivo y siempre garantizando un funcionamiento continuo y exento de errores críticos en entornos de alta demanda
- **Usabilidad**. Una interfaz intuitiva y accesible será la prioridad principal de su diseño. La interacción, incluso para usuarios con conocimientos técnicos limitados, será intuitiva y accesible con la intención final de fomentar una experiencia de usuario positiva y eficiente.
- **Seguridad**. Para proteger la integridad y confidencialidad de la información, se implementarán medidas en el almacenamiento y procesamiento de datos sensibles; de igual modo la implementación se aplicará durante la comunicación entre los distintos módulos y servicios externos (como OWASP ZAP y los LLM)
- **Rendimiento**. Esta plataforma debe proporcionar tiempos de respuesta óptimos y resolver de manera eficiente la gestión, ejecución y programación de escaneos, reduciendo al máximo los retrasos en la presentación de resultados y reportes.
- **Mantenibilidad**. Las futuras ampliaciones y mejoras están garantizadas gracias al diseño del sistema, que facilitará la integración de nuevas funcionalidades o ajustes dependiendo del crecimiento y cambio en las necesidades de los usuarios.

5.2. Arquitectura y análisis

La arquitectura general de la aplicación ha sido orquestada desde un enfoque modular y escalable; se ha empleado una estructura en capas favorecedora de la separación de responsabilidades y facilitadora tanto del mantenimiento como de la evolución futura del sistema. El diseño arquitectónico contempla los siguientes componentes principales:

5.2.1. Descripción general de la arquitectura

En un primer nivel, el sistema se compone de los siguientes módulos principales:

- **Capa de presentación (Frontend)**.

  Constituye el punto de interacción del usuario y se ha desarrollado con HTML, Bootstrap y componentes JavaScript, Oferta un panel para producir escaneos inmediatos o programados, un calendario para el seguimiento de auditorías, gráficas de métricas y un chatbot integrado. Este diseño prioriza usabilidad y accesibilidad, de modo que perfiles sin experiencia previa con OWASP ZAP puedan, de forma intuitiva, operar con esta herramienta

- **Capa de lógica de negocio (Backend)**.

  Orquesta la funcionalidad principal de la plataforma y está implementada en un único servicio Flask desplegado con Gunicorn. Es la gestora de la autenticación y de las sesiones; programa tareas mediante APScheduler; invoca OWASP ZAP a través de la librería zapv2; procesa los resultados, genera reportes DOCX conpython-docx; se comunica con los modelos de lenguaje (LangGraph+Gemini/OpenAI) y envía notificaciones por correo vía SendGrid. Todas las integraciones externas se realizan desde esta capa.

- **Seguridad**

  El diseño incluye: protección contra fuerza bruta con Flask-Limiter, cabeceras de endurecimiento(X-XSS-Protection, X-Frame-Options, Cache-Control, etc.) y gestión de secretos mediante variables de entorno. Asimismo, nunca se almacenan en el código fuente ni las claves de API ni credenciales sensibles, de esta manera se garantiza tanto la integridad como la confidencialidad de la información.

- **Capa de persistencia**:

  Esta capa almacena los escaneos programados / completados, alerta, estadísticas y también reportes al utilizar el sistema de gestión de base de datos PostgreSQL. El modelo de datos se gestiona con SQLAlchemy y

Flask-Migrate, asegurando consistencia transaccional y un esquema fácilmente versionable. Se prioriza tanto la integridad como la confidencialidad, habilitando consultas eficientes que alimentan paneles y chatbot.

- **Capa de despliegue y orquestación**:

  El sistema está preparado para ejecutarse en contenedores Docker. Un docker-compose define tres servicios —aplicación(web), base de datos y OWASP ZAP— compartiendo un volumen para los reportes generados. Esta estrategia aporta portabilidad, facilita la escalabilidad horizontal de la capa web y simplifica la replicación del entorno en distintas fases (desarrollo, pruebas y producción).

Como consecuencia, se genera una arquitectura flexible, robusta y lista para integrar nuevas funcionalidades, garantizando la calidad y la sostenibilidad del producto a largo plazo.

5.2.2. Modelo conceptual del sistema

Esta sección representa de forma abstracta las principales entidades del sistema y sus relaciones fundamentales. No se consideran detalles de implementación, sino que se busca una comprensión funcional del sistema desde la perspectiva del usuario, el flujo de información y las responsabilidades entre elementos clave.

La figura 5.2 recoge este modelo conceptual. A continuación, se describen las clases principales identificadas y sus relaciones:

- **Usuario**: Representa al operador humano que interactúa con el sistema. El usuario puede iniciar análisis, programarlos o cargar archivos JSON con los resultados.

- **Análisis**: Modela la realización de un análisis de seguridad (utilizando OWASP ZAP) en una URL. Cada análisis da como resultado la generación de varias alertas.

- **Alerta**: Captura cada vulnerabilidad detectada en un análisis, incluyendo atributos tales como el nombre de la vulnerabilidad y su nivel de riesgo.

- **Informe**. Agrupa y presenta la información de las alertas. Tras completar un escaneo o tras cargar un archivo JSON, en ambos casos, se genera el informe utilizando una plantilla DOCX. Durante dicha generación, se realizan llamadas secuenciales al LLM que rellena los detalles, el riesgo y la solución para cada alerta y escribe un resumen ejecutivo.

- **LLM**. Es el modelo de lenguaje encargado de generar el texto (descripciones, recomendaciones, evaluaciones) para enriquecer los informes. Todas las llamadas al LLM se efectúan para obtener contenido descriptivo y contextual.
- **Chatbot conversacional**. Es la entidad que se encarga de la interacción en lenguaje natural con el usuario. Implementado en el marco LangGraph (una extensión de LangChain), este gestor conversacional controla y maneja los flujos multiagente, permitiendo, según el contexto, realizar consultas a la base de datos o coordinar acciones (como ejecutar o programar escaneos). Su función es servir de interfaz inteligente entre el usuario y los demás componentes del sistema.

Relaciones destacadas:

- Un usuario puede ejecutar uno o más escaneos.
- Cada escaneo produce múltiples alertas.
- Cada escaneo origina un informe que reune las alertas detectadas; también, el usuario puede generar un informe a partir de un archivo JSON.
- El informe utiliza LLM para enriquecer su contenido: cada alerta se completa de manera secuencial con detalles, riesgo y solución, como también un resumen ejecutivo.
- El usuario interactúa directamente con el chatbot conversacional, que utiliza LLM (a través de LangGraph) para administrar el flujo de la conversación y ejecutar consultas a la base de datos según el contexto de la solicitud.

5.2.3. Diseño técnico del sistema

A partir del modelo conceptual, esta sección detalla el diseño técnico del sistema, representado mediante un diagrama de clases de diseño. Aquí se describen las clases concretas que conforman la implementación, incluyendo servicios, controladores, adaptadores y entidades persistentes. Este modelo técnico permite visualizar la arquitectura del software y sus dependencias internas, preparando el terreno para la codificación.

En este nivel de diseño, se distinguen diversos componentes organizados según sus responsabilidades:

- **Componentes de autenticación y control de usuarios**: AuthService y UserController son quienes gestionan el acceso al sistema y las operaciones de los usuarios.

