Witness: Software de reportabilidad para condiciones de mantenimiento de maquinaria industrial

Memoria presentada para la obtención de título de Ingeniero civil Informático

POR

JOAQUÍN MATÍAS NAYEN JARA

Profesor Patrocinante: Geoffrey Hecht
Ingenieros supervisores: Alejandro Sánchez
Rodrigo Vergara

Octubre de 2024, Concepción

Índice

1. Introducción

La Industria 4.0, un proyecto originado en Alemania representa una revolución en la manufactura al impulsar la digitalización, promoviendo la automatización y la interacción humano-máquina. Como describe Gizem Erboz [1], este enfoque integra tecnologías

avanzadas como el Internet de las Cosas (IoT), la inteligencia artificial (IA), el big data y la computación en la nube, creando sistemas de manufactura inteligentes y autónomos. Estos sistemas no solo optimizan procesos y soluciones comerciales, sino que también redefinen los modelos de negocio futuros basados en una mayor eficiencia y eficacia (Erboz 2017).

En este contexto, la empresa SIMPRO se ha enfocado en el desarrollo de su suite Total Condition Monitoring System (TCMS), diseñada para implementar los principios de la Industria 4.0 en el mantenimiento predictivo de maquinaria industrial. El TCMS utiliza tecnologías avanzadas para monitorear y analizar el estado de los equipos en tiempo real, anticipando fallas y optimizando el rendimiento.

En la industria en general, un plan de mantenimiento predictivo robusto es esencial para asegurar la operatividad y eficiencia de la maquinaria. Esta estrategia permite detectar problemas en equipos críticos antes de que se conviertan en fallas graves, minimizando así los tiempos de inactividad costosos y mejorando la disponibilidad de la maquinaria. Esto no solo contribuye a aumentar la seguridad en el lugar de trabajo al prevenir fallas catastróficas, sino que también reduce significativamente los costos operativos y maximiza la productividad, asegurando la continuidad operativa y la rentabilidad de las operaciones industriales.

La implementación de la Industria 4.0 en el mantenimiento industrial permite la recopilación y análisis de grandes volúmenes de datos, facilitando la toma de decisiones informadas y basadas en datos. Esto mejora la gestión de los recursos y la eficiencia operativa, permitiendo que las operaciones sean más sostenibles y menos propensas a interrupciones inesperadas. La capacidad de monitorear continuamente las condiciones de los equipos y realizar ajustes en tiempo real asegura que las operaciones sean más eficientes y seguras.

La visión de la Industria 4.0 incluye varios pilares clave que son esenciales para su implementación exitosa. Entre estos pilares se encuentran el big data y la analítica, los robots autónomos, la simulación, la integración horizontal y vertical de sistemas, el IoT, la computación en la nube, la fabricación aditiva, la realidad aumentada y la ciberseguridad (Erboz 2017). Cada uno de estos componentes juega un papel crucial en la creación de fábricas inteligentes y en la transformación de procesos industriales tradicionales en sistemas altamente automatizados y conectados.

En conclusión, la digitalización y la automatización impulsadas por la Industria 4.0 ofrecen a la industria herramientas poderosas para mejorar sus operaciones. Empresas como SIMPRO están a la vanguardia de esta transformación, desarrollando soluciones

innovadoras como el TCMS que integran estas tecnologías para optimizar el mantenimiento predictivo. Este avance no solo tiene el potencial de transformar la eficiencia y la seguridad en el mantenimiento industrial, sino que también establece un nuevo estándar para la industria en términos de rentabilidad y sostenibilidad.

2. Marco Teórico

Para poder entender de mejor manera este documento es vital entender algunos conceptos previos.

2.1 La industria 4.0

La Industria 4.0 se refiere a la integración de tecnologías digitales avanzadas en los procesos de producción y manufactura, transformando los métodos tradicionales de

trabajo hacia un modelo inteligente e interconectado. También conocida como la Cuarta Revolución Industrial, esta evolución combina la Internet de las Cosas (IoT), la inteligencia artificial (IA), el big data, la robótica avanzada y la computación en la nube para crear sistemas de producción más autónomos y eficientes (Schwab, 2016) [4].

El concepto de Industria 4.0 surgió en Alemania como parte de una estrategia gubernamental para mejorar la competitividad del sector manufacturero. Este enfoque conecta todas las fases de la producción —desde el diseño hasta la distribución—, permitiendo una comunicación en tiempo real entre máquinas, sistemas y personas (Kagermann, Wahlster, & Helbig, 2013) [5]. La conectividad permite a las empresas adaptarse rápidamente a las demandas del mercado, personalizar productos y optimizar el uso de los recursos.

En el contexto de la Industria 4.0, la integración de sensores, sistemas ciberfísicos y análisis de datos avanzados permite identificar patrones, mejorar la toma de decisiones y minimizar el tiempo de inactividad. Estos sistemas se caracterizan por su capacidad de autoajuste, lo que permite una gestión más eficiente y proactiva del mantenimiento y del ciclo de vida de los equipos (Lee, Bagheri, & Kao, 2015) [6].

2.2 LogBook

LogBook es una aplicación de software desarrollada por SIMPRO como parte de su suite TCMS (Total Condition Monitoring System). Su principal objetivo es optimizar la gestión de activos y el registro de eventos de mantenimiento en la industria minera, con especial enfoque en la gran minería. La aplicación facilita la recopilación y almacenamiento de información técnica y de eventos de mantenimiento relacionados con diversos equipos, subunidades y componentes de una planta minera.

Como describe Matías Cáceres [3], la jerarquía en LogBook permite organizar la información de equipos y componentes en una planta minera, facilitando así su gestión. Esta estructura se basa en la norma ISO 14224:2016 y utiliza los niveles 4 al 9 de la clasificación taxonómica propuesta en la sección 8.2 de dicha norma. Los 6 niveles implementados en LogBook son los siguientes:

- **Planta (Nivel 4):** Representa la información básica de una planta minera, incluyendo código, nombre, categoría y descripción. Este es el nivel más alto de la jerarquía en LogBook.

- **Sección (Nivel 5):** Describe las distintas secciones dentro de una planta, como chancado primario, secundario o terciario. Cada sección está asociada a una planta específica.
- **Equipo (Nivel 6):** Representa los equipos dentro de una sección, según la tabla 5 de la norma ISO 14224:2016 ("Datos comunes de equipos para todas las clases de equipo"). Incluye detalles generales como nombre, tipo, fabricante, modelo y estado.
- **Subunidad (Nivel 7):** Describe las subunidades que componen un equipo, como el sistema motriz de un harnero. Este nivel también sigue en parte la tabla 5 de la norma ISO 14224:2016.
- **Componente (Nivel 8):** Representa los componentes de una subunidad, como el motor 1 del sistema motriz, y sigue en parte la tabla 5 de la norma ISO 14224:2016.
- **Parte (Nivel 9):** Detalla las distintas partes que conforman un componente, como mangueras o tuercas, y se basa parcialmente en la tabla 5 de la norma ISO 14224:2016.

La relación entre estos niveles se organiza en una estructura de padre-hijo, donde cada elemento de un nivel inferior pertenece exclusivamente a un elemento del nivel superior. Por ejemplo, una parte pertenece a un componente, que a su vez pertenece a una subunidad, y así sucesivamente hasta llegar a la planta.

3. Antecedentes y problemática

3.1. Descripción del problema

En el contexto del desarrollo del Total Condition Monitoring System (TCMS) por parte de SIMPRO, surge una problemática significativa con una de las aplicaciones que componen esta suite: RCM Tool. Esta plataforma, desarrollada en Power Apps, está diseñada para la gestión de reportes de inspecciones técnicas de mantenimiento, pero presenta serias limitaciones que afectan su funcionalidad e integración con la arquitectura del resto de la suite TCMS.

RCM Tool no cumple con los estándares de integración requeridos, lo que genera un entorno fragmentado y dificulta el flujo de datos y procesos. Esta desconexión reduce la eficiencia operativa y la colaboración dentro del sistema. Además, se proyecta un aumento

en la demanda de la plataforma, lo que presenta un desafío considerable en términos de mantenimiento y escalabilidad. La solución actual no está preparada para manejar un incremento en la carga de usuarios y datos, lo que podría llevar a problemas de rendimiento y tiempos de inactividad.

Por estas razones, se hace evidente la necesidad de rehacer la plataforma RCM Tool. La nueva versión, denominada Witness, se desarrollará para integrarse perfectamente con la arquitectura del TCMS y será escalable y fácil de mantener, asegurando un rendimiento óptimo en el futuro.

En resumen, rehacer RCM Tool es crucial para superar las limitaciones actuales, mejorar la integración con la suite TCMS y enfrentar los desafíos de escalabilidad y mantenimiento, garantizando la eficiencia y sostenibilidad del sistema de monitoreo y mantenimiento de SIMPRO.

3.1.1 RCM Tool

RCM Tool es una aplicación desarrollada en Power Apps para la gestión de reportes de inspecciones técnicas de mantenimiento. Esta herramienta permite registrar la condición de los equipos, generar avisos basados en estas condiciones y crear reportes detallados. Los reportes pueden ser enviados a una lista de difusión específica, facilitando la comunicación y coordinación entre los equipos de mantenimiento.

Figura 1. Pantalla Principal de RCM Tool

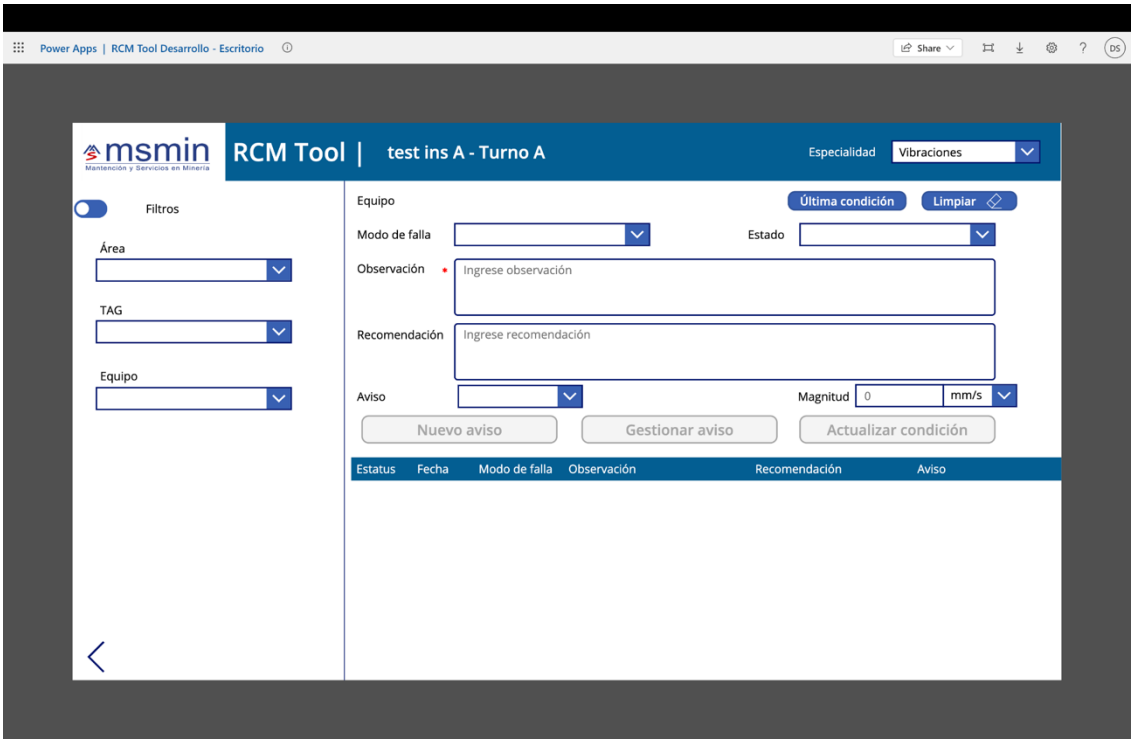


Figura 2. Pantalla de Registro de Condiciones de RCM Tool

3.2 Limitaciones y contexto

3.2.1 Limitaciones de RCM Tool

Aunque RCM Tool ha demostrado ser una herramienta útil para la gestión de reportes de inspecciones técnicas de mantenimiento, presenta varias limitaciones significativas:

Limitaciones de Power Apps:

- **Capacidad de Personalización**: Power Apps, aunque potente, puede tener limitaciones en cuanto a la personalización y flexibilidad comparado con plataformas de desarrollo más robustas. Esto puede restringir la capacidad de adaptar la herramienta a necesidades específicas que puedan surgir con el tiempo.
- **Integración con Otros Sistemas**: La integración de Power Apps con otros sistemas y aplicaciones puede ser compleja y limitada, lo que dificulta la creación de un entorno cohesivo dentro del Total Condition Monitoring System (TCMS).

Aumento de Demanda:

- **Escalabilidad**: La plataforma actual puede no ser capaz de manejar eficientemente un incremento significativo en la carga de usuarios y datos. Esto podría resultar en problemas de rendimiento, como tiempos de respuesta lentos y posibles tiempos de inactividad.
- **Rendimiento**: Con el aumento de la demanda, es probable que surjan problemas de rendimiento que afecten la experiencia del usuario y la eficiencia operativa de la aplicación.

