UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño e implementación de la ciberseguridad y el aseguramiento de la calidad para una plataforma web de gestión administrativa de un ingenio azucarero en Guatemala

Trabajo de graduación en la modalidad de trabajo profesional
presentado por Mario Antonio Guerra Morales
para optar al grado académico de Licenciado en Ingeniería en Ciencias
de la Computación y Tecnologías de la Información

Guatemala,

2025

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño e implementación de la ciberseguridad y el aseguramiento de la calidad para una plataforma web de gestión administrativa de un ingenio azucarero en Guatemala

Trabajo de graduación en la modalidad de trabajo profesional
presentado por Mario Antonio Guerra Morales
para optar al grado académico de Licenciado en Ingeniería en Ciencias
de la Computación y Tecnologías de la Información

Guatemala,

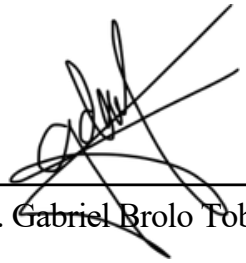
2025

Vo.Bo.:

(f) 

Ing. Gabriel Brolo Tobar

Tribunal Examinador:

(f) 

Ing. Gabriel Brolo Tobar

(f) 

Ing. Marlon Osiris Fuentes López

Fecha de aprobación: Guatemala, 22 de noviembre de 2025.

Prefacio

La elaboración de este proyecto surge de la necesidad de dotar a una plataforma web de formularios adaptables orientada al sector agrícola, por medio de un módulo de ciberseguridad y aseguramiento de calidad que garantice la protección de la información, la confiabilidad y la continuidad operativa. El autor, quien cuenta con interés en la ciberseguridad y en plasmar un enfoque orientado al desarrollo, la seguridad y las operaciones dentro de una plataforma, tuvo la oportunidad de aplicar sus conocimientos previos y así como para aprender e investigar más allá de esta área que le ha fascinado tanto.

El documento en cuestión adopta un enfoque de *DevSecOps* con la integración de diversas prácticas de ciberseguridad y aseguramiento de calidad en cada etapa del ciclo de vida del desarrollo de software. Tomando en cuenta esta visión, se orientaron las decisiones técnicas y metodológicas, tomando en cuenta el balance de los posibles riesgos, costos de implementación y mantenibilidad. Esto con el objetivo de no sólo ofrecer una lista de herramientas para utilizar, sino también proponer un marco reproducible y auditable que fortalezca la calidad y seguridad en proyectos web con requerimientos reales.

Finalmente, se agradece al Ingenio Santa Ana por brindar la oportunidad de realizar este proyecto y estar atentos ante cualquier necesidad que se presentase. A los asesores Gabriel Brolo y Erick Marroquín, por brindar su contribución y apoyo con sus conocimientos y su retroalimentación para llevar a cabo el módulo de la mejor manera posible. A los compañeros y amigos: Linda Jiménez, Diego Hernández, Javier Alvarado, Daniel Valdez, Andrea Ramírez, Adrian Flores y Emilio Solano, con quienes se desarrolló este proyecto grupal y acompañaron durante todo el tiempo de la carrera en la universidad. Y por último, pero no menos importante, a sus hermanos, José Guerra y Luis Guerra, y a sus padres, Arnoldo Guerra y Rozana Morales, por su apoyo constante tanto en lo material, como en lo emocional.

Índice

Lista de figuras

Figura **Página**

Lista de cuadros

Cuadro **Página**

Resumen

Este trabajo diseña e implementa un módulo de ciberseguridad y aseguramiento de calidad para una plataforma web de formularios adaptables en el sector agrícola. El módulo cuenta con el objetivo de integrar controles de acceso, detección de vulnerabilidades, pruebas automatizadas, flujos *CI/CD*, políticas de protección de la información bajo prácticas de seguridad de la información, seguridad de aplicaciones, *DevSecOps* y aseguramiento de calidad. Para alcanzarlo, se configuraron análisis estáticos con SonarCloud (vulnerabilidades, *issues*, cobertura, mantenibilidad y confiabilidad); escaneo de secretos en TruffleHog; análisis de dependencias con Snyk; implementación de *Content Security Policy* en el *frontend*; cifrado TLS en servicios publicados; pruebas de penetración automatizadas con OWASP ZAP en entornos locales; validación del despliegue en *Google Cloud Run*, junto con el refuerzo de la imagen de contenedor Docker.

Los principales resultados incluyen la superación de la *quality gate* con calificaciones A en mantenibilidad, seguridad y confiabilidad; más del 65 % de cobertura total (71 % en *frontend* y 84 % en *backend*) y menos del 5 % de código duplicado; ausencia de *security hotspots* y de vulnerabilidades en dependencias; sin alertas de alta severidad en análisis dinámico; imagen del contenedor sin vulnerabilidades y correcta operación del servicio en *Cloud Run*. Se documentaron las directivas CSP y la presencia de TLS y Cloudflare en la infraestructura. En conjunto, se estableció un proceso reproducible que reduce la exposición de la plataforma a vulnerabilidades críticas, mediante la integración de herramientas de escaneo y políticas de seguridad, fortaleciendo así la postura de seguridad del sistema.

Abstract

This work designs and implements a cybersecurity and quality assurance module for a web platform of adaptable forms in the agricultural sector. The module aims to integrate access controls, vulnerability detection, automated testing, CI/CD pipelines, and information-protection policies, following information security, application security, DevSecOps, and quality assurance practices. To achieve this, the solution configures static analysis with SonarCloud (vulnerabilities, issues, coverage, maintainability, and reliability), secret scanning with TruffleHog, dependency analysis with Snyk, Content Security Policy on the frontend, TLS encryption for published services, automated penetration testing with OWASP ZAP in local environments, and deployment validation on Google Cloud Run, alongside hardening of the Docker container image.

The main results include passing the quality gate with A ratings in maintainability, security, and reliability; overall test coverage above 65 % (71 % in the frontend and 84 % in the backend) with less than 5 % duplicated code; no security hotspots or vulnerable dependencies; no high-severity alerts in dynamic analysis; a vulnerability-free container image; and correct service operation on Cloud Run. CSP directives, as well as the presence of TLS and Cloudflare in the infrastructure, were documented. Overall, the project establishes a reproducible process that reduces the platform's exposure to critical vulnerabilities through the integrated use of scanning tools and security policies, thereby strengthening the system's security posture.

CAPÍTULO 1

Introducción

La digitalización en el sector agrícola transformó significativamente en la forma de recopilar, organizar y analizar datos de campo. Los formularios digitales permiten capturar información en sitio, procesarla en servicios de bases de datos y consultarla a través de plataformas web y móviles. Conjuntamente, este proceso se realiza mediante libretas de campo en papel, hojas de cálculo en las cuales se ingresan los datos recolectados durante el día, y también cargas masivas de datos desde sensores y archivos semiestructurados. Los formularios digitales han permitido una mejora significativa en el uso de recursos, al reducir errores manuales, disminuir el consumo de papel y automatizar procesos en entornos industriales como un ingenio azucarero. Sin embargo, estas soluciones no cuentan con la suficiente escalabilidad ni flexibilidad para considerarse formularios totalmente adaptables.

Actualmente, el Ingenio Santa Ana utiliza una plataforma de terceros para realizar procesos de recolección de datos y formularios digitales. Esta herramienta lleva el nombre de Digiforms. Si bien, es capaz de brindar analíticas y dispone de características que ayudan a la elaboración y adaptación de formularios, los datos almacenados en los servidores no los controla directamente el ingenio. Impidiendo así, cumplir con los principios básicos de ciberseguridad: confidencialidad (al exponer datos sensibles a un tercero); integridad (al existir un riesgo de manipulación de información no detectado); y disponibilidad (la incapacidad de ejercer medidas preventivas ante alguna falla o incidente). Además de no contar con la evidencia pública de cumplimiento de requisitos y políticas actualizadas en criterios de aceptación, métricas, cobertura de pruebas automatizadas ni evidencias reproducibles de aseguramiento de calidad alineadas con los principios del ciclo de vida de desarrollo de software y con *DevSecOps*.

El megaproyecto titulado “Recolección, visualización y análisis de formularios adaptables en campos de acción agrícola”, propone un desarrollo de una solución que ayude a resolver todos estos problemas. El megaproyecto desarrolla una plataforma web de gestión administrativa para estos formularios, y una plataforma móvil para recolección de datos desde el campo. La plataforma web ayuda a definir formularios que puedan ser creados de forma dinámica, por medio de reglas condicionales, validaciones, y la configuración adicional de

ciertos campos dependiendo de la necesidad y del objetivo del formulario en cuestión. Por otra parte, la plataforma móvil interpreta de manera visual los formularios en los dispositivos para facilitar su llenado y posterior envío al servidor de base de datos del ingenio.

Dentro de este megaproyecto está definido un módulo de ciberseguridad y de aseguramiento de calidad en la plataforma web. El módulo realiza, según las fases del ciclo de vida del desarrollo de software, una investigación y análisis, un diseño de la aplicación con la arquitectura que se dispondrá para la plataforma y su posterior implementación. Este módulo nace como respuesta a las necesidades del ingenio Santa Ana de proteger los datos operativos y personales, tanto de la recolección de datos en formularios adaptables, como también para sus trabajadores y quienes utilicen esta plataforma. Además de garantizar la continuidad del servicio después de la entrega del mismo, estandarizar prácticas de desarrollo y de operación y demostrar cumplimiento en la plataforma por medio de evidencia auditable, desde la definición de controles hasta su verificación continua.

Para priorizar la mitigación de riesgos con mayor impacto sobre los datos y el servicio, ofrecer trazabilidad y métricas objetivas y alinearse con las restricciones operativas y tecnológicas de la organización, el módulo integra seguridad desde el diseño bajo los procesos orientados a *DevSecOps* (desarrollo, seguridad y operaciones). Automatizando controles mediante *CI/CD* (integración y entrega continua). También, la integración de controles lógicos de acceso y de la plataforma como control de sesiones, autorización basada en roles y atributos para operaciones críticas para los empleados del ingenio que la emplearán. Escaneo de vulnerabilidades y analizadores de código estático y dinámico para prevenir defectos antes y después del despliegue. Contando con principios de *DevEx* (*developer experience*) para asegurar una aplicación constante y consistente de los controles establecidos, y la implementación de diversos tipos de pruebas unitarias, de integración, *end-to-end* y de desempeño para comprobar el rendimiento adecuado de la plataforma en conjunto con flujos de integración continua y de entrega continua.

CAPÍTULO 2

Justificación

En los últimos años, Guatemala se ha visto envuelta en un constante aumento de ataques cibernéticos en el ámbito laboral como también en el personal. Abarcando desde estafas y fraudes en línea por medio de phishing e ingeniería social, hasta ataques a instituciones gubernamentales y a empresas guatemaltecas. A medida que los criminales se vuelven más sofisticados y hábiles, las vulnerabilidades en sistemas de todo el país se verán cada vez más comprometidas. Por lo que es evidente fortalecer la infraestructura digital y educar a los ciudadanos y a las empresas (Observatorio Guatemalteco de Delitos Informáticos, 2024).

Durante la primera semana de mayo de 2024, varios virus y botnets han sido las principales amenazas cibernéticas en el país, además de amenazas de tipo ransomware como los denominados troyanos, siendo estos relacionados a diversas versiones del sistema operativo Windows (Comité Nacional de Seguridad Cibernética, 2024).

Además, en el marco internacional, se pueden encontrar diversas estadísticas relacionadas a ataques cibernéticos y vulnerabilidades. Por ejemplo, se estima que el costo promedio de ataques hacia la información y los datos a nivel mundial es de 4.45 millones de dólares. También, el 4.6 por ciento de las aplicaciones web suelen tener vulnerabilidades críticas al sistema y que, antes de comenzar el año 2025, se encontraron un total de 37,902 vulnerabilidades y exposiciones comunes (CVE) (ZeroThreat, 2024).

En el caso de aseguramiento de calidad, se estima que los negocios relacionados a tecnologías de la información son capaces de destinar el 23 por ciento de su presupuesto anual hacia el mejoramiento de la calidad del producto, prevenir errores después del lanzamiento del proyecto y asegurar la satisfacción del cliente (Dushevin, 2024). Estos siendo algunos de los principales enfoques de aseguramiento de calidad (QA) para el desarrollo eficiente y apropiado de un proyecto.

Para el desarrollo de aplicaciones web, la importancia de aplicar aseguramiento de calidad en los proyectos proporciona una gran ventaja y una importancia crucial para reducir el margen de errores de este. Identificando potenciales problemas durante el desarrollo y resolviéndolos antes de la entrega al cliente ayuda a generar experiencias positivas entre los

clientes hacia un producto final (Indeed Editorial Team, 2025). Aunado a esto, se añaden conceptos de *developer experience* (*DevEx*). Este se refiere a la experiencia en general de los desarrolladores en su interacción con su ambiente de trabajo, procesos y entornos para el desarrollo de software. Implementar *DevEx* con *QA* ayuda enormemente a la calidad de un software, ya que esto permite que se creen entornos donde los procesos de calidad y flujos de trabajo se adecúen a los desarrolladores, facilitando así, la adopción de prácticas de aseguramiento de calidad y la entrega de un software de calidad (Perry, 2024).

El aseguramiento de calidad, como uno de sus objetivos, consiste también en alcanzar una sostenibilidad a largo plazo. Para ello, se requieren políticas que vayan con los estándares y regulaciones definidas por el cliente y por el equipo de desarrollo o empresa. Además de la usabilidad y la confiabilidad que se espera en ciertas condiciones para el software (Murali, 2024).

CAPÍTULO 3

Objetivos

3.1. Objetivo general

Implementar procesos, herramientas y técnicas de ciberseguridad y de aseguramiento de calidad en un sistema web de elaboración de formularios adaptables para el sector agrícola que fortalezcan la protección de la información y operación continua del sistema.

3.2. Objetivos específicos

- Desarrollar procesos de ciberseguridad para control de acceso, gestión de información y organización de servicios mediante autenticación segura en rutas críticas, encriptación de datos en todos los servicios publicados, *hashing* de credenciales, políticas de acceso y monitoreo de seguridad.
- Integrar detección y prevención de vulnerabilidades en servicios y flujos de desarrollo con herramientas de análisis de código, pruebas automatizadas y escaneo de dependencias durante todo el ciclo de vida del software, alcanzando puertas de calidad con calificación A, cobertura de código de más del 65 % y duplicación de código de menos del 5 %.
- Configurar e implementar protocolos, herramientas y flujos de trabajo mediante pruebas automatizadas, validaciones de código, integración continua y revisión de cambios que faciliten la aplicación de buenas prácticas de aseguramiento de calidad que aseguren la mantenibilidad del sistema.

CAPÍTULO 4

Marco teórico

4.1. Definiciones generales

4.1.1. Ciberseguridad

La ciberseguridad es la práctica de proteger equipos, redes, aplicaciones de software, sistemas críticos, datos e información de posibles amenazas digitales (Amazon Web Services, 2024b). Esta se rige en base a sus tres principios: confidencialidad, integridad y disponibilidad. Debe salvaguardar frente a diversas amenazas hacia la información o los servicios frente a accesos no autorizados, ataques cibernéticos, entre otras.

Los principios de la ciberseguridad son los siguientes:

- Confidencialidad: garantizar que la información esté disponible sólo a las personas autorizadas. Los datos deben estar restringidos para terceros no autorizados (Universidad de San Marcos, 2024).
- Integridad: la información debe permanecer sin cambios, a menos que esto sea autorizado. Nadie debe modificar los datos sin permiso, garantizando la veracidad de la información almacenada (Universidad de San Marcos, 2024).
- Disponibilidad: garantiza que la información esté disponible cuando los usuarios autorizados los soliciten (Universidad de San Marcos, 2024).

4.1.2. Aseguramiento de calidad (QA)

El aseguramiento de calidad o *quality assurance* (*QA*) es un proceso sistemático que asegura a ciertos productos, procesos y servicios cumplir con ciertos estándares de calidad que estén definidos (ComplianceQuest, 2025). Es definido por la norma ISO 9000 como

"una parte del control de calidad enfocada a proveer confiabilidad y asegurarse de que los requisitos de calidad serán alcanzados".

La importancia del aseguramiento de calidad en el desarrollo de software radica en puntos importantes como la capacidad de detectar y corregir problemas antes de entregar el producto al usuario final; promover y alcanzar mejoras de la calidad del producto; brindar una estrategia de pruebas efectiva para garantizar el buen funcionamiento del software; y brindar satisfacción al cliente con un producto final libre de errores (ACL, 2024).

La optimización de procesos de *QA* reduce el riesgo técnico y operativo al detectar defectos y vulnerabilidades de manera temprana, mejorando así la fiabilidad del producto final. Entre los beneficios que se brindan al realizar la optimización se encuentran un desarrollo más ágil, la reducción de incidentes y costos de producción, y un mejoramiento continuo en el desarrollo, cumpliendo e incrementando la confianza del cliente para brindar una mejor calidad en el software (Dushevin, 2024).

4.1.3. *DevSecOps*

DevSecOps es la práctica de integrar las pruebas de seguridad en cada etapa de proceso de desarrollo de software (Amazon Web Services, 2024a). Es también, un acrónimo de "desarrollo, seguridad y operaciones", en donde el desarrollo se concentra en el proceso de planificación, codificación y prueba, la seguridad se enfoca en introducirla desde etapas tempranas del desarrollo de software, y las operaciones supervisan y solucionan problemas que surjan del software. *DevSecOps* también se aplica hacia la infraestructura como código. Asegurándose que, todo aquello relacionado a la infraestructura en un software, como flujos de trabajo, plantillas de infraestructura y *scripts* automatizados sean validados antes de llegar a producción. Logrando la detección temprana de desconfiguraciones de seguridad, reducción de riesgo de vulnerabilidades en producción, monitoreo continuo de la infraestructura y revisiones de cumplimiento en *pipelines* automatizados (ThinkCloudly, 2025).

El objetivo de *DevSecOps* es ayudar al equipo de desarrollo a abordar problemas de seguridad y operaciones de forma eficaz, apoyándose también de las etapas del ciclo de vida del desarrollo de software. Aporta beneficios para la detección de vulnerabilidades en fases tempranas del desarrollo. Añade confiabilidad al proyecto debido al añadirse procesos automatizados para hacer análisis del código que se adhiera a los estándares requeridos. Ayuda a que el desarrollo y la infraestructura sean más eficientes y escalables al implementar mejoras continuas y colaborar entre los equipos de desarrollo, seguridad y operaciones (Patni, 2024).

4.1.4. *Developer experience (DevEx)*

La experiencia del desarrollador o *DevEx* es cómo los desarrolladores perciben e interactúan con el entorno o ambiente de desarrollo y trabajo. Esto incluye herramientas y procesos. *DevEx* encuentra su importancia al momento de codificar, realizar y operar software rápidamente y de forma eficaz, garantizando también que los desarrolladores se encuentren cómodos con sus herramientas y flujos de trabajo (Villalba, 2023).

4.1.5. ITIL

ITIL o Information Technology Infrastructure Library es el marco de referencia de las mejores prácticas de Service Management más popular en el mundo. ITIL en su versión 4 incluye conceptos como el sistema de valor del servicio, pretendiendo mostrar cómo el uso e implementación de varios componentes en un servicio asiste en la conversión de la demanda y de las oportunidades en valor en el negocio al cual se busca aplicar, además de apoyarse sobre cuatro dimensiones de Service Management (ITIL, 2020).

Las cuatro dimensiones tienen como concepto ser perspectivas relevantes dentro del sistema, estas son: *organizations and people, information and technologies, partners and providers, y value streams and processes* (ITIL, 2020).

Dentro del sistema de valor del servicio, se encuentran cinco elementos fundamentales para la conversión de demanda en valor. Estos son la cadena de valor del servicio (SVC), sus siete principios guía, gobernanza, las treinta y cuatro prácticas de gestión y la mejora continua.

La cadena de valor del servicio es la parte operativa del SVS y ayuda a conceptualizar la creación, entrega y mejora continua de los servicios a nivel de operaciones. En esta hay seis actividades principales: *plan, improve, engage, design and transition, obtain/build, deliver and support* (Anand, 2025).

Los principios rectores de ITIL 4 son recomendaciones que guían a las organizaciones en todas las circunstancias posibles sin importar los cambios en objetivos, estrategias, estructura o tipo de trabajo (Mann, 2025). Los siete principios rectores de ITIL son los siguientes:

- Centrarse en el valor: todo lo que la organización hace debe relacionarse, directa o indirectamente con el valor para las partes interesadas.
- Empiece donde está: no empezar de cero y construya algo nuevo sin tener en cuenta lo que ya está disponible para aprovechar.
- Progresar iterativamente con comentarios: no intentar hacer todo a la vez, las grandes iniciativas deben llevarse a cabo de manera iterativa.
- Colaborar y promover la visibilidad: trabajar juntos más allá de las fronteras produce resultados que tienen una mayor aceptación, mayor relevancia para los objetivos y probabilidad de éxito a largo plazo.
- Pensar y trabajar de manera integral: ningún servicio o elemento utilizado para prestar un servicio es independiente. Los resultados obtenidos se resentirán a menos que la organización trabaje en el servicio como un todo, no sólo por partes.
- Mantenerlo simple y práctico: si un proceso, servicio, acción o métrica no aporta valor o un resultado útil, debe eliminarlo.
- Optimizar y automatizar: los recursos de todo tipo, particularmente los recursos humanos, deben aprovecharse al máximo. Las prácticas de gestión de ITIL son un conjunto

de recursos organizacionales los cuales fueron diseñados para realizar un trabajo o alcanzar un objetivo. Para estas hay tres tipos: prácticas generales de gestión, prácticas de gestión de servicios y prácticas de gestión técnica (Mann, 2025).

La gobernanza para ITIL 4 cubre tres aspectos clave: la evaluación del cambio, la dirección y el monitoreo del rendimiento de la organización. Mientras que, la mejora continua identifica procesos de mejora en la organización en general, alineando continuamente los servicios y operaciones de TI con las necesidades del negocio (Mann, 2025).

4.1.6. Privacidad de los datos y *compliance*

La privacidad de los datos (*data privacy*) son las prácticas, políticas y procedimientos que una organización implementa para asegurarse que se adhieren a todos los estándares y regulaciones legales referentes a la información privada de los usuarios. Implementar *data privacy and compliance* provee varios beneficios, entre ellos está brindar confidencialidad y privacidad al cliente, debido a que, al no hacerlo, puede llevar a una pérdida de confianza en los consumidores y clientes. A su vez, también ayuda a contar con una mejor gobernanza de datos, además de hacer más eficientes las operaciones como resultado, esto debido a la implementación de políticas más estrictas en el ciclo de vida de los datos en una organización (Davis, 2024).

Al momento de realización de este informe, Guatemala no cuenta con una ley que vele por la protección de datos y la privacidad de estos, únicamente existe la iniciativa de ley 6572, la cual se basa en principios y mejores prácticas reconocidas por la comunidad internacional, como lo es el Reglamento General de Protección de Datos de la Unión Europea, adicionalmente, considerando que es imprescindible establecer un marco normativo que asegure el tratamiento legítimo, proporcional, informado y seguro de los datos personales, garantizando la responsabilidad de quienes recaban, almacenan y procesan esta información (Congreso de la República de Guatemala, 2025).

Por su parte, el Reglamento General de Protección de Datos de la Unión Europea (GDPR) es un reglamento el cual el Parlamento Europeo, el Consejo de la Unión Europea y la Comisión Europea tienen la intención de reforzar y unificar la protección de datos de todos los individuos dentro de la Unión Europea, incluyendo la exportación de estos fuera de la UE. El GDPR cuenta con el objetivo de dar control a sus residentes y ciudadanos sobre sus datos personales, debido al uso que las empresas le dan a sus datos tanto para intercambio de información, como para el uso de sus servicios. Además de darle un entorno jurídico más simple a las empresas para realizar sus operaciones, haciendo que la ley de protección de datos sea idéntica en todo el mercado (PowerData, 2025).

4.2. Áreas de la ciberseguridad implementadas en el sistema web

4.2.1. Seguridad en el ciclo de vida del software

La seguridad en el ciclo de vida del software o *SDLC* implica la integración de prácticas de seguridad en cada etapa del proceso de desarrollo. Incluye servicios de integración y mitigación de riesgos, implementación de controles de seguridad, y realización de pruebas para garantizar que el software sea seguro desde etapas tempranas hasta su despliegue en producción (Xyleni, 2025).

Las fases que comprenden la seguridad en el ciclo de vida del software son las siguientes:

- Planificación y análisis: establece las metas y el alcance del proyecto en el desarrollo del software. Esta fase incluye problemas y casos de uso a considerar y cómo resolverlos, además de cómo serán las interacciones con los usuarios, sistemas y aplicaciones. Esto con el objetivo de que el equipo de desarrollo entienda las metas a alcanzar y los riesgos que conlleva el proyecto (IBM, 2024).

- Diseño: esta fase desarrolla la arquitectura del sistema, incluyendo los flujos de usuario, funcionalidades del software, el control de acceso y mitigación de amenazas (IBM, 2024).

- Desarrollo: inicia cuando el equipo comienza a implementar código para el proyecto. En esta fase se encuentran los mayores riesgos para vulnerabilidades o componentes poco confiables para integrar el software (IBM, 2024).

- Pruebas: luego de haber codificado una parte funcional del software, procede la eliminación de bugs, análisis de vulnerabilidades, pentesting y de riesgos para asegurar la calidad del software y a su vez, incrementarla (IBM, 2024).

- Despliegue: al corregirse los problemas detectados para los hallazgos críticos y altos, se despliega el ambiente de producción, para que los usuarios accedan a él. Este paso involucra en desplegar el software por partes para usuarios de prueba, además de brindar toda la información y conocimiento para hacerlo, con tal que los usuarios finales posean toda la documentación necesaria (IBM, 2024).

- Mantenimiento: el software puede necesitar algunos ajustes y nuevos flujos de usuario una vez ya ha sido desplegado, incluyendo actualizaciones y optimizaciones. Por lo que es necesario brindar soporte y mantenimiento para asegurar la continuidad y longevidad del software. Para ello, se emplean modelos de desarrollo, como *DevOps*, *DevSecOps* e integración y despliegue continuos (*CI/CD*) (IBM, 2024). Estos modelos de desarrollo se operacionalizan mediante *pipelines* que compilan, prueban, analizan y se despliegan de forma repetible.

4.2.2. Seguridad de aplicaciones

La seguridad de aplicaciones es un conjunto de prácticas recomendables, funciones y características añadidas al software de una empresa para ayudar a prevenir y resolver diversas amenazas que atenten contra la integridad del software. Esta se produce en varias etapas, teniendo un mayor enfoque en las fases de desarrollo de las aplicaciones (Nutanix, 2025).

En las plataformas web, la seguridad de aplicaciones se refiere a procesos, tecnologías y métodos para la protección de los servidores y servicios de las amenazas que suponen ataques basados en internet (F5, 2025a). Es recomendable adoptar las guías y controles de OWASP, comunidad global cuya misión es mejorar la seguridad en aplicaciones web para cumplir ciertos requerimientos y pruebas, y que también ofrece diversas herramientas, pautas y recursos para ayudar a las organizaciones a proteger sus aplicaciones web (García, 2025). Esto debido a que OWASP ayuda a reducir los riesgos y vulnerabilidades más comunes a nivel mundial, haciendo que la plataforma web cuente con mayor seguridad y protección respecto a vulnerabilidades frecuentes.

Adicionalmente, existe el marco MITRE ATT&CK, el cual es un grupo de tácticas organizadas por matrices (enterprise, pre-attack y mobile). Estas tácticas evaluan el riesgo de una organización y clasifican los ataques. Su objetivo es fortalecer los pasos a tomar luego de que una organización ha sido comprometida (Fortinet, 2025a). MITRE ATT&CK identifica y clasifica estas tácticas organizadas y empleadas por atacantes para facilitar el diseño de estrategias de defensa más efectivas contra ellas.

4.2.3. Seguridad de la información

Se refiere a la protección de la información en todas las formas, en reposo, en tránsito, formato electrónico o en papel, incluyendo también datos confidenciales (Infosecurity México, 2023). Incluye las herramientas y procesos necesarios para identificar, prevenir y corregir ataques contra todo tipo de información (Fortinet, 2025b).

4.2.4. Seguridad en la nube

La seguridad en la nube es la disciplina dedicada a la protección de sistemas informáticos alojados en la nube, incluyendo la protección de datos privados y seguros en la infraestructura, aplicaciones y plataformas en línea. Es el conjunto de tecnologías, herramientas y buenas prácticas que protegen entornos de computación en la nube y todo lo que contienen. La seguridad en la nube permite la recuperación de datos en caso de pérdida, protección de almacenamiento y redes contra el robo de datos y reducir el impacto de cualquier puesta en peligro del sistema (Kaspersky, 2025).

Algunos estándares para seguridad en la nube son propuestos por la Cloud Security Alliance (*CSA*), por medio de su programa *CSA Star*, el cual evalúa a los proveedores de servicios en la nube según cómo se adhieren a los principios de seguridad del *CSA* (Wiz Experts Team, 2024). También está el *CIS Benchmarks*, propuesto por el Center for Internet Security (*CIS*). Es un conjunto de estándares de ciberseguridad para implementar

servicios y tecnologías en la nube, siendo algunos de ellos los lineamientos para controles de acceso administrativo, autenticación y seguridad en contenedores y máquinas virtuales (Wiz Experts Team, 2024).