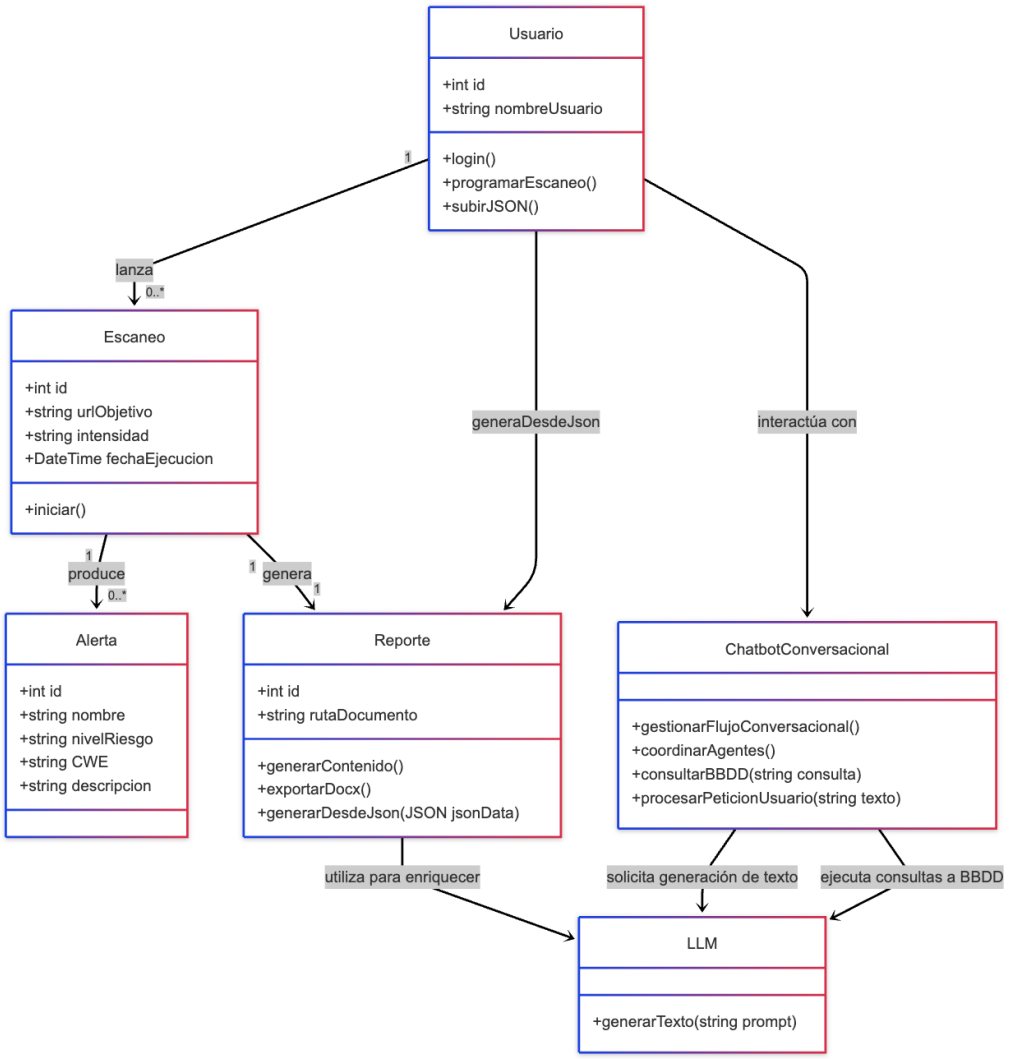


Figura 5.2: Diagrama de clases de análisis

- **Servicios de escaneo y programación**: ScanService y SchedulerService son los encargados de coordinar la ejecución y planificación de los análisis de seguridad.

- **Adaptadores externos**: ZAPClient, LLMClient y EmailService encapsulan, respectivamnete, la comunicación con OWASP ZAP, los modelos de lenguaje y el servicio de correo.

- **Generación de reportes**: ReportGenerator implementa la lógica para crear documentos estructurados a partir de los resultados obtenidos.

- **Entidades de datos**: ScanJob, ScanResult y AlertDetail representan la información persistente almacenada en la base de datos.

Los controladores delegan tareas a los servicios, que a su vez utilizan adaptadores para interactuar con sistemas externos. Los datos generados se almacenan como entidades y se procesan mediante generadores especializados. Esta organización modular favorece la escalabilidad y el mantenimiento del sistema.

5.2.4. Diagrama de casos de uso

El sistema desarrollado permite a los usuarios gestionar auditorías de seguridad web de forma intuitiva, automatizada y asistida por inteligencia artificial. Los usuarios pueden lanzar escaneos inmediatos o programados, ambos generando reportes personalizados en formato DOCX. Estos reportes se elaboran mediante una plantilla y se completan automáticamente gracias a un modelo LLM, que rellena de forma secuencial los detalles, el nivel de riesgo y la solución para cada alerta detectada, además de redactar un resumen ejecutivo. El reporte final es enviado al usuario por correo electrónico (SendGrid).

Adicionalmente, el usuario puede subir un archivo JSON con resultados de un escaneo (realizado previamente o con otra herramienta compatible), y el sistema genera un reporte exactamente igual al que se obtiene tras un escaneo, utilizando la misma plantilla y el mismo proceso de enriquecimiento con el LLM.

El sistema también integra un chatbot conversacional implementado con Langraph, que permite ejecutar o programar escaneos mediante lenguaje natural, solicitar resúmenes de escaneos pasados, consultar vulnerabilidades almacenadas en la base de datos o pedir análisis sobre URLs concretas. Esta interacción conversacional facilita el acceso a la información y la gestión de auditorías incluso a usuarios sin conocimientos técnicos avanzados. En la Figura 5.3 proporcionada se puede observar visualmente cómo el usuario interactúa con el sistema a través de distintas opciones y funcionalidades, tal como se describe en este texto.

5.3. Diseño e implementación

En esta sección se detalla el proceso de desarrollo del proyecto, en el que se va a ir describiendo paso a paso las decisiones tomadas, los problemas encontrados y las soluciones implementadas. Se abordarán las principales dificultades que surgieron durante el trabajo, así como las estrategias utilizadas para superarlas, con el objetivo de ofrecer una visión clara y reflexiva de los retos afrontados y las lecciones aprendidas a lo largo del proceso.

Sin duda, este proyecto ha sido experimental, ya que no sabíamos cuál sería su alcance final desde el principio. Al tratarse de un desarrollo iterativo y incremental, probamos, ajustamos y he ibamos perfeccionando la aplicación con cada

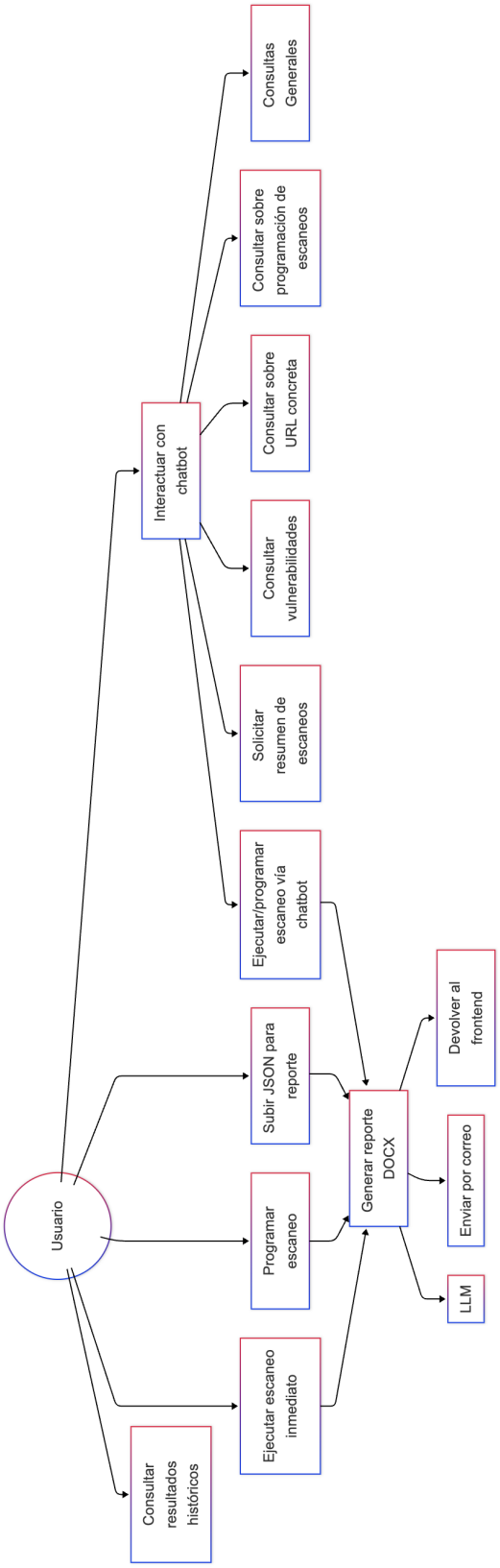


Figura 5.3: Diagrama Casos de Uso

iteración y reunion semanal con el tutor. Cada fase del trabajo se caracterizó por la exploración de nuevas ideas, la prueba de diferentes tecnologías y la integración de mejoras que respondían a los retos encontrados.

En particular, a la hora de integrar el modelos de lenguaje (LLM) supuso un pequeño reto. Dado que estos modelos, como ChatGPT o Gemini, están en constante evolución, fue necesario adaptar ciertos elementos del la herramienta para evitar que quedaran obsoletos. Esto nos obligó a replantearnos y modificar aspectos clave del proyecto a medida que se iban incorporando nuevas herramientas, lo que nos permitió que la herramienta final fuera más robusta y adaptable a las nuevas tecnológicas.

5.3.1. Fase 1: exploración de APIs y comunicación

En esta etapa inicial se hizo un estudio profundo para llegar a entender cómo es el funcionamiento y la comunicación con las APIs que integran el sistema, fundamentalmente la API de OWASP ZAP y las de distintos modelos de lenguaje (LLM).