Requerimientos del Software:

- **Mantenimiento**: A medida que el sistema crece, la necesidad de un mantenimiento más complejo y frecuente podría superar las capacidades actuales de Power Apps, requiriendo una reingeniería significativa.
- **Seguridad**: La seguridad de los datos y la protección contra amenazas cibernéticas son aspectos críticos que podrían no estar completamente cubiertos con la solución actual, especialmente con un aumento en la cantidad de datos y usuarios.

3.2.2 Contexto

El desarrollo de RCM Tool se enmarca en el esfuerzo de SIMPRO por crear un sistema de monitoreo integral bajo la suite Total Condition Monitoring System (TCMS). Este sistema tiene como objetivo mejorar la eficiencia operativa y la seguridad en el mantenimiento de maquinaria industrial mediante la digitalización y el análisis avanzado de datos.

La Industria 4.0, con su enfoque en la digitalización, la automatización y la interconexión, ofrece un contexto ideal para la implementación de soluciones como RCM Tool. Sin embargo, para alinearse completamente con los principios de la Industria 4.0 y responder adecuadamente a las demandas futuras, es necesario abordar las limitaciones actuales de RCM Tool.

El rediseño de RCM Tool en una versión mejorada (Witness) buscará superar estas limitaciones, integrándose de manera efectiva con la arquitectura del TCMS y asegurando la escalabilidad y el rendimiento necesarios para enfrentar el aumento de la demanda y los requerimientos de seguridad.

4. Propuesta de mejora

4.1 Descripción de la propuesta

Se propone rehacer la plataforma integrada bajo la arquitectura de microservicios adoptada en TCMS, lo que permitiría asegurar que la solución sea escalable en el tiempo ante el aumento de la demanda, nuevos requerimientos y los avances de la industria. Como bien indican Pasquati, Bigheti, Rodrigues de Sá y Godoy [2], la integración entre tecnologías, equipos y sistemas en diferentes niveles de la industria requiere una migración de la arquitectura vertical heredada a una arquitectura plana basada en servicios. La arquitectura orientada a servicios (SOA) y, más recientemente, los microservicios, desempeñan un papel crítico en la Industria 4.0 al proporcionar un marco para integrar sistemas complejos y cumplir con esos requisitos .

4.2 Enfoque

4.2.1 Adopción de Microservicios:

- Redefinir RCM Tool utilizando microservicios, permitiendo que cada componente funcione de manera autónoma y se comunique mediante API. Esto permitirá una mayor flexibilidad, facilidad de mantenimiento y la capacidad de escalar componentes específicos sin afectar a todo el sistema.

4.2.2 Uso de Tecnologías Avanzadas:

- Utilizar lenguajes de programación y frameworks modernos como Python y Flutter para desarrollar microservicios eficientes y de alto rendimiento. Estas tecnologías permitirán superar las limitaciones actuales de Power Apps.

4.2.3 Enfoque en seguridad:

- Implementar diferentes medidas de seguridad, como la encriptación de datos y el uso de roles para manejar permisos dentro de la aplicación, proteger la información sensible y garantizar la integridad de los datos.

4.2.4 Diseño de una Interfaz Intuitiva:

- Rediseñar la interfaz de usuario para mejorar la experiencia del usuario, haciendo que la aplicación sea más intuitiva y fácil de usar. Esto facilitará el registro de condiciones, la generación de avisos y la creación de reportes.

4.3 Beneficios esperados

- **Escalabilidad**: La arquitectura de microservicios permitirá escalar la solución de manera eficiente para satisfacer el crecimiento de la demanda.
- **Flexibilidad y Adaptabilidad**: La plataforma podrá adaptarse rápidamente a los cambios tecnológicos y a las nuevas necesidades de la industria.
- **Eficiencia Operativa Mejorada**: La centralización de la información y el análisis de big data permitirán optimizar los procesos de mantenimiento y reducir los tiempos de inactividad.
- **Seguridad Robusta**: Las medidas de seguridad avanzadas garantizarán la protección de los datos y la confianza de los usuarios.
- **Mejoras en Experiencia de Uso**: Una interfaz mejorada y más intuitiva aumentará la satisfacción del usuario y facilitará la adopción de la herramienta.

En resumen, esta propuesta de mejora para RCM Tool no solo aborda las limitaciones actuales, sino que también prepara la plataforma para enfrentar los desafíos futuros de la Industria 4.0. Con una arquitectura de microservicios y tecnologías avanzadas, la nueva versión estará equipada para integrarse completamente con el ecosistema TCMS, proporcionando una solución robusta, segura y escalable que abordará de mejor manera las necesidades del mantenimiento de maquinaria industrial.

4.3 Objetivos

4.3.1 Objetivo general

Desarrollar una plataforma de reportabilidad innovadora y escalable para inspecciones técnicas de mantenimiento, diseñada para centralizar y optimizar la gestión de la información sobre la condición de los equipos. Esta plataforma permitirá a inspectores y analistas registrar, generar, almacenar y consultar reportes y avisos de manera eficiente. Con una arquitectura moderna basada en microservicios, la solución no solo integrará datos en tiempo real, sino que también facilitará la toma de decisiones informadas, mejorando la operatividad y el mantenimiento predictivo en el entorno industrial.

4.3.2 Objetivos específicos

Análisis y Homologación de la Solución Actual:

Identificar y describir los elementos de la solución actual que serán homologados, modificados o agregados, asegurando que cada componente cumpla con los requisitos de la nueva plataforma.

Diseño e Implementación de una Base de Datos Robusta:

Crear una base de datos eficiente y escalable que soporte todas las funcionalidades de la plataforma, garantizando un almacenamiento seguro y un acceso rápido a la información sobre la condición de los equipos.

Desarrollo de un Sistema de Gestión de Reportes:

Diseñar e implementar un sistema integral para la generación, almacenamiento y consulta de reportes y avisos, que permita a los inspectores y analistas manejar la información de manera eficaz.

Integración con el Ecosistema TCMS:

Asegurar la integración fluida de la nueva plataforma con el resto de las aplicaciones de la suite TCMS, facilitando el intercambio de datos y la colaboración entre sistemas.

Diseño e Implementación de una Interfaz de Usuario Intuitiva:

Crear una interfaz de usuario amigable y eficiente que soporte todas las funcionalidades de la plataforma, mejorando la experiencia del usuario y facilitando la adopción de la herramienta.

Verificación del Funcionamiento de la Solución:

Realizar pruebas exhaustivas para verificar el correcto funcionamiento de la plataforma, asegurando que todos los componentes operen sin problemas y cumplan con los requisitos especificados.

Validación de la Solución con Enfoque en Mejoras Futuras:

Validar la solución desarrollada mediante pruebas con usuarios reales, recopilando feedback valioso que permita identificar oportunidades de mejora y definir posibles áreas de trabajo futuro.

Documentación Completa de la Plataforma:

Elaborar una documentación detallada de la plataforma desarrollada, incluyendo manuales de usuario, guías de implementación y mantenimiento, y especificaciones técnicas, asegurando que cualquier usuario o desarrollador pueda comprender y utilizar la herramienta eficazmente.

5. Metodología

El desarrollo de este proyecto se llevó a cabo bajo la metodología ágil SCRUM, organizada en 8 sprints de tres semanas cada uno. Alejandro Sánchez asumió el rol de Scrum Master, y Rodrigo Vergara el de Product Owner, ambos colaboradores de la empresa SIMPRO.

El trabajo se distribuyó en tres etapas clave para estructurar el proceso de desarrollo de manera eficiente y mantener el enfoque en los objetivos de cada fase.

5.1 Etapa 1: Análisis y Planificación

La primera fase se centró en el análisis de la plataforma existente, RCM Tool, para identificar qué elementos debían ser homologados y cuáles requerían redefinición. Se realizó un estudio exhaustivo de la plataforma, que permitió:

- Definir los requerimientos funcionales y no funcionales del nuevo sistema.
- Redactar las historias de usuario necesarias para guiar el desarrollo.
- Planificar y priorizar los elementos en un backlog de producto que facilitara la organización de los sprints.

En esta etapa, se establecieron los objetivos de cada sprint y se definió la estrategia de desarrollo, teniendo en cuenta el impacto de cada funcionalidad en el objetivo final de la plataforma.

5.2 Etapa 2: Desarrollo e implementación

La segunda fase abarcó el desarrollo e implementación del sistema. Aquí se trabajó en la construcción de los distintos componentes de la plataforma:

- Backend: Se desarrolló la API y se diseñó la base de datos, asegurando su integridad y escalabilidad para soportar las funciones requeridas.
- Frontend: Se construyó la aplicación web, diseñando una interfaz intuitiva y eficiente para el usuario.
- Integración: Se realizó la integración de la API con el frontend, testeando y ajustando las conexiones para optimizar el rendimiento y la experiencia de usuario.

Cada sprint culminó con una revisión y retrospectiva para validar avances y ajustar el enfoque según los desafíos encontrados, permitiendo mantener un flujo de trabajo alineado con los objetivos.

5.3 Etapa 3: Evaluación y documentación

Al finalizar la implementación, se llevó a cabo la evaluación y documentación del proyecto. Durante esta etapa:

- El equipo de SIMPRO realizó una evaluación del funcionamiento y usabilidad de la plataforma, verificando que se cumplieran los requerimientos establecidos al inicio del proyecto.
- Se generó la documentación del proceso, que incluye:
    - Descripción de las funcionalidades desarrolladas.
    - Registro de decisiones clave y cambios realizados durante los sprints.
    - Manuales de usuario y guías técnicas para facilitar el mantenimiento y futuras mejoras.

6. Diseño

6.1 Análisis de requerimientos

Para desarrollar una plataforma de reportabilidad para inspecciones técnicas de mantenimiento, es crucial definir claramente los requerimientos funcionales y no funcionales. Este análisis asegurará que la solución final sea robusta, escalable y alineada con las necesidades de los usuarios y las mejores prácticas de la industria.

6.1.1 Requerimientos Funcionales

Registro de Condiciones de Equipos: La plataforma debe permitir a los inspectores registrar las condiciones de los equipos de manera detallada.

Generación de Avisos: La plataforma debe permitir a los inspectores generar avisos basados en las condiciones registradas.
Gestión de Reportes: Los usuarios deben poder generar, almacenar y consultar reportes detallados sobre las inspecciones y el estado de los equipos.
Interfaz de Usuario Intuitiva: La plataforma debe contar con una interfaz de usuario amigable que facilite la navegación y el uso de las funcionalidades.
Integración con TCMS: La plataforma debe integrarse perfectamente con las otras aplicaciones del ecosistema TCMS.
Historial y Trazabilidad: La plataforma debe mantener un historial completo y trazabilidad de todas las inspecciones y reportes.

6.1.2 Requerimientos No Funcionales

Escalabilidad: La plataforma debe poder manejar un creciente número de usuarios y datos sin afectar su rendimiento.
Seguridad: Los datos almacenados y procesados deben estar protegidos contra accesos no autorizados y vulnerabilidades.
Rendimiento: La plataforma debe responder rápidamente a las acciones de los usuarios y procesar grandes volúmenes de datos de manera eficiente.
Disponibilidad y Confiabilidad: La plataforma debe estar disponible y operativa la mayor parte del tiempo, minimizando el tiempo de inactividad.
Mantenibilidad: El código y la arquitectura de la plataforma deben ser fáciles de mantener y actualizar.
Usabilidad: La plataforma debe ser fácil de usar y aprender para todos los usuarios, independientemente de su nivel técnico.

6.2 Usuarios

Los usuarios de la plataforma corresponden principalmente a trabajadores de la empresa y están divididos en los siguientes roles:

- **Inspector (INS):** Corresponde a inspectores de mantenimiento los cuales se encargan de revisar los equipos en terreno, realizar mediciones y registrar la condición de estos. Además, pueden generar nuevos avisos y reportes a partir de la condición de un equipo.
- **Líder (LDR):** Corresponde a los líderes del equipo en turno, quienes se encargan de la revisión y toma de decisiones sobre acciones realizadas por los inspectores, aprobando o rechazando avisos, cerrando avisos cuando estos se han cumplido, aprobando o rechazando reportes y despachándolos a sus respectivas listas de difusión.

- **Administrador (ADM):** Este usuario tiene permiso para modificar otros usuarios roles, agregar y eliminar usuarios y modificar diferentes aspectos de la plataforma, como puede ser opciones de modos de falla, condición o unidades de medida.