4.2.5. Seguridad de infraestructura

Es un proceso de protección a la infraestructura mediante la instalación de medidas preventivas para denegar un acceso no autorizado, modificación o robo de datos. Requiere de un enfoque holístico de los procesos y las prácticas en curso para garantizar una infraestructura protegida, proporcionando beneficios como ahorro en costos de producción, mejoras en productividad y protección de datos. Algunos tipos de seguridad de infraestructura de red son controles de acceso, seguridad de aplicaciones, *firewalls* y redes privadas virtuales (*VPN*) (VMWare, 2025).

La seguridad de infraestructura está en diversas áreas como lo son *containers* como Docker, Kubernetes (*K8S*) y también se concentra en el apartado de seguridad de plataformas. Esta última se encarga de proveer seguridad por medio de herramientas, procesos y arquitectura a una o varias plataformas computacionales (Broadcom, s. f.). Tanto la seguridad de infraestructura como también la seguridad de plataformas ofrecen como parte del ciclo de vida del desarrollo de software la capacidad de brindar protección a incidentes en la infraestructura que compone al software o al servicio, adaptabilidad y también en brindar soporte en la nube (Broadcom, s. f.).

4.2.6. Gestión de identidades y control de acceso

La gestión de identidades y accesos o (*IAM*) es una disciplina de ciberseguridad que se encarga de cómo los usuarios pueden acceder a recursos digitales y qué pueden o no hacer con dichos recursos. Estos ayudan a que los usuarios gocen de los permisos necesarios para realizar su trabajo alejados de cualquier tercero con intenciones maliciosas (Forrest y Kosinski, 2024). Algunas soluciones y servicios son proveedores de identidad (*IdP*), los cuales pueden utilizar estándares abiertos como Security Assertion Markup Language (*SAML*) y OpenID Connect (*OIDC*); herramientas de autenticación como autenticación de dos factores (*2FA*) y multifactores (*MFA*), e incluso métodos más avanzados como inicio de sesión único (*SSO*) (Forrest y Kosinski, 2024).

4.2.7. Escaneo de vulnerabilidades

Es una técnica en ciberseguridad que involucra la identificación y la evaluación de servicios y aplicaciones en una infraestructura, esto para hallar posibles puntos débiles en los sistemas de un entorno o red (Alcarria, 2024). Su objetivo es construir varias estrategias de ciberseguridad robustas y que permitan comprender y mitigar los riesgos que pueden existir en un sistema o red. Las herramientas de escaneo de vulnerabilidades en el ciclo de vida del desarrollo de software se enfocan en el análisis e informar acerca de ellas a medida de que el desarrollo avance. Y documentar los casos en donde las vulnerabilidades no sean mitigadas luego de desplegar el software en producción (Contrast Security, s. f.).

4.2.8. Encriptación en tránsito y en reposo

Los datos en tránsito son aquellos que se mueven activamente de una ubicación a otra dentro de una red o de una infraestructura. La protección o encriptación de estos datos ocurre mientras se transmiten a través de las redes, asegurándose que no puedan ser interceptados o alterados. Mientras que, los datos en reposo son aquellos que no se mueven activamente de un dispositivo o de un sistema. Garantizando que se mantengan seguros y confidenciales, aunque el equipo se vea comprometido (DigitalGuardian, 2023).

Para asegurar datos en tránsito se utiliza regularmente una implementación de protocolos de encriptación tales como Transport Layer Security (*TLS*), ya sea su versión 1.2 o 1.3 para encriptar paquetes basados en HTML. O también *Mutual TLS* (*mTLS*) que autentica ambas partes en la conexión de red, además de usar conexiones seguras con *VPN*, conexiones directas y otros servicios que protejan la infraestructura tanto local como en la nube para detectar algún tipo de anomalía (Popat, 2025).

Para asegurar datos en reposo se debe de asegurar la encriptación de los datos almacenados por medio de servicios o estándares tales como AES-256, el cual es el Advanced Encryption Standard. Sin embargo, también existen otros métodos como módulos de seguridad de hardware y encriptadores en proveedores de servicios en la nube. Se realizan implementaciones de controles de acceso, tanto basados en roles (*RBAC*), como gestión de identidades y accesos, además de proveer respaldo a la información mediante almacenamiento offline y planes de recuperación de la información, complementándose con monitoreo y auditoría para riesgos de seguridad (Popat, 2025).

Adicionalmente, existen otras maneras para encriptación de datos en reposo como Customer-Managed Encryption Keys (*CMEK*), el cual permite a los usuarios crear y manejar sus propias llaves de encriptación, y Customer-Supplied Encryption Keys (*CSEK*), que permite a los usuarios utilizar sus llaves de encriptación para encriptar datos en reposo (Bulbule, 2023).

4.3. Herramientas de seguridad investigadas para el sistema web

4.3.1. Autenticación y gestión de identidades

Auth0

Es una herramienta que gestiona el acceso de usuarios por medio de autenticación y autorización, este es un ID as a Service (*IDaaS*) y un proveedor de acceso e identidad (*IAM*). Es compatible con distintas aplicaciones y frameworks, permitiendo también la configuración de las autenticaciones (Okta, 2025b).

Para la aplicación web, puede aportar como una plataforma *IDaaS* por medio de flujos *OIDC* y OAuth 2.0, así como también políticas de seguridad, registros de auditoría, y también con centralización de autenticación, autorización y reducción de riesgos en credenciales.

OAuth 2.0

Open Authorization 2.0 es un estándar diseñado para permitir que tanto un sitio web como una aplicación puedan acceder a recursos alojados por otras aplicaciones web. Proporciona un acceso consentido y restringe las acciones que el cliente puede realizar en los recursos de la aplicación con un nombre de usuario, sin compartir sus credenciales (Okta, 2025a). Este estándar utiliza OpenID Connect (*OIDC*) para la identidad y contempla provisionamiento *just-in-time* (*JIT*) para reducir tareas manuales y errores en sincronización.

Este estándar es considerado gracias a su capacidad de delegación de accesos sin exponer contraseñas, algo fundamental en la viabilidad y seguridad de una arquitectura web moderna. Además de su integración mencionada con *OIDC* y soporte *just-in-time* para la automatización de accesos de usuarios.

Okta

Es un servicio de gestión de identidad a nivel empresarial compatible con la nube. Administra el acceso de cualquier usuario en cualquier aplicación o dispositivo con un servicio en ejecución en la nube de forma segura y confiable. Sus características incluyen *provisioning*, *single sign-on* (SSO), *active directory* (AD), autenticación multifactor (*MFA*) e integración con *LDAP* (Okta, 2025c). Puede aportar catálogos de integraciones y auditoría centralizada por medio de su gobernanza, lo cual es útil para controles de cumplimiento y trazabilidad.

LDAP

El Lightweight Directory Access Protocol o *LDAP*, es un protocolo basado en estándares y en protocolos de *TCP* e *IP*, que permite a los clientes realizar varias operaciones en un servidor, incluyendo consultar y almacenar datos, como también la autenticación de clientes (LDAP, s. f.). *LDAP* cuenta con integraciones con aplicaciones locales y sincronización con proveedores de identidades modernos, lo cual es útil en diversos tipos de arquitecturas web.

SAML

SAML, o el lenguaje de marcado de confirmación de seguridad es un protocolo el cual permite a un proveedor de identidad utilizar y enviar las credenciales de un usuario hacia un proveedor de servicios, con el objetivo de autorizar y autenticar al usuario para su acceso. Utiliza XML para estandarizar comunicaciones entre varios sistemas, y es capaz de administrar las contraseñas e identidades asociadas a los empleados y clientes (Fortinet, 2025c). Este protocolo es bastante utilizado en organizaciones corporativas debido a su *SSO*, reducción de contraseñas locales y trazabilidad del inicio de sesión para auditoría.

Google Identity

Es un servicio de autenticación segura para usuarios por medio del uso de cuentas de Google. Es útil para administración y verificación de credenciales, y también ayuda a compartir datos con otros servicios y dispositivos de Google en la aplicación (Google, s. f.). Es relevante su mención al ser un servicio con permisos delegados a APIs de Google y contar con una experiencia de usuario agradable para adopción rápida en una aplicación web.

Microsoft Entra

Es un servicio de gestión de identidades en la nube ofrecido por Microsoft. Anteriormente era conocido como Azure Active Directory. Permite a las organizaciones implementar una estrategia de arquitectura Zero Trust para la verificación de identidades, validación de acceso, revisión de permisos y encriptación de canales y conexiones (Microsoft, 2025). Microsoft Entra aporta con flujos *OIDC* y *OAuth2*, acceso por roles, registro de eventos y la integración con aplicaciones y *pipelines* externos.

4.3.2. Encriptación y protección de datos

Bcrypt

Bcrypt es una librería para hasheo de contraseñas y puede utilizarse en varios entornos de desarrollo. Su funcionamiento se basa en recibir contraseñas creadas por el usuario, generando una cadena de caracteres aleatorios y siendo combinada con la contraseña, produciendo el hasheo de la contraseña al concatenarse e iterar el algoritmo de hasheo para proteger la contraseña, dependiendo del valor de costo que se emplee, siendo posteriormente almacenada en la base de datos (Cooper, 2023). La librería Bcrypt cuenta con compatibilidad multi-lenguaje y una protección muy resistente a ataques de fuerza bruta hacia las contraseñas almacenadas, lo cual ayuda a disminuir el impacto de una filtración de bases de datos.

OpenSSL

OpenSSL es una librería de criptografía de código abierto que, permite implementaciones con protocolos de seguridad de *TLS* y *SSL*. Es ampliamente conocido por implementar conexión segura y cifrados en varias aplicaciones, como servidores web, correo electrónico, *VPN*, entre otros. Algunos casos de uso que se le atribuyen a OpenSSL son la comunicación segura entre cliente y servidor, cifrado, integridad y autenticación de los datos, y la alta flexibilidad que posee (F5, 2025b). OpenSSL puede aportar a la aplicación web con canales cifrados por *TLS*, validación de certificados dentro de una API y seguridad de transporte estricta HTTP (*HSTS*).

4.3.3. Escaneo de vulnerabilidades y seguridad del código

Snyk

Snyk es una plataforma basada en inteligencia artificial enfocada en el desarrollo seguro de software. Cuenta con diversas plataformas y productos para ofrecer una cobertura más amplia, técnicas de prevención y supervisión del software mediante técnicas de ciberseguridad y de IA. Entre sus usos se encuentra brindar código seguro por IA, junto a su desarrollo y manutención, mitigación de riesgos y escaneo de vulnerabilidades (Snyk Limited, 2025). En un proyecto como puede ser una plataforma web, Snyk aporta con alertas por severidad, *pull requests* de remediación de dependencias en un repositorio, y también políticas de visibilidad de riesgo de terceros en la cadena de software.

SonarQube/SonarCloud

Es una plataforma de código abierto para la inspección continua de código en diversas aplicaciones por medio de análisis de código estático. Es esencial para la fase de *testing* y auditoría dentro del ciclo de desarrollo de software y es perfecta para guiar al equipo de desarrollo durante dichas fases. Es capaz de interpretar código de 29 lenguajes de programación bastante utilizados e importantes, como Python, Java, JavaScript, TypeScript, C/C++, entre otros (Sentrio, 2021). SonarQube y también SonarCloud, ayudan con detección de *bugs*, *code smells*, puntos críticos de seguridad (*security hotspots*) y reportes reproducibles que soportan el aseguramiento de calidad.

TruffleHog

TruffleHog es una herramienta open-source o de código abierto que detecta y resuelve *secrets* expuestos en el *stack* tecnológico del sistema a través de todo el ciclo de vida del software (Truffle Security Co., 2025). TruffleHog funciona con cuatro objetivos principales: Detectar en repositorios de código *secrets* desapercibidos en comentarios, imágenes de Docker, historial de versiones, etc; analizar recursos y permisos con claves de acceso en *APIs*, prevención de filtrado de claves e inclusión de datos sensibles antes de realizar cambios, y remediar estados de distintas claves y *secrets* para verificación (Truffle Security Co., 2025). Todos estos objetivos van relacionados por un análisis de múltiples ramas dentro de un mismo repositorio y verificación de credenciales, con tal de proveer un nivel de seguridad consistente en el proyecto y evitar falsos positivos (Truffle Security Co., 2025).

4.3.4. Pruebas de penetración y auditoría

OWASP ZAP

Es una herramienta *open-source* de *pentesting*, diseñada para realizar pruebas en aplicaciones web y que se basa en contar con comunicación entre el navegador del *tester* y la aplicación web para la inspección de mensajes y de paquetes. Cuenta con versiones en varios

sistemas operativos y en Docker (ZAP, 2025). ZAP realiza escaneos automatizados en integración continua, hallazgos clasificables por severidad, creación de reportes HTML y XML con los resultados obtenidos y seguimiento de regresiones de seguridad, haciendo de esta una herramienta útil para escaneo de vulnerabilidades dentro de un sistema web.

BurpSuite

Es una herramienta para pruebas de ciberseguridad en sistemas web. Funciona como un interceptor en el proxy, capturando, analizando y manipulando tráfico *HTTPS* y *WebSockets*, además de incluir escaneo automático, elaboración de reportes y *crawling* de los sitios web (Covic, 2023). Posee una versión gratis y versiones de paga, tanto profesional como empresarial, al igual que cuenta con herramientas de explotación controlada, reportes formales y validación de casos específicos para las aplicaciones.

Nmap

Es una herramienta de código abierto para la exploración de redes y auditoría de seguridad, contando con descubrimiento de *hosts*, detección de servicios del sistema operativo, escaneo de puertos y motores de *scripts* para detección avanzada de vulnerabilidades en la red o servidor (Nmap, s. f.). Nmap es útil para hacer inventario de puertos y servicios, detección de versiones de sistemas operativos y *scripts* para verificar configuraciones de un equipo.

Metasploit

Permite hacer un escaneo, detección y explotación de vulnerabilidades dentro de un sistema operativo, red, servidor o servicio. Posee una base de datos de *exploits*, herramientas auxiliares y *payloads* como Meterpreter para postexplotación (Rapid7, s. f.). Cuenta con módulos de *exploits* y *payloads*, al igual que pruebas de postexplotación para evidenciar el riesgo real para priorizar mitigaciones de vulnerabilidades dentro del sistema web.

4.4. Aseguramiento de calidad en proyectos web

4.4.1. Principios de aseguramiento de calidad

El aseguramiento de la calidad o *QA* es un proceso de control y de evaluación de un proyecto, servicio o sistema para garantizar que los estándares de calidad establecidos se cumplan debidamente. Se refiere a todas las actividades que garanticen un nivel de calidad en los servicios, incluyendo diseño, codificación, revisión y pruebas (ACL, 2024). Su importancia radica en la calidad del producto para la prevención de defectos, errores o *bugs* durante su desarrollo, en que el software cumplirá con los requerimientos y estándares del cliente, en la satisfacción del usuario para su usabilidad y la confiabilidad de que el software podrá operar en varias condiciones, y también en el éxito del negocio para la lealtad del cliente al

estar satisfecho con el servicio, la reputación de la marca y también la reducción de costos (Murali, 2024).

4.4.2. Objetivos de aseguramiento de calidad

Entre las principales funciones y objetivos de *QA* se encuentran:

- Planificar y mantener el cumplimiento de la calidad del producto en base a los estándares definidos (ComplianceQuest, 2025).
- Coordinar y asegurar que este se lleve a cabo en todas las fases del desarrollo de software, ya sea en *testing*, auditoría y reportería (ComplianceQuest, 2025).
- Revisar procedimientos y especificaciones de acuerdo con control de calidad (ComplianceQuest, 2025).
- Realizar auditorías técnicas para garantizar que los estándares se cumplan consistentemente (ComplianceQuest, 2025).

4.4.3. Automatización de pruebas

Es el proceso de automatizar el *testing* de aplicaciones de software con el objetivo de demostrar su funcionamiento y si estas cumplen con los requerimientos específicos establecidos. Este proceso ayuda a mejorar la eficiencia y reducir esfuerzo manual de los *testers* y desarrolladores, permitiendo a los equipos de desarrollo enfocarse en tareas más complejas mientras que las tareas repetitivas se hacen de forma automática (Szahidewicz, 2025).

El enfoque de la automatización de pruebas en el ciclo de vida del desarrollo de software se da por medio del análisis para validación de requerimientos, en las fases de diseño y desarrollo, al realizarse pruebas que verifiquen el funcionamiento del código apenas este es implementado. En la fase de pruebas para el monitoreo, manejo y optimización de pruebas para una mayor eficiencia y cobertura. Así como también para las fases de despliegue y mantenimiento, para observar que el sistema funcione bien sin inconvenientes (Szahidewicz, 2025).

4.4.4. Control de versiones y revisiones de código

El control de versiones o control de revisiones es una práctica de desarrollo de software para hacer un seguimiento y gestión de cambios realizados en código o en otros archivos dentro de un proyecto. Con ella, se hace un seguimiento de cada cambio que se ha realizado en el código base. Permitiendo a los desarrolladores tener un historial y tener posibilidad de retroceso a una versión anterior. Siendo útil en temas de seguridad para protección de daños irreparables, y en materia de aseguramiento de calidad para brindar documentación del progreso en cada una de las fases del desarrollo (GitLab, 2025).

Entre los tipos de sistemas de control de versiones más populares se encuentran el distribuido, que permite a los usuarios acceder a un repositorio desde distintas ubicaciones. Y el centralizado, en donde todos los desarrolladores trabajan con el mismo repositorio central y se encuentra en un servidor o máquina local (GitLab, 2025).

4.4.5. *CI/CD*

La integración continua (*CI*), es una práctica de desarrollo en la cual los desarrolladores integran código a un repositorio en conjunto de manera constante, con cada integración contando con sus respectivas pruebas para detección rápida de errores. Esto ayuda al código base a mantener su salud y estabilidad, asegurándose de que el software siempre se encuentre en un estado disponible (Compunnel, s. f.).

La entrega continua (*CD*) toma en cuenta los principios de *CI*. Así como también, cada cambio que logre pasar las pruebas automatizadas es desplegado directamente al ambiente de producción. Entregando actualizaciones de software a los usuarios de forma rápida y frecuente (Compunnel, s. f.).

Ambas prácticas son complementarias entre sí, *CI* se enfoca en las fases de desarrollo y de pruebas. Mientras *CD* trata de brindar actualizaciones de manera rápida a los clientes, siendo este framework muy importante para el aseguramiento de calidad (Compunnel, s. f.).

4.4.6. Pipelines de validación

Un pipeline de validación es un proceso estructurado en el ciclo de vida de desarrollo de software, el cual automatiza flujos de trabajo de pruebas para asegurar la calidad, confiabilidad, y funcionalidad del código antes de ser enviado a producción. Integra varias fases de *CI/CD* para detectar *bugs* de forma temprana en el desarrollo y mantiene la salud del código (Kansara, 2025).

Además de validar funcionalidad y calidad, los *pipelines* incorporan controles automáticos de seguridad tales como el escaneo de dependencias con Snyk, detección de secretos con TruffleHog y también análisis de código estático y dinámico en etapas tempranas del desarrollo. Pueden bloquear integraciones al detectarse vulnerabilidades altas o críticas, así como también fugas de secretos, para registrar evidencia reproducible como reportes y métricas para auditoría continua.

4.4.7. Pruebas continuas

Son unas de las mejores prácticas y de las más importantes en *QA*. Involucran retroalimentación automatizada a través del ciclo de vida del desarrollo de software. Además, ofrecen diversas ventajas, como la detección rápida de errores, la experiencia de usuario y la reducción de costos asociados a los errores durante la implementación (Testlio, 2024). En la actualidad, este tipo de pruebas se aplican por su retroalimentación en fases tempranas del desarrollo, soporte a ciclos de entrega rápidos, y se utilizan para *gates* automáticos en

flujos de *CI/CD*, entornos efímeros para *PRs* y métricas operativas.

4.4.8. Pruebas de integración

Las pruebas de integración o *integration testing* son una parte importante en el ciclo de vida de desarrollo de software. Verifican la interacción entre los componentes, módulos y servicios dentro de un sistema, además de ser especialmente críticas y fundamentales en aplicaciones web distribuidas y complejas (Khan y Singh, 2012). Se aplican en la industria por medio de servicios reales acotados o contenedores por prueba, para idempotencia y limpieza, observabilidad en pruebas y ejecución en integración continua.

4.4.9. Pruebas *end-to-end*

Las pruebas *end-to-end* simulan flujos reales de interfaz de usuario y en una base de datos o servicio. Su implementación se da en pipelines *CI/CD* con enfoques robustos. Entre sus principios se encuentran el determinismo, el aislamiento y la mantenibilidad entre sus pruebas para asegurar resultados confiables (Runcheva, 2022). Se aplican para selección de rutas críticas, diseño mantenible por medio de pruebas robustas e integración a *CI/CD*.

4.5. Herramientas de aseguramiento de calidad investigadas en el sistema web

4.5.1. *Testing* automatizado y pruebas

Jest

Framework para pruebas en JavaScript, con soporte para React, TypeScript, Vue, entre otros. Es ideal para pruebas unitarias y de integración. Jest destaca por su facilidad de configuración, concurrencia en sus pruebas, y su velocidad de ejecución (OpenJS Foundation, s. f.). Jest es útil debido a su *feedback* veloz, *mocks* o *snapshots* del código a testear y por su cobertura estable para prevenir regresiones en el *frontend*.

Pytest

Pytest es un *framework* de Python especializado en casos de prueba para *APIs*, así como también para casos de pruebas simples y complejos en diversos programas. Pytest además de utilizarse en *APIs*, permite escribir código para pruebas en interfaces de usuario y bases de datos (GeeksForGeeks, 2025a). Aporta parametrización y *plugins* mediante cobertura y pruebas unitarias que simplifican pruebas reproducibles en *backend*.

Mocha

Framework flexible para pruebas en JavaScript y en TypeScript, el cual también es compatible con Node.js y *frontend*. Mocha permite definir *suites*, *hooks*, y el uso de librerías de aserciones (GeeksForGeeks, 2025b). Posee para el *frontend* de una aplicación web diversas integraciones con múltiples librerías de aserciones y adaptación a necesidades específicas dentro del código.

Cypress

Cypress es una herramienta para pruebas *end-to-end* en aplicaciones web. Realiza y ejecuta las pruebas directamente desde el navegador, simulando interacción real con el usuario, brindando depuración en tiempo real y facilitando integración y procesos de *CI/CD* (Cypress, s. f.). Es capaz de realizar la ejecución real de flujos críticos, al igual que proporciona videos y capturas de pantalla durante esta, mientras posee una fácil integración en *CI*.

Enzyme

Es una biblioteca de pruebas de JavaScript desarrollada por Airbnb para hacer testing en componentes React. Con ella, es posible montar y renderizar varios componentes, así como también acceder a su estado y a sus propiedades, simulando eventos y recorridos del *DOM*, con ayuda de varios modos de renderizado para el ajuste del detalle de las pruebas (Enzymejs, s. f.).

Selenium

Es un conjunto de herramientas de código abierto para la automatización de web browsers, el cual es principalmente utilizado para pruebas funcionales de aplicaciones web. Es compatible con Windows, MacOS y Linux, como también permite pruebas cruzadas en múltiples navegadores y entornos (BrowserStack, 2025b).

4.5.2. Entornos de prueba y compatibilidad

BrowserStack

Plataforma en la nube para realizar pruebas de compatibilidad cruzada en más de 3500 navegadores, dispositivos móviles y sistemas operativos. Incluye pruebas automatizadas como también pruebas manuales junto a su visualización en tiempo real, depuración y captura de errores sin necesidad de emuladores o dispositivos físicos (BrowserStack, 2025a).

LambdaTest

Plataforma de pruebas automatizadas y manuales basada en la nube, la cual admite más de 3000 plataformas móviles y sistemas operativos. Proporciona una vista holística de actividades de pruebas automatizadas, desplegando métricas clave y enfoque en las anomalías y fallos en las pruebas (Saini, 2025).

4.5.3. Automatización y despliegue

Jenkins

Es un servidor de automatización open source escrito en Java para la integración de *CI/CD*. Permite configurar *pipelines* personalizados y cuenta con una amplia compatibilidad de *plugins*, permitiendo también dirigir el flujo del ciclo de vida del software (Kulkarni, 2023).

GitHub Workflows

Integración de GitHub que permite definir flujos de trabajo automatizados mediante archivos *YAML*. GitHub Workflows soporta CI/CD, pruebas automatizadas, despliegue y diversas tareas personalizadas. Este se puede utilizar en *runners* de GitHub o en *runners* alojados (GitHub, 2025).

Terraform

Terraform es una herramienta de infraestructura como código (*IaC*), que tiene como función definir y versionar recursos de infraestructura mediante archivos de configuración declarativa. Esto facilita la creación, modificación y la destrucción segura de infraestructura con ayuda del control de versiones (HashiCorp, s. f.).

4.5.4. Contenedorización y orquestación

Docker

Plataforma *open source* en la cual se empaquetan aplicaciones junto a sus dependencias en contenedores ligeros e independientes. Esta permite ejecutar el software en cualquier entorno sin ningún problema de compatibilidad, asegurando así portabilidad, consistencia y eficiencia (Docker Inc., 2025).

Kubernetes

Kubernetes (*K8S*) es un sistema *open source* para la orquestación de contenedores el cual automatiza el despliegue, escalado, balanceo de carga y gestión de aplicaciones contenedo-

rizadas. Permite y facilita la configuración declarativa y alta disponibilidad del contenedor y de la aplicación (The Kubernetes Authors, 2024).

Rancher

Plataforma de gestión de clústeres de Kubernetes la cual simplifica la gestión multi-clúster, la seguridad, el despliegue de aplicaciones y la configuración del acceso. Rancher ayuda al usuario proporcionando una interfaz visual y herramientas avanzadas para *edge computing* y entornos empresariales (Shwartz, 2023).

4.5.5. Seguridad y conectividad en pruebas

WireGuard

Es un protocolo *VPN* moderno, robusto y minimalista que se implementó originalmente en Linux y se extendió a múltiples plataformas. Reduce la superficie de ataque y con ello ofrece un rendimiento superior gracias a sus cifrados y *roaming* entre redes (WireGuard, 2022).

Tailscale

Tailscale es una solución de *VPN* que utiliza el protocolo Wireguard para establecer conexiones punto a punto cifradas entre dispositivos sin necesidad de una infraestructura tradicional de *VPN*. Es multiplataforma, ofrece un control de acceso basado en identidad, *SSH* y conexión automática entre clústers o contenedores sin exponer puertos (Tailscale, 2025).

Content Security Policy

Es una política declarada mediante cabeceras HTTP que define un conjunto de directivas para limitar qué recursos puede cargar o ejecutar un navegador en una página web. Sirve como un control para mitigar ataques *XSS*, al restringir las fuentes en las que el navegador carga *scripts* y recursos, inyecciones de código, *clickjacking*, al evitar el *framing* de páginas. Evita que la aplicación sea embedida maliciosamente para la inducción de clicks no intencionados, entre otros tipos de ataques (Foundeo Inc., 2023).

Cloudflare

Cloudflare es un proveedor de servicios de infraestructura de seguridad web el cual ofrece una diversa cantidad de productos diseñados para proveer seguridad y un mejor rendimiento a un sitio web. Algunos beneficios que trae utilizar Cloudflare son los de entrega de conte-

nidos, *firewall* de aplicaciones web, además de servicios de Domain Name Services (*DNS*) y certificados *SSL/TLS* gratuitos (Wnpower, 2024).

4.6. Arquitectura y seguridad del sistema web

4.6.1. Principios de arquitectura *Zero Trust*

Zero Trust (*ZT*) es un término para un modelo de "nunca confiar, siempre verificar" dentro del paradigma de la ciberseguridad, el cual cuenta con defensas desde análisis estático, hasta recursos de red para usuarios, módulos y recursos. Asume que no hay confianza garantizada ni implícita hacia los usuarios basada únicamente en su ubicación física o de red. Una arquitectura *Zero Trust* se enfoca en proteger a los recursos, no a la segmentación de red (Rose et al., 2020).

4.6.2. Infraestructura como código y reproducibilidad

Infraestructura como código (*IaC*) es una metodología que permite a la infraestructura ser manejada y resguardada vía código y herramientas de automatización. Involucra el proceso de escribir código que describa y resguarde los recursos de infraestructura como pueden ser máquinas virtuales, redes, almacenamiento y datos, entre otros componentes (Gordon, 2025).

Los beneficios que esta tiene son la consistencia en ambientes de infraestructura, la reproducibilidad de estos, la escalabilidad y flexibilidad que el código de infraestructura ofrece para una organización, y la automatización que puede ofrecer (Gordon, 2025).

4.6.3. Seguridad por diseño y defensa en profundidad

La seguridad por diseño integra seguridad en cada aspecto en un sistema o en un servicio, incluyendo el manejo de riesgos, principios del mínimo privilegio, buenas prácticas de código, pruebas de seguridad y planificación de respuestas ante incidentes (Cyber Tzar, 2023).

La defensa en profundidad por su parte, indica que la seguridad debe implementarse en múltiples capas de protección, confiando en que, si una capa no puede con la amenaza, la siguiente podrá. Esta contempla la seguridad de la información, seguridad de red e infraestructura, seguridad en aplicaciones, y seguridad de los datos (Cyber Tzar, 2023).