Investigación de la API de OWASP ZAP

- **Consultas Iniciales con "requests"**:

  Empezamos realizando sencillas consultas utilizando la librería estándar requests de Python. Estas pruebas ayudaron a familiarizarnos con la estructura y los métodos de la API, aunque pronto se hizo evidente que la aproximación manual resultaba compleja para una integración eficiente.

- **Descubrimiento de la librería ZAPv2**:

  Después de varias pruebas, descubrimos la utilidad de la librería ZAPv2 de Python, creada específicamente para interactuar con la API de OWASP ZAP. Esta herramienta facilitó la integración al abstraer la dificultad y complejidad de las solicitudes HTTP y nos permitió acceder directamente a las funcionalidades de escaneo, lo que agilizó el desarrollo y la ejecución de pruebas automatizadas.

Experimentación con APIs de modelos de lenguaje (LLM):

Al mismo tiempo, se inició un proceso de investigación sobre las APIs de distintos modelos de lenguaje (LLM) pues queríamos evaluar su capacidad de respuesta y entender sus métodos de autenticación y formatos de respuesta.

- **Pruebas exploratorias y prompt engineering**:

Se realizaron consultas sencillas y aleatorias a los LLM, lo que permitió familiarizarnos con el concepto de prompt engineering. Se exploró cómo estructurar las peticiones para obtener respuestas útiles y que resultasen relevantes contextualmente. Esto no solo favoreció el entendimiento de su funcionamiento, también abrió la posibilidad de incorporar estas funcionalidades en el sistema.

- **Integración Básica del LLM**: Con el conocimiento que adquirimos, se llevaron a cabo integraciones básicas del LLM, probando funciones elementales que devolvían resultados textuales. El objetivo era comprobar la viabilidad de usar el modelo para ejecutar partes del proceso de escaneo o para enriquecer reportes.

Integración de funcionalidades basadas en LLM y ZAPv2: Como resultado de esta exploración, se concibió la idea de ejecutar funciones del sistema basadas en la información que nos devolvía el LLM. Con este postulado o premisa, se desarrollaron tres funciones esenciales:

1. **Conectar con ZAP**: Usando la librería ZAPv2, se implementó una función para establecer la conexión con OWASP ZAP garantizando así que la API estuviera perfectamente integrada y lista para recibir solicitudes.

2. **Comprobar sitios en ZAP**:

   Otra función creada es la que verifica si la URL objetivo se encuentra registrada en la lista de "sites"de ZAP, asegurando así que la conexión y la gestión de los objetivos sean coherentes antes de que se realice un escaneo activo.

3. **Ejecutar escaneo activo basado en consulta al LLM**:

   Se llevó a cabo, igualmente, el desarrollo de una función en la que, a partir de una consulta básica al LLM —por ejemplo, .^{es}caneame esta URL con esta intensidad"—, el modelo extrae datos concretos como la URL y la intensidad requerida. Estos parámetros se pasan, de manera automatizada, a las funciones previamente definidas para conectar con ZAP y ejecutar el escaneo activo, de esta manera se comprueba la viabilidad y precisión de la integración.

Resultó esenecial esta fase para establecer las bases de la integración de tecnologías críticas en el proyecto. El proceso de exploración no solo proporcionó conocimientos técnicos sobre las APIs de ZAP y los LLM, sino que a la par, también permitió validar la idea de incluir y agregar funciones complejas a través de simples consultas al modelo, estableciendo de esta manera un flujo de trabajo que sería consolidado y ampliado en etapas posteriores.

5.3.2. Fase 2: Creación de la interfaz web

El objetivo principal de esta etapa era desarrollar una aplicación web práctica y sencilla para tener una idea de cómo sería la interacción entre el usuario y la aplicación. Para nuestra primera versión de la interfaz utilizamos Flask, y a la que en la que se implementó con los formularios (forms) necesarios para que el usuario ingresara la URL que se iba a escanear y especificara la intensidad de la auditoría. Nuestra idea era proporcionar un punto de interacción directo y amigable para poner en marcha los escaneos de seguridad.

En un principio, la implementación permitió realizar escaneos simplemente introduciendo los datos que se requerían y se enviaban al servidor, donde se establecía la comunicación con la API de OWASP ZAP. Sin embargo,un inconvreniente importante nos salió al paso durante las pruebas: cuando se iniciaba un escaneo, la aplicación "se congelaba". Ello se debía a que la ejecución del proceso de escaneo bloqueaba el hilo principal, e impedía así que la interfaz respondiera y, además evitaba que el usuario realizara otras acciones. La primera interfaz web se puede ver en la siguiente Figura 5.4

Para solucionar este problema, se implementaron dos mejoras clave:

1. **Uso de AJAX**: Introdujimos AJAX para conseguir que la comunicación entre el navegador y el servidor se realizara de forma asíncrona. De esta modo, ni el envío ni la recepción de datos bloqueaban la experiencia del usuario, lo que permitía seguir navegando o interactuando con otras partes de la aplicación mientras el escaneo se ejecutaba en un segundo plano.

2. **Implementación de Threading en Python**: En paralelo, utilizamos la librería threading de Python para ejecutar el proceso de escaneo en un hilo separado. Esto permitió que el trabajo pesado se realizara de forma simultánea, se liberaba así el hilo principal y se aseguraba que la interfaz permaneciera activa y responsiva durante la ejecución del escaneo.

Estas mejoras resultaron fundamentales para proporcionar una experiencia de usuario más fluida y evitar largos tiempos de espera que afectasen la interacción.

5.3.3. Fase 3: Desarrollo de los mecanismos de la aplicación

La atención , en esta fase, se centró en la construcción e integración de los elementos esenciales que permitieran que la aplicación alcanzase sus objetivos operativos. Entre estos componentes se incluyen la mejora y conexión de la interfaz web, la persistencia de los resultados en una base de datos, la automatización

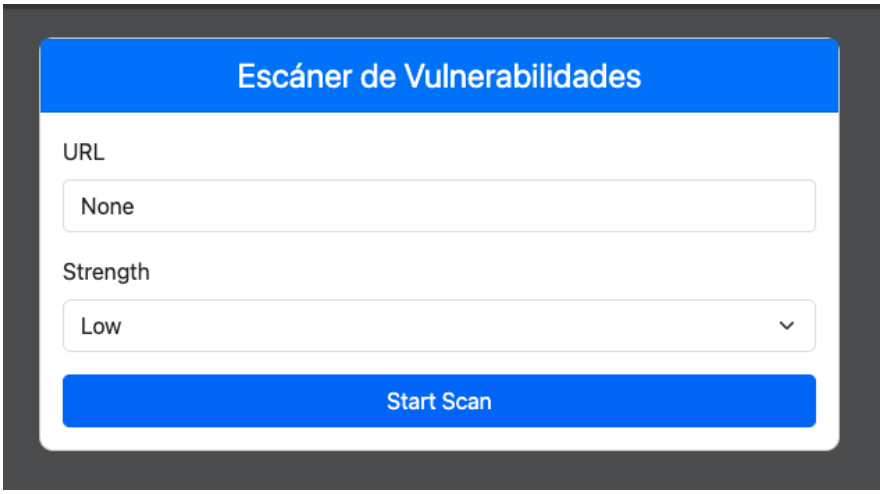


Figura 5.4: Primera Interfaz Web

organizada y coordinada de un chatbot conversacional inteligente y, finalmente,la configuración de un sistema de programación de escaneos junto con el envío automático de correos electrónicos.

1.Integración y persistencia de datos

En este punto de la Fase 3, nos centramos en cómo la aplicación podría almacenar y acceder a la información generada por los escaneos de seguridad Para conseguirlo, llevamos a cabo varias acciones importantes:

- **Elegimos una base de datos**: Al principio, se probó con SQLite. Una base de datos sencilla y fácil de usar; perfecta para las primeras etapas de desarrollo. Sin embargo, nos dimos cuenta de que cuando la aplicación creciera y tuviera más trabajo, necesitaríamos una base de datos mucho más grande y potente, por lo que a largo plazo nuestro objetivo fue remplazarla por PostgreSQL, una base de datos mucho más robusta y preparada para gestionar cantidades mucho mas grandes de información.
- **Conectamos la aplicación a la base de datos**: Para que la aplicación web hecha con Flask pudiera "hablar´´ con la base de datos, se usó una herramienta llamada SQLAlchemy. Esta librería es como un intérprete que traduce las instrucciones de Python a un lenguaje que la base de datos entiende (SQL). Esto permitió crear, leer, actualizar y eliminar información de la base de datos de una forma mucho más sencilla y segura, sin necesidad de escribir directamente en SQL todo el tiempo.
- **Gestionamos los cambios en la base de datos**: A medida que la aplicación evolucionaba y surgía la necesidad de añadir información nueva o

cambiar la manera como se guardaban los datos, utilizamos Flask-Migrate. Esta herramienta nos ayudó a realizar "migraciones.[en] la base de datos de forma controlada, de tal manera que podiamos actualizar la base de datos sin perder información.