6.3 Historias de Usuario

Como inspector quiero:

1. Registrar una nueva condición asociada a un equipo para guardar la información relacionada a la inspección.
2. Editar una condición existente para modificar la información registrada.
3. Listar las condiciones registradas de un equipo para poder seleccionar una condición.
4. Visualizar el detalle de una condición para poder ver la información registrada.
5. Crear un nuevo aviso a partir de la condición de un equipo registrar una orden de trabajo asociada a una condición.
6. Editar un aviso existente para modificar la información registrada.
7. Solicitar el cierre de un aviso aprobado para indicar que la orden de trabajo ya se ha cumplido y que el aviso sea cerrado.
8. Listar avisos creados para poder seleccionar un aviso.
9. Visualizar el detalle de un aviso para poder ver la información registrada.
10. Filtrar avisos de acuerdo a su estado para poder ver solo los avisos que tengan un estado en particular (ej: ver solo avisos pendientes de cierre).
11. Crear un nuevo reporte a partir de la condición de un equipo para presentar la condición de un equipo y avisos asociados en un formato en específico.
12. Editar un reporte existente para modificar la información registrada.
13. Listar reportes existentes para poder seleccionar uno.
14. Visualizar el detalle de un reporte para poder ver la información registrada.
15. Obtener el pdf de un reporte para poder visualizarlo fuera de la aplicación.
16. Listar las listas de difusión existentes para seleccionar una lista.
17. Visualizar el detalle de una lista de difusión para poder ver los correos pertenecientes a la lista.
18. Listar los equipos existentes en la planta para poder seleccionar un equipo.
19. Seleccionar un equipo dentro de la planta para poder visualizar sus condiciones asociadas.
20. Tener una vista general de los datos en un dashboard para poder monitorear de manera fácil y rápida la condición de la planta.

Como Lider quiero:

1. Eliminar una condición existente para que esta no esté disponible.
2. Aprobar un aviso creado para confirmar que la información registrada es correcta.
3. Rechazar un aviso creado para confirmar que la información registrada es incorrecta.
4. Cerrar un aviso pendiente de cierre para confirmar que la solicitud de cierre y el motivo registrado son correctos.
5. Eliminar un aviso cerrado para que este no esté disponible.
6. Eliminar un reporte existente para que este no esté disponible.
7. Despachar un reporte existente a una lista de difusión para enviarlo a una lista de correos electrónicos.
8. Crear una nueva lista de difusión para guardar correos que puedan ser elegibles para el despacho de un reporte.
9. Editar una lista de difusión existente para modificar los correos pertenecientes a esta lista.
10. Eliminar una lista de difusión existente para que esta no esté disponible.

6.4 Arquitectura de software

Para este proyecto, se ha optado por una arquitectura moderna y escalable que integra diversas tecnologías avanzadas para garantizar un rendimiento óptimo y una fácil mantenibilidad. A continuación, se presenta la arquitectura del software en diferentes niveles del modelo c4.

6.4.1 Diagrama de contexto

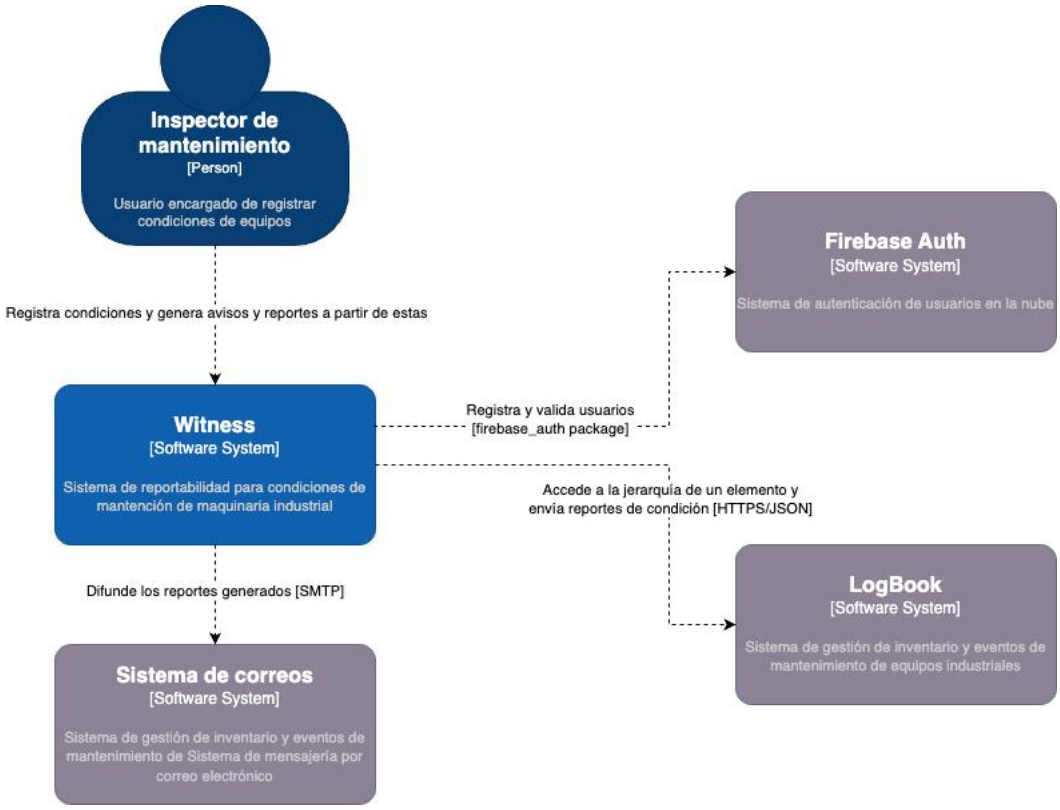


Figura3. Diagrama de contexto del modelo c4.

El diagrama de contexto proporciona una visión general del sistema y sus interacciones con usuarios y otros sistemas externos. En este nivel, se puede observar cómo Witness, el sistema que estamos desarrollando, se integra con otros servicios existentes como Firebase Auth, LogBook y el sistema de correos. El usuario principal es el inspector de mantenimiento, quien se encarga de registrar condiciones de los equipos y generar reportes a partir de estas.

6.4.2 Diagrama de contenedores

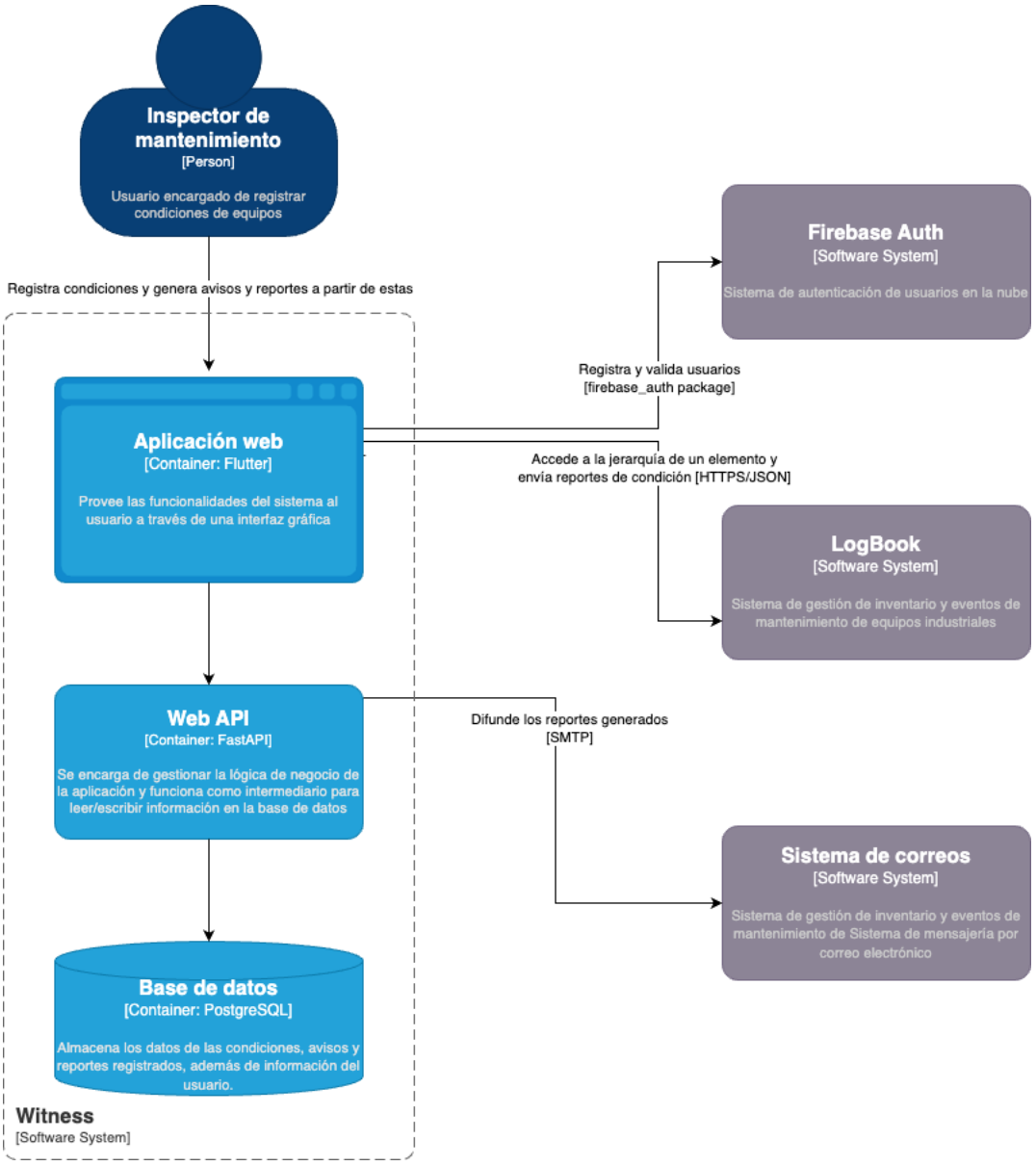


Figura 4. Diagrama de contenedores del modelo c4.

El diagrama de contenedores detalla cómo está estructurado Witness en términos de sus principales módulos o contenedores, y cómo cada uno de ellos interactúa tanto entre sí como con los sistemas externos.

6.4.3 Diagrama de componentes

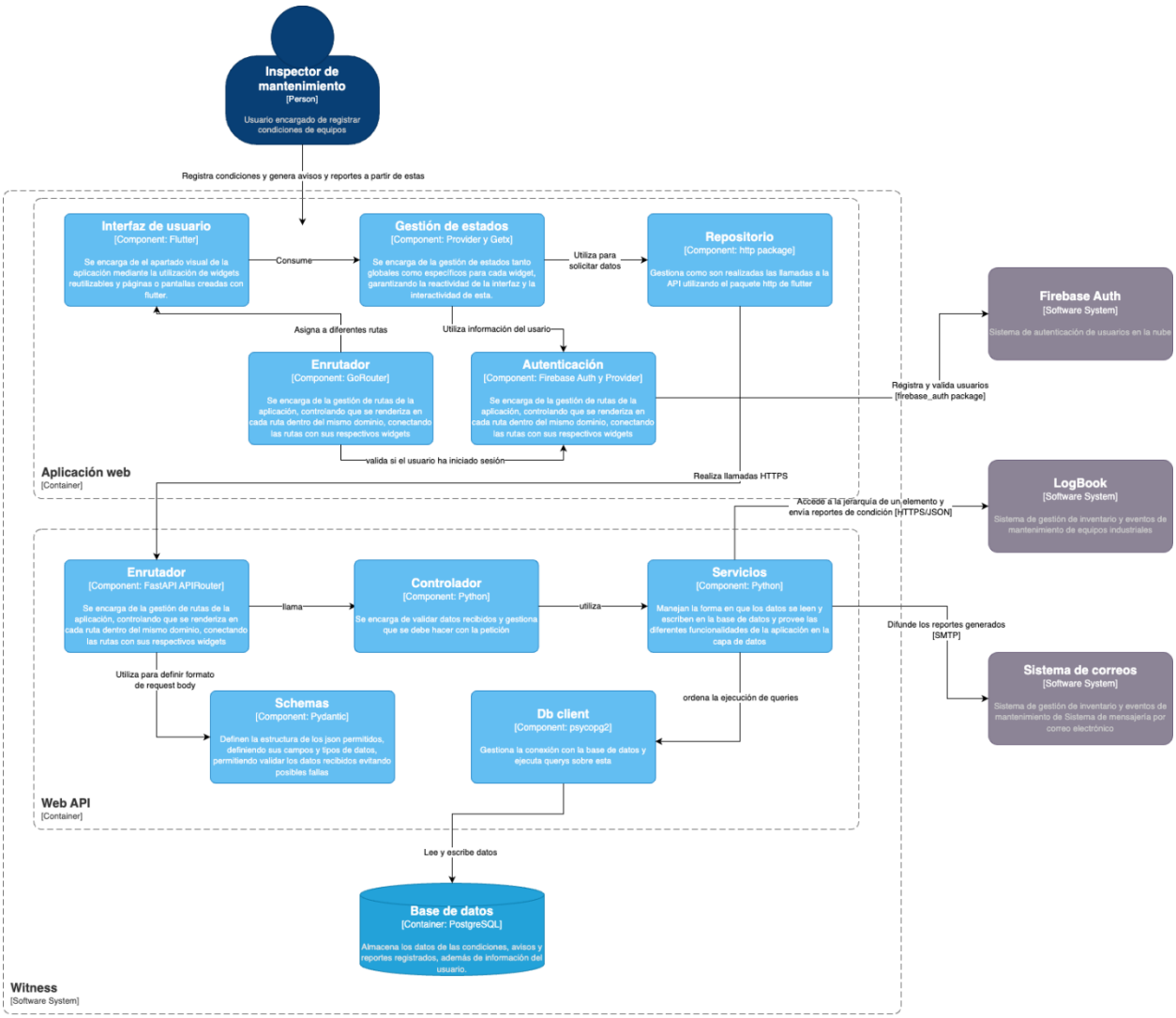


Figura 5. Diagrama de componentes del modelo c4.

