Universidad ORT Uruguay

Facultad de Ingeniería

Renovus: sistema de administración de archivos para parques eólicos.

Entregado como requisito para la obtención del título

Ingeniero en Sistemas

Alejandro Bermúdez - 222313

Diego Franggi - 210434

Federico Martínez - 210072

Tutor: Ricardo Szyfer

2023

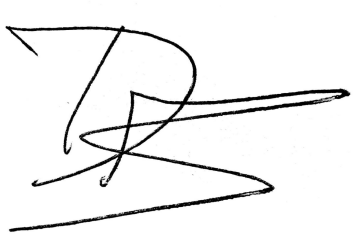
Declaratoria de Autoría

Nosotros, Alejandro Bermúdez, Diego Franggi y Federico Martínez, declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el proyecto final de carrera.
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad.
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra.
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros.
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Alejandro Bermúdez

30 de marzo de 2023



Diego Franggi

30 de marzo de 2023

Federico Martínez

30 de marzo de 2023

Abstract

Los parques eólicos generan una gran cantidad de datos durante las actividades de mantenimiento de sus equipos. En un solo parque se producen cientos de documentos por mes, que incluyen información sumamente importante para su funcionamiento. Sin embargo, las personas encargadas de la gestión suelen carecer del tiempo y los recursos necesarios para analizar y sintetizar toda esta información de manera detallada, ya que la misma suele estar dispersa y se pierde la oportunidad de obtener valiosas conclusiones que podrían mejorar el rendimiento, aumentar la energía generada, y, por lo tanto, los beneficios.

Estos datos suelen ser procesados por empresas que apoyan a los gestores de los parques, integrando distintas fuentes de información, permitiendo el monitoreo en tiempo real, generando reportes, alertas, y contextualización rápida de miles de informes de mantenimiento, entre otros beneficios.

La empresa Renovus, comprometida a lograr que los parques eólicos de sus clientes alcancen su máximo potencial, se presenta en la Feria de Proyectos de la Universidad ORT para solventar un problema vinculado a la gestión de la gran cantidad de documentos generados.

Los estudiantes tomaron este proyecto y llevaron a cabo un proceso con duración de un año que incluyó investigación, capacitación, análisis de requerimientos, diseño, planificación de arquitectura y construcción del sistema, así como su correspondiente documentación. Todo supervisado por rigurosos procesos como el de gestión de proyecto y el aseguramiento de calidad.

Al comienzo del proyecto, se plantearon un grupo de objetivos académicos, personales y de producto a fin de definir un horizonte y desarrollar un sistema de la mejor forma posible. El equipo logró cumplir con éxito los objetivos planteados así como también el desarrollo de un producto que funciona como SaaS (*Software as a Service*), es decir, un producto altamente eficiente que reduce el tiempo de búsqueda de documentos particulares a meros segundos, que antes podían tomar horas. El mismo, se encarga de analizar información de documentos mediante la utilización de un conjunto de diversas herramientas de la nube de Amazon, como por ejemplo Textract y Comprehend, así como una base de datos especializada en búsquedas de textos

llamada OpenSearch. De esta manera se proporciona una solución a los responsables de la gestión de parques eólicos que necesitan localizar, procesar y actuar sobre la información de miles de documentos de forma rápida.

Palabras Clave

Aerogeneradores, análisis, AWS, Comprehend, documentos, dotnet core, energías renovables, gestión, inteligencia artificial, OCR, OpenSearch, parques eólicos, Python, React Js, Renovus, Scrum, Textract, Universidad ORT Uruguay.

Glosario

Aerogenerador: Dispositivo utilizado para la generación de energía eléctrica a partir del viento.

Agile: Metodología de desarrollo de *software* que se centra en la adaptabilidad, la colaboración y la entrega continua.

API: Interfaz de programación de aplicaciones que permite la interacción entre diferentes aplicaciones o sistemas.

AWS: Amazon Web Services, plataforma de servicios de computación en la nube.

Azure: Plataforma de servicios de computación en la nube de Microsoft.

Backend: Parte de una aplicación no visible por el usuario que se encarga del procesamiento y almacenamiento de los datos.

Bitácora: Traducción de *log*, registro de eventos en el sistemas.

Branch: Una versión independiente del código fuente que se crea para desarrollar una nueva funcionalidad o solucionar un problema específico.

Bucket: Contenedor de almacenamiento en la nube.

Bug: Error o defecto en el *software* que causa un comportamiento no deseado o incorrecto.

Ciclo de vida: Serie de etapas por las que pasa un *software* desde su concepción hasta su obsolescencia.

Cloud computing: Modelo de entrega de servicios de computación en la nube.

Docker: Plataforma para la creación, el envío y la ejecución de aplicaciones en contenedores.

Feedback: Retroalimentación sobre el desempeño o la calidad de una aplicación, proceso o producto.

Forked: Proceso de copiar un repositorio de código fuente para desarrollar una versión independiente.

Framework: Conjunto de herramientas y bibliotecas de *software* que se utilizan para desarrollar aplicaciones.

Frontend: Parte de una aplicación que se encarga de la interacción con el usuario.

Función Lambda: Función de AWS que se utiliza para ejecutar código en respuesta a eventos.

GitFlow: Modelo de ramificación y fusión de ramas en Git.

Hardware: Componentes físicos de un sistema informático.

Historia de usuario: Descripción de una funcionalidad de una aplicación escrita desde la perspectiva del usuario final.

Hosting: Servicio que permite alojar una aplicación en un servidor y hacerla accesible a través de Internet.

IA: Inteligencia artificial, conjunto de tecnologías y técnicas que permiten a las máquinas realizar tareas que requieren inteligencia humana.

IAM: Identidad y administración de acceso, conjunto de prácticas y herramientas para la gestión de identidades y accesos de los usuarios.

IDE: Entorno de desarrollo integrado, conjunto de herramientas de *software* para la programación.

Kanban: Metodología ágil de gestión de proyectos que se centra en la visualización del flujo de trabajo.

Mergear (hacer *merge*): Proceso de combinar cambios de diferentes ramas de código en una sola rama.

Merged: Rama de código en la que se han fusionado cambios de diferentes ramas.

Metodología ágil: Enfoque de gestión de proyectos que se centra en la adaptabilidad, la colaboración y la entrega continua.

Microservicio: Arquitectura de *software* en la que una aplicación se compone de servicios independientes y autónomos.

Mínimo producto viable (MVP): Producto que tiene suficientes características para satisfacer a los primeros usuarios y obtener retroalimentación valiosa, pero que aún no está completamente desarrollado.

Mockup: Representación visual del diseño de una aplicación o sitio *web*.

OCR: Siglas de "Optical Character Recognition", tecnología utilizada para la lectura y reconocimiento de texto en imágenes o documentos escaneados.

Planning poker: Técnica utilizada en metodologías ágiles para estimar el esfuerzo de las tareas o historias de usuario en un proyecto.

Pull request: Solicitud de incorporación de cambios realizados en una rama de código a otra rama del repositorio.

Refactor: Proceso de reorganización del código existente con el objetivo de mejorarlo, hacerlo más legible o fácil de mantener, o mejorar su rendimiento.

Release: Lanzamiento de una nueva versión de un *software* o aplicación.

REST: Siglas de "Representational State Transfer", estilo arquitectónico utilizado en el desarrollo de aplicaciones *web* que se basa en el protocolo HTTP.

RESTful: Término utilizado para describir un servicio *web* que sigue los principios del estilo arquitectónico REST.

Scrum: Marco de trabajo utilizado en el desarrollo de *software* que se basa en metodologías ágiles y se enfoca en la entrega continua de valor al cliente.

Secuencia de Fibonacci: Serie de números en la que cada número es la suma de los dos números anteriores. Se utiliza a menudo en la estimación del esfuerzo en proyectos de desarrollo de *software*.

Serverless: Modelo de arquitectura de *software* en la que el proveedor de servicios en la nube se encarga de la gestión de la infraestructura y cobra por el uso.

Software: Conjunto de programas, datos y archivos necesarios para el funcionamiento de una computadora o sistema informático.

Software as a Service (SaaS): Modelo de entrega de *software* en el que el proveedor ofrece acceso a una aplicación a través de internet y cobra por su uso.

Sprint: Periodo de tiempo en el que se desarrolla una parte del trabajo en el marco de un proyecto ágil.

SQL: Lenguaje de programación utilizado para el manejo y gestión de bases de datos relacionales.

Tradeoff: Toma de decisiones en la que se acepta un compromiso entre diferentes aspectos de un proyecto, por ejemplo, entre el tiempo, la calidad y el costo.

UI/UX: Siglas de "User Interface" y "User Experience", conjunto de disciplinas relacionadas con la experiencia del usuario en la utilización de una aplicación o sitio *web*.

WebAPI: Interfaz de programación de aplicaciones que permite la comunicación entre diferentes aplicaciones a través de internet.

Índice

1. Introducción

Este documento tiene como propósito describir el proyecto de grado “Renovus: Sistema de administración de archivos para parques eólicos”, llevado a cabo desde abril de 2022 hasta marzo de 2023, como parte de los requisitos para obtener el Título de Ingeniero en Sistemas de la Universidad ORT Uruguay.

El objetivo del proyecto fue desarrollar una solución tecnológica innovadora para abordar un problema específico en el campo de los sistemas de información referentes a los altos volúmenes de documentos producidos por los generadores de energías renovables para la empresa Renovus.

A lo largo del proyecto, se aplicaron diversas metodologías, así como técnicas de investigación y análisis para identificar las necesidades y requerimientos del usuario, y con ello diseñar una solución que cubriera dichas necesidades.

El proyecto contó con un equipo de estudiantes y tutores, quienes colaboraron estrechamente en cada etapa del proceso para garantizar la calidad y eficacia de la solución desarrollada.

Es importante destacar que al momento de comenzar este proyecto, la empresa Renovus ya contaba con un sistema en funcionamiento que permitía a sus clientes el monitoreo y control de sus parques eólicos. Sin embargo, identificaron una oportunidad de mejora para agregar la posibilidad de subir y administrar documentos y tareas relacionadas con dichos parques, a través de una plataforma.

Es aquí donde entra en juego el equipo estudiantil, quienes asumieron la responsabilidad de diseñar y desarrollar estas nuevas funcionalidades y agregarlas al sistema ya existente. Trabajando estrechamente con Renovus, el equipo logró desarrollar una solución escalable y efectiva que permite a la empresa ofrecer a sus clientes, la posibilidad de mejorar su gestión de información y consecuentemente sus procesos internos. Como resultado, la empresa puede brindar un servicio más completo y personalizado para así continuar con su contribución a la transición hacia un modelo energético sostenible y eficiente.

1.1 Elección del proyecto

Para la selección del proyecto es importante conocer los perfiles de los alumnos encargados en elegirlo:

- **Alejandro Bermúdez:** QA Engineer - Clockwork.
- **Diego Franggi:** Hacker Ético - Deloitte.
- **Federico Martínez:** Mobile Developer - Moove It.

Con estos perfiles que abarcan ramas de conocimientos bastante diferenciadas dentro del campo de la tecnología, el equipo se plantea la selección de un proyecto presentado por la facultad en la llamada “Feria de Proyectos”. Durante este proceso diversas empresas postulan sus ideas y es responsabilidad de los estudiantes seleccionar el que más les atraiga para así obtener el título de grado.

Renovus no fue la elección inicial del equipo, su propuesta parecía intimidante y abarcaba áreas de conocimiento como la ciencia de datos y la inteligencia artificial (IA) que el equipo desconocía. Sin embargo, al no obtener el proyecto solicitado inicialmente, los estudiantes optaron por el desafío de continuar con este.

Aunque no era su elección inicial, la selección de Renovus resultó ser una suerte para el equipo, ya que les permitió ampliar su abanico de conocimientos y adentrarse en áreas en las que quizás no habrían incursionado de otra manera.

1.2 Objetivos

Al iniciar el proyecto, el equipo se enfocó en definir los objetivos a cumplir, para considerar el mismo como exitoso, logrando de este modo estar alineados con una meta definida desde el punto de partida.

Se opta por dividir estos objetivos en tres grupos principales: académicos, de equipo y de producto.

Académicos:

- **Lograr la aprobación del proyecto final de carrera:** Uno de los objetivos planteados por el equipo es que el proyecto cumpla los estándares de calidad esperados por la Universidad ORT y por consiguiente, los de la Industria a fin de obtener el Título de grado.

- **Aplicación de conocimientos relacionados con gestión y arquitectura de *software* adquiridos durante la carrera en un contexto real:** Remover las barreras de la teoría y la práctica, aplicando los conceptos y conocimientos ganados en la facultad, en un contexto real donde puedan dar valor tangible a un cliente.

Personales

- **Aprender a utilizar nuevas tecnologías y expandir nuestro conocimiento:** Para la correcta construcción de una solución que cumpla lo esperado por el cliente, el equipo ha de aprender y manejar tecnologías desconocidas de forma que cumpla con los estándares definidos en el plan de SQA.

- **Lograr una dinámica de trabajo en la que se obtenga un promedio mayor o igual a 3 en la evaluación interna del trabajo en equipo por cada integrante:** El equipo se esforzará por establecer una dinámica de trabajo colaborativa y eficiente que permita a cada integrante contribuir de manera equitativa y, por ende, lograr una evaluación satisfactoria por parte del equipo en su desempeño.

- **Aportar a una empresa cuyo negocio es el de mejorar la gestión de parques de energías renovables:** El equipo se compromete a trabajar con un enfoque en la sostenibilidad y la mejora del medio ambiente al colaborar con una empresa cuyo objetivo es mejorar la gestión de parques de energías renovables.

Producto:

- **Construir un producto que satisfaga las necesidades del cliente establecidas en los requerimientos:** El equipo se enfocará en comprender y satisfacer las necesidades del

cliente, asegurándose que el producto final entregado cumpla con los requisitos y expectativas establecidos.

- **Desarrollar una solución que cumpla los parámetros establecidos en QA:** El equipo se asegurará de desarrollar una solución de alta calidad que cumpla con los parámetros y estándares establecidos en los procesos de control de calidad.

- **Generar los entregables acordados con el cliente:** El equipo se compromete a generar y entregar todos los entregables acordados con el cliente en tiempo y forma, asegurándose de cumplir con todas las expectativas y requerimientos establecidos en el proyecto.

1.3 Estructura del documento

Sobre el problema

Se brinda contexto sobre la empresa cliente y cómo manejan sus operaciones al día de hoy, así cómo se realiza una introducción al problema que hemos de resolver y cómo afecta a los diferentes interesados.

Descripción de la solución

Se realiza un análisis de la solución de alto nivel, describiendo los cambios en el alcance a lo largo del tiempo.

Desarrollo y Proceso

Se describen todos los elementos que conforman la metodología de trabajo referente al proyecto, sus particularidades, cómo se realizaron los procesos de relevamiento de requerimientos y la gestión del trabajo.

Arquitectura

Se explican y justifican todas las decisiones arquitectónicas relevantes al proyecto, describiendo las mismas con diferentes vistas y haciendo hincapié en los atributos de calidad seleccionados junto a las tácticas relevantes que fueron empleadas.

Gestión de proyecto

En este capítulo se detalla la gestión del proyecto y sus diferentes etapas. Se describen las diferentes gestiones realizadas, como la de riesgo y configuración, así como las herramientas utilizadas y las métricas resultantes de estos procesos.

Gestión y aseguramiento de calidad

Se detallan actividades y procesos relacionados con la calidad del proyecto, describiendo actividades y planes pertinentes.

Conclusiones y lecciones aprendidas

En este capítulo se realiza una retrospectiva del proyecto en general, se busca brindar una visión objetiva del proceso y trabajo realizado, se identifican las oportunidades de mejora, así como también se lleva a cabo el análisis de los objetivos planteados al inicio del proyecto.

2. Sobre el problema

2.1 La empresa cliente

Renovus es una solución tecnológica desarrollada por una empresa homónima que utiliza ciencia de datos e inteligencia artificial para ayudar a empresas de energías renovables a mejorar su rendimiento. [1]

La solución está dirigida a las empresas propietarias y gestores de parques eólicos y solares, así como a los responsables de *O&M (Operations and Maintenance)* que necesitan realizar un seguimiento de sus equipos. Estos equipos generan una gran cantidad de datos que pueden resultar difíciles de analizar y organizar, perdiéndose entonces, oportunidades de mejorar la performance y consecuentemente la rentabilidad de los parques.

El equipo de Renovus cuenta con más de 10 años de experiencia en energías renovables, tecnología, *marketing* y finanzas, y se compromete a ayudar a los activos renovables y los equipos de gestión a alcanzar todo su potencial. La misión de la empresa es llevar las mejores soluciones tecnológicas a aquellos responsables de la gestión y mantenimiento de parques eólicos y solares.

Tan solo diez meses después de su inicio en el Centro de Innovación y Emprendimientos de la Universidad ORT [2], en diciembre de 2021, Renovus fue seleccionado por la Agencia Nacional de Investigación e Innovación (ANII) [3] en el marco de la convocatoria de emprendimientos innovadores, denotando el crecimiento acelerado de esta empresa y el potencial de su solución tecnológica en el mercado de las energías renovables.

2.2 El problema propuesto

2.2.1 Situación actual de Renovus

Hasta el momento, Renovus permite el acceso a tres grandes funcionalidades, para optimizar el rendimiento y eficiencia de parques generadores de energía eólica y solar de sus clientes, a través de una cómoda interfaz *web*.

En primer lugar, ofrece un monitoreo preciso de la producción energética en megavatios hora (MWh), la velocidad del viento (m/s) y la irradiación solar en (kW/m2), mientras que puede generar alertas si la performance de un generador es baja. Además, monitorea tendencias climáticas y el impacto de los pronósticos en los costos y producciones energéticas futuras.

En segundo lugar, Renovus proporciona un análisis detallado de las turbinas de viento y su generación a lo largo de semanas, meses o años, así como un análisis de la curva de poder de los aerogeneradores [4]. Este análisis permite identificar áreas de mejora y optimizar la eficiencia energética.

Finalmente, la solución de Renovus ofrece diagnósticos precisos y producción de reportes detallados de los generadores en el sistema, los cuales pueden ser enviados al correo electrónico del usuario. Esto permite a los clientes tener un registro detallado del rendimiento de sus parques y tomar decisiones informadas sobre mejoras y mantenimiento.

2.2.2 El problema

Es en este contexto planteado, Renovus identifica un problema en la administración de parques eólicos, cuya solución, podría agregar valor a su portafolio de servicios, fundamentalmente orientado a este tipo de empresas: la gestión de grandes cantidades de documentos generados por el mantenimiento de los equipos que componen los parques.

En un solo parque eólico, por ejemplo, se generan millones de datos relevantes por mes, y decenas de documentos que incluyen textos, imágenes y más información significativa (como se puede ver en el documento de ejemplo del Anexo 10.1). Para contextualizar un poco más, un parque eólico promedio de Uruguay con aproximadamente 20 aerogeneradores produce por año alrededor de 244 documentos, lo cual implica una gran cantidad de fuentes de información que ha de ser analizada para un único parque de los múltiples que podría tener una sola empresa cliente de Renovus.

A menudo, las personas encargadas de gestionar parques no tienen el tiempo o los recursos necesarios para analizar, sintetizar y organizar todos estos datos y documentos. Como resultado, la información puede quedar dispersa y se desaprovecha la oportunidad de extraer valiosas

conclusiones que puedan mejorar el rendimiento, aumentar la producción energética y con ello los ingresos.

Los documentos a procesar pueden ser de todo tipo, pero están vinculados a los parques eólicos y al mantenimiento que conlleva mantenerlos en funcionamiento. Por ejemplo, documentos de planificación de trabajos, de inspección de equipos (como palas de aerogeneradores) y de trabajos realizados, de los cuales se busca extraer información valiosa como fecha, número de aerogenerador, título de informe y conclusiones. Toda esta información procesada en tiempo y forma, podría simplificar enormemente la capacidad de análisis del encargado de mantenimiento al tenerla en un solo lugar y no dispersa en documentos que llegan a ocupar en promedio casi cincuenta páginas. Lograr reducir la búsqueda y vista de datos relevantes de un archivo particular a segundos, permite ahorrar tiempo y mejorar la eficiencia en el análisis de la información, actividad que tradicionalmente suele requerir de varias horas.

Para abordar este problema y agregarlo a su solución SaaS destinada a clientes con parques eólicos, Renovus busca la ayuda de estudiantes de la Universidad ORT para desarrollar un sistema de administración de archivos que permita la extracción de datos, así como la generación de resúmenes y tareas a partir de los mismos.

El objetivo final del proyecto es desarrollar un sistema que permita cargar y organizar los archivos relacionados con los generadores de los parques eólicos, y que, además, posibilite la extracción de valores clave de los documentos, para facilitar su análisis y búsqueda por parte de los responsables de los parques. Adicionalmente, se busca la posibilidad de producir tareas específicas para cada aerogenerador. Por ejemplo, si un supervisor de operación y mantenimiento revisa una inspección realizada a un equipo y detecta una tarea de seguimiento, se generaría un "ticket" dirigido a la persona adecuada. [5]

Si el resultado del proyecto satisface a Renovus es posible que el mismo se integre a su sistema *web* en el futuro, aunque en la actualidad, solo se busca crear un mínimo producto viable (MVP).

En resumen, el proyecto busca solucionar el problema de la gestión de grandes cantidades de datos y documentos generados por parques eólicos, mediante la creación de un sistema de administración de archivos que permita la extracción de datos relevantes y la generación de

tareas específicas para facilitar el estudio y seguimiento de los de los generadores de aquellos clientes de Renovus.

2.3 Objetivos del cliente

A modo de recapitulación el equipo en diálogo con los representantes de Renovus, identificó los siguientes objetivos a realizar:

- **Solucionar el problema de la gestión de grandes documentos generados por los parques eólicos:** construir un sistema capaz de soportar la creciente demanda de cientos de documentos por parque, mediante el uso de herramientas escalables que permitan realizar las acciones de gestión a definir en el alcance.

- **Desarrollar un sistema que permita cargar y organizar los archivos relacionados con los generadores de los parques eólicos:** a fin de realizar la correcta gestión, los archivos han de poder ser cargados al sistema y permitir ser encontrados más adelante.

- **Posibilitar la extracción de valores clave de los documentos para facilitar su análisis y búsqueda por parte de los responsables de los parques:** mediante el uso de técnicas avanzadas de procesamiento de lenguaje natural y análisis de datos, el sistema permitirá la identificación y extracción precisa de información relevante contenida en los documentos de los generadores, tales como la fecha, número de aerogenerador y conclusiones. Además, el sistema brindará una opción de búsqueda en base a estos campos, lo que permitirá a los responsables de los parques eólicos encontrar rápidamente la información que necesitan para tomar decisiones informadas.

- **Posibilitar la generación de tareas específicas para facilitar el estudio y seguimiento de los generadores de aquellos clientes de Renovus:** a través del sistema desarrollado, los responsables de los parques podrán generar tareas específicas para cada aerogenerador en base a la información relevante extraída de los documentos, lo que permitirá un seguimiento más efectivo del estado de los generadores.

Estos objetivos fueron los que moldearon el alcance inicial y final de la solución y permitieron al equipo de desarrollo, enfocar sus esfuerzos en la creación de un sistema de administración de

archivos que cumpliera con las necesidades específicas de los clientes de Renovus. Además, estos objetivos también sirvieron como guía para la evaluación del éxito del proyecto al comparar los resultados obtenidos con los objetivos establecidos.

2.4 Interesados y necesidades

A continuación se presentan las necesidades de cada uno de los interesados en beneficiarse de la solución al problema planteado por Renovus:

- **Los encargados de administración de parques eólicos y solares:** necesitan un sistema de administración de archivos y una herramienta de seguimiento de tareas para mejorar la eficiencia y productividad de sus parques.

- **Los inversores y accionistas de Renovus:** requieren de un servicio más competente que les permita realizar una gestión eficiente de la información que producen los equipos, lo cual podría producir un aumento de la rentabilidad y un mejor posicionamiento de la empresa en el mercado.

- **El equipo estudiante de la Universidad ORT:** dentro de los requisitos para la obtención del Título de Ingeniero en Sistemas, deberán responder a la necesidad de un cliente, y en este caso en particular, trabajarán el desarrollo de un sistema de administración de archivos, permitiéndoles aplicar sus habilidades y conocimientos a un proyecto práctico y colaborar con una empresa líder en energía renovable.

3. Descripción de la solución

3.1 Descripción funcional de la aplicación

En esta sección comentaremos sobre el alcance de la aplicación a nivel funcional. Para esto se expondrá el proceso de transformación que sufrió la misma, ya que se partió de una idea que, luego de algunas reuniones, intercambios de ideas, y de análisis realizados, fue mutando hasta llegar a lo que hoy es el alcance final del producto.

3.1.1 Alcance inicial

En un principio, el alcance de este proyecto fue definido por Renovus y compartido por la Universidad ORT a través de *sharepoint* [5] donde, a grandes rasgos, el alcance definido era el siguiente:

- Carga/*Upload* de documentos (PDF y Excel).
- Organizar documentos por aerogenerador.
- Creación de *tickets*/tareas para cada aerogenerador a partir de los documentos cargados.
- Integración con el sistema Renovus.

Partiendo de esta base, el equipo se reunió con el cliente para profundizar sobre el proyecto, conocer de primera mano más detalles de lo que ellos esperaban y poder recabar la información necesaria para establecer el alcance que se presentaría en la postulación del proyecto (Anexo 10.2). Esta postulación, que fue determinada como alcance inicial por parte del equipo, incluyó los siguientes puntos:

- Carga/*Upload* de documentos.
- Gestión de documentos por aerogenerador y parque.
- Búsqueda de documentos utilizando diferentes parámetros (fecha, criticidad, tipo de documento).

- Prevención de documentos repetidos a través de la correcta organización de los mismos por diferentes factores.
- Generación de reportes estadísticos con base en fecha, criticidad, tipo de documento.
- Generación de respaldos adicionales.

También se definieron los siguientes aspectos no funcionales:

- Arquitectura basada en microservicios.
- Posible desarrollo de una *app* para Android e iOS.
- Despliegue en la nube de Amazon (AWS).
- Autenticación, autorización y *tenancy*: solo subir archivos de usuarios o dispositivos autorizados.
- Confiabilidad y disponibilidad.
- Observabilidad.
- Identificación de fallas y generación de *logs*.
- Performance de las consultas/búsquedas de documentos.

Este alcance se veía completo, y planteaba un desafío interesante para el equipo, pero tampoco sería el alcance final. Una vez iniciado el proyecto, se mantuvieron varias reuniones con el cliente, profundizando en sus necesidades, problemas actuales, y su visión a futuro. Esto trajo aparejado, modificaciones al alcance del problema que lo fueron transformando y aumentando de esta forma, el desafío al que se enfrentó el equipo. Asimismo, le brindó la posibilidad de generar una solución que le ofrecía aún más valor al cliente, motivo por el cual el equipo volvió a modificar el alcance del producto, pero esta vez ya con miras a dejarlo lo más establecido y estático posible de cara al futuro.

3.1.2 Alcance final

Cómo se comentó en secciones anteriores, una vez iniciado el proyecto, y luego de varias reuniones, el alcance volvió a verse modificado, pero como se mencionó, fue con la intención de aumentar el valor que se le iba a brindar al cliente. Dichos ajustes, también generaron un aumento en el desafío inicial que se proponía al comienzo del proyecto.

A medida que las reuniones de coordinación avanzaron, se fueron definiendo acuerdos donde se establecieron las características de un nuevo alcance, junto a la definición de documentos y entregables que serán presentados en los capítulos siguientes de este trabajo. Finalmente, se determinó la base sobre la cual el equipo trabajaría, siendo conscientes que podría surgir la necesidad de nuevas modificaciones, pero teniendo ahora el rumbo final del proyecto establecido.

Teniendo en cuenta este recorrido, el nuevo alcance del proyecto, establecía los siguientes puntos:

- Carga/*Upload* de documentos (PDF y Excel).
- Análisis mediante *IA* de los documentos.
- Procesamiento y almacenado de la información.
- Gestión de documentos en el sistema.
- Búsqueda de documentos en base a diferentes parámetros (fecha, tipo de documento, título, palabras clave dentro del mismo).
- Prevención de documentos repetidos a través de la correcta organización de los mismos por diferentes factores.
- Visualización de los documentos y su información.
- Creación y gestión de tareas por documento.

Llegados a este punto se quiso establecer claramente el tipo de proyecto a realizar, el producto a construir, y, si bien podían existir modificaciones en algunos requerimientos, o agregarse o quitarse alguno, ésta iba a ser la base sobre la cual se comenzaría a trabajar para alcanzar la construcción de un MVP (*Minimum Viable Product*). Un aspecto no menor, es que dicho MVP en caso de ser del agrado del cliente, pasaría luego a formar parte de su producto. Junto a esta definición del alcance funcional, también se establecieron aspectos no funcionales, manteniendo inicialmente los mismos puntos que se comentaron en la sección anterior, con una única diferencia que consiste en la realización de una *app mobile* para iOS y Android. Si bien no se descartó por completo, quedó relegada esta opción, como una posibilidad a futuro que sería evaluada junto al avance del proyecto y otros aspectos.

A medida que el proyecto avanzó, se realizaron algunas modificaciones sobre el alcance funcional, así como también sobre aspectos no funcionales. En la evolución del proyecto fueron surgiendo diferentes situaciones y problemas, que, gracias a la continua y positiva comunicación con el cliente, se pudo llegar a diferentes acuerdos que permitieron la correcta realización del proyecto. De esta manera, se logró alcanzar un producto que satisfizo a todas las partes. Algunos de los acuerdos alcanzados fueron los siguientes:

- Dada la complejidad programática que presentaba la conversión de archivos Excel al formato PDF y que ésta fuese exitosa, se decidió que la carga fuese solo de archivos PDF y que la conversión la realizara el usuario previo a la carga en el sistema considerando que no aumentaba la complejidad ni dificultad para la utilización de la solución.

- Se decide que el proyecto no va a ser un sistema externo a la solución Renovus, sino que el mismo se vería incorporado al sistema existente, para lo cual se le concede al equipo acceso al código propietario y se le brinda la información necesaria para poder trabajar sobre los módulos existentes de la solución.

- El análisis de los documentos con IA pasa a realizarse con servicios de AWS, aprovechando que ellos cuentan con su solución desplegada en ese entorno y que Amazon presenta servicios de IA más avanzados de los que el equipo podría desarrollar. Además cuenta con un mayor soporte, entre otros aspectos, lo que permitiría reducir el

porcentaje de errores de esta funcionalidad, así como también liberar cierto tiempo del equipo para dedicar a otros aspectos de la solución a construir.

- Dada una necesidad concreta del cliente, se modifica la funcionalidad de las tareas, y es que estas inicialmente iban a estar asociadas únicamente a un documento, pero entendiendo que pueden existir tareas que se vinculen a más de un documento, o que simplemente tengan dependencia del generador y no dependan de ningún documento, se ajusta el alcance de la funcionalidad para permitirle al cliente poder gestionar tareas con varios documentos asociados, así como también tareas que solo dependan de generadores.

3.2 Descripción general de la solución

3.2.1 Funcionalidad: Subida y análisis de archivos

Esta funcionalidad podemos dividirla en dos partes, primero la subida, visible al usuario, y la segunda parte el análisis, que para el mismo es transparente ya que el procesamiento se realiza en la nube. Para la carga y subida de documentos, se cuenta con una pantalla donde el usuario puede seleccionar el parque eólico para el cual quiere cargar los documentos, y además puede seleccionar el aerogenerador al cual se asociará el o los documentos cargados (Figuras 1 y 2).

