

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“IMPLEMENTACIÓN DE MEJORAS A UNA PLATAFORMA
PARA LA REPORTERÍA DE AUDITORÍAS DE
CIBERSEGURIDAD, GUIADA POR UNA EVALUACIÓN DE
ARQUITECTURA DE SOFTWARE: CASO AUDITFORGE”

JULIO AMARU CERDA VILLABLANCA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Pablo Cruz Navea
Profesor Correferente: Pedro Godoy Barrera

Julio - 2025



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Implementación de mejoras a una plataforma para la reportería de auditorías de ciberseguridad, guiada por una evaluación de arquitectura de software: Caso AuditForge

Nombre del candidato(a): Julio Amaru Cerda Villablanca

Carrera / Grado: Ingeniería Civil Informática

Campus: Casa Central Valparaíso ; **Departamento:** Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Pablo Cruz Navea, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 05-08-2025

; Firma:

Estudiante o Candidato(a):

Fecha: 05-08-2025

; Firma:

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

AGRADECIMIENTOS

Quiero agradecer a todas las personas que me han acompañado a lo largo de mi vida, tanto en lo académico como en lo personal. Cada momento vivido, reído y también sufrido, ha sido parte de un camino que me ha formado profundamente. Este proyecto es la culminación de ese recorrido, y también quiero agradecer especialmente a quienes fueron parte de él.

A mi pareja, Sofía: gracias por tu apoyo incondicional, por tu cariño y por el amor que me entregas cada día. Has sido un pilar fundamental en mi desarrollo académico, profesional y personal. Estar contigo me da fuerzas para seguir adelante, me hace creer en mí mismo incluso en los momentos difíciles, y me recuerda siempre la importancia de compartir este camino con alguien que te impulsa a ser mejor.

A mis padres, Julio y Margarita: gracias por acompañarme con paciencia y amor en este largo proceso. Sé que no ha sido fácil, pero su apoyo constante ha sido esencial para llegar hasta aquí. Su ejemplo, sus enseñanzas y los valores que me inculcaron han sido la base sobre la que he construido lo que soy y lo que he logrado.

A mi hermana mayor, Evaluna: eres una de las personas más esforzadas y animadas que conozco. Aunque tal vez no lo diga con frecuencia, me inspiras a seguir creciendo, a no rendirme y a dar siempre lo mejor de mí.

A mis hermanas, Almendra y Millaray: agradezco profundamente su compañía, sus ánimos, su amor y su apoyo.

A mi difunta hermana, Violeta: sé que no leerás esto, pero quiero agradecerte por tu amor y por los ánimos que siempre me entregaste. Te pienso y te extraño constantemente. Tu recuerdo me acompaña y sigue siendo una fuente de fuerza y cariño en mi vida.

A mi familia: gracias por su cariño, apoyo y por estar presentes en los momentos importantes. Cada uno, a su manera, ha contribuido a que este camino sea posible. Especialmente a Mauricio, valoro profundamente tu presencia, y el respaldo que siempre nos has brindado.

A mis amigos: gracias por estar ahí, por los buenos momentos, el apoyo cuando hizo falta y por ayudarme a mantener el equilibrio entre el estudio y la vida. Su compañía hizo este proceso mucho más llevadero y entretenido.

RESUMEN

Resumen— El desarrollo de software en contextos educacionales es cada vez más frecuente debido a que permite acercar a los estudiantes a la práctica profesional de la ingeniería de software. Con todo, en el contexto educacional existen algunas restricciones de tiempo y dedicación que típicamente influyen en la priorización de los requisitos, haciendo que tanto estudiantes como profesores pongan énfasis en la satisfacción de necesidades funcionales por sobre aspectos extra-funcionales (i.e., atributos de calidad). La postergación de la atención de requisitos extra-funcionales hace que el producto de software desarrollado en estos contextos difícilmente pueda pasar a producción. En este trabajo de memoria se utiliza la práctica “evaluación de arquitectura de software” para determinar la idoneidad de la arquitectura “AuditForge”, un software de reportería de ciberseguridad, que se crea en un entorno educacional llamado “Taller de Desarrollo de Proyectos de Informática” de la carrera de pregrado de “Ingeniería Civil Informática” de la Universidad Técnica Federico Santa María. El análisis realizado permitió determinar atributos de calidad desatendidos, así como una serie de mejoras potencialmente desarrollables con miras a llevar a “AuditForge” a un estado compatible con las necesidades del nivel producción. Algunas de estas mejoras identificadas fueron priorizadas e implementadas.

Palabras Clave— Análisis de Software, Análisis de Arquitectura, Mejoras Arquitectónicas, Ingeniería de Software, Ciberseguridad

ÍNDICE DE CONTENIDOS

RESUMEN	III
ÍNDICE DE FIGURAS	V
ÍNDICE DE TABLAS	V
1 INTRODUCCIÓN	1
Contexto y motivación	1
Definición del problema	2
Objetivos	3
Estrategia de trabajo	4
Estructura del informe	4
2 MARCO TEÓRICO Y ESTADO DEL ARTE	7
Modelos de calidad de software	7
Arquitectura de software	9
Ciberseguridad	10
3 EVALUACIÓN DE ARQUITECTURA E IMPLEMENTACIÓN DE MEJORAS	13
Arquitectura AuditForge	13
Metodología utilizada para el trabajo	14
Análisis de arquitectura de AuditForge	14
Resultados del análisis de arquitectura	19
Implementación de mejoras priorizadas	22
Resultado de las mejoras	25
4 ANÁLISIS RETROSPECTIVO DEL TRABAJO REALIZADO	27
5 CONCLUSIONES	31
ANEXOS	34
REFERENCIAS BIBLIOGRÁFICAS	39

ÍNDICE DE FIGURAS

2.1	Calidad del producto ISO 25010:2011	8
3.1	Diagrama de secuencia, Caso de Uso "Recomendación de CWE"	18
3.2	Roadmap para la implementación de mejoras	22
5.1	Diagrama de despliegue	34
5.3	SIG inicial: Seguridad	34
5.2	ADR: Despliegue del modelo de IA en contenedor separado	35
5.4	SIG inicial: Fiabilidad	36
5.5	SIG inicial: Portabilidad	36
5.6	Diagrama inicial de despliegue backend	37
5.7	SIG actual: Seguridad	37
5.8	SIG actual: Fiabilidad	38
5.9	Diagrama actual de despliegue backend	38

ÍNDICE DE TABLAS

3.1	Estimación de tiempo de mejoras arquitectónicas y funcionales	20
-----	---	----

Capítulo 1

INTRODUCCIÓN

Contexto y motivación

La arquitectura de software juega un papel vital en la calidad de un sistema de software. Los atributos de calidad de un sistema de software son de alcance a nivel de sistema y, por tanto, la arquitectura del sistema facilita (o inhibe) alcanzar los requisitos de calidad. Algunos de estos atributos son apropiabilidad funcional, performance, compatibilidad, usabilidad, confiabilidad, seguridad, mantenibilidad, y portabilidad, entre otros.

En muchas soluciones de software construidas en contextos educativos (i.e., productos de software que se desarrollan como parte de las asignaciones de tareas en cursos de pregrado y postgrado), los esfuerzos suelen centrarse en cumplir con la funcionalidad principal del sistema, dejando de lado atributos de calidad y sus requisitos (i.e., requisitos extra-funcionales). Este tipo de características, no siempre son visiblemente urgentes en etapas tempranas del desarrollo, pero son fundamentales para la madurez del software.

AuditForge, un sistema de reportería de ciberseguridad, surge en un entorno académico: Taller de Desarrollo de Proyectos de Informática de la carrera de pregrado “Ingeniería Civil Informática” en la Universidad Técnica Federico Santa María en Chile. El desarrollo inicial no contempló una evaluación sistemática de su arquitectura ni un análisis del impacto de esta sobre atributos clave de calidad. Se intuye que, como suele ocurrir en este tipo de proyectos, existen aspectos extra-funcionales que no han sido debidamente atendidos, y cuya mejora contribuiría a llevar el sistema a un nivel más robusto, apto para su uso en contextos productivos.

Este trabajo adopta una técnica de evaluación de la arquitectura de un sistema para revisar el diseño de *AuditForge* con miras a encontrar cuestiones que puedan poner en riesgo el paso del sistema a un contexto productivo. La experiencia realizada en este trabajo muestra que atributos como seguridad, confiabilidad y usabilidad exponen asuntos que pueden ser atendidos con el objetivo de promover la calidad del sistema evaluado.

Esta iniciativa busca no solo fortalecer la calidad del sistema AuditForge, sino también demostrar la relevancia práctica de incorporar evaluaciones arquitectónicas desde etapas tempranas, incluso en contextos educativos, donde su ausencia puede comprometer la evolución futura del software.

En lo sucesivo, esta memoria se estructura de la siguiente manera: el capítulo 2 articula los principales conceptos atinentes a este trabajo así como también un estado del arte respecto de las técnicas de evaluaciones de arquitectura de software; el capítulo 3 presenta los resultados de la ejecución de la evaluación de la arquitectura de *AuditForge*; finalmente, el capítulo 5 cierra este informe con las conclusiones y trabajo futuro.

Definición del problema

En la era digital actual, la ciberseguridad se ha convertido en un pilar esencial para la operación y continuidad de organizaciones de todos los sectores y tamaños. El creciente aumento de amenazas cibernéticas presenta desafíos complejos a nivel global, afectando a infraestructuras y softwares tanto públicos como privados. En este contexto, las auditorías de ciberseguridad son fundamentales para identificar vulnerabilidades, evaluar riesgos, y fortalecer las defensas de los sistemas informáticos. Sin embargo, muchas organizaciones aún carecen de herramientas eficientes y confiables para realizar auditorías exhaustivas, lo que aumenta su exposición a ataques y compromete la seguridad de sus activos.

AuditForge es un software diseñado específicamente para facilitar y optimizar el proceso de auditoría de ciberseguridad. Este software es una solución enfocada al auto-alojamiento del servicio, lo que permite a las organizaciones mantener un control total sobre sus datos y procesos. AuditForge integra herramientas para la generación eficiente de informes, recomendación automática de clasificación de vulnerabilidades y nivel de criticidad utilizando modelos de IA, y colaboración en tiempo real entre auditores.

AuditForge fue inicialmente desarrollado en un contexto educativo, específicamente para su presentación en la Feria de Software (FESW [1]), donde los plazos limitados y los objetivos específicos dificultan una revisión exhaustiva del software. Este tipo de entorno a menudo lleva a que se descuiden aspectos clave como el análisis detallado de software y la creación de una arquitectura robusta alineada con los objetivos del negocio, lo que deja un amplio margen para mejoras. Estas limitaciones, comunes en proyectos académicos, pueden comprometer la capacidad de la herramienta para satisfacer las necesidades del entorno productivo, donde la confiabilidad, seguridad y eficiencia son fundamentales. Por lo tanto, es crucial llevar el software de un nivel educacional a uno apto para contextos productivos, optimizando su arquitectura y funcionalidad para enfrentar desafíos reales del mercado, solucionando un problema complejo de ingeniería.