El diagrama de componentes desciende un nivel más en el detalle, mostrando cómo están estructurados internamente los contenedores que conforman el sistema Witness. Este diagrama proporciona una vista detallada de los componentes individuales dentro de cada contenedor, y cómo estos interactúan entre sí para cumplir con las funcionalidades del sistema.

7. Implementación

7.1 Implementación del Backend

El desarrollo del backend es una de las fases más cruciales del proyecto, ya que se encarga de gestionar la lógica de negocio, la base de datos y la comunicación entre el frontend y los datos. A continuación, se detallan las etapas clave del desarrollo del backend:

7.1.1 Base de Datos

Tecnología: PostgreSQL

Definición del Esquema:

- Se tomo en cuenta la naturaleza y estructura de los datos de RCM Tool para diseñsar un esquema de base de datos que incluye tablas para condiciones, reportes, avisos, usuarios, listas de difusión, imágenes, correos, solicitudes de cierre de aviso e historial de actualizaciones para avisos y reportes.
- Se definieron las relaciones entre las tablas para asegurar la integridad referencial y se crearon índices para optimizar el rendimiento de las consultas.

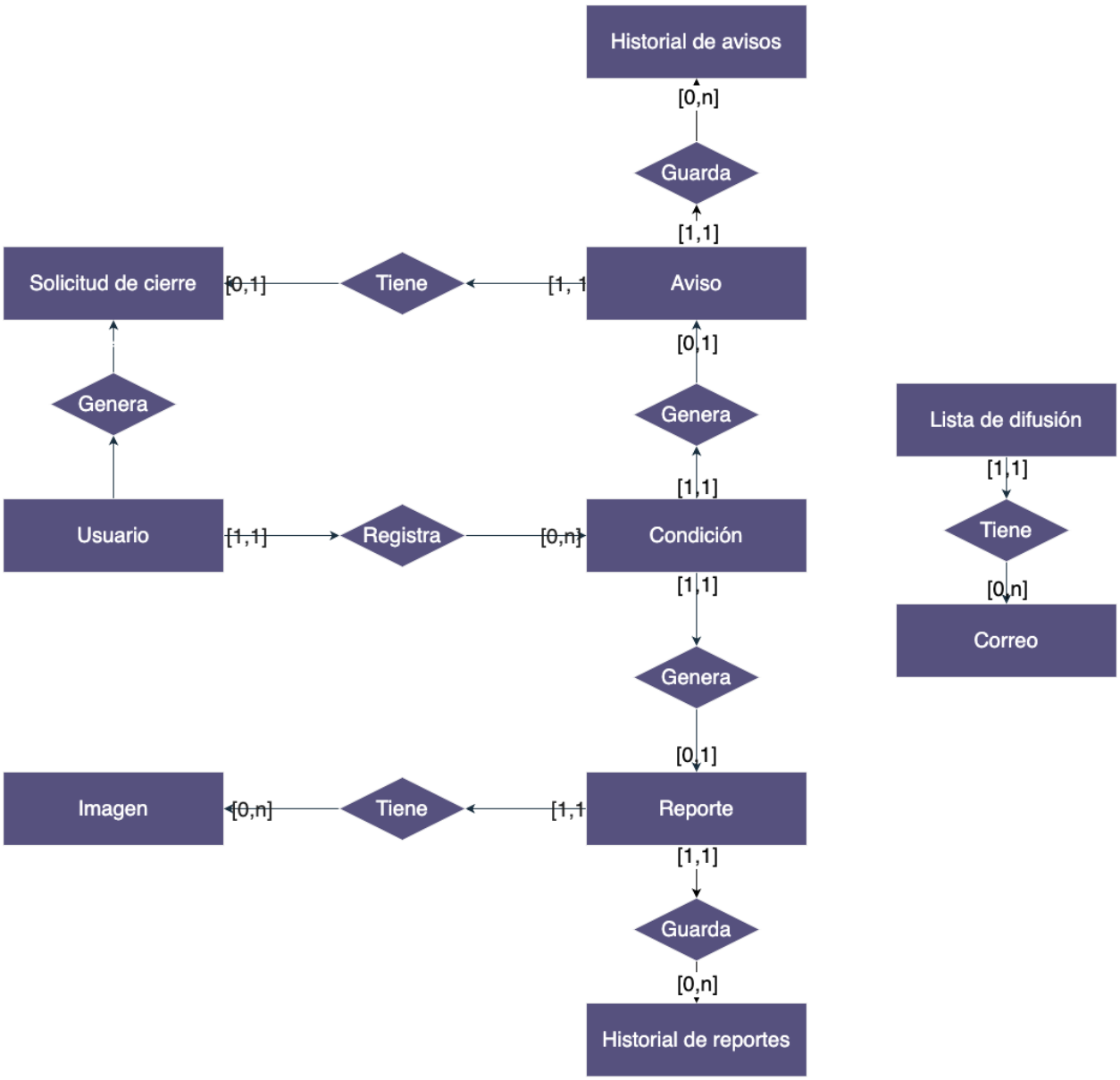


Figura 6. Modelo entidad-relación de la base de datos de Witness

Creación de Tablas:

Se implementaron las tablas necesarias según el esquema diseñado.
A continuación, se detallan las tablas creadas junto a sus atributos:

- **Tabla de usuarios:**

  A pesar de que el flujo de autenticación de la aplicación se realiza a través de firebase se ha creado esta tabla para almacenar información importante de los usuarios como puede ser su rol, el cual otorga o restringe permisos para el uso de ciertas funcionalidades.

Atributo	Descripción
PK: u_id	Identificador autogenerado por postgresql
name	Nombre del usuario
email	Correo electrónico el usuario
role	Rol del usuario (‘INS’, ‘LDR’, ‘ADM’)
created	Timestamp de la creación del usuario en la base de datos

Tabla 1. Descripción tabla sql de usuarios

- **Tabla de avisos:**

Atributo	Descripción
PK: a_id	Identificador autogenerado por postgresql
FK: solicitor_id	Referencia al usuario que generó el aviso
FK: responsible_id	Referencia al usuario que aprobó/rechazó el aviso
intervention_date	Fecha de intervención solicitada en la creación del aviso
recommendation	Recomendaciones registradas por el usuario al momento de la creación del aviso
priority	Prioridad del aviso. (‘Emergencia’, ‘Correctivo’, ‘Planificado’, ‘Parada de planta’)
status	Estado del aviso. (‘Nuevo’ ‘Aprobado’, ‘Rechazado’, ‘Pendiente de cierre’, ‘Cerrado’)
created_at	Timestamp de la creación del aviso
closed_at	Timestamp del cierre del aviso
a_number	Número del aviso registrado al aprobar el aviso
lub_ammount	Cantidad en gramos de lubricante a aplicar al equipo al cual el aviso está relacionado

Tabla 2. Descripción tabla sql de avisos

- **Tabla de condiciones:**

Atributo	Descripción
PK: c_id	Identificador autogenerado por postgresql
FK: inspector_id	Referencia al usuario que registró la condición
equipment_id	Id del elemento de la jerarquía sobre el cual se registró la condición
visibility	Valor booleano que indica si la condición debe ser mostrada o no (true = visible, 0=oculta)
observation	Observaciones registradas por el usuario al momento de la inspección
recommendation	Recomendaciones registradas por el usuario al momento de la inspección
failure_mode	Modo de falla registrado por el usuario al momento de la inspección
magnitude	Magnitud de la medición registrada por el inspector. (“[valor] [unidad de medida]”)
Created_at	Timestamp creado al momento de inserción en la base de datos
status	Estado de salud del equipo registrado en la condición (‘Normal’, ‘Observación’, ‘Alerta’, ‘Alarma’, ‘Emergencia’)
a_id	Referencia al aviso generado a partir de la condición registrada

Tabla 3. Descripción tabla sql de condiciones

- **Tabla de solicitud de cierre de avisos:**

Atributo	Descripción
PK: cp_id	Identificador autogenerado por postgresql
FK: a_id	Referencia al aviso sobre el que se genera la solicitud de cierre
FK: u_id	Referencia al usuario que solicita el cierre del aviso
reason	Motivo de cierre de la solicitud de cierre
status	Estado de la solicitud (‘Pendiente’, ‘Resuelta’)
created_at	Timestamp de la creación de la solicitud

Tabla 4. Descripción tabla sql de cierre de avisos

- **Tabla de historial de avisos:**

Atributo	Descripción
FK: a_id	Referencia al aviso sobre el cual se registra el historial
short_text	Texto breve del aviso antes de la actualización
recommendation	Recomendaciones del aviso antes de la actualización
a_number	Número del aviso antes de la actualización
priority	Prioridad del aviso antes de la actualización
Intervention_date	Fecha de intervención solicitada del aviso antes de la actualización
updated_at	Timestamp de la actualización del aviso

Tabla 5. Descripción tabla sql de historial de avisos

- **Tabla de listas de difusión:**

Atributo	Descripción
PK: dl_id	Identificador autogenerado por postgresql
name	Nombre de la lista de difusión creada por el usuario
area_id	Referencia al elemento de la jerarquía de LogBook de nivel 5 a la cual pertenece la lista de difusión
created_at	Timestamp de la creación de la lista de difusión

Tabla 6. Descripción tabla sql de listas de difusión

- **Tabla de correos:**

Atributo	Descripción
mail	Dirección de correo electrónico

FK: dl_id	Referencia a la lista de difusión a la cual pertenece el correo

Tabla 7. Descripción tabla sql de correos

- **Tabla de reportes:**

Atributo	Descripción
PK: r_id	Identificador autogenerado por postgresql
FK: c_id	Referencia a la condición sobre la que se genera el reporte
observation	Observaciones registradas por el usuario
recommendation	Recomendaciones registradas por el usuario
code	Código único generado por el sistema al crear el reporte
format	Formato del reporte que representa el estilo de visualización en el pdf.(‘Vibraciones 3’, ‘Vibraciones 4’, ‘Galería’, ‘Ejecutivo’)
status	Estado del reporte. (‘Nuevo’, ‘Despachado’)
ot	Número de orden de trabajo registrado por el usuario en la creación del reporte
FK: responsible_id	Referencia al usuario que realiza el despacho del reporte a una lista de difusión
FK: inspector_id	Referencia al usuario que crea el reporte
created_at	Timestamp de la creación del reporte
dispatched_at	Timestamp del despacho del reporte

Tabla 8. Descripción tabla sql de reportes

- **Tabla de imágenes:**

Atributo	Descripción
PK: rimg_id	Identificador autogenerado por postgresql
FK: r_id	Referencia al reporte al cual pertenece la imagen
img	Imagen en base64
description	Descripción de la imagen registrada

Tabla 9. Descripción tabla sql de imágenes

- **Tabla de historial de reportes:**

Atributo	Descripción

FK: r_id	Referencia al reporte sobre el cual se registra el historial
observation	Observaciones del reporte antes de la actualización
recommendation	Recomendaciones del reporte antes de la actualización
format	Formato del reporte antes de la actualización
updated_at	Timestamp de la actualización del aviso

Tabla 10. Descripción tabla sql de historial de reportes

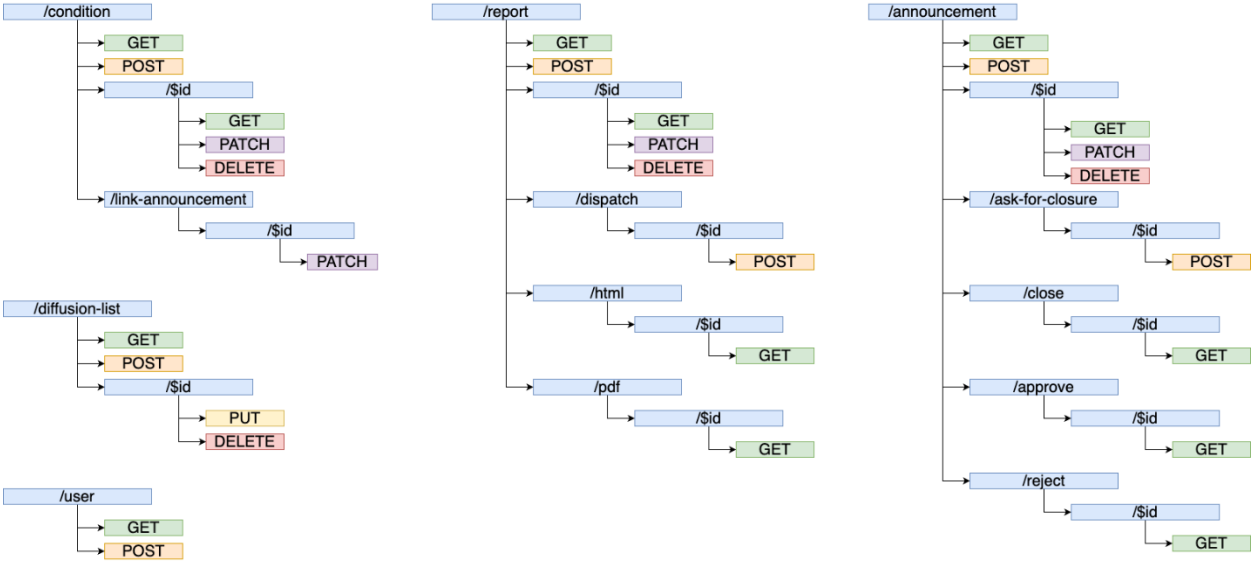
7.1.2 API web

Tecnología: FastAPI

Para la implementación de la APIweb se utilizó una arquitectura de 3 capas (presentación, negocio y datos) permitiendo modularizar el código lo que lo hace más fácil de refactorizar y/o escalar

Creación de Rutas:

Se crearon endpoints RESTful para manejar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos, teniendo un archivo router para cada recurso (ej: condition_router.py) donde se definen los distintos endpoints.