4.7. Buenas prácticas de estándares y certificaciones seguidas en el sistema web

4.7.1. ISO/IEC 27001

Es la norma más conocida del mundo en materia de ciberseguridad y para sistemas de gestión de seguridad de la información. Define los requisitos que debe cumplir un sistema. Proporciona a empresas de cualquier tamaño y todos los sectores las orientaciones para establecer, mejorar y mantener de manera continua un sistema de gestión de la seguridad de la información (International Organization for Standardization, 2022). La norma ISO/IEC 27001 ayuda a las organizaciones a ser conscientes de los riesgos de ciberseguridad y saber gestionarlos.

4.7.2. ISO/IEC 25010

Representa el modelo de la calidad para establecer un sistema para la evaluación de la calidad de un producto. La norma define nueve características como lo son la adecuación funcional, eficiencia de desempeño, compatibilidad, fiabilidad, seguridad, mantenibilidad, flexibilidad y protección (International Organization for Standardization, 2023).

4.7.3. *NIST Cybersecurity Framework*

Provee apoyo y guía a la industria, a agencias del gobierno, y a demás organizaciones a lidiar con riesgos de ciberseguridad. Ofrece taxonomía de diversas técnicas e información de alto nivel en materia de ciberseguridad para entender, priorizar y asesorar de mejor manera dichos conceptos. Las funciones principales de *NIST CSF* son las de gobernar, identificar, proteger, detectar, responder y recuperar (National Institute of Standards and Technology, 2024).

4.7.4. OWASP Top 10

Es un documento el cual lista las diez amenazas más potentes y críticas para las aplicaciones web. Sirve como referencia para pruebas de seguridad y políticas de desarrollo seguro (OWASP Foundation, 2025).

4.7.5. Center for Internet Security

CIS es una comunidad sin fines de lucro la cual es globalmente reconocida por promover las mejores prácticas para seguridad en sistemas y en datos. Ellos son responsables por *CIS Benchmarks*, el cual ayuda a resguardar sistemas, software, y redes ante ciberataques y de *CIS Controls*, que es un conjunto de varias recomendaciones de defensa y seguridad en organizaciones (Center for Internet Security, 2025).

4.7.6. MITRE ATT&CK

MITRE ATT&CK es una base de tácticas y técnicas basadas en observaciones reales de acceso global. Estas tácticas se organizan en las matrices enterprise, pre-attack y mobile y se utilizan para la evaluación de riesgos en una organización y así clasificar los ataques cibernéticos (Fortinet, 2025a). Son utilizadas para el desarrollo de modelos y metodologías específicas para la ciberseguridad en diversos sectores laborales y de la industria (The MITRE Corporation, 2025).

CAPÍTULO 5

Metodología

5.1. Selección de herramientas de seguridad

El proceso de selección de las herramientas de seguridad se basó en la investigación de diversas fuentes con relación a herramientas, protocolos y *frameworks* de ciberseguridad que velen por todo el sistema web en conjunto a las capacidades y funcionalidades que este posee; y también en la comparación de estas mismas basadas en las necesidades del Ingenio Santa Ana y de las propias funcionalidades de la plataforma. Todo esto valorado en un *framework* de riesgo en el que se consideraron como necesidades la identificación, priorización y mitigación de dichos riesgos para alcanzar los objetivos propuestos. Este proceso fue dividido en categorías de autenticación y gestión de identidades, encriptación y protección de datos, escaneo de vulnerabilidades, seguridad del código y pruebas de penetración y auditoría, y, por último, seguridad en infraestructura y contenedores.

- Autenticación y gestión de identidades: se decidió utilizar el estándar de OAuth 2.0 únicamente. Esto debido a factores relacionados con los servicios del ingenio a quien se le realizará la plataforma web, como lo son la arquitectura de servicios que se utiliza administrativamente, siendo una opción Microsoft Entra como un *IdP* ideal como solución, pero el cual cuenta con tres opciones de planes para cotización de servicios, siendo el P1, P2 y Suite. Estos a su vez, cuentan con una versión gratuita en la cual, por un mes en los planes P1 y P2, se pueden contar con los servicios de Entra ID, mientras que, con la prueba gratuita de 90 días de Microsoft Entra Suite, se cuenta con todo el paquete de aplicaciones de Microsoft Entra. Por lo cual, al utilizar OAuth 2.0 como su estándar base con una capa de identidad *OIDC*, se acopla perfectamente a las necesidades de la plataforma, como un autenticador que pueda verificar de manera constante a los usuarios que ingresen a esta.

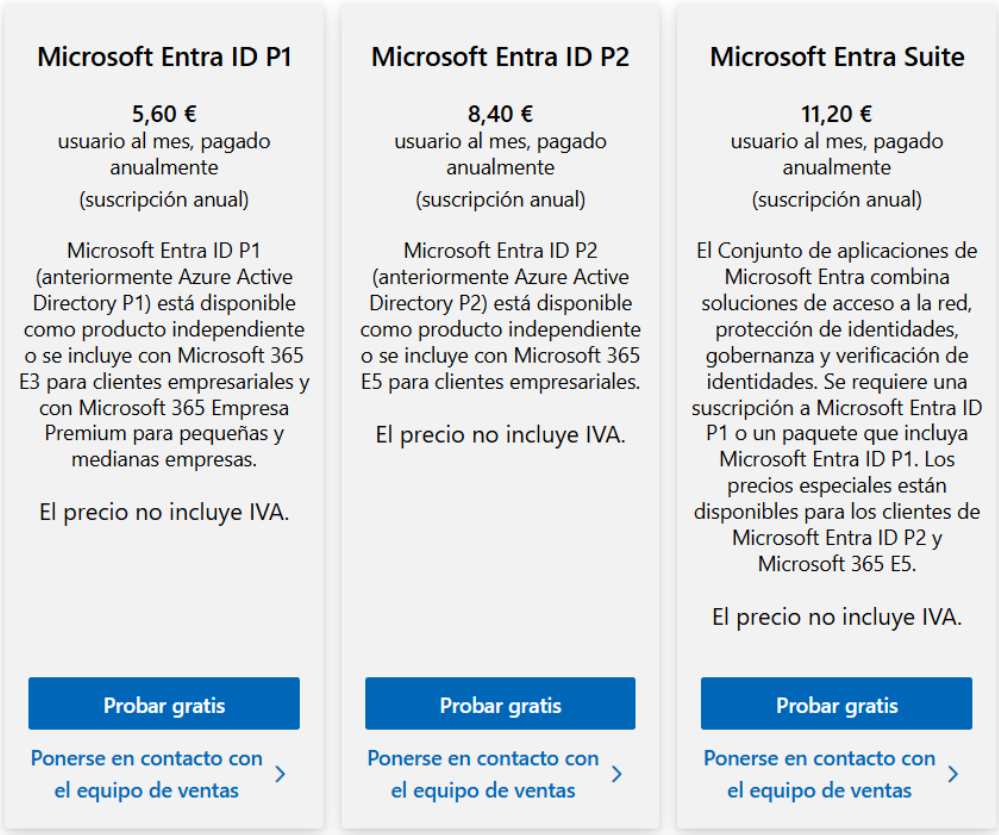


Figura 1: Cotización de planes de Microsoft Entra

Por el contrario, se decidió omitir las demás herramientas como Google Identity, debido a que el *tenant* o inquilino que se desea utilizar no estará alojado en Google Workspace. Auth0 y Okta podrían añadir complejidad innecesaria, considerando que el workforce estará en la arquitectura de Microsoft, y por parte de LDAP y SAML, no sería una buena práctica exponer LDAP para un login web en internet debido a las implicaciones de seguridad que podría tener, y al contar Entra con OIDC, SAML no sería totalmente necesario para lo que se pretende realizar con la autenticación.

- Encriptación y protección de datos: para la encriptación y protección de datos de la plataforma, se utilizaron los servicios de Render por medio de un modelo de terminación TLS gestionado por el proveedor para la encriptación en tránsito de la información. Con esto, se ofrece que todo el tránsito de la plataforma sea con el protocolo HTTPS con HSTS y demás cifrados modernos, con la aplicación Django utilizada en el *backend* detrás del *proxy* confiando en los encabezados de protocolo seguro, ofreciendo una mayor brecha de confidencialidad e integridad al ser compatible con cualquier *reverse proxy* y servicio, tanto en Windows, MacOS y Linux. Adicionalmente, en el caso del guardado de contraseñas, se contó también con OAuth 2.0, el cual también se encargó de realizar *password hashing* al ingresar, modificar y validar las credenciales a la base de datos, por lo que el uso de Bcrypt no fue necesario.
- Escaneo de vulnerabilidades, seguridad del código y pruebas de penetración y auditoría: para este apartado, se decidió incluir los *frameworks* SonarCloud, Snyk y OWASP

ZAP y la herramienta TruffleHog. Al decidirse por estas cuatro herramientas para brindar seguridad al código en conjunto a la detección de vulnerabilidades, se cubre con SAST o las pruebas estáticas de código, con las cuales se verifica la calidad y seguridad de este. Las DAST o pruebas dinámicas de código, en las cuales se suscitan cuando la plataforma está ejecutándose. Y SCA o análisis de composición del software para las dependencias que puedan presentar vulnerabilidades. Además de la utilización de TruffleHog para la detección de *secrets* en todo el sistema, incluyendo repositorios de GitHub y en el código.

El uso de ZAP cuenta con ventajas tales como contar con una ejecución bastante rápida, y su cobertura se basa meramente en web con una API en *runtime*. SonarQube Cloud ayuda con la capacidad de interpretar varios lenguajes de programación, tales como TypeScript y Python, que son los lenguajes utilizados para el desarrollo, teniendo así la capacidad de añadir reglas de seguridad, *quality gates* y *security hotspots*, además de interactuar con los repositorios en GitHub desde GitHub Workflows, añadiendo así más *pipelines* hacia la seguridad de la plataforma. Por parte de Snyk, sus integraciones hacia el SCA, contenedores y las políticas dentro de las indicaciones de infraestructura como código, es bastante práctico, únicamente tomando en cuenta las consideraciones y precios que la plataforma maneja.

5.2. Selección de herramientas de aseguramiento de calidad

El proceso de selección de herramientas y métodos de aseguramiento de calidad se basó igualmente en una investigación de diversas fuentes y de bibliografía de *frameworks*, plataformas y herramientas para velar el aseguramiento de calidad durante las fases de desarrollo y de pruebas, así como también en su fase de despliegue y velando que la plataforma cumpla con todos los requerimientos base con los que fue ideada y diseñada al inicio de este trabajo. Además de una evaluación de los estándares de calidad que se buscan cumplir dentro de todo el ciclo de vida del desarrollo de esta plataforma web por medio de la evaluación de los posibles riesgos que deben prevenirse y mitigarse y de las necesidades que se deben cumplir de acuerdo a los objetivos propuestos. El proceso se dividió en categorías de *testing* automatizado, entornos de prueba y compatibilidad, automatización y despliegue, contenedorización y orquestación, y, por último, seguridad y conectividad en pruebas.

- *Testing* automatizado y pruebas: Para realizar pruebas automatizadas según lo investigado, se decidió utilizar las librerías Jest y Pytest, y los *frameworks* de Mocha, Cypress y Selenium. Jest, en conjunto con React Testing Library, posee pruebas de componentes alineadas hacia el comportamiento del usuario, además de contar con el ecosistema idóneo para el ambiente de desarrollo en *frontend*. Pytest, para *backend*, cuenta con *fixtures* y parametrización idónea para las pruebas en el código.

  Por el lado de Mocha, Cypress y Selenium, Mocha presenta también pruebas para JavaScript y TypeScript, siendo bastante flexible con ello, aunque también, puede requerir de *plumbing* para la realización de pruebas. Cypress por su parte, posee un acercamiento a *DevEx* muy apropiado y excelente, ya que se alinea mucho hacia los conceptos de integración continua y para la realización de pruebas *end-to-end*. Se integra apropiadamente con Google Chrome y puede hacerlo también con Firefox y Safari.

Finalmente, con Selenium, brinda utilidad para compatibilidad extrema y capacidad multi-lenguaje, además de contar con integraciones en entornos de prueba como BrowserStack o en LambdaTest.

- Entornos de prueba y compatibilidad: para esta parte del trabajo, se decidió utilizar LambdaTest. El motivo de esta decisión es, si bien la mejor opción sería BrowserStack debido a la cantidad de cobertura que este *framework* posee, siendo un total de 3500 navegadores y dispositivos en los cuales se pueden realizar pruebas automatizadas y manuales, en la Figura 2 se puede observar que los planes de cotización del servicio para su implementación son bastante elevados, debido a que el plan más barato, el cual sería para únicamente escritorio o para automatización en navegadores sería de $99 anuales, es decir, unos Q758.31 anuales.

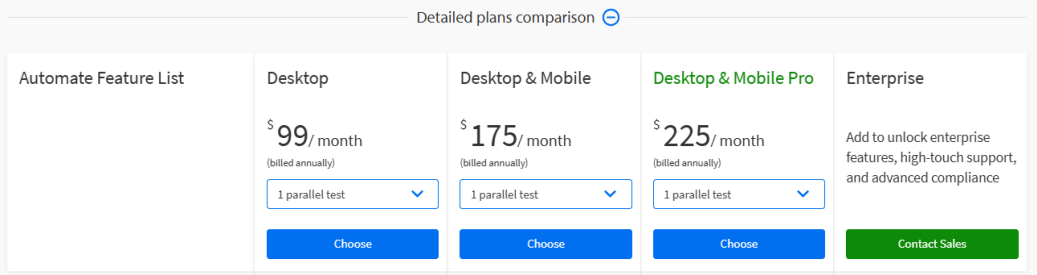


Figura 2: Planes de cotización de BrowserStack

Mientras que, por parte de LambdaTest, la implementación para pruebas automatizadas en navegador cuenta con un plan gratuito con 100 minutos para *testing* en plataformas web. Este además cuenta con integraciones para Selenium, Cypress y una cobertura para más de 3000 dispositivos, al igual que las mismas características que ofrece BrowserStack.

- Automatización y despliegue: se definieron tanto GitHub Workflows como Terraform para este apartado. GitHub Workflows es nativo a los repositorios alojados en GitHub, lo cual ayuda mucho a su implementación, además de contar con libre paralelización en sus servicios y una mayor facilidad para integrar sus herramientas. Mientras que, en el caso de Terraform, es bastante útil para la parte de infraestructura como código, y es capaz de ejecutarse en cualquier nube. Terraform fue escogida debido a la madurez del ecosistema, además de contar con amplio soporte en el proveedor de *Google Cloud Run*, proveedor el cual se utilizará para el despliegue. Se escoge por encima de herramientas como Terragrunt y OpenTofu ya que Terraform ofrece un balance apropiado entre mantenibilidad, soporte y velocidad de ejecución para los objetivos propuestos.

- Contenedorización y orquestación: se utilizaron los servicios de Docker como contenedor para la aplicación de *frontend*, la cual ya se encuentra vinculada a la *API* destinada para la plataforma. Esto para favorecer en la fase de despliegue, y en diversas alternativas para el lanzamiento de la plataforma web, además de ser algo bastante útil para el ingenio debido a cómo el Ingenio levanta sus propios servicios utilizando Docker y GitLab.

- Seguridad y conectividad en pruebas: para asegurar la seguridad y la conectividad en las pruebas, se decidió utilizar *Content Security Policy* y Cloudflare, esta última siendo

implementada debido a los servicios de Netlify y Google Cloud Run para el *frontend*, y Render para el *backend*, proveedores en donde fueron alojados respectivamente, al contar con la posibilidad de un *hosting* gratuito y de bajo coste por parte de ambas partes del proyecto. Cloudflare es empleado cuando se enruta un dominio o ruta propio hacia el *frontend* en Netlify y Google Cloud Run y/o al *backend* en Render, aportando así WAF, CAF y DNS administrado. Por parte de *Content Security Policy*, se decidió implementar desde el *frontend* y no por medio de Cloudflare con la intención de brindar control granular por ruta y el recurso desde el propio cliente para dar versionamiento junto con el código, así como también para dar mantenibilidad y trazabilidad al quedar sujeto a revisiones de código, pruebas automatizadas y auditorías, todo esto para demostrar compatibilidad con la plataforma al integrarse al proceso de *build*, evitando desalineaciones entre encabezados y recursos utilizados por la aplicación.

5.3. Implementación de la cadena *CI/CD* y controles automáticos

La implementación de los *pipelines* de integración y entrega continua, así como también de los controles automatizados en los dos repositorios utilizados contó con las siguientes herramientas:

- GitHub Workflows
- TruffleHog
- SonarCloud
- Snyk

Tal como se menciona en la sección 4.5.3, se definieron flujos de trabajo por medio de archivos *YAML* para automatizar los procesos de integración continua por medio de las pruebas automatizadas implementadas en diversos frameworks tanto para la plataforma web como para la *API* desarrolladas por los módulos designados para ello, y también para las herramientas dedicadas a la seguridad del código y detección de vulnerabilidades y secretos, como lo son las ya mencionadas TruffleHog, SonarCloud y Snyk. Adicionalmente, se realizaron flujos de trabajo para el despliegue y lanzamiento de los contenedores Docker del código realizado para la aplicación web y la *API*.

En total, se cuenta con ocho flujos realizados específicamente para el repositorio de GitHub destinado al desarrollo de la plataforma web, y cuatro flujos realizados para el repositorio de GitHub del desarrollo de la *API*, además del flujo de contenedorización realizado para este último por la encargada de dicho módulo.

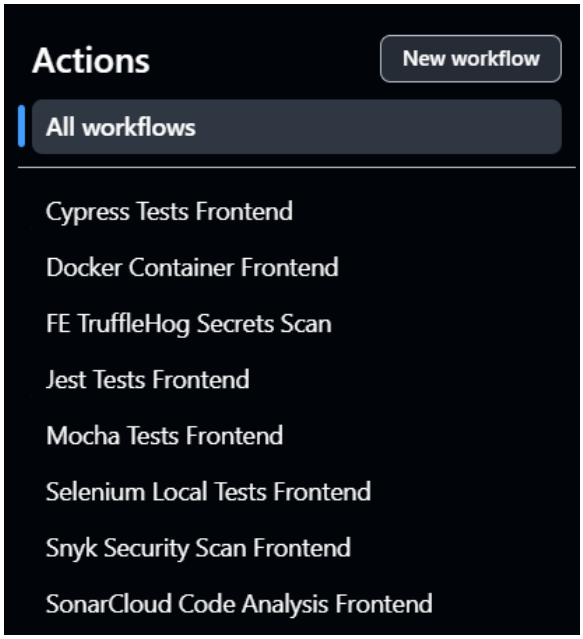


Figura 3: Nombres de flujos de trabajo implementados en el *frontend*

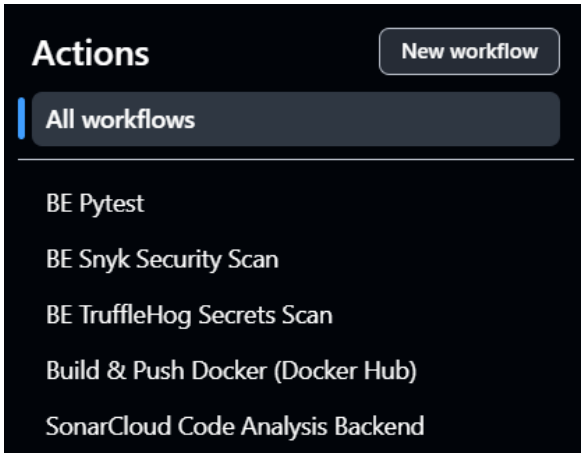


Figura 4: Nombres de flujos de trabajo implementados en el *backend*

Se definió la ejecución de estos flujos cada vez que se realice un *pull request* y un *push* hacia las ramas de desarrollo y principales, con el objetivo de velar y cumplir con los estándares de calidad definidos en el código que se aporta al desarrollo de este.

Con esta implementación, se optó por la implementación de TruffleHog como uno de estos flujos, debido a que durante el desarrollo de la plataforma web, se utilizaron diversas variables y secretos tanto en los repositorios como en el código del proyecto. Por lo que, para velar por la seguridad del código y de los repositorios, se utilizó esta herramienta para la detección y análisis de estas variables y secretos en todo el historial de *commits* y de archivos agregados al repositorio y en el caso afirmativo de hallarlos, proceder a informar las posibles soluciones para su prevención y remediación.

```
name: FE TruffleHog Secrets Scan

on:
  pull_request:
    branches: [ "main", "dev" ]
  push:
    branches: [ "dev" ]

jobs:
  trufflehog:
    runs-on: ubuntu-latest
    permissions:
      contents: read
    steps:
      - name: Checkout
        uses: actions/checkout@v4
        with:
          fetch-depth: 0  # Necesario para escanear todo el historial de git

      - name: TruffleHog
        uses: trufflesecurity/trufflehog@main
        with:
          path: ./
          extra_args: --only-verified --exclude-paths=.trufflehogignore

      - name: Exportar reporte JSON (filesystem)
        if: ${{ always() }}
        run: |
          curl -sSfL https://raw.githubusercontent.com/trufflesecurity/trufflehog/main/scripts/install.sh | sh -s -- -b /usr/local/bin
          if [ -f ".trufflehogignore" ]; then
            trufflehog filesystem --only-verified --json --exclude-paths=.trufflehogignore . > trufflehog_filesystem.json 2>&1 || true
          else
            trufflehog filesystem --only-verified --json . > trufflehog_filesystem.json 2>&1 || true
          fi
      - name: Exportar reporte JSON (git history)
        if: ${{ always() }}
        run: |
          if [ -f ".trufflehogignore" ]; then
            trufflehog git --only-verified --json --exclude-paths=.trufflehogignore . > trufflehog_git.json 2>&1 || true
          else
            trufflehog git --only-verified --json . > trufflehog_git.json 2>&1 || true
          fi
      - name: Subir artefactos
        if: ${{ always() }}
        uses: actions/upload-artifact@v4
```

```
        with:
          name: fe-trufflehog-reports
          path: |
            trufflehog_filesystem.json
            trufflehog_git.json
          retention-days: 30

      - name: Resumen de resultados
        if: ${{ always() }}
        run: |
          echo "### TruffleHog Scan Results" >> $GITHUB_STEP_SUMMARY
          echo "" >> $GITHUB_STEP_SUMMARY

          if [ -f "trufflehog_filesystem.json" ]; then
            FILESYSTEM_COUNT=$(grep -c "DetectorType" trufflehog_filesystem.json
            ↪  2>/dev/null || echo "0")
            echo "**Filesystem scan:** $FILESYSTEM_COUNT secretos verificados
            ↪  encontrados" >> $GITHUB_STEP_SUMMARY
          fi

          if [ -f "trufflehog_git.json" ]; then
            GIT_COUNT=$(grep -c "DetectorType" trufflehog_git.json 2>/dev/null ||
            ↪  echo "0")
            echo "**Git history scan:** $GIT_COUNT secretos verificados
            ↪  encontrados" >> $GITHUB_STEP_SUMMARY
          fi

          echo "" >> $GITHUB_STEP_SUMMARY
          echo "Los reportes completos están disponibles en los artifacts de este
          ↪  workflow." >> $GITHUB_STEP_SUMMARY
```

Figura 5: Archivo YAML del flujo de TruffleHog en *frontend*

```
    name: BE TruffleHog Secrets Scan

on:
  pull_request:
    branches: [ "main", "dev", "feat/SANTAANA-v1.0"]
  push:
    branches: [ "dev", "feat/SANTAANA-v1.0" ]

jobs:
  trufflehog:
    runs-on: ubuntu-latest
    permissions:
      contents: read
    steps:
      - name: Checkout
        uses: actions/checkout@v4
        with:
          fetch-depth: 0  # Necesario para escanear todo el historial de git

      - name: TruffleHog
```

```
      uses: trufflesecurity/trufflehog@main
      with:
        path: ./
        extra_args: --only-verified --exclude-paths=.trufflehogignore

    - name: Exportar reporte JSON (filesystem)
      if: ${{ always() }}
      run: |
        curl -sSfL https://raw.githubusercontent.com/trufflesecurity/trufflehog/main/scripts/install.sh | sh -s -- -b /usr/local/bin
        if [ -f ".trufflehogignore" ]; then
          trufflehog filesystem --only-verified --json --exclude-paths=.trufflehogignore . > trufflehog_filesystem.json 2>&1 || true
        else
          trufflehog filesystem --only-verified --json . > trufflehog_filesystem.json 2>&1 || true
        fi
    - name: Exportar reporte JSON (git history)
      if: ${{ always() }}
      run: |
        if [ -f ".trufflehogignore" ]; then
          trufflehog git --only-verified --json --exclude-paths=.trufflehogignore . > trufflehog_git.json 2>&1 || true
        else
          trufflehog git --only-verified --json . > trufflehog_git.json 2>&1 || true
        fi
    - name: Subir artefactos
      if: ${{ always() }}
      uses: actions/upload-artifact@v4
      with:
        name: be-trufflehog-reports
        path: |
          trufflehog_filesystem.json
          trufflehog_git.json
        retention-days: 30

    - name: Resumen de resultados
      if: ${{ always() }}
      run: |
        echo "### TruffleHog Scan Results" >> $GITHUB_STEP_SUMMARY
        echo "" >> $GITHUB_STEP_SUMMARY

        if [ -f "trufflehog_filesystem.json" ]; then
          FILESYSTEM_COUNT=$(grep -c "DetectorType" trufflehog_filesystem.json 2>/dev/null || echo "0")
          echo "**Filesystem scan:** $FILESYSTEM_COUNT secretos verificados encontrados" >> $GITHUB_STEP_SUMMARY
        fi

        if [ -f "trufflehog_git.json" ]; then
```

```
        GIT_COUNT=$(grep -c "DetectorType" trufflehog_git.json 2>/dev/null ||
        ↪  echo "0")
        echo "**Git history scan:** $GIT_COUNT secretos verificados
        ↪  encontrados" >> $GITHUB_STEP_SUMMARY
      fi

      echo "" >> $GITHUB_STEP_SUMMARY
      echo "Los reportes completos están disponibles en los artifacts de este
      ↪  workflow." >> $GITHUB_STEP_SUMMARY
```

Figura 6: Archivo YAML del flujo de TruffleHog en *backend*

El flujo se implementó exitosamente y mostraría como resultado el resumen informando los resultados del escaneo en el historial de Git como también del *filesystem*, ya sea si encontró secretos encontrados o no, y también generando archivos JSON con los reportes completos.

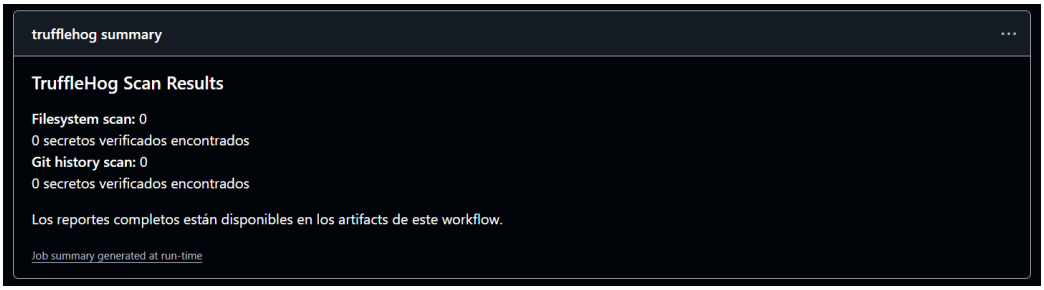


Figura 7: Resultados del flujo de TruffleHog

Por parte de SonarCloud, para realizar su implementación se procedió a enlazarlo con la cuenta de GitHub con la que se estuvo desarrollando el trabajo y también la organización en donde se crearon y se alojaron los repositorios.