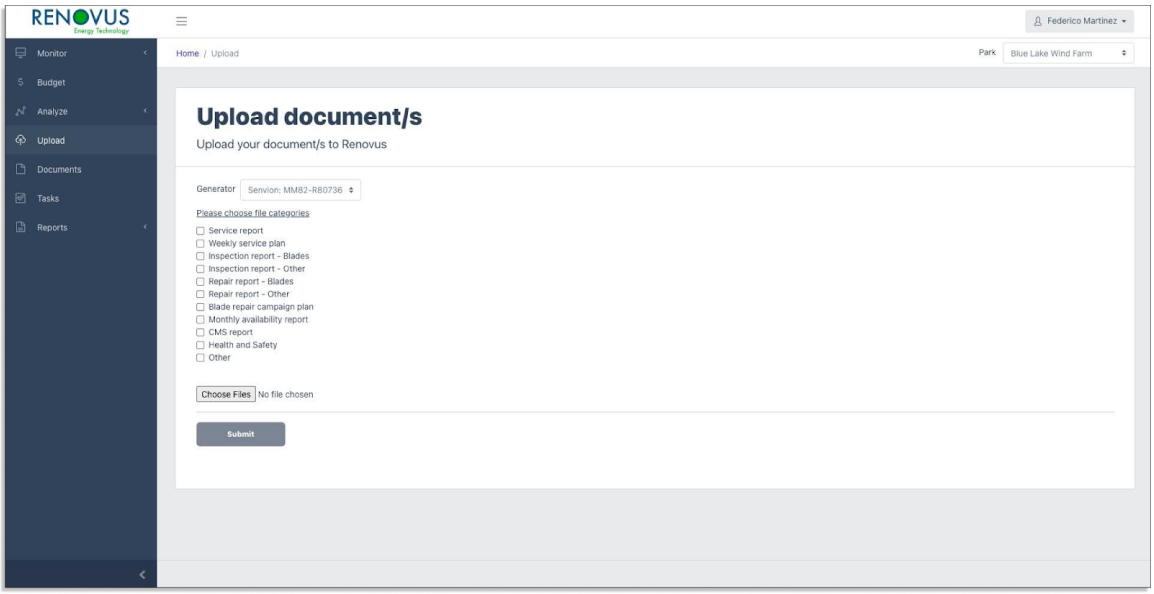


Figura 1: Pantalla Upload

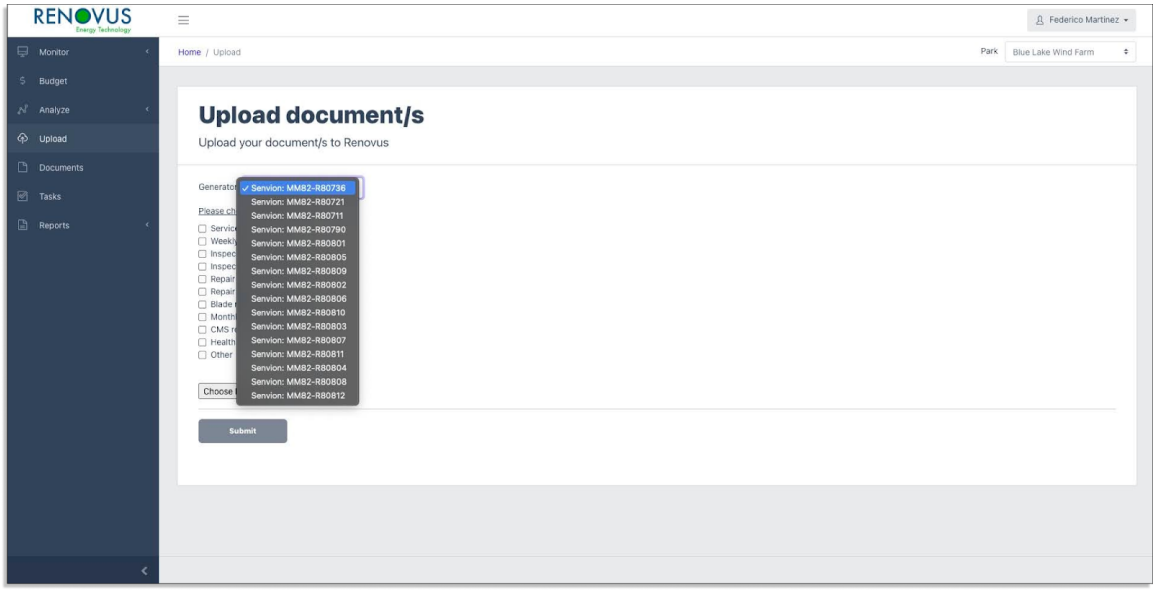


Figura 2: Selección de generador, pantalla Upload

En esta pantalla también contamos con las diferentes categorías que se pueden asociar al o a los documentos que el usuario desea cargar (Figura 3), y luego el botón para la selección de documentos, donde el usuario puede elegir para cargar desde su máquina uno o más documentos (Figura 4).

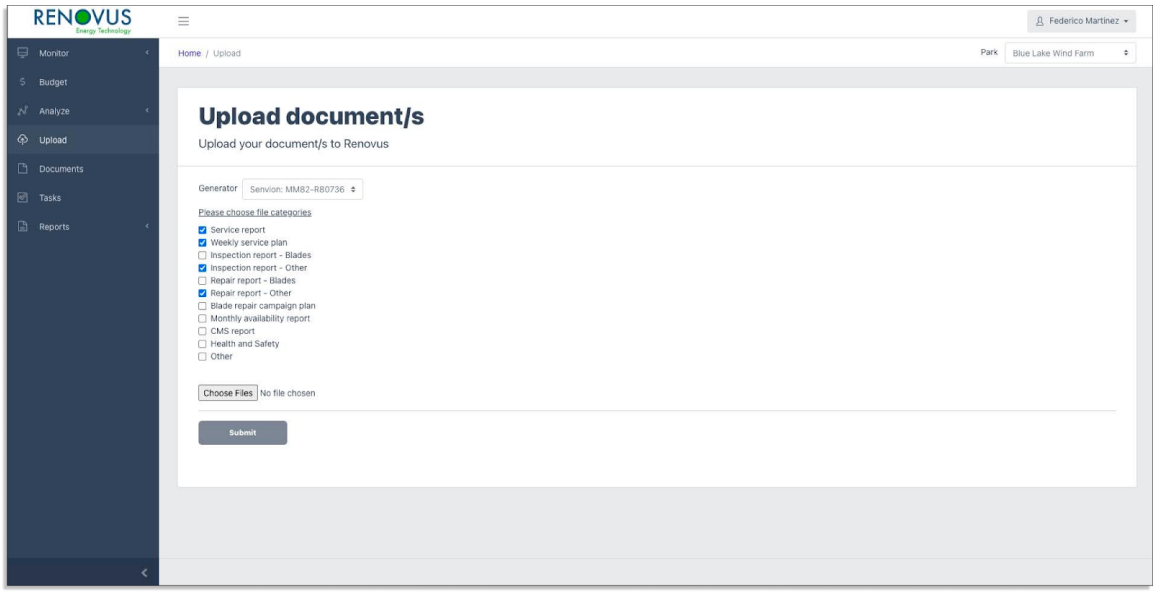


Figura 3: Selección de categorías, pantalla Upload

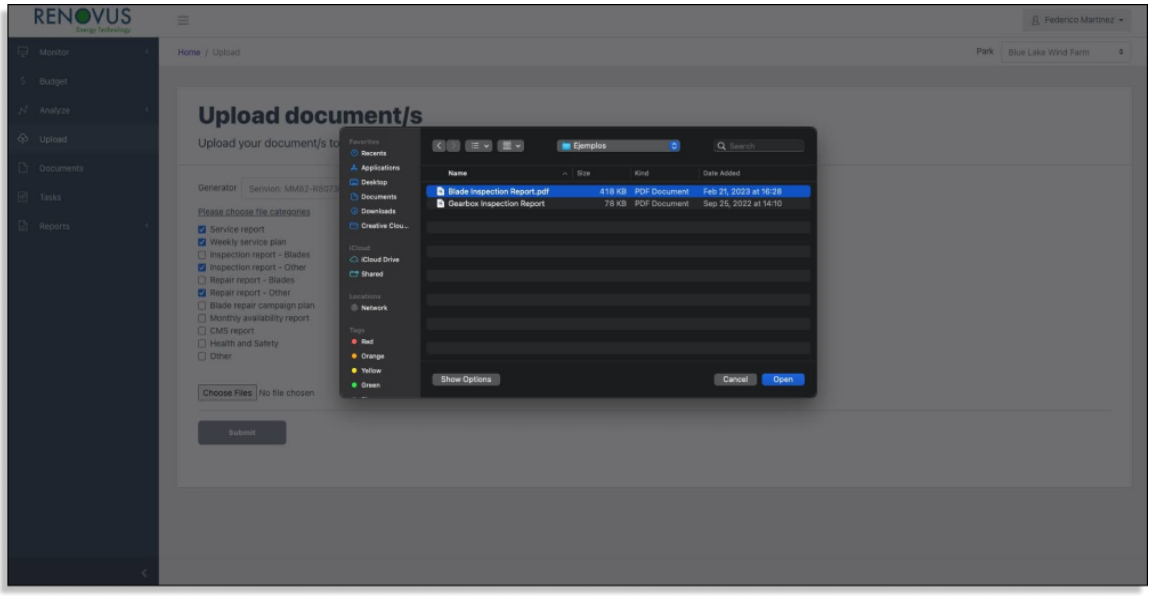


Figura 4: Selección de documento, pantalla Upload

Luego de este paso, cuando ya se cargaron los documentos y se seleccionaron las categorías, se habilita el botón de confirmación (“Submit”) y además se muestra en pantalla un nuevo botón para cancelar (“Cancel”),en caso de que sea necesario (Figura 5). Al confirmar la carga se muestra una pantalla con un ícono que indica que se está subiendo el documento al sistema, tanto para su análisis, como para su almacenamiento (Figura 6). Finalizada la subida del documento, aparece una pantalla que muestra si la carga fue exitosa o no (Figuras 7 y 8).

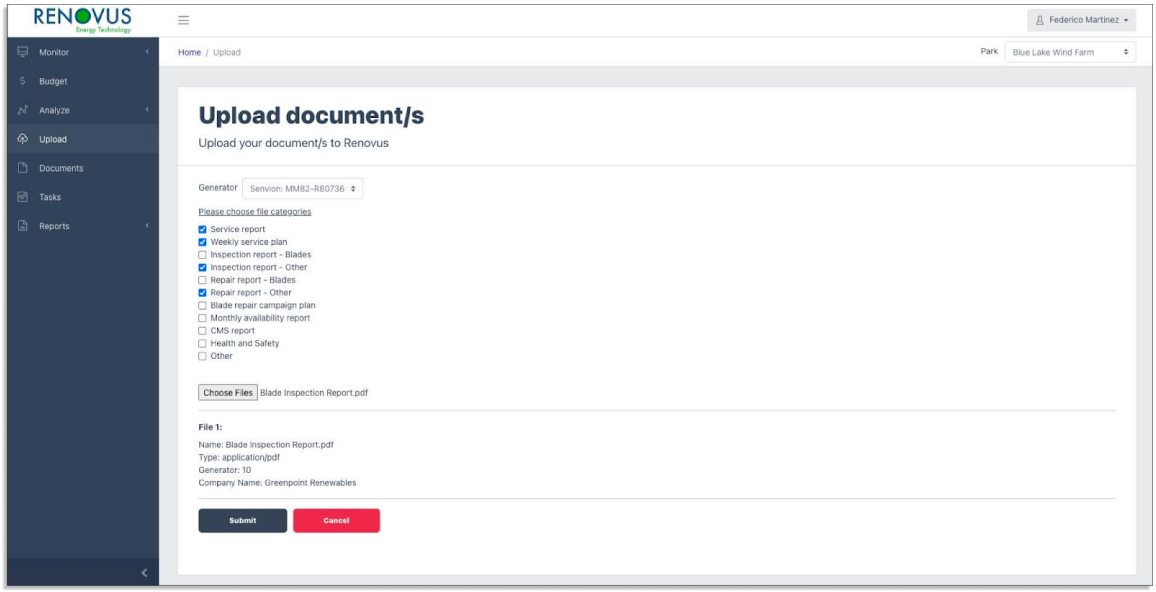


Figura 5: Documento seleccionado, pantalla Upload

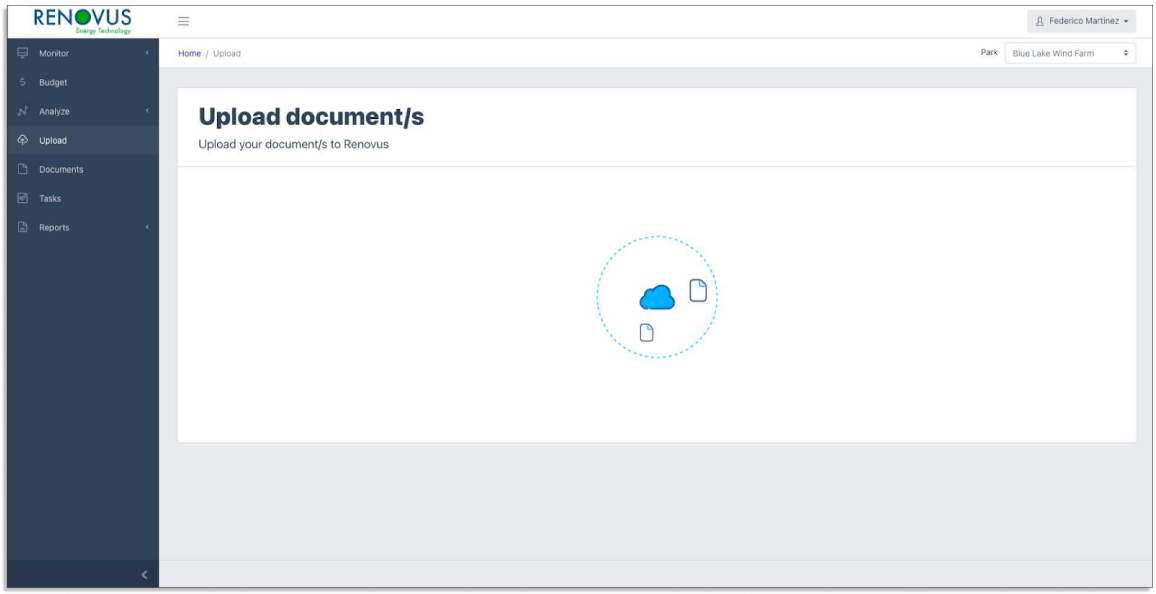


Figura 6: Documento en subida, pantalla Upload

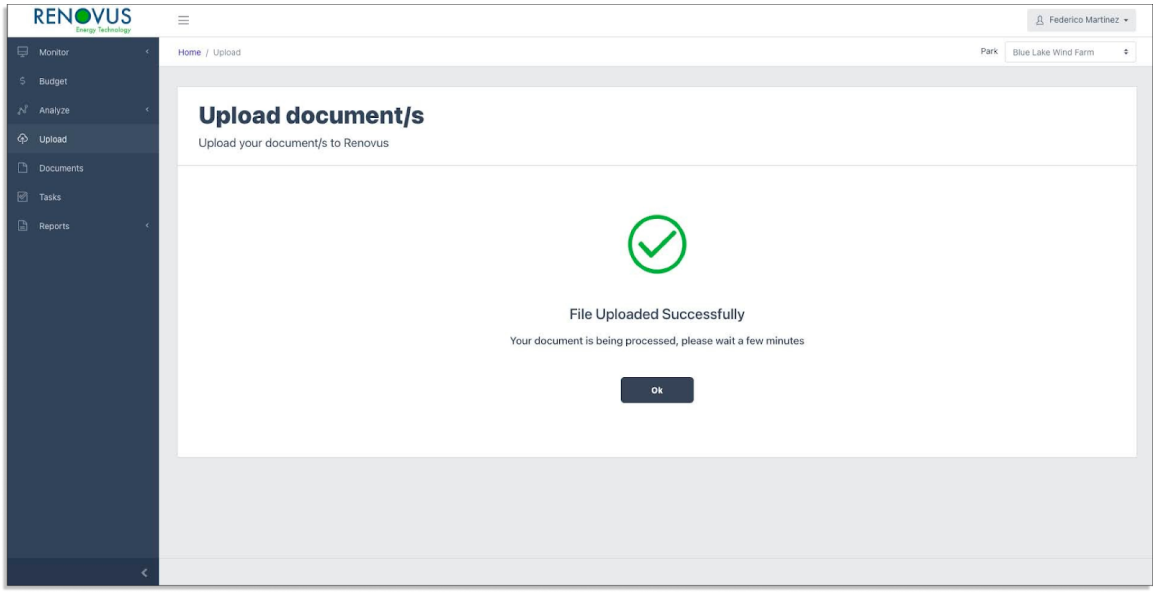


Figura 7: Documento subido exitosamente, pantalla Upload

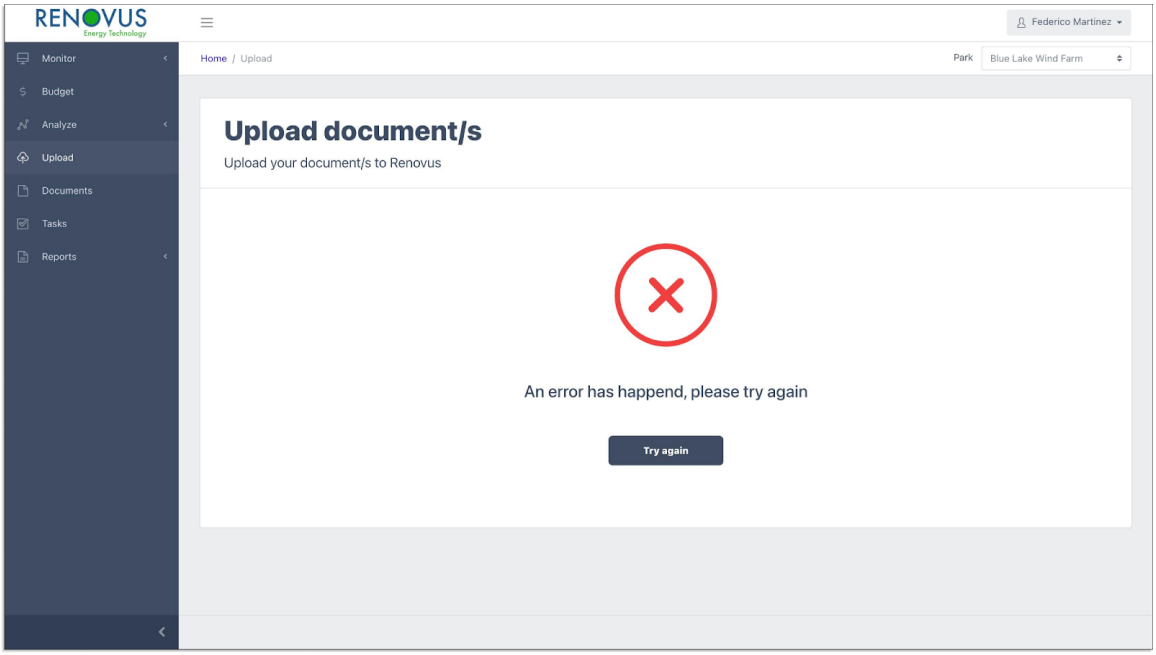


Figura 8: Falla en carga de documento, pantalla Upload

Si la carga de documentos es exitosa, los mismos se almacenan en un *bucket* S3, y ejecutan el disparador de AWS que se va a encargar de ejecutar el análisis de estos documentos. Una vez que comienza el análisis, estos son enviados al servicio de AWS denominado Textract, donde se

analiza el documento para obtener los valores claves del mismo, así como también todo el texto existente en el documento sin procesar, que luego será utilizado por otro servicio. Cuando Textract finaliza el procesamiento del documento, y devuelve la información, el texto plano devuelto, es reutilizado y enviado a otro servicio de Amazon llamado Comprehend, el cual lo analizará para detectar las entidades que se puedan hallar en el documento, así como también las frases principales y el idioma de este.

Luego de que ambos análisis finalicen y el sistema cuente con las respuestas, la información es enviada a una base de datos, donde se almacenará el resultado del análisis, así como también la ubicación del documento en el *bucket* mencionado anteriormente para facilitar la búsqueda del mismo en caso de que sea necesario.

3.2.2 Funcionalidad: Búsqueda rápida sobre archivos analizados

Para la búsqueda de archivos también se diseñó una nueva pantalla dentro del sistema Renovus en la cual encontraremos los filtros y campos por los cuales se puede buscar, así como también se visualiza una tabla donde se listan los documentos. Al igual que en la anterior, en esta pantalla contamos con la posibilidad de seleccionar el parque eólico con el cual queremos trabajar, así como también de qué generador queremos ver los documentos y los filtros que podemos utilizar, los cuales son: filtro por nombre de documento, por fecha, por palabras clave y/o por tipo de documento (Figuras 9 y 10).

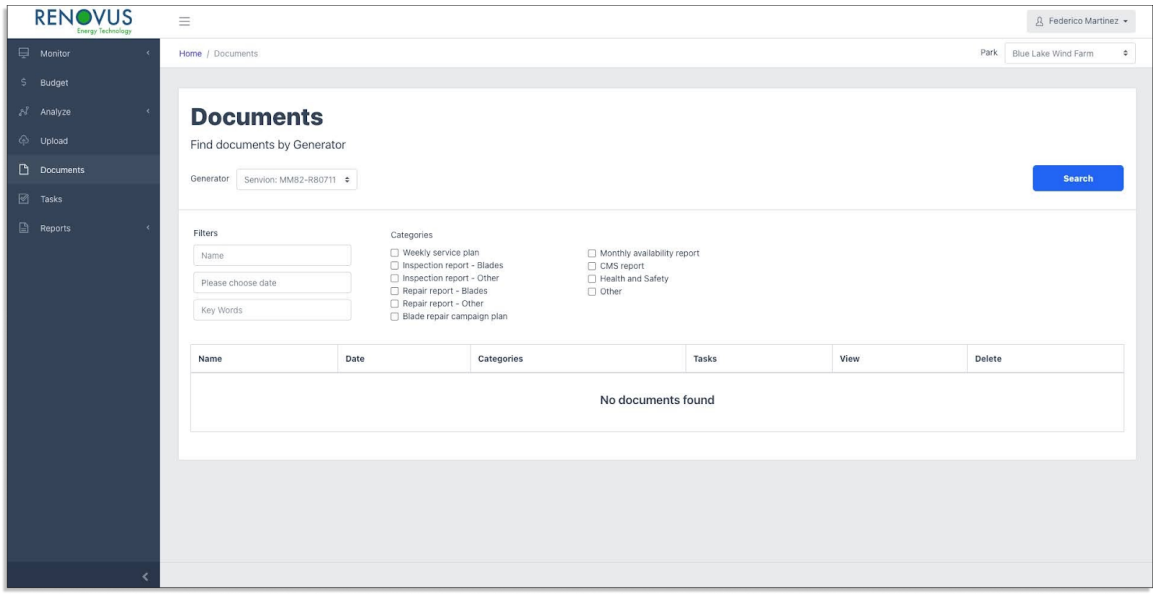


Figura 9: Pantalla Documents

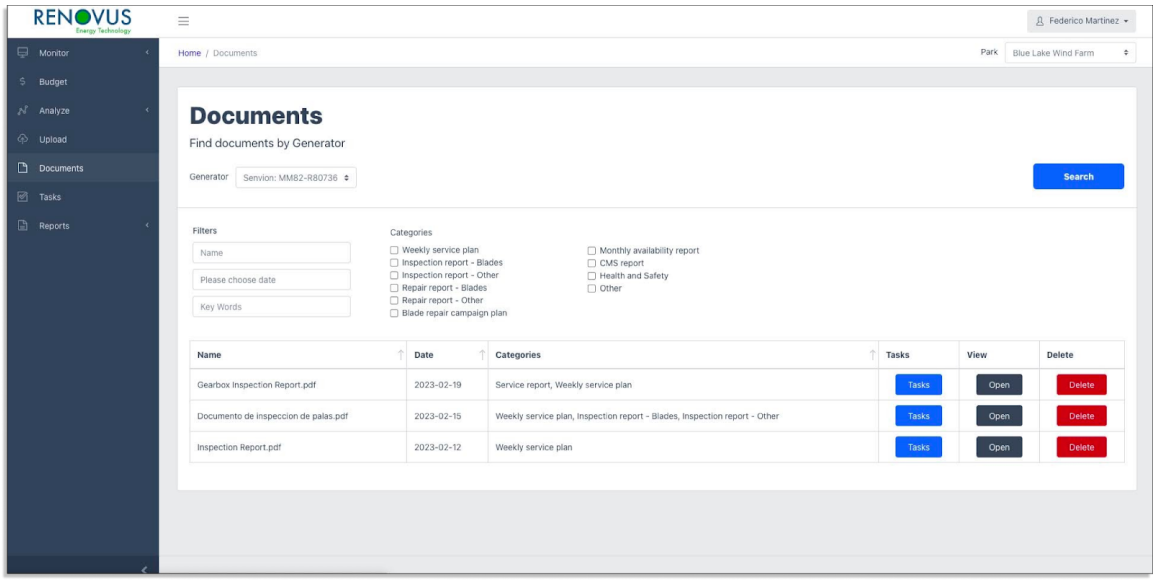


Figura 10: Pantalla Documents con documentos

Como se puede ver, ya al ingresar a la pantalla se trae por defecto un generador asociado al parque que el usuario tiene seleccionado, y con este se listan todos los documentos que el usuario tenga en una tabla que le permitirá ordenarlos por nombre, fecha o categorías definidas por los tipos de documentos. En caso de no tener documentos cargados para ese generador, la lista queda vacía.

Al momento de filtrar, como se comentó anteriormente, existen cuatro tipos de filtro (quitando parque y generador), que funcionan de la siguiente manera: el filtro con mayor prioridad es el de fecha, por lo que, cuando se utiliza, el resto se ignora. En otro caso, los otros tres pueden utilizarse al mismo tiempo, por lo que es posible filtrar por nombre/título del documento, palabras claves y categorías en simultáneo.

Entrando en mayor detalle, para filtrar por fechas es necesario ingresar un rango para que se realice la búsqueda. Por su parte, en el caso del filtro del nombre/título del documento no es necesario incluir el nombre del documento en su totalidad, podemos buscar documentos con una selección de palabras que lo conformen e incluso con una sola palabra y el sistema se encarga de devolver aquellos documentos que tengan ese nombre/título o contengan esas palabras. Para la búsqueda de palabras claves o frases claves, no es necesario incluir toda la frase, o que estén en orden las palabras ingresadas. El sistema se encarga de realizar la búsqueda contemplando ciertos aspectos e intercambiando el orden de las palabras para traer los resultados más acertados para la búsqueda (Figuras 11 y 12).

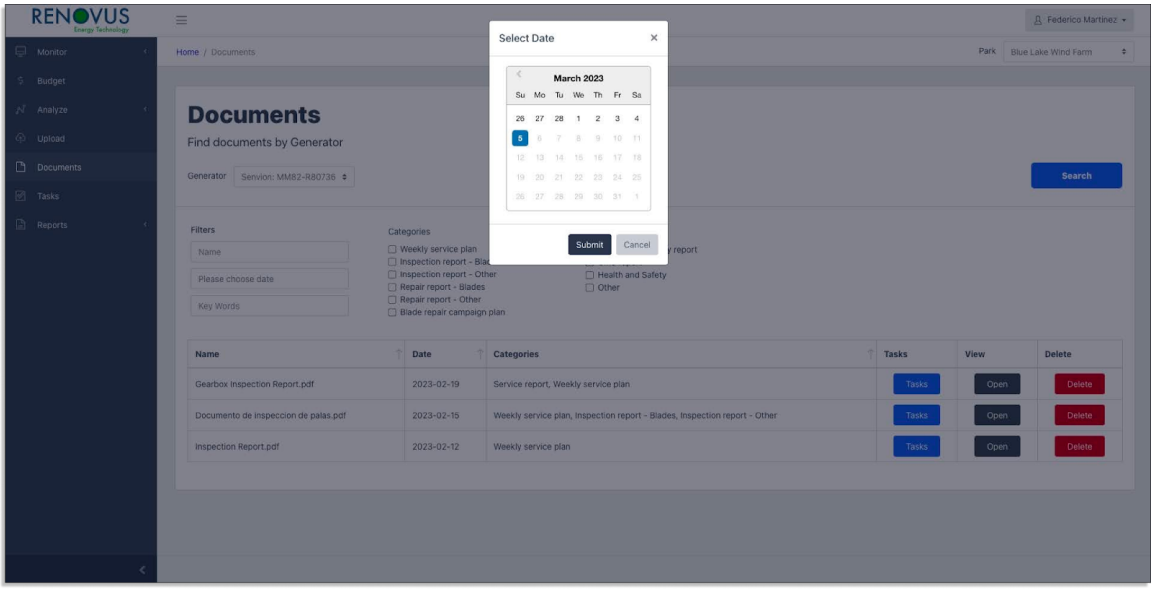


Figura 11: Pantalla Documents selección de fecha

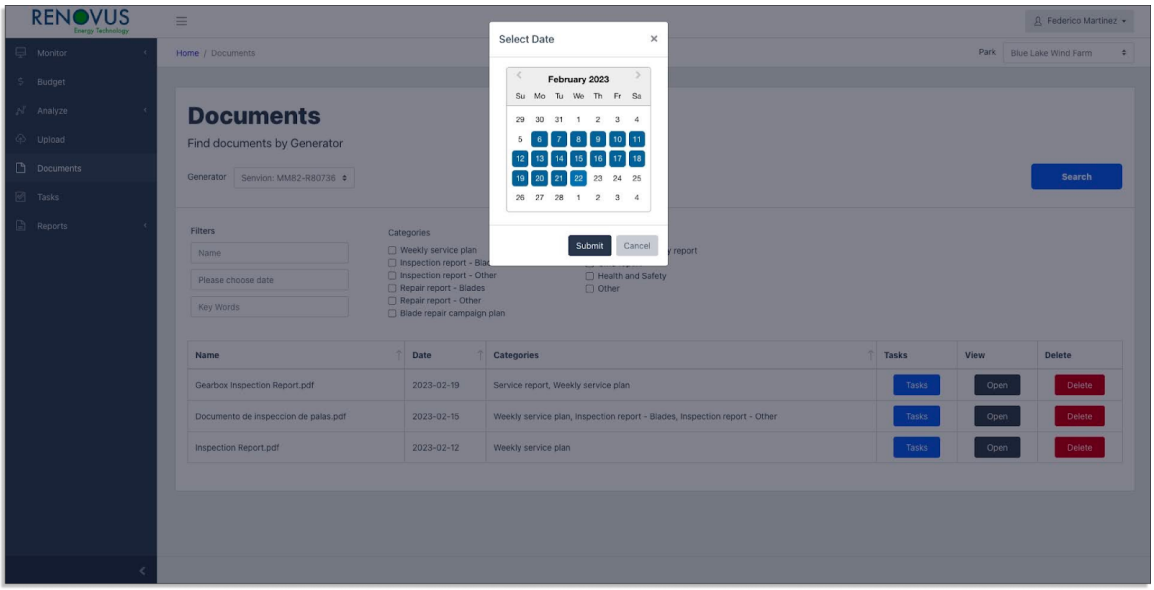


Figura 12: Pantalla Documents selección de rango de fechas

Finalmente, el filtro de categorías se encarga de filtrar aquellos documentos que tengan la o las categorías seleccionadas al momento de realizar la búsqueda. Todo este funcionamiento se detalla más adelante con mayor profundidad. De momento solo se pretende que el lector entienda a un alto nivel cómo funcionan los filtros. También cabe mencionar que luego de filtrar y darle al botón de buscar (“Search”), aparece un nuevo botón a la izquierda de este, que es para limpiar los filtros de búsqueda (Figura 13).

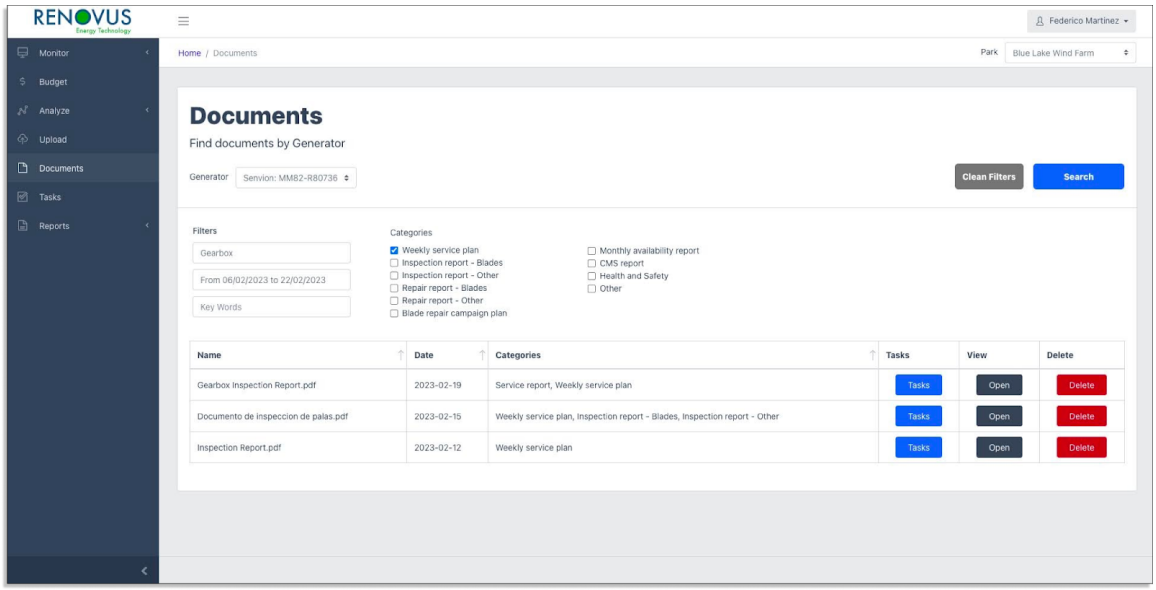


Figura 13: Pantalla Documents selección de filtros

En esta pantalla también se pueden gestionar los documentos, la visualización de los mismos, la administración de tareas asociadas (funcionalidad que comentaremos en la siguiente subsección), y la eliminación de estos. Al querer visualizar un documento se oprime el botón “Open” y este redirecciona a una nueva pantalla en la cual veremos no solo el PDF desplegado, con la posibilidad de hacerle *zoom*, descargarlo, ver las notas que el mismo tuviera al momento de cargarlo en la plataforma, sino que también despliega información del mismo.

Inicialmente, se muestra el nombre del archivo, la fecha en la que se cargó y las categorías a las cuales pertenece, pero existe la opción “Show more” que al hacerle clic despliega en el lateral del documento y en la parte inferior dos tablas que muestran diferentes informaciones, desde datos, como por ejemplo en dónde está almacenado o el nombre de la compañía, hasta información relevante extraída del análisis, realizado por la inteligencia artificial (Figuras 14, 15 y 16).