Dado que el software es de uso libre, las limitaciones actuales pueden generar incertidumbre en su adopción por parte de futuros clientes y usuarios. Además, la falta o deficiencia

de ciertas funcionalidades clave podría afectar directamente a los usuarios actuales. Entre estos se encuentran auditores de seguridad independientes, auditores de seguridad internos dentro de empresas, y empresas auditoras de ciberseguridad. Estos grupos dependen de herramientas confiables y eficientes para realizar sus análisis de vulnerabilidades, por lo que cualquier carencia en el software impacta directamente en su capacidad de cumplir con sus tareas de manera efectiva.

La transformación de AuditForge en un sistema apto para contextos productivos presenta un **problema complejo de ingeniería** debido a la naturaleza multifacética del análisis y las mejoras que se deben implementar. En primer lugar, es esencial realizar un análisis exhaustivo de la arquitectura actual del software utilizando técnicas ampliamente aceptadas y reconocidas, para identificar áreas críticas de mejora que impacten en atributos de calidad del sistema, como la seguridad. Esto implica abordar problemas de diseño que, aunque no fueron prioritarios en su desarrollo inicial, son fundamentales para su adopción en entornos productivos. Además, las mejoras prioritarias requieren una evaluación cuidadosa del impacto, beneficio y costo de implementación, que son factores necesarios a considerar. Todo esto debe lograrse manteniendo altos niveles de usabilidad y mantenibilidad. Este conjunto de factores técnicos, operativos y estratégicos convierte el proceso de mejora en un problema complejo de ingeniería.

Objetivos

El **objetivo general** es llevar a AuditForge [2], un software desarrollado en un contexto educacional, a un nivel superior de madurez, apto para su uso en contextos productivos, mediante la utilización de técnicas sistemáticas de la ingeniería de software.

Los **objetivos específicos** son:

- Identificar áreas de mejora utilizando metodologías y técnicas ampliamente aceptadas en la evaluación de arquitecturas de software.
- Diseñar una solución arquitectónicamente adecuada que responda tanto a los objetivos del negocio como a las necesidades identificadas durante el análisis inicial, priorizando las mejoras en función de su impacto y factibilidad, y estableciendo un *roadmap*¹ para su implementación.
- Implementar las mejoras prioritarias según el *roadmap* definido en el punto anterior y llevar a cabo un análisis posterior para evaluar los resultados obtenidos, verificando el cumplimiento de los objetivos planteados y el impacto de las modificaciones en el software.

¹También llamado hoja de ruta, es una herramienta visual que representa la planificación estratégica de un proyecto.

Estrategia de trabajo

Se propone utilizar una técnica conocida como análisis de arquitectura de software para identificar riesgos y oportunidades de mejora en la arquitectura actual del sistema AuditForge. Esta evaluación se llevará a cabo mediante la aplicación del método SIG (Softgoal Interdependency Graph), en conjunto con el modelo de calidad definido por el estándar ISO/IEC 25010:2011, ampliamente aceptado en la industria del software. Este enfoque permite analizar cómo las decisiones arquitectónicas afectan atributos de calidad como seguridad, mantenibilidad, usabilidad, entre otros.

A partir de este análisis, se identificarán y priorizarán mejoras arquitectónicas en base a su impacto y factibilidad. De estas, se seleccionarán las más relevantes para ser implementadas, con el objetivo de utilizar su desarrollo como caso de estudio, y así, determinar empíricamente el aporte que se ha hecho. Esta estrategia permite no solo mejorar la calidad de AuditForge, sino también generar evidencia sobre la efectividad práctica del enfoque propuesto.

Estructura del informe

Este informe se encuentra organizado en cinco capítulos, los cuales se describen a continuación:

- **Introducción:** Se presenta el contexto general de la problemática de AuditForge planteada, junto con los objetivos y la estrategia de trabajo para la implementación de mejoras.
- **Marco teórico y estado del arte:** Se introducen los fundamentos técnicos necesarios para comprender el desarrollo de la solución y se presenta la arquitectura de AuditForge.
- **Evaluación de arquitectura e implementación de mejoras:** Se describe la metodología empleada, el análisis arquitectónico realizado sobre AuditForge y las mejoras implementadas en el sistema: "Log de acciones", "Detección de desconexión" y "Comunicación segura". La sección finaliza con una evaluación de los resultados obtenidos tras la incorporación de dichas mejoras.
- **Análisis retrospectivo del trabajo realizado:** Se presenta una reflexión crítica sobre el desarrollo del trabajo, abordando las contribuciones realizadas, las herramientas y recursos utilizados, las dificultades enfrentadas, los aprendizajes obtenidos durante el proceso y recomendaciones para futuras iniciativas similares.
- **Conclusiones:** Se discuten los principales hallazgos del trabajo, contrastándolos con la hipótesis inicial. También se abordan los alcances y limitaciones del estudio, y se proponen posibles líneas de trabajo futuro.

Finalmente, se presentan los anexos y referencias bibliográficas utilizadas para el presente trabajo.

Capítulo 2

MARCO TEÓRICO Y ESTADO DEL ARTE

Modelos de calidad de software

Un modelo de calidad de software, es un conjunto de propiedades estructuradas que define, organiza y relaciona atributos de calidad de un sistema. Dichos modelos pueden ser de tipo jerárquicos o relacionales, para efectos de la presente memoria, se profundizará en el primer tipo. Como se define en [3, p. 124], “Un modelo jerárquico define un conjunto de aspectos de calidad de software, luego descompone cada aspecto en varios factores, que a su vez se descomponen en varios atributos, y estos se descomponen aún más en diversas métricas.”.

¿Qué modelos de calidad de software existen?

A lo largo de los años, numerosos modelos de calidad de software han sido desarrollados, cada uno con un enfoque y método de implementación distintos. Los modelos de calidad de software más conocidos y utilizados históricamente son:

- *McCall Model* [4].
- *Boehm Model* [5].
- *FURPS Model* [6].
- *Dromey Model* [7].
- *ISO IEC 9126 Model* [8].
- *ISO IEC 25010 - Product Quality Model* [9].

Hasta el año 1991, cuando se introdujo el modelo de calidad del estándar ISO/IEC 9126 con una estructura jerárquica definida, la mayoría de los modelos de calidad de software presentaban una estructura plana. Una excepción a esto fue el modelo McCall, que ya proponía una estructura jerárquica organizada en tres niveles. En [10] se presenta un análisis comparativo entre los cinco primeros modelos de calidad presentados.

El sexto modelo de calidad presentado, correspondiente al estándar *ISO/IEC 25010 Product Quality*, fue publicado en el año 2011 con el objetivo de reemplazar al estándar ISO/IEC 9126. Este nuevo modelo también adopta una estructura jerárquica, en este caso más definida y adaptada a los sistemas actuales. Además, el estándar ISO/IEC 25010, cuenta actualmente con dos modelos de calidad vigentes, la versión original de 2011 y su actualización, publicada en el año 2023.

Estándar ISO/IEC 25010

Como se mencionó anteriormente, el estándar *ISO/IEC 25010* posee dos versiones, la versión 2011 (*ISO 25010:2011*), que ha sido ampliamente referenciada y adoptada, y la versión 2023 (*ISO 25010:2023* [11]), la cual posee un enfoque más actual, pero no ha sido tan utilizada, por lo reciente de esta.

La versión 2011 del estándar *ISO/IEC 25010* surgió debido a la necesidad de establecer criterios específicos para la evaluación de software, ya que en esos momentos se estaba viviendo una creciente complejidad tecnológica. Esta normativa propuso una estructura basada en dos modelos complementarios: calidad del producto y calidad en uso. La calidad del producto considera ocho características fundamentales o también llamados “atributos de calidad”, estos atributos abarcan diversos enfoques, desde funcionalidad, hasta seguridad y portabilidad. Su implementación representa un desafío, ya que requiere la transformación de principios abstractos en métricas concretas.

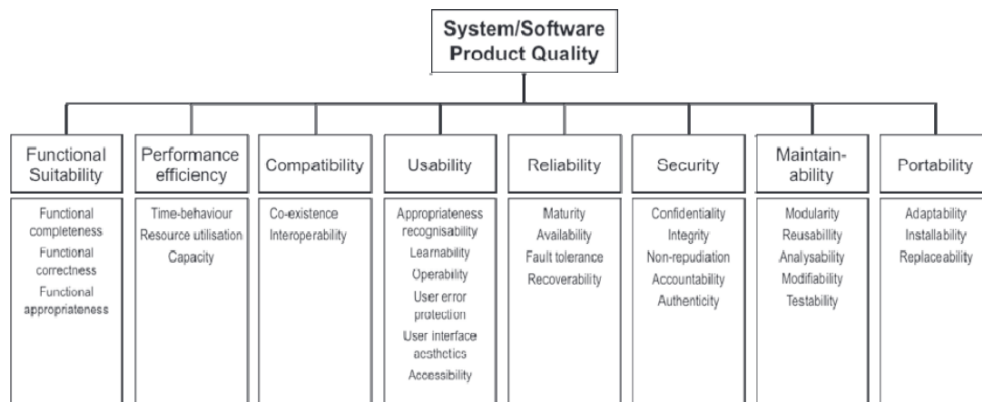


Figura 2.1: Calidad del producto ISO 25010:2011

Fuente: *Estándar ISO/IEC 25010*, [9]

El modelo de calidad del estándar *ISO/IEC 25010* en su versión 2023, incorpora una nueva característica fundamental, la cual es “safety”. Además, mejora la precisión terminológica y descriptiva con respecto a su versión del 2011¹.

Arquitectura de software

La arquitectura de software de un sistema representa las decisiones de diseño relacionadas con la estructura y comportamiento general del sistema. La arquitectura ayuda a los *stakeholders*² a entender y analizar, cómo el sistema logrará cumplir con los requisitos de cualidades esenciales como modificabilidad, disponibilidad, y seguridad [12].

¿Qué es la evaluación de arquitectura?

La evaluación de una arquitectura de software es una de las prácticas de la ingeniería de software que permite determinar la aptitud de esta para el propósito del sistema de software [13, 14, 15]. En la práctica, esto significa determinar si las preocupaciones respecto de la calidad de un sistema de software han sido consideradas en el diseño de la arquitectura [16, 17, 18].

Métodos de evaluación de arquitectura

Existen diversos métodos de evaluación de arquitectura, donde cada método posee un enfoque y postura distinta, junto con la etapa ideal en que se debe encontrar el software para su correcta implementación. Los métodos de evaluación de arquitectura son ampliamente utilizados y referenciados en la industria del análisis de software. En [19] se describen, clasifican y comparan algunos de los métodos existentes, junto a sus enfoques. A continuación se presentan dos de los más conocidos:

- **Análisis de escenarios:** El análisis de arquitectura de software basado en escenarios evalúa cómo la arquitectura se comporta ante diversos atributos de calidad, mediante la definición de escenarios. Para el planteamiento de dichos escenarios, se utilizan los puntos de vista tanto de los usuarios como de los desarrolladores, con el fin de asegurar que el software cumpla con los requisitos de calidad necesarios ante diferentes contextos. El uso de escenarios es de los enfoques más utilizados por los métodos de evaluación, entre los métodos más conocidos están **SAAM** [20] (Software Architecture Analysis Method), **ATAM** [21] (Architecture Tradeoff Analysis Method), **CBAM** [22]

¹Más detalles de los cambios entre los estándares mencionados se pueden encontrar en el [sitio oficial](#).