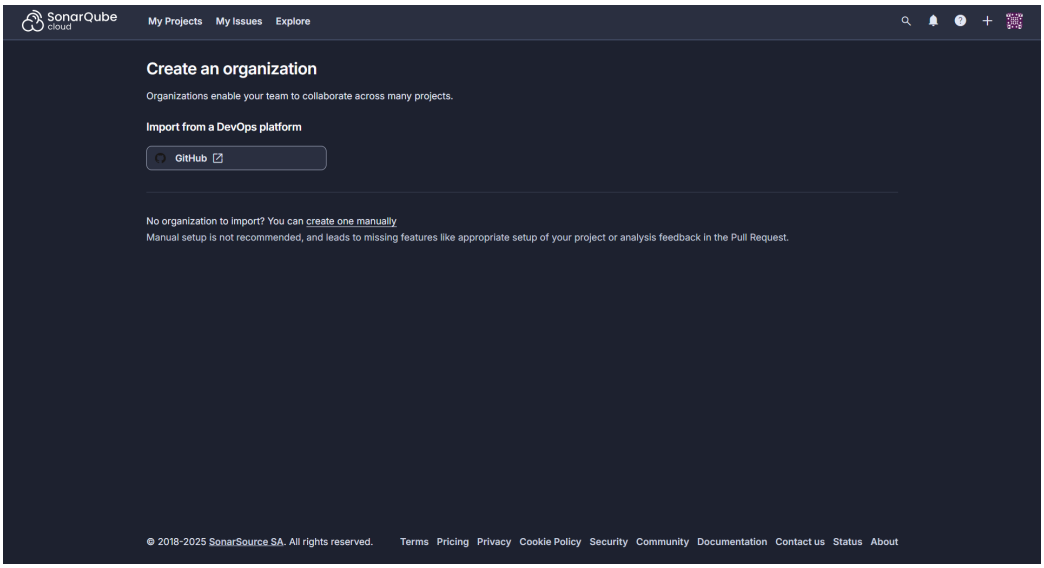


Figura 8: Menú para creación de organización en SonarQube Cloud

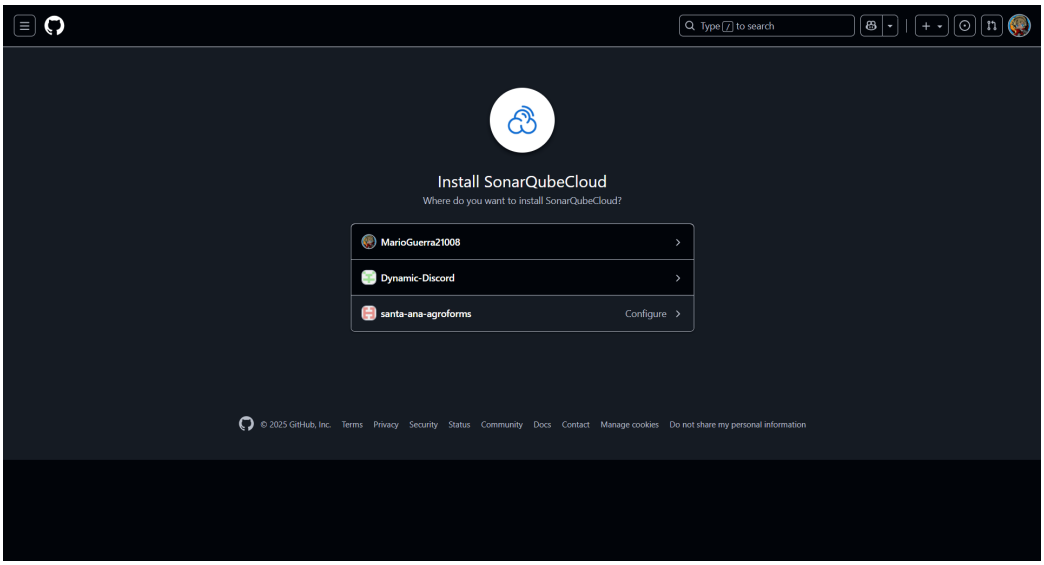


Figura 9: Pantalla para instalación de SonarQube Cloud en la organización

Una vez realizado el proceso de instalación, se procedió a configurar los repositorios a los cuales se realizaron los correspondientes análisis tanto en GitHub como en la página de SonarCloud.

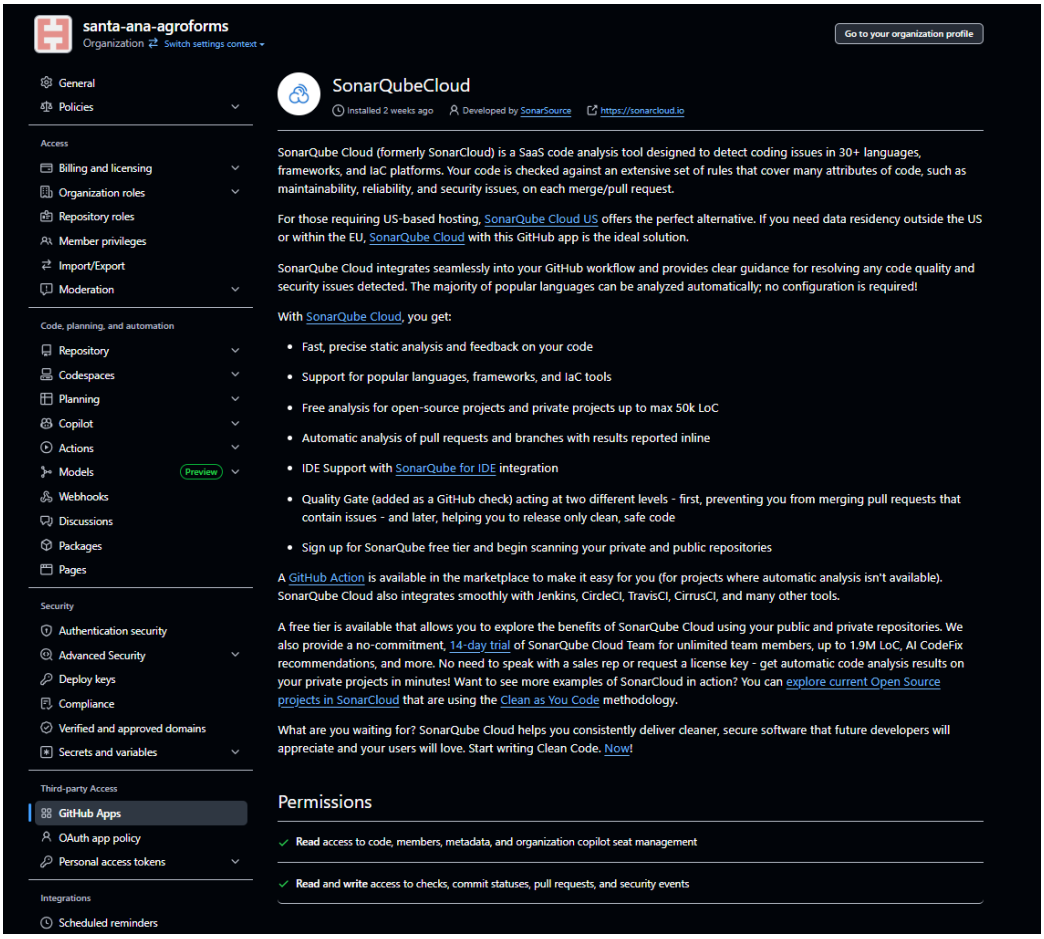


Figura 10: Instalación de SonarQube Cloud dentro de la organización de GitHub

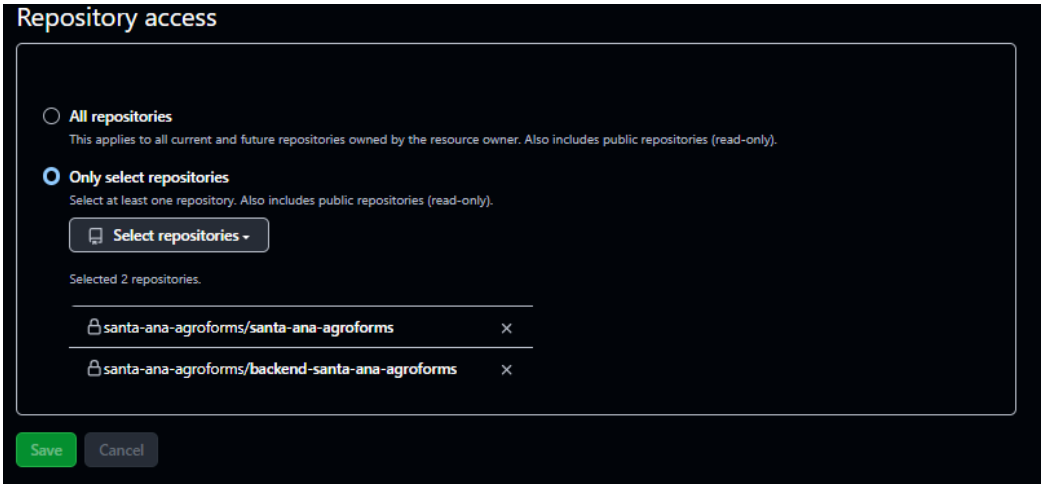


Figura 11: Accesos brindados a los repositorios de *frontend* y *backend*

Posteriormente, se configuraron los análisis de código estático para las ramas principales de ambos repositorios, al igual que las indicaciones de nuevo código en el repositorio por medio de *pull requests* o de *pushes*.

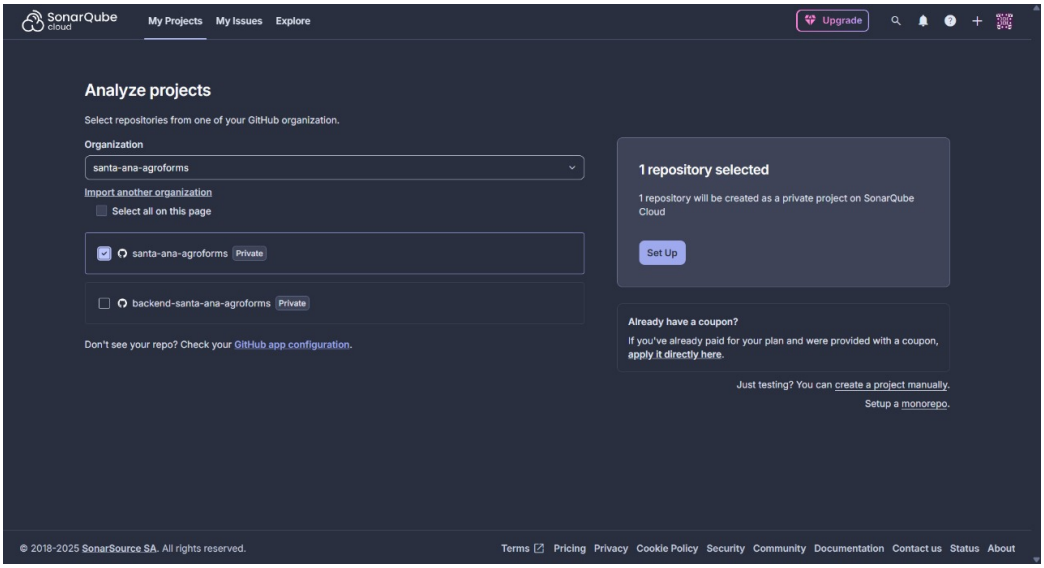


Figura 12: Configuración del repositorio del *frontend* en SonarQube Cloud

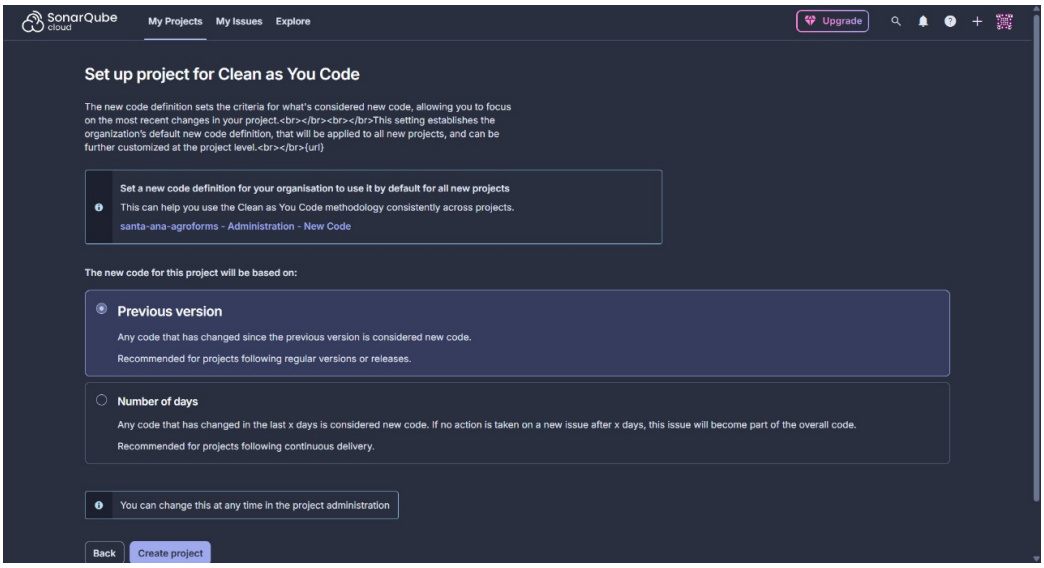


Figura 13: Configuración de consideraciones de nuevo código dentro del repositorio

Finalmente, al ejecutarse el análisis de ambos repositorios, se pudieron ejecutar los flujos de trabajo de SonarCloud designados para evaluación de nuevo código y para detección de nuevos *issues*.

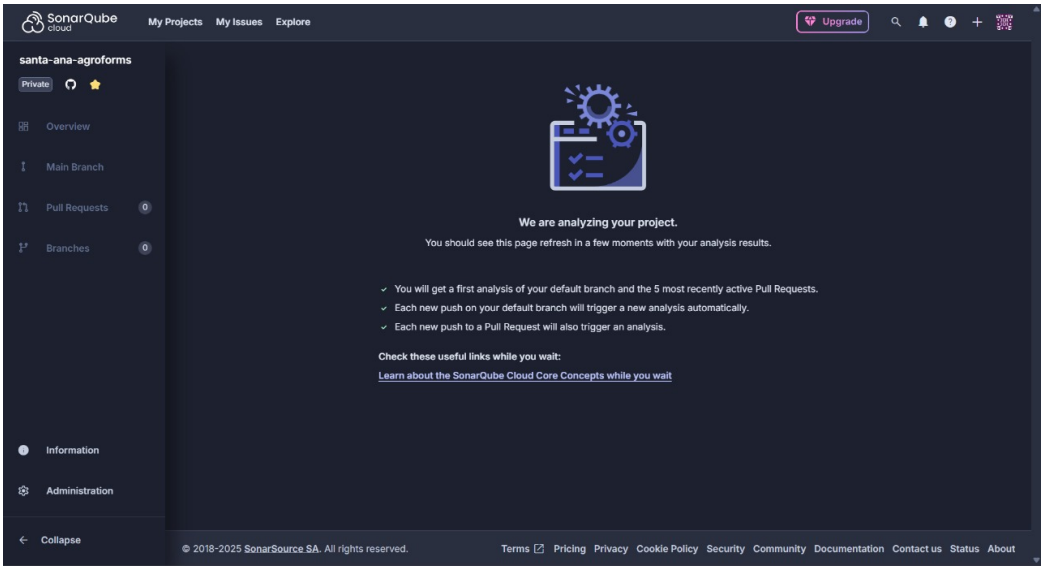


Figura 14: Análisis de código estático del *frontend* en ejecución

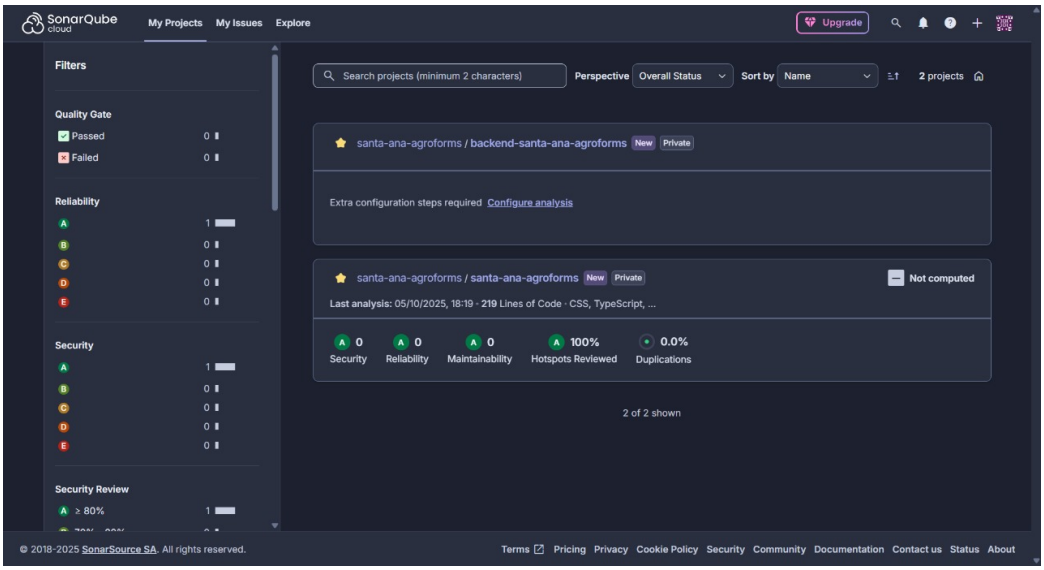


Figura 15: Página principal de SonarQube Cloud con ambos proyectos creados

Por parte de Snyk, se creó de igual manera una organización, con la cual se vinculó la cuenta de GitHub de la organización en la que los repositorios de GitHub fueron creados. Y, una cuenta de Google para realizar los análisis de dependencias en ambos proyectos. Snyk contó con una implementación local por medio de la *Command Line Interface* (CLI) y también su integración con GitHub como el ya mencionado flujo de trabajo en GitHub Workflows.

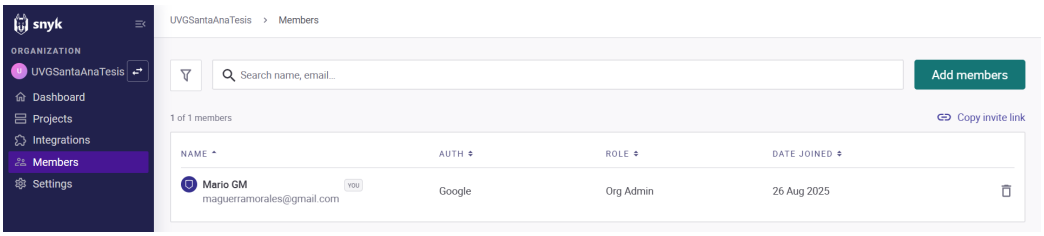


Figura 16: Organización creada para utilizar Snyk

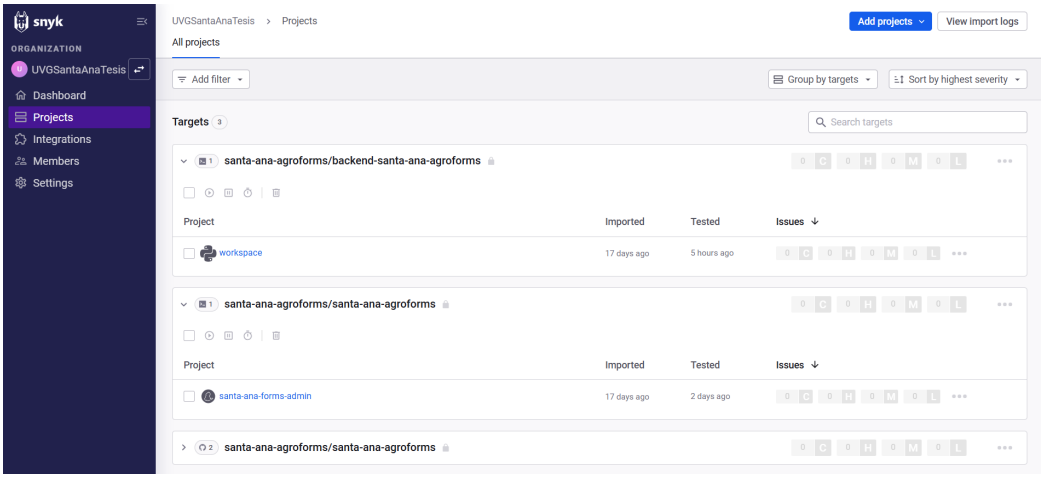


Figura 17: Proyectos creados para repositorios de *frontend* y *backend*

Snyk contribuyó en gran manera en la detección de vulnerabilidades en dependencias en ambos repositorios, encontrando hallazgos en las librerías utilizadas por los demás desarrolladores y también aportando actualizaciones a las dependencias ya existentes por medio de *pull requests*. Esto con el objetivo de velar por la mantenibilidad del sistema y prevenir que se desactualice, ocasionando posibles vulnerabilidades en el largo plazo.

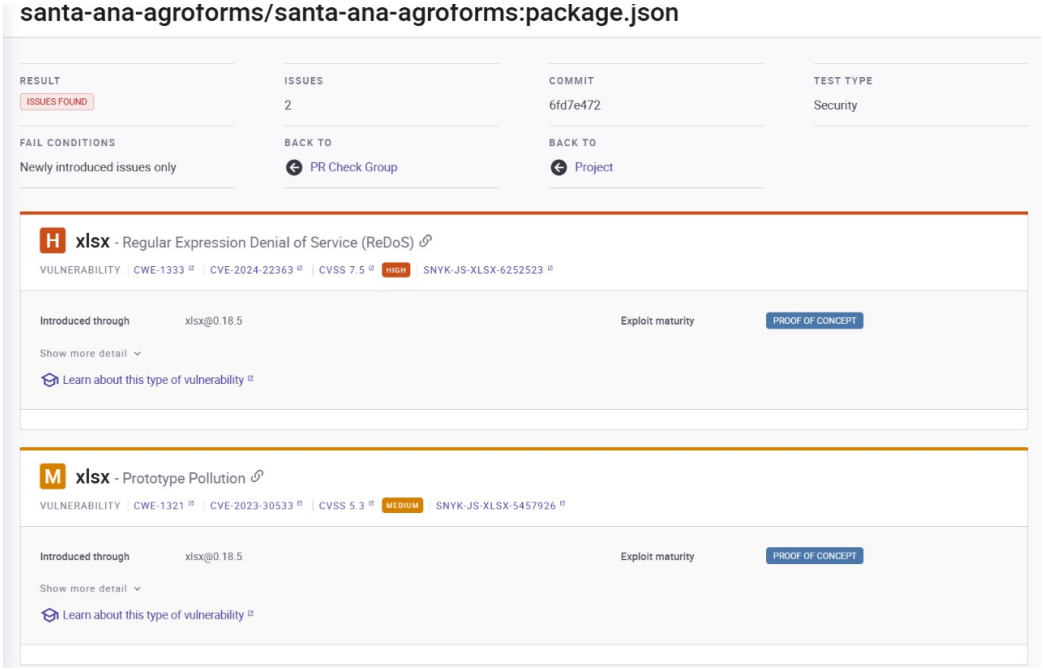


Figura 18: Vulnerabilidades críticas reportadas por Snyk en package.json

Figura 19: *Pull requests* de dependencias propuestas por Snyk

5.4. Implementación de pruebas automatizadas

En el apartado de pruebas automatizadas, se implementaron un total de 490 pruebas repartidas entre *frontend* y *backend* y de la siguiente manera:

- 193 pruebas unitarias de Jest en la plataforma web.
- 50 pruebas unitarias de Mocha en la plataforma web.

- 167 pruebas unitarias de Pytest en la API.

- 50 pruebas *end-to-end* de Cypress en la plataforma web.

- 30 pruebas *end-to-end* de Selenium en conjunto con la integración de pruebas de compatibilidad cruzada con LambdaTest en la plataforma web.

Esto se hizo debido a los requerimientos y porcentaje de cobertura en el código que se buscaba como parte de la validación de funcionalidades de la plataforma web. Buscando un porcentaje entre el 70 al 80 por ciento como mínimo dentro del marco de referencia de cobertura entre los *frameworks* principales, Pytest y Jest.
La delimitación de las pruebas por *framework* implementado se hizo de la siguiente forma:

Framework	**Cantidad**	**Tipo**	**Alcance**
Jest	193	Unitaria	Componentes y servicios *frontend*
Mocha	50	Integración	Funcionalidades *frontend*
Cypress	50	*End-to-End*	Flujos completos de usuario
Selenium + LambdaTest	30	*End-to-End*	Pruebas *cross-browser*
Pytest	167	Unitaria/Integración	API y lógica *backend*
Total	**490**	-	-

Cuadro 1: Resumen de pruebas implementadas por *framework*

Las pruebas al dividirse en tres principales tipos, siendo unitarias, integración y *end-to-end*, se concentraron en brindar pruebas a diversas funcionalidades de la aplicación como conjunto, desde los componentes, servicios y funcionalidades en *frontend* y *backend*, además de brindar flujos completos de usuario mediante las pruebas en Cypress y Selenium. Estas como conjunto se dividieron como se indica en el Cuadro 2.

Tipo	**Cantidad**	**Descripción**
Unitarias	360	Pruebas de componentes, *hooks*, servicios y funciones individuales
Integración	50	Pruebas de interacción entre módulos
End-to-End	80	Pruebas de flujos completos de usuario
Total	**490**	-

Cuadro 2: Distribución de pruebas por tipo

Tomando en consideración lo mencionado, el conjunto total de pruebas se utilizó para probar funcionalidades que van desde la autenticación de usuarios, hasta toda la lógica implementada referente a la creación y gestión de formularios adaptables, importación de archivos para formatos de formularios, gestión de dispositivos, rutas de aprobación, fuentes de datos e integración con sistemas externos, entre otras.

Funcionalidad	Framework	Tipo
Autenticación de usuarios	Jest, Cypress	Unitaria, E2E
Creación de formularios	Jest, Cypress	Unitaria, E2E
Importación desde Excel	Jest, Cypress	Unitaria, E2E
Dashboard y métricas	Jest, Selenium	Unitaria, E2E
Gestión de dispositivos	Jest, Mocha	Unitaria, Integración
Rutas de aprobación	Jest, Cypress	Unitaria, E2E
Asignación de formularios	Jest, Cypress	Unitaria, E2E
Exportación de datos	Jest, Cypress	Unitaria, E2E
Navegación y routing	Mocha, Selenium	Integración, E2E
Accesibilidad (WCAG)	Mocha	Funcional

Cuadro 3: Principales funcionalidades probadas en el *frontend*

Módulo	Funcionalidad
Autenticación	Login, tokens JWT, permisos, roles
Gestión de Formularios	CRUD, validaciones, estructura de datos
Gestión de Usuarios	Administración de usuarios y permisos
Gestión de Dispositivos	Registro, actualización, sincronización
Exportación de Datos	Generación de archivos Excel, PDF
Fuentes de Datos	Integración con sistemas externos
Asignación de Formularios	Lógica de asignación y validaciones
Rutas de Aprobación	Workflows, estados, transiciones
Categorías	Organización y clasificación
Validaciones	Reglas de negocio, *constraints*

Cuadro 4: Módulos del *backend* probados con Pytest

La ejecución de las pruebas se realizó de dos maneras: Localmente en el ambiente de desarrollo y también por medio de flujos de trabajo en los repositorios respectivos, obteniendo resultados positivos como parte de la validación del código subido por los desarrolladores, asegurándose de que el código nuevo que se agregue al proyecto no llegue a romper otras funcionalidades, ni tampoco ocasionando que las pruebas sean desactualizadas al realizar *code refactoring*.

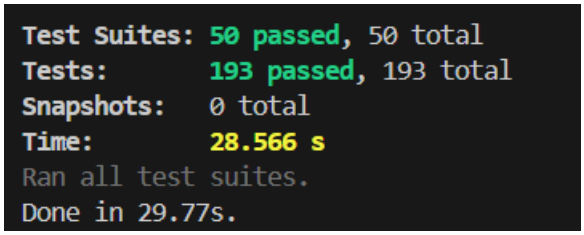


Figura 20: Resultado de ejecución de pruebas unitarias en Jest

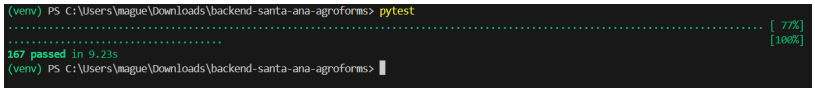


Figura 21: Resultado de ejecución de pruebas unitarias e integración en Pytest

Figura 22: Resultado de ejecución de pruebas de integración en Mocha

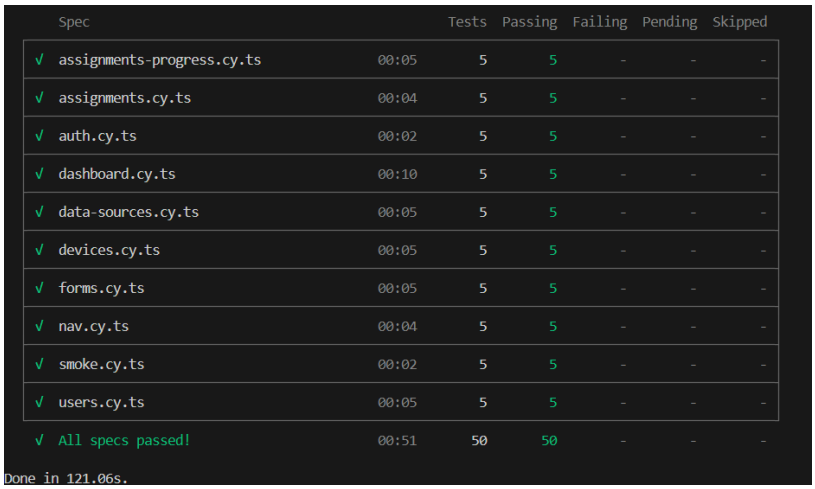


Figura 23: Resultado de ejecución de pruebas *end-to-end* en Cypress

Para la ejecución de las pruebas *end-to-end* en Selenium, además de su ejecución local, se contó también con la integración para pruebas de compatibilidad cruzada por medio de LambdaTest. Para realizar esto, se configuró un túnel proporcionado por LambdaTest para ejecutar las pruebas desde *localhost* o la dirección IP 127.0.0.1, además de habilitar el entorno de desarrollo en el puerto 5173, y mediante las credenciales proporcionadas desde el entorno, las pruebas fueron ejecutadas exitosamente.