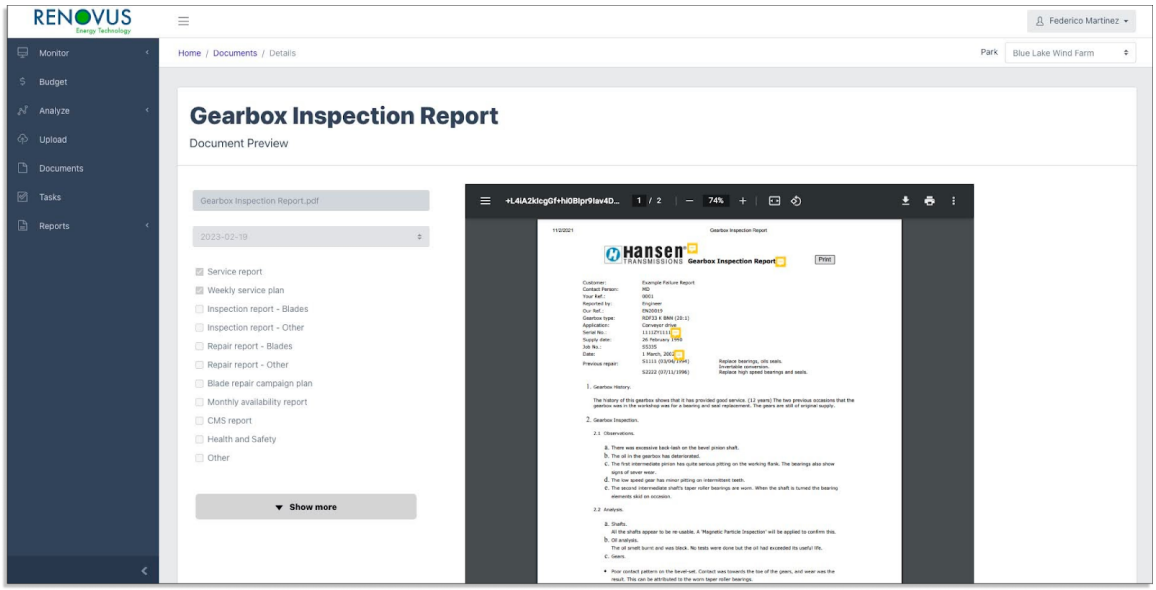


Figura 14: Pantalla detalles de documento 1

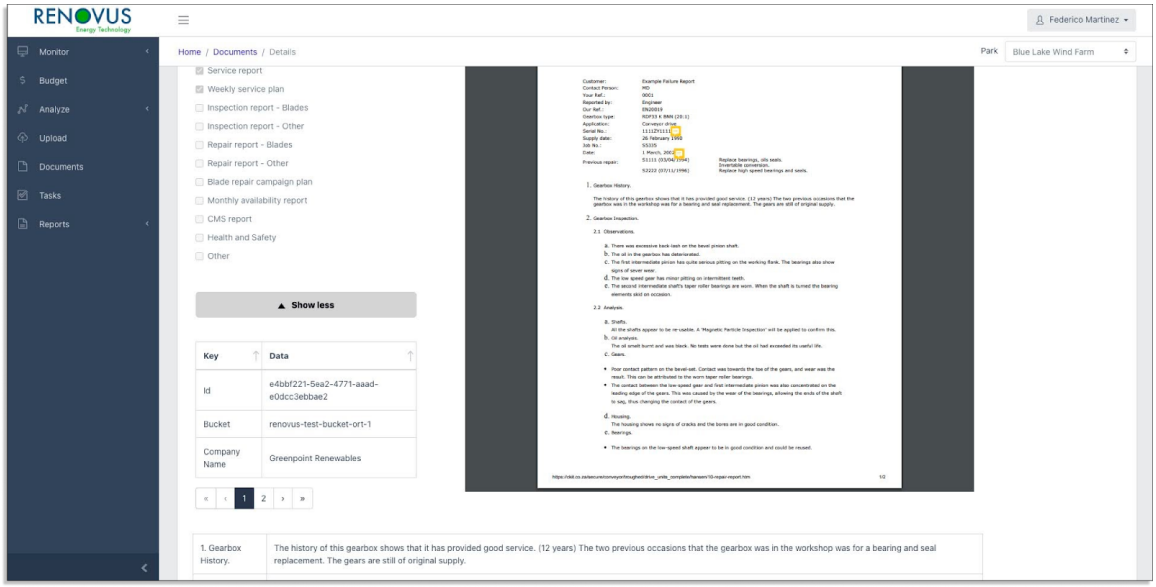


Figura 15: Pantalla detalles de documento 2

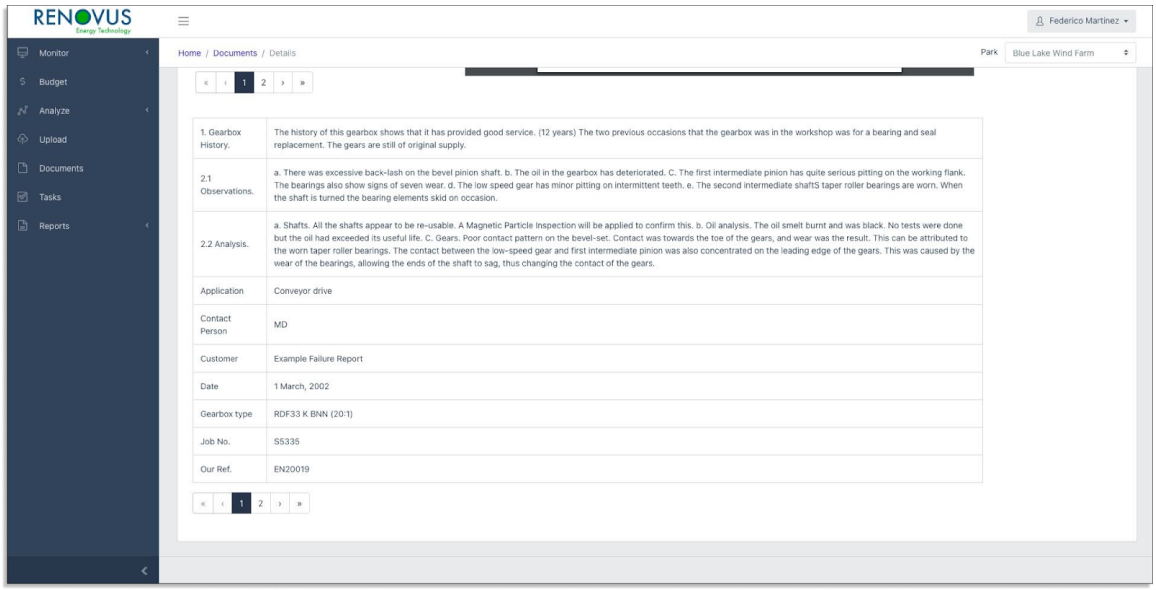


Figura 16: Pantalla detalles de documento 3

Volviendo a la pantalla de búsqueda de documentos, otra funcionalidad que presenta, es la que permite eliminarlos mediante el botón “Delete”. Una vez que se selecciona el documento a borrar, se despliega un mensaje de confirmación para darle al usuario la posibilidad de anular la acción en caso de que haya sido un error o accidente. Si el usuario cancela, no ocurre nada, y si confirma se realiza la eliminación del documento, y con esto también se borran todos los datos del análisis. Sin embargo, no se borran las tareas, ya que estas, dado el último cambio que se realizó en la funcionalidad con el cliente, pueden no tener un documento asociado, por lo que se borra la relación dejando las tareas que tuviera ligadas a él, como aquellas que no tengan documento asociado, o que sí lo tengan, pero con otros documentos (Figura 17).

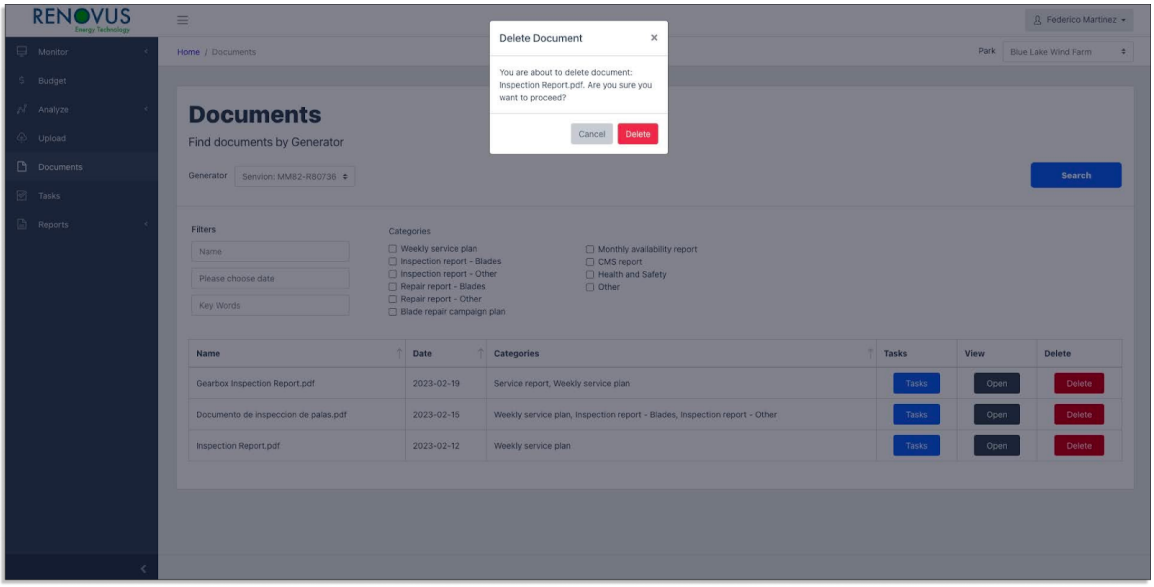


Figura 17: Borrado de documentos

3.2.3 Funcionalidad: Administración de tareas

La administración de tareas es la última funcionalidad destacada que presenta el sistema. Como se desarrolló en la sección anterior, se realizaron modificaciones en su especificación, lo cual tuvo un impacto en otras funcionalidades del sistema, como por ejemplo el borrado de documentos.

En un principio, las tareas solo podían estar asociadas a un documento, el cual podía tener ninguna, una o varias tareas asociadas. Sin embargo, tras los últimos ajustes, se agregó la posibilidad de que una tarea pudiera estar asociada a cero o más documentos, correspondiendo las primeras a las desarrolladas por el generador.

Es importante recalcar esta modificación porque la funcionalidad de administración de tareas cuenta ahora con dos pantallas, lo cual es relevante tomando en cuenta que originalmente contaba sólo con una. Para explicar la primera pantalla de administración de tareas, es necesario retomar la subsección anterior. Como se explicó en ese punto, cada documento listado tiene tres botones (Figura 10), siendo el botón Tasks el que permite acceder a la gestión de tareas de ese documento. Una vez que se presiona este botón, se redirecciona a la pantalla de tareas por

documento (Figura 18) donde se puede visualizar la lista de tareas asociadas al mismo, que puede llegar a estar vacía, en caso de que el documento no tenga tareas asociadas.

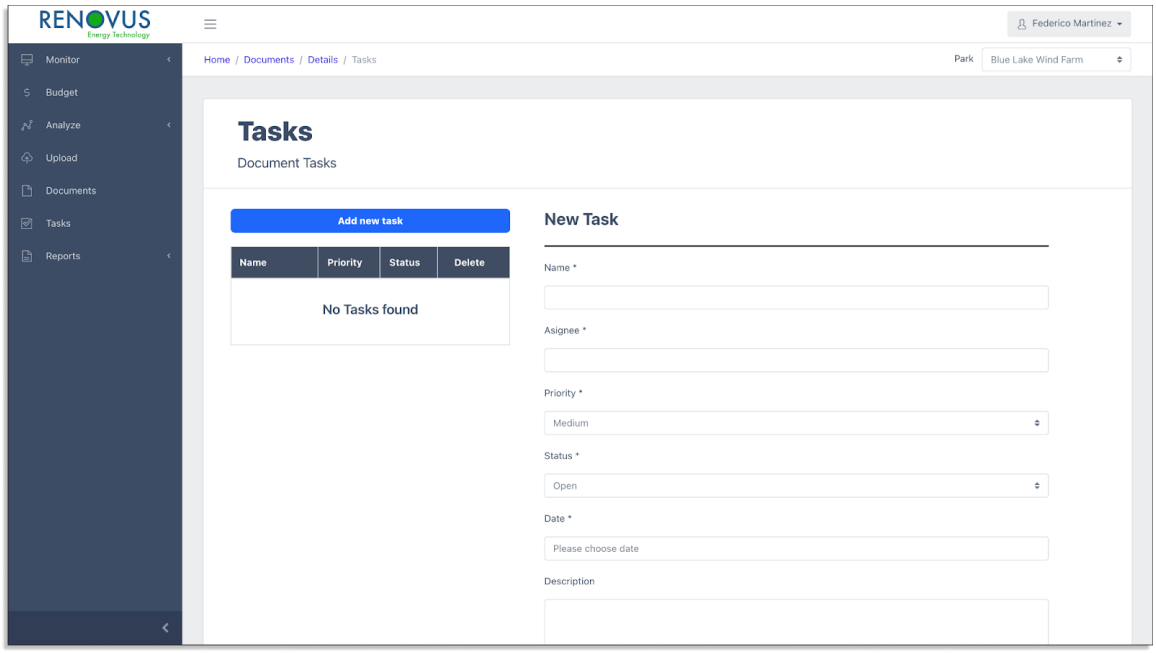


Figura 18: Pantalla de gestión de tareas por documento

También es posible visualizar todas las tareas listadas en una tabla con la opción de ordenarlas por nombre, prioridad o estado. Adicionalmente, hay un botón para abrir el formulario de creación de tareas, queel cual es visible por defecto (Figura 19)

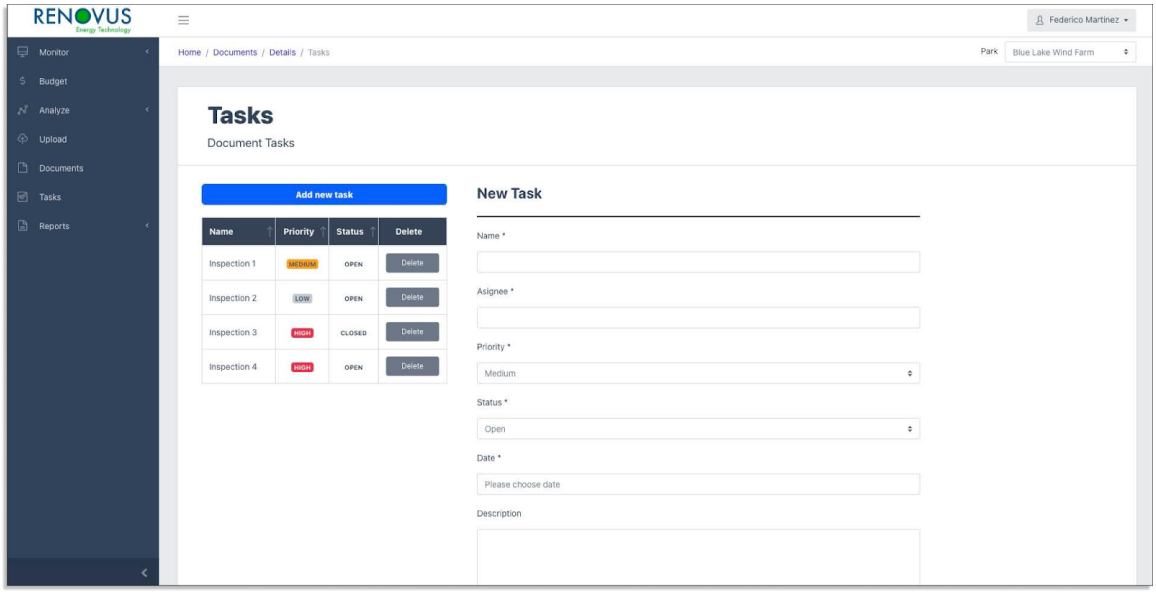


Figura 19: Pantalla de tareas de documento

Al seleccionar una de las tareas listadas, podremos visualizar en el formulario, toda la información de la tarea seleccionada, y en la parte superior derecha del mismo, un botón para editar la tarea que habilitará los campos para su edición. Además, al seleccionar una tarea, podremos ver una lista con todos los documentos que estén asociados a ella (Figuras 20 y 21).

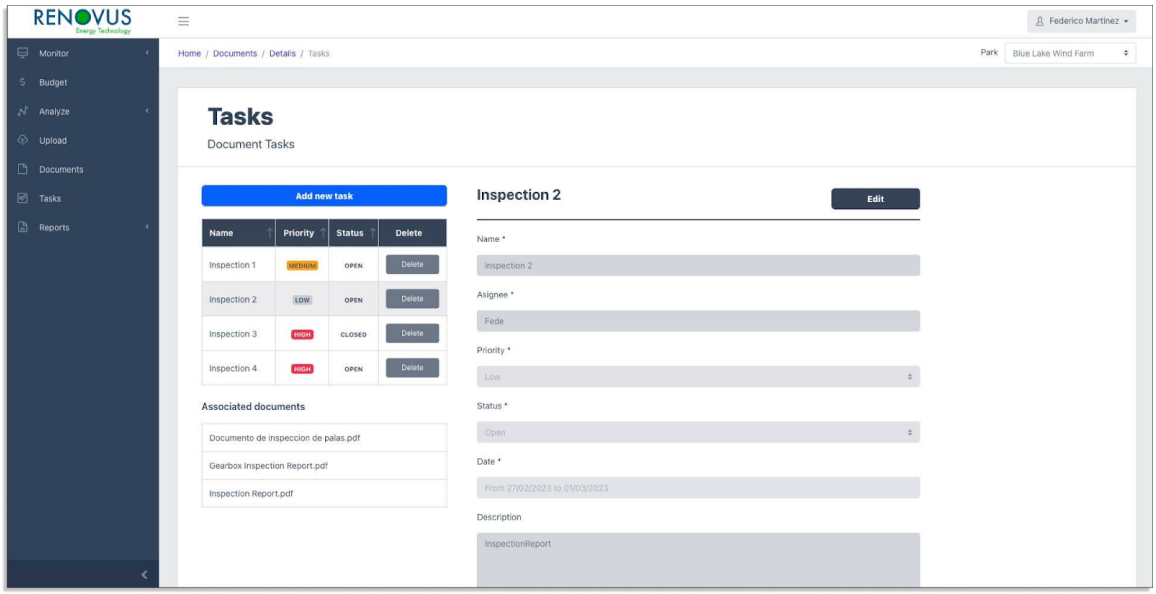


Figura 20: Pantalla de tareas con documentos asociados.

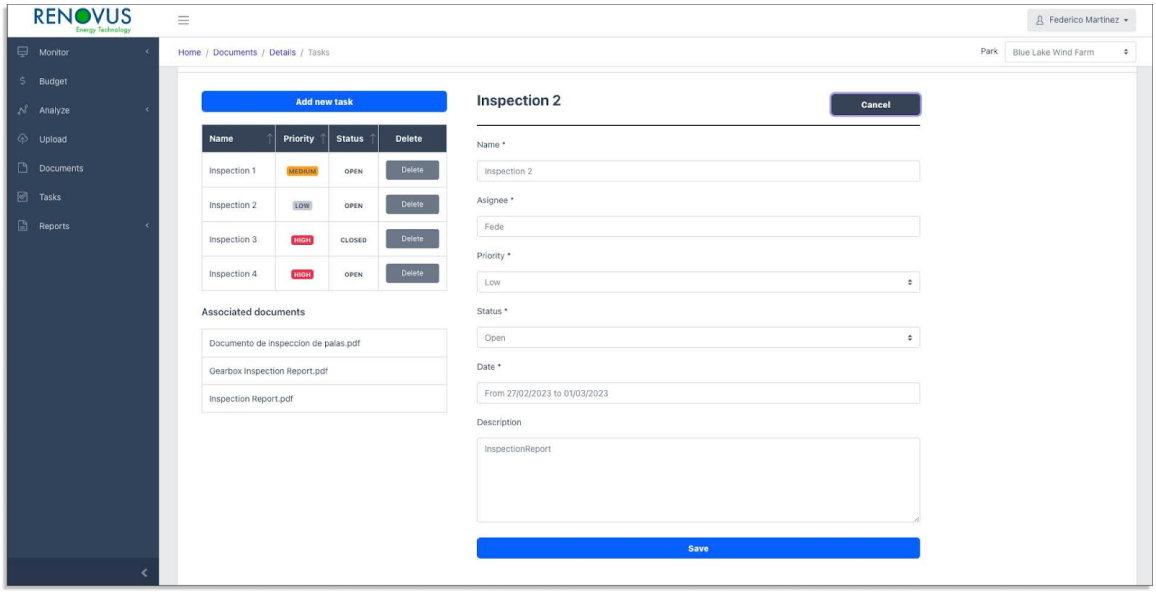


Figura 21: Pantalla de edición tareas de documento

Al crear o editar una tarea, podemos ver que todos los campos, exceptuando la descripción, son requeridos para la creación de la misma, y desde esta pantalla tampoco se puede asociar la tarea a otros documentos, sólo se puede editar la información de la tarea. Finalmente, tenemos el borrado de las tareas, que al igual que para los documentos, al cliquear sobre el botón “Delete” se despliega un mensaje pidiendo confirmación, ya que es una acción que no se puede deshacer (Figura 22).

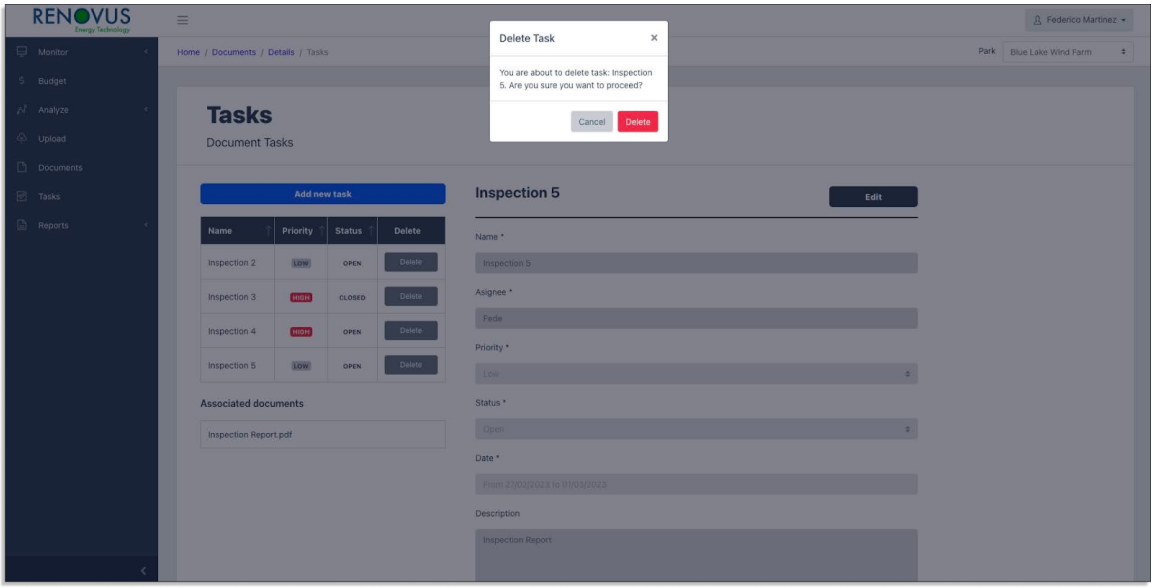


Figura 22: Borrado de tarea

Finalmente, tenemos la otra pantalla de tareas, en este caso, por generador, a la que se accede desde la barra lateral izquierda. Haciendo clic sobre la sección “Tasks”, nos encontraremos con una pantalla bastante similar a la presentada anteriormente, pero con el agregado de tener en la parte superior izquierda, la posibilidad de elegir el generador para el cual queremos ver las tareas (Figuras 23, 24 y 25).

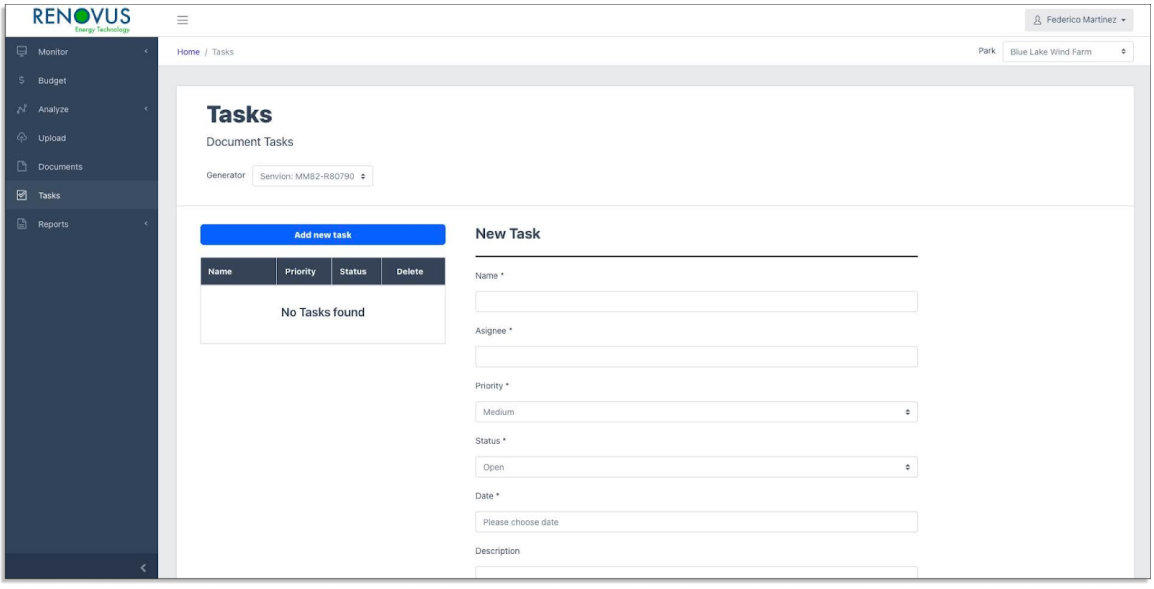


Figura 23: Pantalla de gestión de tareas por generador

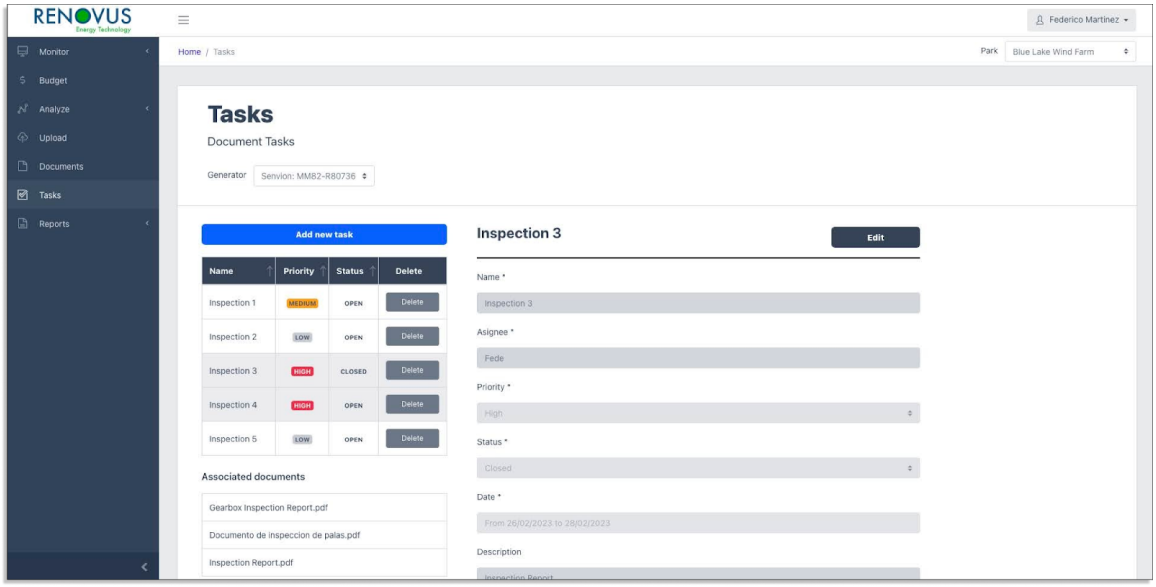


Figura 24: Pantalla de gestión de tareas por generador con tareas

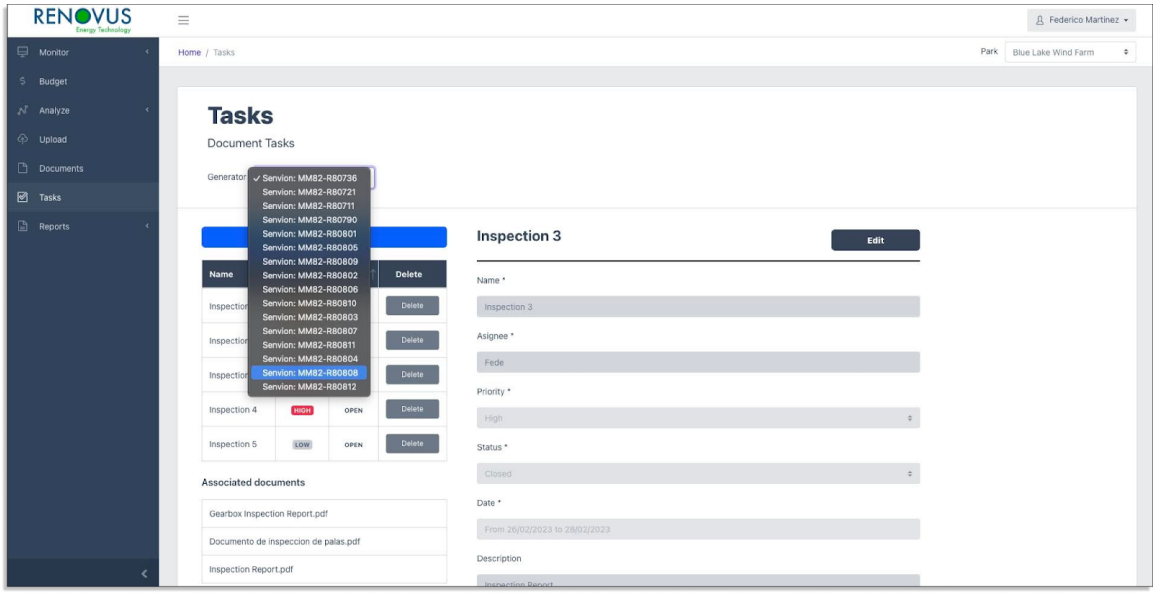


Figura 25: Pantalla de gestión de tareas por generador, selección de generador

En general, el comportamiento de esta pantalla no difiere del mencionado anteriormente, la diferencia aquí se encuentra al editar o crear tareas, y es que no solo hay un botón para crear la tarea o guardar los cambios, sino que además de este aparece un nuevo botón a la derecha que es para la selección de documentos a los cuales se quiere asociar la tarea.

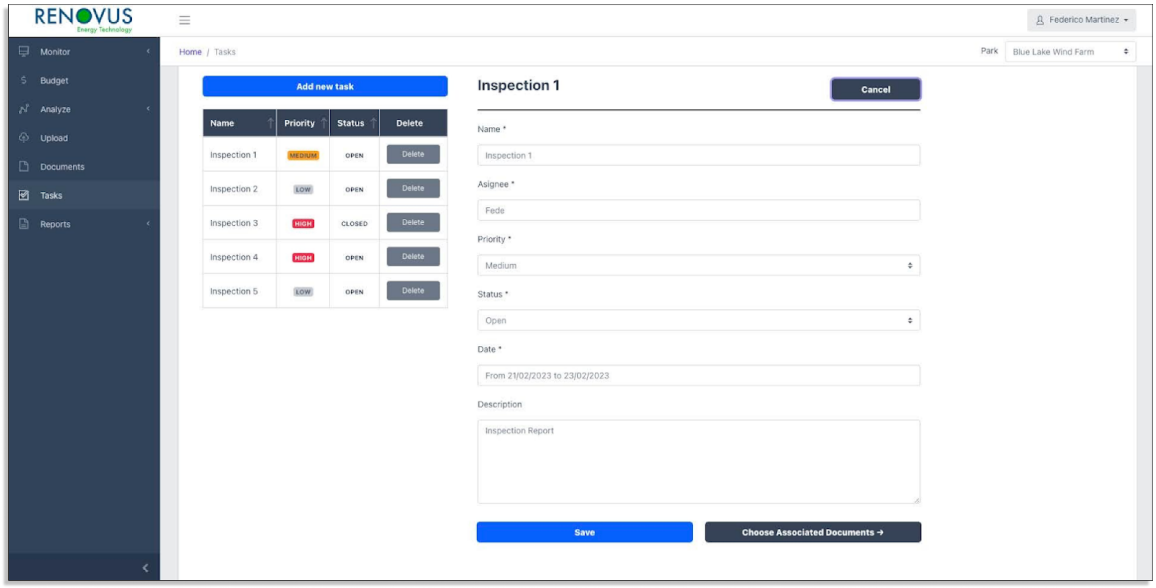


Figura 26: Pantalla de gestión de tareas por generador, edición

Luego de completar todos los campos, si guardamos la tarea, la misma se crea sin documentos asociados, ya que ninguno se seleccionó, pero en caso de querer vincular la tarea a los documentos, se debe hacer clic en el botón “Choose Associated Documents”, lo cual va a desplegar en el área donde se muestra el formulario, una tabla con todos los documentos que se hayan cargado para ese generador con nombre y fecha de carga, además de un campo para poder filtrar por nombre los documentos, que cuenta con un botón de búsqueda al lado, y un botón adicional para limpiar los filtros que se muestra solo cuando el usuario utiliza esta función.