- **Definimos los modelos para la base de datos**: En la parte inicial, se definieron los modelos para la base de datos: escaneos programados, escaneos completos, vulnerabilidades totales, reportes-vulnerabilidades-url. No solo almacenamos escaneos completos, también los programados.

- **Guardamos los resultados de los escaneos y los escaneos programados**: Cada vez que se acababa un escaneo de seguridad, implementábamos un proceso automático para tomar todos los hallazgos (las vulnerabilidades encontradas) y guardarlos de forma permanente en la base de datos. También se almacenaban los escaneos programados.

2. Desarrollo del Chatbot conversacional

En esta se explica el proceso de creación y desarrollo del asistente virtual que se ha integrado en la aplicación para facilitar cómo los usuarios interaccionan con la aplicación. El propósito del chatbot es comprender las preguntas de los usuarios y, junto a los resultados de los escaneres de seguridad, proporcionar respuestas útiles a partir del conocimiento que tiene el LLM, junto con la información almacenada en la base de datos.

En un primer momento, decidimos crear manualmente diferentes «agentes» o roles dentro del chatbot. La idea era que, al recibir la entrada del usuario, un primer agente, implementado mediante un LLM, analizara y extrajera el contexto de la consulta. De este modo, si por ejemplo un usuario pregunta por las vulnerabilidades encontradas en una URL, este primer agente detectaría la URL y el contexto relacionado y redirigiría la consulta a un segundo agente especializado en la tarea requerida.

La función de este segundo agente variaba en función de la entrada del usuario.

- **Generación de consultas SQL:** Si la pregunta requería información detallada de la base de datos (por ejemplo, "¿cuántas vulnerabilidades de tipo X se encontraron en el último escaneo?"), este agente intentaba generar una consulta SQL adecuada para que se obtuviesen los datos requeridos.

- **Extracción de información del texto:** En aquellas ocasiones en las que la consulta se refería a detalles específicos descritos en el informe, el agente se encargaba de extraer la información relevante del texto, proporcionando así respuestas precisas sobre los resultados.

- **Ejecución de funciones específicas:** En algunos casos, si las peticiones implicaban la ejecución de funciones internas del sistema, el agente era capaz de identificar la necesidad de activar dichas funciones.

Una parte crucial del desarrollo fue el ajuste de los *prompts* que se enviaban al LLM. Un *prompt* es, básicamente, la instrucción que se le proporciona al modelo para que genere una respuesta. Dedicamos una gran cantidad de tiempo a probar y ajustar estos *prompts*, queríamos asegurarnos de que fuesen lo suficientemente claros y precisos como para que el LLM entendiese correctamente las preguntas de los usuarios y generara respuestas adecuadas. Este proceso fue iterativo e implicó numerosas pruebas y correcciones, lo cual fue esencial para mejorar la precisión y relevancia de las respuestas generadas por el chatbot.

A medida que avanzaban las pruebas, nos dimos cuenta de las limitaciones inherentes a la creación manual de agentes. Fue entonces cuando, después de una tutoría, y bajo la recomendación del tutor, vimos el potencial de una herramienta/libreria llamada **LangChain**; en nuestro caso, optamos por **LangGraph**. LangGraph es un marco basado en LangChain que permite estructurar sistemas multiagente complejos y gestionar, de forma controlada, flujos de trabajo conversacionales , a través de la definición de grafos cíclicos y una robusta gestión del estado de las interacciones. Este hallazgo nos condujo a reconocer que, para un desarrollo futuro del chatbot, sería esencial aprovechar las avanzadas capacidades que nos ofrece LangChain y sus extensiones.

3. Envío automático de correos electrónicos

En este punto, nos centramos en cómo la aplicación podía comunicar automáticamente los resultados de los escaneos a los usuarios y facilitarles informes detallados. Imaginémonos recibir un correo electrónico justo después de finalizar un análisis, con un resumen de lo encontrado y con un informe completo para que lo examinemos.

Para lograr este objetivo, implementamos lo siguiente:

- **Generación de reportes**: Usamos la capacidad de OWASP ZAP para generar informes detallados sobre las vulnerabilidades halladas durante un escaneo. Estos informes suelen estar en formatos como HTML o XML y contienen una exhaustiva información técnica sobre cada hallazgo.

- **Envío automático de correos electrónicos**: Dearrollamos una función dentro de la aplicación que se activaba automáticamnete una vez finalizado el análisis. Era la encargada de crear un correo electrónico. De esta forma, se empezó a utilizar y la librería “email“ de Python.

  - **email.mime.multipart**: Esto nos permitió crear correos electrónicos que podían contener diferentes tipos de contenido, como texto y archivos adjuntos (el informe).
  - **email.mime.text**: Se usó para crear el cuerpo del correo electrónico, que podía ser texto simple o incluso formato HTML para una presentación mucho mejor.
  - **email.mime.base**: Proporcionó la base para incorporar el archivo del reporte que se había generado al correo electrónico.

- **Adjuntar el reporte**: El informe generado por ZAP se adjuntaba al correo electrónico para que el usuario pudiera descargarlo y revisarlo.

- **Enviar el correo electrónico**: en último lugar, la función usaba los protocolos de envío de correo electrónico (como SMTP mediante SENDGRID) para enviar el mensaje a la dirección de correo electrónico que se había configurado para el usuario.

4. Programación de escaneos automatizados

Aquí, el objetivo era que la aplicación pudiese ejecutar análisis de seguridad de forma programada, sin que el usuario estuviera obligado a iniciarlos manualmente cada vez. Imaginemos configurar un horario para que la aplicación chequee, de forma regular, la seguridad de tus sitios web automáticamente.

Al principio, tuvimos la idea de implementarlo a través de ZAP:

- **Automatización de la creación de planes en el framework de ZAP**: Se nos ocurrió la idea de usar un LLM para generar automáticamente el archivo YAML que el ZAP Automation Framework utiliza para ejecutar escaneres. El Automation Framework de ZAP nos permite definir planes de escaneo complejos con múltiples pasos. La idea era que el LLM actuase como un .ªrquitecto"de estos planes. No obstante, como dijimos antes, el LLM que con el que estábamos trabajando en ese momento no fue lo suficientemente preciso para generar, de manera fiable estos archivos de configuración.

Las dificultades con la automatización a través del LLM y el ZAP Automation Framework, nos obligó a adoptar un enfoque diferente utilizando la librería APScheduler de Python:

- **Implementación de funciones de programación**: Utilizamos APScheduler para crear funciones dentro de nuestra aplicación que se ocupaban de gestionar la programación de los escaneos.

- **Revisión de escáneres no añadidos**: Implementamos una función que comprobaba de forma periódica si existía algún escaneo que el usuario hubiese configurado, pero que aún no se había añadido a la lista de tareas pendientes para ser ejecutadas en la base de datos. Esto funcionaba como un mecanismo de seguridad para asegurar que no se pasaba por alto ningún escaneo que estuviese programado.

- **Bucle de verificación de horarios**: Creamos un bucle que se ejecutaba cada diez minutos. Lo que hacía ese bucle era consultar la base de datos para comprobar si había algún escaneo programado para ese momento o para un futuro próximo. Si se encontraba con un escaneo que debía ejecutar, iniciaba el correspondiente proceso.

5. Limitaciones económicas y feedback del usuario:

Este último apartado dentro de la descripción de la Fase 3 destaca dos aspectos cruciales que influyeron en el desarrollo:

- **Limitaciones económicas**: Es muy importante señalar que el uso de las APIs de los modelos de lenguaje (LLMs) tenía un costo asociado. Siempre que hacíamos una consulta al modelo, teníamos que pagar una cantidad. Esto, lógicamente, nos obligó a ser muy prudentes y estratégicos a la hora de utilizar los LLMs durante el desarrollo del chatbot y en nuestra tentativa de automatizar la creación de planes de ZAP. Nos vimos obligados a priorizar las pruebas y el ajuste de los prompts en los casos más importantes y buscar formas de optimizar el uso de estos recursos limitados. Esta limitación económica fue un factor importante en el momento de tener que tomar decisiones técnicas y explorar alternativas.