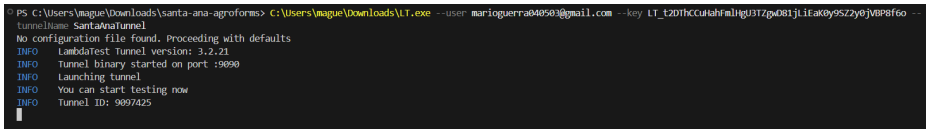


Figura 24: Ejecución de túnel de LambdaTest

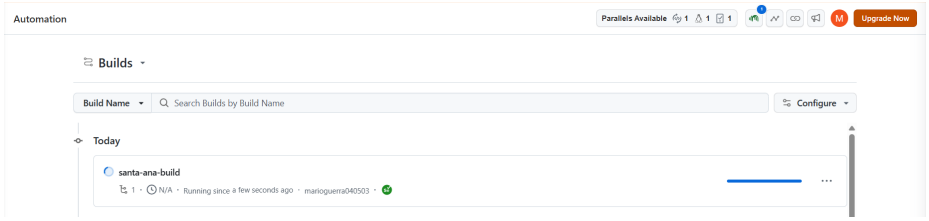


Figura 25: Inicio de ejecución de pruebas en LambdaTest

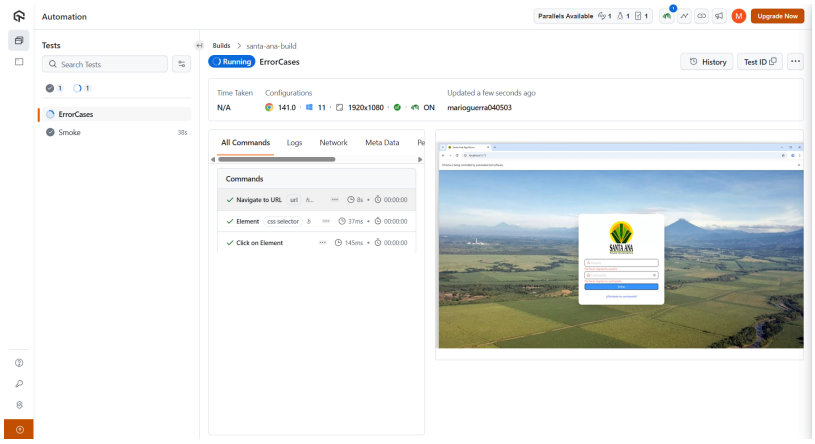


Figura 26: Captura de pantalla durante la ejecución de pruebas *cross-browser*

```
PS C:\Users\mague\Downloads\santa-ana-agroforms> yarn test:selenium:lambdatest
    ✓ [C0306] abre Sesión → Terminales (o Dispositivos/Devices) (7634ms)
    ✓ [C0307] abre Sesión → Usuarios/Users (7256ms)
    ✓ [C0308] abre Formularios → Fuentes de Datos (6599ms)

  Selenium Pages elementos clave
    ✓ [C0309] Login tiene inputs (9083ms)
Iniciando login...
Esperando React...
Seteando token mock en localStorage...
Navegando a /home...
URL actual: http://localhost:5173/home
Login exitoso (mock)
    ✓ [C0310] puede hacer login (16320ms)
    ✓ [C0311] Home tiene sidebar (149ms)
    ✓ [C0312] Dashboard tiene gráficas (canvas/svg si existen) (5125ms)
No se encontró tabla
    ✓ [C0313] Formularios tiene tabla (9673ms)

  Selenium Extras
Iniciando login...
Esperando React...
Seteando token mock en localStorage...
Navegando a /home...
URL actual: http://localhost:5173/home
Login exitoso (mock)
Sin botón Nueva/Agregar
    ✓ [C0314] Fuentes de Datos muestra botones de acción (8720ms)
    ✓ [C0315] Fuentes de Datos → modal con selects (143ms)
    ✓ [C0316] Listado de Formularios → buscador (6558ms)
Menú "Asignación de Formularios" no encontrado - skipping
    ✓ [C0317] Asignación de Formularios → combos/tabla/botón (4921ms)
Menú "Asignaciones en proceso" no encontrado - skipping
    ✓ [C0318] Asignaciones en proceso → filtros (opcionales) (6240ms)
Menú "Rutas de Aprobación" no encontrado - skipping
    ✓ [C0319] Rutas de Aprobación carga (5938ms)
    ✓ [C0320] Procesos de Exportación muestra tabla (7579ms)
    ✓ [C0321] Crear desde Excel muestra UI básica (6333ms)
    ✓ [C0322] Terminales muestra lista/acciones (8287ms)
    ✓ [C0323] Usuarios muestra tabla/buscador (7903ms)


  30 passing (5m)

Done in 315.62s.
```

Figura 27: Resultados de ejecución de pruebas en Selenium

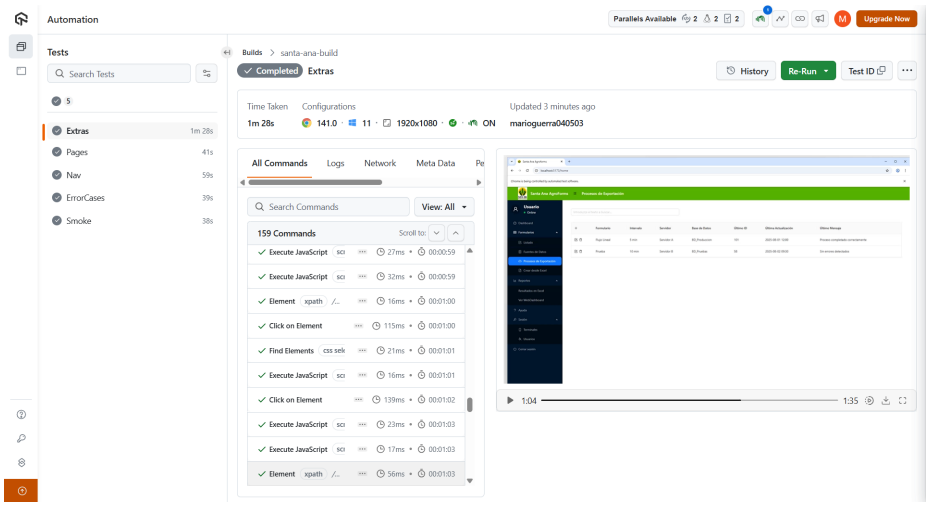


Figura 28: Resultados de ejecución de pruebas en LambdaTest

5.5. Controles de seguridad en ejecución

La implementación de estos controles de seguridad incluyen la integración del estándar OAuth 2.0 como parte de la gestión de identidades y acceso, en conjunto con el módulo de *backend*. Así como también la inclusión de *Content Security Policy* para la regulación y límites de recursos para la carga y ejecución de estos, y OpenSSL como parte de la encriptación de datos en tránsito y en reposo.
Para OAuth 2.0, se utilizó el paquete *django-oauth-toolkit*, la cual es una implementación completa de OAuth 2.0 en Django, por medio del archivo de configuración de la API.

```
# Configuración de OAuth2
OAUTH2_PROVIDER = {
    'SCOPES': {
        'read': 'Read scope',
        'write': 'Write scope',
    },
    'ACCESS_TOKEN_EXPIRE_SECONDS': 36000,  # 10 horas
    'REFRESH_TOKEN_EXPIRE_SECONDS': 86400,  # 24 horas
}

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'oauth2_provider.contrib.rest_framework.OAuth2Authentication',
        'rest_framework.authentication.SessionAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
        'formularios.permissions.IsWebAllowed',
    ],
    # ← Agregar esto para drf-spectacular
    'DEFAULT_SCHEMA_CLASS': 'drf_spectacular.openapi.AutoSchema',
}
```

Figura 29: Configuración de OAuth 2.0 en settings.py

Con este *toolkit*, se procedió a crear el proceso de autenticación con el cual sigue el siguiente flujo:

- Solicitud de login: el cliente envía credenciales al *endpoint* POST /api/auth/login.
- Validación de usuario: el *backend* verifica la existencia y estado activo del acceso web.
- Verificación de contraseña mediante Argon2 *hash*.
- Revocación de *tokens* previos por medio de *single sign-on*.
- Generación de *tokens* como *access token* y *refresh token*.
- Se retornan los *tokens* y datos del usuario como respuesta.

```python
@api_view(['POST'])
@permission_classes([AllowAny])
def login(request):
    """
    Login para WEB — solo usuarios con acceso_web=True
    """
    nombre_usuario = (request.data.get('nombre_usuario') or "").strip()
    password = request.data.get('password') or ""

    if not nombre_usuario or not password:
        return _json_error("nombre_usuario y password son requeridos", status.HTTP_400_BAD_REQUEST)

    try:
        # 1) buscar usuario
        try:
            usuario = Usuario.objects.get(nombre_usuario=nombre_usuario)
        except Usuario.DoesNotExist:
            return _json_error("Credenciales inválidas", status.HTTP_401_UNAUTHORIZED)

        # 2) reglas de acceso antes de validar credenciales
        if not bool(usuario.activo):
            return _json_error("Usuario inactivo", status.HTTP_403_FORBIDDEN)

        if bool(usuario.acceso_web) is not True:
            return _json_error(
                "Este usuario no tiene acceso a la plataforma web. Use la aplicación móvil.",
                status.HTTP_403_FORBIDDEN
            )

        # 3) validar contraseña
        try:
            if not verify_password(usuario.password, password):
                return _json_error("Credenciales inválidas", status.HTTP_401_UNAUTHORIZED)
        except Exception:
            return _json_error("Credenciales inválidas", status.HTTP_401_UNAUTHORIZED)
```

```python
        # 4) obtener/crear app OAuth2
        app, _ = Application.objects.get_or_create(
            name='Default App',
            defaults={
                'client_type': Application.CLIENT_CONFIDENTIAL,
                'authorization_grant_type': Application.GRANT_PASSWORD,
            }
        )

        # 5) revocar tokens previos
        AccessToken.objects.filter(user=usuario).delete()
        RefreshToken.objects.filter(user=usuario).delete()

        # 6) emitir nuevos tokens
        expires = timezone.now() + timedelta(seconds=36000)
        access_token = AccessToken.objects.create(
            user=usuario,
            token=generate_token(),
            application=app,
            expires=expires,
            scope='read write'
        )
        refresh_token = RefreshToken.objects.create(
            user=usuario,
            token=generate_token(),
            application=app,
            access_token=access_token
        )
```

Figura 30: Lógica de *login* en auth_views.py

Además, se definieron los *tokens* con una duración de 10 horas para los *tokens* de acceso, tomando en cuenta el horario laboral de los trabajadores del Ingenio, y también los *tokens* de actualización con 24 horas para renovar los permisos de acceso sin necesidad de hacer un nuevo inicio de sesión.

En los casos de revocación de *tokens*, se implementaron los escenarios de un *logout* explícito que se da al cerrar sesión habitualmente en la página, un nuevo *login*, el cual revoca los permisos previos por los principios de *SSO*, un usuario desactivado, el cual revoca todos los *tokens* de ese usuario; y por último, cuando la *flag* de acceso_web está desactivado, por ende, los *tokens* se revocan automáticamente.

```python
@receiver(pre_save, sender=Usuario)
def revoke_tokens_on_flag_disable(sender, instance: Usuario, **kwargs):
    if not instance.pk:
        return

    try:
        prev = sender.objects.get(pk=instance.pk)
    except sender.DoesNotExist:
        return

    turned_off = (prev.acceso_web and not instance.acceso_web) or \
                 (prev.activo and not instance.activo)
    if turned_off:
        AccessToken.objects.filter(user=instance).delete()
        RefreshToken.objects.filter(user=instance).delete()
```

Figura 31: Función para revocación de permisos en signals.py

Posteriormente, se muestran los *endpoints* de autenticación con los que cuenta la API con una breve descripción de qué es lo que estos hacen:

Endpoint	**Método**	**Descripción**
/api/auth/login/	POST	*Login* con credenciales, retorna *tokens*
/api/auth/logout/	POST	Revoca *token* actual del usuario
/api/auth/me/	GET	Información del usuario autenticado
/o/token/	POST	*Refresh token endpoint*
/o/revoke_token/	POST	Revocación manual de *token*

Cuadro 5: *Endpoints* de autenticación implementados en la API

Para la demostración de su implementación se cuenta también con algunos casos en el inicio de sesión de la plataforma.

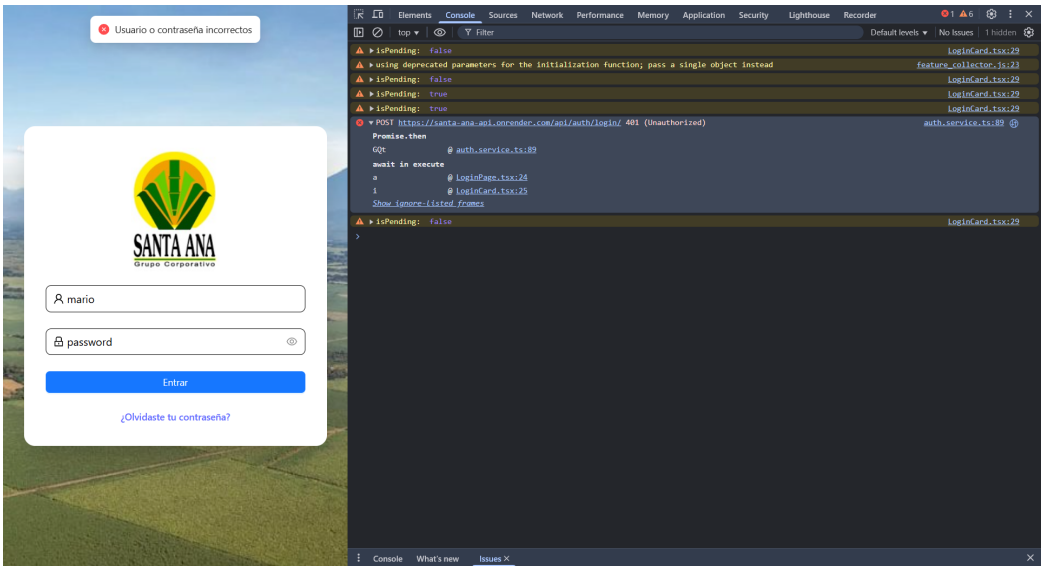


Figura 32: Mensaje de error 401 en consola al ingresar datos incorrectos

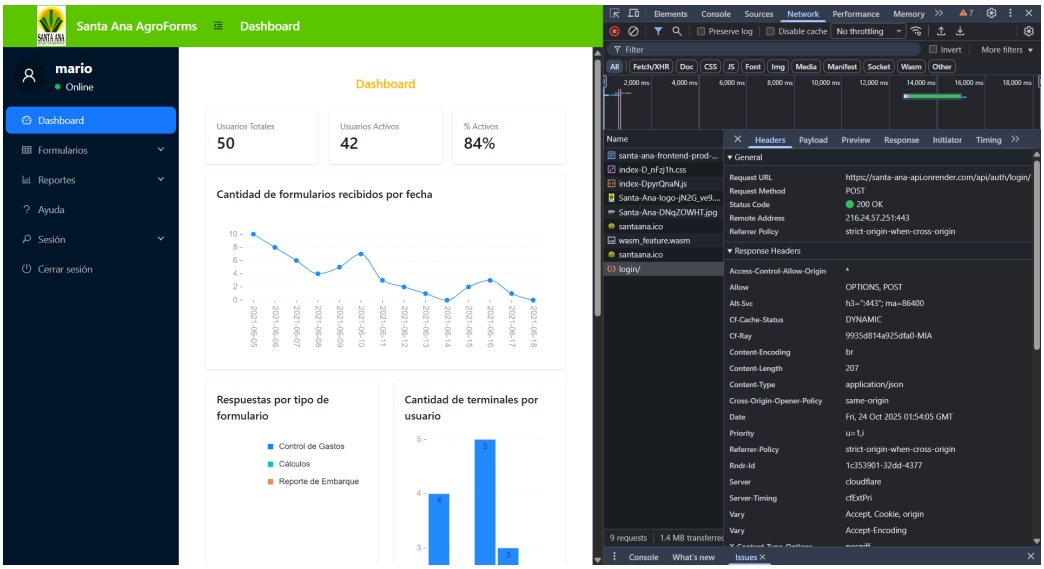


Figura 33: Evidencia de *endpoint* en la plataforma al iniciar sesión

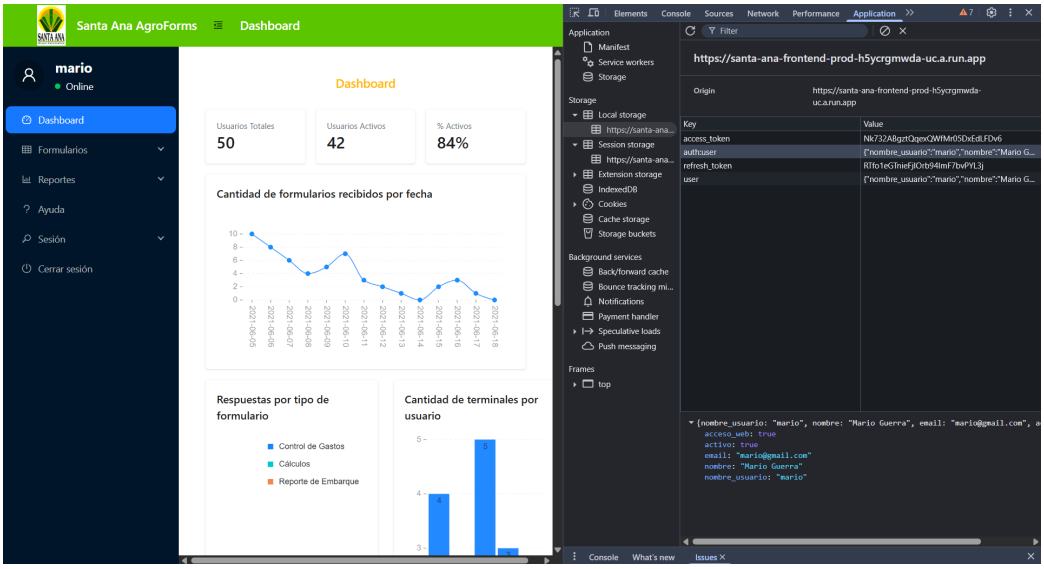


Figura 34: Contenido de los *tokens* de acceso y de actualización

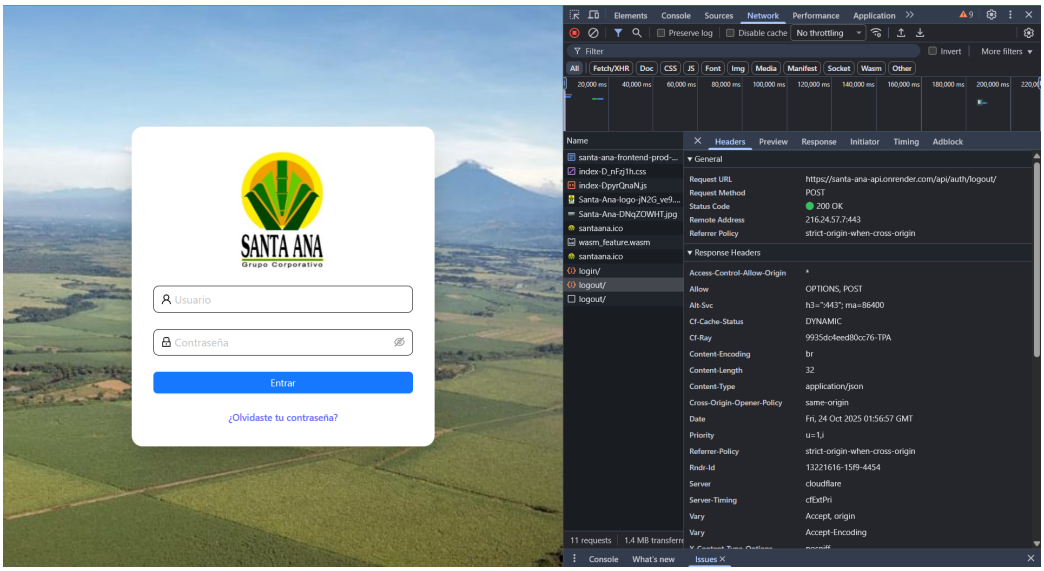


Figura 35: Evidencia de *endpoint* en la plataforma al cerrar sesión

Para la implementación de *Content Security Policy* se basa primordialmente en la prevención de ataques de *cross-site scripting*, inyección de código malicioso, *clickjacking* y robo de datos de la aplicación. Este fue integrado a la plataforma web mediante el archivo vite.config.js para el desarrollo y en el archivo nginx.conf para el contenedor de Docker para producción.

```
const DEV_CSP = [
  "default-src 'self'",
  "base-uri 'self'",
  "object-src 'none'",
  "frame-ancestors 'none'",
  "script-src 'self' 'unsafe-inline' 'unsafe-eval' blob:",
  "style-src 'self' 'unsafe-inline'",
  "img-src 'self' data: blob:",
  "font-src 'self'",
  "connect-src 'self' ws: wss: https://santa-ana-api.onrender.com https://santaana-api-latest.onrender.com",
  "media-src 'self' blob:",
  "worker-src 'self' blob:",
  "form-action 'self'",
].join("; ");

function cspHeaderPlugin() {
  return {
    name: "dev-csp-header",
    configureServer(server) {
      server.middlewares.use((req, res, next) => {
        res.setHeader("Content-Security-Policy", DEV_CSP);
        res.setHeader("Referrer-Policy", "strict-origin-when-cross-origin");
        res.setHeader("X-Content-Type-Options", "nosniff");
        res.setHeader("X-Frame-Options", "DENY");
        res.setHeader("Permissions-Policy", "geolocation=(), microphone=(), camera=(), accelerometer=(), gyroscope=(), payment=()");
        res.setHeader("Cross-Origin-Opener-Policy", "same-origin");
        res.setHeader("Cross-Origin-Resource-Policy", "same-origin");
        next();
      });
    },
  };
}
```

Figura 36: Encabezados en ambiente de desarrollo de *Content Security Policy*

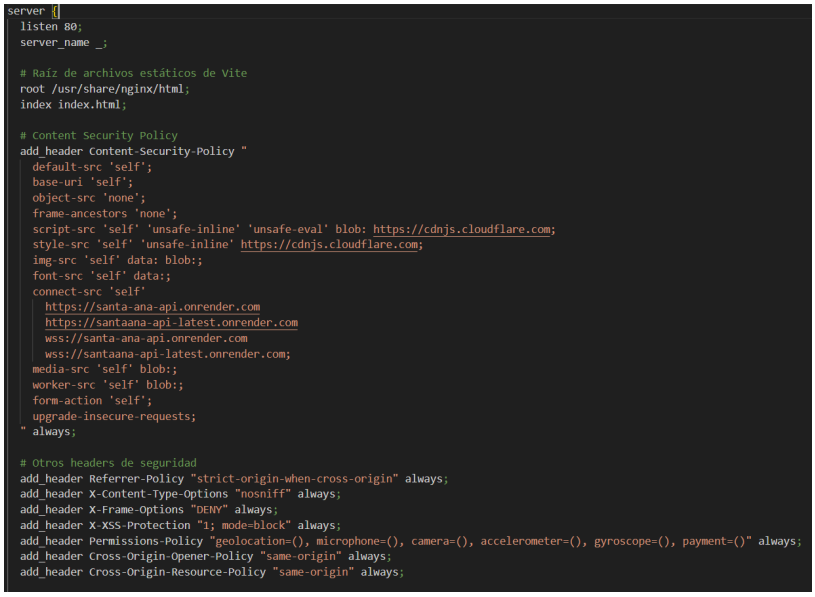


Figura 37: Encabezados en ambiente de producción de *Content Security Policy*

A continuación, se presentan en el Cuadro 6, las directivas de *CSP* implementadas y su propósito en la aplicación web:

Directiva	**Propósito**
default-src 'self'	Solo recursos del mismo origen por defecto
base uri 'self'	Prevenir inyecciónes de código en URL base
object-src 'none'	Elimina vectores de ataque de *plugins* obsoletos
frame-ancestors 'none'	Previene *clickjacking* bloqueando embebido en *iframes*
script-src ...	Permite *scripts* de React, Vite HMR, y Web Workers
style-src ...	Permite el contenido CSS de la plataforma
img-src ...	Permite las imágenes utilizadas en la plataforma
font-src 'self'	Permite *scripts* de React, Vite HMR, y Web Workers
connect-src ...	Autoriza conexión a API *backend* y WebSockets de desarrollo
media-src ...	Permite videos y audio
worker-src ...	Permite de igual manera Web Workers
form-action 'self'	Permite los formularios sólo en el mismo dominio
upgrade-insecure-requests	Fuerza protocolo HTTPS en la plataforma

Cuadro 6: Directivas CSP implementadas en Santa Ana Agroforms

Se tomaron en cuenta también ciertas excepciones con algunas funcionalidades de la plataforma con respecto a las directivas implementadas. Tal es el caso con *script-src* para la validación de los tokens en el inicio de sesión, además de la función de generación de códigos QR para la aplicación móvil desde la plataforma web en el ambiente de desarrollo. Mientras que para el ambiente de producción se añadió la directiva *upgrade-insecure-requests*, la cual sirve para forzar el protocolo HTTPS en la plataforma. Finalmente, se hicieron pruebas de seguridad con bloqueos de *scripts* maliciosos y de control de conexiones en la API para validar el buen funcionamiento de CSP.

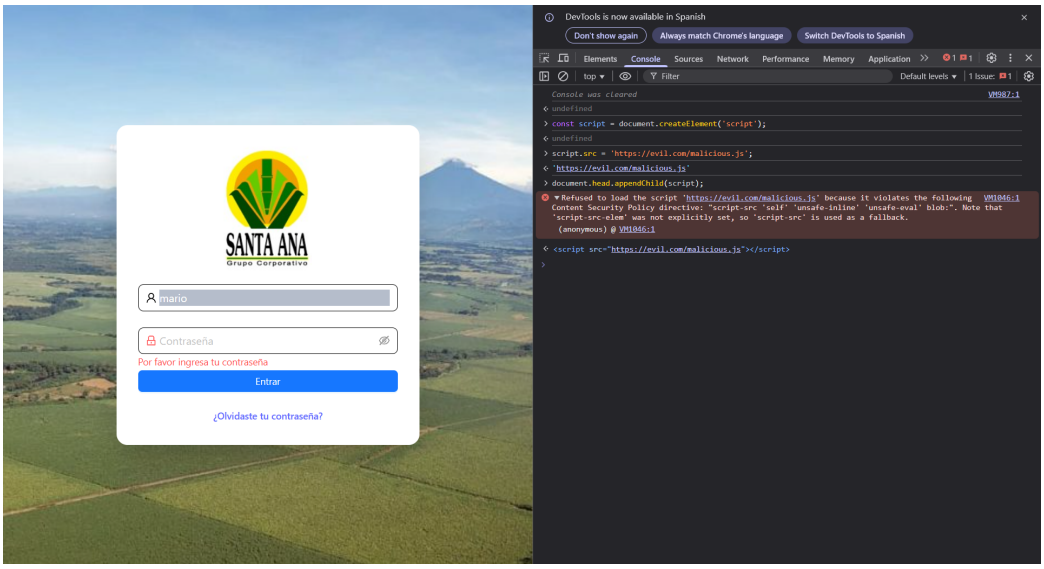


Figura 38: Prueba de seguridad de CSP con un *script* malicioso

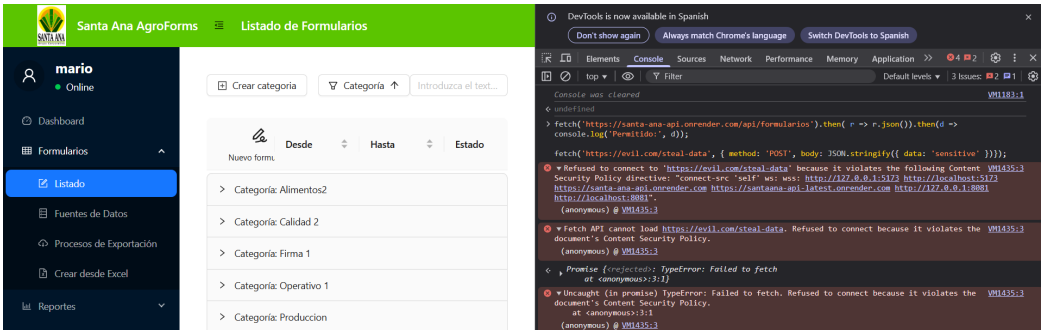


Figura 39: Prueba de seguridad de CSP de control de conexiones en la API

Finalmente, se muestra evidencia del uso de Cloudflare y de cifrados TLS en la infraestructura del sistema por medio de Netlify, Cloud Run y Render como proveedores de servicios en la nube.