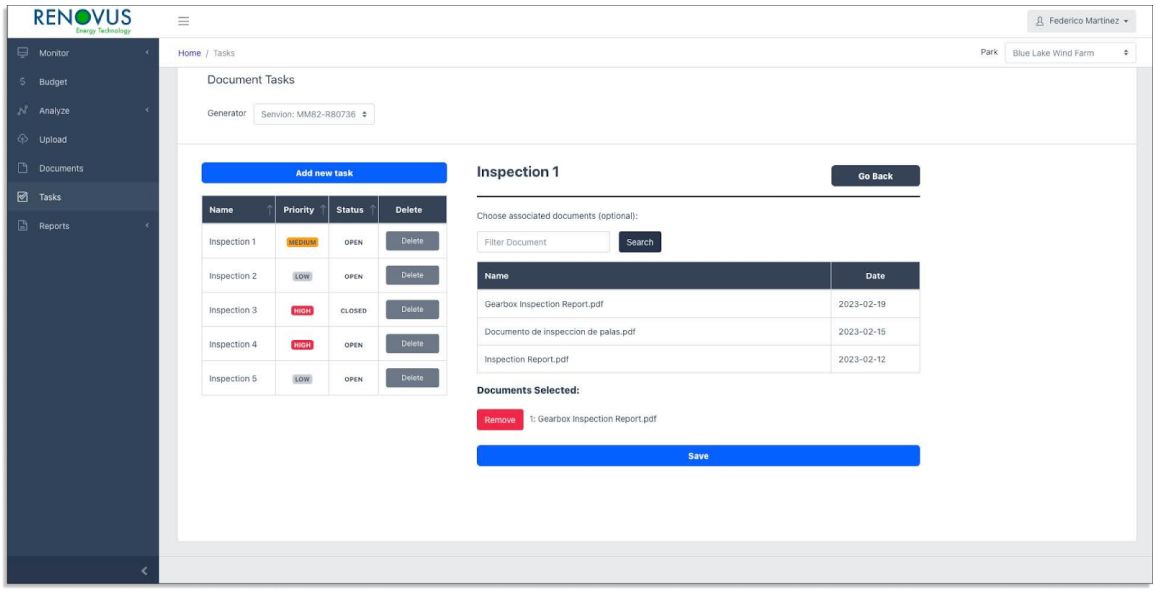


Figura 27: Pantalla de gestión de tareas por generador, edición, selección de documento

En esta pantalla tendremos además un botón para volver atrás, en caso de ser necesario, un botón para guardar los cambios, y también se irán listando debajo de la tabla de documentos aquellos documentos que el usuario haya seleccionado para vincular con la tarea, y, a la derecha de estos, la opción de removerlos de dicha selección. En caso de estar editando una tarea, la pantalla se muestra con los documentos cargados para ese generador, y debajo de esta tabla aquellos documentos que hayan sido asociados a la tarea.

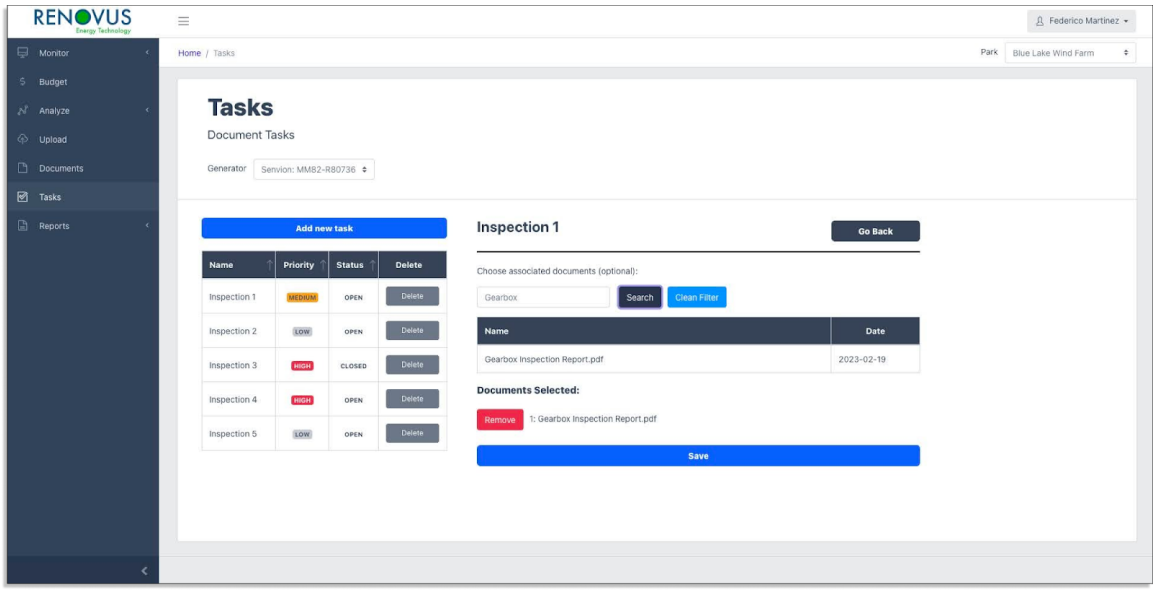


Figura 28: Pantalla de gestión de tareas por generador, edición, filtrado de documentos

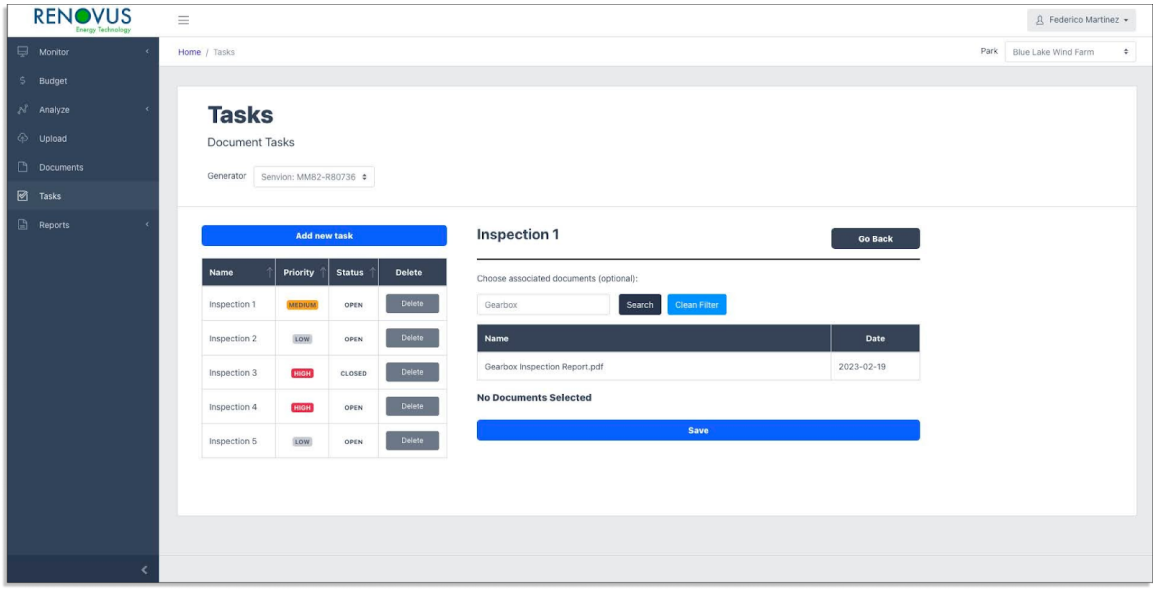


Figura 29: Pantalla de gestión de tareas por generador, edición, sin documentos asociados

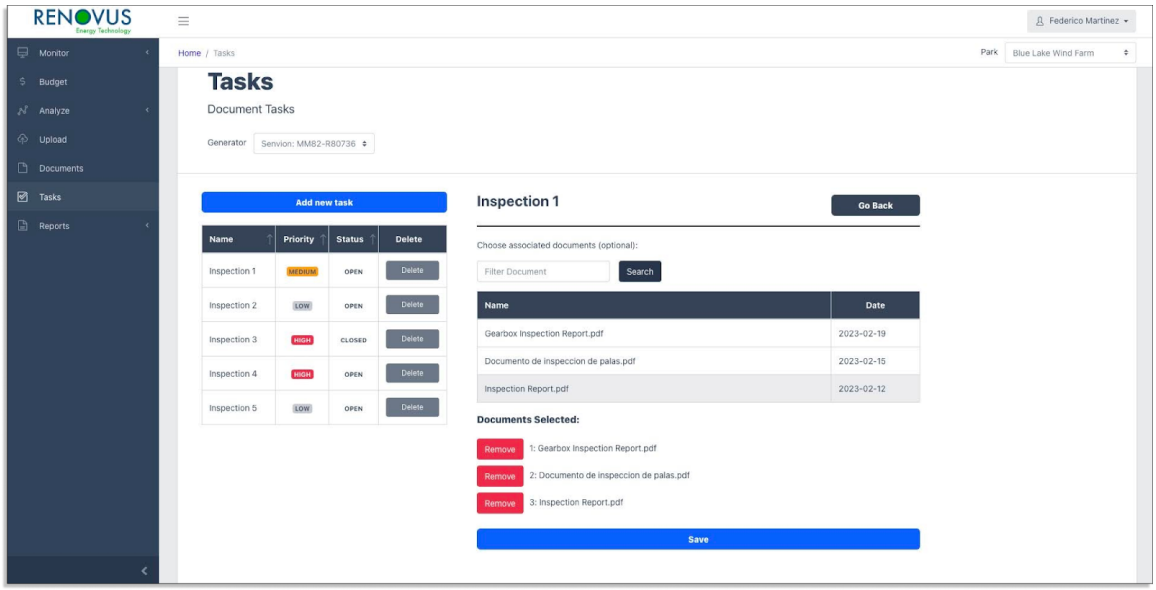


Figura 30: Pantalla de gestión de tareas por generador, edición, selección de varios documentos

4. Desarrollo y proceso

4.1 Características del proyecto

A continuación, se exponen las características más destacadas del proyecto, cuyo análisis fue crucial tanto para seleccionar y definir la metodología de trabajo como para organizar el equipo de manera eficiente y eficaz.

Estas particularidades engloban diversos aspectos del negocio y del cliente, así como las limitaciones tecnológicas a considerar, entre otros aspectos relevantes, para el éxito del proyecto.

4.1.1 Negocio existente y expectativas del proyecto

Al inicio del proyecto, Renovus ya poseía una solución tecnológica que aplicaba ciencia de datos e inteligencia artificial en la industria de energías renovables, con el objetivo de mejorar la rentabilidad de los parques eólicos de sus clientes. El sistema ya disponía de una página *web* y una plataforma *backend* que se enfocaban en el análisis de aerogeneradores, evaluación de su eficiencia y la generación de los correspondientes informes mensuales.

Dadas las características del desafío planteado, éste requería que el equipo recibiera en primer lugar una adecuada capacitación y llevara a cabo un análisis exhaustivo de los módulos existentes para poder realizar las modificaciones necesarias y cumplir con los requisitos del cliente. Además, se necesitó una coordinación constante con el equipo de soporte para trabajar en paralelo sin generar conflictos y para planificar las entregas conjuntas, asegurando que los cambios del equipo no fueran afectados por los de Renovus durante el período de desarrollo.

Al concluir el proyecto, se tenía previsto tener disponible la solución basada en la nube, totalmente integrada con la plataforma *web* de Renovus a modo de un MVP, lo que permitiría a futuro que las empresas clientes de Renovus pudieran utilizar el sistema desarrollado sin inconvenientes.

4.1.2 Disposición el cliente

El cliente siempre se mostró accesible y dispuesto a colaborar en el proceso de construcción del sistema, poniendo en todo momento a disposición a sus expertos en el área de negocio para el mejor desarrollo del mismo y para responder a cualquier duda que el equipo tuviese. Desde el primer hasta el último día, constantemente se mostraron dispuestos a ayudar y aportar su experiencia en el desarrollo de las nuevas funcionalidades.

Además, es importante destacar que el cliente brindó un valioso apoyo en la planificación del proyecto al ofrecer retroalimentación constante y prestar gran atención a los comentarios y las tareas que ellos debían realizar, sugeridas por el equipo. Este nivel de colaboración contribuyó en gran medida al éxito del proyecto, ya que permitió una comunicación clara y efectiva, lo que a su vez, posibilitó al equipo poder cumplir con los objetivos establecidos en el plazo previsto.

4.1.3 Investigación y capacitación

La investigación y capacitación representaron una parte significativa del tiempo dedicado al proyecto. Estas actividades resultaron cruciales, no solo porque el equipo se enfrentaba por primera vez a tecnologías como inteligencia artificial, ciencias de datos y tecnologías como computación en la nube, sino también porque se requería adaptarse al sistema existente de Renovus.

La etapa de investigación fue, sin duda, la más importante, ya que estableció el rumbo del proyecto. El cliente había planteado un resultado esperado y fue responsabilidad del equipo analizar y determinar la solución más adecuada para desarrollarla. Durante esta etapa, se recopiló información de diversas fuentes, incluyendo artículos, libros y cursos en línea. El equipo mantuvo una comunicación constante con el cliente y se aseguró de favorecer sus intereses.

Una vez finalizada la investigación, comenzó la fase de capacitación. Al igual que en la etapa anterior, esta actividad demandó alrededor de dos semanas y media, debido a que el equipo no estaba familiarizado con las tecnologías requeridas. Además, se tuvo que aprender cómo se había desarrollado el producto actual de Renovus, ya que el objetivo era adaptarse al sistema existente

para garantizar que el cliente pudiese a futuro mantener el producto sin dificultades ni problemas.

Es importante destacar que, aunque no todo lo investigado y aprendido en esta fase formó parte del resultado final del proyecto, estos conocimientos fueron esenciales en la consecución de los objetivos y el logro de un resultado óptimo. Una buena base teórica permite un mejor desempeño en la práctica, incluso con herramientas y tecnologías que no fueron parte del resultado final. Tanto la investigación como la capacitación fueron etapas fundamentales en el proceso de desarrollo y la inversión de tiempo y recursos en ellas, fue muy valioso para el resultado final.

4.2 Características del equipo

4.2.1 Tamaño del equipo

Como fue mencionado en la sección 1.1, el equipo se encuentra conformado por tres miembros, cada uno de ellos dedicado a diferentes áreas en el desarrollo de *software*. El tamaño del equipo se considera adecuado dadas las circunstancias del proyecto, ya que permitió la asignación de roles específicos y la división de tareas. Además, posibilitó la participación activa de todos los integrantes en todas las etapas del proyecto, y la distribución equitativa del esfuerzo requerido.

4.2.2 Relación del equipo

La relación entre los miembros del equipo fue altamente efectiva gracias a su previa experiencia de trabajo conjunto. Durante su formación académica en la carrera Ingeniería en Sistemas, el equipo desarrolló de manera exitosa diversos proyectos para varias materias de la Universidad ORT, lo que les permitió conocer las habilidades y fortalezas de cada uno así como dividir responsabilidades de manera inmejorable.

Asimismo, el equipo mantiene una sólida relación por fuera de la académica, con una confianza mutua en las habilidades y capacidades de cada miembro para ejecutar sus responsabilidades de manera efectiva.

4.3 Proceso y ciclo de vida

4.3.1 Ciclo de vida

El ciclo de vida de un proyecto de *software* es crucial para su éxito, ya que determina cómo se planifica, se desarrolla y se entrega el producto. En este caso, el equipo consideró dos opciones: el ciclo de vida evolutivo y el incremental iterativo [6].

Después de analizar las posibilidades, el equipo se terminó decantando por el ciclo de vida incremental iterativo. Este modelo implica dividir el proyecto en iteraciones que se asemejan a pequeñas partes que componen el todo. Cada iteración tiene su propio proceso de trabajo que se repite, lo que permite que el cliente obtenga los beneficios del proyecto de forma incremental.

En cada iteración, el equipo trabaja para evolucionar el producto y hacer una entrega incremental basada en los resultados completados en las iteraciones anteriores. Los nuevos objetivos y requisitos se añaden en cada iteración, lo que permite al equipo adaptarse a las necesidades del cliente y gestionar sus expectativas. Además, el proceso permite obtener resultados importantes de forma temprana y hacer una mejor gestión general del proyecto.

Los factores clave que influyeron en la decisión de este ciclo por parte del equipo, fueron por un lado, la posibilidad de contar con los requerimientos claros del cliente pero posiblemente sujetos a cambios en los detalles, así como la ausencia de grandes variaciones esperadas y por otro, la oportunidad de entregar en forma temprana, una arquitectura que posiblemente no fuese a sufrir variaciones significativas.

4.3.2 Metodología de referencia

Para respetar el ciclo de vida seleccionado, el equipo considera el marco de trabajo ágil Scrum.

A la hora de trabajar con una metodología de referencia cabe destacar la diferencia del concepto de “metodología ágil” con la de un marco de referencia (ágil) puntual como Scrum. El primer concepto refiere a un conjunto de principios mencionados en el manifiesto ágil [7] y el segundo es la aplicación de esos principios llevándolos a la práctica como una forma de estructurar el trabajo.

Basándonos en lo mencionado en la sección anterior, se opta por la adopción de una metodología acorde a los principios del manifiesto, que cumpla el ciclo de vida seleccionado a modo de obtener sus beneficios, y por este motivo, se elige tomar de referencia al marco de trabajo Scrum [8].

Este marco de trabajo ofrece una estructura flexible que permite al equipo adaptarse a los cambios de requisitos y expectativas del cliente a lo largo del proyecto, asegurando que se cumplan los objetivos establecidos en cada iteración, así como garantizando la satisfacción del cliente a lo largo del proyecto, algo alineado con los objetivos establecidos por el equipo estudiante al comienzo del trabajo.

Otro aspecto importante a destacar es que Scrum realiza el manejo de los requerimientos con un formato de "historias de usuario". Estas son descripciones breves y sencillas de los requisitos del cliente, enfocadas en el valor que se espera obtener de cada una de ellas. De esta manera, el equipo se enfoca en requerimientos valiosos al cliente de forma incremental y continua a lo largo del proyecto.

La aplicación del marco Scrum también promueve la colaboración entre los miembros del equipo y la comunicación frecuente con el cliente, lo que permite una mayor comprensión de los requisitos y expectativas del cliente, así como una mejor gestión del proyecto en general. Además, Scrum facilita la identificación temprana de problemas y riesgos, lo que permite al equipo abordarlos de forma rápida y efectiva.

4.3.3 Descripción general del proceso

Una vez seleccionado el ciclo de vida y la metodología de referencia, se comenzó a realizar la adaptación de la misma para sacar el máximo provecho de sus beneficios y ajustarla a la realidad de trabajo del equipo.

El grupo de estudiantes, debido a sus compromisos externos, trabajos y realidades académicas, enfrentaba dificultades para cumplir estrictamente con las indicaciones del marco de trabajo ágil Scrum. Sin embargo, el marco de trabajo ofrece la flexibilidad necesaria para adaptarse a diferentes situaciones, incluyendo la del equipo.

A grandes rasgos el proyecto fue dividido en grandes iteraciones llamadas *releases* y, a su vez, cada uno de estos estuvo subdividido en iteraciones más pequeñas con el nombre de Sprints. Durante esta iteración, el equipo trabajó en estrecha colaboración con el representante del cliente, el Product Owner, para lograr los objetivos acordados y al final del Sprint, el equipo fue capaz de generar un incremento de trabajo potencialmente entregable, sobre el que se continuó iterando.

Inicialmente, se consideró la posibilidad de utilizar una metodología híbrida llamada Scrumban [9], la cual combina Scrum con el marco de trabajo Kanban [10]. No obstante, tras una votación, el equipo decidió no implementar esta propuesta, con el fin de evitar retrasos en el inicio del proyecto y optó por mantenerse dentro de la metodología Scrum clásica, con la que ya tenían experiencia previa y también, que se utilizaría la herramienta JIRA [11] para la gestión de la misma.

En las siguientes secciones se detalla el resultado de la metodología adoptada por el equipo, adentrándose en los roles, artefactos y ceremonias resultantes, así como el flujo final con el que trabajó el equipo.

Roles

El marco de trabajo Scrum requiere la definición de diferentes roles, cada uno con sus propias responsabilidades a lo largo del proyecto.

En primer lugar, tenemos al **Product Owner** quién es el representante del cliente y se asegura que la solución final represente adecuadamente los intereses y necesidades de su equipo. Este rol contribuye con la gestión de requerimientos para la realización de las historias de usuario así como la validación de las mismas y determina su distribución en los Sprints. Es el responsable de la priorización de historias y es el principal referente a la hora de la realización de la retroalimentación (*feedback*) para el equipo estudiante. Este rol fue ocupado por un integrante del equipo cliente.

Luego tenemos al equipo de **desarrolladores** quienes son los responsables de entregar el producto. Este rol abarca tareas que cubren desde el diseño, análisis, desarrollo, pruebas y generación de documentación, entre otras. Es importante destacar que en este equipo no se

asignaron roles específicos, lo que significó que cualquier desarrollador podría asumir tareas relacionadas con cualquier área, ya sea *frontend*, *backend*, arquitectura o cualquier otra.

Finalmente, Scrum designa un **Scrum Master**, pero, en la adaptación del equipo, este rol fue obviado. Esto se debe a que las responsabilidades del rol se centran en el alto entendimiento y cumplimiento de Scrum. El marco no rechaza la idea que un miembro del equipo sea Scrum Master, pero, tras considerarlo, el equipo determina que ninguno de sus integrantes poseía ni el conocimiento ni las certificaciones (por ejemplo, PSM [12]) para realizar esta tarea. Por este motivo, dicho rol fue eliminado de la adaptación del equipo, dejando a los desarrolladores la responsabilidad de autoevaluarse y buscar mejorar continuamente en la aplicación del proyecto con una mirada crítica y objetiva a sus acciones realizadas durante el proceso.

Ceremonias

A lo largo de las iteraciones, Scrum reconoce algunas ceremonias que toman lugar en un momento específico dentro del ciclo, en la que participan ciertos roles a modo de obtener una salida o un artefacto actualizado.

Scrum Guide [13] reconoce oficialmente cinco ceremonias dentro de Scrum: Sprint, Sprint Planning, Daily Scrum, Sprint Review y Sprint Retrospective. El equipo basó su metodología casi en su totalidad, de acuerdo a esta guía, a excepción del Daily Scrum y agregó un conjunto de ceremonias adicionales.

El **Sprint** es el corazón del marco de trabajo Scrum, aquí las ideas se convierten en valor para el cliente. Son eventos de duración fija. En el caso del equipo estudiante fue de dos semanas, buscando llevar adelante los objetivos planteados en la Sprint Planning Meeting y lograr la entrega de un incremento de trabajo, "potencialmente entregable".

La **Sprint Planning Meeting** es la segunda ceremonia a considerar que toma lugar al principio de cada Sprint y se conforma con los miembros del equipo y el Product Owner. Su objetivo es obtener el alcance de un Sprint mediante la selección de historias para el Sprint Backlog.

Durante esta ceremonia se pretende responder a las siguientes preguntas como guía para buenos resultados:

- ¿Qué hace a este Sprint valioso?
- ¿Qué podemos hacer en este Sprint?
- ¿Cómo podemos hacer el trabajo seleccionado?

El **Weekly Scrum** actuó como reemplazo para el Daily Scrum, definido a realizar una vez por semana con los miembros del equipo. Como el Product Owner no trabaja activamente como miembro, no es un requisito su asistencia a estas reuniones, pero se le informa de la misma y su participación puede ser aprovechada para eliminar alguna traba en el proyecto. En esta ceremonia se plantean los trabajos a realizar durante la semana y se busca guiar al equipo basándose en tres preguntas:

- ¿Qué hice esta semana?
- ¿Qué haré la próxima semana?
- ¿Qué problemas relacionados con el proyecto tuve durante esta semana?

La **Sync Meeting** fue definida como una reunión a ser realizada cada dos semanas, en la que participan los desarrolladores y el Product Owner así como los principales referentes del equipo cliente (*Technical Lead* y *CEO*). El objetivo de esta ceremonia es poner al corriente del estado del proyecto al cliente y se aprovecha para despejar dudas de cualquier índole, ya sean de carácter técnico o para realizar validaciones puntuales, así como obtener retroalimentación de nuevas ideas o propuestas.

Al finalizar cada Sprint se ejecutan dos ceremonias. En primer lugar, la **Sprint Review** que cuenta con la presencia del Product Owner y los desarrolladores. En esta se verifica que el desarrollo del Sprint haya cumplido con los criterios de aceptación definidos en las historias de usuario. Luego tenemos la **Sprint Retrospective** donde participan los desarrolladores y dejan sus impresiones sobre el Sprint realizado. El propósito de la retrospectiva es realizar una mejora continua del proceso.

Análogamente, tras completar cada *release* se llevan a cabo la **Release Review** y la **Release Retrospective**, con los mismos participantes y objetivos que en sus respectivas contrapartes de Sprint, pero diferenciándose en que la iteración a evaluar tiene un enfoque macro del proyecto.

Artefactos

Un artefacto se refiere a una salida o resultado que se obtiene a partir de otros procesos. La adaptación del equipo posee los artefactos que se detallan a continuación.

En primer lugar, encontramos el **Product Backlog** donde se lista todo lo necesario en formato de historias de usuario (de las que se hablará en detalle en la sección 4.5.4) para realizar el proyecto. Este artefacto evoluciona constantemente y su responsable es el Product Owner quien ha de priorizarlo acorde a sus necesidades.

Vinculado a este último, encontramos el artefacto llamado **Burndown Chart** que se visualiza con una gráfica que mide el esfuerzo restante para el desarrollo en una unidad llamada Puntos de Historia. Este artefacto permite una vista rápida de Sprint a Sprint del progreso real del equipo contra el esperado. Lo deseable para esta gráfica, es que sea descendente hasta llegar al eje horizontal, marcando el fin del desarrollo. Si se agregaran nuevos requerimientos, la recta tendrá una pendiente ascendente en determinados segmentos y de modificarse alguno podría ser de cero en algún tramo.

Luego se define el **Release Backlog**, un listado de aquellas Historias de Usuario que pertenecen a un *release* particular. Son seleccionadas del Product Backlog por el equipo según la priorización previamente realizada por el Product Owner para lograr un *release* que aporte valor lo antes posible al cliente.

En tercer lugar, tenemos el **Sprint Backlog** que al igual que los artefactos anteriores se trata de una lista de Historias de Usuario agrupadas para una iteración, en este caso un Sprint. Se hace el seguimiento de estas en la ceremonia correspondiente a la modificación del marco por parte del equipo, el Weekly Scrum.

Relacionado a este último artefacto, encontramos el llamado **Definition of Done** que nos marca cuando una tarea o historia está completa, actividad que se verifica en las Sprint Review y se

asegura que todos los miembros del equipo tengan una comprensión clara y común de lo que significa completar una tarea y qué nivel de calidad se espera de su trabajo.

Al final de cada Sprint debe existir un **Producto Incrementado** que buscará ser usable, pero no necesariamente será un MVP. Esto quita la presión del equipo para no apresurarse a desarrollar, algo que en su afán de ser usable a toda costa, pueda contener imperfecciones que dañen un sistema crítico del cliente.

Finalmente, tenemos aquellos artefactos que buscan la mejora del equipo sobre el proceso: la **Encuesta de satisfacción del cliente** y la **Encuesta de Clima de trabajo en equipo (Team Pulse)**, ubicadas en el Anexo 10.7. La primera busca ser una evaluación por escrito de parte del cliente sobre el desempeño del equipo, ya sea en interacciones, en calidad de desarrollo y cuestiones similares. La segunda se refiere a una encuesta desarrollada con base en dos documentos presentes en el sitio Aulas de la Universidad ORT [14], titulados “Encuesta para medir si las reuniones son efectivas” y “Encuesta sobre clima interno en el equipo”. Esta encuesta/artefacto ha de ser completada al final de cada iteración por los miembros del equipo a fin de evaluar sus relaciones interpersonales, su trabajo, relación con el cliente y elaborar una conclusión sobre la cual se puedan realizar ajustes con el objetivo de mejorar el proceso.

Flujo

El resultado de la definición de los roles, artefactos y ceremonias planteadas resulta en un flujo muy similar al propuesto por Scrum, con algunas ligeras desviaciones. La siguiente figura (Figura 31) se trata del flujo presente en la Scrum Guide con la modificación de la ceremonia Daily Scrum a Weekly Scrum adaptada al trabajo del equipo estudiante.

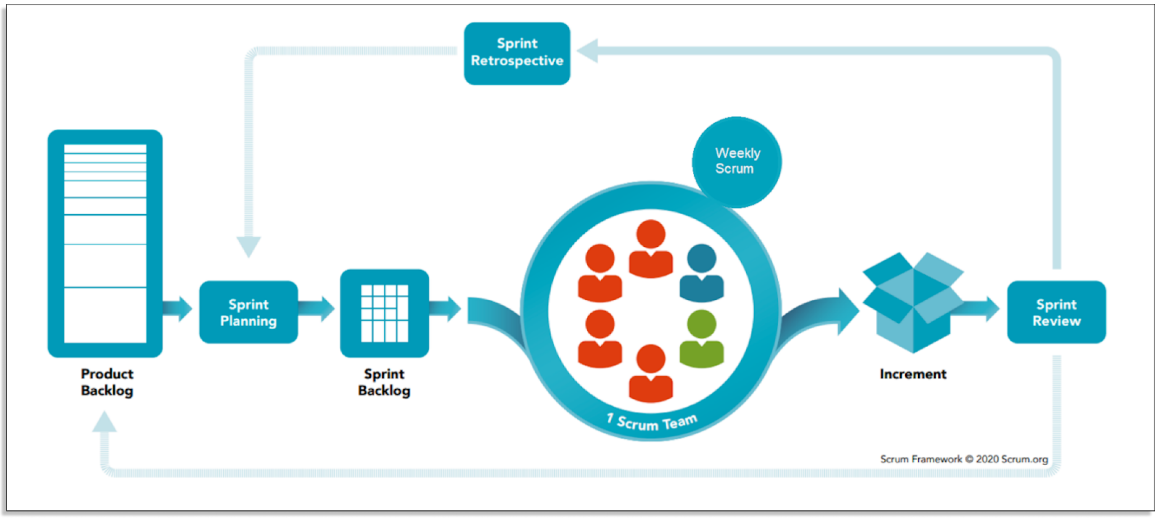


Figura 31: Flujo de trabajo

Como ausencia significativa en la imagen se destaca que no está presente la denominada Sync Meeting. Esta ceremonia podría visualizarse junto a la Sprint Review al ocurrir usualmente el mismo día.

4.4 Reflexiones del proceso

Gracias a la metodología ágil, el equipo pudo adaptarse a los cambios que se presentaron en el transcurso del proyecto, lo que se puede observar en la comparación del alcance inicial y final del proyecto en el Capítulo 3. Además, el modelo ágil permitió simplificar el trabajo en un entorno desconocido como lo era AWS, facilitando la implementación de una solución efectiva.

El cliente también se benefició del modelo ágil al poder ver el producto construirse y dar valiosos comentarios sobre sus intereses, lo que permitió al equipo mejorar y ajustar el producto de acuerdo a sus necesidades.

La utilización de JIRA [11] fue fundamental en el proceso, aunque al principio el equipo tuvo algunos problemas para administrarla cómodamente, el uso de esta herramienta permitió a los estudiantes llevar un seguimiento detallado del proyecto y mantener una buena comunicación con el cliente que podía revisar las historias generadas y su estado en todo momento.

En algunos momentos del proyecto, el equipo experimentó desafíos al presentar contribuciones de gran valor al cliente, especialmente durante la fase inicial en la que predominaba un enfoque investigativo. Se observó una mejora en las retroalimentaciones iniciales a medida que avanzaba el proyecto, lo que indica que el equipo logró superar los desafíos iniciales y logró entregar contribuciones más significativas al cliente.

Como conclusión, la adaptación de Scrum propuesta por el equipo de estudiantes fue muy acertada, adaptando un modelo que les permitió mejorar su capacidad de entregar valor al cliente y su trabajo en equipo.

4.5 Ingeniería de requerimientos

4.5.1 Relevamiento inicial

Desde el primer contacto con el cliente, el equipo se propuso la elaboración de una lista de requerimientos deseados para la aplicación. A lo largo del proceso, el documento experimentó ajustes menores, manteniendo siempre los requerimientos iniciales establecidos por el mismo.

Esta metodología permitió al equipo contar con una guía informal para investigar los diferentes requerimientos deseados, así como también agilizar el proceso de investigación y colaborar con el cliente para poder formalizar los requerimientos.

4.5.2 Uso de Design Thinking

Para lograr desarrollar una herramienta útil para los usuarios finales del proyecto, se decidió aplicar la metodología Design Thinking [15]. Esta técnica innovadora permitió al equipo generar ideas y encontrar soluciones prácticas y efectivas a las necesidades reales de los usuarios. Aunque el cliente había identificado claramente sus requerimientos, era necesario contar con un mecanismo que asegurara una correcta comprensión por parte del equipo, sin margen de error en la interpretación. Por esta razón, se hizo uso de la metodología antes mencionada.

El equipo optó por utilizar esta técnica, pero, no siguiéndola al pie de la letra, sino que utilizando sus cinco etapas como base y adaptándola a las necesidades específicas del proyecto. Por tanto, se diseñó un proceso que constaba de las cinco etapas fundamentales del Design Thinking:

empatizar, definir, idear, prototipar y testear en usuarios. Se tomó especial cuidado en la fase de empatía y de prototipación ya que estas permitieron al equipo entender las necesidades reales de los clientes y asegurarse que la solución propuesta fuese adecuada para ellos. El resultado de este proceso se puede visualizar en el infograma del Anexo 10.3, donde se presenta una descripción detallada de las etapas, incluyendo el proceso de prototipación en detalle.