.Figura 7. Diagrama de endpoints de la API web

A continuación, se detallan los endpoints creados:

- **/condition:**

    - **GET**: Listar todas las condiciones que sean visibles. Este endpoint cuenta con paginación y retorna las 20 últimas condiciones correspondientes a la página indicada. Además, se puede filtrar por equipo a través del query parameter 'e_id'.
    - **POST**: Crear una nueva condición. Recibe un JSON como cuerpo de la request, valida los datos, crea la condición y retorna la información de esta.
- **/condition/$id:**
    - **GET:** Retorna la información de la condición con el id indicado
    - **PATCH:** Actualizar una condición, recibe un JSON con los datos a actualizar y retorna la condición actualizada.
    - **DELETE:** Elimina la condición con el id indicado.
- **/condition/link-announcement/$id**
    - **PATCH:** Asocia la condición con el id indicado a un aviso creado.

- **/report:**
    - **GET:** Listar todos los reportes. Este endpoint cuenta con paginación y retorna los 20 últimos reportes correspondientes a la página indicada.
    - **POST:** Crear un nuevo reporte. Recibe un JSON con la información de este y retorna el nuevo reporte.
- **/report/$id:**
    - **GET:** Retorna la información del reporte con el id indicado
    - **PATCH:** Actualizar un reporte, recibe un JSON con los datos a actualizar y retorna el reporte actualizado.
    - **DELETE:** Elimina el reporte con el id indicado.
- **/report/dispatch/$id:**
    - **POST:** Despacha el reporte con el id indicado a una lista de correos electrónicos indicada en el cuerpo de la request
- **/report/html/$id:**
    - **GET:** Retorna una página html para visualizar el reporte con el id indicado
- **/report/pdf/$id:**
    - **GET:** Retorna una página pdf para visualizar el reporte con el id indicado

- **/announcement:**
    - **GET:** Listar todos los avisos. Este endpoint cuenta con paginación y retorna los 20 últimos avisos correspondientes a la página indicada.
    - **POST:** Crear un nuevo aviso. Recibe un JSON con la información de este y retorna el nuevo aviso.

- **/announcement/$id:**
    - **GET:** Retorna la información del aviso con el id indicado
    - **PATCH:** Actualizar un aviso, recibe un JSON con los datos a actualizar y retorna el aviso actualizado.
    - **DELETE:** Elimina el aviso con el id indicado.
- **/annoucement/ask-for-closure/$id:**
    - **POST:** Crea una nueva solicitud de cierre asociada al aviso con el id indicado.
- **/announcement/close/$id:**
    - **GET:** Aprueba el cierre de un aviso, actualizando su estado a 'Closed' y el estado de la solicitud de cierre a 'Solved'.
- **/announcement/approve/$id:**
    - **GET:** Aprueba un aviso con estado 'New' y le asigna un número de aviso.
- **/announcement/reject/$id:**
    - **GET:** Rechaza el aviso con el 'id' indicado, actualizando su estado a 'Rejected'.
- **/diffusion-list:**
    - **GET:** Retorna todas las listas de difusión creadas.
    - **POST:** Crea una nueva lista de difusión.
- **/diffusion-list/$id:**
    - **PUT:** Actualiza la lista de difusión con el id indicado
    - **DELETE:** Elimina la lista de difusión con el id indicado.
- **/user:**
    - **GET:** Retorna una lista con los usuarios registrados. Se puede filtrar por id a través del query parameter 'u_id'.
    - **POST:** Se registra un nuevo usuario en la base de datos.

Validación y Seguridad:

Para la validación de los datos recibidos a través de las solicitudes, se utilizaron esquemas definidos con Pydantic, lo que permitió asegurar que la información recibida cumpliera con los formatos y tipos esperados. Además de estos esquemas, se implementaron métodos adicionales para validar la creación y actualización de recursos, garantizando que se cumplieran las reglas de negocio establecidas.

Asimismo, se hizo uso de diferentes middlewares para gestionar aspectos críticos como el manejo de CORS (Cross-Origin Resource Sharing), así como la definición de métodos y headers permitidos. Estas configuraciones son esenciales para asegurar el correcto

funcionamiento y uso seguro de la API, protegiendo el sistema de solicitudes no autorizadas y mejorando la interoperabilidad con otros servicios.

Conexión con base de datos:

Para la conexión con la base de datos y la ejecución de queries SQL se implementó una clase DBClient utilizando la librería psycopg2 con un patrón singleton para evitar bloqueos en la base de datos. Además, se utilizó un pool de conexión para asegurar la disponibilidad de esta

Integraciones:

Para la integración con un sistema de correo electrónico se utilizó la librería smtplib, que permitió crear un servidor SMTP (Simple Mail Transfer Protocol) que junto con las credenciales correspondientes fue utilizado para la difusión de reportes.

7.2 Desarrollo del Frontend

7.2.1 Implementación aplicación web

Tecnología: Flutter

- **Interfaz (Pantallas y widgets):**

Se implementaron diferentes pantallas asociadas a las diferentes funcionales definidas para este proyecto.

A continuación, se detallan las diferentes pantallas de la interfaz:

- **Pantalla de condiciones**

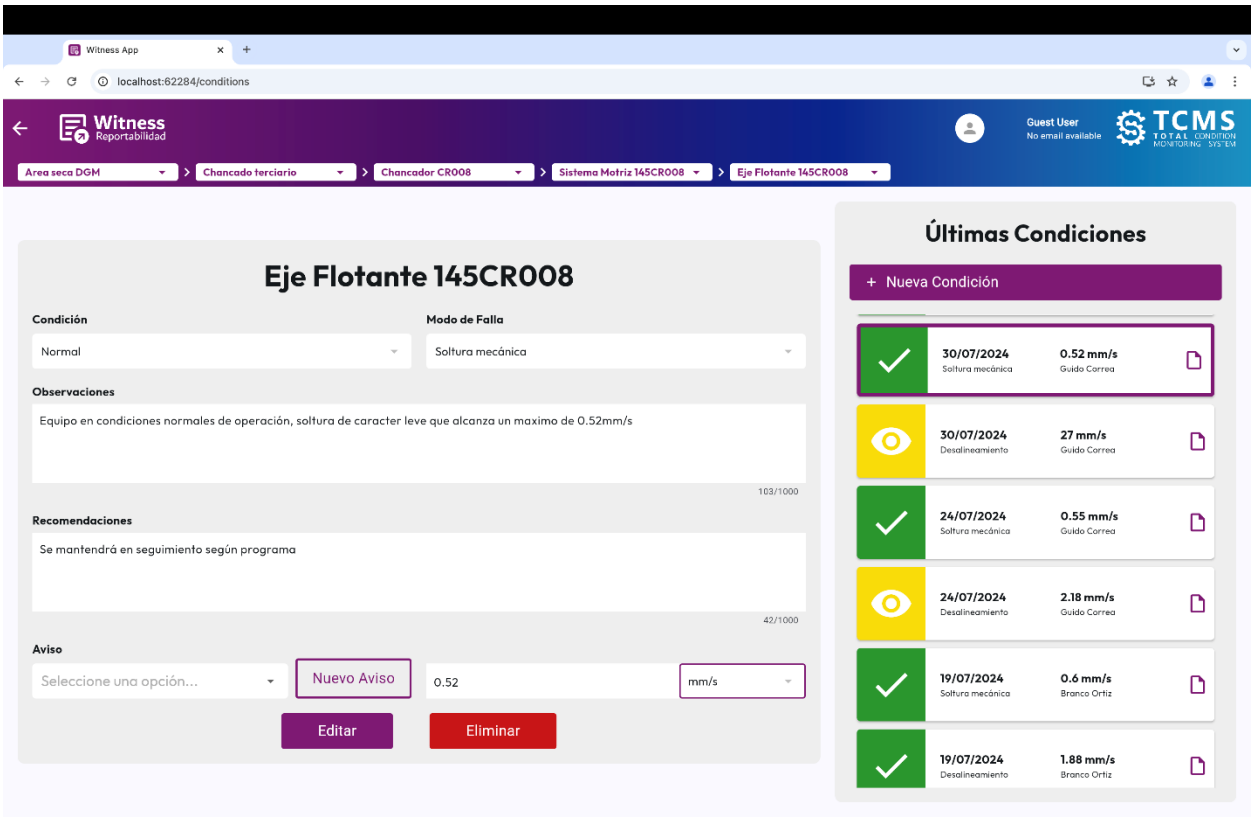


Figura 8. Pantalla de Condiciones de Witness

Esta pantalla fue diseñada e implementada para la visualización y gestión de condiciones, desempeñando un rol crítico en la aplicación, ya que también permite redirigir a la creación de avisos y reportes. Fue implementada como un Stateful Widget y utiliza un controlador separado, en un archivo aparte, para la gestión de estado mediante la librería GetX.
En la parte superior de la pantalla, se dispone una barra de selección jerárquica que permite navegar entre niveles mediante una serie de dropdowns. A la izquierda, se muestra el detalle de la condición seleccionada junto con botones para editar y eliminar. Esta sección también facilita la creación y edición de condiciones cuando se activa el modo de edición, gestionado por una variable observable de tipo booleano en el controlador.

Finalmente, a la derecha se encuentra una lista de las condiciones registradas, ordenadas del registro más reciente al más antiguo, y filtradas según el equipo seleccionado.

- **Pantalla de avisos**

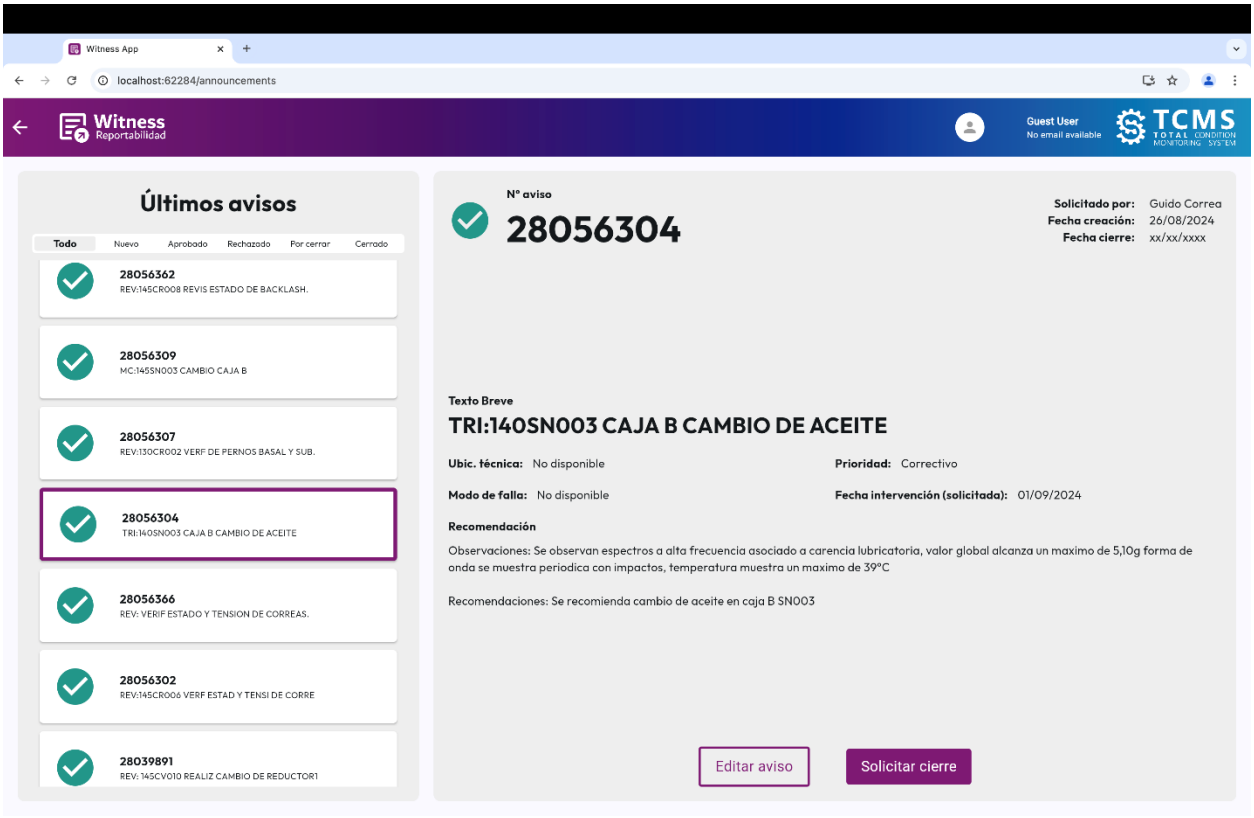


Figura 9. Pantalla de avisos de Witness

Esta pantalla está diseñada para la visualización de avisos existentes, mostrando el estado de cada aviso (nuevo, aprobado, rechazado, pendiente de cierre, cerrado) a través de un sistema de íconos y colores en las tarjetas correspondientes.