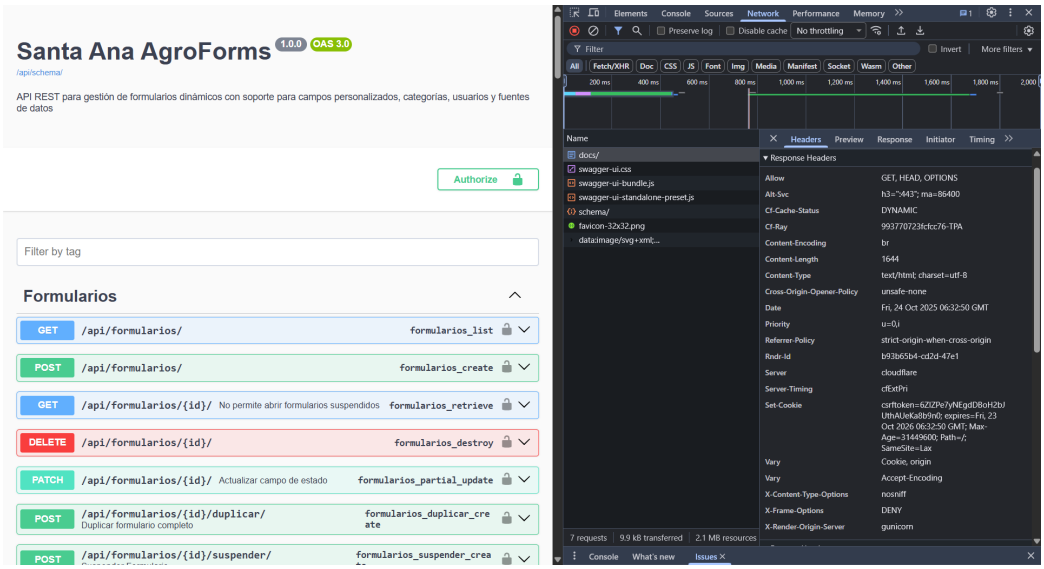


Figura 40: Headers del servidor de Cloudflare presentes en *backend*

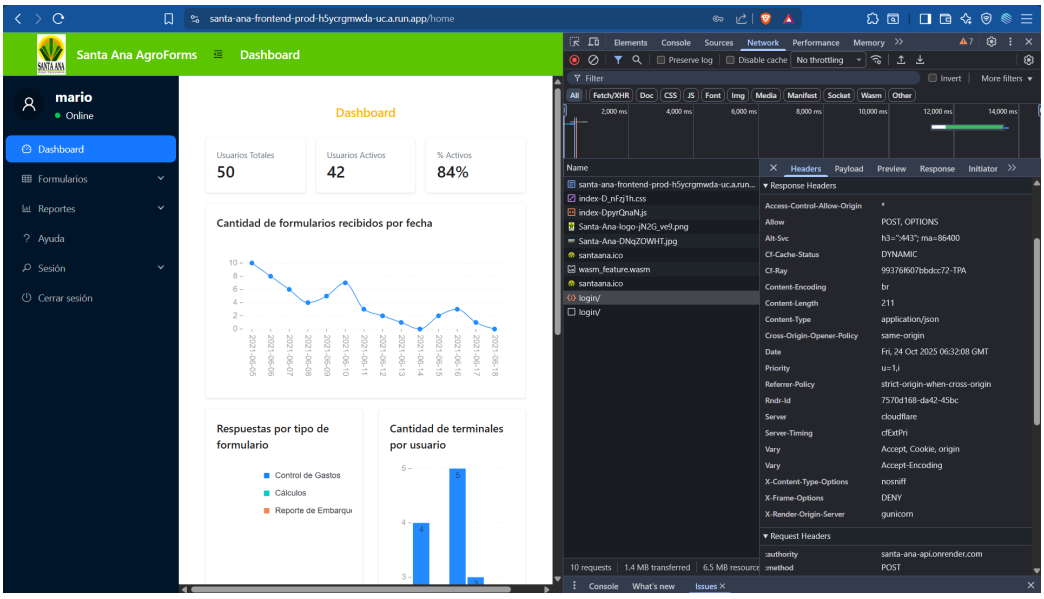


Figura 41: Headers del servidor de Cloudflare presentes en *frontend*

Los encabezados inyectados por Cloudflare para ambas partes del sistema son los siguientes:

Encabezado	**Propósito**
allow (GET, HEAD, OPTIONS / POST, OPTIONS)	Indica los métodos HTTP permitidos por el recurso.
access-control-allow-origin: *	Permite que cualquier origen consuma la respuesta.
alt-svc: h3=":443"; ma=86400	Anuncia soporte HTTP/3/QUIC para mejorar latencia y tiempos de transferencia.
cf-cache-status: DYNAMIC	Estado de caché en Cloudflare (HIT/MISS/DYNAMIC)
cf-ray: <id-POP>	Identificador único de la solicitud y punto de presencia de Cloudflare.
content-encoding: br	Compresión Brotli para reducir el tamaño de la respuesta y mejorar tiempos de descarga.
content-length: <bytes>	Indica el tamaño de la respuesta.
content-type: text/html; charset=utf-8 / application/json	Permite interpretar correctamente la respuesta en el navegador/cliente.
cross-origin-opener-policy: unsafe-none / same-origin	Aislamiento de contexto entre ventanas y procesos.
date	Marca temporal (UTC) de la respuesta.
priority: u=0,i	Sugerencias de priorización para el planificador de recursos del cliente.
referrer-policy: strict-origin-when-cross-origin	Al salir del origen, sólo envía el origen en Referer.
rndr-id: <uuid>	Trazabilidad del servidor o plataforma que atendió la petición.
server: cloudflare	Indica el servidor o edge que respondió.
server-timing: cfExtPri	Métricas *Server-Timing* inyectadas por el edge para análisis de rendimiento.
set-cookie: csrftoken=...; SameSite=Lax; Path=/; Max-Age=...	Envía cookies. SameSite=Lax: ayuda a mitigar CSRF y usar Secure en HTTPS.
vary: Cookie, Origin / Accept, Cookie, Origin / Accept-Encoding	Instruye a variar la respuesta según cabeceras que cambian la representación.
x-content-type-options: nosniff	Previene *MIME sniffing*; el navegador no "adivina" tipos de contenido.
x-frame-options: DENY	Evita *clickjacking* impidiendo incrustación en iframe.
x-render-origin-server: gunicorn	Expone el servidor de aplicación upstream para depuración.

Cuadro 7: Encabezados de cloudflare inyectados en Santa Ana Agroforms

De igual manera, se evidencian los cifrados TLS para la encriptación en tránsito de los datos.

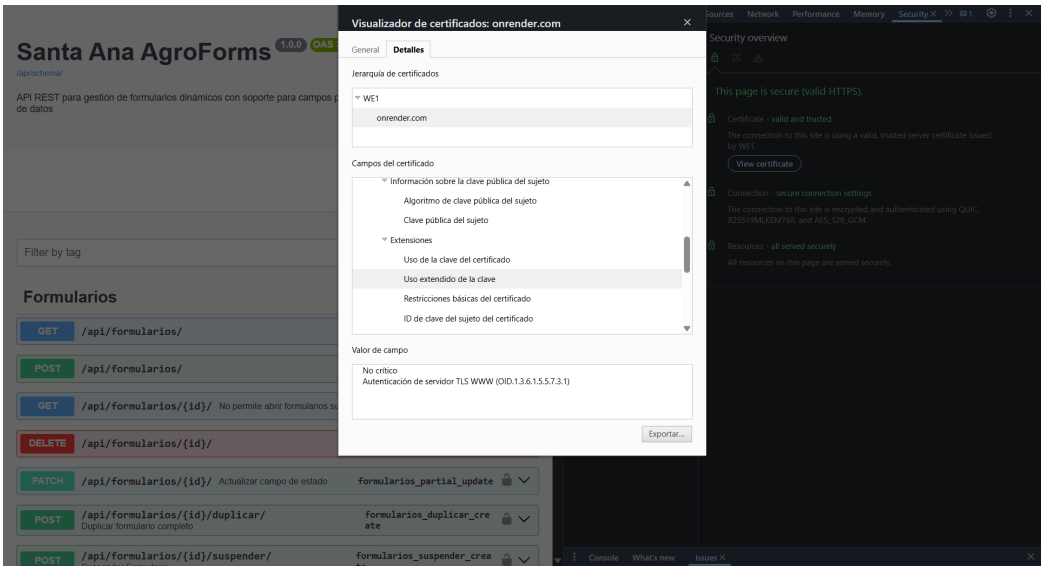


Figura 42: Certificados TLS en *backend*

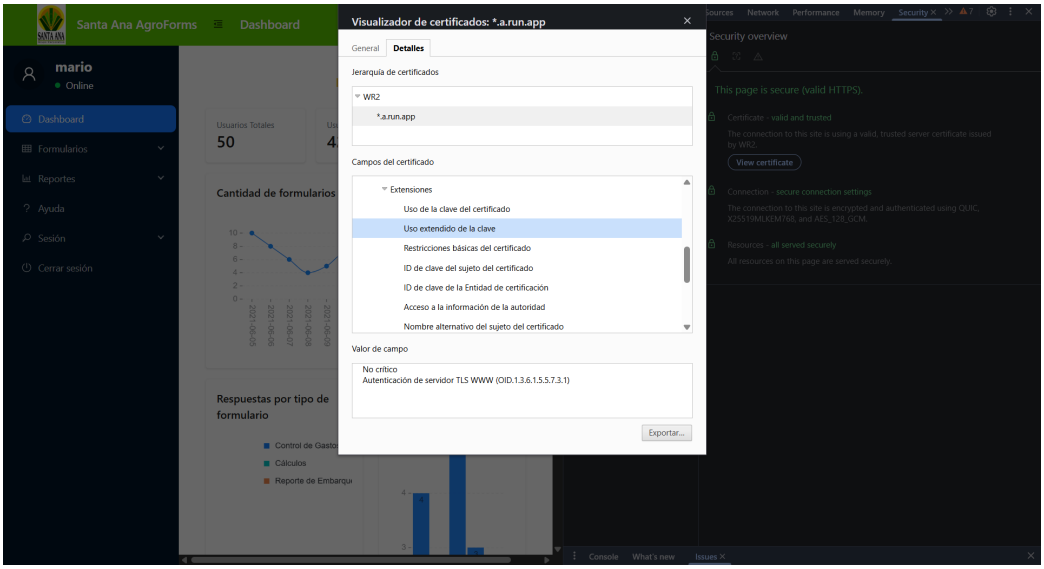


Figura 43: Certificados TLS en *frontend*

5.6. *Pentesting* automatizado y validación de seguridad

Para realizar pruebas de penetración y pruebas dinámicas de seguridad de aplicaciones se implementó la herramienta OWASP ZAP, con la cual se identificaron las vulnerabilidades tanto en la aplicación web como también en la API. El proceso se llevó a cabo mediante la opción de escaneo automatizado en sesiones distintas para cada apartado y en el ambiente de desarrollo en *localhost* y en los puertos 5173 y 8081 respectivamente. Esto para evitar inconvenientes con los proveedores de servicios en los cuales están siendo *hosteadas* la API y la plataforma web.

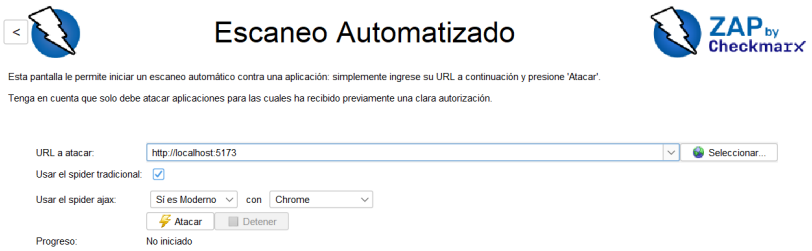


Figura 44: Configuración de escaneo automatizado en el puerto 5173

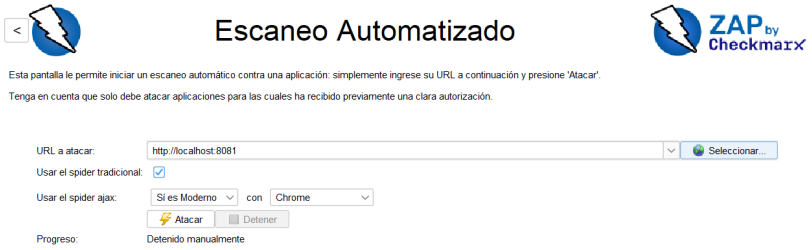


Figura 45: Configuración de escaneo automatizado en el puerto 8081

Durante los escaneos, se suscitaron diversas consultas a ambas URL, utilizando *Spider Ajax* para la recolección de datos y de endpoints en ambas partes que componen el sistema. A su vez, se incluyeron diversos análisis de *WebSockets* en la plataforma web y también se procedió a realizar escáneres activos para encontrar vulnerabilidades altas, medias, bajas e informacionales. A continuación, se muestra el procedimiento que la herramienta ZAP llevó a cabo para realizar estas tareas.

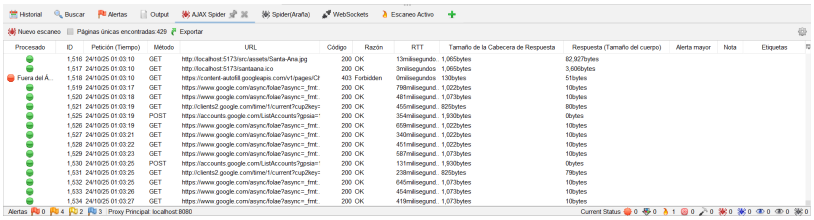


Figura 46: *Ajax Spider* ejecutándose en *frontend*

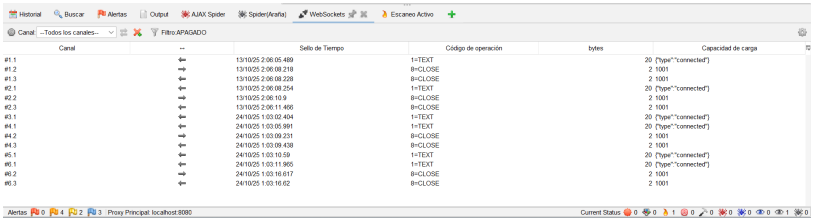


Figura 47: *WebSockets* empleados en *frontend*

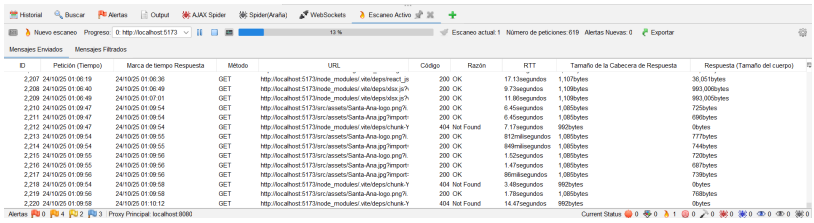


Figura 48: Escaneo activo en *frontend*

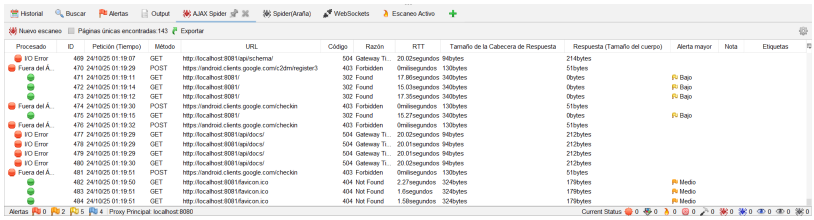


Figura 49: *Ajax Spider* ejecutándose en *backend*

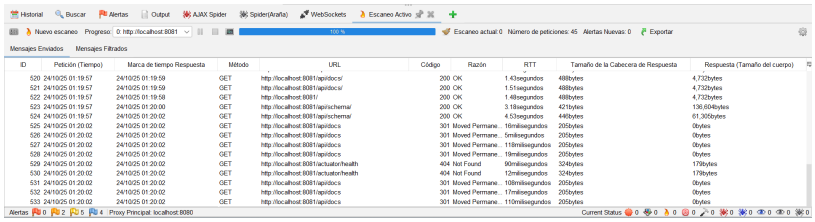


Figura 50: Escaneo activo en *backend*

5.7. Contenedorización y despliegue

Para la parte de contenedorización y despliegue de la plataforma web, se implementó una estrategia la cual abarca la creación de un contenedor Docker hasta el despliegue automatizado en la nube por medio de infraestructura como código con Terraform. Esto garantiza la reproducibilidad, escalabilidad y la facilidad del mantenimiento del sistema en entornos de producción, al contarse con soluciones las cuales se pueden interpretar más fácilmente y permiten configurar entornos de producción de manera más sencilla.
En la Figura 51, se muestra el flujo de la contenedorización y despliegue realizado en conjunto con los servicios descritos.

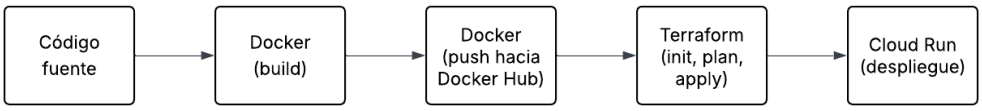


Figura 51: Diagrama de flujo del proceso de contenedorización y despliegue

El primer paso consistió en la implementación de un Dockerfile para la aplicación del frontend, definiendo los pasos necesarios para construir una imagen de manera funcional, siendo

estos organizados por varias etapas para optimizar el tamaño final de la imagen y mejorar la seguridad de esta, utilizando una imagen base de Nginx con Alpine Linux y definiendo variables de entorno para la conexión con la API, además de establecer un límite de memoria considerable para Node.js, en conjunto con las dependencias y configuraciones necesarias.

```
WORKDIR /app

# Aumentar límite de memoria para Node.js
ENV NODE_OPTIONS="--max-old-space-size=4096"

# Argumentos de build
ARG VITE_API_BASE_URL
ARG VITE_API_MOBILE_URL
ARG VITE_API_MOBILE_KEY

# Convertir ARG a ENV para que Vite las vea durante el build
ENV VITE_API_BASE_URL=$VITE_API_BASE_URL
ENV VITE_API_MOBILE_URL=$VITE_API_MOBILE_URL
ENV VITE_API_MOBILE_KEY=$VITE_API_MOBILE_KEY

# Habilitar corepack y yarn
RUN corepack enable \
 && corepack prepare yarn@1.22.22 --activate \
 && yarn --version

# Copiar archivos de dependencias
COPY package.json yarn.lock ./

# Instalar dependencias con timeout extendido
RUN yarn install --network-timeout 600000

# Copiar código fuente
COPY . .

# Build de producción con más memoria
RUN yarn build

# Verificar que el build fue exitoso
RUN ls -la /app/dist
```

Figura 52: Parte del contenido del Dockerfile

Una vez definido el Dockerfile, se procedió a construir la imagen del contenedor, ejecutándose con el comando 'docker build' y llevando a cabo cada paso definido por el Dockerfile.

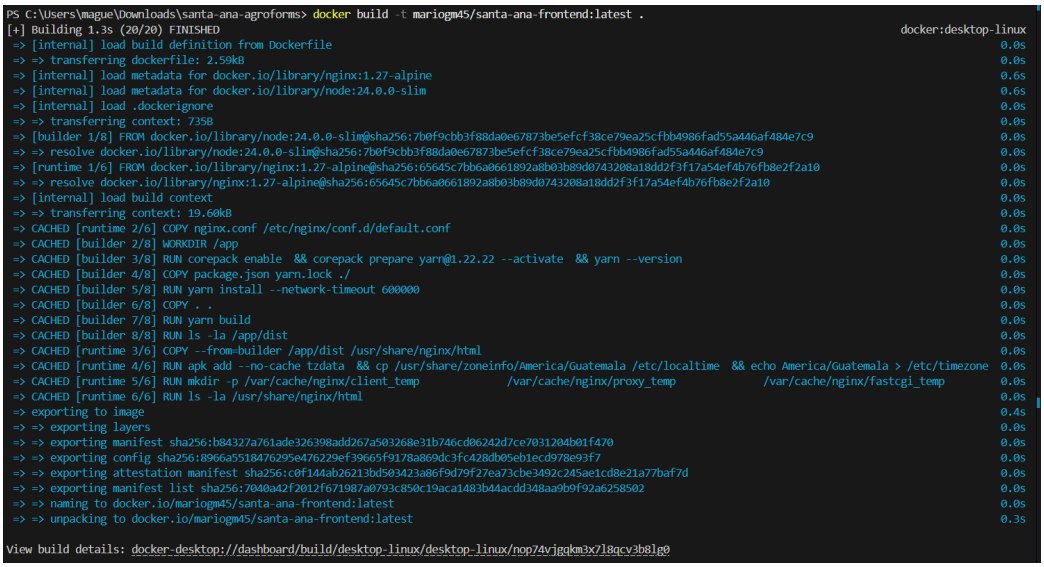


Figura 53: Construcción de la imagen de Docker

Posteriormente, con el comando 'docker run', se verificó el funcionamiento del contenedor en un entorno local antes de desplegarlo en la nube. Por ello, se mapeó el puerto 8080 del host al puerto 80 del contenedor, permitiendo así acceder a él a través de localhost:8080 y desplegando los *logs* generados y el contenedor en el navegador, para demostrar que su inicio fue realizado correctamente.

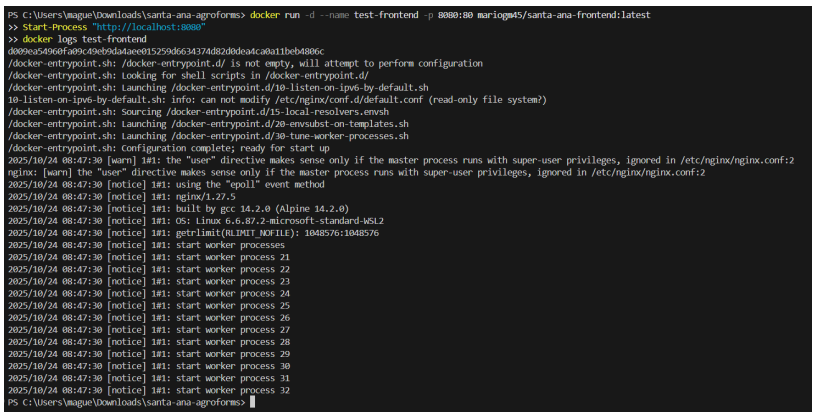


Figura 54: Ejecución de la imagen de Docker desde la línea de comandos

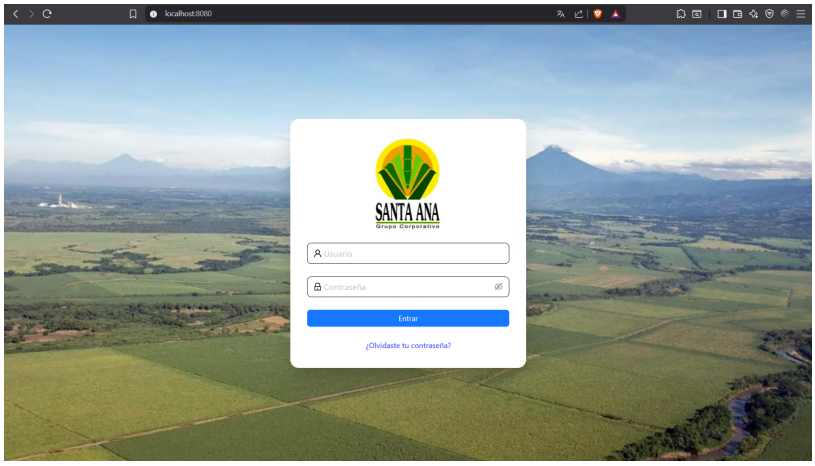


Figura 55: Contenido de la imagen de Docker desplegado exitosamente en *localhost*

Con la imagen Docker validada localmente, el siguiente paso fue configurar el entorno de despliegue en la nube. La herramienta seleccionada fue Google Cloud Run, esto debido a sus capacidades de autoescalado, facturación basada en uso con una capacidad de prueba gratis de hasta USD 300.00, y su facilidad de integración con otros servicios e infraestructura como código.

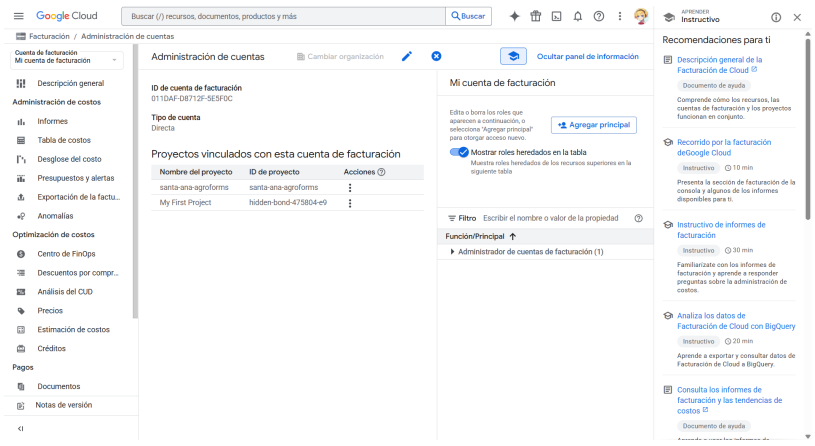


Figura 56: Página principal de Google Cloud Run con proyecto creado

Posteriormente, se procedió a realizar la configuración de Terraform para su integración con Google Cloud Run por medio de archivos de configuración los cuales se pueden observar en la sección 10.4 de este documento, definiendo así los recursos necesarios como el servicio de GCR, los permisos *IAM*, las políticas de *firewall*, y la configuración de tráfico en el proyecto. Especificando también la región de despliegue, la imagen del contenedor a utilizar y las políticas de escalado automático.

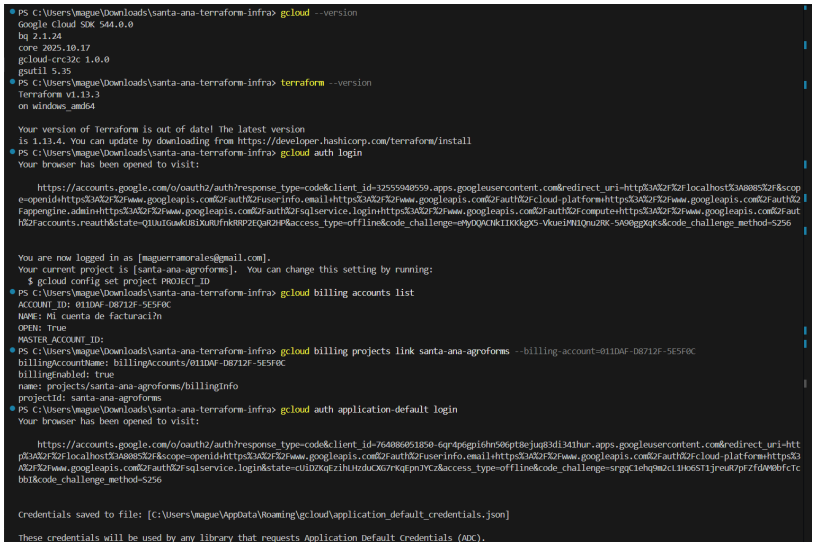


Figura 57: Configuración e integración de Google Cloud Run con Terraform

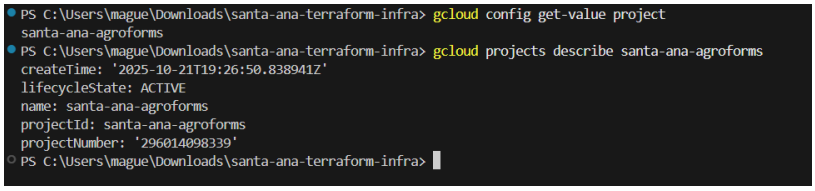


Figura 58: Configuración del proyecto de Cloud Run en la línea de comandos

Una vez se configuró exitosamente el *SDK* de Google Cloud y de Terraform y la vinculación de la cuenta utilizada para el despliegue automatizado, se procedió a hacer el despliegue de la infraestructura con Terraform, iniciándose la fase de inicialización, en donde se instala el proveedor HashiCorp/Google y genera un archivo de bloqueo para garantizar el uso de versiones consistentes en futuras ejecuciones.

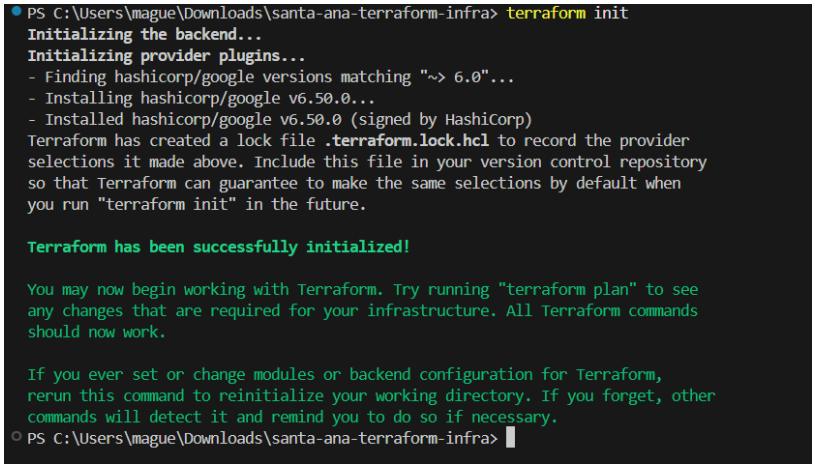


Figura 59: Inicialización de la infraestructura con Terraform

Después de esto, se realizó la ejecución de los comandos 'terraform validate' y 'terraform

plan' para verificar la validez de la configuración y generar un plan de ejecución. Mostrando así, los *outputs* configurados en Terraform, incluyendo la imagen del contenedor y la URL del servicio desplegado, siendo esto útil para consultar información sobre los recursos creados sin utilizar necesariamente la consola de Google Cloud.