La metodología Design Thinking resultó muy beneficiosa para el equipo, no solo por ofrecer un proceso claro y conciso, sino porque también permitió al cliente expresarse de manera más efectiva y establecer una visión compartida de las necesidades y objetivos clave del proyecto.

4.5.3 Especificación de requerimientos

Una vez finalizado el proceso previo, se logró definir una lista formal de requerimientos que se encuentra en el Anexo 10.4. Esta lista tiene en cuenta los objetivos iniciales planteados y la información recopilada durante las reuniones con el cliente. De esta manera, se llegó a un acuerdo vía correo electrónico sobre la lista planteada, logrando así una lista de requerimientos formal, detallada y validada.

Con tal de facilitar la definición de una interfaz gráfica y su posterior construcción, se construyeron varios prototipos utilizando la herramienta Figma [16] que se presentaron al cliente. Estos prototipos se utilizaron como referencia durante la implementación de la interfaz gráfica de la aplicación y fueron un componente vital del proceso presentado en la sección anterior.

4.5.4 Formalización en Historias de Usuario

Después de la decisión de utilizar una metodología basada en Scrum como marco de trabajo, el equipo se enfocó en transformar la lista de requerimientos acordados con el cliente en historias de usuario, en línea con los requisitos de este *framework*.

Las historias de usuario son una técnica utilizada en el desarrollo de *software* para describir una funcionalidad o característica del sistema desde la perspectiva del usuario. Las historias describen quién es el usuario, qué quiere lograr, por qué, y cómo el sistema puede ayudar a

lograrlo. Esta forma de escritura de los requerimientos busca enfocarse en el valor que se le ofrece al cliente.

El equipo definió que para su metodología, cada historia está compuesta obligatoriamente de un **nombre**, un **cuerpo** que describe el requerimiento con el formato "como [actor], quiero [requerimiento], para [motivo]" y un conjunto de **criterios de aceptación** escritos siguiendo la metodología Behavior Driven Development (BDD) [17] con el formato de "dado [actor], cuando [realiza acción] entonces [sucede evento]".

Aunque se utilizó la técnica de INVEST [18] como estándar de definición para generar las historias de usuarios, algunas de ellas no terminaron cumpliendo todos los requisitos, particularmente el ser pequeñas; teniendo algunas que rompen ligeramente este criterio llevando duraciones estimadas de más de un Sprint. Esto se debió a que el cliente prefería trabajar con criterios de aceptación en historias más largas, ya que le resultaban más cómodas de visualizar. Sin embargo, las tareas que se desprendieron de las historias fueron más cercanas a lo que se consideran tradicionalmente como "historias de usuario".

Una vez que se generó una primera versión del documento inicial convertido en historias de usuario, se presentó al cliente para su validación y, a partir de sus observaciones, se desarrolló una segunda versión que finalmente fue aprobada.

Luego de finalizadas las historias y sus criterios de aceptación, el equipo realizó una estimación del tamaño de cada una, utilizando la métrica de puntos de historia y la técnica de Planning Poker [19], en la que participaron todos los desarrolladores.

Después de la estimación, el equipo trabajó en conjunto con el cliente durante las reuniones establecidas para priorizar las diferentes historias. Este enfoque de priorización se realiza para evitar sesgos, de manera que el equipo no se vea influenciado por la importancia que el cliente pueda dar a cada historia al momento de estimar su tamaño, y para que el cliente se centre únicamente en las funcionalidades que desea que se desarrollen primero.

Con el fin de registrar las historias, el equipo optó por utilizar la plataforma Jira de Atlassian [11], la cual permite no solo la creación de tarjetas, sino también la extracción de diversas

métricas y la planificación de iteraciones, entre otras funcionalidades que resultan de gran ayuda para la gestión del proyecto.

En orden de poder estructurar de manera ordenada todas las historias, el equipo decidió desglosarlas en 4 tipos de *tickets*: épicas, historias de usuario, tareas y *spikes*. Las épicas son grandes historias de usuario que se encuentran conformadas por múltiples historias "más pequeñas". Estas últimas son divididas en tareas a la hora de definir en qué se trabajará en el Sprint. Adicionalmente, se tienen los *spikes* que representan esfuerzos de investigación por parte del equipo de duración variable y estimable. En forma de ejemplo podemos tomar la épica "analizar documento"; la misma es un objetivo grande, el cual necesita ser particionado para poder ser ejecutado de forma correcta. Para ello entran las historias de usuario, que se refieren a acciones que puede realizar el usuario dentro de la épica (como por ejemplo en este caso, subir un documento y luego procesarlo) y por último, dentro de las historias de usuarios, se encuentran tareas y *spikes*, las cuales corresponden a pequeños hitos para realizar la historia de usuario y búsqueda de información de cómo realizar la historia respectivamente.

4.5.5 Requerimientos funcionales resultantes

Con el objetivo de mejorar la comprensión de la lista de historias de usuario que se detalla a continuación, se han agrupado por épica. Dentro de cada una se podrá ver el listado de historias de usuario con su explicación, cuerpo, puntos de historia y por último sus criterios de aceptación y si fueron realizados correctamente o no. Asimismo, también se puede notar que las historias fueron agrupadas según su relevancia temática y no en función del orden cronológico en que se llevarán a cabo.

Épica 1: Importación y análisis de documentos

Contexto: El proceso de mantenimiento y gestión de los parques eólicos en nuestro país conlleva un desafío enorme en lo que respecta al seguimiento. Todo desde un chequeo de rutina a una reparación de emergencia genera un documento particular, el cual actualmente es administrado en forma manual. Se busca que se puedan importar archivos al sistema de Renovus con tal de automatizar y organizar documentos. A continuación sus historias de usuario. (Tabla 1, 2 y 3)

Importación de documentos al sistema de Renovus

Explicación: El primer eslabón de la aplicación correspondió a la carga de documentos al sistema de Renovus para luego poder ser analizado y procesado. Al comenzar el proyecto, esta historia fue la de mayor prioridad, dado que fue necesaria para poder desarrollar las siguientes historias.

[REN-12]: Importación de documentos al sistema de Renovus	**Story points: 13**

Cuerpo de la historia: COMO usuario de Renovus QUIERO poder importar documentos al sistema PARA poder recuperarlos y analizarlos más tarde.

Id	Criterios de aceptación	Cumplido
CDA1	Dado un usuario de Renovus, cuando seleccione un documento de su ordenador de forma correcta para procesar, entonces este se subirá al sistema para ser analizado.	Si
CDA2	Dado un usuario de Renovus, cuando suba un documento y asocie un generador al mismo, entonces el sistema vinculará el documento a ese generador cuando se suba.	Si
CDA3	Dado un usuario de Renovus, cuando suba un documento debe ser capaz de asociar una lista de categorías predefinidas al mismo, entonces el sistema vinculará el documento a esas categorías cuando se suba.	Si
CDA4	Dado un usuario de Renovus, cuando subió un documento, con un generador y categorías seleccionadas de forma correcta, entonces el sistema mostrará un resumen de los datos subidos antes de confirmar la acción para dar seguridad de que el sistema está por procesar los datos correctamente.	Si
CDA5	Dado un usuario de Renovus, cuando el sistema confirma la subida de un documento exitoso, entonces se informa al usuario de que el mismo se subió correctamente con un mensaje por pantalla.	Si

CDA6	Dado un usuario de Renovus, cuando ocurre una falla en la subida de un documento, entonces se informa del error al usuario.	Si
CDA7	Dado un usuario de Renovus, cuando suba varios documentos en simultáneo exitosamente, entonces el sistema debe ser capaz de procesarlos por separado, aplicando los parámetros seleccionados por el usuario a todos los documentos.	Si

Tabla 1: Historia de usuario REN-12

Extracción de información de documentos importados

Explicación: Una vez subido el documento, el siguiente paso es el procesado y análisis del mismo, para así luego ser categorizado y organizado.

[REN-14]: Extracción de información de documentos importados	**Story points: 20**

Cuerpo de la historia: COMO usuario de Renovus, QUIERO poder extraer la fecha, el número de aerogenerador, el título del documento y las conclusiones de cada documento PARA poder buscarlo con esos criterios más adelante.

Id	Criterios de aceptación	Cumplido
CDA1	Dado un usuario de Renovus, cuando se sube un documento exitosamente, entonces se extraerá el nombre del documento para ser filtrado a futuro	Si
CDA2	Dado un usuario de Renovus, cuando se sube un documento exitosamente, entonces se extraerá la fecha en la cual fue subido para ser filtrada a futuro.	Si
CDA3	Dado un usuario de Renovus, cuando se sube un documento exitosamente, entonces las categorías a las cuales corresponde son registradas en el análisis.	Si

CDA4	Dado un usuario de Renovus, cuando se sube un documento exitosamente, entonces el generador al que corresponde es registrado en el análisis.	Si
CDA5	Dado un usuario de Renovus, cuando se sube un documento exitosamente, entonces se extraerá del mismo frases clave que servirán para buscarlo más tarde.	Si
CDA6	Dado un usuario de Renovus, cuando se sube un documento exitosamente, entonces se extraerá del mismo el texto del documento (como texto plano) que servirá para buscarlo más tarde.	Si
CDA7	Dado un usuario de Renovus, cuando se sube un documento exitosamente, entonces se extraerá del mismo valores clave que servirán para buscarlo más tarde.	Si
CDA8	Dado un usuario de Renovus, cuando se sube un documento exitosamente, entonces el mismo será asignado un identificador único en el sistema.	Si

Tabla 2: Historia de usuario REN-14

Categorización de documentos

Explicación: Cada documento debe ser categorizado al momento de subir el mismo para poder ser filtrado a futuro.

[REN-15]: Categorizar de documentos	**Story points: 4**
Cuerpo de la historia: COMO usuario de Renovus, QUIERO poder agregar etiquetas a mis documentos PARA poder filtrarlos en futuras búsquedas.	

Id	Criterios de aceptación	Cumplido
CDA1	Dado un usuario de Renovus, cuando un documento es procesado y analizado, entonces se debe categorizar el documento para poder así buscarlo rápidamente.	Si
CDA2	Dado un usuario de Renovus, cuando un usuario selecciona las categorías de un documento exitosamente, entonces se debe vincular las categorías seleccionadas con el documento.	Si

Tabla 3: Historia de usuario REN-15

Épica 2: Gestión de documentos

Contexto: Luego de la importación de documentos y su posterior análisis, el sistema ha de responder acorde a los filtros requeridos. Para esto se forma esta épica compuesta de las siguientes historias (Tablas 4, 5 y 6)

Organización de documentos

Explicación: El usuario debe ser capaz de ver sus documentos organizados para así lograr su búsqueda rápida. Al ser un sistema que maneja miles de documentos, la organización de los mismos fue fundamental para así lograr una buena experiencia de usuario. El objetivo de la

organización es que el usuario pudiese organizar sus documentos por generador y por parque para así su rápida visualización.

[REN-16]: Organización de documentos	**Story points: 14**

Cuerpo de la historia: COMO administrador de Renovus QUIERO tener mis documentos organizados en base a los parámetros analizados PARA así poder filtrarlos cuando el usuario desee.

Id	Criterios de aceptación	Cumplido
CDA1	Dado un documento en el sistema, cuando el mismo haya sido procesado, entonces el documento deberá almacenarse bajo un árbol de carpetas visible a un administrador con la siguiente configuración: compañía/generador	Si
CDA2	Dado un documento en el sistema, cuando el mismo haya sido procesado, entonces la información debe almacenarse con una configuración clara que permita la vinculación de los datos obtenidos con el documento	Si
CDA3	Dado un documento en el sistema, cuando el mismo haya sido procesado, entonces la información almacenada debe tener atributos clave que faciliten su posterior filtrado.	Si

Tabla 4: Historia de usuario REN-16

Búsqueda rápida de documentos

Explicación: Una vez organizados los documentos, el usuario debe ser capaz de filtrar y organizar los mismos para así su búsqueda rápida. De esta manera pudiendo así visualizar una lista de todos los documentos y opciones de filtrado por parque, generador, título, fecha, categorías y palabras claves.

[REN-13]: Búsqueda rápida de documentos	Story points: 27

Cuerpo de la historia: COMO usuario de Renovus QUIERO poder buscar rápidamente documentos en el sistema PARA poder encontrar los mismos.

Id	Criterios de aceptación	Cumplido
CDA1	Dado un usuario de Renovus, al ingresar a la página de búsqueda de documentos exitosamente, entonces se mostrarán todos los documentos filtrados por un generador por defecto y el parque seleccionado.	Si
CDA2	Dado un usuario de Renovus, cuando filtre documentos por un generador en la página de búsqueda exitosamente, entonces se mostrarán todos los documentos para ese generador.	Si
CDA3	Dado un usuario de Renovus, cuando filtre documentos por un título en la página de búsqueda exitosamente, entonces se mostrarán todos los documentos con ese título.	Si
CDA4	Dado un usuario de Renovus, cuando filtre documentos por categorías en la página de búsqueda exitosamente, entonces se mostrarán todos los documentos con esas categorías.	Si
CDA5	Dado un usuario de Renovus, cuando filtre documentos por un rango de fechas en la página de búsqueda exitosamente, entonces se mostrarán todos los documentos subidos en esas fechas.	Si
CDA6	Dado un usuario de Renovus, cuando filtre documentos por palabras claves en la página de búsqueda exitosamente, entonces se mostrarán todos los documentos que contengan las palabras claves.	Si
CDA7	Dado un usuario de Renovus, cuando filtre documentos dentro de la lista por fecha, entonces se mostrarán los documentos ordenados de mayor a menor fecha o viceversa.	Si

CDA8	Dado un usuario de Renovus, cuando filtre documentos dentro de la lista por título, entonces se mostrarán los documentos ordenados alfabéticamente de mayor a menor por título.	Si
CDA9	Dado un usuario de Renovus, cuando filtre documentos dentro de la lista por categorías, entonces se mostrarán los documentos ordenados por categorías.	Si
CDA10	Dado un usuario de Renovus, cuando seleccione la opción de borrar un documento, entonces el sitio mostrará una alerta de borrado previo a confirmar.	Si
CDA11	Dado un usuario de Renovus, cuando confirme la opción de borrar un documento, entonces el sistema borrara el documento seleccionado.	Si
CDA12	Dado un usuario de Renovus, cuando seleccione la opción de ver detalles de un documento, entonces el sistema navegará a la página de detalles del documento seleccionado.	Si
CDA13	Dado un usuario de Renovus, cuando seleccione la opción de ver tareas de un documento, entonces el sistema navegará a la página de tareas del documento seleccionado.	Si

Tabla 5: Historia de usuario REN-13

Detalles de documento

Explicación: Uno de los intereses más grandes para el cliente fue que el usuario pudiese visualizar los detalles de cada documento rápidamente, pudiendo ver datos importantes extraídos del mismo, como también visualizando el documento directamente desde la *web* sin tener que descargarlo.

[REN-17]: Detalles de documento	Story points: 11

Cuerpo de la historia: COMO usuario de Renovus QUIERO poder ver los detalles de un documento PARA poder analizarlo y así ver datos importantes extraídos del mismo.

Id	Criterios de aceptación	Cumplido
CDA1	Dado un usuario de Renovus, cuando ingrese a la página de detalles de un documento, entonces se mostrarán detalles como título, fecha de creación y categorías asociadas.	Si
CDA2	Dado un usuario de Renovus, cuando seleccione la opción de ver más detalles de documento, entonces se mostrará una tabla que contenga número, identificador de documento, su parque y generador asociado.	Si
CDA3	Dado un usuario de Renovus, cuando se ingrese a la página de detalles de un documento exitosamente, entonces se podrá observar datos claves del documento extraídos de la extracción de datos del documento realizado previamente.	Si
CDA4	Dado un usuario de Renovus, cuando se ingrese a la página de detalles de un documento exitosamente, entonces se podrá observar, descargar, rotar e imprimir el PDF del documento subido.	Si

Tabla 6: Historia de usuario REN-17

Épica 3: Generación de tareas

Contexto: Luego de la importación de documentos y su posterior análisis, el sistema ha de responder acorde a los filtros requeridos. Para esto se forma esta épica compuesta de las siguientes historias (Tablas 7 y 8)

Gestión de tareas por documento

Explicación: El usuario debe ser capaz de registrar, editar y borrar tareas asignadas para cada documento, para así facilitar el mantenimiento del generador. De esta forma, el usuario es capaz de, una vez analizado el documento, asignar tareas a distintos usuarios y gestionar las mismas.

[REN-23]: Gestión de tareas por documento	**Story points: 15**

Cuerpo de la historia: COMO usuario de Renovus QUIERO poder gestionar tareas de un documento específico PARA poder realizar un seguimiento de ellas y asignar tareas al personal del parque.

Id	Criterios de aceptación	Cumplido
CDA1	Dado un usuario de Renovus, cuando complete los datos necesarios de una nueva tarea para un documento y confirme, entonces esa tarea se guarda en el sistema y se muestra en la tabla de tareas asignadas para el documento seleccionado.	Si
CDA2	Dado un usuario de Renovus, cuando selecciona una tarea de la lista de tareas, la edita y confirma su edición, entonces el sistema actualizará la tarea seleccionada.	Si
CDA3	Dado un usuario de Renovus, cuando borra una tarea de la lista de tareas exitosamente, entonces se borrara la tarea de la lista y del sistema.	Si
CDA4	Dado un usuario de Renovus, cuando ordena el listado de tareas por nombre, prioridad o estado, entonces la lista se ordenará alfabéticamente o numéricamente por estas categorías.	Si
CDA5	Dado un usuario de Renovus, cuando se selecciona una tarea de la lista de tareas, entonces se mostrará un listado de documentos asociados a esa tarea.	Si
CDA6	Dada una tarea en el sistema, cuando se interactúa con la misma,	Si

	entonces debe tener un estado que puede ser OPEN o CLOSED.	
CDA7	Dada una tarea en el sistema, cuando se interactúa con la misma, entonces debe tener una criticidad que puede ser HIGH, MEDIUM o LOW.	Si

Tabla 7: Historia de usuario REN-23

Gestión de tareas por generador

Explicación: Al igual que con cada documento, el usuario debe ser capaz de visualizar el total de las tareas para cada generador, por ende fue necesaria la creación de una vista en la cual el usuario pudiera seleccionar un generador y así ver todas sus tareas, agregar nuevas, editarlas y borrarlas. A su vez, también el usuario debe ser capaz de asignar varios documentos a una tarea o ninguno.

[REN-24]: Gestión de tareas por generador	**Story points: 14**

Cuerpo de la historia: COMO usuario de Renovus QUIERO poder gestionar tareas de un generador específico PARA poder realizar un seguimiento de ellas y asignar tareas al personal del parque.

Id	Criterios de aceptación	Cumplido
CDA1	Dado un usuario de Renovus, cuando complete los datos necesarios de una nueva tarea para un generador específico y confirme, entonces se creará una tarea nueva para el generador indicado.	Si
CDA2	Dado un usuario de Renovus, cuando edite una tarea de un generador específico y confirme, entonces la tarea se actualizará en el sistema para todos los documentos asociados.	Si

CDA3	Dado un usuario de Renovus, cuando se borre una tarea de un generador específico, entonces se borrara la tarea para el generador y para todos los documentos asociados.	Si
CDA4	Dado un usuario de Renovus, cuando seleccione el filtrado de tareas por prioridad, estado o nombre, entonces la lista se ordenará alfabéticamente o numéricamente dependiendo de la columna.	Si
CDA5	Dado un usuario de Renovus, cuando seleccione una tarea, entonces el sistema mostrará todos los documentos asociados a la tarea seleccionada.	Si
CDA6	Dado un usuario de Renovus, cuando cree o edite una tarea y seleccione varios documentos para una tarea, entonces el sistema asociará la tarea a todos los documentos seleccionados.	Si
CDA7	Dado un usuario de Renovus, cuando se cambie la selección de generador, entonces el sistema mostrará la lista de tareas para el generador seleccionado.	Si
CDA8	Dado un usuario de Renovus, cuando busque documentos para asociar la tarea por título, entonces el sistema mostrará todos los documentos con el título solicitado	Si
CDA9	Dado un usuario de Renovus, cuando borre una tarea, entonces el sistema borrara la tarea para todos los documentos asociados.	Si

Tabla 8: Historia de usuario REN-24

4.5.6 Requerimientos no funcionales

De forma conjunta a los requerimientos anteriores, a medida que progresaba la elicitación de requerimientos se definió una serie de requerimientos no funcionales. Estos, en su mayoría, fueron resultado del análisis realizado tanto por el equipo de desarrollo como por los expertos de Renovus (Tabla 9).

Código	Nombre	Descripción no detallada	Definido por
RNF 1	Repudio a plataformas fuera de AWS	El sistema debe correr en servicios o infraestructura de Amazon Web Services.	Cliente
RNF 2	Performance en subida de documentos	El sistema debe permitir que al subir un documento, este no demore más de un día en ser procesado y analizado, para luego ser mostrado al cliente.	Cliente
RNF 3	Performance en búsqueda de documentos	El sistema debe permitir la búsqueda de documentos en el sistema en un tiempo menor a cinco segundos.	Expertos
RNF 4	Idioma del sistema	El sistema debe ser elaborado en su totalidad en inglés, incluyendo sus interfaces de usuario y código.	Cliente
RNF 5	Crecimiento vertical y horizontal	El sistema debe ser escalable tanto vertical como horizontalmente.	Clientes
RNF 6	Confiabilidad en análisis de documentos	Se requiere alcanzar un análisis mayor o igual a 85% de la información presente en los documentos	Cliente
RNF 7	Fecha límite	El sistema debe ser entregado en la fecha establecida.	Equipo estudiante
RNF 8	Usabilidad del sistema	El sistema debe ser fácil e intuitivo de utilizar, manteniendo el diseño y el flujo de la página de Renovus	Cliente
RNF 9	Bitácoras en Cloudwatch	El sistema debe tener un registro de bitácoras (*logs*) utilizando Cloudwatch	Cliente
RNF 10	Región AWS	El sistema debe utilizar la región *us-east2* de Amazon Web Services cuando sea posible.	Cliente

Tabla 9: Requerimientos no funcionales

Seguidamente se muestran estos requerimientos no funcionales en detalle (Tablas 10, 11, 12, 13, 14, 15, 16, 17, 18 y 19).

RNF 1: [Integrabilidad] Repudio a plataformas fuera de AWS

Cuerpo de la historia: COMO Renovus, QUIERO que el sistema corra en servicios o infraestructura de Amazon Web Services, PARA facilitar la integración al sistema actual.

Criterios de aceptación:

Escenario:	Se desarrolla el sistema
Se quiere:	Que el sistema corra sobre la infraestructura de Amazon Web Services Que sea sencillo de integrar con el sistema existente
No se quiere:	Que el sistema corra en servicios ajenos a la infraestructura de Amazon Web Services Que sea compleja la integración con el servicio existente.

Tabla 10: RNF1

RNF 2: [Performance] Performance en subida de documentos

Cuerpo de la historia: COMO Renovus, QUIERO que el sistema permita la carga de archivos al sistema en un tiempo no superior a un día, PARA poder buscar los documentos de forma rápida más adelante.

Criterios de aceptación:

Escenario:	Se sube un archivo al sistema
Se quiere:	Que el mismo no tarde más de un día en ser procesado
No se quiere:	Que se tarde un tiempo superior a un día en procesar un archivo

Tabla 11: RNF2

RNF 3: [Performance] Performance en búsqueda de documentos

Cuerpo de la historia: COMO Renovus, QUIERO que el sistema permita la búsqueda de archivos en el sistema de forma rápida, PARA poder encontrar los documentos que se necesiten de forma veloz.

Criterios de aceptación:

Escenario:	Se intenta encontrar un archivo en el sistema
Se quiere:	Que se encuentre el mismo en un tiempo aproximado de 5 segundos máximos.
No se quiere:	Que se tarde un tiempo superior a 10 segundos buscando un documento

Tabla 12: RNF3

RNF 4: [Usabilidad] Idioma del sistema

Cuerpo de la historia: COMO Renovus, QUIERO que el sistema PARA usuarios sea desarrollado en idioma inglés para poder apuntar a la clientela norteamericana.

Criterios de aceptación:

Escenario:	Se intenta desarrollar una interfaz de usuario
Se quiere:	Que sus funcionalidades e interfaces se encuentren en su totalidad en idioma inglés
No se quiere:	Que la interfaz y funcionalidades se encuentren en otro idioma que no sea el inglés.

Tabla 13: RNF4

RNF 5: [Escalabilidad] Crecimiento vertical y horizontal

Cuerpo de la historia: COMO Renovus, QUIERO que el sistema sea fácil de escalar tanto vertical como horizontalmente, PARA poder soportar demandas variantes.

Criterios de aceptación:

Escenario:	Se tiene el proyecto corriendo en producción y se le requiere una carga alta inusual.
Se quiere:	Que el sistema tenga posibilidad de escalar rápidamente de forma horizontal o vertical.
No se quiere:	Que el sistema no logre soportar una demanda exigente. Que el tiempo de respuesta varíe dependiendo de la cantidad de documentos subidos y procesados.

Tabla 14: RNF5

RNF 6: [Fiabilidad] Confiabilidad en análisis de documentos

Cuerpo de la historia: COMO Renovus, QUIERO alcanzar un alto grado de detección de la información de los documentos, PARA poder garantizar un filtrado y categorización efectiva.

Criterios de aceptación:

Escenario:	Se carga un documento para analizar al sistema
Se quiere:	Que el sistema logre analizar las entidades y frases clave del documento con una precisión mayor o igual al 85%.
No se quiere:	Que el sistema considere entidades o frases clave con una precisión menor al 70%.

Tabla 15: RNF6

RNF 7: [Entrega] Fecha límite

Cuerpo de la historia: COMO equipo estudiante, QUIERO cumplir con las fechas de entrega planificadas (30 de marzo de 2023), PARA poder cumplir con los objetivos planteados y obtener la aprobación del proyecto.

Criterios de aceptación:

Escenario:	Se plantea una fecha límite de entrega
Se quiere:	Entregar el sistema completo y correcto antes del 30 de marzo del 2023.
No se quiere:	Entregar el sistema pasado el 30 de marzo de 2023

Tabla 16: RNF7

RNF 8: [Usabilidad] Usabilidad del sistema

Cuerpo de la historia: COMO Renovus, QUIERO que las nuevas funcionalidades de la página *web* mantengan el *look and feel* de la página actual y que sean fáciles de usar por un usuario que conozca el sitio, PARA poder permitir una buena experiencia de usuario.

Criterios de aceptación:

Escenario:	Se realizan nuevas páginas necesarias para las nuevas funcionalidades
Se quiere:	Un sistema que en su totalidad comparta el mismo diseño y mismo flujo.
No se quiere:	Un sistema que tenga diseños distintos y sea difícil de interpretar para un usuario familiarizado con el sitio.

Tabla 17: RNF8

RNF 9: [Observabilidad] Bitácoras en Cloudwatch

Cuerpo de la historia: COMO Renovus, QUIERO que el sistema contemple un registro de *logs* utilizando Amazon Cloudwatch PARA poder capturar errores y tener un registro de ejecución de los pasos claves del proceso de análisis de documentos.

Criterios de aceptación:

Escenario:	Se realizan un análisis de un documento
Se quiere:	Un sistema con *logs* que lleve el registro de las operaciones realizadas y resultados obtenidos
No se quiere:	Un sistema sin registro de eventos realizados ni detección de errores.

Tabla 18: RNF9

RNF 10: [Implementación] Región AWS

Cuerpo de la historia: COMO Renovus, QUIERO que todos mis sistemas en la nube se encuentren en la región US-EAST2, PARA así mantener el sistema en la misma plataforma y misma región.

Criterios de aceptación:

Escenario:	Se realiza el sistema de Renovus
Se quiere:	Todos los sistemas se encuentren en la misma región US-EAST2
No se quiere:	Alojar el sistema en un región que no sea US-EAST

Tabla 19: RNF10

4.5.7 Validación de requerimientos

Como fue mencionado anteriormente en el documento, el equipo logró validar muchos de los requerimientos mediante el uso de metodologías como Design Thinking, que promueve la realización de entrevistas con el cliente y la prototipación de la aplicación. De las dos actividades, la de prototipación probó ser la más efectiva, ya que los prototipos brindaron una forma de visualizar los requerimientos y permitieron la capacidad de estudiar la usabilidad de la aplicación antes del desarrollo.

Los prototipos se desarrollaron en Figma, herramienta la cual permitió a los usuarios experimentar con un prototipo casi como si fuese una página *web* real, pudiendo así interactuar con la misma, presionando botones, cambiando de páginas, pero sin la necesidad de codificarla.

Técnicas de prototipación y evolución

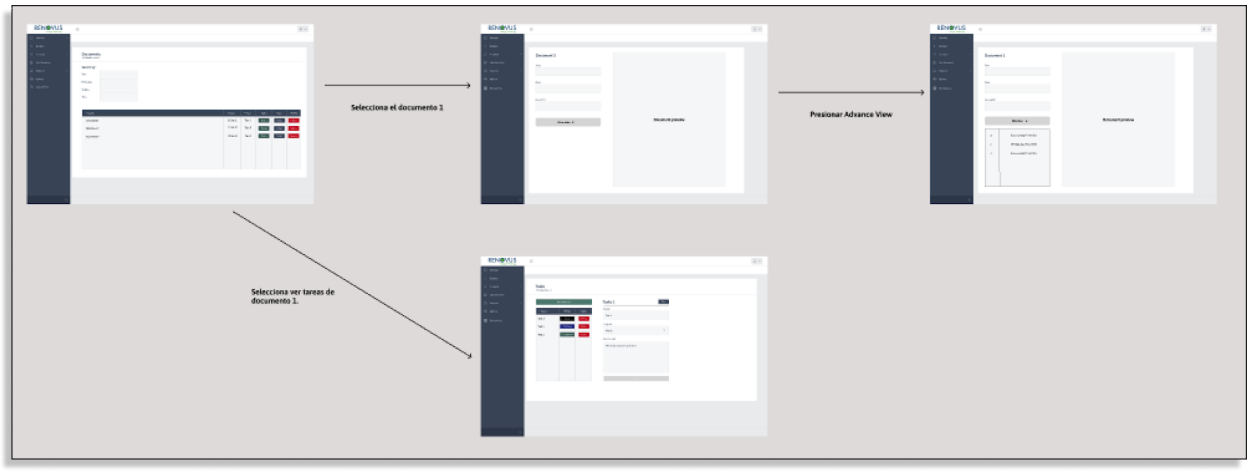


Figura 32: Prototipo inicial

El motivo del primer prototipo fue el de validar la idea general del proyecto, como se puede ver en la Figura 32, si bien no cumplía exactamente la definición de prototipo al ser más cercano a un conjunto de diapositivas de posibles pantallas, permitió al cliente observar el flujo probable de la aplicación. Dado que el equipo sabía que iba a ser un proceso sobre el que se iba a iterar, se optó por comenzar con un entendimiento del flujo de las páginas primero.

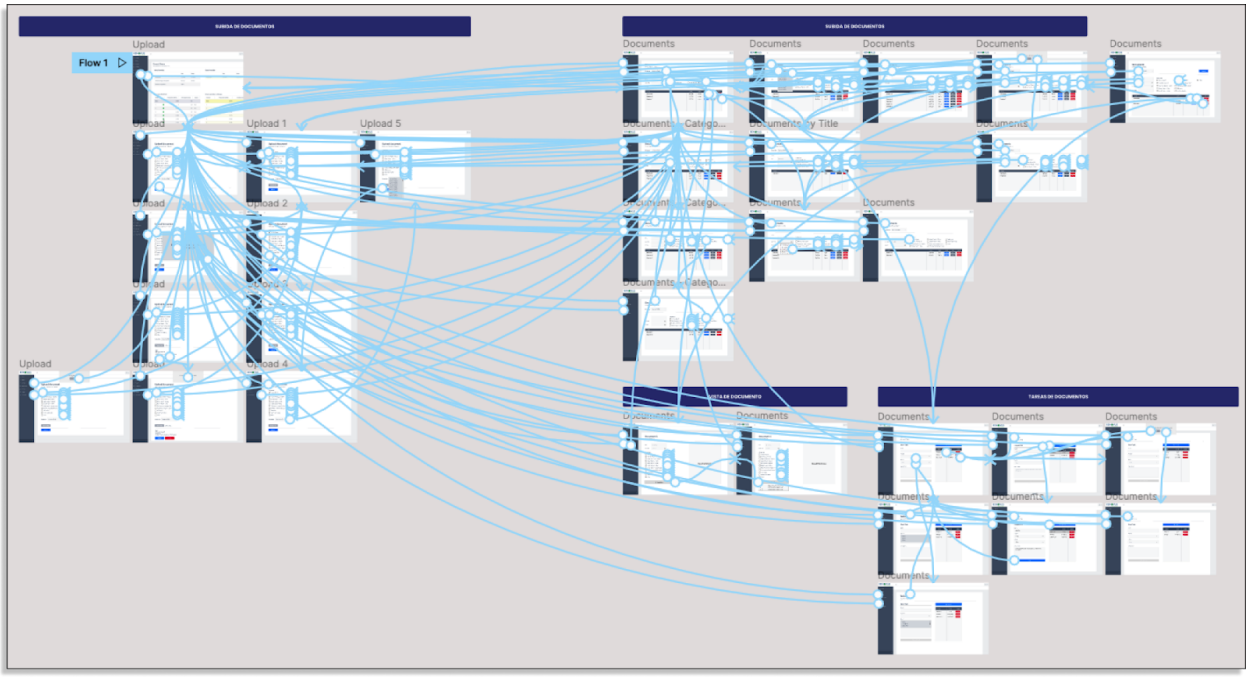


Figura 33: Prototipo intermedio

Una vez validado el flujo inicial se dio comienzo con el prototipo real. Cada pantalla es una posible interacción del usuario con el sistema, las cuales al ser accionadas permiten el desplazamiento a otra pantalla como si de un sitio real se tratase (Figura 33).