²Personas, grupos o entidades que tienen un interés en una organización, proyecto o empresa.

(Cost Benefit Analysis Method), **ARID** [23] (Active Reviews for Intermediate Designs), **ALMA** [24] (Architecture-Level Modifiability Analysis), entre otros.

- **Análisis de compromisos (*trade-off*)**³: Los métodos de análisis centrado en compromisos, también conocidos como enfoques de trade-off, son enfoques que permiten evaluar y comparar los compromisos entre diferentes atributos de calidad del sistema. Con compromisos, se hace referencia a las consecuencias indirectas hacia un atributo de calidad, al inclinarse por otro. Algunos de los métodos más conocidos que utilizan este enfoque son: **ATAM**, **CBAM** y **QAW**[25] (Quality Attribute Workshop).

Los métodos de análisis de software no están limitados a tener un único enfoque, pueden poseer más de uno, como es el caso de **ATAM**, que se basa en el análisis de compromisos, pero para ello utiliza escenarios. De manera adicional, se presenta otro método de análisis de arquitectura, el cual tiene un enfoque más reciente:

- **Centrado en decisiones**: Los métodos de análisis centrado en decisiones, son enfoques que evalúan la calidad arquitectónica de un sistema a partir del análisis de las decisiones arquitectónicas tomadas durante su diseño. Estos métodos no solo utilizan las decisiones documentadas o discutidas, sino que además considera factores de negocio y organizacionales. Algunos ejemplos de este tipo de método es **DCAR** [26] (Decision-Centric Architecture Review) y **CBAM**.

Ciberseguridad

Según [27], "la ciberseguridad es un conjunto de estándares y prácticas que las organizaciones utilizan para proteger sus aplicaciones, datos, programas, redes y sistemas de ciberataques y acceso no autorizado". En la actualidad, existen diversos tipos de ciberseguridad y múltiples enfoques para prevenir ciberataques. Entre las prácticas más comunes se encuentran los ejercicios de *Ethical Hacking* o *pentesting* (pruebas de penetración).

¿Qué es *pentesting*?

Como define el **ISTQB**⁴ el *pentesting* o pruebas de penetración "Un tipo de prueba dinámica de seguridad en aplicaciones que evalúa debilidades y vulnerabilidades sin causar daño, mediante la simulación del comportamiento de un atacante." [28]. Las pruebas de penetración son realizadas por profesionales de ciberseguridad, donde una etapa crucial del proceso es la reportería de los hallazgos, como detalla **PTES**⁵ en [29].

³Se refiere a la necesidad de sacrificar o reducir ciertos atributos de calidad del sistema para mejorar otros.

⁴International Software Testing Qualifications Board

⁵Penetration Testing Execution Standard

Requerimientos para los informes de ciberseguridad

Durante el diseño del software, con el fin de plantear correctamente las necesidades que el software debía cubrir y darle una correcta dirección al desarrollo, el equipo de AuditForge se reunió con un grupo de expertos en el área de la ciberseguridad, compuesto principalmente por auditores de ciberseguridad e ingenieros de ciberseguridad. En dicha reunión se plantearon las necesidades iniciales identificadas por el equipo, donde luego los auditores tuvieron un espacio para plantear áreas de interés para los auditores y de posibles mejoras para el diseño del software.

De dicho espacio, desde una perspectiva de arquitectura de software, se pueden destilar los siguientes requerimientos mencionados por los expertos:

- **Seguridad:** Debido a la naturaleza sensible de la información que maneja y almacena, el sistema debe garantizar principios fundamentales de seguridad, como confidencialidad, integridad y disponibilidad.
- **Confidencialidad de la información:** El sistema debe proteger la confidencialidad, evitando la divulgación no autorizada de información a terceros. Esto aplica también para los modelos de IA que se fueran a utilizar en el software.
- **Trabajo colaborativo:** El sistema debe permitir a múltiples auditores trabajar de manera simultánea.
- **Portabilidad:** El software debe ser fácilmente desplegable y configurable, facilitando la migración entre diferentes entornos o plataformas.
- **Ayuda en tareas repetitivas:** El sistema debe permitir la integración de modelos de inteligencia artificial para automatizar o apoyar en tareas repetitivas y así, reducir la carga manual.
- **Segmentación de permisos:** El sistema debe implementar una segmentación clara de permisos y roles, que permita una administración adecuada de accesos.
- **Experiencia usuaria:** El software debe ser intuitivo y facilitar la interacción con sus funciones, especialmente para usuarios con conocimientos técnicos, asegurando así una experiencia de uso satisfactoria.

Los requerimientos generados a partir de esta instancia con profesionales del rubro aportaron significativamente al enfoque adoptado en las decisiones de diseño del proyecto y a los atributos de calidad que fueron priorizados.

Capítulo 3

EVALUACIÓN DE ARQUITECTURA E IMPLEMENTACIÓN DE MEJORAS

Arquitectura AuditForge

AuditForge cuenta con una arquitectura compuesta por cuatro componentes principales: frontend, backend, base de datos y modelo de inteligencia artificial. Cada componente está containerizado de forma independiente, incluyendo sus respectivas dependencias¹ y requerimientos, lo que favorece la portabilidad del sistema.

La comunicación entre los componentes se realiza a través de solicitudes HTTP(S)². El despliegue del sistema se gestiona mediante *Docker Compose*³, donde la configuración de Docker genera dos redes diferenciadas, con el objetivo de aislar los servicios entre sí y restringir el acceso únicamente a los componentes que deben interactuar directamente, fortaleciendo así la seguridad y el control del flujo de datos.

El sistema fue diseñado con un enfoque que permita su funcionamiento completamente *offline*⁴ y autoalojado, con el fin de garantizar la privacidad y confidencialidad de la información procesada y almacenada. Con el mismo objetivo, el modelo de inteligencia artificial utilizado para generar recomendaciones también se almacena y ejecuta localmente, evitando dependencias externas o el intercambio de datos con terceros.

Como se observa en el **anexo 5.1**, el backend constituye la base del sistema. Este se encarga de realizar validaciones, ejecutar acciones, consultar el modelo de inteligencia artificial, así como controlar y acceder a la base de datos. La arquitectura del backend puede apreciarse

¹Componente, biblioteca o un paquete, que una aplicación necesita para funcionar correctamente.

²Protocolo de Transferencia de Hipertexto, con una capa de seguridad.

³Herramienta que simplifica la definición y gestión de aplicaciones Docker con múltiples contenedores.

⁴Sin conexión a una red o internet.

con mayor detalle en el diagrama de despliegue⁵ incluido en el **anexo 5.6**.

Metodología utilizada para el trabajo

La metodología contempla la evaluación de la arquitectura de software de *AuditForge* para la identificación de riesgos en el diseño arquitectónico. Para la evaluación de la arquitectura se utiliza una extensión del Kernel de SEMAT [30] que provee cinco elementos esenciales que deben progresar en una revisión del diseño arquitectónico (Decisiones Arquitectónicas, Atributos de Calidad, Objetivos del Negocio, Descripción de la Arquitectura, y Adopción de la Evaluación).

Uno de los aspectos clave en la evaluación de la arquitectura de un sistema de software es la identificación y adopción de un modelo de calidad que permita representar adecuadamente las características de calidad que se esperan en el producto de software sujeto de evaluación. Para este trabajo, se ha adoptado el modelo de calidad propuesto en el estándar ISO/IEC 25010:2011 [9].

El enfoque utilizado para la evaluación de la arquitectura de *AuditForge* es **centrado en decisiones**, el cual nos permite analizar las decisiones tomadas durante el diseño del software junto con sus respectivas justificaciones.

Esta combinación metodológica permite realizar un análisis integral de la arquitectura actual, identificando tanto fortalezas como debilidades, así como alternativas arquitectónicas que pudieron no haber sido consideradas en su momento. De este modo, se establece un marco sistemático para formular propuestas de mejoras alineadas con los atributos de calidad definidos.

Finalmente, debido a la naturaleza del presente trabajo, no se contempla la validación de una hipótesis o solución experimental, sino que se opta por una **retrospectiva del trabajo ejecutado**. Esta retrospectiva tiene por objetivo identificar los aportes concretos realizados, los elementos metodológicos y técnicos utilizados, así como los aspectos que deben ser considerados por quienes deseen replicar un análisis arquitectónico similar.

Análisis de arquitectura de AuditForge

Dado que el software fue desarrollado en un contexto educativo, se reutilizó documentación generada durante las etapas de diseño y desarrollo del *software*, la cual también fue requerida como parte de los entregables para el curso de FESW. Además, se elaboró nueva documentación orientada al análisis arquitectónico, disponible públicamente en GitHub⁶.

⁵Tipo de diagrama UML que se utiliza para modelar la disposición física de los artefactos software en nodos.

⁶Acceso al repositorio mediante el siguiente [enlace](#).

Para la recolección de decisiones arquitectónicas tomadas, consideradas y descartadas, se revisaron conversaciones en el canal oficial de comunicación del equipo, así como los *commits*⁷ y *pull requests* (PR)⁸ del repositorio del proyecto.

Antes de comenzar con el análisis de arquitectura, se debió observar y analizar el estado actual del software a nivel de código. Esto permitió contrastar la documentación existente y generada con el comportamiento real del sistema, identificando posibles inconsistencias, omisiones o desactualizaciones, este proceso fue clave para establecer una línea base sobre la cual realizar el análisis arquitectónico posterior.

El análisis de la arquitectura se organizó siguiendo la progresión establecida por un conjunto de “Alphas”⁹ definidos en la representación basada en el Kernel de SEMAT para las evaluaciones de arquitectura de software (propuesta disponible en [30]), lo que permite estructurar el proceso en actividades coherentes, además de poner énfasis en el análisis del alineamiento de las decisiones técnicas con los objetivos del sistema y del negocio.

La representación basada en el Kernel de SEMAT utilizada en esta memoria define cinco “Alphas” [30]: Adopción de la Evaluación de Arquitectura, Atributos de Calidad, Decisiones de Arquitectura, Objetivos del Negocio, y Descripción de la Arquitectura de Software. Cada “Alpha” fue trabajado para evaluar la arquitectura de “AuditForge”. Esta representación obliga la adopción de un método o enfoque. En esta memoria, el enfoque de evaluación adoptado es marcadamente basado en decisiones de arquitectura.

A partir del uso de esta representación, se desarrollaron las siguientes actividades:

- **Identificación y documentación de decisiones arquitectónicas:** Se revisaron decisiones tomadas por el equipo, incluyendo las justificaciones técnicas o de negocio detrás de ellas.
- **Formalización mediante *Architecture Decision Records* (ADRs)¹⁰:** Estas decisiones se estructuraron utilizando el formato ADR, lo cual facilita su trazabilidad y mantiene la lógica detrás de las elecciones arquitectónicas.
- **Modelado de funcionalidades principales:** Se elaboraron casos de uso y diagramas de secuencia en UML, enfocados en historias de usuario clasificadas como “esenciales” e “importantes”.
- **Documentación de diagramas:** En base a las necesidades de documentación observadas, se desarrollaron los siguientes diagramas UML:

⁷Registro de cambios realizados en el código fuente en un sistema de control de versiones, como Git.

⁸Pull Requests, también conocidos como solicitudes de cambios. Más información en el siguiente [enlace](#).