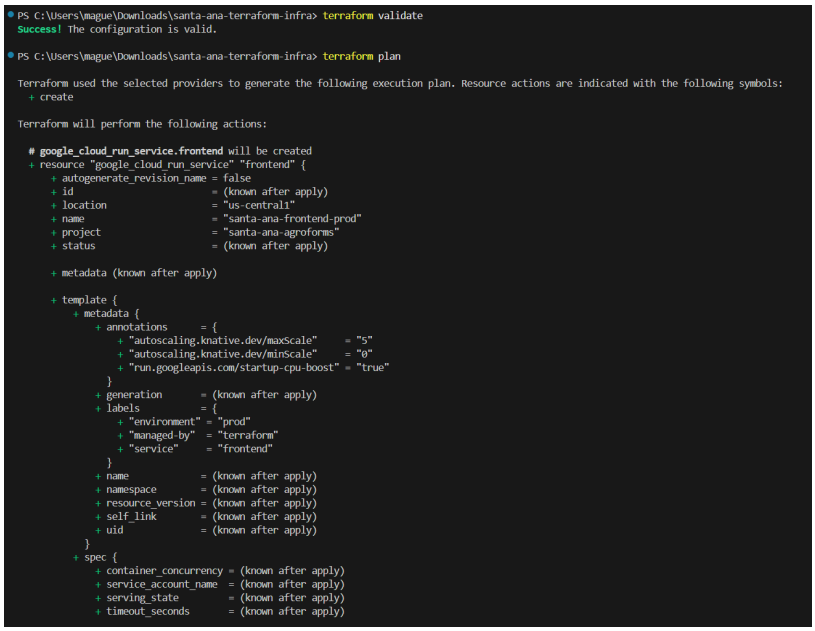


Figura 60: Validación y planificación de la infraestructura en Terraform

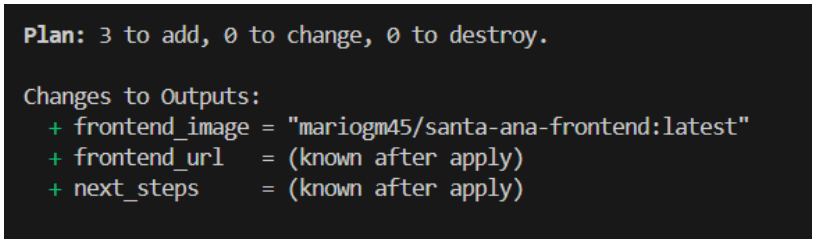
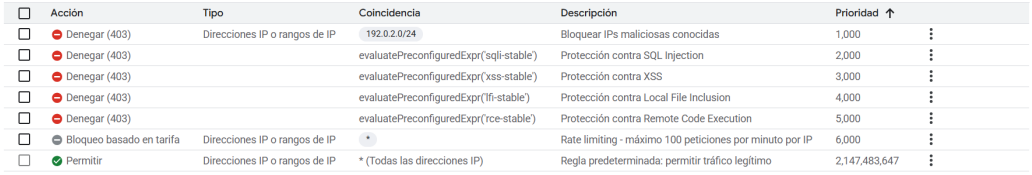


Figura 61: Outputs de la planificación de Terraform con la imagen y URL

Dentro de la planificación, se cuenta también con la forma en la cual se consideró también el endurecimiento del entorno y la prevención de ataques en el mismo. Se utilizaron dos capas de seguridad: Google Cloud Armor como un *web application firewall* (WAF) e *ingress controls*, como un control de tráfico entrante a nivel de servicio. Su implementación contó con seis reglas de protección contra los ataques más comunes según OWASP Top 10 y la configuración del mencionado *ingress controls* para la definición del tráfico permitido hacia el servicio.

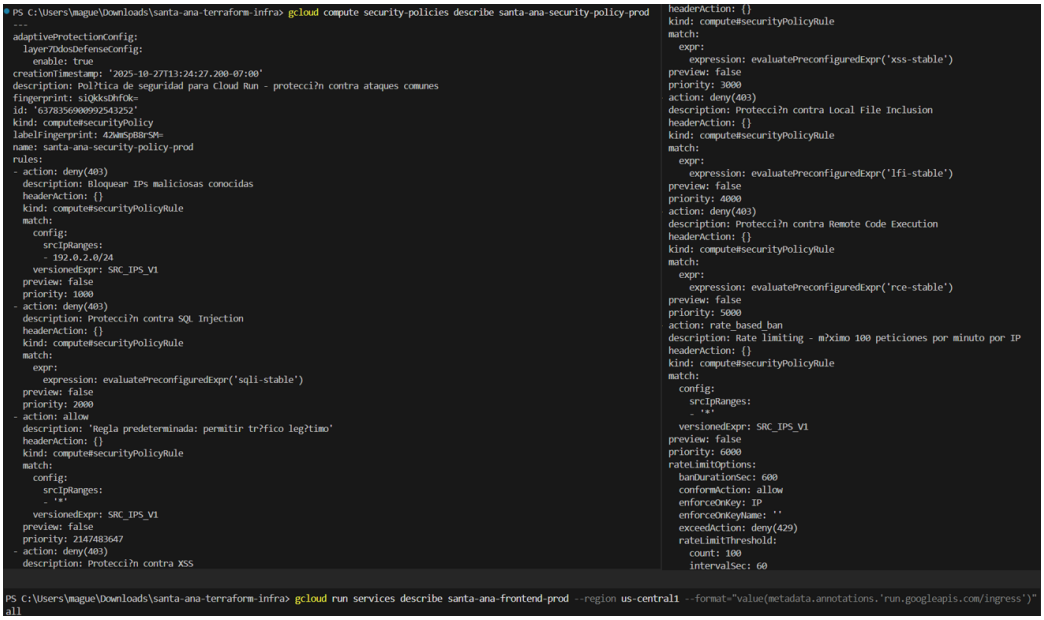
Las seis reglas para el WAF fueron: el bloqueo de IPs maliciosas, las cuales estén identificadas como fuentes de ataques; protección contra intentos de *SQL Injection*, protección contra *cross-site scripting*, protección contra *Local File Inclusion*, la cual evita acceso no autorizado a archivos del sistema, protección contra *Remote Code Execution*, que protege contra la ejecución de código malicioso que pueda comprometer completamente el servidor, *rate limiting* que limita el número de peticiones por dirección IP para prevenir ataques de fuerza

bruta y *DDoS*. Mientras que, por parte de *ingress controls*, se define una regla adicional en la cual permite todo el tráfico que no fue bloqueado por las reglas anteriores.



Acción	Tipo	Coincidencia	Descripción	Prioridad ↑
Denegar (403)	Direcciones IP o rangos de IP	192.0.2.0/24	Bloquear IPs maliciosas conocidas	1,000
Denegar (403)		evaluatePreconfiguredExpr('sqli-stable')	Protección contra SQL Injection	2,000
Denegar (403)		evaluatePreconfiguredExpr('xss-stable')	Protección contra XSS	3,000
Denegar (403)		evaluatePreconfiguredExpr('lfi-stable')	Protección contra Local File Inclusion	4,000
Denegar (403)		evaluatePreconfiguredExpr('rce-stable')	Protección contra Remote Code Execution	5,000
Bloqueo basado en tarifa	Direcciones IP o rangos de IP	*	Rate limiting - máximo 100 peticiones por minuto por IP	6,000
Permitir	Direcciones IP o rangos de IP	* (Todas las direcciones IP)	Regla predeterminada: permitir tráfico legítimo	2,147,483,647

Figura 62: Reglas del *web application firewall* e *ingress controls*



```
PS C:\Users\mague\Downloads\santa-ana-terraform-infra> gcloud compute security-policies describe santa-ana-security-policy-prod
---
adaptiveProtectionConfig:
  layer7DdosDefenseConfig:
    enable: true
creationTimestamp: '2025-10-27T13:24:27.200-07:00'
description: Pol?tica de seguridad para Cloud Run - protecci?n contra ataques comunes
fingerprint: siQkksDhfOk=
id: '6378356000092543252'
kind: compute#securityPolicy
labelFingerprint: 42WmSpB8rSM=
name: santa-ana-security-policy-prod
rules:
- action: deny(403)
  description: Bloquear IPs maliciosas conocidas
  headerAction: {}
  kind: compute#securityPolicyRule
  match:
    config:
      srcIpRanges:
      - 192.0.2.0/24
    versionedExpr: SRC_IPS_V1
  preview: false
  priority: 1000
- action: deny(403)
  description: Protecci?n contra SQL Injection
  headerAction: {}
  kind: compute#securityPolicyRule
  match:
    expr:
      expression: evaluatePreconfiguredExpr('sqli-stable')
  preview: false
  priority: 2000
- action: allow
  description: 'Regla predeterminada: permitir tr?fico leg?timo'
  headerAction: {}
  kind: compute#securityPolicyRule
  match:
    config:
      srcIpRanges:
      - '*'
    versionedExpr: SRC_IPS_V1
  preview: false
  priority: 2147483647
- action: deny(403)
  description: Protecci?n contra XSS
  headerAction: {}
  kind: compute#securityPolicyRule
  match:
    expr:
      expression: evaluatePreconfiguredExpr('xss-stable')
  preview: false
  priority: 3000
  action: deny(403)
  description: Protecci?n contra Local File Inclusion
  headerAction: {}
  kind: compute#securityPolicyRule
  match:
    expr:
      expression: evaluatePreconfiguredExpr('lfi-stable')
  preview: false
  priority: 4000
  action: deny(403)
  description: Protecci?n contra Remote Code Execution
  headerAction: {}
  kind: compute#securityPolicyRule
  match:
    expr:
      expression: evaluatePreconfiguredExpr('rce-stable')
  preview: false
  priority: 5000
  action: rate_based_ban
  description: Rate limiting - m?ximo 100 peticiones por minuto por IP
  headerAction: {}
  kind: compute#securityPolicyRule
  match:
    config:
      srcIpRanges:
      - '*'
    versionedExpr: SRC_IPS_V1
  preview: false
  priority: 6000
  rateLimitOptions:
    banDurationSec: 600
    conformAction: allow
    enforceOnKey: IP
    enforceOnKeyName: ''
    exceedAction: deny(429)
    rateLimitThreshold:
      count: 100
      intervalSec: 60

PS C:\Users\mague\Downloads\santa-ana-terraform-infra> gcloud run services describe santa-ana-frontend-prod --region us-central1 --format="value(metadata.annotations.'run.googleapis.com/ingress')"
all
```

Figura 63: Políticas de seguridad en consola

Finalmente, se realiza exitosamente la ejecución del comando 'terraform apply', en donde se solicita la confirmación para la creación de estos recursos según lo indicado por el plan de ejecución. Una vez se confirma esto, Terraform procede a crear los recursos en Google Cloud Run.

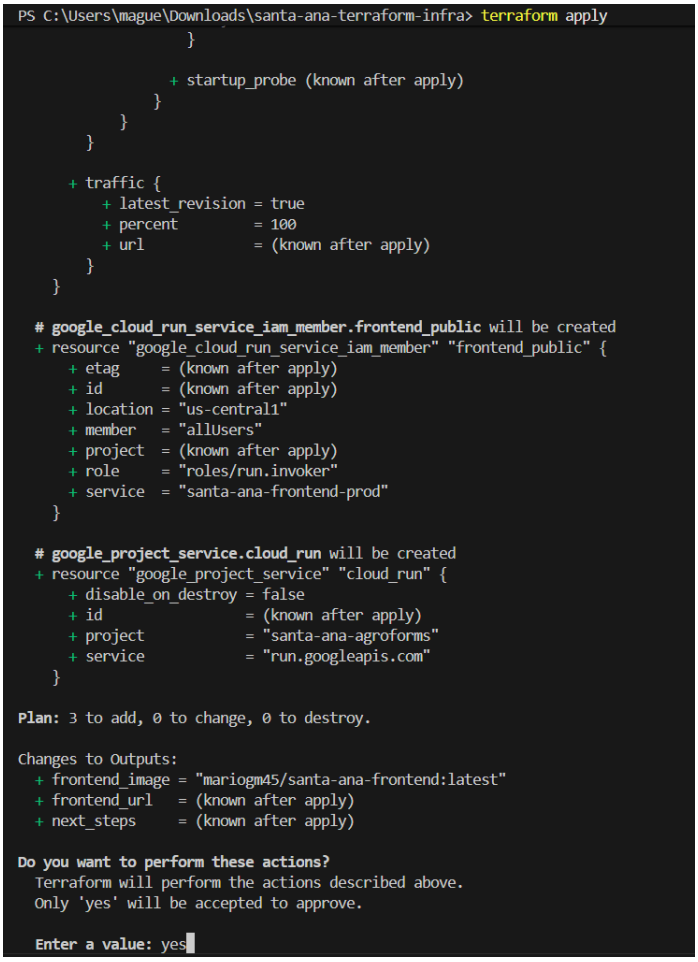


Figura 64: Ejecución del comando terraform apply para desplegar la infraestructura

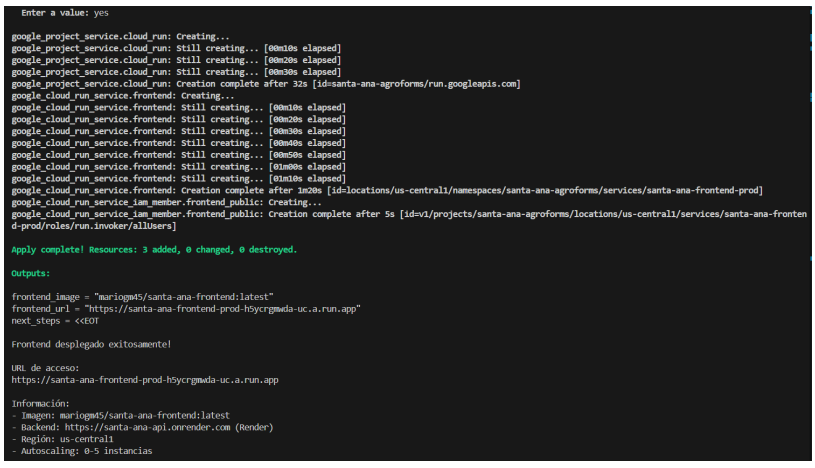


Figura 65: Resultados del comando terraform apply

CAPÍTULO 6

Resultados

6.1. Calidad y cobertura en el código

La implementación de prácticas de integración continua por medio de *workflows* en GitHub Actions ayudó a la automatización de diversas ejecuciones de pruebas y análisis de código tanto para la plataforma web como para la API. Los resultados de todo el desarrollo ocasionaron que estos flujos y *checks* pasaran exitosamente.

Figura 66: Resultados de los *workflows* en GitHub para el *frontend*

Figura 67: Resultados de los *workflows* en GitHub para el *backend*

Por parte de la *quality gate* de SonarCloud, definiendo una *quality gate* como un objetivo en un proyecto de tecnología que requiere que se cumplan criterios definidos antes de pasar a la siguiente fase (Bernstein, 2024). Se definió para fines del proyecto, una *quality gate* con métricas de A en mantenibilidad, confiabilidad y seguridad del código, un 65 % mínimo de cobertura en todo el repositorio y código nuevo, como también un código duplicado menor al 5 %. Con ello, los resultados en ambos repositorios fueron los siguientes:

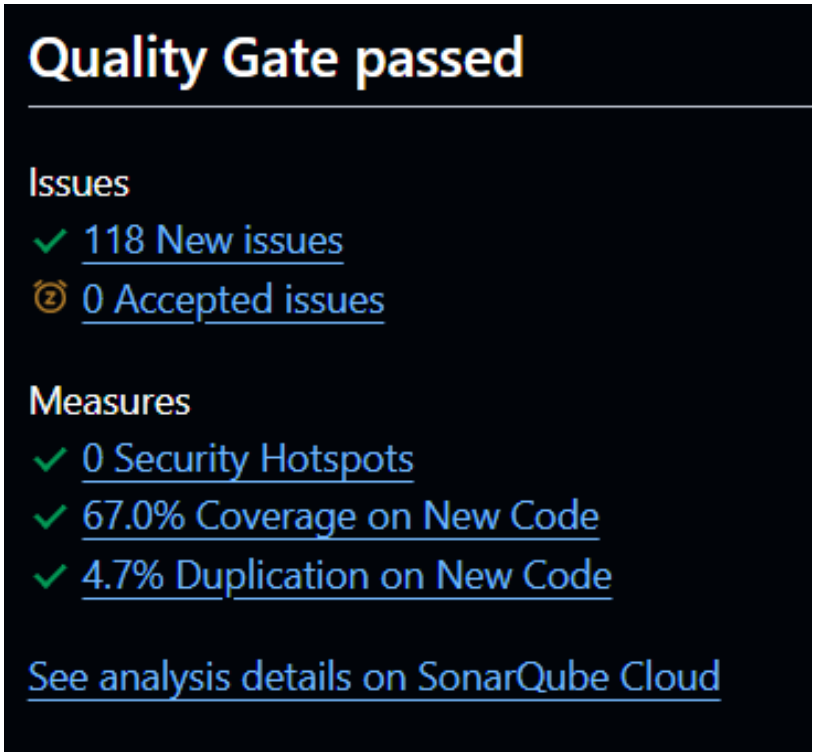


Figura 68: *Quality gate* superada para el repositorio del *frontend*

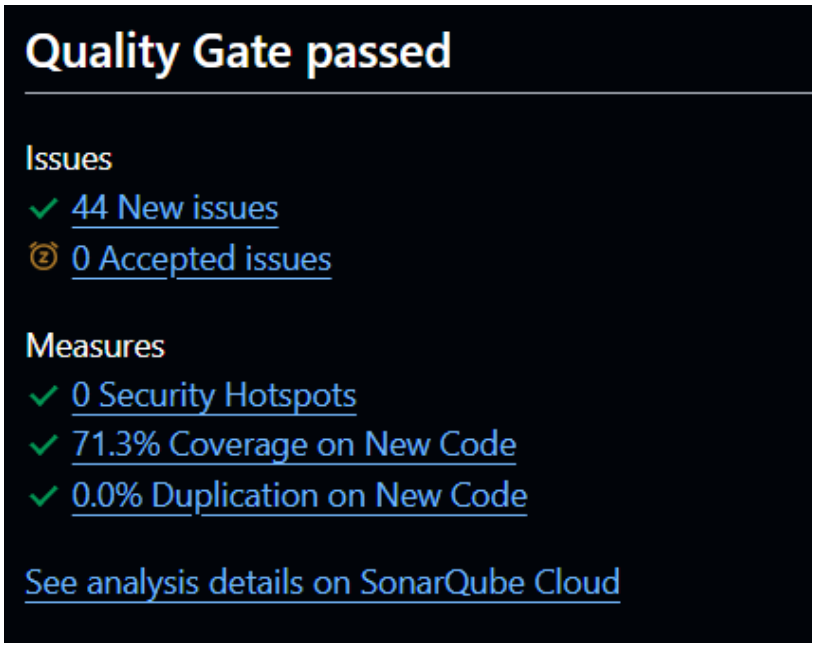


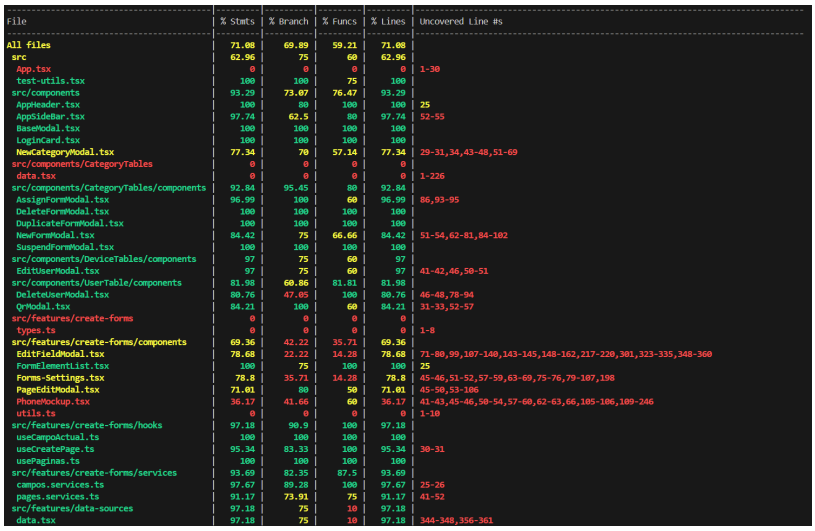
Figura 69: *Quality gate* superada para el repositorio del *backend*

El repositorio de la plataforma web logró un 67 por ciento en cobertura dentro del repositorio, con 4.7 por ciento de duplicación en el código nuevo, así como también 0 *security hotspots* en ello. Mientras que el repositorio de la API obtuvo las mismas calificaciones satisfactorias, únicamente con un mayor porcentaje de cobertura en el código nuevo y 0 por ciento de código duplicado. Incluyendo también en las métricas del proyecto, ambos obtuvieron calificaciones de A en mantenibilidad, confiabilidad o *reliability* y seguridad, lo que quiere decir que estos pasaron los criterios definidos para la calidad del código.

Figura 70: Métricas del repositorio del *frontend* en SonarCloud

Figura 71: Métricas del repositorio del *backend* en SonarCloud

En lo que se refiere específicamente a la cobertura de pruebas, se obtuvo un 71.08 % de cobertura en el código de los archivos estrictamente para el desarrollo de la página web con Jest, mientras que para la API de Django, se alcanzó una cobertura total de 84 %.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	71.08	69.89	59.21	71.08	
src	62.96	75	60	62.96	
App.tsx	0	0	0	0	1-30
test-utils.tsx	100	100	75	100	
src/components	93.29	73.07	76.47	93.29	
AppHeader.tsx	100	80	100	100	25
AppSideBar.tsx	97.74	62.5	80	97.74	52-55
BaseModal.tsx	100	100	100	100	
LoginCard.tsx	100	100	100	100	
NewCategoryModal.tsx	77.34	70	57.14	77.34	29-31,34,43-48,51-69
src/components/CategoryTables	0	0	0	0	
data.tsx	0	0	0	0	1-226
src/components/CategoryTables/components	92.84	95.45	80	92.84	
AssignFormModal.tsx	96.99	100	60	96.99	86,93-95
DeleteFormModal.tsx	100	100	100	100	
DuplicateFormModal.tsx	100	100	100	100	
NewFormModal.tsx	84.42	75	66.66	84.42	51-54,62-81,84-102
SuspendFormModal.tsx	100	100	100	100	
src/components/DeviceTables/components	97	75	60	97	
EditUserModal.tsx	97	75	60	97	41-42,46,50-51
src/components/UserTable/components	81.98	60.86	81.81	81.98	
DeleteUserModal.tsx	80.76	47.05	100	80.76	46-48,78-94
QrModal.tsx	84.21	100	60	84.21	31-33,52-57
src/features/create-forms	0	0	0	0	
types.ts	0	0	0	0	1-8
src/features/create-forms/components	69.36	42.22	35.71	69.36	
EditFieldModal.tsx	78.68	22.22	14.28	78.68	71-80,99,107-140,143-145,148-162,217-220,301,323-335,348-360
FormElementList.tsx	100	75	100	100	25
Forms-Settings.tsx	78.8	35.71	14.28	78.8	45-46,51-52,57-59,63-69,75-76,79-107,198
PageEditModal.tsx	71.01	80	50	71.01	45-50,53-106
PhoneMockup.tsx	36.17	41.66	60	36.17	41-43,45-46,50-54,57-60,62-63,66,105-106,109-246
utils.ts	0	0	0	0	1-10
src/features/create-forms/hooks	97.18	90.9	100	97.18	
useCampoActual.ts	100	100	100	100	
useCreatePage.ts	95.34	83.33	100	95.34	30-31
usePaginas.ts	100	100	100	100	
src/features/create-forms/services	93.69	82.35	87.5	93.69	
campos.services.ts	97.67	89.28	100	97.67	25-26
pages.services.ts	91.17	73.91	75	91.17	41-52
src/features/data-sources	97.18	75	10	97.18	
data.tsx	97.18	75	10	97.18	344-348,356-361

Figura 72: Cobertura del código de *frontend* de las pruebas realizadas con Jest

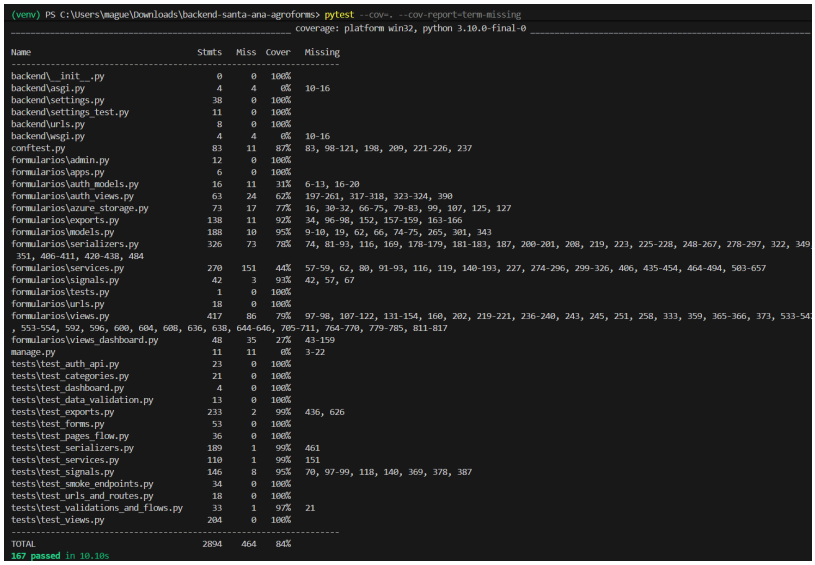


Figura 73: Cobertura del código de *backend* de las pruebas realizadas con Pytest

6.2. Vulnerabilidades y dependencias

Se presentan a continuación los resultados del análisis de dependencias en el código de ambos repositorios.

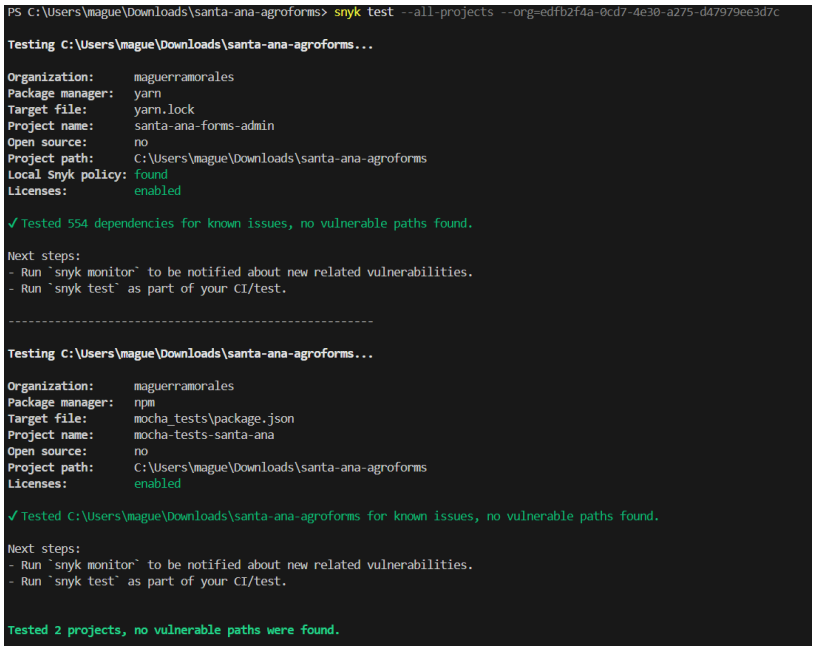


Figura 74: Resultado del escaneo de dependencias con Snyk en el código del *frontend*

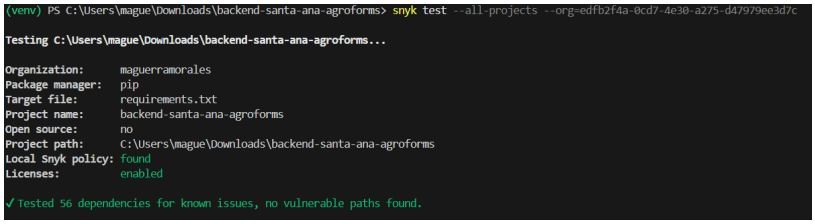


Figura 75: Resultado del escaneo de dependencias con Snyk en el código del *backend*

Como se pudo observar, ninguno de los dos repositorios cuenta con vulnerabilidades dentro de sus dependencias, y además, todas se encuentran actualizadas al momento del escaneo, demostrando así una gestión proactiva de las dependencias, en donde se mantienen actualizadas las versiones de los paquetes y se atienden las alertas de seguridad rápidamente.

6.3. Seguridad en ejecución

Los resultados de las pruebas de penetración automatizadas realizadas por ZAP contaron con un total de 4 alertas medias, 2 bajas y 3 informacionales por parte de la plataforma web y 2 alertas medias, 5 bajas y 4 informacionales de parte de la API. Se consideran en su mayoría excepciones debido a ciertas funcionalidades que presenta la plataforma para evitar algún inconveniente con los usuarios y su uso diario y al ser ambientes de desarrollo, estas se endurecen apropiadamente en producción.