Además, ofrece las funcionalidades de aprobar o rechazar un aviso y de solicitar su cierre. La pantalla presenta una lista de avisos ordenados del más reciente al más antiguo, junto con una vista detallada del aviso seleccionado. Asimismo, la lista incluye una barra de filtros que permite filtrar los avisos según su estado.

- **Pantalla de reportes**

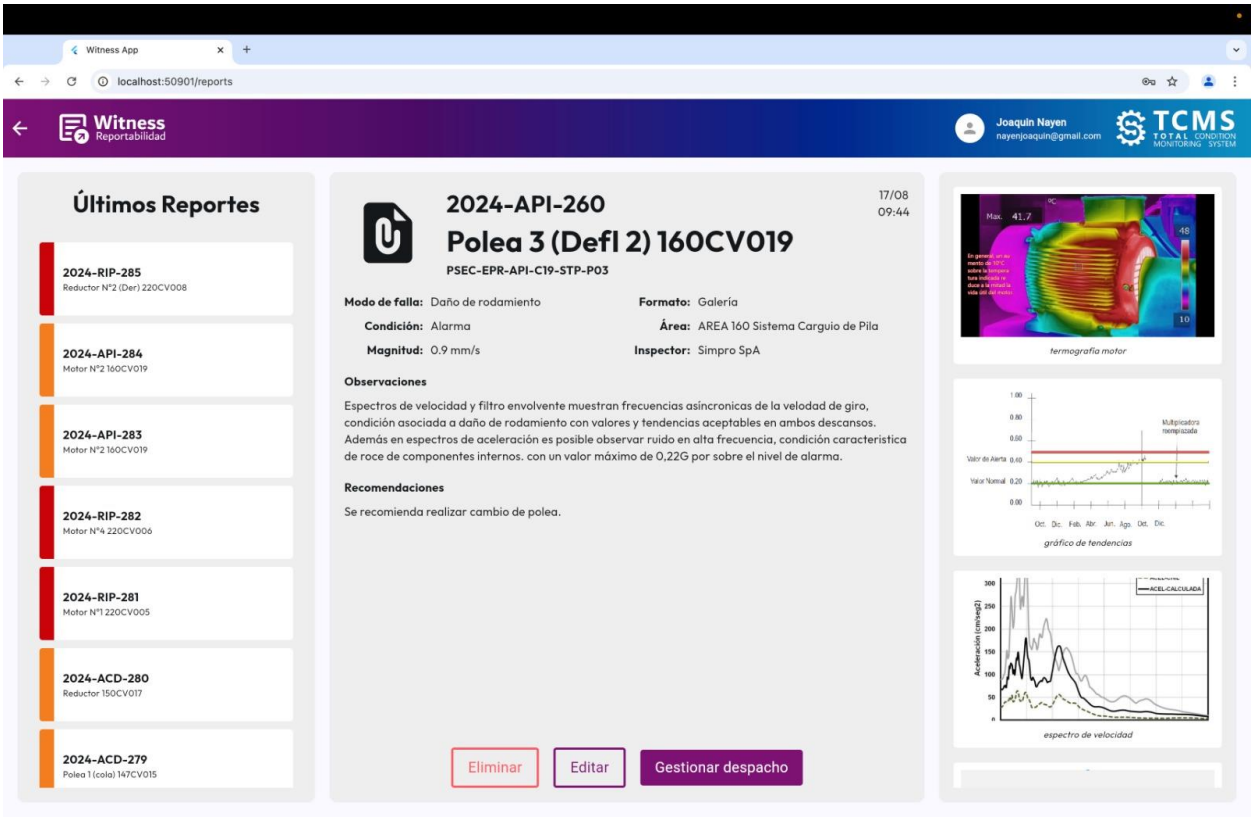


Figura 10. Pantalla de reportes de Witness

La pantalla de reportes fue diseñada e implementada para la visualización y gestión básica de reportes, incluyendo funciones como difundir y eliminar reportes. Las funcionalidades de edición y creación de reportes se encuentran en una pantalla independiente.

Esta pantalla incluye una lista de los reportes existentes, una galería para visualizar las imágenes del reporte seleccionado, y una sección principal que muestra el detalle del reporte.

- **Dashboard**

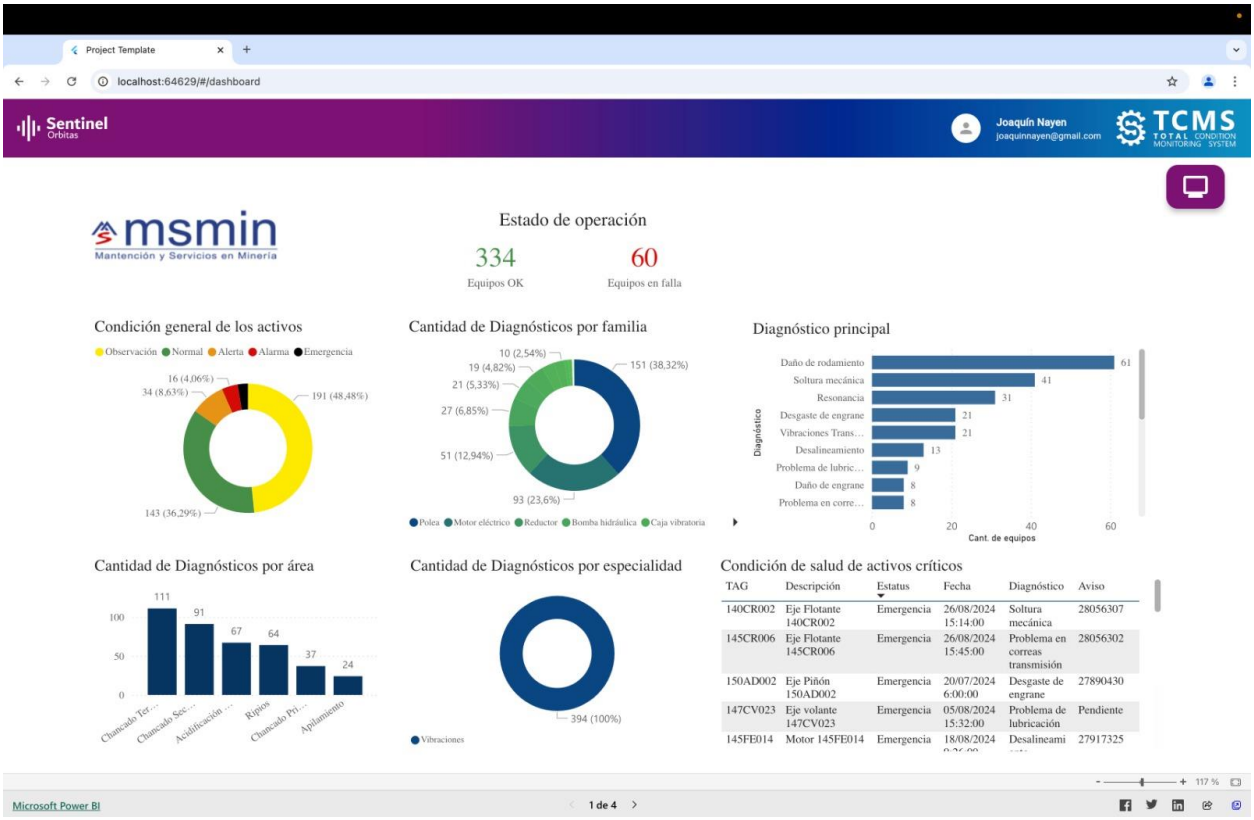


Figura 11. Pantalla de Dashboard de Witness

Esta pantalla se encarga de la visualización de un sitio web que muestra el dashboard del sistema, desarrollado en Power BI.

Además, ofrece la funcionalidad de configurar el dashboard en modo display, que alterna automáticamente entre las diferentes vistas del dashboard de forma periódica.

- **Pantalla de nuevo aviso**

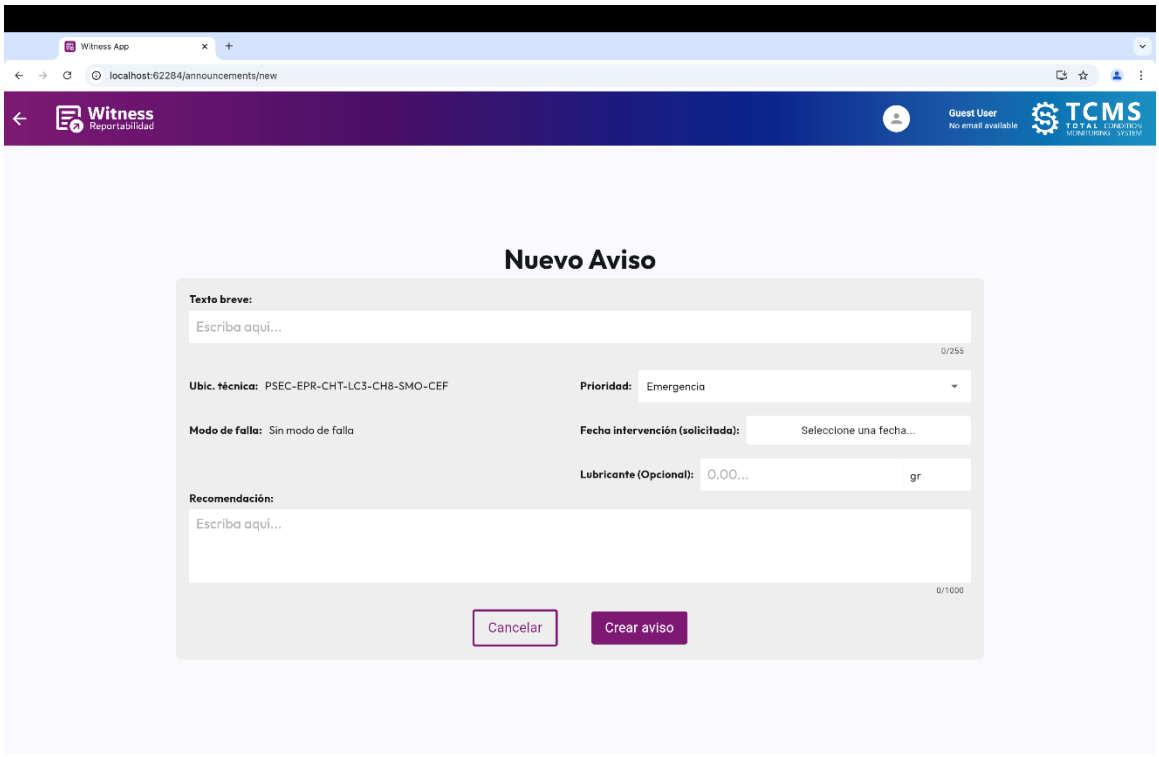


Figura 12. Pantalla de creación de aviso de Witness

Esta pantalla permite la creación de nuevos avisos, recibiendo como parámetro la condición asociada al aviso, además de un callback que se ejecuta al completar la creación.

- **Pantalla de nuevo reporte**

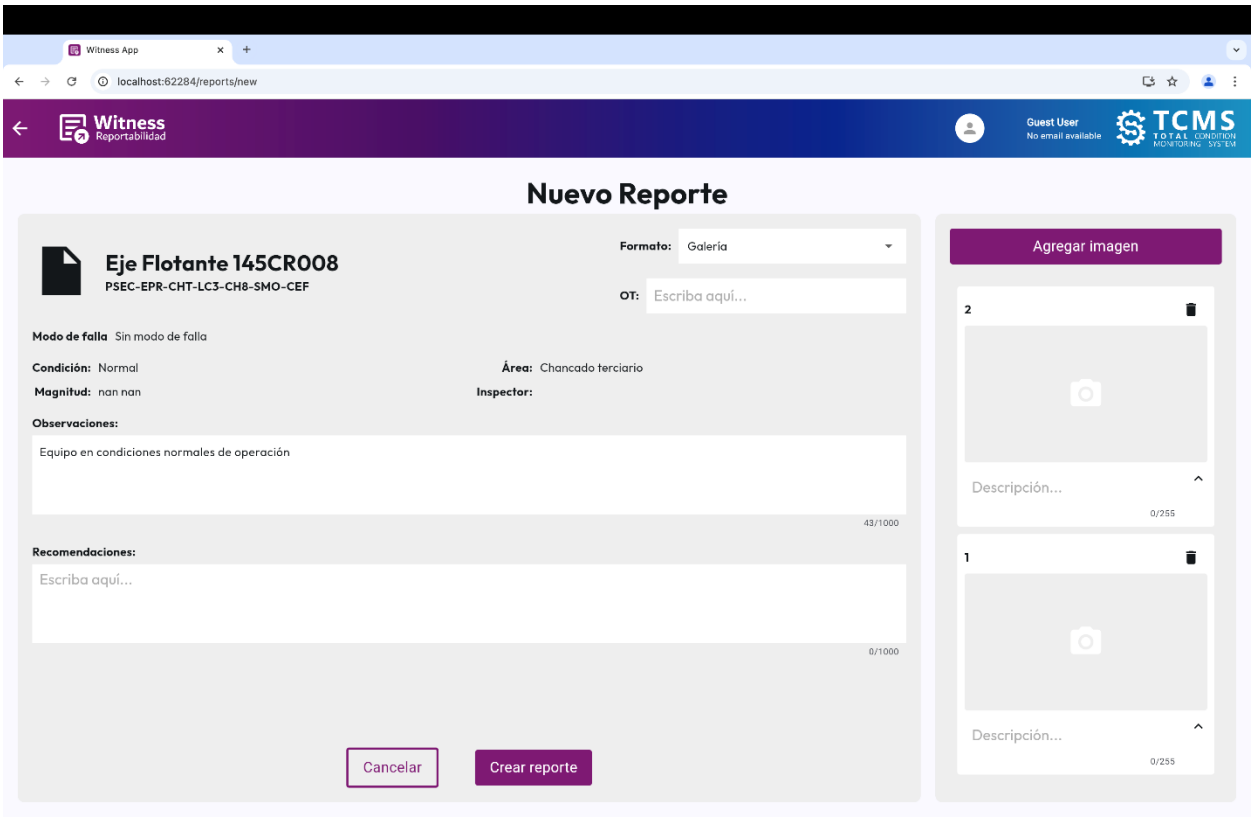


Figura 13. Pantalla de creación de reportes de Witness

Esta pantalla, destinada a la creación de reportes, permite generar nuevos reportes a partir de una condición existente.