Este prototipo fue probado con el equipo cliente y con usuarios finales del sistema cuya retroalimentación fue vital para mejorar el entendimiento del equipo sobre algunos detalles que se habían pasado por alto a la hora de generar alguno de los requerimientos. La experiencia y conocimiento del usuario final resultó especialmente valioso, ya que no sólo estaba familiarizado con el sitio *web* de Renovus, sino que también ofreció un punto de vista realista de la utilidad del sitio en desarrollo para los clientes de Renovus. Gracias a sus comentarios, se pudieron reevaluar algunos puntos clave del sistema y se pudieron implementar mejoras significativas en el producto (Figura 34).

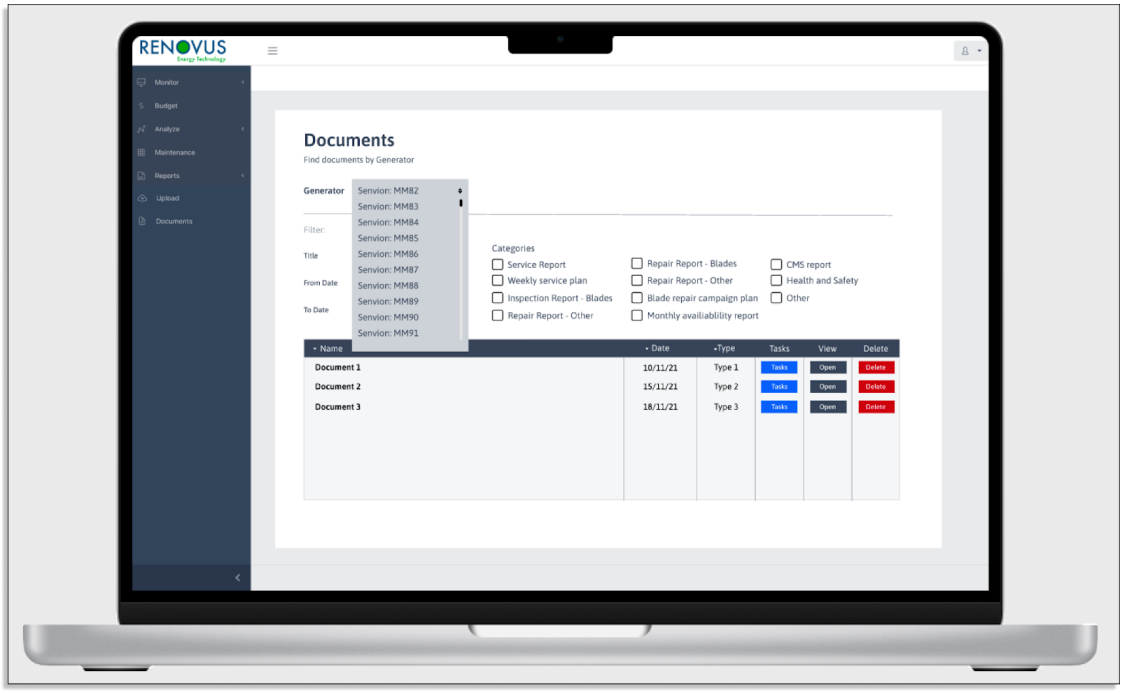


Figura 34: Prototipo en acción

Por último, una vez probada su usabilidad y luego de varias reuniones con el cliente, fueron agregados pequeños detalles como corrección de textos, diseños de botones y pequeñas variaciones estéticas, así concluyendo el prototipo deseado (Figura 35).

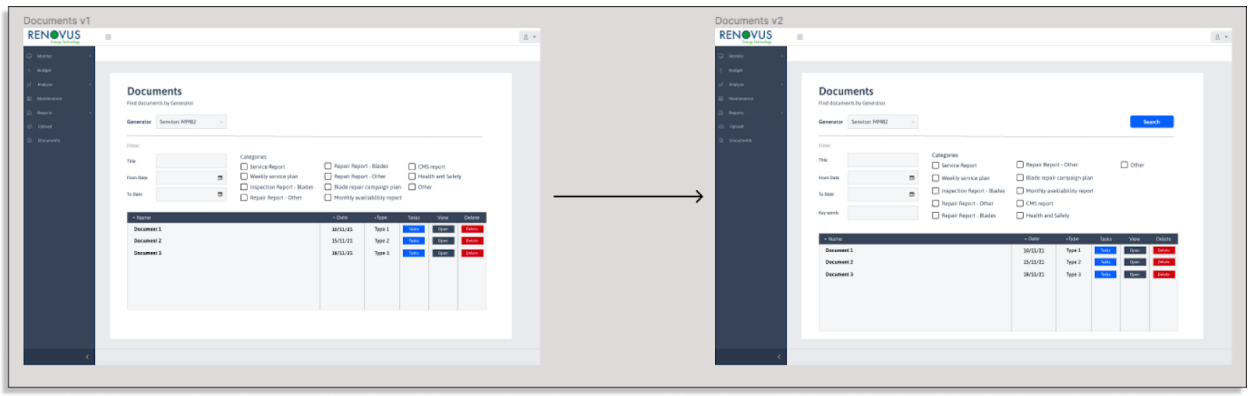


Figura 35: Prototipo final, pequeños cambios

5. Arquitectura

En este capítulo se detallarán las decisiones tomadas en lo que respecta a la arquitectura planteada para la solución. Se hará un recorrido por el sistema, argumentando la elección de las tecnologías y su propósito en lo que respecta a los requerimientos de la aplicación. Para esto, se realizará una introducción general del sistema a fin de lograr una comprensión de alto nivel, previo a la profundización a través de las vistas principales del sistema.

5.1 Principales módulos del sistema

A continuación se presenta un diagrama de referencia de arquitectura de Amazon Web Services [20] como un primer acercamiento a la solución. Este diagrama sustituye en parte a la vista de despliegue, ya que en cierta forma comunica la misma información. Se advierte al lector que el siguiente diagrama no contiene el *backend* del sistema actual de Renovus, es decir, solamente refiere al aplicativo realizado por el equipo estudiante a excepción del *frontend* que representa un trabajo conjunto entre ambas partes (Figura 36)

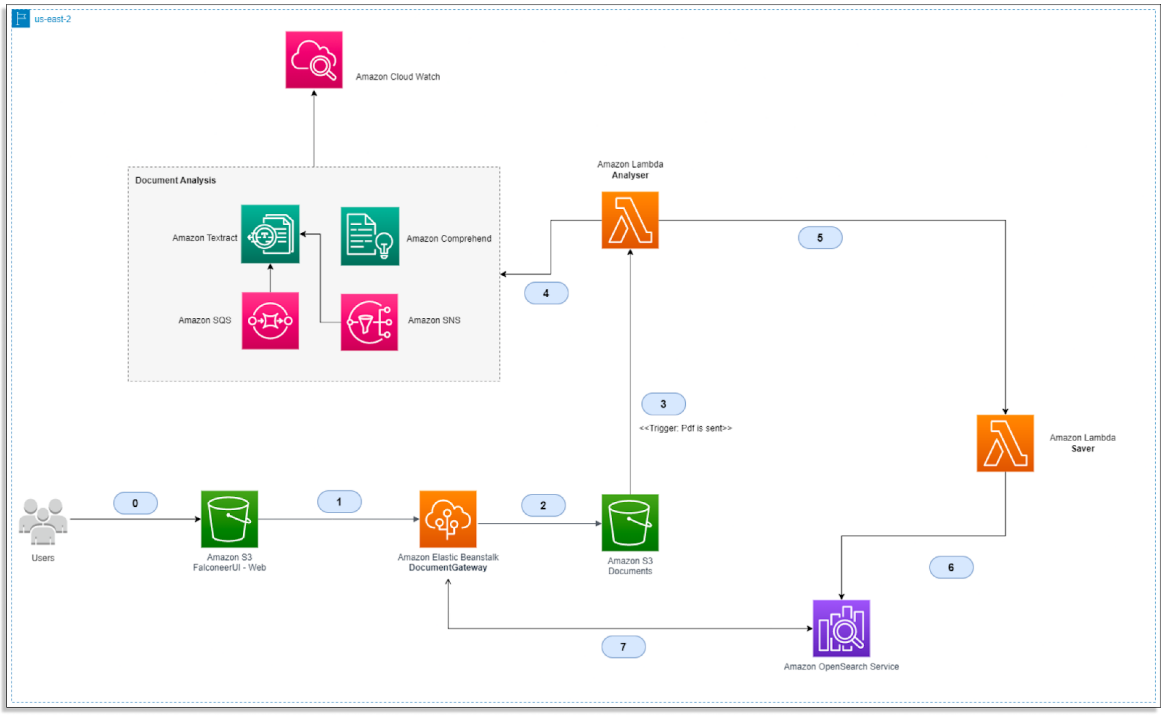


Figura 36: Arquitectura del sistema

5.1.1 Breve explicación de los servicios involucrados

FalconeerUI

Este servicio representa el *frontend web* de la aplicación y se trata del resultado de un esfuerzo conjunto entre el equipo estudiante y Renovus. Los estudiantes trabajaron sobre la solución existente a modo de generar los apartados correspondientes a las funcionalidades planteadas en los requerimientos del proyecto.

Las responsabilidades del servicio son el punto de contacto de los usuarios con las funcionalidades del proyecto, permitiendo la subida de archivos, gestión y búsqueda de documentos, así como las tareas asociadas a los mismos.

DocumentGateway

Se trata de una WebAPI construida en .NET Core que sirve como punto de acceso a la nube de Renovus para la gestión de documentos. Esta WebAPI se encarga de manejar una variedad de tareas relacionadas con la gestión y administración de documentos, así como a tareas vinculadas.

Se presenta entonces como la principal interfaz de búsqueda para que servicios como FalconeerUI puedan buscar documentos en el sistema mediante consultas *web* y adicionalmente ofrece el punto de contacto con las funcionalidades vinculadas a la administración de tareas de los documentos y aerogeneradores del sistema.

Documents Bucket (S3)

Documents Bucket es un contenedor de objetos en AWS, también conocido como *bucket*, que se utiliza para almacenar y acceder a los documentos del sistema de forma rápida y escalable. Aunque el nombre real en el sistema es "renovus-test-bucket-ort-1", se le denomina "Documents Bucket" o “Documents” por simplicidad.

El *bucket* está cuidadosamente controlado para asegurar que los documentos de una compañía no se mezclen con los de otra, y para facilitar el acceso a los documentos específicos de una compañía o generador dentro de la misma.

Analyzer

Se trata de una función Lambda compleja, diseñada como un “orquestador” a los servicios encargados de analizar los documentos. Su invocación es desencadenada por un disparador interno de Amazon al subir un archivo a “Documents Bucket”.

Document Analysis

Document Analysis no es un servicio *per se*, sino que se trata de una agrupación de servicios relacionados con el análisis de documentos. En primer lugar, tenemos Amazon Textract, encargado de la conversión del documento a texto mediante técnicas de OCR y luego Amazon Comprehend, servicio configurado por el equipo para el correcto estudio y obtención de datos claves de documentos. Adicionalmente, se hace uso de distintos servicios de apoyo, como lo es una cola de mensajes para manejar la carga de múltiples documentos que podrían entorpecer las operaciones, un servicio de notificaciones llamado SNS que informa sobre tareas de larga duración completadas (puntualmente el análisis de Textract) y un servicio de Amazon llamado CloudWatch destinado al monitoreo de los *logs* propios de este conjunto de servicios.

Saver

La función Lambda "Saver" tiene como principal tarea guardar la información procesada por el servicio previo en el formato correspondiente dentro de la base de datos. Como su nombre sugiere, su función es asegurar que el almacenamiento se realice correctamente y en el formato adecuado. Además, esta función también encapsula las responsabilidades de cambiar el formato en el futuro, si fuese necesario.

OpenSearch Service

Este servicio es la base de datos principal del proyecto, donde se almacenan los documentos procesados junto con metadatos que permiten un filtrado rápido y eficiente en el sistema, ya sea por nombre, palabras clave, fecha o aerogenerador. Además, esta base de datos también se encarga de almacenar las tareas relacionadas con dichos documentos.

5.1.2 Explicación de los pasos relevantes

Si volvemos a examinar el diagrama de referencia, podremos ver una secuencia numérica del 0 al 7. Los mismos representan pasos relevantes a la funcionalidad de carga y procesado de archivos, la cual fue dictaminada como la más relevante para el entendimiento del sistema y elegida para la numeración requerida por el diagrama.

- 0. El usuario interactúa con la *web*, actualmente alojada en un *bucket* de S3.
- 1. Se efectúa la carga del documento al servicio DocumentGateway, quien valida y procesa la solicitud
- 2. De estar todo correcto se procede a subir el archivo a Documents Bucket.
- 3. Se lanza un disparador interno de Amazon invocando a Analyzer.
- 4. La función Analyzer coordina las llamadas a los diferentes servicios. Primero, encola el documento y se subscribe a la notificación de procesado de Textract en un tópico SNS. Una vez que el documento se ha convertido en texto plano, lo envía a Comprehend para su análisis. Este servicio extrae información importante como palabras clave, datos relevantes, entidades y una comprensión general del texto, ya que ha sido entrenado para procesar este tipo de datos. Todas las operaciones se registran en las bitácoras (*logs*) de CloudWatch.
- 5. El documento es enviado a Saver.
- 6. Saver optimiza el formato de los documentos para mejorar la búsqueda en OpenSearch Service y facilitar su lectura, garantizando un almacenamiento adecuado en esta base de datos.
- 7. Con la información indexada de forma correcta, DocumentGateway es capaz de realizar consultas a OpenSearch Service quien le suministra la información procesada y le brinda la ubicación del archivo en el bucket apropiado para que lo pueda obtener más adelante.

Nota: aunque la funcionalidad de tareas no se tiene en cuenta para este flujo, no se agregan servicios adicionales ni relaciones imprevistas al diagrama, por lo que no fueron consideradas para no causar confusiones.

5.1.3 Omisiones del documento

Este documento no incluye la gestión de usuarios, ya que esta tarea es responsabilidad del cliente y está fuera del alcance del proyecto. Dichos usuarios fueron suministrados por Renovus y se administran en sus propios servicios. Este módulo específico de usuarios y todo lo relacionado con el *backend* realizados únicamente por Renovus sin intervención del equipo estudiante, no forman parte de este documento.

Adicionalmente, el trabajo del equipo estudiantil relacionado con las alertas de facturación por sobrecostos y la gestión de roles IAM (*Identity and Access Management*) realizada por el equipo, no se considera en este documento, ya que no aportan valor al entendimiento del sistema. Sin embargo, estos elementos se tienen en cuenta en algunos componentes individuales.

5.2 Descripción de la arquitectura

En esta sección se dará lugar a una descripción de la arquitectura más detallada, partiendo inicialmente de la selección del estilo arquitectónico como los microservicios y las tecnologías en la nube, pasando a vistas detalladas como las de componentes y conectores.

5.2.1 Comentarios sobre microservicios

En el desarrollo de *software*, la arquitectura de microservicios se ha convertido en una tendencia popular en los últimos años. Esta técnica de desarrollo se centra en la construcción de aplicaciones complejas a través de la combinación de pequeños servicios que trabajan juntos de forma independiente. A continuación, se detalla la elección del equipo de adoptar la arquitectura de microservicios y el uso de algunos módulos *serverless*, en lugar de integrarse directamente en la solución existente.

Inicialmente, al equipo se le fue ofrecido la opción de integrarse en la solución existente, lo que implicaría trabajar con algunos módulos del *backend* existentes junto al líder de tecnologías del cliente. Sin embargo, después de un análisis detallado, se decidió que la mejor opción era desacoplarse completamente y adoptar una solución basada en microservicios y módulos *serverless*.

En la siguiente sección se revisarán los beneficios asociados al estilo arquitectónico de microservicios, los cuales se derivan de ciertos atributos de calidad. En secciones posteriores se profundizará en estos atributos.

Beneficios microservicios

En cuanto a la escalabilidad, los microservicios permiten escalar horizontalmente cada servicio. Esto se logra individualmente en función de la carga que se esté experimentando en ese momento. Ello significa que si un servicio está recibiendo una gran cantidad de tráfico, podemos agregar más instancias o más poder para ese servicio sin afectar el resto del sistema. Formalmente, esto quiere decir que los microservicios permiten el escalado vertical y horizontal de los mismos. De esta manera, podemos asegurar que la aplicación se mantenga rápida y eficiente, incluso en momentos de alta demanda, por ejemplo, al realizarse varias consultas particularmente difíciles a la base de datos, que pueda requerir más poder para no comprometer la performance.

En cuanto a la mantenibilidad, los microservicios facilitan la comprensión del sistema y la localización de errores. Cada servicio es más pequeño y se enfoca en una sola responsabilidad específica, lo que hace que sea más fácil entender cómo funciona y cómo interactúa con los demás. Además, si un servicio presenta un problema, solo se necesita trabajar en ese en particular, en lugar de tener que revisar toda la aplicación.

Este estilo arquitectónico permite el desarrollo y la implementación de servicios de forma independiente, lo que significa que se pueden actualizar, reemplazar o eliminar servicios sin afectar al resto de la aplicación. Esto es especialmente útil en un contexto de cambio constante, donde las necesidades del negocio pueden cambiar rápidamente. Con esta arquitectura, se puede

adaptar la aplicación de manera más ágil y rápida a las nuevas necesidades del negocio sin tener que rehacer toda la aplicación.

El equipo se enfocó en la separación de responsabilidades en los microservicios para cumplir con el principio de clausura común [21], agrupando módulos en componentes con las mismas responsabilidades.

Esta separación de responsabilidades fue la que garantizó la independencia del *frontend* y el *backend*. Incluso funciona manteniendo la independencia entre servicios, en donde cada uno ignora el funcionamiento interno del otro, comunicándose mediante interfaces bien definidas y totalmente agnósticas a un lenguaje particular.

En lo que respecta a desplegabilidad, los microservicios permiten una mayor agilidad en la introducción de cambios. En lugar de tener que desplegar toda la aplicación para realizar una pequeña actualización, los cambios se pueden realizar en un servicio específico y desplegarse de forma independiente. Esto significa que es posible introducir cambios más rápido y con menos riesgo de interrupciones del servicio.

Es importante destacar que existe un compromiso en términos de performance al adoptar el estilo arquitectónico de microservicios. Si bien esta opción permite escalar rápidamente la solución, también introduce una sobrecarga (*overhead*) en la comunicación interna, como puede ser la cola de mensajes o el envío de mensajes HTTP. No obstante, tras un estudio realizado por el equipo, se determinó que este *tradeoff* era la mejor opción para el proyecto.

5.2.2 Vista de componentes y conectores

A la hora de adentrarse en el desafío de entender un sistema complejo de *software*, una de las labores más importantes es identificar y definir la arquitectura del mismo. Al hacer esto, se puede obtener una visión clara y detallada de la estructura del sistema, sus relaciones entre componentes y conectores, así como la forma de manejar la información. Esta descripción detallada puede ayudar a los desarrolladores a comprender mejor cómo funciona el sistema, cómo se comunican sus componentes y cómo se gestionan los datos.

La vista de Componentes y Conectores es una de las representaciones más estandarizadas de la arquitectura de un sistema, y se centra en los componentes que tienen mayor importancia en tiempo de ejecución, como los servicios, procesos y bases de datos, así como en los conectores y caminos por donde circula la información (Figura 37 y Tabla 20).

Para un acercamiento al sistema desde el punto de vista de los módulos (es decir, código estático) del sistema se recomienda ver el Anexo 10.5 donde se detallan estas vistas.

Representación primaria de la aplicación

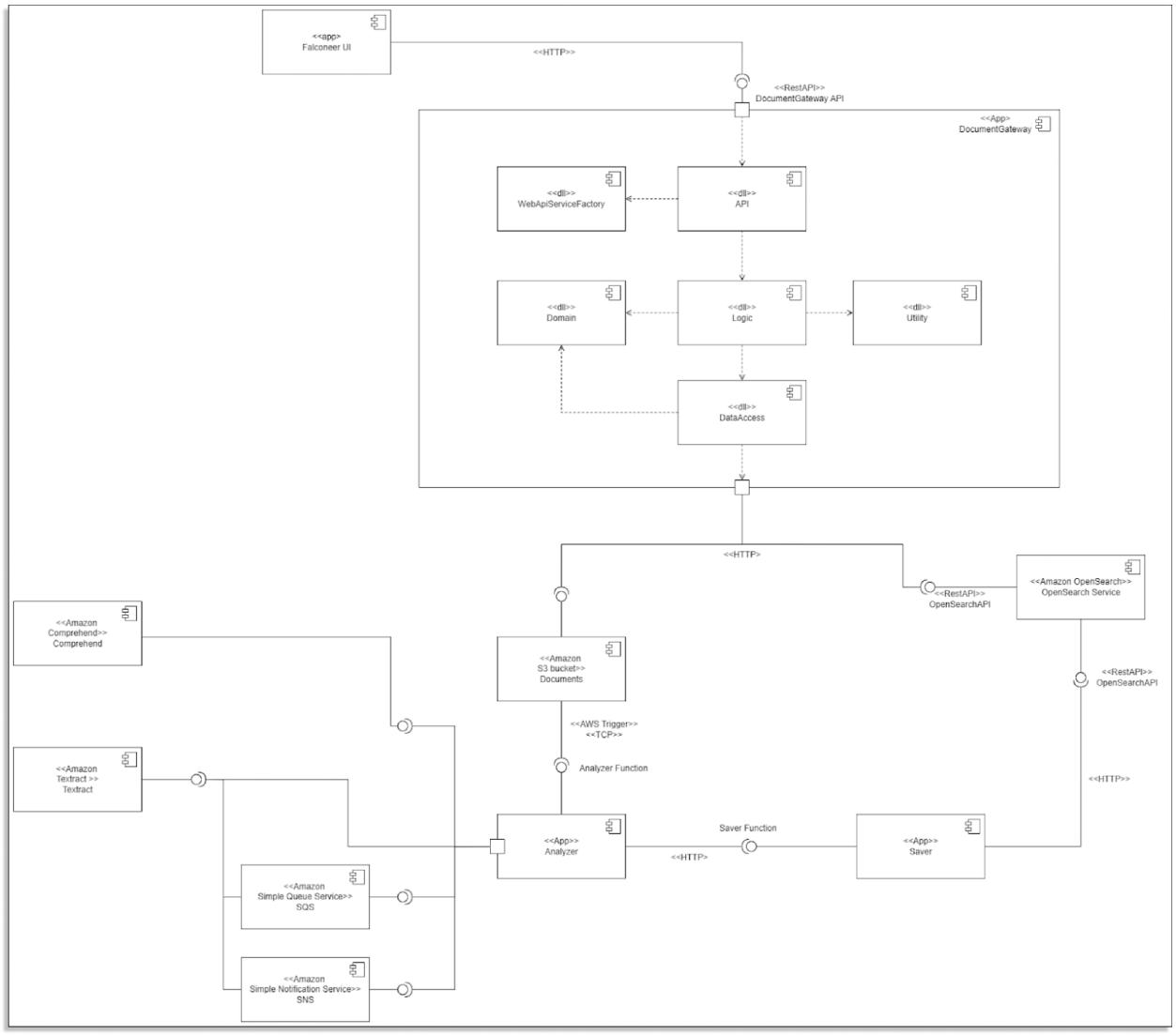


Figura 37: Representación primaria de la aplicación

Catálogo de elementos

Elemento	Tipo	Responsabilidad
Analyzer	APP	Orquestador del análisis de documentos. Posee un algoritmo de apoyo a los componentes a los que llama.
Api	DLL	Interfaz que provee el *backend* como medio de interacción entre este, la *web* y el resto del sistema siguiendo el estilo arquitectónico REST.
Comprehend	Amazon Comprehend Service	Componente encargado de analizar los documentos mediante inteligencia artificial. El mismo fue entrenado por el equipo para mejores resultados.
DataAccess	DLL	Componente donde se realizan las comunicaciones pertinentes al acceso a datos del sistema.
DocumentGateway	APP	API encargada de manejar una variedad de tareas relacionadas con la administración de documentos.
Documents	Amazon S3 Bucket	Contenedor de documentos del sistema.
Falconeer UI	APP	Servicio *frontend* de la aplicación, encargada de presentar la misma al usuario de forma sencilla. No se detallan por completo sus componentes individuales a modo de simplificar el diagrama.
Logic	DLL	Componente encargado de ejecutar la lógica de la aplicación.

OpenSearch	Amazon OpenSearch Service	Base de datos no relacional especializada para el almacenaje y búsqueda de texto plano. Motor de las búsquedas de documentos en el sistema.
Saver	APP	Componente encargado de guardar la información procesada sobre los documentos con cierto formato en la base de datos.
SNS	Amazon Simple Notification Service	Servicio encargado de controlar y notificar sobre la completitud del procesamiento del texto en Textract a la función Lambda Analyzer.
SQS	Amazon Simple Queue Service	Cola de mensajes encargada de limitar las llamadas al componente de Textract, permitiendo procesar cada documento de forma apropiada
Textract	Amazon Textract Service	Servicio de Amazon que representa el componente encargado de realizar el reconocimiento óptico de caracteres (OCR) a los documentos suministrados
Utility	DLL	Servicio de apoyo a Logic encargado de ejecutar funciones comunes o reutilizables.
WebApiServiceFactory	DLL	Servicio que mediante inyección de dependencias elimina relaciones entre los módulos inferiores del sistema y la API.

Tabla 20: Catálogo de elementos

Elementos y sus interfaces

En el diagrama presentado, cada componente expone una interfaz definida que ha sido cuidadosamente diseñada para asegurar su correcto funcionamiento y fácil integración con otros componentes del sistema.

Es importante destacar que para aquellas interfaces que son de tipo HTTP, se ha seguido el estilo arquitectónico REST [22]. Asimismo, se ha mantenido una postura agnóstica al lenguaje de programación utilizado, lo que facilita su implementación y evita posibles problemas de compatibilidad.

Finalmente, cabe destacar que se ha puesto especial atención en documentar de manera clara y concisa cada una de las interfaces, con el fin de que sean fáciles de comprender y consumir desde el exterior del componente. En el Anexo 10.5 se pueden ver los detalles de algunas de estas interfaces.

Diagramas de comportamiento

A continuación, se presentan los diagramas de comportamiento más relevantes de la aplicación como una guía ilustrativa. Cabe destacar que estos diagramas representan flujos correctos sin errores y están destinados a ayudar al lector a comprender mejor el caso que se va a probar, omitiendo pasos que no contribuyan al correcto entendimiento del flujo (Figuras 38 y 39).

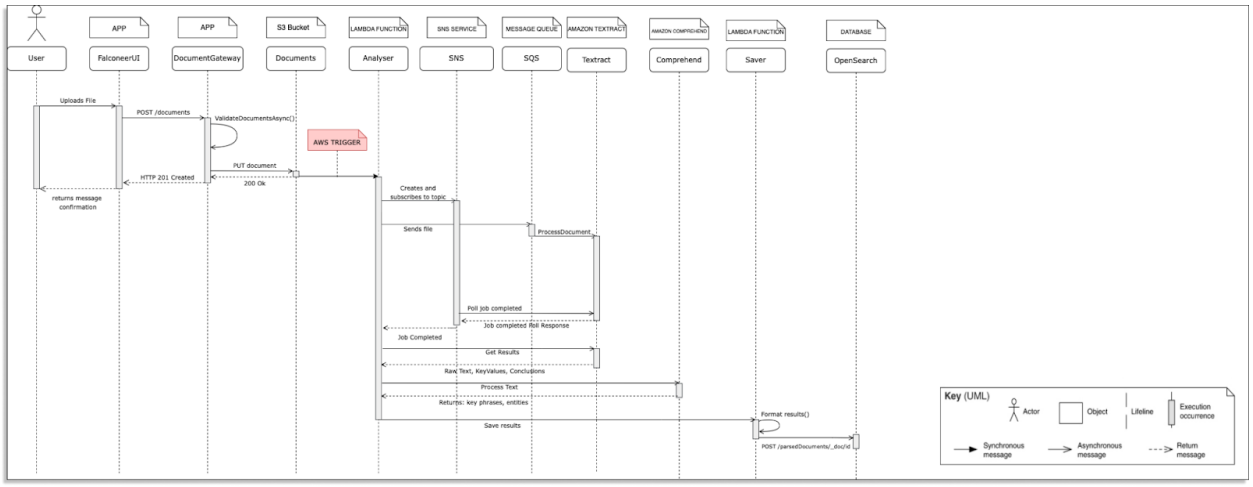


Figura 38: Diagrama de secuencia para subida de archivo

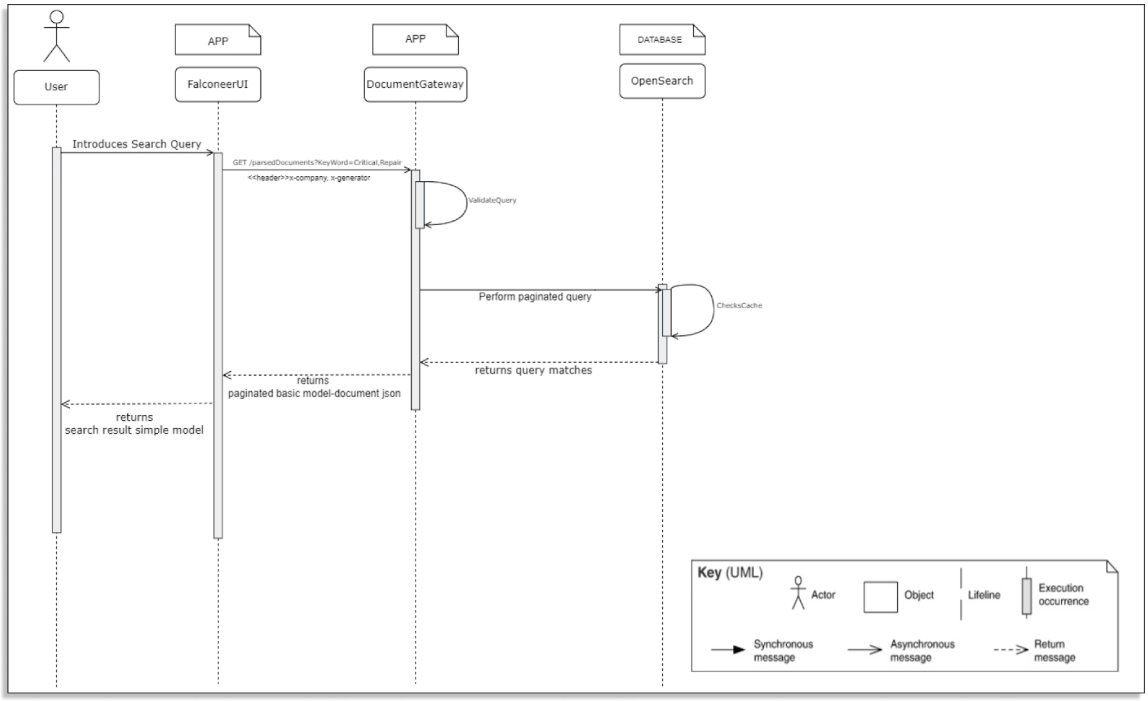


Figura 39: Diagrama de secuencia para búsqueda compleja de documentos según palabra clave

Luego de revisar los diagramas, se pueden realizar algunos comentarios de interés. Primero, se puede notar que realizar una consulta sobre documentos del sistema requiere menos servicios involucrados que la carga de documentos, ya que el procesamiento pesado se realiza antes. Adicionalmente, algún lector quizás esté considerando la posibilidad de agregar un microservicio intermediario entre DocumentGateway y OpenSearch, esto fue considerado brevemente por el equipo, pero en este caso, el *trade-off* no vale la pena en términos de performance al introducir *overhead* para una consulta pesada únicamente en la base de datos.

Es importante señalar que en el diagrama de flujo para la carga de documentos se utiliza el patrón Background Job, que será explicado con más detalle en la sección de tácticas y patrones.

Finalmente, se destaca que todas las llamadas en los diagramas son asíncronas, lo que contribuye al rendimiento del sistema.

5.2.3 Comentarios generales sobre Cloud Computing

Cloud computing es una tendencia cada vez más extendida en el mundo del desarrollo de *software*, y en el caso del proyecto, no fue una elección en sí, sino un requerimiento impuesto

por el cliente. Sin embargo, el equipo estaba contento de explorar este modelo de computación en la nube, con la que ya había tenido algún encuentro en su vida académica y laboral. Renovus contaba con su solución ya desplegada en AWS, lo que fue una gran referencia a la hora de despejar algunas dudas.

Una de las ventajas principales de *cloud computing* es que no requiere mantenimiento de infraestructura, lo que significa que no es necesario actualizar ni monitorear *hardware*. Además, resulta sencillo migrar el negocio a otra zona de disponibilidad (*AZ* por sus siglas en inglés), y el escalar con políticas definidas es una tarea fácil.

El uso de contenedores Docker, a los que el equipo está acostumbrado, también simplifica el proceso de implementación de la aplicación en la nube. Los contenedores son fáciles de crear y distribuir, lo que significa que el proceso de implementación se puede automatizar fácilmente. Además, los contenedores pueden ser desplegados y escalados de manera independiente, lo que permite una mayor eficiencia en el uso de recursos y una mayor capacidad de respuesta a las fluctuaciones en la demanda de la aplicación.