- **Feedback de usuario**: Al terminar de esta fase, presentamos la aplicación a varias personas para que la probasen y nos dieran sus impresiones. Este proceso de retroalimentación fue bastante valioso. Observamos de esta manera cómo interactuaban los usuarios con la interfaz web; pudimos probar la utilidad y la claridad de las respuestas del chatbot, y recopilamos ideas y sugerencias para mejorar la aplicación en el futuro. Este feedback nos ayudó a identificar áreas donde la aplicación podría ser más intuitiva, más útil y más eficiente, y sentó las bases para las futuras mejoras que se llevarían a cabo en las siguiente fases del proyecto.

Los comentarios recibidos de los usuarios y las pruebas realizadas a lo largo del proceso nos permitieron identificar áreas críticas para la mejorara de la aplicación. A medida que avanzábamos, se hizo evidente la necesidad de migrar la persistencia de datos a una solución más robusta, cambiando de SQLite a PostgreSQL;con

ello conseguíamos garantizar una gestión de información más eficiente y escalable en un entorno de producción.

Del mismo modo, en el proceso de desarrollo del chatbot conversacional, la experiencia adquirida por la implementación de agentes manuales reveló ciertas limitaciones: la fluidez del sistema se veía comprometida por la incapacidad para mantener el contexto de conversación y se producían errores en la interpretación de las solicitudes. Estas deficiencias nos llevaron a buscar la solución a través del uso de agentes basados en LangChain, y finalmente descubrimos que, para ofrecer una experiencia conversacional más controlada y estructurada, se hacía necesario migrar a LangGraph. Esta herramienta nos permite gestionar grafos cíclicos y mantener un estado de interacción coherente, aspectos fundamentales que se habían quedado cortos en las anteriores implementaciones.

Además, el feedback de los usuarios también puso de manifiesto la necesidad de contar con una herramienta visual para planificar y supervisar los escaneos, lo que originó que concibiéramos la idea de desarrollar un calendario interactivo donde se visualizaran tanto los escaneos ejecutados como los programados. Esta funcionalidad facilitará la monitorización y organización de las auditorías de seguridad, a la par que aportará un valor añadido al sistema.

Resumiendo, esta fase ha sido decisiva para detectar y afrontar las limitaciones del prototipo inicial, lo que ha orientado el proyecto hacia soluciones más robustas y eficientes, tanto en la gestión de datos como en la interacción conversacional y la planificación de tareas. Estas mejoras son reflejo nuestro compromiso continuo con la optimización de la herramienta y su capacidad para adaptarse a las necesidades reales de los usuarios.

5.3.4. Fase 4: Optimización y depuración de módulos

Desplegadas las funcionalidades de la anterior fase, el trabajo se centró en integrar las sugerencias que habíamos recibido a través del feedback. Trabajamos por tanto en la inclusión de un calendario interactivo para la planificación de auditorías y la ampliación de las capacidades del chatbot. Al principio, se optó por construir el asistente conversacional mediante agentes de LangChain; no obstante, durante la compilación del código, aparecieron avisos de obsolescencia, lo que nos hizo migrar la lógica a LangGraph; esto se tradujo en un flujo de conversación más eficiente y en una coordinación de herramientas orientada a grafos de estado. Al mismo tiempo, se agregó un módulo generador de informes que, a partir de una plantilla DOCX personalizada y con el apoyo de la librería python-docx, genera automáticamente la descripción de cada una de las vulnerabilidades, evalúa su riesgo y propone recomendaciones de mitigación, todo ello gracias a su integración con un LLM.

1. Actualización y mejora del chatbot conversacional

Como se mencionó en fases anteriores, primero optamos por una implementación manual del chatbot, donde el flujo de la conversación se gestionaba de forma secuencial y con escasa flexibilidad. Las limitaciones de este enfoque pronto dieron la cara: el chatbot no podía mantener el contexto de la conversación, se olvidaba de interacciones previas y, en la mayoria de los casos, era incapaz de resolver preguntas encadenadas o complejas. Además, la falta de una estructura clara para el manejo de diferentes tipos de solicitudes hacía que el sistema fuera propenso a errores.

La necesidad de mejora y tras una reunión de seguimiento que mantuvimos con nuestro tutor, nos llevó a probar los agentes de LangChain. Esto nos proporcionaría una capa de autonomía y modularidad. -Un agente en LangChain es un componente capaz de tomar decisiones de forma autónoma, interactuar con diversas herramientas y seleccionar la mejor estrategia para resolver una solicitud del usuario- . Esto supuso una mejora significativa: el chatbot podía, por ejemplo, decidir si debía consultar la base de datos, generar un resumen o invocar una función específica. No obstante, durante la integración, se nos informó de que los agentes de LangChain iban a quedar obsoletos; por ello nos vimos obligados a buscar una alternativa más sostenible y moderna.

En este punto, e investigando un poco, dimos con LangGraph. LangGraph es un marco desarrollado sobre LangChain, diseñado para construir sistemas multiagente complejos y flujos de trabajo conversacionales de manera organizada, estructurada y controlada. Entre sus principales ventajas están la capacidad de definir grafos cíclicos y gestionar el estado de la conversación de forma robusta, lo que permite al chatbot recordar el contexto, mantener la coherencia y adaptar su comportamiento a medida que evolución la interacción.

Arquitectura del chatbot con LangGraph

La migración a LangGraph implicó aprender una nueva forma de estructurar el código y el flujo conversacional. En el núcleo de este sistema se encuentra el StateGraph, una estructura que representa el estado actual del flujo conversacional, incluidos tanto los datos relevantes como el contexto de la interacción. El StateGraph se define utilizando un TypedDict, que es un tipo especial de diccionario en Python en el que cada clave corresponde a un tipo de mensaje o estado, y su valor es una lista de mensajes asociados. Esto nos permite organizar y acceder rápidamente a la información más relevante de cada conversación.

El Graph Builder es el componente responsable de construir el grafo conversacional. Aquí se definen los distintos nodos que componen el flujo, las herramientas disponibles, y las posibles transacciones entre estados. El resultado es una estruc-

tura clara y visualmente interpretable, que facilita tanto el desarrollo como el mantenimiento del sistema.

Dentro de este grafo, los nodos representan los diferentes pasos o estados por los que la conversación puede pasar. Los nodos principales suelen ser:

- **chatbot**: donde el LLM procesa la entrada del usuario, interpreta la intención y decide la siguiente acción.
- **tools**: donde se ejecutan funciones específicas, tales como consultar vulnerabilidades, obtener resúmenes o acceder a información de escaneos programados. Estas funciones están decoradas con @tool, lo que las hace accesibles para el LLM.
- **output**: nodo de salida que se utiliza cuando la respuesta que se ha generado es definitiva (ejemplos: un resumen ejecutivo o una respuesta directa).
- **start**: nodo inicial que marca el inicio del flujo de la conversación.

Las edges (transiciones) y edges condicionales determinan el movimiento dela conversación entre nodos. Por ejemplo, después de procesar la entrada en el nodo chatbot, el flujo puede dirigirse al nodo tools si es necesario ejecutar una función, o bien ir directamente al nodo output si la respuesta está lista. Además, tras ejecutar una herramienta, el flujo puede volver al chatbot para continuar la conversación o terminar en output si la información solicitada ya se ha conseguido.

Un componente fundamental es la memoria (memory), que le permite al chatbot recordar interacciones previas, manteniendo el contexto a lo largo de la conversación. Esto se gestiona mediante objetos como MemorySaver y checkpointer, los cuales almacenan el estado del grafo y permiten recuperar el historial cuando se necesita. La configuración de la memoria se puede adaptar a cada hilo de conversación, lo que garantiza que cada usuario tenga una experiencia personalizada y coherente.

Diagrama de flujo del LangGraph

- El flujo comienza en el nodo START, que da inicio a la conversación.
- El Nodo Chatbot interpreta la información introducida por el usuario y decide si debería invocar una herramienta (Nodo Tools) o si podría dar una respuesta directa (Nodo Output).
- En caso de que se necesitase una herramienta, el flujo puede volver al Nodo Chatbot para continuar la conversación o finalizar en el Nodo Output si la respuesta es definitiva.

- Este diseño permite ciclos y ramificaciones, gestionando el contexto y la memoria de la conversación continuamente.