- **Pantalla de editar aviso**

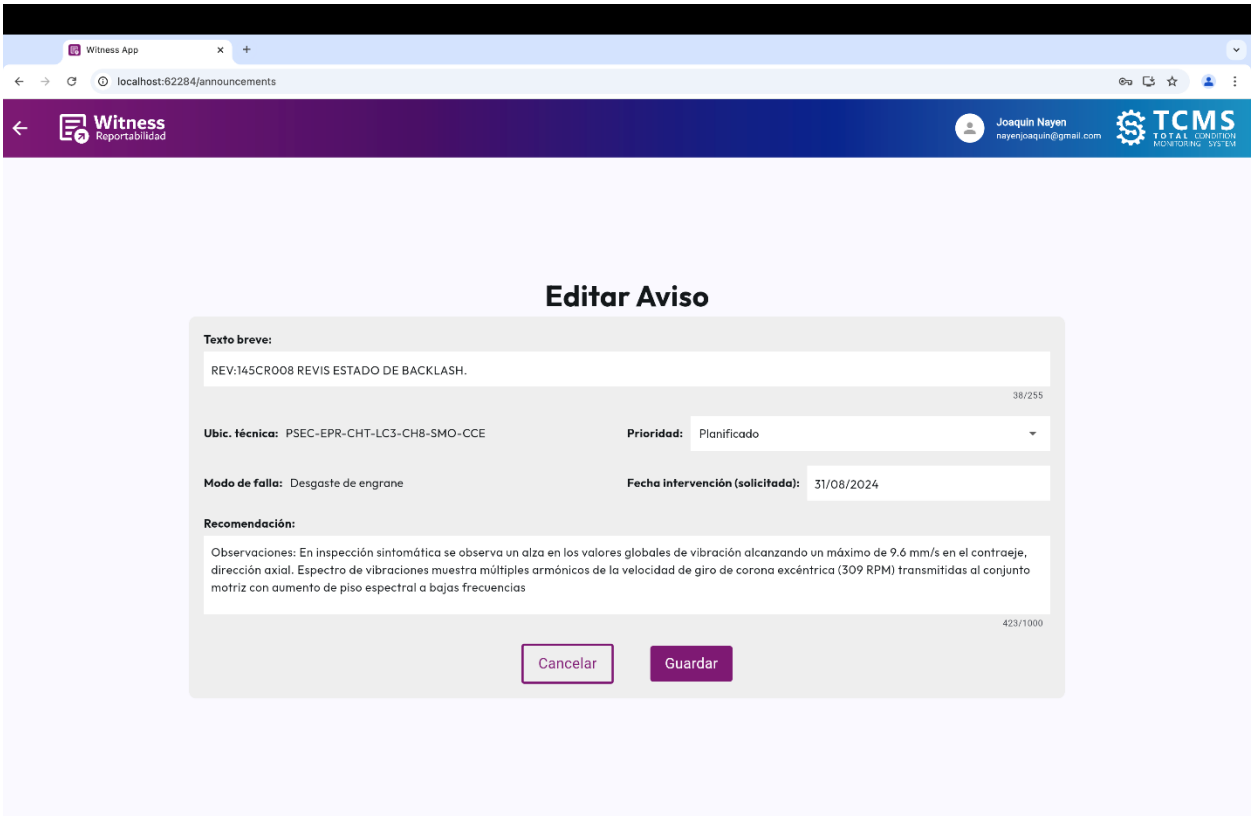


Figura 14. Pantalla de edición de avisos de Witness

Pantalla diseñada para la edición de avisos existentes, con validación de formularios. Recibe como parámetro el aviso a editar.

- **Pantalla de editar reporte**

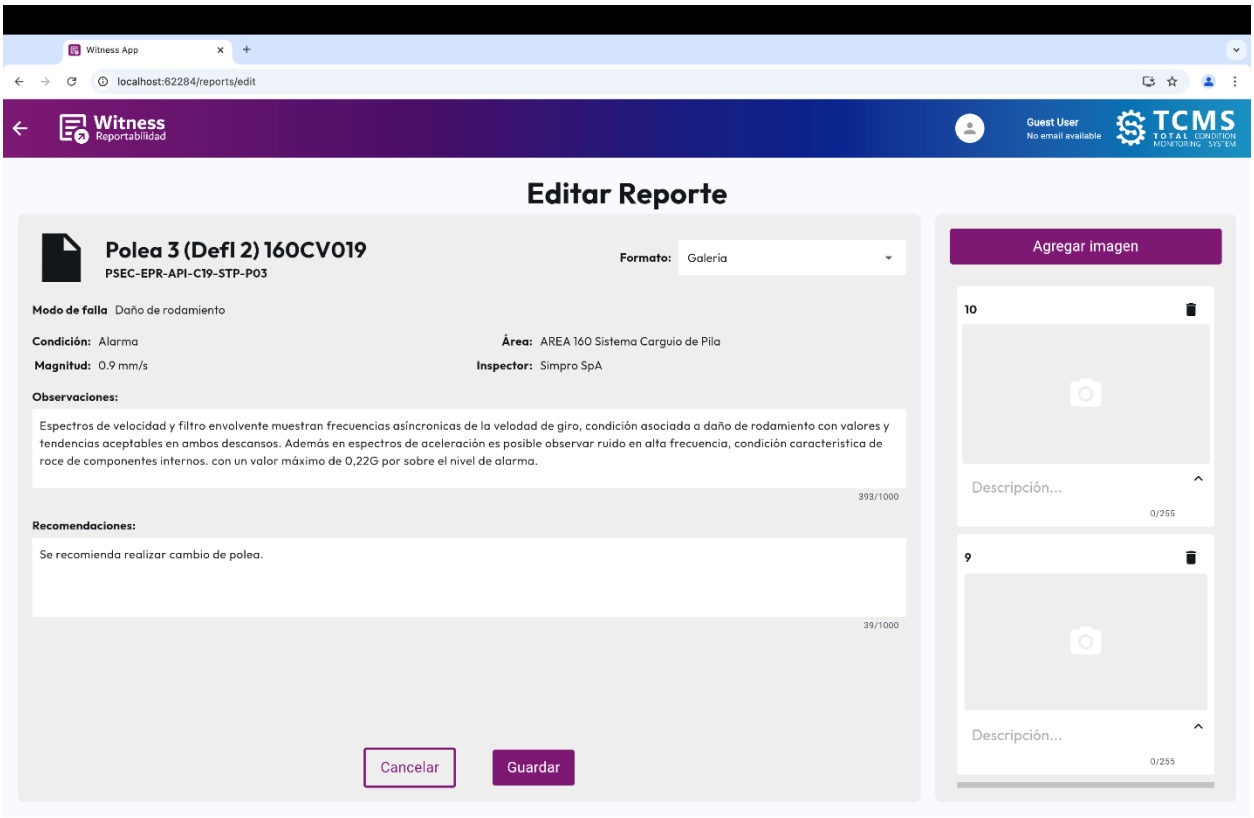


Figura 15. Pantalla de edición de reportes de Witness

Pantalla diseñada para la edición de reportes existentes, con validación de formularios. Recibe como parámetro el reporte a editar. A la derecha, se muestra una lista de imágenes en la que es posible agregar nuevas imágenes, así como editar o eliminar las ya existentes.

- **Pantalla de inicio**

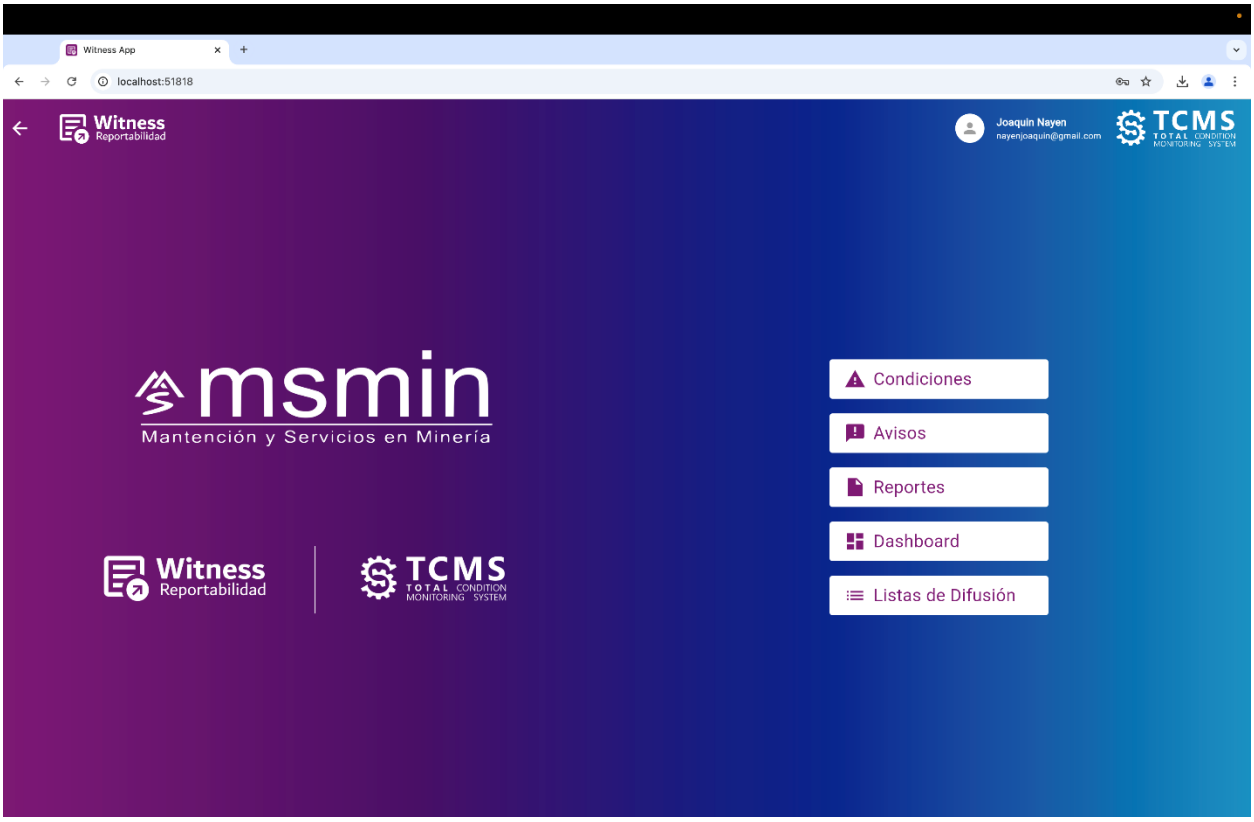


Figura 16. Pantalla de inicio de Witness

Esta pantalla es el punto de entrada de la aplicación y funciona como un menú principal, desde el cual el usuario puede acceder a las vistas principales de la aplicación.