La implementación de microservicios se alinea muy bien con el uso de *cloud computing*, lo que le permitió al equipo aprovechar los servicios de Amazon para facilitar el desarrollo. La adopción de la computación en la nube resultó fundamental para alcanzar los objetivos del proyecto

5.3 Atributos de calidad, tácticas y patrones

En esta sección se describe como la arquitectura del sistema buscó satisfacer distintos atributos de calidad. Estos atributos de calidad se desprenden en su mayoría de los RNF definidos para el proyecto, los cuales se detallan en la sección 4.5.6 junto al atributo al cual se los vincula. Además, algunas decisiones tomadas por el equipo en cuanto a la arquitectura, las herramientas utilizadas y otros aspectos, también introdujeron aspectos vinculados a atributos de calidad. A continuación, se desarrolla para cada atributo qué tácticas y patrones se aplicaron a fin de satisfacerlos.

Usabilidad

Para empezar, se consideró la importancia de mantener el "*look and feel*" de la aplicación para evitar que los usuarios tengan que aprender algo nuevo. Por lo tanto, se decidió mantener la misma fuente de letra que se ha utilizado en la aplicación hasta el momento, para no generar confusiones en los mismos. También se establecieron las dimensiones y colores de los botones para que sean coherentes con el estilo visual y se evite la sensación de que se trata de una herramienta diferente.

Asimismo, se establecieron los formatos de tabla que ya están familiarizados por los usuarios para que la navegación y la visualización de la información sea lo más intuitiva posible. En cuanto a los colores principales, se decidió mantener los mismos que se han utilizado en la aplicación hasta el momento, para evitar que el cliente tenga que aprender una nueva gama de colores y pueda identificar rápidamente la marca.

Se consideró además, agregar los elementos nuevos a la barra lateral izquierda, sin que esto afecte la usabilidad o el diseño general de la aplicación. De esta forma, se espera que los usuarios puedan interactuar con la herramienta de manera fluida y sin ningún tipo de inconveniente, manteniendo la misma apariencia y estilo visual que ya conocen y valoran.

También fueron realizadas tareas de validación con usuarios y se tomó en cuenta su *feedback* para hacer cambios en la *UI/UX* y asegurarnos de que se adapte bien a sus necesidades y preferencias.

En cuanto a la subida de archivos, fue implementada una retroalimentación clara y fácil de entender para el usuario que busca indicar si la operación fue exitosa o no, en forma de una gran marca de verificación (*checkmark* en inglés) verde en la pantalla, mientras de fondo fue aplicado el patrón *Background Job* [23]. Además, se evita que los usuarios realicen acciones erróneas al tener campos requeridos y validaciones intuitivas en la aplicación.

También se consideró la importancia de los filtros fáciles de usar y que permitan volver al estado inicial para el usuario, lo que ofreció una forma fácil y rápida de ordenar los datos en la aplicación. Una *web* intuitiva es crucial para garantizar que los usuarios puedan utilizarla fácilmente.

Otro aspecto importante para el equipo fue seguir las heurísticas de Nielsen [24] en el aseguramiento de la calidad. Estas heurísticas se han desarrollado para garantizar una buena experiencia de usuario y la facilidad de uso.

Por último, el rendimiento es un atributo de calidad muy importante en relación con la usabilidad. Asegurarse de que la aplicación tenga tiempos de respuesta rápidos y eficientes, mejora la experiencia de usuario y hace que la aplicación sea más usable.

Performance

La velocidad de respuesta del sistema está directamente vinculada al atributo de calidad anterior, por lo que la performance tuvo que ser altamente cuidada, especialmente para las consultas a la base de datos que se realizarán sobre miles de documentos.

Puntualmente, se habla de que un parque típico en Uruguay de 20 aerogeneradores maneja entre 100 archivos de bitácoras y de planificación anuales, sumado a un rango de entre 20 a 40 archivos adicionales de inspección y otros 100 reportes de equipos por año. La solución Renovus está pensada para escalar con una cantidad indefinida de parques y ha de mantener consistencia en sus tiempos de respuesta y de ejecución.

La elección de algunas de las tecnologías que se detallarán en la siguiente sección están vinculadas directamente a este atributo de calidad. A continuación se presentan las principales tácticas para favorecer al performance que el equipo aplicó durante el desarrollo.

- Incrementar recursos: La primera táctica utilizada fue el incremento de los recursos, lo que permite aumentar la capacidad de respuesta del sistema, agregando más poder de cómputo. Esta fue aplicada a la hora de seleccionar el entorno donde se corre la base de datos responsable de consultas costosas en términos de procesamiento. Para esto se resolvió darle una máquina más poderosa que al resto de los servicios. Existe, obviamente, un *tradeoff* entre este incremento y el costo que implica mantenerlo.

- Mejora de la eficiencia de los recursos usados: Se buscó de forma adicional la mejora de eficiencia de los recursos a usar, por ejemplo, mediante la implementación de algoritmos de paginado para consultas difíciles a la base de datos a fin de obtener tiempos de

respuesta menores, ya que se corta la ejecución de búsqueda de coincidencias de forma más temprana.

- Mantener múltiples copias de datos: Esta táctica apunta a mantener copias de datos en algún medio de almacenamiento secundario con mejor velocidad de acceso. En el caso del equipo se posee un caché propio de OpenSearch que permite el almacenado temporal de consultas complejas para fácil acceso.

- Introducción de concurrencia: Para prevenir bloqueos en la ejecución, se ha adoptado una estrategia consciente de introducir concurrencia en todos los servicios implementados. Esto significa que se pueden procesar múltiples solicitudes simultáneamente, lo que aumenta la eficiencia y mejora la capacidad de respuesta del sistema en general.

Modificabilidad

La modificabilidad es un atributo de calidad crucial a considerar cuando se trabaja en una solución existente, especialmente si se planea que el proyecto quede en manos del cliente y deba ser mantenido sin la ayuda del equipo estudiante. Dado el ciclo de vida elegido, el equipo planeaba estar construyendo e iterando sobre el *software* existente, por lo que era importante garantizar que fuera fácilmente modificable.

Esta modificabilidad incluye tener una arquitectura clara y bien definida que facilite la comprensión de cómo funciona el sistema y cómo se relacionan sus componentes a disposición del cliente, así como la buena documentación del código para que otros desarrolladores puedan entenderlo y realizar cambios sin problemas. Se listan seguidamente, tácticas consideradas de modificabilidad que buscan que cuando un cambio sea requerido, el mismo pueda ser logrado dentro del menor tiempo y presupuesto posible.

- Separar módulos: Si un módulo tiene una cantidad significativa de funcionalidades (posiblemente resultado de una baja cohesión) se optaba por descomponer el mismo en módulos más pequeños a fin de reducir el costo del cambio de cada uno.

- Incrementar coherencia semántica: Si las responsabilidades A y B están en el mismo módulo y no están relacionadas, entonces hemos de dividirlas en diferentes módulos. El

propósito de mover estas responsabilidades reduce la probabilidad de efectos secundarios indeseados como que una responsabilidad dependa de otra diferente. Siguiendo el *Single Responsibility Principle* de los patrones SOLID [25], el equipo se propuso considerar cuidadosamente cada responsabilidad asignada a los módulos presentes y si debían mover las mismas a otro módulo. Esta táctica, junto a otras, derivan en acciones como, por ejemplo, la de tener un módulo de coordinación de procesado (Analyzer) y otro módulo diferente que fue separado del original para el guardado de la información (Saver). Aunque el ejemplo presentado refiera a la arquitectura en su totalidad, también se implementó esta técnica internamente a cada módulo del sistema.

- Encapsular: Sin presentarla de forma explícita en este capítulo ya se ha tratado indirectamente una noción del encapsulamiento. Este principio básico del desarrollo de *software* implica que un cambio a un módulo no debe propagarse a otro, mediante el rechazo consciente a la exposición de detalles de implementación internos y la dependencia de otros módulos, a excepción de interfaces definidas, se puede garantizar, que un cambio futuro en un módulo impactará mínimamente en los demás.

- Abstraer servicios comunes: Esta táctica es empleada al detectar que más de un módulo ofrece o requiere de un comportamiento similar, por lo que se abstrae el mismo eliminando módulos duplicados, garantizando que si, a futuro, se introduce un cambio en dicho comportamiento, sólo es necesario modificar un módulo.

- Restringir dependencias: Fue utilizada para restringir la comunicación entre componentes, evitando que cualquier componente se comunique con otros. Esta táctica se implementó teniendo en cuenta el patrón de capas, donde cada capa del sistema tiene una relación unilateral de "permitido usar". Es decir, cada capa solo puede utilizar los servicios de la capa inmediatamente inferior, sin tener conocimiento o acceso directo a las capas superiores.

- Diferir enlaces: La idea detrás de esta técnica es dejar que el sistema trabaje lo más flexiblemente posible sin dejar enlaces o dependencias difíciles de cambiar a lo largo de los módulos. Para esto, el equipo utilizó de forma obligatoria entornos “.env”, archivos de configuración y variables de entorno a lo largo de los diferentes módulos del sistema.

Adicionalmente, módulos como el de DocumentGateway utilizan sistemas como inyección de dependencias [26] para que las mismas se configuren en tiempo de ejecución. De esta manera, se garantizó que el código compilado pudiera realizar la parametrización de aquellas dependencias requeridas sin que estas estuvieran vinculadas de forma rígida a otros módulos.

Disponibilidad

La disponibilidad del sistema es de vital importancia, ya que se trata de un sistema diseñado para clientes que pagan por su uso y lo necesitan en todo momento. Esta es, al final del día, una herramienta de apoyo crucial para el éxito de su negocio, por lo tanto, se requiere una alta disponibilidad para garantizar la satisfacción del cliente y la imagen positiva de Renovus.

Para lograr esto se aplicaron diversas técnicas listadas a continuación.

- Monitor: De forma nativa, muchos de los componentes alojados en Amazon Web Services poseen la posibilidad de visualizar el estado de salud del mismo. Esto permite que, de encontrarse una falla, sea posible entender mediante un vistazo rápido cuál componente presenta el problema.

- Manejo de excepciones: Las excepciones se detectan y se informan al usuario de forma adecuada en caso de manifestarse. De forma adicional, mediante la observabilidad requerida en el sistema se pueden estudiar las bitácoras para determinar en qué momento del flujo fueron lanzadas.

- Repuesto (*Spare*): Mediante un simple cambio de configuración es posible activar la redundancia de datos en la base de datos y en el *bucket* del sistema. Esto evita la potencial pérdida de información ante un fallo y permite que estos repuestos tomen su lugar ante esta ocurrencia. Vale la pena aclarar que estos repuestos no fueron activados previos a la puesta en operación del sistema, dado el potencial aumento en costo que implican. Esta técnica no garantiza rápidamente la disponibilidad del sistema, pero sí asegura que, cuando el mismo se vuelva a poner en operación, mantenga la información previa al fallo.

Además, el uso de componentes nativos de AWS garantiza la disponibilidad de acuerdo al SLA de AWS con un 99.999%. Esto significa que, en caso de que se produzca una falla en uno de los componentes del sistema, AWS se cerciora que la misma será resuelta en un plazo de tiempo determinado, lo que permite mantener la disponibilidad del sistema y evitar interrupciones en el servicio.

Otros atributos considerados

Además de los atributos ya mencionados, existen otros atributos que también han sido considerados en el desarrollo del sistema.

Uno de ellos es la **portabilidad**, considerada según lo establecido en la norma ISO 25010 [27] y no con el libro tratado hasta este momento Software Architecture In Practice [28]. En este caso, se ha tenido en cuenta la necesidad de eliminar cualquier problema de mantenibilidad con el entorno, por lo que se ha hecho obligatorio el uso de Docker [29] para garantizar la portabilidad del *software*. Es posible considerar esto último como parte del estudio de la Modificabilidad, ya que ambos atributos están estrechamente relacionados.

En cuanto a la **interoperabilidad**, se ha aplicado, como fue mencionado previamente, una táctica de orquestación que permite una mejor integración entre los diferentes componentes del *software* de AWS. Esta táctica permite que, en caso de querer adicionar nuevos pasos al flujo de análisis de un documento, por ejemplo, alguna nueva inteligencia artificial para resumir el mismo, sea relativamente sencillo integrar estos pasos dentro de tiempo y costos aceptables.

Es importante tener en cuenta que, aunque no se haya detallado algún otro atributo de calidad en este texto, esto no significa que no se haya considerado en el proceso de desarrollo. Los atributos mencionados anteriormente son los que se han considerado como más importantes para el desarrollo del *software* en cuestión.

5.4 Justificación de tecnologías

5.4.1 Frameworks y lenguajes de programación

.Net Core y C#

El cliente solicitó para el módulo planteado de DocumentGateway el uso de Java o un lenguaje similar a Java, que fuera popular y que no se dejara de mantener en el futuro cercano. DotnetCore (.NET Core) cumple con estos requisitos al ser un *framework* que utiliza C#, un lenguaje muy popular, ampliamente utilizado en la industria y respaldado por Microsoft, lo que garantiza su mantenimiento a largo plazo.

Además, como el equipo ya había trabajado con estas herramientas durante su formación universitaria, tenía la confianza y la experiencia necesarias para desarrollar un producto de calidad y mantenible. Asimismo, .NET Core 6 es un *framework* muy eficiente en cuanto a consumo energético, rápido gracias a su compilador optimizado, posible uso de asincronía, y muy escalable para plataformas de gran calibre, lo que lo convierte en una excelente opción para desarrollar aplicaciones *web* de alta demanda. Adicionalmente, el marco y su lenguaje son nativos a la plataforma Linux, dejando atrás la reputación que su compañía madre ha tenido con esta en el pasado.

Finalmente, se aprovecharon numerosas de las funcionalidades que ofrece .NET Core 6, como el inyector de dependencias, las posibilidades de escribir sentencias LINQ [30], y numerosas bibliotecas NuGet [31] que simplificaron enormemente el desarrollo y las pruebas como lo fueron Bogus [32] para las pruebas unitarias, OpenSearchClient [33] para simplificar la sintaxis de consultas a la base de datos y Swagger UI [34] como forma de presentar los *endpoints* de la API al usuario.

React JS

Es en realidad el cliente quien solicita el uso de esta librería de Javascript para el *frontend web*. React JS es una tecnología ampliamente conocida por sus interfaces de usuario ricas y sencillas de usar, lo que hace que sea una opción ideal para garantizar una buena experiencia de usuario y de desarrollo.

El uso de esta tecnología supuso un aprendizaje por parte del equipo, pero gracias a la buena disposición del responsable de tecnología del cliente y de una enorme comunidad detrás de esta tecnología, se brindó la posibilidad de resolver cualquier dificultad y aprovechar al máximo las herramientas creadas por la comunidad. Adicionalmente, la velocidad, flexibilidad y rendimiento de esta tecnología, combinada con un código legible y sencillo de mantener, supuso un resultado de alta calidad acorde a los estándares del equipo y del cliente.

Python

Este lenguaje es el recomendado por AWS para el desarrollo de funciones Lambda, lo que garantiza su compatibilidad y eficiencia en la plataforma.

Se trata de un lenguaje simple y fácil de leer, lo que le permitió al equipo desarrollar sus funciones de forma rápida y eficiente. Si bien existían ciertas preocupaciones sobre su velocidad de ejecución, esto no fue un problema, ya que las funciones creadas no necesitan responder al cliente directamente, sino que realizan tareas en segundo plano.

Otra razón por la se eligió Python es porque el cliente ya conocía este lenguaje y poseían un módulo escrito en él. Esto implica que el futuro mantenimiento de las funciones sea mucho más sencillo.

5.4.2 Herramientas dentro de entorno AWS

Lambda

Las funciones Lambda de AWS son una forma de computación *serverless* que permite a los desarrolladores ejecutar código en la nube sin tener que preocuparse por mantener servidores.

Estas funciones son invocadas por eventos y ejecutan un código totalmente aislado al que se le asignan los recursos justos y necesarios para la ejecución del mismo.

Una de las principales ventajas de estas funciones es que las invocaciones son sencillas y no es necesario mantener ningún servidor. Esto reduce los costos de mantenimiento que se limitan al código en sí y se garantiza que las Lambdas se puedan ejecutar independientemente de una creciente demanda.

Además, las funciones Lambda son tolerantes a errores y se pueden configurar fácilmente para reintentar automáticamente en caso de fallos, lo que significa que el equipo pudo escribir un código más robusto y resistente a errores.

También son fáciles de monitorear, lo que significa que se pueden identificar rápidamente y solucionar problemas. Esto se logra a través de herramientas integradas en AWS, como CloudWatch, que permiten a los desarrolladores supervisar el rendimiento de sus funciones Lambda y tomar medidas para mejorar el rendimiento.

Por último, las funciones Lambda de AWS también admiten el uso de capas (*layers*), las cuales en este caso se utilizaron para bibliotecas de Python, lo que significa que el equipo pudo usar bibliotecas comunes sin tener que incluirlas directamente en su función. Dichas funciones fueron usadas para las conexiones a los módulos externos con librerías de HTTP comunes a ambas. Ello simplifica el desarrollo y reduce el tamaño del paquete de la función Lambda, reduciendo así costos y su mejora en el rendimiento.

Elastic Beanstalk (EB)

Elastic Beanstalk es un servicio de plataforma como servicio (PaaS, por sus siglas en inglés) de AWS que facilita el despliegue, la escalabilidad y la administración de aplicaciones *web* en la nube. Fue seleccionado por sobre una tradicional máquina virtual EC2 a fin de evitar complicaciones relacionadas con la administración de esta.

Elastic Beanstalk abstrae todo lo relacionado con el despliegue, generando políticas de autoescalado, versionado, control de entornos y garantiza costos bajos a demandas bajas de uso.

Amazon S3 buckets

Un *bucket* es un servicio de almacenamiento de archivos en la nube propio de Amazon Web Services. Inicialmente, el equipo cuestionó seriamente si utilizar este servicio o si era mejor codificar los documentos subidos al sistema en Base64 y guardar los mismos en una base de datos. Luego de evaluar cuidadosamente ambas opciones, se decidió utilizar AWS Buckets por diversas razones.

En primer lugar, los *buckets* de AWS ofrecen una solución de almacenamiento escalable, lo que significa que se puede almacenar y recuperar cualquier cantidad de datos sin preocuparse por los límites de almacenamiento. Además, el servicio ofrece una alta durabilidad al replicar los datos en múltiples zonas de disponibilidad dentro de una región, lo que minimiza el riesgo de pérdida de datos.

En segundo lugar, AWS Buckets proporciona una seguridad de datos sólida, lo que significa que es posible proteger los archivos contra accesos no autorizados y asegurar su integridad y confidencialidad.

En tercer lugar, los *buckets* son una opción rentable para el almacenamiento de archivos en la nube, ya que solo se paga por el almacenamiento utilizado, sin costos iniciales ni tarifas mínimas. Además, el servicio es fácil de usar, con una interfaz *web* simple y APIs que permiten la gestión del almacenamiento, el control de acceso y la transferencia de datos que fueron aprovechados desde servicios como DocumentGateway.

Finalmente, los S3 se integran perfectamente con otros servicios de AWS, lo que permite una fácil transferencia de datos entre diferentes servicios como apreciamos en los flujos presentados previamente.

Amazon Simple Queue Service (SQS)

Similar al servicio anterior, el equipo no estaba seguro de si debía usar éste en particular. A lo largo de la vida académica el equipo estaba acostumbrado al uso de otras colas de mensajes y existían ciertas dudas sobre la simplicidad y efectividad del servicio a tratar.

Sin embargo, tras adentrarse en la documentación y ejemplos proporcionados por Amazon, el equipo pudo comprender la simplicidad de integración de AWS SQS en su entorno y los grandes beneficios asociados a la disponibilidad y posibilidades que ofrece.

AWS SQS es una cola de mensajes totalmente administrada que permite a los desarrolladores desacoplar y escalar los componentes de su sistema. Esto significa que los mensajes pueden ser enviados y recibidos de forma asincrónica, lo que reduce el acoplamiento y aumenta la escalabilidad del sistema. Adicionalmente, esta cola proporciona alta disponibilidad y durabilidad, evitando pérdidas accidentales de información.

Amazon Simple Notification Service (SNS)

Manteniendo la misma línea de los últimos componentes de esta solución, el equipo no tenía la claridad de utilizar esta herramienta para la solución. Al realizar la investigación necesaria, el equipo entendió que era pertinente su inclusión para obtener los mejores resultados del módulo Document Analysis y complementar el SQS presentado anteriormente.

AWS SNS es un servicio de mensajería totalmente administrado que aplica el patrón de productores y consumidores para permitir la entrega de mensajes de manera eficiente. En este contexto, los productores, como Amazon Textract, envían mensajes de forma asincrónica a un tema o topic, que es un punto de acceso lógico y de comunicación. Los consumidores, como SQS, pueden suscribirse al canal de comunicación del productor para recibir la información deseada.

CloudWatch

Se trata de una plataforma de monitoreo y gestión de recursos de AWS que permite la recopilación, seguimiento y análisis de métricas, *logs* y eventos en tiempo real. Esto significa que se puede monitorear y analizar los *logs* del sistema y recibir alertas si algo está fuera de los parámetros deseados. La selección de esta tecnología llega un poco más tarde en el desarrollo que el resto, junto al requerimiento no funcional vinculado a la observabilidad (RNF 9), principalmente por la falta de conocimiento del equipo respecto a este servicio.

CloudWatch es altamente escalable, lo que significa que puede manejar grandes cantidades de *logs* sin afectar el rendimiento de nuestra aplicación. Además, la plataforma ofrece una alta durabilidad y disponibilidad de datos, lo que significa que los *logs* están siempre disponibles para su análisis y procesamiento.

CloudWatch ofrece una integración fácil con otros servicios de AWS, lo que permite monitorear y analizar *logs* en diferentes servicios de AWS como las funciones Lambda, Textract, Comprehend y demás.

Textract

Como se ha detallado en capítulos anteriores, la utilización de un servicio encargado del procesamiento de lenguaje natural (OCR) fue un punto de inflexión en el proyecto. Se manejaron dos rutas posibles que impactarían al proyecto enormemente, la creación de un algoritmo de reconocimiento propio que llevaría la mayor parte del tiempo destinado a la construcción del mismo (considerando la falta de experiencia del equipo en inteligencia artificial) o la utilización de herramientas de apoyo que podrían acortar tiempos enormemente y brindar resultados que maximicen el valor real al cliente.

Textract fue parte de la segunda ruta. Se trata de una herramienta basada en *machine learning* que extrae texto de documentos y busca identificar y comprender parte de la información obtenida. Automatiza entonces la extracción de información de documentos subidos al sistema Renovus incluso si estos fueran documentos con letra hecha a mano, como un documento de firma o similar.

Durante el proceso de selección de una herramienta para el procesamiento de lenguaje natural, el equipo tuvo en cuenta la importancia de garantizar la confidencialidad de la información debido a los acuerdos *Non-Disclosure Agreement (NDA)* especificados por Renovus. Fue por esto que se decidió por Amazon Textract, ya que cumple con los estándares de seguridad establecidos por marcos importantes como HIPAA y RGPD, que (junto a otros reglamentos propios de AWS como el acuerdo de clientes) garantizan este pilar de la seguridad de la información.

Conforme avanzaba el proyecto, el equipo quedó cada vez más satisfecho con la elección de Textract. La precisión alcanzada por esta herramienta, desarrollada por expertos en IA, habría sido imposible de lograr a tiempo con un algoritmo de reconocimiento propio.

Sin embargo, el costo era una preocupación. Afortunadamente, Textract ofrece un rango de capa gratuita muy generoso y acepta limitar el número de hojas a procesar mediante archivos de configuración, permitiendo mantener el costo bajo control.

Comprehend

Luego de procesar los documentos con Textract el equipo se encontró con una gran cantidad de texto sin estructurar que podría ser analizado a fin de obtener información relevante como lo deseaba Renovus. Fue en este punto donde se consideró la posibilidad de utilizar Amazon Comprehend para obtener esta valiosa información a partir de los datos recopilados.

Comprehend es una herramienta de procesamiento de lenguaje natural que se encarga de analizar y extraer información relevante de grandes volúmenes de texto. En este caso, su uso permitió identificar patrones y tendencias en los documentos procesados por Textract.

Adicionalmente, el equipo aprovechó la capacidad de Comprehend para entrenar su IA y mejorar los resultados obtenidos. Gracias a esto, se pudo ajustar el modelo de procesamiento de lenguaje natural a las necesidades específicas del proyecto y obtener resultados aún más precisos y personalizados.

La combinación de Textract y Comprehend permitió obtener resultados más precisos y valiosos para el cliente, minimizando los tiempos de desarrollo y permitiendo una integración sencilla al ser servicios de AWS, ayudando a maximizar el valor real ofrecido por el proyecto.

OpenSearch

La base de datos OpenSearch es el corazón del sistema. Se trata de una base de datos no relacional de código abierto altamente escalable, que se ajusta perfectamente a los requerimientos de alojamiento de los documentos procesados por Textract y Comprehend. Esta base está integrada con OpenSearch Dashboards, lo que facilita el análisis de información y brinda una interfaz gráfica cómoda para usuarios avanzados o administradores.

Una de las características más destacables de OpenSearch es su capacidad de búsqueda de texto completo y análisis, así como su capacidad de "búsqueda de k vecinos más cercanos" (*k-Nearest Neighbor* o k-NN), traducción a consultas SQL nativa, detección de anomalías, análisis de rastreos, búsqueda de texto completo y más. Esto permite que la base de datos sea muy útil para una variedad de casos de uso, como el análisis de registros, la funcionalidad de búsqueda de aplicaciones o también la búsqueda empresarial, casos que el equipo aprovechó para los documentos procesados con *machine learning*.

Además de estas funciones, OpenSearch también ofrece otros beneficios importantes. Por ejemplo, la función de "Fuzzy Searching" (utilizada por el equipo) que significa que los usuarios no tienen que escribir correctamente lo que buscan, ya que hace coincidencias aproximadas como si fuera un buscador como Google o DuckDuckGo. También cuenta con la capacidad de búsquedas definidas por el usuario, lo que permite que los mismos mejoren las búsquedas con el uso continuo. El caché propio a solicitud del desarrollador para consultas frecuentes (usado en los "get all" de la primera búsqueda por generador y compañía) permite un acceso más rápido y eficiente a información frecuentemente solicitada. Además, la función de "Sharding" permite que los datos se separen en varios fragmentos (*shards*) y se busquen en paralelo, lo que lo hace aún más rápido.

Inicialmente, el equipo consideró utilizar ElasticSearch, el proyecto original del que OpenSearch es un *fork* creado por Amazon. Sin embargo, se decidió utilizar OpenSearch, ya que eran prácticamente iguales y OpenSearch es más barato dentro del entorno de AWS.

OpenSearch también está distribuido en nodos, con uno maestro que recibe las peticiones y otros nodos de datos. Esto hace que sea muy rápido al repartir la carga de trabajo y tiene su propia "API Gateway". Esta funciona con consultas REST, lo que significa que todos los lenguajes que tienen la posibilidad de hacer consultas HTTP pueden consumirla sencillamente.

Es innegable que existió cierta curva de dificultad a la hora de entender como realizar consultas en el lenguaje propio de OpenSearch pese a realizarse desde la API, pero, con el avance del proyecto y la realización de diversos *spikes* el equipo pudo adaptarse y hacer uso de muchas de las funcionalidades listadas previamente para hacer, por ejemplo, búsquedas de palabras clave

que retornen documentos coincidentes basados en los resultados de las herramientas anteriores, todo aplicando paginado de una forma sencilla y fácil de entender.

OpenSearch probó ser una gran herramienta y una muy buena elección de base de datos, defendiendo claramente las postulaciones en el *marketing* de la herramienta y probando ser efectivamente rápido, escalable, flexible y fácil de usar una vez se supera esa curva de dificultad inicial.

5.5 Comentarios de cierre

Se ha logrado desarrollar una arquitectura del sistema que cumple con los requerimientos no funcionales y funcionales establecidos al inicio del proyecto. Además, esta arquitectura cumple con los atributos de calidad que sirvieron de guía al proyecto: performance, modificabilidad y disponibilidad. El equipo de desarrollo ha trabajado arduamente para lograr este resultado, y quedó satisfecho con el trabajo realizado.

Finalmente, como complemento al estudio del sistema, se presentan en los anexos los documentos de mantenimiento necesarios para garantizar un correcto funcionamiento a largo plazo (Anexo 10.5). Estos documentos incluyen información detallada sobre la arquitectura, el diseño, la implementación y la operación del sistema, lo que permite a los encargados del mantenimiento realizar las tareas necesarias con la mayor eficiencia y eficacia posibles.

6. Gestión del proyecto

6.1 Introducción

En este capítulo se expondrá minuciosamente la gestión del proyecto en distintas áreas a lo largo del proceso de desarrollo. Se analizarán las etapas del proyecto, así como también su planificación, ejecución y gestión.

6.2 Etapas del proyecto

A continuación se muestra una tabla que desglosa las fases fundamentales que se llevaron a cabo a lo largo del proyecto en cuestión (Tabla 21).

Fase	Ubicación en el tiempo	Hitos
Relevamiento	15 de abril - 31 de mayo	- Confirmación de proyecto - Primeras reuniones con el cliente. - Análisis y relevamiento de requerimientos.
Inicial	1 de junio - 23 de julio	- Prototipación. - Investigación y capacitación de tecnologías. - Validación de requerimientos. - Definición de arquitectura.
Construcción	24 de julio - 31 de enero	- Estimación y planificación de Product Backlog. - Planificación y ejecución. - Pruebas. - Primeras versiones de la aplicación.

Final	1 de febrero - 30 de marzo	- Traspaso de conocimiento. - Documentación del entregable. - Documentación de proyecto de grado.

Tabla 21: Etapas del proyecto

Etapa de relevamiento

En esta primera etapa el equipo debió hacer una puesta a punto sobre el proyecto, lo cual implicó realizar reiteradas reuniones con el cliente para entender su negocio, conocer el alcance solicitado y desglosar el problema. A su vez, el equipo conoció a su tutor y también comenzó la planificación semanal y la proyección hacía 1 año de trabajo.

Una vez asentados en el proyecto, el equipo comenzó con el análisis y relevamiento de requerimientos, etapa que conllevó varias reuniones con motivo de entender a fondo y no malinterpretar el proyecto.

Etapa inicial

La etapa inicial se vio definida por la necesidad de obtener todo lo necesario para comenzar con la codificación de manera correcta. Fue en este momento que se buscó definir una arquitectura tentativa para así poder analizar con los expertos del cliente y proyectar la futura investigación, así como la capacitación necesaria. Durante este periodo, el equipo se concentró en las tecnologías necesarias para realizar el proyecto, así como también en la validación de requerimientos que iban a ser necesarios.

Dado que las tecnologías necesarias para el proyecto no eran conocidas por el equipo, el proceso de capacitación cubrió gran parte de la etapa inicial.

Por último se realizaron los prototipos, que fueron, como se mencionó en capítulos anteriores, fundamentales para la validación de los requerimientos.

Etapa de construcción

En esta etapa se comenzó con la construcción del producto planificado, así como también con la aplicación de las metodologías seleccionadas.

Esta etapa cubrió una gran parte del proyecto, dado que contemplaba varios hitos, entre ellos la planificación, la ejecución y las liberaciones (*releases*), las cuales detallaremos más adelante.

Etapa final

Por último, la etapa de cierre del proyecto, fue en la que se produjo el traspaso de conocimiento con el cliente, detallando y explicando cada particularidad del sistema para que los profesionales a cargo de Renovus pudiesen mantener el mismo, no solo de manera escrita, sino a través de reuniones. Al mismo tiempo, y a fin de concluir el proyecto de grado, el equipo desarrolló la documentación correspondiente de la misma.

6.3 Planificación temporal

6.3.1 Sobre estimación

La estimación del esfuerzo en el desarrollo de *software* es una parte esencial de la gestión de proyectos de este campo. Es el proceso de predecir la cantidad de esfuerzo (expresada en términos de puntos de historia) necesaria para desarrollar el producto. Estas estimaciones ayudaron al equipo a prever el tiempo necesario para construir el proyecto.

Establecer una estimación precisa en las etapas tempranas del proyecto hace que sea menos probable que se incumpla el compromiso. A medida que se adquiere más conocimiento, las estimaciones se vuelven más acertadas y útiles.

La técnica que utilizó el equipo se llama estimación relativa y utiliza la secuencia de Fibonacci. Para esta técnica, se le asignó un número (también llamados puntos) de esta secuencia a cada una de las tareas que se necesitó desarrollar. Es relativa porque se basa en el hecho de que una tarea estimada con 2 puntos requerirá menos de la mitad del esfuerzo necesario que una tarea de 5 puntos.

El número de puntos completados por un equipo durante un Sprint se considerará la velocidad del equipo. Es importante mencionar que estos puntos no representan horas, sino esfuerzo estimado en base a la complejidad que se presenta.

6.3.2 Proceso de planificación

Una vez elaborado el Product Backlog, el proceso para ejecutar estimaciones y planificar fue realizado cada dos semanas al finalizar un Sprint. El equipo se reunió para realizar la correspondiente Sprint Planning y así discutir y estimar las tareas relacionadas con la próxima iteración.