El diagrama de flujo de LangGraph se puede ver en la siguiente Figura 5.5

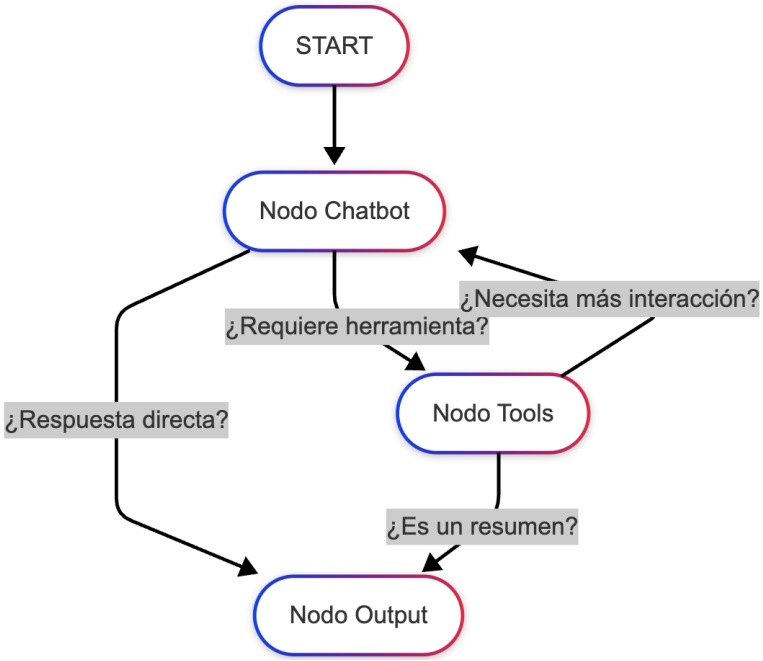


Figura 5.5: Diagrama-LangGraph

Generación de reportes customizados a partir de un template DOCX

Otra de las funcionalidades implementadas durante esta etapa fue la generación de reportes a partir de un template DOCX, lo que permite presentar los resultado de los escaneos de seguridad de forma mas profesional y estructurada. Para ello, se empleó un documento Word (DOCX) customizado, previamente diseñado con imágenes, texto y etiquetas de sustitución delimitadas por llaves. Estas etiquetas sirven como marcadores donde se insertará la información dinámica lo que permite adaptar el contenido del template a cada uno de los informes generados.

El proceso de generación de reportes se desarrolla de la siguiente manera:

1. **Lectura del template y reemplazo de Etiquetas**: Se utiliza la librería python-docx para cargar el documento template. Tanto el texto predefinido como las etiquetas de remplazo se encuentran dentro del template. Esto nos garantiza que el reemplazo de texto se realice de forma correcta y que se mantenga el estilo original del documento(incluyendo imágenes y formatos).

2. **Iteración sobre las alertas**: El sistema revisa cada una de las alertas detectadas durante el proceso de escaneo (o las extraídas de un archivo JSON en caso de generación a partir de datos existentes). Para cada alerta, se creará una copia de una tabla con las etiquetas que deben sustituirse.La siguiente información se inserta en cada etiqueta dentro de cada tabla:

   - El detalle enriquecido de la alerta.
   - La evaluación del riesgo asociado.
   - La solución recomendada.
   - La categoría según el OWASP Top 10 2021 a la que pertenece la vulnerabilidad.

   Para conseguir esta información, se invoca al LLM de forma secuencial, lo que garantiza que cada alerta se procese individualmente. Esto permite una respuesta individual para cada alerta, lo que la hace mucho mas precisa.

3. **Generación del resumen ejecutivo**: Además de procesar cada alerta de forma individualizada, el sistema genera un resumen ejecutivo para el reporte. Esta síntesis global se obtiene mediante otra llamada al LLM, el cual analiza el conjunto de alertas y otros datos relevantes para producir un texto que resuma la situación general tras el escaneo. Dicho resumen se inserta en una sección específica del documento mediante etiquetas, proporcionando así una visión global de la situación actual del activo.

4. **Integración y envío del reporte**: Una vez completado el documento con las tablas generadas para cada alerta y el resumen ejecutivo, el archivo final se guarda en formato DOCX y se envía por correo a través de un sistema de correo electrónico (SendGrid).

5.3.5. Fase 5: Maduración y validación en entorno real

En esta última fase del proyecto nos centramos en dotar a la plataforma el nivel de madurez necesario para funcionar correctamente en entornos reales. Queríamos asegurarnos de que fuera estable, fácil de compilar en distintos sistemas y fácil de mantener. Era el momento de unir todo lo aprendido en las etapas anteriores y dejar la herramienta lista para su despliegue y validación en condiciones de uso real.

Corrección de errores y mejora continua

En primer lugar, nos enfocamos en depurar los últimos errores y mejorar el aspecto y el rendimiento de la plataforma. Para mantener el trabajo bien organizado, creamos una rama de corrección en el repositorio donde anotábamos

y solucionábamos cada una de las incidencias que surgían durante las pruebas. Esto nos permitió mejorar la aplicación , añadir funciones nuevas y mejorar la experiencia del usuario.

Contenerización y portabilidad con Docker

Uno de los objetivos que nos pusimos a medida que avanzaba el desarrollo de la aplicación fue poder desplegar la aplicación en cualquier sistema operativo por lo que la creacion de contenedores Docker fue fundamental para cumplirlo. Desarrollar tres contenedores principales: uno dedicado a la aplicación(zap-web), otro para la base de datos(db) y un tercero para OWASP ZAP(zap). Además, integramos Gunicorn, configurado para producción, y que actúa como un servidor HTTP WSGI(Web Server Gateway Interface).

Esta solución de contenerización no solo asegura que la herramienta se ejecute de manera rápida y reproducible en distintos entornos, sino que también reduce significativamente los problemas relacionados con dependencias y mejora la portabilidad global del sistema. Docker nos permitió orquestar de forma sencilla los distintos componentes del sistema –backend, base de datos y servicios externos–.

Ampliación de funcionalidades: endpoint para reportes desde JSON

En esta fase, desarrollamos un punto final que permite generar informes de forma automática a partir de archivos JSON creados manualmente con la herramienta ZAP. Esta funcionalidad no fue muy difícil de implementar, ya que se reutilizó el código usado en la generación de informes en la sección anterior. Además, permite que los usuarios puedan generar informes no solo a partir de análisis en tiempo real, sino también a partir de resultados obtenidos previamente.
.

El punto final procesa el archivo JSON, extrae las alertas, los datos relevantes, y utiliza el mismo mecanismo que en el apartado anterior para generar informes DOCX utilizando el LLM.

Validación en entorno real y seguimiento del tutor

Durante esta fase, fue necesario validar y probar la herramienta en un entorno real para verificar que era capaz de funcionar en condiciones reales. Para ello, la universidad nos proporcionó acceso a dos máquinas virtuales desplegadas específicamente para las pruebas: una actuaría como atacante, ejecutando escáneres web de seguridad, y la otra funcionaria como máquina objetivo, simulando un entorno real en el que se pudieran detectar las vulnerabilidades y generar los

informes correspondientes a esos escáneres. Esta infraestructura nos permitió recrear un escenario de ataque y defensa muy similar al que se encontraría en un entorno profesional.

La experiencia de implementar la aplicación en este entorno real fue especialmente positiva, ya que pudimos comprobar que podía ser una aplicación eficaz y eficiente. No solo permitió validar la eficacia de la plataforma, sino que también pudimos tanto probar como validar la portabilidad y la facilidad de implementación que buscábamos desde el principio. El hecho de poder configurar toda la infraestructura con Docker nos ahorró tiempo y evitó las complicaciones habituales relacionadas con las dependencias o incompatibilidades.

5.4. Análisis estático de código

Para garantizar la calidad y seguridad de la solución implementada, se ha llevado a cabo un análisis del código fuente utilizando SonarQube, una plataforma de análisis estático que permite identificar vulnerabilidades y bugs.

5.4.1. Análisis Inicial

Como se puede observar en la Figura 5.6, el análisis inicial del código reveló varias áreas de mejora:

- **Seguridad**: Se detectaron 6 problemas de seguridad (calificación E) relacionados principalmente con claves API en texto plano en el código.
- **Fiabilidad**: Se encontraron 2 problemas de fiabilidad (calificación C) que incluían código no utilizado o inservible.
- **Mantenibilidad**: Se identificaron 48 problemas de mantenibilidad (calificación A).
- **Puntos Críticos de Seguridad (Security Hotspots)**: 29 áreas del código fueron señaladas como potencialmente problemáticas desde el punto de vista de seguridad.