⁹Alpha es el acrónimo de “Abstract-Level Progress Health Attribute” y se define como un elemento esencial en alguna práctica de la ingeniería de software. Cada “Alpha” define una progresión que va desde niveles discernibles de menor avance hasta niveles en que se considera que el “Alpha” ha sido atendido completamente en la ejecución de la práctica.

¹⁰Documento que registra una decisión arquitectónica importante tomada durante el desarrollo de software, junto con su contexto, justificación y consecuencias.

- **Despliegue:** Representa la arquitectura actual del software, incluyendo a los autores involucrados, sus ambientes y canales de comunicación.
- **Comunicación¹¹:** Refleja el flujo normal que sigue la comunicación entre componentes, aislando la comunicación con el modelo de IA y con la base de datos.
- **Definición de *Business Drivers*¹²:** En conjunto con el equipo de *DevForge*, se definieron los objetivos de negocio, considerando misión, metas estratégicas y contexto de uso del sistema.
- **Selección y aplicación de un modelo de calidad:** Se investigaron modelos de calidad existentes y se eligió la norma ISO/IEC 25010:2011 por su amplitud y enfoque práctico. Esta norma permitió categorizar y evaluar los atributos de calidad relevantes para el sistema.
- **Priorización de atributos de calidad:** Se listaron los atributos propuestos por la norma y se priorizaron según el contexto y necesidades del sistema. Aquellos no prioritarios fueron justificados explícitamente.
- **Formulación y análisis de *soft goals*¹³:** A partir de los antecedentes recopilados (decisiones, objetivos del negocio, atributos de calidad), se formularon *soft goals*, los cuales fueron representados mediante diagramas SIG (Softgoal Interdependency Graphs¹⁴). Se analizaron los compromisos (*trade-offs*) entre alternativas arquitectónicas para evaluar el impacto relativo de las decisiones.

Architecture Decision Records (ADRs)

Antes de realizar un análisis formal de la arquitectura o adoptar un método específico para dicho fin, se desarrollaron Registros de Decisiones Arquitectónicas con el objetivo de documentar las decisiones clave tomadas durante el diseño inicial del sistema. Esta práctica permitió capturar el razonamiento detrás de cada elección técnica, así como sus posibles implicancias, facilitando posteriormente el análisis estructurado de la arquitectura.

Los ADRs no solo sirvieron como un mecanismo de trazabilidad, sino que también fueron un insumo fundamental para identificar decisiones relevantes a representar en el análisis SIGs.

A continuación, se listan algunos de los ADRs más relevantes registrados:

1. Como se realizan las consultas HTTPS en el sistema

Se decidió utilizar *fetch*¹⁵ como método para realizar las consultas en el sistema utilizando el protocolo HTTP, evitando así el uso de clientes HTTP adicionales.

¹¹Herramienta visual que muestra cómo interactúan los objetos y actores en un sistema mediante el intercambio de mensajes, destacando la secuencia y el orden de estas interacciones.

¹²Factor, recurso o actividad que impulsa el éxito operativo y financiero de una empresa.

¹³Método para identificar y evaluar propiedades de calidad no funcionales pero relevantes para un sistema.

¹⁴Técnica para modelar y analizar las decisiones de arquitectura de software.

¹⁵Función que permite realizar peticiones HTTP a servidores web para obtener recursos de forma asíncrona.

2. **Validación directa en endpoints¹⁶**

Se decidió realizar las validaciones directamente en los endpoints, para tener un mayor control y granularidad sobre los permisos en el sistema.

3. **Despliegue del modelo de IA en contenedor separado**

Se definió que el modelo IA sería desplegado en un contenedor separado para evitar sobrecargar el contenedor del backend, donde además, se puede tener una mejor segmentación de acceso y permisos sobre el modelo IA.

4. **Tecnología Backend Modelo IA**

El equipo debe elegir una tecnología para el backend del modelo de IA que sea sencilla de mantener, que esté alineada con las habilidades del equipo y que utilice sin problemas el modelo.

Cada ADR fue documentado con un formato estándar¹⁷ que incluyó el contexto de la decisión, las alternativas consideradas, la decisión tomada y sus consecuencias. Esta documentación anticipada fue crucial para orientar las discusiones arquitectónicas posteriores y aportar claridad en la evaluación de trade-offs entre distintas alternativas de mejora. En el **anexo 5.2** se presenta un ejemplo de ADR titulado “Despliegue del modelo de IA en contenedor separado”.

Casos de uso

Como se mencionó anteriormente, a partir de las historias de usuario principales —aquellas clasificadas como “importantes” y “esenciales”— se derivaron los casos de uso que sirvieron para estructurar el enfoque del análisis arquitectónico. Estos casos de uso permitieron identificar decisiones relevantes tomadas durante el diseño, así como formalizar los actores, precondiciones, flujos críticos y alternativos del sistema. A continuación, se presentan los casos de uso desarrollados:

1. **Registro de clientes:** Permite a un auditor de ciberseguridad registrar un nuevo cliente en el sistema.
2. **Reutilización de servicios:** Permite a un auditor de ciberseguridad asociar una auditoría a un cliente previamente registrado.
3. **Inicio de Sesión¹⁸:** Permite a un auditor de ciberseguridad autenticarse en la aplicación bajo una segmentación de permisos robusta.

¹⁶Un endpoint de API es una URL que actúa como punto de contacto para acceder a recursos o servicios ofrecidos.

¹⁷Más información sobre el formato seguido en el siguiente [repositorio](#).

¹⁸En ingeniería de software, “iniciar sesión” no se considera un caso de uso, sino una precondición para muchos otros casos. Sin embargo, en “AuditForge” el “inicio de sesión” realiza una segmentación de permisos (basado en roles) y por tanto se ha mencionado como caso de uso.

4. **Generación de visualizaciones de datos en *dashboard*:** Permite a un auditor generar visualizaciones gráficas de datos en el *dashboard* de una auditoría.
5. **Generación de patrones de vulnerabilidades del cliente:** Permite visualizar estadísticas de vulnerabilidades de un cliente en un dashboard.
6. **Edición de Hallazgos:** Permite a un auditor editar los hallazgos detectados y previamente añadidos en una auditoría.
7. **Recomendación de CWE¹⁹:** Permite a un auditor de ciberseguridad recibir sugerencias automáticas de CWE relevantes, basadas en la descripción de un hallazgo.

Posteriormente, se desarrollaron diagramas de secuencia en UML para cada uno estos casos de uso. A continuación se muestra un ejemplo correspondiente al caso de uso "Recomendación de CWE":

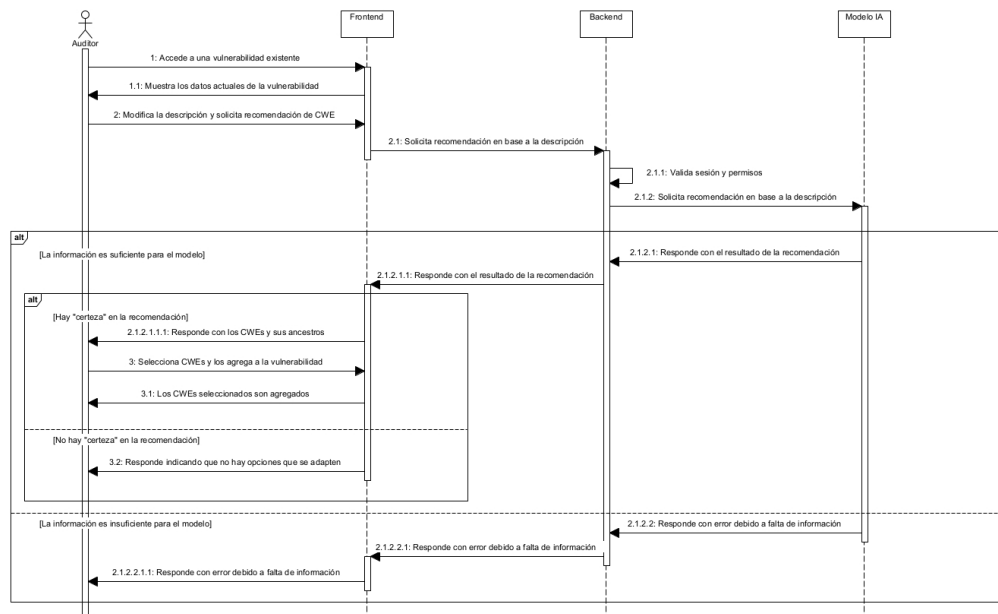


Figura 3.1: Diagrama de secuencia, Caso de Uso "Recomendación de CWE"

Fuente: Elaboración propia

Softgoal Interdependency Graphs

La parte fuerte y culminación del análisis de arquitectura se encuentra en el análisis mediante SIGs. Para plantear este análisis se consideró el orden de prioridad de los atributos de

¹⁹Common Weakness Enumeration, más información en el siguiente [enlace](#)

calidad, aquellos atributos que se identificaron como los que podían tener un mayor impacto sobre la arquitectura, así como los atributos directamente relacionados con las decisiones arquitectónicas tomadas en etapas anteriores del proyecto.

A partir de dichas consideraciones, se elaboraron los SIGs correspondientes a los atributos de calidad “Seguridad”, “Fiabilidad” y “Portabilidad”. En cada uno de estos casos, se identificaron las principales operacionalizaciones²⁰, y se analizó cómo estas se vinculan con las decisiones arquitectónicas tomadas por el equipo. Finalmente, se evaluó cómo dichas decisiones impactan en otros atributos de calidad, considerando los *trade-offs* inherentes entre ellos.

Cabe aclarar que, como parte del análisis mediante SIGs, se elaboraron los diagramas correspondientes, los cuales pueden consultarse en los anexos (Ver figuras 5.3, 5.4 y 5.5). Además, se incluyen detalles adicionales sobre las ventajas y desventajas identificadas en relación con los atributos de calidad en el repositorio de GitHub²¹.

Resultados del análisis de arquitectura

Como resultado del análisis arquitectónico realizado, se concluye que, en general, el equipo tomó decisiones adecuadas durante el desarrollo del sistema. Sin embargo, se identificaron algunas omisiones o aspectos que no fueron completamente considerados y que podrían mejorar la calidad global del software. A partir de este análisis, se levantaron los siguientes requerimientos:

1. **Log de acciones:** Implementar un mecanismo de registro de eventos que permita dejar constancia de las acciones realizadas por los usuarios dentro del sistema, con el fin de garantizar trazabilidad y servir como soporte para el principio de no repudio.
2. **Comunicación segura:** Asegurar que toda la comunicación dentro de la aplicación (cliente-servidor y servidor-servidor) se realice de forma segura. En caso de identificarse vulnerabilidades en este aspecto, deben ser corregidas oportunamente.
3. **Healthcheck²² de servicios:** Evaluar la viabilidad de incorporar un mecanismo de *healthcheck* al servicio, útil para detectar situaciones en las que el proceso se mantiene activo, pero el servicio no está operativo.
4. **Importar/exportar datos:** Implementar funcionalidades presentes en el sistema predecesor (Pwndoc²³) que permiten la importación y exportación de datos. Para ello, se

²⁰Las operacionalizaciones corresponden a la forma en que un atributo de calidad se traduce en acciones concretas o decisiones arquitectónicas.