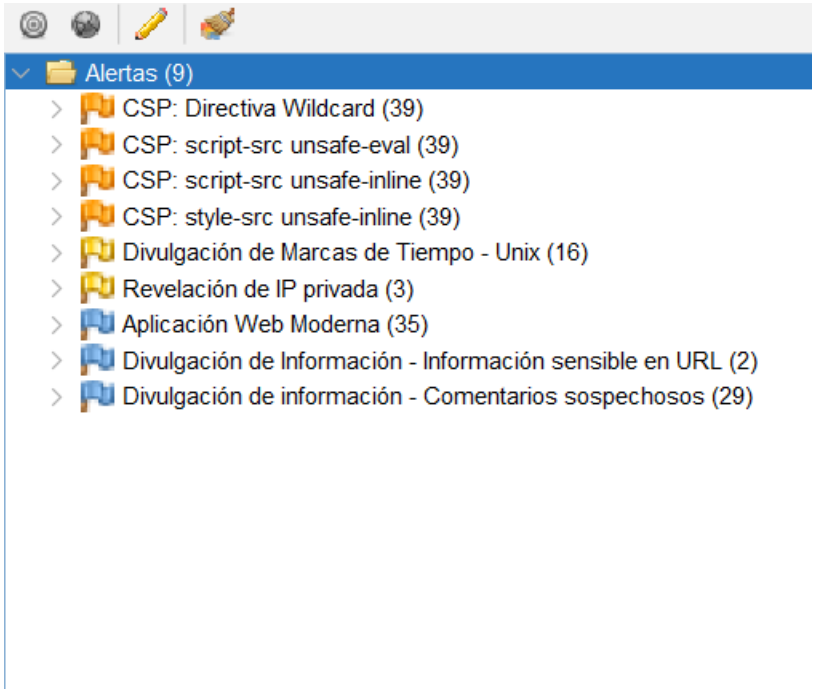


Figura 76: Resultado del análisis de vulnerabilidades con ZAP en la plataforma web

Por parte de las alertas medias en *frontend*, su causa raíz se enfoca en la configuración de desarrollo con directivas abiertas para facilitar recursos embedidos, estas se mitigan en producción por medio de políticas más restrictivas contemplando uso de *hashes* para *scripts*

legítimos, eliminación de '*' en *script-src* y *style-src*. Mientras que, las alertas bajas e informacionales, estas se deben a comentarios y líneas de código encontradas en las dependencias por parte de los paquetes instalados.

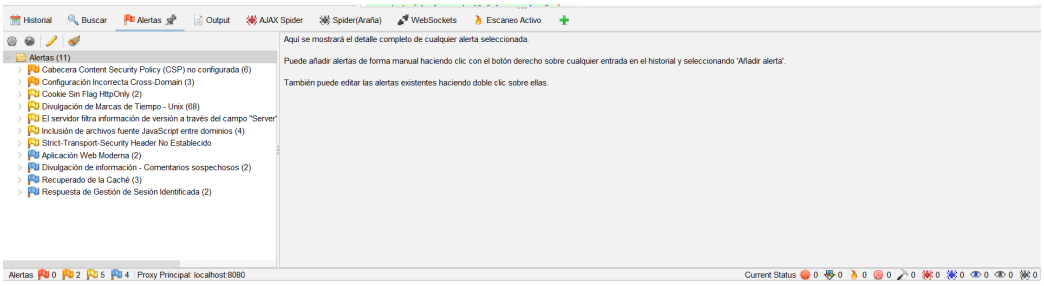


Figura 77: Resultado del análisis de vulnerabilidades con ZAP en la API

Por parte de las alertas medias en *backend*, se encuentra que CSP está ausente en algunos recursos del mismo, por lo que las acciones a considerar es de reforzar CSP desde la aplicación a todas las rutas públicas, verificando desde el despliegue. Adicionalmente, se incluye la configuración incorrecta de *cross-domain* en un recurso externo, con lo cual se acepta el riesgo al contar con controles adicionales para las dependencias.
Se incluye también, las matrices de riesgos para ambas partes del sistema.

Hallazgo	**Sev.**	**Afecta**	**Acción**	**Estado**
CSP permisiva: `unsafe-inline/eval`, comodines en `script-src/style-src`	Medio	Frontend	Endurecer CSP en prod: sin `unsafe-*`; usar `nonce`/hash; *allow-list* explícita en `script-src`, `style-src`, `img-src`, `connect-src`. Mantener CSP separada para dev.	Mitigado
Leak de IP privada en artefactos	Bajo	Dev (bundles/logs)	Excluir artefactos de dev en build prod; revisión de logs; minimizar & ofuscar.	Aceptado
Marcas de tiempo/metadata en estáticos	Bajo	Dev (bundles)	*Build hardened*: remover comentarios/metadata; minificar.	Aceptado
Información sensible en URL / comentarios	Info	Navegación / bundles	Evitar parámetros sensibles en query; remover comentarios en build de producción	Mitigado

Cuadro 8: Matriz de riesgos en *frontend*

Hallazgo	Sev.	Afecta	Acción	Estado
CSP ausente/débil en ciertos recursos	Medio	Backend dev	Asegurar envío de CSP para todas las rutas públicas; alinear con CSP de frontend (sin `unsafe-*`; `nonce`/hash; *allowlist*).	Mitigado
Recurso de tercero `cdn.jsdelivr.net` (*cross-domain*)	Medio	Cliente	Mantener con **SRI** & versión fijada; restringir orígenes vía CSP; documentar aceptación de riesgo residual del CDN confiable.	Aceptado
Cookie sin `HttpOnly`	Bajo	Sesión/CSRF	Marcar cookies sensibles con `HttpOnly`, `Secure`, `SameSite=Lax/Strict`; revisar si es necesaria en client-side.	Mitigado
HSTS ausente en una ruta	Bajo	Canal TLS (edge)	Habilitar **HSTS** en producción con ventana adecuada; evaluar `includeSubDomains`/*preload*.	Mitigado
Exposición de versión en `Server`	Bajo	Cabeceras	Estandarizar/anonimizar header `Server` en proxy/edge.	Mitigado
Comentarios/metadata en respuestas	Info	Respuestas	*Build hardened*: remover comentarios/metadata; minificar.	Mitigado

Cuadro 9: Matriz de riesgos en *backend*

ZAP reportó 0 alertas de alta severidad en *frontend* y en *backend*, indicando que no existen vulnerabilidades críticas explotables en la aplicación. En la sección 10.2, se incluyen los reportes elaborados por ZAP en los cuales se enlistan las vulnerabilidades y su justificación, así como también el plan de acción definido para cada una. Estos reportes fueron entregados al Ingenio Santa Ana para su documentación.

6.4. Entrega y despliegue

En los resultados de entrega y despliegue, se presenta el análisis de vulnerabilidades del contenedor realizado con Docker Desktop, como también el despliegue exitoso en el entorno de producción. Como se observa en la Figura 78, la imagen no presenta vulnerabilidades como tal, analizando un total de 32 capas con las que cuenta la imagen, contribuyendo a esto el uso de Alpine Linux como imagen base y la actualización regular de paquetes como se demostró en la sección 6.2.

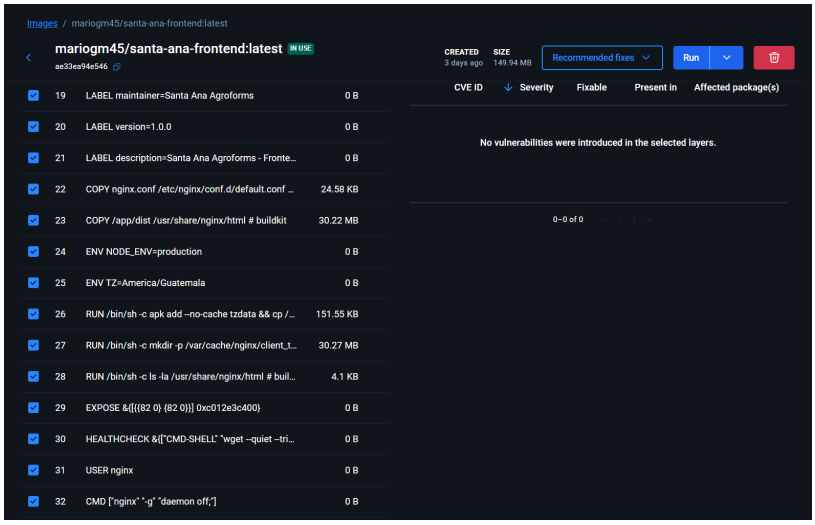


Figura 78: Análisis de vulnerabilidades del contenedor con Docker Desktop

A continuación, se muestra también la captura de pantalla de la URL de Google Cloud Run ejecutándose con normalidad una vez se ha iniciado la sesión.

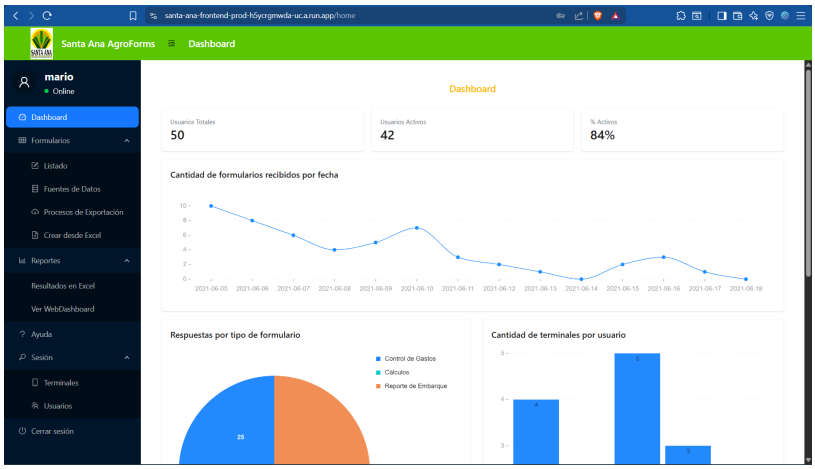


Figura 79: Captura de pantalla del proyecto en Google Cloud Run

Finalmente, en la Figura 80, se muestra el archivo YAML generado automáticamente para la descripción completa de la configuración del servicio desplegado. Este archivo sirve como documentación de la infraestructura y puede ser versionado con el código para realizar un registro de cambios en la configuración del servicio.

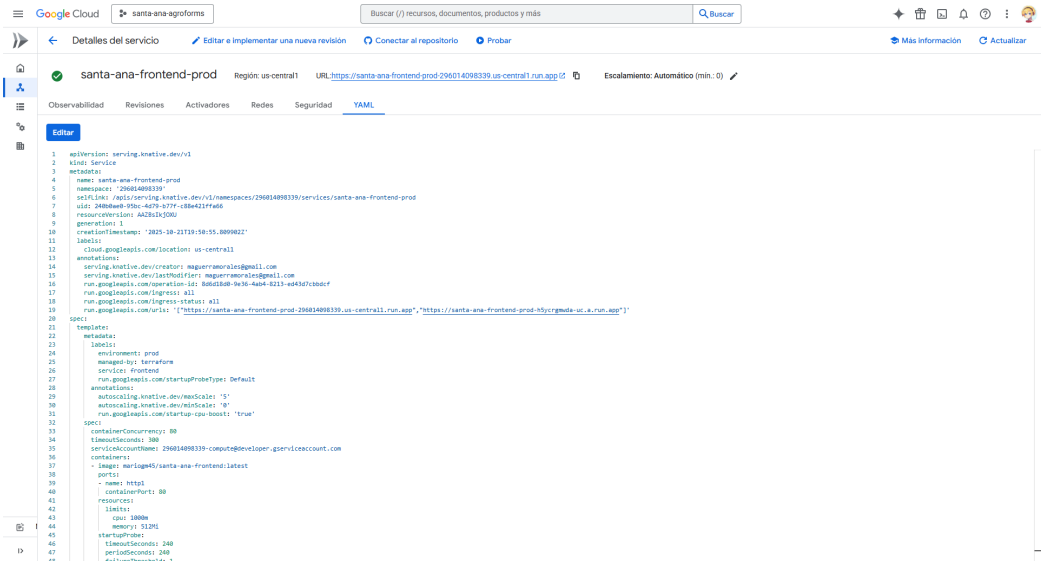


Figura 80: Captura de pantalla del service.yaml generado por Cloud Run del proyecto

CAPÍTULO 7

Conclusiones

- El módulo integró, a lo largo del desarrollo, prácticas de seguridad como controles de acceso, detección y prevención de vulnerabilidades y prácticas de aseguramiento de calidad como pruebas automatizadas y flujos de *CI/CD* con evidencias de análisis estático, dinámico, políticas de conexión y protección de la información, disciplina de integración y entrega continua y despliegue auditable.
- Se desarrollaron exitosamente mecanismos para autenticación, autorización, encriptación y protección de datos, así como también la gestión de identidades dentro del marco de ciberseguridad, fortaleciendo así la confidencialidad, integridad y disponibilidad requeridas para el sistema.
- Se integraron *pipelines* de escaneo de dependencias, análisis de código estático, dinámico y pruebas de penetración exitosamente, obteniendo resultados satisfactorios con *quality gates* superadas en mantenibilidad, confiabilidad y seguridad, cobertura efectiva del 71 % y 84 % en *frontend* y *backend* respectivamente, y sin vulnerabilidades ni alertas críticas en dependencias ni en el código, mitigando las alertas de seguridad media para producción mediante CSP estricta, SRI y encabezados.
- Se configuró e implementó un ecosistema de pruebas unitarias, integración, *end-to-end*, acoplado a la integración y entrega continua en todo el sistema, siendo un total de 490 pruebas realizadas, lo cual respalda la mantenibilidad en conjunto con flujos de integración y entrega continua y permite validación del código previa al despliegue, facilitando así la aplicación sistemática de buenas prácticas de aseguramiento de calidad.
- La entrega continua de la plataforma web contó con un *pipeline* confiable y observable, al no registrar vulnerabilidades y su despliegue documentado evidenció un camino de entrega auditable y robusto, siendo coherente con todo el enfoque de *DevSecOps*.

CAPÍTULO 8

Recomendaciones

- Se recomienda para futuras implementaciones profundizar en controles de acceso y protección de datos formalizando matrices versionadas de roles y permisos, como también realización de pruebas de autorización en integración continua, para reducir errores de permiso y riesgo de acceso indebido, facilitando auditorías y rotación de personal.
- Se recomienda endurecer progresivamente la *quality gate* de la plataforma, así como también todo el circuito de detección y prevención automatizando pruebas de penetración y definir políticas de bloqueo en casos de vulnerabilidades de severidad alta y crítica para evitar la deuda técnica y riesgo de regresiones que se puedan presentar, elevando así la confiabilidad del servicio.
- Se recomienda escalar la estrategia de aseguramiento de calidad manteniendo la base de las pruebas implementadas y priorizando los casos de riesgo altos, con el beneficio de maximizar el retorno de las pruebas sobre los flujos más críticos del servicio.
- Se recomienda consolidar métricas y alertas en los servicios de despliegue como parte de observabilidad y operación segura por la parte de la infraestructura para la detección temprana de fallas y tiempo de respuesta menor ante incidentes.
- Se recomienda mantener la capacitación *DevSecOps* como parte del ciclo de mejora continua en la plataforma, estableciendo revisiones trimestrales o cada cierto tiempo para priorizar remediaciones con criterios de riesgo y sostener las buenas prácticas y la gobernanza en el sistema, esto para mantener la postura de seguridad ante cambios de entorno.

CAPÍTULO 9

Bibliografía

ACL. (2024, julio). *¿Qué es QA? Descubre la Importancia del Quality Assurance.* https://www.aclti.com/es/blog/qu%C3%A9-es-qa-descubre-la-importancia-del-quality-assurance

Alcarria, P. (2024, abril). *Escaneo de vulnerabilidades. Herramientas y técnicas.* https://openwebinars.net/blog/escaneo-de-vulnerabilidades/

Amazon Web Services. (2024a). *¿Qué es DevSecOps?* https://aws.amazon.com/es/what-is/devsecops/

Amazon Web Services. (2024b). *¿Qué es la ciberseguridad?* https://aws.amazon.com/es/what-is/cybersecurity/

Anand, A. (2025, enero). *ITIL 4 Explained – ITIL 4 IT Service Management Practices.* https://itsm.tools/itil-4-explained/

Bernstein, C. (2024, agosto). *What is a quality gate?* https://www.techtarget.com/searchsoftwarequality/definition/quality-gate

Broadcom. (s. f.). *What is Platform Security?* https://www.broadcom.com/topics/platform-security

BrowserStack. (2025a). *Cross browser testing on desktop & mobile.* https://www.browserstack.com/live

BrowserStack. (2025b, mayo). *Selenium Automation Framework: A Detailed Guide.* https://www.browserstack.com/guide/selenium-framework

Bulbule, V. (2023, febrero). *Data Encryption techniques in Google Cloud (GMEK/CMEK/CSEK).* https://medium.com/google-cloud/data-encryption-techniques-in-google-cloud-gmek-cmek-csek-928d072a1e9d

Center for Internet Security. (2025). *About us.* https://www.cisecurity.org/about-us

Comité Nacional de Seguridad Cibernética. (2024, mayo). *Boletín Informativo 011-2024.* https://conciber.gob.gt/wp-content/uploads/2024/05/Boletin-011-2024.pdf

ComplianceQuest. (2025). *What Is Quality Assurance? A Guide to QA Importance and Benefits.* https://www.compliancequest.com/quality/what-is-quality-assurance/

Compunnel. (s. f.). *What is CI/CD in Quality Assurance? Benefits and Best Practices.* https://www.compunnel.com/blogs/continuous-integration-and-continuous-deployment-ci-cd-in-quality-assurance-qa/

Congreso de la República de Guatemala. (2025, agosto). *Iniciativa de Ley 6572. Ley de Protección de Datos Personales.* https://www.congreso.gob.gt/assets/uploads/info_legislativo/iniciativas/1c48c-6572.pdf

Contrast Security. (s. f.). *Vulnerability Scanning.* https://www.contrastsecurity.com/glossary/vulnerability-scanning

Cooper, B. (2023, agosto). *What is bcrypt and how to implement into your project?* https://medium.com/@brcooper247/what-is-bcrypt-and-how-to-implement-into-your-project-43ad9fec28a7

Covic, D. (2023, julio). *Burp Suite Overview.* https://medium.com/%40dancovic/burp-suite-overview-3401280d05a5

Cyber Tzar. (2023, octubre). *Defence in Depth, Security by Design.* https://cybertzar.com/defence-in-depth-security-by-design

Cypress. (s. f.). *Testing Frameworks for Javascript.* https://www.cypress.io/

Davis, M. (2024, enero). *What is Data Privacy Compliance and How Can You Achieve It.* https://www.osano.com/articles/data-privacy-compliance

DigitalGuardian. (2023, mayo). *Protección de Datos: Datos en Tránsito vs. Datos en Descanso.* https://www.digitalguardian.com/blog/data-protection-data-in-transit-vs-data-at-rest

Docker Inc. (2025). *What is a container?* https://www.docker.com/resources/what-container/

Dushevin, V. (2024, noviembre). *How to Improve QA Process: Best Practices.* https://luxequality.com/blog/qa-process-improvemen/

Enzymejs. (s. f.). *Introduction – Enzyme.* https://enzymejs.github.io/enzyme/

F5. (2025a). *¿Qué es la seguridad de aplicaciones web?* https://www.f5.com/es_es/glossary/web-application-security

F5. (2025b). *¿Qué es OpenSSL?* https://www.f5.com/es_es/glossary/openssl

Forrest, A., & Kosinski, M. (2024, marzo). *¿Qué es la gestión de identidades y accesos?* https://www.ibm.com/es-es/topics/identity-access-management

Fortinet. (2025a). *¿Qué es el marco MITRE ATTACK?* https://www.fortinet.com/lat/resources/cyberglossary/mitre-attck

Fortinet. (2025b). *¿Qué es la seguridad de la información?* https://www.fortinet.com/lat/resources/cyberglossary/information-security

Fortinet. (2025c). *¿Qué es SAML? ¿Cómo funciona la autenticación SAML?* https://www.fortinet.com/lat/resources/cyberglossary/saml

Foundeo Inc. (2023). *Content Security Policy (CSP) Quick Reference Guide.* https://content-security-policy.com/

García, F. (2025, julio). *¿Qué es OWASP y cómo usar esta metodología?* https://www.arsys.es/blog/owasp

GeeksForGeeks. (2025a, julio). *Getting Started with Pytest.* https://www.geeksforgeeks.org/python/getting-started-with-pytest/

GeeksForGeeks. (2025b, julio). *Introduction to Mocha.* https://www.geeksforgeeks.org/javascript/introduction-to-mocha/

GitHub. (2025). *Entender las GitHub Actions.* https://docs.github.com/es/actions/get-started/understand-github-actions

GitLab. (2025). *Qué es el control de versiones.* https://about.gitlab.com/es/topics/version-control/

Google. (s. f.). *Google Identity | Google for Developers.* https://developers.google.com/identity?hl=es-419

Gordon, B. (2025, junio). *What is Infrastructure as Code (IaC)?* https://www.harness.io/harness-devops-academy/what-is-infrastructure-as-code-iac

HashiCorp. (s. f.). *Terraform.* https://www.terraform.io/

IBM. (2024, diciembre). *What is the software development life cycle?* https://www.ibm.com/think/topics/sdlc

Indeed Editorial Team. (2025, marzo). *What is quality assurance and why is it so important?* https://uk.indeed.com/career-advice/career-development/what-is-quality-assurance

Infosecurity México. (2023, mayo). *Seguridad de la información y ciberseguridad: desafíos y oportunidades para las empresas.* https://www.infosecuritymexico.com/es/blog/seguridad-de-la-informacion-y-ciberseguridad.html

International Organization for Standardization. (2022). *ISO/IEC 27001:2022 – Information Security Management Systems.* https://www.iso.org/es/norma/27001

International Organization for Standardization. (2023). *ISO/IEC 25010:2023 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model.* https://www.iso.org/es/contents/data/standard/07/81/78176.html

ITIL. (2020). *ITIL 4: Las mejores prácticas en gestión de servicios de TI.* https://www.itil.com.mx/

Kansara, H. (2025). *QA in CI/CD Pipeline: Best Practices for Continuous Integration and Testing.* https://marutitech.com/qa-in-cicd-pipeline/

Kaspersky. (2025). *¿Qué es la seguridad en la nube?* https://latam.kaspersky.com/resource-center/definitions/what-is-cloud-security

Khan, I. A., & Singh, R. (2012, julio). *Quality Assurance And Integration Testing Aspects In Web Based Applications.* https://arxiv.org/abs/1207.3213

Kulkarni, S. (2023, noviembre). *DevOps Tool: Jenkins (CI/CD).* https://medium.com/%40mesagarkulkarni/devops-tool-jenkins-ci-cd-a942b7b53876

LDAP. (s. f.). *Learn About LDAP.* https://ldap.com/learn-about-ldap/

Mann, S. (2025). *¿Qué es la norma ITIL 4?* https://www.manageengine.com/latam/service-desk/itsm/que-es-la-norma-itil-4.html

Microsoft. (2025, agosto). *¿Qué es Microsoft Entra?* https://learn.microsoft.com/es-es/entra/fundamentals/what-is-entra

Murali, H. (2024, septiembre). *Your QA can now be quantified: A practical guide on how to measure QA and its business impacts.* https://blog.aspiresys.com/testing/your-qa-can-now-be-quantified-a-practical-guide-on-how-to-measure-qa-and-its-business-impacts/

National Institute of Standards and Technology. (2024, febrero). *The NIST Cybersecurity Framework 2.0.* https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.29.pdf

Nmap. (s. f.). *Chapter 15. Nmap Reference Guide.* https://nmap.org/book/man.html

Nutanix. (2025). *¿Qué es la seguridad de aplicaciones?* https://www.nutanix.com/es/info/what-is-application-security

Observatorio Guatemalteco de Delitos Informáticos. (2024). *El estado de la ciberdelincuencia en Guatemala.* https://ogdi.org/archivos/13125

Okta. (2025a). *¿Qué es OAuth 2.0?* https://auth0.com/es/intro-to-iam/what-is-oauth-2

Okta. (2025b). *Auth0 Overview.* https://auth0.com/docs/get-started/auth0-overview

Okta. (2025c, marzo). *¿Qué es Okta y qué hace Okta?* https://support.okta.com/help/s/article/what-is-okta?language=en_US

OpenJS Foundation. (s. f.). *Jest – Delightful JavaScript Testing.* https://jestjs.io/

OWASP Foundation. (2025). *OWASP Top Ten.* https://owasp.org/www-project-top-ten/

Patni, A. (2024, junio). *Maximizing DevSecOps ROI: 6 Key Benefits You Can't Ignore.* https://www.practical-devsecops.com/maximizing-devsecops-roi-6-key-benefits-you-cant-ignore/

Perry, M. (2024, mayo). *What is Developer Experience and Why It Matters?* https://www.qovery.com/blog/what-is-developer-experience-devex-and-why-it-matters/

Popat, M. (2025, marzo). *How to Secure Data in Transit and at Rest in a Hybrid Cloud Model?* https://mihirpopat.medium.com/how-to-secure-data-in-transit-and-at-rest-in-a-hybrid-cloud-model-7bc533cc9a37

PowerData. (2025). *GDPR: Lo que debes saber sobre el reglamento general de protección de datos.* https://www.powerdata.es/gdpr-proteccion-datos

Rapid7. (s. f.). *Metasploit Framework | Metasploit Documentation.* https://docs.rapid7.com/metasploit/msf-overview/

Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). *Zero Trust Architecture.* https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf

Runcheva, N. (2022, diciembre). *10 Guiding Principles for Effective E2E Test Automation in CI/CD.* https://www.testdevlab.com/blog/10-guiding-principles-for-effective-e2e-test-automation-in-ci-cd

Saini, P. (2025). *LambdaTest | NewRelic.* https://newrelic.com/instant-observability/lambdatest

Sentrio. (2021, diciembre). *¿Qué es SonarQube?: Verifica y analiza la calidad de tu código.* https://sentrio.io/blog/que-es-sonarqube/

Shwartz, I. (2023, mayo). *Kubernetes Rancher: The Basics and a Quick Tutorial.* https://komodor.com/learn/kubernetes-rancher-the-basics-and-a-quick-tutorial/

Snyk Limited. (2025). *Plataforma de seguridad para desarrolladores impulsada por IA de Snyk | Plataforma de seguridad y herramienta de AppSec con tecnología de IA | Snyk.* https://snyk.io/es/

Szahidewicz, D. (2025, mayo). *QA Automation – Full Guide to QA Test Automation.* https://bugbug.io/blog/test-automation/qa-automation/

Tailscale. (2025, mayo). *What is Tailscale?* https://tailscale.com/kb/1151/what-is-tailscale

Testlio. (2024, junio). *QA Testing Best Practices.* https://testlio.com/blog/qa-testing-best-practices/

The Kubernetes Authors. (2024, septiembre). *Overview | Kubernetes.* https://kubernetes.io/docs/concepts/overview/

The MITRE Corporation. (2025). *MITRE ATT&CK.* https://attack.mitre.org/

ThinkCloudly. (2025, septiembre). *Integrating DevSecOps into Infrastructure as Code for Stronger Security.* https://thinkcloudly.com/blog/integrating-devsecops-into-infrastructure-as-code-for-stronger-security/

Truffle Security Co. (2025). *What is TruffleHog?* https://trufflesecurity.com/trufflehog

Universidad de San Marcos. (2024). *¿Cuáles son los principios de la Seguridad Informática?* https://www.usanmarcos.ac.cr/blogs/cuales-son-los-principios-de-la-seguridad-informatica

Villalba, F. (2023, septiembre). *What is Developer Experience?* https://www.opslevel.com/resources/devex-series-part-1-what-is-devex

VMWare. (2025). *¿Qué es la seguridad en infraestructura de red?* https://www.vmware.com/topics/network-infrastructure-security

WireGuard. (2022). *WireGuard: fast, modern, secure VPN tunnel.* https://www.wireguard.com/

Wiz Experts Team. (2024, noviembre). *10 Cloud Security Standards Explained: ISO, NIST, CSA and More.* https://www.wiz.io/academy/cloud-security-standards

Wnpower. (2024, mayo). *¿Qué es Cloudflare y cuáles son sus beneficios?* https://www.wnpower.com/blog/que-es-cloudflare-beneficios/

Xyleni. (2025, mayo). *¿Cuáles son las fases del ciclo de vida del software?* https://xygeni.io/es/blog/what-are-the-phases-of-software-development-life-cycle/

ZAP. (2025). *ZAP – Getting Started.* https://www.zaproxy.org/getting-started/

ZeroThreat. (2024, diciembre). *Estadísticas sobre Ciberseguridad 2025: Descubriendo Insights Behind The Numbers.* https://zerothreat.ai/blog/cybersecurity-statistics-and-facts

CAPÍTULO 10

Anexos

10.1. Archivos YAML de GitHub Actions

Los archivos YAML con los *workflows* fueron subidos en su totalidad a este Gist a continuación:

Flujos de trabajo en repositorios GitHub de frontend y backend.

10.2. Reportes completos de seguridad

Los reportes generados por OWASP ZAP se subieron a la plataforma de Google Drive en formato PDF, para permitir su lectura de mejor manera.

Reporte de escaneo automatizado de vulnerabilidades en plataforma web.

Reporte de escaneo automatizado de vulnerabilidades en plataforma web.

10.3. Evidencias de despliegue

Los archivos utilizados para la creación del contenedor de la aplicación, con excepción de los archivos .dockerignore y env.docker, se encuentran en este Gist para su documentación.

Archivos utilizados para contenedorización de plataforma web.

10.4. Evidencias de infraestructura como código

La infraestructura realizada con Terraform y Google Cloud Run se encuentra en el Gist que se proporciona a continuación, contando con el archivo principal, los archivos de variables, y un README.md para una mayor documentación de esta parte del módulo.

Archivos utilizados para infraestructura como código.