5.4.2. Mejoras implementadas

Tras identificar estos problemas, se llevó a cabo un proceso de revisión y corrección del código, enfocándose primeramente en las vulnerabilidades de seguridad y fiabilidad. Como resultado, el segundo análisis (mostrado en la segunda 5.7 imagen) refleja una mejora significativa:

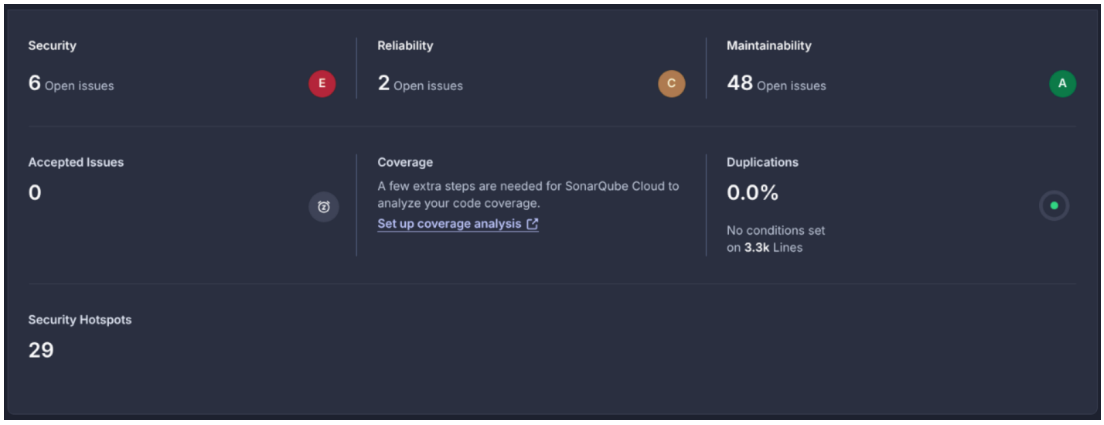


Figura 5.6: SonarQube Análisis Inicial

- **Seguridad**: Resolución completa de los 6 problemas iniciales, alcanzando una calificación A.
- **Fiabilidad**: Corrección de los 2 problemas identificados, alcanzando también una calificación A.
- **Mantenibilidad**: Reducción a 38 problemas (manteniendo calificación A).
- **Puntos Críticos de Seguridad (Security Hotspots)**: Disminución a 24 áreas de revisión.

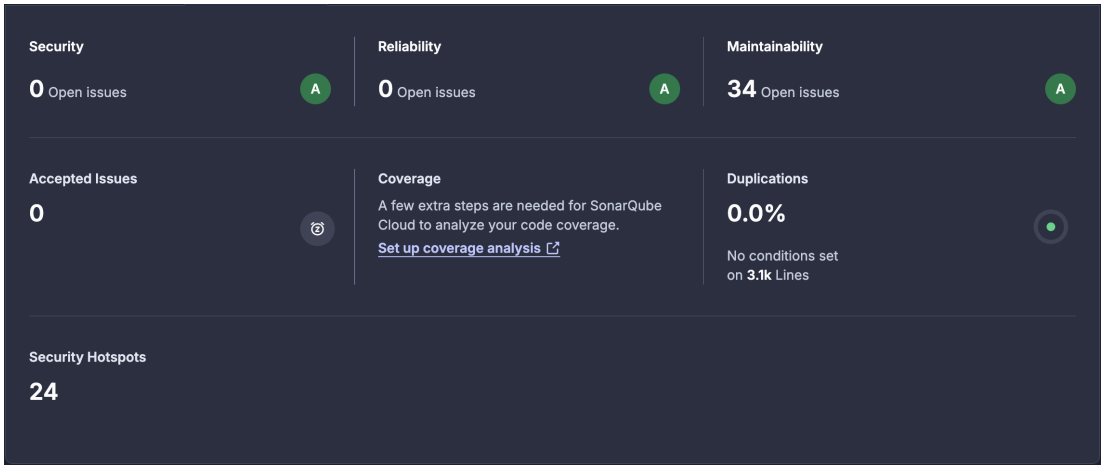


Figura 5.7: SonarQube Mejoras

Tras revisar y mejorar el código, siguen apareciendo algunas advertencias en el análisis de SonarQube. Sin embargo, es importante señalar que se trata en su mayoría de advertencias habituales en proyectos web. Entre algunas de las advertencias destacan:

- El uso de métodos HTTP seguros e inseguros en determinadas rutas responde a la necesidad de interactuar correctamente tanto con la herramienta como la API de OWASP ZAP
- La copia recursiva de archivos en el Dockerfile es una práctica habitual para garantizar que todos los recursos necesarios estén presentes en el contenedor.
- La inclusión de scripts externos como chart.js sin el atributo de integridad es una advertencia habitual en el desarrollo frontend.

5.5. Distribución y despliegue

Una vez finalizadas las fases de desarrollo, se procedió a empaquetar la aplicación utilizando Docker para asegurar que el sistema se desplegara de forma rápida, reproducible y en cualquier sistema operativo. Con Docker se logró encapsular todos los componentes de la aplicación—la aplicación, la base de datos y OWASP ZAP— en contenedores individuales, facilitando tanto la integración entre ellos como su posterior mantenimiento y escalabilidad. De esta manera, simplemente será necesario clonarse el repositorio de Github y seguir los pasos que se mencionan un poco mas abajo.

Estructura del repositorio

El repositorio GitHub del proyecto contiene los siguientes archivos:

- **Dockerfile**: Documento que define la imagen de la aplicación. En él se especifica la base de imagen (en este caso, python:3.13-slim), se instalan las dependencias, se copia el código fuente y se configura Gunicorn como servidor en producción.
- **Docker-compose.yml**: Archivo que orquesta la puesta en marcha de la aplicación a través de varios contenedores. Se definen tres servicios principales:
  - **web**: El contenedor que aloja la aplicación, configurado para exponer el puerto 5000 y recibir las variables de entorno necesarias (como claves API, configuración de Flask y parámetros de la base de datos).
  - **db**: Un contenedor basado en PostgreSQL (versión 13), encargado de gestionar la persistencia de datos.
  - **zap**: El contenedor de OWASP ZAP, que actúa en modo daemon, recibiendo la configuración necesaria para operar y generar reportes.

- **Toda la lógica de la aplicación**

Adicionalmente, se incluye un archivo README que explica la aplicación, el método de despliegue y la configuración necesaria del entorno, así como la forma de establecer las variables necesarias en un archivo .env, el cual debe contener los parámetros que se ven en el apéndice.

Proceso de despliegue

```
docker-compose up -d --build

docker exec -it zap_web bash.
    flask db init
    flask db migrate -m "fist migration"
    flaks db upgrade
```

Además, se incluirá un ejemplo práctico en formato de video, en el que se demuestra de forma detallada y visual cada uno de los pasos necesarios para desplegar y utilizar la aplicación. Este material multimedia está diseñado para que los usuarios puedan seguir de manera sencilla el proceso completo y facilitando la comprensión del funcionamiento interno de la plataforma.

6

Conclusiones y trabajos futuros

6.1. Reflexión personal sobre el trabajo realizado

Desarrollar esta plataforma de auditoría web ha sido una experiencia increíblemente enriquecedora. He tenido la oportunidad de descubrir el verdadero potencial de los modelos de lenguaje (LLMs) cuando se combinan con herramientas técnicas como puede ser el OWASP ZAP. Desde el inicio, mi meta fue facilitar el acceso a la ciberseguridad, creando una solución intuitiva que aprovechera el potencial de los LLM. Aunque creo que he cumplido este objetivo satisfactoriamente, el camino también me ha mostrado claramente las fortalezas y las limitaciones reales de la inteligencia artificial en entornos prácticos.

6.2. Logros alcanzados

Una de las cosas más destacadas de este proyecto ha sido la sorprendente habilidad del LLM para producir contenidos útiles, claros y perfectamente contextualizados. Algunas contribuciones clave han sido:

- **Reportes enriquecidos**: El LLM automatiza eficientemente partes importantes de los informes técnicos, ofreciendo descripciones detalladas, evaluaciones precisas del riesgo y recomendaciones prácticas específicas.

- **Resúmenes ejecutivos**: El modelo ha facilitado mucho la comunicación con equipos directivos, convirtiendo las aletas del ZAP en informes claros sin entrar en detalles técnicos profundos.

- **Explicaciones sencillas**: El chatbot ha ayudado enormemente a traducir conceptos técnicos avanzados como "SQL Injection."[en] explicaciones fáciles de entender usando analogías y ejemplos prácticos. Esto ha beneficiado especialmente a desarrolladores junior o profesionales sin formación especializada.

- **Mejoras constantes**: Además, el LLM ha sido capaz de sugerir mejoras prácticas basadas en la interacción real con los usuarios, como incluir filtros personalizados o priorizar automáticamente ciertos análisis según el historial previo de vulnerabilidades.