²¹Acceso al repositorio mediante el siguiente [enlace](#).

²²Mecanismo utilizado para monitorear la salud y el estado de un sistema o componente, más información en el siguiente [enlace](#).

²³Software en el que se basó AuditForge, más información en el siguiente [enlace](#).

debe analizar la forma más adecuada de integrar estas capacidades en la arquitectura actual.

Adicionalmente, se identificó un requerimiento que posee implicancias tanto a nivel arquitectónico como funcional, particularmente en lo relativo a la experiencia de usuario:

5. **Detección de desconexión:** Incorporar una funcionalidad que permita al cliente detectar la pérdida de conexión con el backend. Esta característica, presente en la aplicación predecesora (Pwndoc), podría implementarse mediante el uso de WebSockets u otra tecnología adecuada.

Los requerimientos mencionados anteriormente representan los principales hallazgos del análisis de arquitectura; sin embargo, las posibles mejoras no se limitan a estos requerimientos.

¿Qué se implementó?

Tal como se explicó previamente, el presente trabajo abordó la implementación de mejoras prioritarias en base a su impacto y factibilidad. Para evaluar qué mejoras serían implementadas, fue necesario estimar el esfuerzo o tiempo requerido para completar satisfactoriamente dichas tareas (Ver tabla 3.1).

Mejora	Tiempo estimado
Log de acciones	5 semanas
Comunicación segura	2 semanas
Detección de desconexión	3 semanas
Healthcheck de servicios	4 semanas
Importar/exportar datos	5 semanas

Tabla 3.1: Estimación de tiempo de mejoras arquitectónicas y funcionales

Para realizar el cálculo del tiempo estimado, se consideraron las siguientes etapas en el proceso de implementación de mejoras:

- Revisión detallada del código relacionado a la mejora.
- Planificación de la solución.
- Implementación.

- Validación de la solución.
- Revisión de *PRs* y resolución de posibles conflictos de código.
- Posible modificación de documentación actual del software.

A partir de esta estimación y del tiempo disponible para esta etapa del trabajo, se optó por implementar las mejoras esenciales y de mayor impacto, las cuales son "**Log de acciones**", "**Detección de desconexión**" y "**Comunicación segura**". Las otras mejoras que no fueron implementadas se agregaron como trabajo futuro y registradas como *issues*²⁴ en el repositorio principal del proyecto.

La incorporación de estas mejoras y requerimientos contribuiría a que AuditForge avance hacia un mayor nivel de madurez. Dichas mejoras no solo responden a los objetivos actuales del sistema y de sus usuarios, sino que también están alineadas con las necesidades detectadas en el análisis preliminar. En este sentido, la solución propuesta representa una opción arquitectónicamente adecuada para satisfacer dichas necesidades.

Roadmap para la implementación de mejoras

Una vez definidas las mejoras a implementar, se procedió con la creación de la hoja de ruta. Para su elaboración, se identificaron las actividades clave, las cuales fueron organizadas en etapas:

1. Inicio – Estado actual
2. Revisión de código: Evaluar el estado actual de todos los componentes involucrados en las mejoras.
3. Planteamiento de los pasos específicos a realizar, basados en la revisión anterior.
4. Desarrollo, implementación y documentación de mejoras (incluye sus respectivas implicancias técnicas).
5. Análisis posterior de la solución, para confirmar el cumplimiento de los objetivos propuestos.
6. Estado final del software

Se muestra una representación visual del *roadmap* en la **figura 3.2**.

²⁴Es una manera estructurada de realizar seguimiento de problemas, nuevas funcionalidades o mejoras, más información en el siguiente [enlace](#).

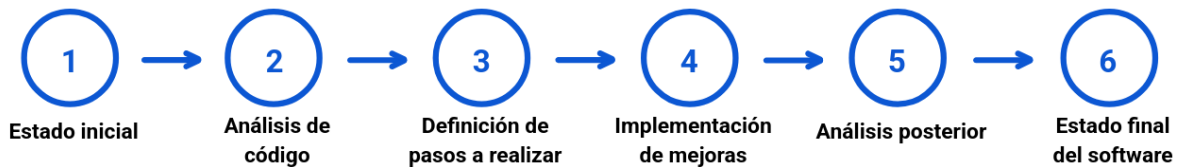


Figura 3.2: Roadmap para la implementación de mejoras

Fuente: Elaboración propia

Implementación de mejoras priorizadas

Tal como se indicó previamente, se implementaron tres mejoras consideradas prioritarias, evaluadas en función de su relevancia y la viabilidad de implementación dentro del tiempo disponible.

Mejora 1: Comunicación segura

En los desarrollos de software realizados en contextos académicos, donde se tienen plazos fijos para entregar avances o *features*²⁵, muchas veces se "aceptan" ciertos riesgos o soluciones no ideales para poder finalizar la entrega. Este es el caso de AuditForge con la seguridad en la comunicación, ya que en cuatro rutas se utilizó un *workaround*²⁶ para permitir la comunicación del backend con la API del modelo de IA.

Estado previo Dicha solución alternativa consistía en deshabilitar de manera temporal o permanente la verificación de certificados SSL/TLS en las conexiones HTTPS, pudiendo derivar en ataques de tipo MitM (Man in the Middle^{27,28}) y no es recomendada en entornos de producción.

Exceptuando este *workaround*, no se presentan problemas adicionales en la comunicación cliente-servidor o servidor-servidor (API Backend - API Modelo IA), ya que se utilizan protocolos TLS v1.2 y v1.3, que son los considerados seguros en la actualidad. Adicionalmente, se recomienda enfáticamente al usuario no utilizar los certificados por defecto en el proyecto.

Este problema surgió debido a las limitaciones de la función utilizada para hacer las solicitudes al backend, se trataba del método *fetch* nativo de *Node.js*²⁹.

²⁵Funcionalidad o característica de un software que cumple con un propósito o necesidad del usuario.

²⁶Solución alternativa utilizada cuando la solución estándar no funciona.

²⁷Man in the Middle, más información sobre la vulnerabilidad en el siguiente blog de [Fortinet](#).

²⁸Se muestra una vulnerabilidad similar reportada por GitLab en el siguiente [enlace](#).

²⁹Node JS es un entorno de ejecución de JavaScript, más información en el [enlace](#).

Mejoras implementadas Los cambios implementados se pueden ver detalladamente en el PR⁸⁰ de GitHub. Estos cambios consistieron en:

- Cambio de función utilizada para realizar las solicitudes: Se cambió *fetch* por el método *requests* de la dependencia *https*, el cual permite manejar certificados autofirmados sin deshabilitar la validación en todo el servidor. Este cambio implicó una refactorización⁸¹ en toda la lógica de solicitudes hacia la API del modelo IA.
- Nuevo volumen: Se agregó un nuevo volumen al backend, el cual permite que el backend acceda al certificado de la API del modelo IA, con una limitación *read-only* para que no se pueda modificar desde el punto de montaje.
- Certificados: Para que la nueva solución funcione sin problemas, se debió cambiar los certificados por defecto, para no generar problemas de seguridad y/o funcionamiento en los usuarios, donde los nuevos certificados requieren que el SAN (Subject Alternative Name) coincida con el nombre del contenedor de la API del modelo IA.
- Documentación: Se actualizó la documentación sobre la instalación del software, especificando los requisitos del certificado a utilizar y un comando de ejemplo para su utilización.

Mejora 2: Log de acciones

Se trata una funcionalidad necesaria para poder tener una trazabilidad de las acciones de los usuarios en la aplicación. Además permite dar cumplimiento al principio de no repudio. Estas propiedades se consideran necesarias en un software de ciberseguridad.

Como se define en el estándar *ISO/IEC 25010:2011*, el no repudio es "grado en el que se puede demostrar que las acciones o eventos han tenido lugar, de modo que no puedan ser repudiados posteriormente"^[9]. El principio de no repudio es algo crucial en los sistemas, especialmente si manejan información sensible y confidencial.

Estado previo Antes de esta mejora, AuditForge no contaba con una funcionalidad que permitiera tener un registro de acciones de los usuarios, lo que impedía garantizar la trazabilidad y el cumplimiento del principio mencionado.

Mejoras implementadas Los cambios implementados se pueden ver detalladamente en el PR⁸² de GitHub. Estos cambios consistieron en:

⁸⁰ Acceso al PR en el siguiente [enlace](#).

⁸¹ Proceso de reescritura o reestructuración de código, sin cambiar el funcionamiento.

⁸² Acceso al PR en el siguiente [enlace](#).

- Almacenado de acciones: Se almacenan las llamadas a endpoints en la base de datos, a través de un *middleware*³³ implementado en el backend.
 - Se almacena información relevante, incluyendo: nombre de usuario, identificador del usuario, rol, *endpoint*, método *HTTP* utilizado, código de respuesta obtenido, *timestamp*³⁴, cuerpo de la solicitud e información adicional.
 - Se excluye información del cuerpo que posea información sensible, como que contenga las palabras "password", "token", "secret", "key" y "totp".
 - Se genera una firma digital de los registros utilizando *SHA-256*³⁵ para asegurar la integridad.
- Consulta de registros: Se habilita la consulta de dichos registros mediante dos *endpoints*³⁶, los cuales fueron agregados a la API del backend.
 - *Endpoints*: El primer *endpoint* habilitado obtiene los últimos cien *logs* de la aplicación, mientras que el segundo agrega flexibilidad en la consulta, permitiendo filtrar, limitar y ordenar el resultado.
 - Datos válidos: La consulta de registros mediante los endpoints solo devuelve *logs* que no han sido modificados, esto mediante un método que valida la integridad de ellos.
 - Permisos: Se agregó un nuevo permiso en la aplicación, cuyo propósito es permitir al usuario obtener los registros, ya sea con o sin filtrado. Por defecto, este permiso solo está asignado al usuario administrador.
 - Documentación: Se agregó a la documentación de AuditForge como consultar dichos *logs*, con variantes como son el filtrado y ordenamiento. Además, se documentó también el procedimiento para otorgar permisos a otros roles para hacer uso de la funcionalidad.

Puede observarse que las mejoras implementadas en este apartado se enfocaron únicamente en la parte funcional del sistema, sin abarcar el frontend. Esto se debe a que el desarrollo de la interfaz requeriría un tiempo de implementación considerable, excediendo los plazos establecidos para el curso. Esta omisión no afecta la funcionalidad del sistema, ya que todas las operaciones son completamente accesibles a través de la API. Por lo tanto, el desarrollo del frontend queda planteado como trabajo futuro y ha sido registrado como *issue* en el repositorio principal.

³³En este contexto, *middleware* es una función que se puede ejecutar antes y/o después del manejo de una ruta.

³⁴Una marca temporal o *timestamp*, es una secuencia de caracteres que denotan la hora y fecha en la que ocurrió un evento.

³⁵SHA-256 es parte del actual estándar según NIST. Más detalles en el siguiente [enlace](#).

³⁶Un endpoint es el punto de contacto entre una aplicación cliente y el servidor de la API, más información en el siguiente blog de [IBM](#).