- **Pantalla de inicio de sesión**

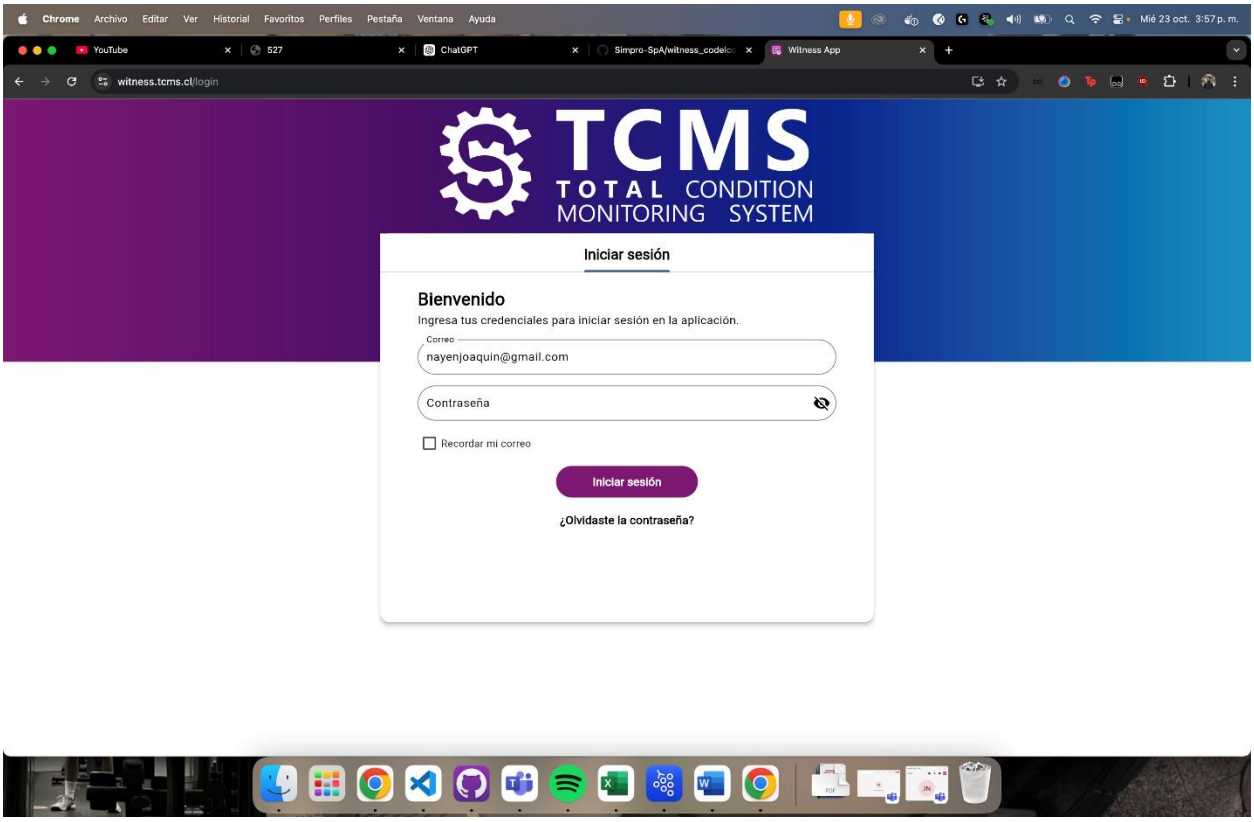


Figura 17. Pantalla de inicio de sesión de Witness

Esta pantalla es responsable de la autenticación de los usuarios y mantiene una estética consistente con todas las aplicaciones de la suite TCMS.

- **Gestión de estados**

  Para la gestión de estados globales, se utilizó la librería Provider, creando clases que extienden de ChangeNotifier. Esto permite que los widgets envueltos en un Consumer de este provider se actualicen de forma reactiva mediante el método notifyListeners desde el provider.

  Por otro lado, cada pantalla gestiona sus estados locales a través de un controlador, una clase que extiende de GetxController de la librería GetX. Esto permite declarar variables observables que reaccionan de manera reactiva al método update() desde el controlador, siempre que el widget en la pantalla esté envuelto en un observador Obx.

- **Rendimiento**

  Con el objetivo de reducir el tiempo de carga y mejorar la experiencia del usuario, se implementó lazy loading en las listas (condiciones, reportes y avisos). Esta técnica permite que los datos se soliciten a la API a medida que son requeridos en la interfaz.

8. Verificación

Para verificar la efectividad de la solución, se llevaron a cabo pruebas unitarias en el backend durante todo el proceso de desarrollo, utilizando pytest como herramienta principal. Estas pruebas permitieron garantizar que cada componente de la lógica de negocio funcionara correctamente. Adicionalmente, se utilizaron Postman para validar la interacción entre la API y la base de datos, asegurando que las solicitudes y respuestas se manejaran de manera adecuada y cumplieran con los requisitos establecidos.

Asimismo, la interfaz de usuario fue sometida a pruebas por otros miembros del equipo de SIMPRO. Este enfoque colaborativo permitió obtener retroalimentación valiosa, identificando áreas de mejora y optimizando la usabilidad del software. El objetivo de estas pruebas era lograr un producto final de la más alta calidad posible, dentro de los límites y objetivos del proyecto.

9. Validación y resultados

Para poder evaluar los resultados obtenidos del desarrollo de este proyecto, se aplicaron dos encuestas para medir la usabilidad y la calidad técnica del software respectivamente.

9.1 Usabilidad

A pesar de que el software no fue utilizado por el usuario final durante el periodo de desarrollo de este proyecto, se llevó a cabo una evaluación de usabilidad utilizando la escala SUS (System Usability Scale)[6], la cual se compone de 10 afirmaciones que se responden con una escala del 1 al 5 donde 1 es “en desacuerdo” y 5 “de acuerdo. Esta evaluación se aplicó al equipo de Desarrollo e innovación de la empresa SIMPRO, obteniendo un total de 4 respuestas.

A continuación, se detallan las preguntas de la encuesta.

1. Creo que me gustaría usar este software con frecuencia.
2. Encontré el software innecesariamente complejo.
3. Considero que el software es fácil de usar.

4. Creo que necesitaría el apoyo de una persona técnica para usar este software.
5. Encontré que las diversas funciones de este software están bien integradas.
6. Creo que hay demasiada inconsistencia en este software.
7. Imagino que la mayoría de las personas aprenderían a usar este software muy rápidamente.
8. Encontré el software muy engorroso de usar.
9. Me sentí muy seguro/a usando el software.
10. Necesité aprender muchas cosas antes de poder comenzar a usar el software.

Pregunta	Respuesta 1	Respuesta 2	Respuesta 3	Respuesta 4
Pregunta 1	5	5	4	5
Pregunta 2	1	2	3	1
Pregunta 3	5	5	5	5
Pregunta 4	3	2	2	3
Pregunta 5	5	5	4	5
Pregunta 6	1	1	2	2
Pregunta 7	5	4	4	4
Pregunta 8	1	2	2	1
Pregunta 9	5	4	4	4
Pregunta 10	2	2	3	2

Tabla 11. Resultados encuesta de usabilidad

En promedio se obtuvo un puntaje de 82.5/100.

A continuación, se detalla la escala:

- **Por debajo de 50**: Se considera pobre, lo que indica problemas importantes de usabilidad.
- **50-68**: Indica una usabilidad promedio o aceptable, pero aún hay áreas para mejorar.
- **68-80**: Un puntaje en este rango se considera bueno, mostrando una experiencia de usuario sólida.

- **80-90**: Muy bueno, lo que refleja una usabilidad alta y usuarios bastante satisfechos.
- **90-100**: Excelente, indicando una experiencia excepcional y muy bien diseñada en términos de usabilidad.

Con base en estos resultados, se puede concluir preliminarmente que la usabilidad del software es, en términos generales, buena. No obstante, al analizar más detenidamente las respuestas, se observa una percepción de alta complejidad en su uso. Esto se refleja en las respuestas a la pregunta 5 ("Creo que necesitaría el apoyo de una persona técnica para usar este software") y la pregunta 3 ("Encontré el software innecesariamente complejo"), lo cual señala un posible punto de mejora en la usabilidad del sistema.

9.2 Calidad Técnica

Se realizó una encuesta de calidad técnica del software dirigida a Alejandro Sánchez, ingeniero civil informático y líder técnico de la unidad informática, y a Matías Cáceres, ingeniero civil informático, ambos colaboradores de la empresa SIMPRO. El propósito de esta encuesta fue medir y evaluar la calidad técnica del software.

La encuesta se basó en los principales aspectos de la norma ISO 25010 (Systems and Software Quality Models), que establece criterios clave para evaluar la calidad del software, tales como funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad, y se compone de 10 afirmaciones que se responden con una escala del 1 al 5 donde 1 es “en desacuerdo” y 5 “de acuerdo. Esta evaluación permitió recopilar información valiosa sobre las fortalezas y áreas de mejora del sistema, además de proporcionar recomendaciones para el desarrollo futuro de la plataforma.

A continuación, se detallan las 14 preguntas incluidas en la encuesta:

1. Funcionalidad

- ¿El software cumple con los requisitos funcionales? (1-5)
- ¿Está libre de errores críticos o fallas importantes? (1-5)

2. Usabilidad

- ¿La interfaz de usuario es intuitiva y fácil de navegar? (1-5)

- ¿Los usuarios necesitan poca capacitación para comenzar a usar el software? (1-5)

3. Fiabilidad

- ¿El software funciona de manera consistente bajo diferentes condiciones? (1-5)
- ¿Son raros los tiempos de inactividad o fallos? (1-5)

4. Rendimiento

- ¿El software responde rápidamente a las acciones del usuario? (1-5)
- ¿Maneja bien la carga bajo un alto uso? (1-5)

5. Mantenibilidad

- ¿El código está bien documentado y es fácil de entender? (1-5)
- ¿Se pueden implementar nuevas características o correcciones con poco esfuerzo? (1-5)

6. Seguridad

- ¿Se manejan los datos de manera segura dentro del software? (1-5)
- ¿Existen medidas efectivas para prevenir accesos no autorizados? (1-5)

7. Compatibilidad

- ¿El software es compatible con las plataformas y dispositivos necesarios? (1-5)
- No hay limitaciones al integrarse con otros programas (1-5)

Los resultados se detallan a continuación:

Pregunta	Respuesta 1	Respuesta 2
Pregunta 1	5	5
Pregunta 2	5	5
Pregunta 3	4	5
Pregunta 4	3	4
Pregunta 5	5	4
Pregunta 6	5	4
Pregunta 7	5	4
Pregunta 8	5	4
Pregunta 9	4	4
Pregunta 10	5	3
Pregunta 11	4	4
Pregunta 12	4	4
Pregunta 13	5	3

Pregunta 14	3	4

Tabla 12. Resultados encuesta de calidad técnica.

Como se puede observar, los resultados reflejan una tendencia positiva en cuanto a la calidad técnica del software, evidenciando un alto nivel de satisfacción por parte de la unidad informática de SIMPRO.

10. Conclusiones

A modo de conclusión, se puede destacar la importancia de este proyecto no solo como un reto académico, sino también como una oportunidad para poner en práctica los conocimientos adquiridos a lo largo de los cursos de la carrera. Este proyecto ha sido un espacio para consolidar una formación integral, abordando diferentes áreas de la ingeniería informática, desde lo técnico hasta lo metodológico, lo que contribuyó significativamente al desarrollo de habilidades analíticas y de resolución de problemas.

Sin duda, el camino hacia el aprendizaje y el perfeccionamiento es continuo y queda mucho por explorar. Sin embargo, este proyecto marca un hito en el proceso formativo, siendo una base sólida para enfrentar los desafíos futuros.

Con este proyecto se logró llevar a cabo el desarrollo desde 0 de una plataforma completa, agregando valor a la empresa y repercutiendo directamente en el crecimiento de esta.

Por otra parte, se logró identificar diferentes factores o puntos clave en el proceso de desarrollo de software que tienen un gran impacto en la calidad de este, cómo puede ser, testing, diseño y planificación.

10.1 Limitaciones

Una de las principales limitaciones fue el tiempo y es que la totalidad del desarrollo fue planificada en 8 sprints de 3 semanas cada uno, trabajando 10 horas por semana, lo que sin duda se tradujo en trabajo bajo presión y cumplir con plazos a coste de calidad técnica del software, lo que más adelante resultó en una enorme deuda técnica.

Otra de las grandes limitaciones fue la. Elección de tecnologías ya que al ser un software desarrollado para una empresa la cual solicitó el uso de ciertas tecnologías en particular, no se pudo hacer provecho de la experiencia en tecnologías ya conocidas y se dedicó gran parte del tiempo en capacitación.

11. Recomendaciones y trabajo futuro

De cara a trabajos futuros, y a pesar de haber abordado ciertas optimizaciones durante este proyecto, se recomienda profundizar en la mejora tanto del backend como del frontend. En el backend, sería útil modificar la forma en que se realizan las consultas a la base de datos, optimizando su estructura y el procesamiento de los datos para mejorar la eficiencia y reducir la latencia. En el frontend, se podrían implementar consultas por streaming, lo que permitiría manejar grandes volúmenes de información de manera más fluida, disminuyendo aún más los tiempos de respuesta y mejorando la experiencia del usuario. Estas mejoras garantizarán un sistema más ágil y preparado para enfrentar futuras necesidades tecnológicas.

12. Referencias

[1] G. Erboz, «How to Define Industry 4.0: The Main Pillars Of Industry 4.0., » 2017.

[2] Pontarolli Ricardo Pasquati, Jeferson André Bigheti, Lucas Borgues Rodrigues de Sá, y Eduardo Paciencia Godoy. (2023). “Microservice-Oriented Architecture for Industry 4.0”.

[3] Matías Caceres (2024). “LogBook: Sistema de gestión de eventos de mantenimiento enfocada a la industria de la gran minería.”

[4] Schwab, K. (2016). “*The Fourth Industrial Revolution”*. Geneva: World Economic Forum.

[5] Kagermann, H., Wahlster, W., & Helbig, J. (2013). “*Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry”; final report of the Industrie 4.0 Working Group*. Frankfurt am Main: Forschungsunion.

[6] Brooke, J. (1996). “SUS: A quick and dirty usability scale”. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, & I. L. McClelland (Eds.), Usability Evaluation in Industry (pp. 189-194). London: Taylor and Francis.