Estas funcionalidades no solo mejoran significativamente la utilidad práctica de OWASP ZAP, sino que también ofrecen una capa adicional de interpretación y análisis ausente en muchas herramientas tradicionales.

6.3. Limitaciones observadas

A pesar de estos avances, también me he encontrado con limitaciones importantes, especialmente en la capacidad del LLM para distinguir falsos positivos. En muchos casos, el modelo asumía que la vulnerabilidad era real cuando en realidad no lo era.

Esto se debe a que el modelo solo recibía información muy limitada, como el vector de ataque y la evidencia proporcionada por ZAP. Al presentar un ataque con una evidencia asociada, el LLM, sin acceso a contexto adicional como el código fuente o la capacidad de ejecutar el ataque para verificar su validez, no podía saber con certeza si se trataba de una vulnerabilidad real o no. Como resultado, en muchos casos terminaba clasificando como válidos algunos falsos positivos.

Estas limitaciones surgen principalmente por dos razones:

1. **Falta de contexto más amplio**: El modelo solo recibe información puntual de ZAP, sin acceso a logs completos, código fuente o configuraciones detalladas del servidor.

2. **Sesgo precautorio**: Los modelos tienden a alertar en exceso como mecanismo preventivo, lo que incrementa considerablemente la cantidad de falsos positivos.

6.4. Ideas para futuras mejoras

Para abordar estas limitaciones propongo las siguientes mejoras en futuras versiones:

- **Fine-tuning con datos específicos**: Entrenar al modelo con casos etiquetados de falsos positivos frecuentes para mejorar su capacidad de discernimiento.
- **Integración con herramientas de análisis estático**: Complementar los resultados dinámicos con análisis estáticos mediante herramientas como Semgrep o SonarQube, para ofrecer más contexto al modelo.
- **A nivel de aplicación** se plantea la posibilidad de segmentar las conversaciones y reportes por activos dentro de la aplicación, de modo que cada desarrollador pueda acceder y consultar únicamente la información relevante a las auditorías de los sistemas o aplicaciones.

6.5. Conclusiones personales

Este proyecto ha reforzado mi creencia en el gran potencial de los modelos de lenguaje, pero también me ha hecho consciente de sus limitaciones prácticas y de la importancia crucial del juicio humano experto. La IA tiene el poder de transformar cómo generamos y interpretamos el contenido técnico, pero no puede reemplazar por completo la necesidad del análisis crítico especializado, especialmente en un área tan delicada como la ciberseguridad.

Como desarrollador, he aprendido mucho sobre cómo la combinación de diferentes tecnologías (como ZAP, LangGraph y Docker) puede resultar en soluciones útiles y accesibles para un público amplio. Sin embargo, también he aprendido que, por muy avanzada que sea la inteligencia artificial actualmente, todavía no es capaz de resolver todos los problemas, particularmente la identificación precisa de amenazas reales frente a falsas amenazas.

Considero este proyecto no solo es una aportación valiosa en ciberseguridad, sino también una reflexión crítica importante sobre el rol de la inteligencia artificial en nuestra sociedad.

Bibliografía

[1] "ZAP community - LLM integration (accedido en mayo de 2025)," https://groups.google.com/g/zaproxy-users/c/Rrd3SOga4AE?pli=1, 2024.

[2] "ZAP updates - octubre 2024: avances en integración con LLM (accedido en mayo de 2025)," https://www.zaproxy.org/blog/2024-11-01-zap-updates-october-2024/, 2024.

[3] "Vulbot: escáner de vulnerabilidades basado en chatbot (accedido en mayo de 2025)," https://www.gutech.edu.om/research-and-consultancy-office/research-projects/block-funding-program/vulbot-the-system-vulnerability-scanner-chatbot/, 2024.

[4] "Llm_report_generator: generación automática de informes con LLM (accedido en mayo de 2025)," https://github.com/mrjxtr/LLM_Report_Generator, 2024.

[5] OWASP Foundation, "OWASP ZAP Documentation," 2022, recuperado de https://owasp.org/www-project-zap/.

[6] ——, "Getting Started – Zed Attack Proxy (ZAP)," n.d., recuperado de https://www.zaproxy.org/getting-started/.

[7] Jit.io, "6 Essential Steps to Using OWASP ZAP for Penetration Testing," n.d., recuperado de https://www.jit.io/resources/owasp-zap/6-essential-steps-to-use-owasp-zap-for-penetration-testing.

[8] OWASP Foundation, "ZAP API Documentation," 2022, recuperado de https://www.zaproxy.org/docs/api/.

[9] Python Software Foundation, "Python 3 Reference Manual," 2022, recuperado de https://docs.python.org/3/.

[10] Pallets Projects, "Flask Documentation," 2022, recuperado de https://flask.palletsprojects.com/.

[11] PostgreSQL Global Development Group, "About PostgreSQL," n.d., recuperado de https://www.postgresql.org/about/.

[12] T. B. Brown, B. Mann, N. Ryder, M. Subbiah *et al.*, "Language Models are Few-Shot Learners," 2020, recuperado de https://arxiv.org/abs/2005.14165.

[13] OpenAI, "ChatGPT API Documentation," 2022, recuperado de https://platform.openai.com/docs/.

[14] "Google gemini api: documentación oficial (accedido en mayo de 2025)," https://ai.google.dev/gemini-api/docs?hl=es-419, 2024.

[15] jQuery Foundation, "jQuery Official Documentation," 2021, recuperado de https://api.jquery.com/.

[16] J. Resig, "jQuery," 2006, recuperado de https://jquery.com.

[17] M. Bayer *et al.*, "SQLAlchemy Documentation," 2022, recuperado de https://docs.sqlalchemy.org/.

[18] LangChain Community, "LangChain Documentation," 2023, recuperado de https://langchain.readthedocs.io/.

[19] LangGraph Project, "LangGraph Documentation," 2023, información adicional disponible en recursos oficiales o repositorios públicos.

[20] GetZep, "Complete Guide to Building LangChain Agents with the LangGraph," 2024, 1 Jan 2024, https://www.getzep.com/ai-agents/langchain-agents-langgraph.

[21] APScheduler authors, "APScheduler Documentation (stable)," 2023, recuperado de https://apscheduler.readthedocs.io/en/stable.

[22] ——, "APScheduler Documentation," 2022, recuperado de https://apscheduler.readthedocs.io/.

[23] python-docx contributors, "python-docx Documentation," 2023, recuperado de https://python-docx.readthedocs.io/.

[24] J. Mitchell, "Working with python-docx," n.d., recuperado de recurso similar disponible en línea.

[25] Docker Inc., "Docker Documentation," 2023, recuperado de https://docs.docker.com.

[26] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," 2014, recuperado de https://www.docker.com.

[27] "Twilio sendgrid: documentación oficial (accedido en mayo de 2025)," https://www.twilio.com/docs/sendgrid, 2024.

[28] Microsoft, "Visual Studio Code," 2023, recuperado de https://code.visualstudio.com.

[29] GitHub, "GitHub Documentation," 2023, recuperado de https://docs.github.com.

[30] S. Chacon and B. Straub, "Pro Git," 2014, recuperado de https://git-scm.com/book/en/v2.

[31] Atlassian, "Git Tutorial: What is Git?" 2022, recuperado de https://www.atlassian.com/git/tutorials/what-is-git.

[32] SonarQube Documentation, "SonarQube Documentation," 2022, disponible en https://docs.sonarqube.org.

[33] SonarQube Community, "Recursos y publicaciones especializadas," n.d., información adicional disponible en línea.

[34] Codeium, "Windsurf: Next-generation AI IDE," 2025, recuperado de https://www.codeium.com/.

Apéndices

A

Primer apéndice

A.1. Repositorio de Github

Este es el link al repositorio de Github para poder visualizar el código completo.

`https://github.com/ZAP-TFG/zap_automatice_web`

B

Segundo apéndice

B.1. Parametros de configuración .env

A continuación se detallan los parámetros de configuración necesarios en el archivo .env para el correcto funcionamiento de la aplicación. Estas variables de entorno son requisitos indispensables para la inicialización y operación adecuada del sistema.

```
ZAP_API_KEY=
ZAP_URL=http://zap:8090

GEMINI_API_KEY=
OPENAI_API_KEY=
SENDGRID_API_KEY=

REPORT_DIR=/app/reportes

APP_USERNAME=
APP_PASSWORD=
FLASK_SECRET_KEY=

PSQL_HOST=db
PSQL_PORT=5432
PSQL_USER=
PSQL_PASSWORD=
DB_NAME=zap_data_base
```