Mejora 3: Detección de desconexión

En sistemas distribuidos compuestos por múltiples componentes, como es el caso de AuditForge (que posee cuatro), es posible que uno esté completamente operativo mientras otro enfrente fallos o se encuentre fuera de servicio. Por ejemplo, el frontend puede estar funcional mientras el backend presenta problemas. En dichos casos, si el sistema no informa al usuario sobre estos problemas de conectividad u operatividad, se vería afectada tanto la experiencia de usuario como la percepción de madurez del software.

Estado previo Actualmente, si el frontend se encuentra operativo, mientras que el backend no, entonces al intentar ejecutar acciones el usuario únicamente recibiría mensajes de error, sin ser informado sobre la situación real.

Mejoras implementadas Los cambios implementados y detalles técnicos se pueden ver detalladamente en el PR³⁷ de GitHub. Estos cambios consistieron en:

- **Dependencia:** Se agregó una nueva dependencia al frontend para habilitar el uso de *websockets*³⁸.
- **Implementación:** Se desarrolló e implementó una solución que contempla el uso de *websockets*, que establece una conexión persistente desde el cliente hacia el backend. Esta conexión detecta cuando hay una interrupción del servicio y, si la desconexión no es provocada por el usuario (por ejemplo, al refrescar la página), entonces se muestra un *overlay*³⁹ informativo al usuario.

Resultado de las mejoras

Las mejoras implementadas tuvieron un impacto claro en el sistema, fortaleciendo aspectos como la experiencia de usuario, la seguridad y el no repudio. Aunque algunas de estas decisiones pueden parecer centradas en la interacción o usabilidad, se consideran mejoras arquitectónicas, ya que afectan directamente a la estructura fundamental y al comportamiento interno del sistema.

Como consecuencia de estas modificaciones, los diagramas de arquitectura elaborados inicialmente también debieron ser actualizados. En el **anexo 5.9**, por ejemplo, se puede observar la incorporación de un componente *middleware* encargado de gestionar la trazabilidad, con una conexión directa a la base de datos para almacenar dicha información.

³⁷ Acceso al PR en el siguiente [enlace](#).

³⁸ Protocolo de comunicación que permite una conexión bidireccional y persistente entre cliente y servidor.

³⁹ Técnica que superpone información o elementos visuales sobre una interfaz sin reemplazar el contenido subyacente.

Del mismo modo, los Softgoal Interdependency Graphs (SIGs) también se vieron modificados, reflejando el impacto de las nuevas decisiones sobre atributos de calidad clave. Esto puede apreciarse en los **anexos 5.7 y 5.8**.

Capítulo 4

ANÁLISIS RETROSPECTIVO DEL TRABAJO REALIZADO

El desarrollo de este trabajo no se estructuró a partir de una hipótesis experimental, sino mediante una estrategia práctica orientada al análisis arquitectónico de AuditForge y la implementación de mejoras técnicas orientadas a atributos de calidad específicos.

Como resultado, se logró realizar una contribución concreta en términos de documentación, experiencia de usuario, mejoras en la seguridad y análisis de calidad basado en el Modelo de Calidad de Producto del estándar *ISO/IEC 25010:2011*.

Una de las principales contribuciones del trabajo fue visibilizar aspectos no funcionales del sistema que hasta el momento no habían sido formalmente evaluados, tales como la portabilidad y seguridad, mantenibilidad. Donde dicha evaluación es comúnmente dejada de lado o desatendida en los proyectos realizados bajo el alero de un contexto educacional. A partir de este análisis, se implementaron mejoras que reforzaron el sistema sin alterar su funcionalidad principal, con una clara orientación a su madurez futura.

Durante el desarrollo, fue necesario utilizar diversas herramientas de apoyo, tales como diagramas de arquitectura (despliegue, componentes y flujos), casos de uso, y registros de decisiones arquitectónicas (ADRs) para documentar alternativas y *trade-offs*. También se emplearon SIGs (Softgoal Interdependency Graphs) para evaluar dichos *trade-offs*, junto con documentación propia que permitió mantener trazabilidad de las distintas etapas del proyecto, como las mejoras por implementar o las decisiones pendientes de análisis.

Dado el alcance limitado de tiempo, resultó esencial estimar previamente el esfuerzo asociado a cada mejora. Esto permitió priorizar intervenciones factibles dentro del marco temporal del proyecto, postergando aquellas tareas de mayor complejidad, como las altamente centradas en el frontend, las cuales fueron registradas como trabajo futuro.

En conjunto, este trabajo demuestra que es posible aplicar una estrategia sistemática de

mejora sobre un sistema preexistente, incluso en contextos educativos, logrando fortalecer su calidad sin comprometer su funcionalidad ni requerir una reescritura completa.

Dificultades presentadas El autor del presente trabajo es estudiante de último año de la carrera de Ingeniería Civil en Informática, desempeñándose profesionalmente como *Ethical Hacker* y Consultor de Seguridad Ofensiva, con cerca de dos años de experiencia en el rubro. Su labor profesional exige el análisis de arquitecturas, la identificación de vulnerabilidades, la proposición de mejoras y correcciones de seguridad en diversos sistemas.

A pesar de contar con esta experiencia previa, el desarrollo de este trabajo conllevó una serie de dificultades que vale la pena destacar, dado que muchas de ellas son comunes en este tipo de análisis cuando se realizan en contextos educacionales o post-desarrollo. Entre las principales se encuentran:

- Creación de diagramas: El software analizado no contaba originalmente con diagramas de arquitectura, flujo o componentes, por lo que fue necesario crearlos desde cero. A pesar de que el autor está habituado a interpretar diagramas en su labor profesional, el proceso de diseñarlos desde una perspectiva técnica y académica resultó más complejo de lo esperado, especialmente en lo relativo a la elección de herramientas adecuadas y al nivel de detalle requerido.
- Falta de conocimiento sobre metodologías: Durante el desarrollo surgieron múltiples herramientas, técnicas y marcos teóricos que inicialmente eran desconocidos para el autor, como los *Architecture Decision Records* (ADR), los Casos de Uso, los *Soft Goals* y los *Softgoal Interdependency Graphs* (SIGs). Comprender e incorporar adecuadamente estos elementos fue un proceso de aprendizaje paralelo al análisis en sí mismo.
- Desconocimiento de estándares en ingeniería de software: Aunque el autor está familiarizado con estándares aplicables a la ciberseguridad (como el estándar *ISO/IEC 27001*), desconocía qué normativas y estándares son comúnmente utilizados en ingeniería de software. Esta brecha dificultó inicialmente la identificación y adopción de un marco de referencia adecuado para guiar el análisis arquitectónico.
- Contexto de la FESW: El software objeto de análisis fue desarrollado en el marco de la Feria de Software (FESW), lo que trajo consigo ciertas dificultades al presente trabajo:
 - Tiempo transcurrido: El proyecto fue diseñado y desarrollado hace al menos medio año del presente análisis, lo que dificultó la recolección de información técnica precisa y obligó a depender en parte de la memoria del autor y de la información accesible.
 - Colaboración del equipo original: En algunas instancias fue necesario solicitar opiniones o revisiones respecto de las mejoras propuestas o implementadas. Si bien el equipo mostró disposición para colaborar, sus responsabilidades laborales y limitaciones de tiempo dificultaron la disponibilidad constante para apoyar el proceso.

- Investigación adicional para propuestas de mejora: Para proponer mejoras viables al sistema, no solo se estimó el esfuerzo y la factibilidad de implementación, sino que también se requirió investigar en profundidad ciertas funcionalidades. Un caso destacable fue la implementación del principio de no repudio, donde, pese a trabajar en el ámbito de la seguridad, el autor tuvo que profundizar en aspectos técnicos y normativos para asegurar una propuesta sólida y adecuada.

Dado lo anterior, se hace un llamado a futuros estudiantes o profesionales que consideren emprender un trabajo de naturaleza similar: es probable que se enfrenten a desafíos comparables, especialmente si el análisis se realiza sobre un software previamente desarrollado sin documentación formal. Se recomienda anticiparse a estas dificultades mediante una planificación cuidadosa, establecer canales claros de comunicación con los equipos involucrados, y destinar tiempo para la adquisición de conocimientos teóricos complementarios. Enfrentar este tipo de trabajo requiere no solo habilidades técnicas, sino también una considerable capacidad de autogestión y aprendizaje autónomo.

Capítulo 5

CONCLUSIONES

El trabajo reportado en esta memoria da cuenta de la implementación de mejoras a *Audit-Forge*, un sistema de reportería de ciberseguridad, con el objetivo de acercar su madurez técnica y tecnológica a un nivel de producción.

La visión del producto y su implementación se enmarcan en un entorno académico: Taller de Desarrollo de Proyectos de Informática de la carrera de pregrado “Ingeniería Civil Informática” en la Universidad Técnica Federico Santa María en Chile. Varios de los requisitos de atributos de calidad que soportan el paso a producción de un sistema de software son pasados por alto en pos de la satisfacción de requisitos funcionales, así como del cumplimiento de los hitos que se imponen en el Taller.

Para la implementación de mejoras, se estableció como objetivo general la aplicación de técnicas sistemáticas de la ingeniería de software (específicamente, la práctica “evaluación de arquitectura de software”) que permitieran identificar, analizar e implementar mejoras significativas a nivel arquitectónico. El énfasis en el nivel arquitectónico radica en el hecho de que la ingeniería de software entiende a la arquitectura de software como el principal habilitador (o inhibidor) de los requisitos de atributos de calidad.

En cuanto al cumplimiento de los objetivos específicos, se logró:

- Identificar áreas críticas de mejora mediante un enfoque estructurado, basado en una evaluación de arquitectura de software siguiendo los lineamientos dados en [30]. La identificación de atributos de calidad se basó en la adopción del modelo de calidad *ISO/IEC 25010:2011* [9]. Además, el enfoque utilizado para la evaluación de la arquitectura de software es centrado en decisiones arquitectónicas.
- Evaluar los resultados mediante la actualización de diagramas arquitectónicos, análisis retrospectivo de decisiones clave y uso de “Softgoal Interdependency Graphs” (SIGs) que evidenciaron el efecto, tanto positivo como negativo (tradeoffs), de las mejoras sobre atributos como la seguridad, fiabilidad y usabilidad.

- Diseñar e implementar una serie de mejoras priorizadas, considerando criterios de impacto, factibilidad técnica y costo temporal, lo que derivó en una evolución significativa del sistema sin comprometer su funcionalidad.

Una de las principales contribuciones del trabajo fue visibilizar y abordar atributos de calidad frecuentemente desatendidos en proyectos académicos, como la portabilidad, trazabilidad, mantenibilidad y seguridad. Esto se tradujo en mejoras concretas que refuerzan el sistema para futuros escenarios de despliegue o evolución. Asimismo, se consolidó una base documental robusta —incluyendo ADRs, casos de uso y modelado arquitectónico— que facilita su entendimiento y extensión futura.

El alcance del trabajo se vio naturalmente limitado por el tiempo disponible, lo que implicó priorizar mejoras técnicamente viables y dejar otras como trabajo futuro. A pesar de estas restricciones, se evidenció que una estrategia sistemática, apoyada en herramientas formales de documentación y análisis, permite lograr avances significativos en la calidad de sistemas existentes, incluso cuando estos fueron desarrollados en contextos educativos.

A partir del análisis arquitectónico realizado y considerando los criterios de impacto, factibilidad y aporte a la madurez del sistema, se seleccionaron e implementaron las siguientes mejoras prioritarias:

1. Comunicación segura: Mediante una refactorización en la comunicación Backend-Modelo IA, se corrigieron problemas introducidos durante el desarrollo del software, fortaleciendo así la seguridad en la comunicación de dichos componentes. Además, se implementó una correcta validación de los certificados en la comunicación *HTTP(S)*.
2. Log de acciones: Se incorporó un sistema de registro de eventos en el backend, que almacena acciones relevantes ejecutadas por los usuarios. Esto permite llevar una trazabilidad y apoyar el principio de no repudio. Los registros almacenan información como el usuario, la acción realizada, el estado de la respuesta y una firma digital para asegurar su integridad, excluyendo campos sensibles como contraseñas, tokens o similares. Esta mejora sienta las bases para un sistema más auditable y seguro.
3. Detección de desconexión: Se implementó un mecanismo en el frontend utilizando WebSockets, que permite detectar la pérdida de conexión con el backend. Ante una desconexión inesperada, se informa al usuario mediante una superposición visual (overlay), mejorando la experiencia de usuario y la percepción de madurez del sistema.

Estas mejoras fortalecen atributos clave de calidad del sistema, tales como la seguridad, fiabilidad y experiencia de usuario, alineándose con los objetivos establecidos en la evaluación arquitectónica.

Como trabajo futuro, se sugiere profundizar en las mejoras relacionadas con el frontend, abordar los *issues* identificados durante la ejecución del proyecto, e integrar pruebas auto-

matizadas que permitan verificar el correcto funcionamiento del sistema. Asimismo, se propone extender la funcionalidad del componente de trazabilidad, incorporando la capacidad de detectar actividades sospechosas y generar alertas dirigidas al administrador. También se recomienda implementar mecanismos que faciliten la adopción o migración de datos, lo cual favorecerá la experiencia de los usuarios y la adopción del software.

Para quienes deseen abordar trabajos similares, se aconseja realizar una planificación detallada del esfuerzo requerido por cada tarea, definir con claridad los objetivos de mejora y apoyarse en herramientas visuales y documentales que faciliten la comprensión y evaluación del sistema a lo largo del proceso.

En síntesis, este trabajo demostró que es posible aplicar principios avanzados de la ingeniería de software en entornos académicos, logrando fortalecer la arquitectura de un sistema real sin dejar de lado los objetivos del software y/o negocio.

ADR 13: Despliegue del modelo de IA en contenedor separado

Estado: Aceptada

Contexto: El sistema debía incorporar una funcionalidad de análisis asistido por inteligencia artificial. Se plantearon dos opciones: incorporar el modelo IA directamente dentro del backend o desplegarlo como un microservicio en un contenedor independiente.

Decisión: Se optó por desplegar el modelo IA como un servicio separado, con su propia API REST, dentro de un contenedor específico. Esta decisión permite mantener una separación clara de responsabilidades y facilitar el mantenimiento y la escalabilidad del sistema.

Consecuencias:

- **Ventajas:**
 - Permite escalar el modelo IA de manera independiente del resto del sistema.
 - Facilita actualizaciones, mejoras o sustituciones del modelo sin afectar el backend.
 - Mejora la modularidad y el aislamiento de fallos.
- **Desventajas:**
 - Requiere definir y mantener una interfaz clara (API) entre backend y modelo IA.
 - Puede aumentar la latencia ligeramente al depender de una comunicación entre servicios.

Figura 5.2: ADR: Despliegue del modelo de IA en contenedor separado

Fuente: Elaboración propia

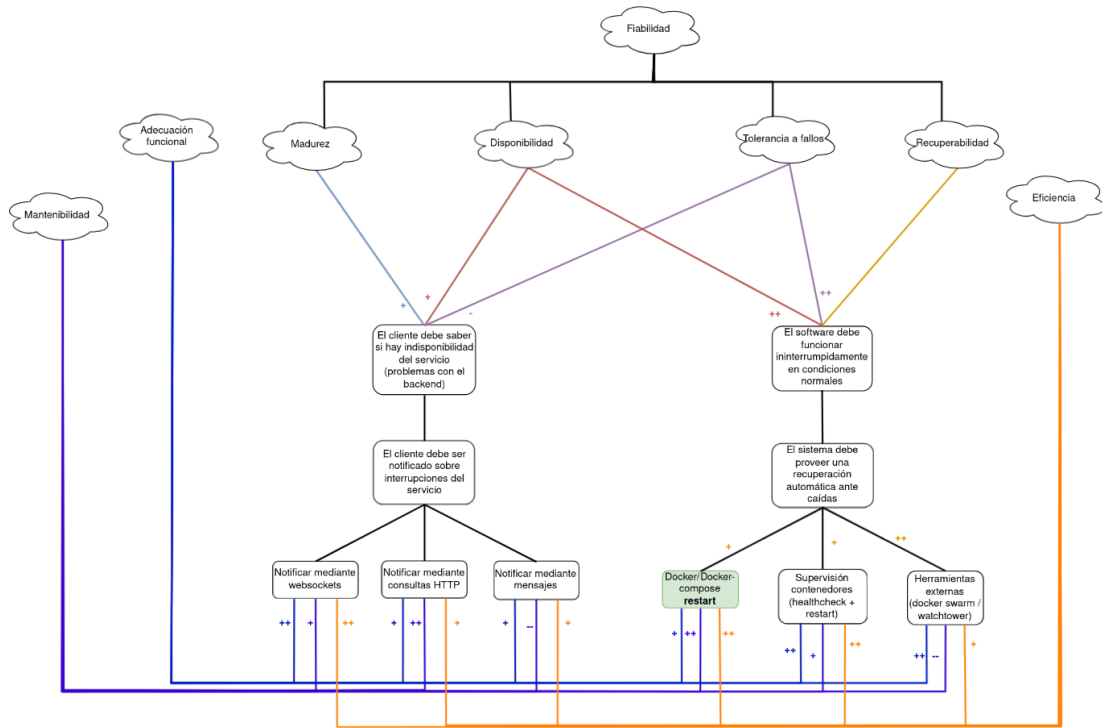


Figura 5.4: SIG inicial: Fiabilidad

Fuente: Elaboración propia

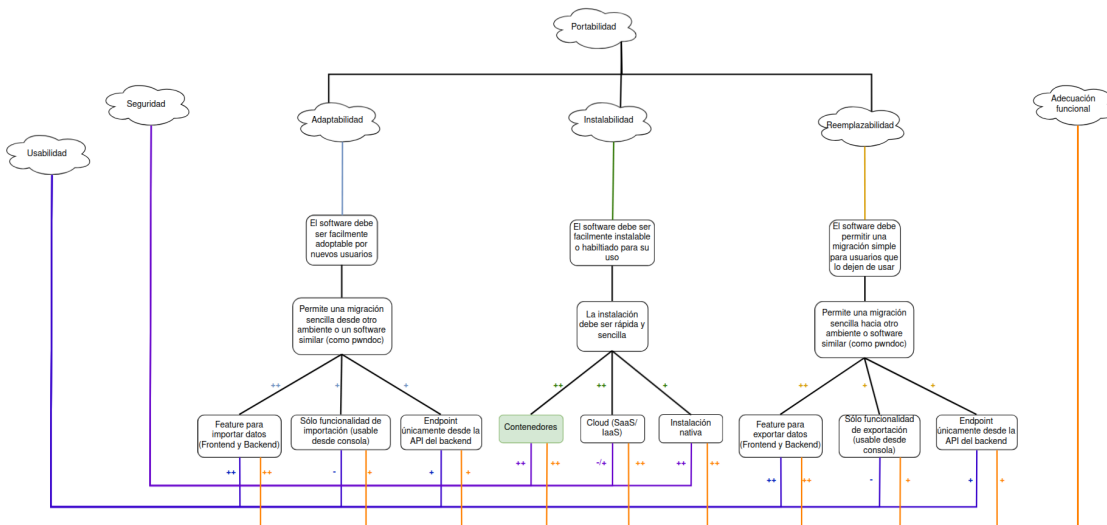


Figura 5.5: SIG inicial: Portabilidad

Fuente: Elaboración propia

IMPLEMENTACIÓN DE MEJORAS A UNA PLATAFORMA PARA LA REPORTERÍA DE AUDITORÍAS DE CIBERSEGURIDAD, GUIADA POR UNA EVALUACIÓN DE ARQUITECTURA DE SOFTWARE: CASO AUDITFORGE

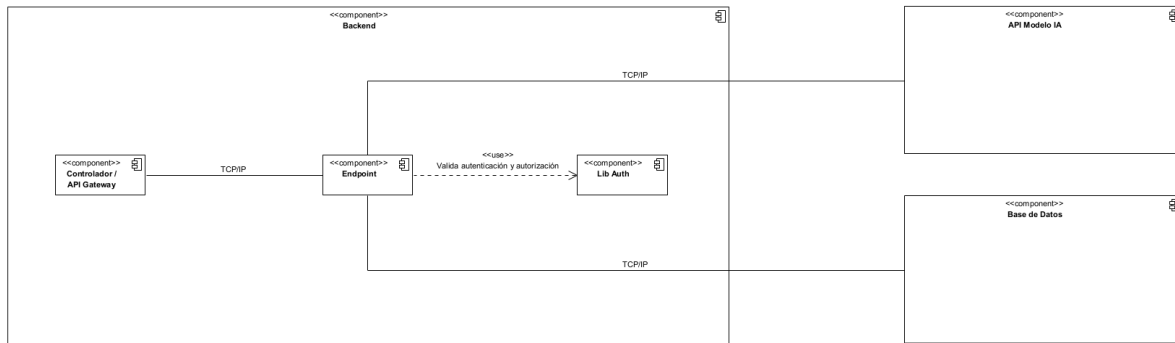


Figura 5.6: Diagrama inicial de despliegue backend
Fuente: Elaboración propia

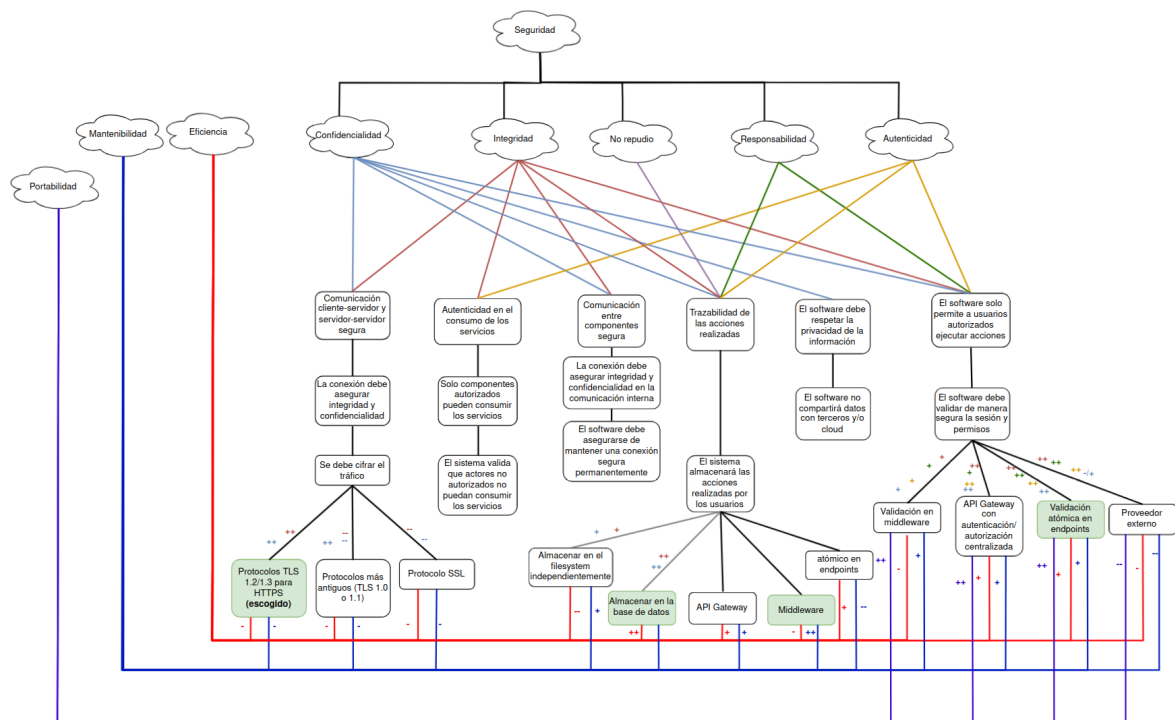


Figura 5.7: SIG actual: Seguridad
Fuente: Elaboración propia

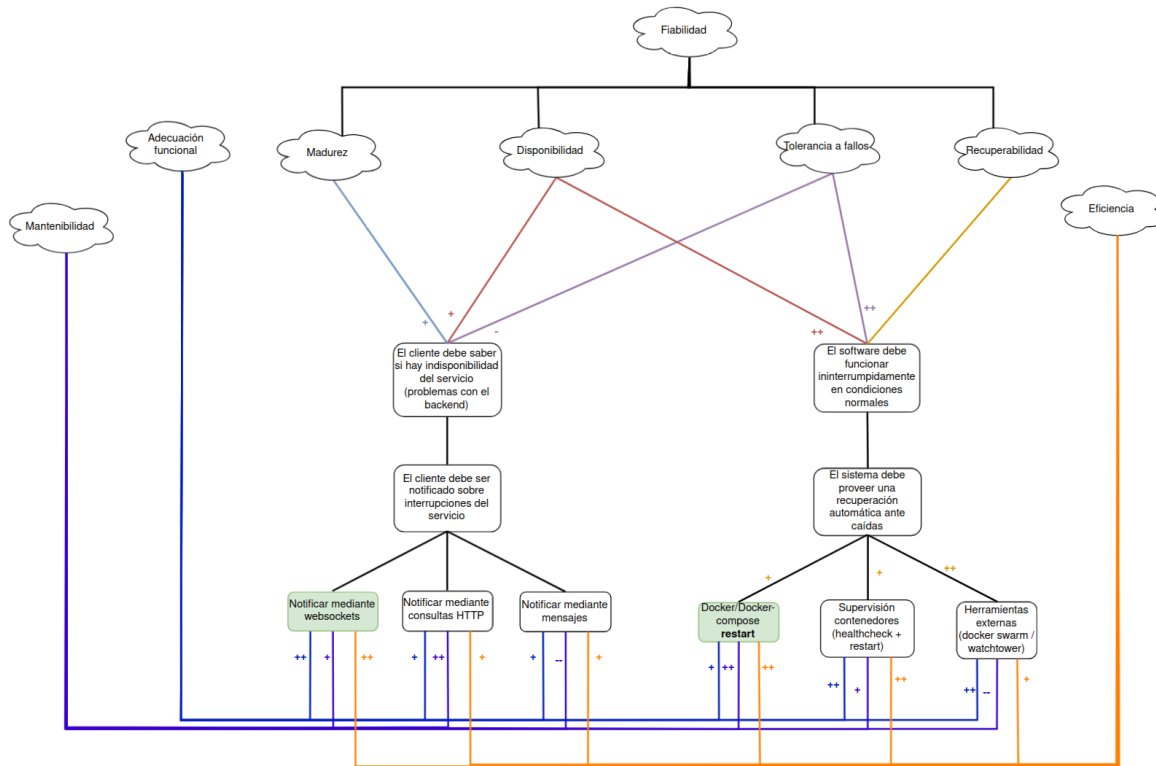


Figura 5.8: SIG actual: Fiabilidad
Fuente: Elaboración propia

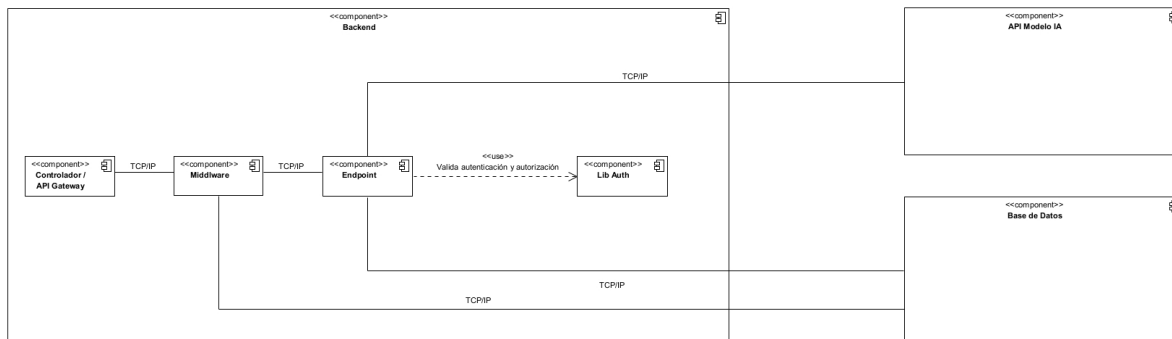


Figura 5.9: Diagrama actual de despliegue backend
Fuente: Elaboración propia

REFERENCIAS BIBLIOGRÁFICAS

- [1] FESW, "Departamento de Informática, Universidad Técnica Federico Santa María," 2024, <https://www.feriadesoftware.cl/>. Consultado el: 24 de Octubre, 2024. [Online]. Available: <https://www.feriadesoftware.cl/>
- [2] AuditForge, "Auditforge," 2024, <https://auditforge.feriadesoftware.cl/>. Consultado el: 24 de Octubre, 2024. [Online]. Available: <https://auditforge.feriadesoftware.cl/>
- [3] H. Zhu, Q. Zhang, and Y. Zhang, *HASARD*, 12 2014, pp. 123–156.
- [4] J. A. McCall, s. P. K. Richard, and G. F. Walters, "Factors in Software Quality: Volume II," 1977, <https://apps.dtic.mil/sti/tr/pdf/ADA049015.pdf>.
- [5] B. Boehm, *Characteristics of Software Quality*. North Holland Press, 1978.
- [6] R. B. Grady and D. L. Caswell, *Software metrics: establishing a company-wide program*. USA: Prentice-Hall, Inc., 1987.
- [7] R. Dromey, "A model for software product quality," *IEEE Transactions on Software Engineering*, vol. 21, no. 2, pp. 146–162, 1995.
- [8] ISO/IEC 9126, *International Standard ISO/IEC 9126, Information technology ? Product Quality ? Part1: Quality Model*, Std., 2001.
- [9] ISO/IEC 25010, *ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*, Std., 2011.
- [10] A. AL-Badareen, M. Selamat, M. A. Jabar, J. Din, and S. Turaev, "Software quality models: A comparative study," vol. 179, 01 2011, pp. 46–55.
- [11] ISO/IEC 25010, *ISO/IEC 25010:2023, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model*, Std., 2023.
- [12] Carnegie Mellon University, "Software architecture," <https://insights.sei.cmu.edu/software-architecture/>. Consultado el: 30 de Abril, 2025. [Online]. Available: <https://insights.sei.cmu.edu/software-architecture/>
- [13] L. Dobrica and E. Niemela, "A survey on software architecture analysis methods," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 638–653, 2002.
- [14] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, ser. SEI Series in Software Engineering. Boston, MA: Addison-Wesley, 2001.

- [15] N. A. Ernst, J. Klein, M. Bartolini, J. Coles, and N. Rees, "Architecting complex, long-lived scientific software," *Journal of Systems and Software*, vol. 204, p. 111732, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0164121223001279>
- [16] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory and Practice*. Addison-Wesley, 2007.
- [17] N. Rozanski and E. Woods, *Software Systems Architecture: Working with Stakeholders using Viewpoints and Perspectives*, 2nd ed. Addison Wesley, 2011.
- [18] H. Cervantes and R. Kazman, *Designing Software Architectures: A Practical Approach*, 2nd ed. Addison-Wesley Professional, 2024.
- [19] F. Losavio and C. Guillen-Drija, "Comparación de métodos para la arquitectura del software: Un marco de referencia para un método arquitectónico unificado," *Revista de la Facultad de Ingeniería Universidad Central de Venezuela*, vol. 25, pp. 71-87, 03 2010.
- [20] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-based analysis of software architecture," *Software, IEEE*, 1996.
- [21] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method." 1998.
- [22] R. Kazman, J. Asundi, and M. Klein, "Quantifying the costs and benefits of architectural decisions," in *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, 2001, pp. 297-306.
- [23] P. Clements, "Active reviews for intermediate designs," p. 26, 08 2000.
- [24] P. Bergtsson, N. Lassing, J. Bosch, and H. Vliet, "Architecture-level modifiability analysis (alma)," *Journal of Systems and Software*, vol. 69, pp. 129-147, 01 2004.
- [25] Barbacci, Mario and Ellison, Robert and Lattanze, Anthony and Stafford, Judith and Weinstock, Charles and Wood, William, "Quality Attribute Workshops (QAWs), Third Edition," Carnegie Mellon University, Software Engineering Institute's Digital Library, Tech. Rep. CMU/SEI-2003-TR-016, 2003, consultado el: 12 de Junio, 2025.
- [26] U. Heesch, V.-P. Eloranta, P. Avgeriou, K. Koskimies, and N. Harrison, "Decision-centric architecture reviews," *Software, IEEE*, vol. 31, pp. 69-76, 2014.
- [27] Fortinet, " ¿Qué es Ciberseguridad? ," 2024, <https://www.fortinet.com/lat/resources/cyberglossary/what-is-cybersecurity>. Consultado el: 10 de Junio, 2025. [Online]. Available: <https://www.fortinet.com/lat/resources/cyberglossary/what-is-cybersecurity>
- [28] International Software Testing Qualifications Board, "Glossary," https://glossary.istqb.org/en_US/term/penetration-testing?term=&exact_matches_first=true&page=17. Consultado el: 17 de Junio, 2025. [Online]. Available: https://glossary.istqb.org/en_US/term/penetration-testing?term=&exact_matches_first=true&page=17

- [29] Penetration Testing Execution Standard, “ Main Page ,” 2014, http://www.pentest-standard.org/index.php/Main_Page. Consultado el: 20 de Mayo, 2025. [Online]. Available: http://www.pentest-standard.org/index.php/Main_Page
- [30] P. Cruz, H. Astudillo, and C. M. Zapata-Jaramillo, “Extending the semat kernel to represent and assess software architecture evaluations,” in *2023 XLIX Latin American Computer Conference (CLEI)*, 2023, pp. 1–8.