El equipo llevó a cabo sesiones de planificación para precisar cuántas y cuáles tareas debían abordarse para el próximo Sprint. Las tareas fueron definidas por el equipo, mientras que el número de tareas fue definido por la velocidad ágil del equipo. Cabe destacar que, mientras la suma de los puntos de todas las tareas agregadas al Sprint sea menor que la velocidad del equipo, se pueden agregar nuevas tareas. Una vez alcanzado este valor ágil, el equipo no podía agregar más tareas, ya que es probable que fuese incapaz de completarlas. En forma de ejemplo, si la velocidad del equipo es de 10 puntos, entonces el número total de puntos de todas las tareas sumadas en un Sprint no puede ser mayor a 10.

Al final de cada Sprint, el equipo debía calcular el número de puntos logrados, debido a que durante el mismo el puntaje podría haber sufrido modificaciones (por ejemplo, al sumar una nueva tarea a mitad de la iteración). Este valor siempre debió ser considerado y la velocidad del equipo ajustada en caso de ser necesario.

La utilización de la velocidad del equipo fue introducida al comenzar la segunda iteración, dado que al comenzar la primera, el equipo aún desconocía la estimación posible y debió determinar cuál sería la duración ideal o posible.

Para poder estimar las tareas/historias, el equipo utilizó una técnica llamada *planning poker* sobre la base de la estimación relativa (Figura 40), la cual cumple el siguiente proceso:

1. Se presenta y explica una tarea al equipo por un miembro del mismo.

2. Cada uno de los miembros del equipo hace una estimación secreta, con el fin de no influir en los demás.

3. Una vez que todos han elegido sus puntos, se revelan las estimaciones y los puntos.

4. Se discuten los valores más altos y más bajos.

5. Al final de la discusión, se lleva a cabo una nueva estimación (ahora con más información que antes).

6. Los puntos 3 y 4 se repiten hasta que el equipo llega a un acuerdo.

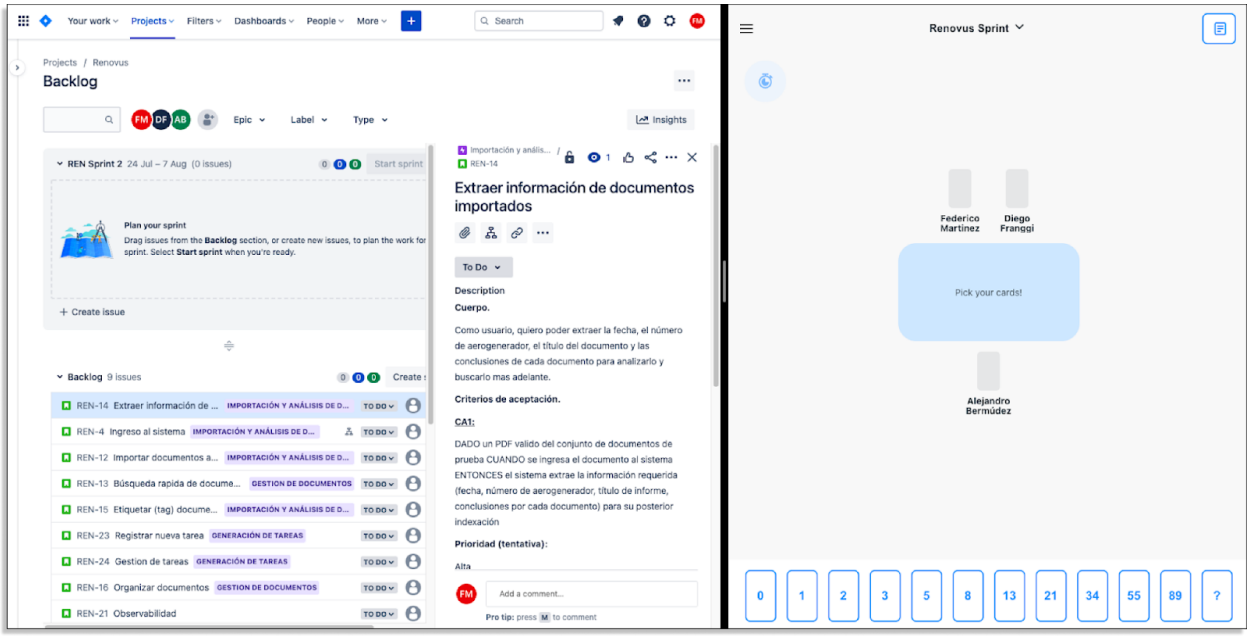


Figura 40: Sprint planning

Una vez finalizada la planificación, el equipo daba comienzo al Sprint, con la libertad de que, en caso de ser necesario, se podían incluir nuevas tareas, las cuales debían estar estimadas para no perjudicar la métrica de velocidad del equipo.

6.4 Ejecución del proyecto

En esta sección se proporciona una descripción detallada de la ejecución del proyecto desde el primer Sprint hasta su finalización. Observaremos el progreso a lo largo de cada iteración, analizando y apreciando detalles referentes a la ejecución de las mismas.

Una vez concluido el último Sprint, se determinó el total de puntos de historia ejecutados en el proyecto. De esta manera, se pudo representar gráficamente la cantidad total de puntos consumidos a lo largo del desarrollo del sistema.

Además, se estableció una guía aproximada del progreso que se debería alcanzar en cada Sprint para completar el proyecto en tiempo y forma. Así pudiendo ir comparando en cada momento si el equipo se encontraba a tiempo en la ejecución de las tareas o si debía redoblar el esfuerzo.

Con toda esta información, se elaboró un *Burndown Chart* que reflejó la ejecución del proyecto de manera clara y concisa. Se puede apreciar gráficamente la ejecución de las tareas distribuidas temporalmente y contrastar así lo esperado contra lo sucedido (real) (Figura 41).

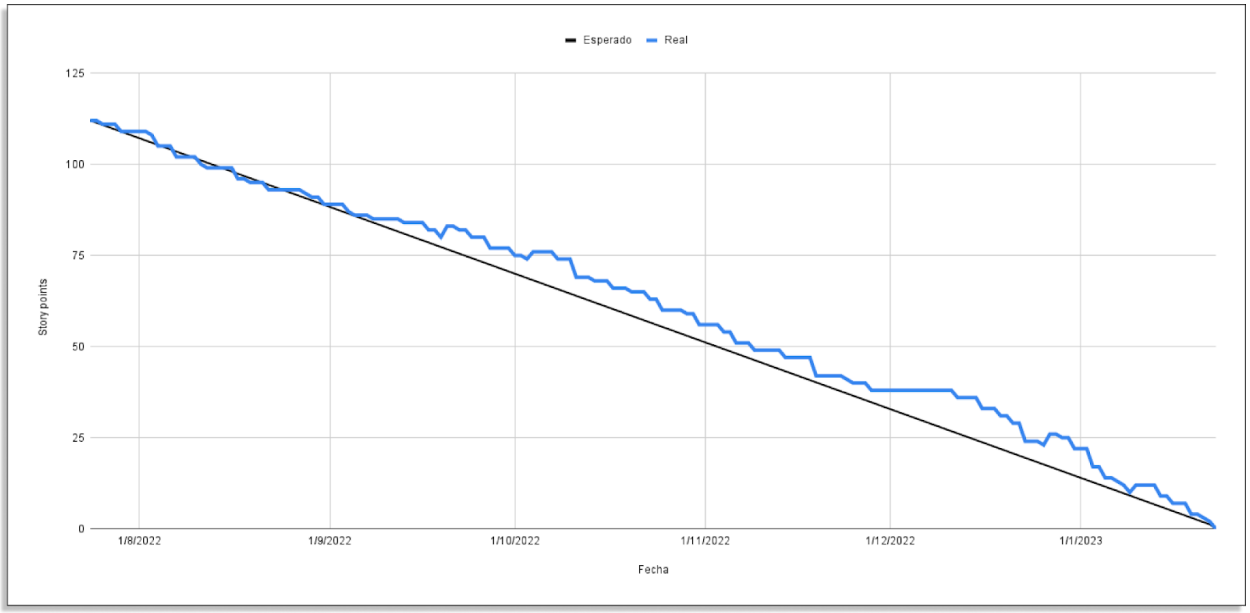


Figura 41: Burndown chart

Como se puede apreciar en la gráfica, el equipo logró llegar en tiempo y forma a la fecha de entrega planificada. Se puede ver que el esfuerzo en algunos puntos fue desproporcionado, existiendo momentos de poco avance y otros de un gran progreso. A continuación se presenta un desglose mensual para entender mejor las causas de estas irregularidades (Tabla 22):

Mes (Sprints)	Resumen
Julio-Agosto Sprint 1 Sprint 2 Sprint 3	Este mes fue el comienzo de la codificación del proyecto, donde si bien había una base teórica aprendida sobre las tecnologías, era momento de comenzar a plasmarla. Como se puede apreciar en la gráfica, el equipo comenzó muy bien, avanzando más rápido de lo esperado. El equipo sabía que era un momento clave, dado que era el mes en el que las tareas externas al proyecto (otras materias de la Universidad y trabajo) no requerían tanto esfuerzo como en otros momentos del año, por ende el equipo duplicó el esfuerzo sabiendo que posiblemente en los próximos meses, el tiempo disponible para realizar tareas iba a reducirse.
Setiembre Sprint 3 Sprint 4 Sprint 5	Setiembre fue el mes en el que se puede comenzar a notar que la línea “real” (azul) comienza a verse por momentos horizontal, lo cual demuestra lapsos de tiempo donde el equipo logró pocos avances, algo que fue separando cada vez más lo esperado de lo real. Esto sucedió en parte por motivo de lo explicado anteriormente, los integrantes del equipo de desarrollo comenzaron a tener otras obligaciones y eso requirió una baja en la cantidad de horas semanales. Pese a esto, siempre se tuvo en cuenta que toda hora de trabajo atrasada debía ser recuperada lo antes posible. Como detalle no menor, también se puede observar a mediados del mes una pequeña elevación, la cual se debe a nuevas tareas (correcciones) agregadas posteriormente al primer *release* del producto.
Octubre Sprint 5	A diferencia del mes anterior, pese a que existieron retrasos, se pueden notar grandes descensos verticales en la gráfica, demostrando por instancias una

Sprint 6 Sprint 7	gran disposición del equipo no tan solo a realizar las tareas, sino también a recuperar las horas perdidas anteriormente.
Noviembre Sprint 8 Sprint 9 Sprint 10	En noviembre, entre comienzos y mediados del mes, se puede apreciar un avance similar al de los meses anteriores, mientras que a finales del mismo se puede notar una gran franja horizontal. Esta corresponde a un lapso de tiempo en el cual el equipo no logró ningún tipo de avance significativo en el sistema. Esto ocurrió debido a la época de entregas obligatorias y evaluaciones de la Universidad. El equipo debió bajar la cantidad de horas dedicadas notoriamente, debido al esfuerzo necesario para realizar las evaluaciones. Este suceso fue previamente discutido con el cliente, al cual se le comunicó la baja de horas y su pronta recuperación en próximas semanas, algo a lo cual accedió sin inconvenientes.
Diciembre Sprint 10 Sprint 11 Sprint 12	Durante los primeros días de este mes se mantuvo la misma tendencia que el mes anterior debido a las evaluaciones de la Universidad. Una vez finalizado el periodo se puede observar en la gráfica un rápido descenso en las tareas, de esta manera el equipo terminó duplicando el esfuerzo para recuperar las horas perdidas.
Enero Sprint 12 Sprint 13	Enero fue el mes de finalización del proyecto, donde el equipo debió dedicar todo el esfuerzo posible a la entrega del producto definitivo. Se puede notar como avanzó notoriamente, pero también se puede ver como la cantidad de tareas aumenta, esto ocurrió debido a cambios solicitados por el cliente en las últimas semanas del proyecto. Pese a esto, el equipo igualmente logró llegar en tiempo y forma a la fecha planificada.

Tabla 22: Desglose mensual

El equipo dedicó una media de 42 horas por semana, algo que como se mencionó anteriormente no fue uniforme, sino que varió dependiendo de la situación del equipo.

Desde el comienzo del proyecto, hasta la última entrega, se pudo apreciar la conformidad del cliente. A lo largo de todo el proceso, mediante las encuestas de satisfacción (sección 6.5), se fue recogiendo evidencia que demostró que pese a no cumplir con las horas dedicadas de forma uniforme, el equipo siempre supo cumplir las entregas solicitadas y se mostró atento y a disposición del cliente.

6.4.1 Ejecución por Sprint

Con el objetivo de alcanzar un alto nivel de eficiencia y organización en el trabajo, el equipo optó por emplear un Scrum Task Board como herramienta para gestionar las tareas durante los Sprints en curso (Figura 42). El tablero se encuentra compuesto por distintas columnas en las cuales se agrupan las historias de usuario y otros tipos de incidentes, que se van desplazando de una columna a otra de acuerdo a su estado (abierta, bloqueada, en progreso o cerrada, etc). De esta manera, el equipo pudo tener un seguimiento diario sencillo del estado de cada tarea, identificar al responsable asignado y conocer el progreso general del Sprint

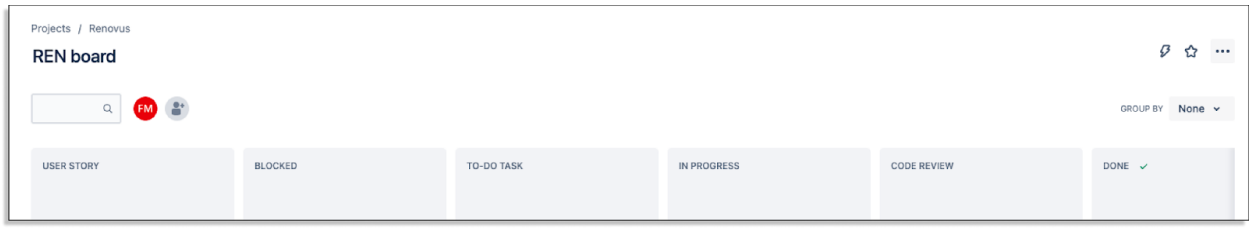


Figura 42: Scrum Task Board en la herramienta Jira

El *board* siempre contempló los siguientes estados:

- **User Story:** Historias de usuario en las cuales el equipo se encuentra trabajando en el Sprint. De las mismas se extraen las tareas que ocupan lugar en las demás columnas.
- **Blocked:** Tareas cuya realización se encuentra bloqueada debido a distintos motivos.
- **To-do Task:** Tareas por comenzar que no fueron realizadas previamente.
- **In Progress:** Tareas en las que se está trabajando.

- **Code Review:** En esta columna se encuentran las tareas que han sido completadas y están listas para ser revisadas. Para que una tarea llegue a esta etapa, es necesario que se haya creado un *pull-request* en el repositorio de GitHub correspondiente, el cual se encuentra disponible para que los integrantes del equipo de desarrollo lo revisen y aprueben.

- **Done:** Tareas aprobadas por el equipo y fusionadas (*merged*) a las ramas de desarrollo de los repositorios de Renovus.

6.4.2 Evaluación por Sprint

Al finalizar cada uno de los Sprints ejecutados, el equipo llevó a cabo los eventos de cierre definidos por Scrum, las Sprint Review y Sprint Retrospective.

Durante las revisiones de Sprint, se llevaron a cabo evaluaciones grupales conjuntas entre todo el equipo de desarrolladores para verificar el progreso de cada iteración y determinar si se lograron alcanzar los objetivos establecidos. Asimismo, se utilizaron estas instancias para generar contenido destinado a la próxima iteración.

Durante todo el proceso de desarrollo, se midió la satisfacción del cliente en varias ocasiones, observando sus reacciones en las reuniones habituales al final de cada Sprint.

Además, se llevaron a cabo las ceremonias de Sprint Retrospective, en las cuales el equipo identificó los aspectos positivos y negativos de cada iteración y estableció acciones específicas a implementar durante la siguiente. Durante estas reuniones, también se evaluaron métricas relevantes para cada iteración, lo que permitió obtener una visión global del desempeño del equipo a lo largo del proyecto.

En el Anexo 10.6 se exponen de manera detallada los periodos, objetivos y resultados más destacados correspondientes a cada Sprint.

6.5 Ejecución de liberaciones

Al comienzo del proyecto y siguiendo la metodología basada en Scrum, el equipo acordó con el cliente realizar 3 *releases* los cuales contemplaban 3 hitos grandes del sistema.

- **Release 1:** Importación de documentos al sistema y la ejecución del análisis y extracción de datos.
- **Release 2:** Vista de documentos, filtrado y detalles de extracción de datos de los mismos.
- **Release 3:** Gestión de tareas.

Los *releases* 1, 2 y 3 fueron planificados para entregarse luego de los Sprints 5, 11 y 13 respectivamente. Como fue mencionado en secciones anteriores, las fechas de entrega se realizaron de manera adecuada y acorde a lo planificado.

Luego de cada *release*, el equipo realizó encuestas de satisfacción al cliente (Anexo 10.7), pudiendo analizar de esta forma su agrado con el trabajo realizado. A continuación se presenta una gráfica donde se puede contemplar la satisfacción promedio del cliente a través de los *releases* (Figura 43).

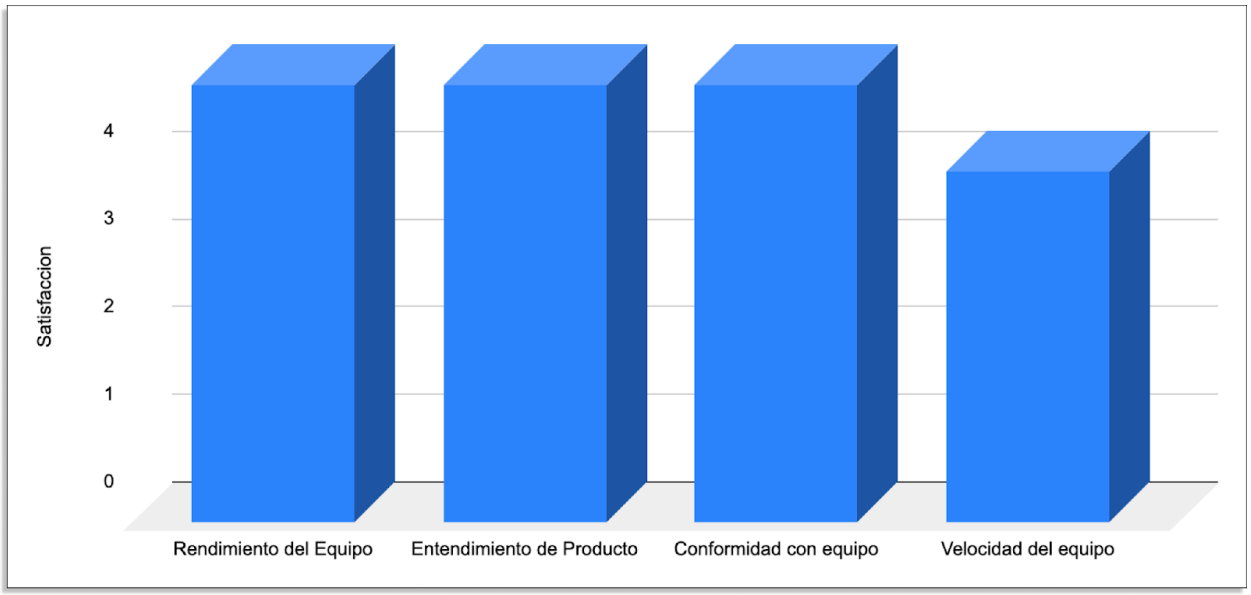


Figura 43: Satisfacción de *releases*.

Como se puede apreciar en la gráfica el equipo logró el puntaje máximo establecido, tanto para el área de rendimiento, como para el de entendimiento del producto y conformidad. Por otro lado, se puede observar un puntaje de 4 sobre 5 en la velocidad del equipo, lo cual pese a no ser

preocupante es atribuido a los altibajos en dedicación que tuvo el equipo durante el proyecto, como se vio en la gráfica de Burndown Chart (Figura 41) y se detalló mediante la Tabla 22.

6.6 Métricas resultantes

6.6.1 Burndown chart

A continuación se podrá ver el Burndown Chart del proyecto, el cual ha sido previamente descrito en la sección 6.4, ejecución del proyecto junto con su correspondiente desglose. La razón por la cual se incluye nuevamente, es debido a que es una de las métricas de gestión más importantes que se han utilizado en el proyecto. El propósito de esta herramienta es representar visualmente el progreso del trabajo del equipo en comparación con el progreso esperado o ideal del proyecto.

En esta sección agregaremos tanto, el desglose mensual (Figura 44) como por Sprint (Figura 45), que si bien simbolizan lo mismo, permite al lector visualizar de mejor manera la totalidad del proyecto.

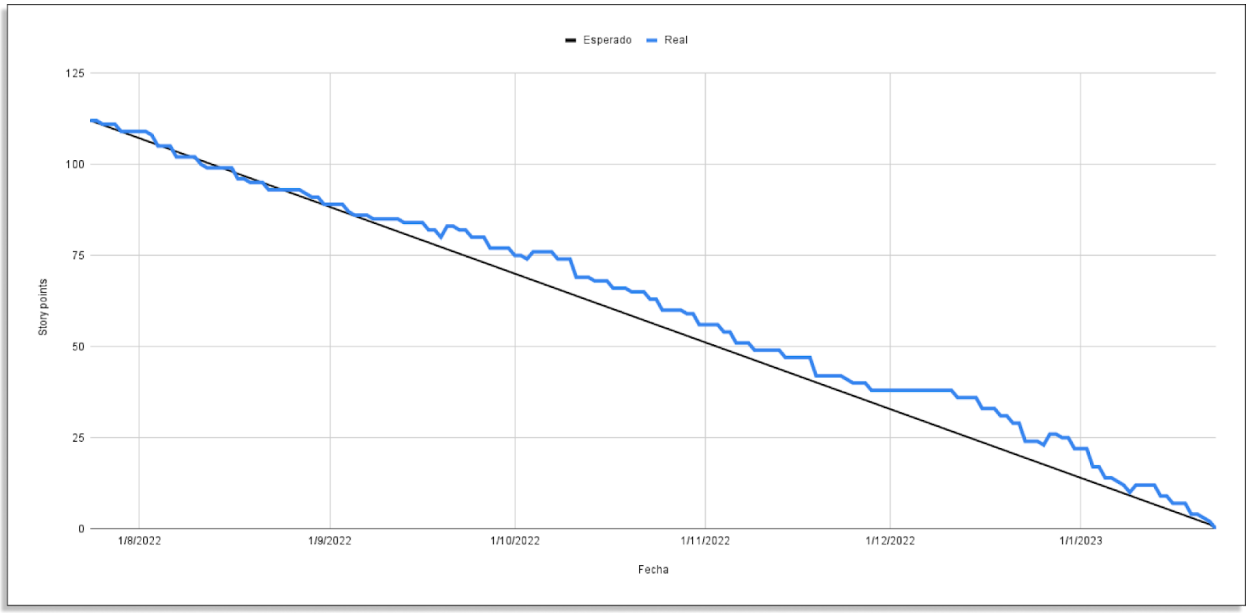


Figura 44: Burndown chart separado mensualmente

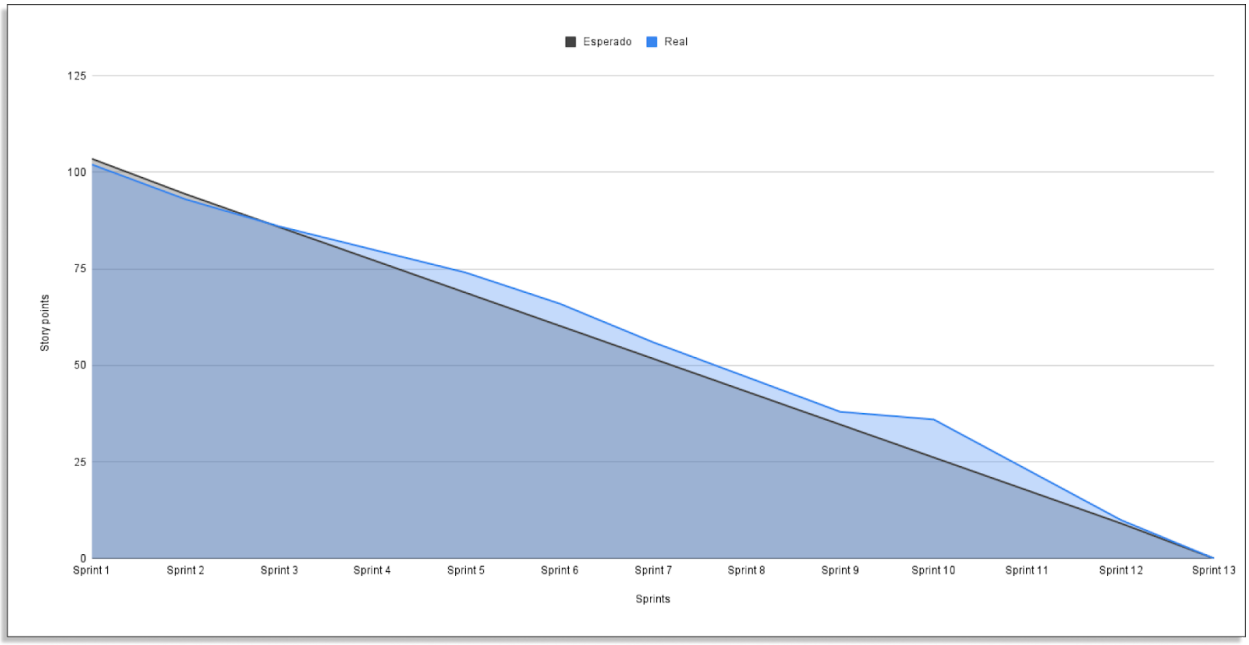


Figura 45: Burndown chart separado por Sprints

6.6.2 Velocidad ágil

A continuación se puede observar el registro de puntos de historia alcanzados por Sprint, el cual permitió al equipo poder analizar la velocidad y de esa forma tomar decisiones al momento de evaluar la cantidad de puntos esperados para cada iteración (Figura 46).

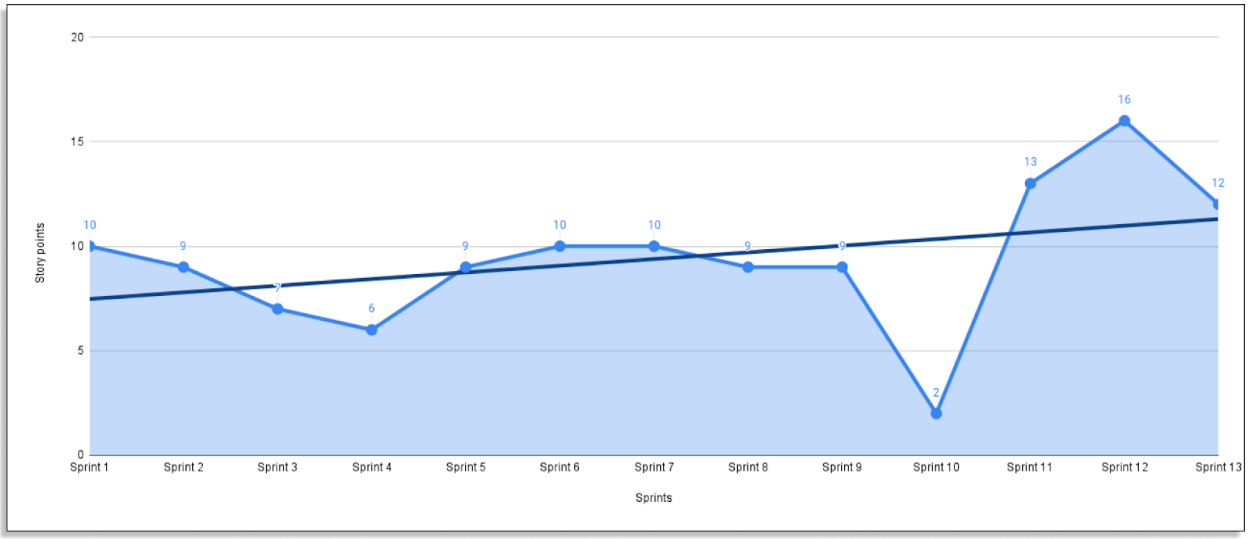


Figura 46: Velocidad del equipo por Sprints

Como se puede observar en la Figura 46 y como fue mencionado en secciones anteriores, la velocidad del equipo fue cambiando gradualmente, pero la misma siempre se mantuvo alrededor de 9-10 puntos, dándonos así un promedio de 9.5 puntos por Sprint.

Otra particularidad que se puede observar en la gráfica es la línea horizontal que la atraviesa y que simboliza la tendencia de los puntos de historia por Sprint y a su vez, demuestra una tendencia del equipo a desarrollar más historias de usuario a medida que progresa el proyecto. Esto se debe al aumento de conocimiento sobre el proyecto del equipo a lo largo de todas las iteraciones. Al comienzo, al no tener mucho conocimiento el avance se tornaba dificultoso, pero a finales del proyecto, al ya conocer el producto y la solución, la realización de cambios e implementaciones se volvió más rápida y eficaz.

6.6.3 Desviación story points estimados / realizados

Durante cada ciclo de desarrollo se llevó un registro de la cantidad de puntos de historia que se lograron completar. Luego de pocas iteraciones, se pudo calcular un promedio y, de este modo, obtener la velocidad del equipo. Esta información resultó de gran utilidad para realizar estimaciones de tiempo y complejidad que se abordaron en cada Sprint durante su planificación (Figura 47 y Tabla 23).

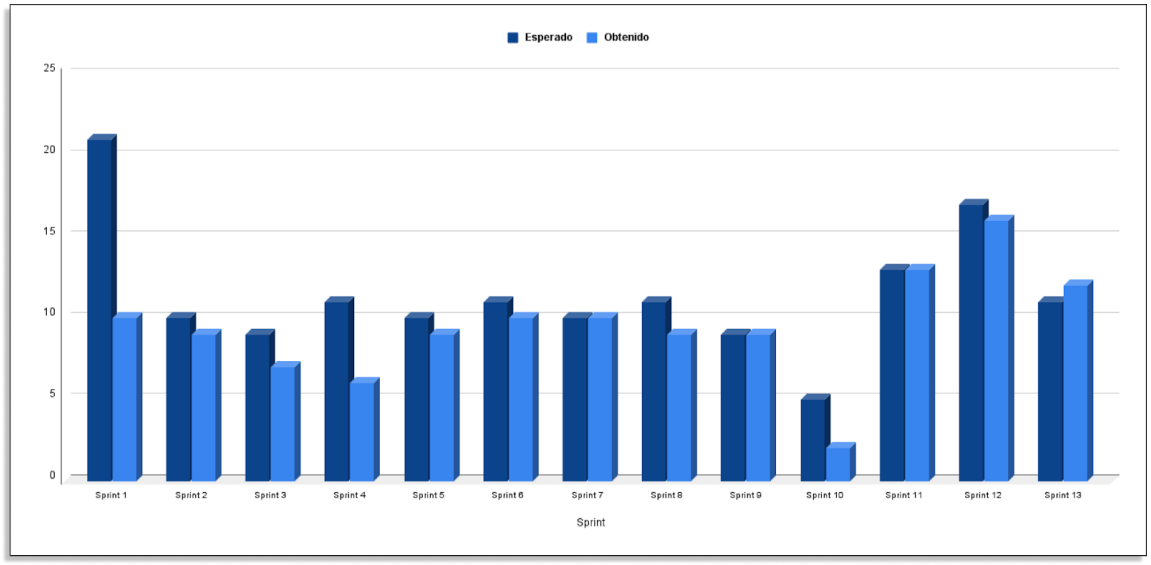


Figura 47: Desviacion story points

Sprint	Esperado	Obtenido
1	21	10
2	10	9
3	9	7
4	11	6
5	10	9
6	11	10
7	10	10
8	11	9
9	9	9
10	5	2
11	13	13
12	17	16
13	11	12

Tabla 23: Desglose desviación story points por Sprint

Tanto en la gráfica como en la tabla se puede apreciar que el primer Sprint tuvo un desempeño considerablemente bajo. Esto se debió, en parte, al limitado conocimiento del sistema por parte del equipo, pero también a que aún no se conocía la velocidad del equipo, que lo llevó a una estimación poco precisa.

Como fue mencionado en secciones anteriores, aquí también se puede observar momentos del proyecto en los cuales el equipo debió bajar la carga horaria y esto se tradujo en una baja de puntos por Sprint. Esto, por ejemplo, se puede observar tanto en el cuarto Sprint como en el décimo, en los cuales debido a obligaciones por fuera del proyecto, el equipo no pudo avanzar como estimaba. Si bien en el caso del cuarto Sprint no fue esperado, se puede notar como en el décimo Sprint hay una baja notoria de los puntos previstos, debido a que ya se sabía las pocas horas de trabajo disponibles que podía llegar a tener el equipo.

Otra particularidad que se puede observar es el gran aumento de puntos por Sprint en la última fase del proyecto (Sprint 11, 12 y 13). Pasado diciembre, el equipo aprovechó el aumento de horas disponibles, provocados por el descenso de obligaciones de sus actividades académicas ajenas al proyecto, traduciéndose en más horas de dedicación y una recuperación notoria de puntos no alcanzados previamente.

Finalmente se puede observar que en el último Sprint se obtuvieron más puntos de los esperados, debido a que a mitad de la iteración, el cliente solicitó pequeños cambios y ya que el equipo se encontraba bien de tiempos, accedió a incorporarlos al Sprint, lo cual implicó más puntos de los estimados inicialmente.

6.6.4 Horas trabajadas

A continuación se presenta un gráfico que muestra la distribución de horas que el equipo dedicó a cada área a lo largo del proyecto. En total, se registraron aproximadamente 2100 horas-hombre de trabajo (Figura 48 y Tabla 24).

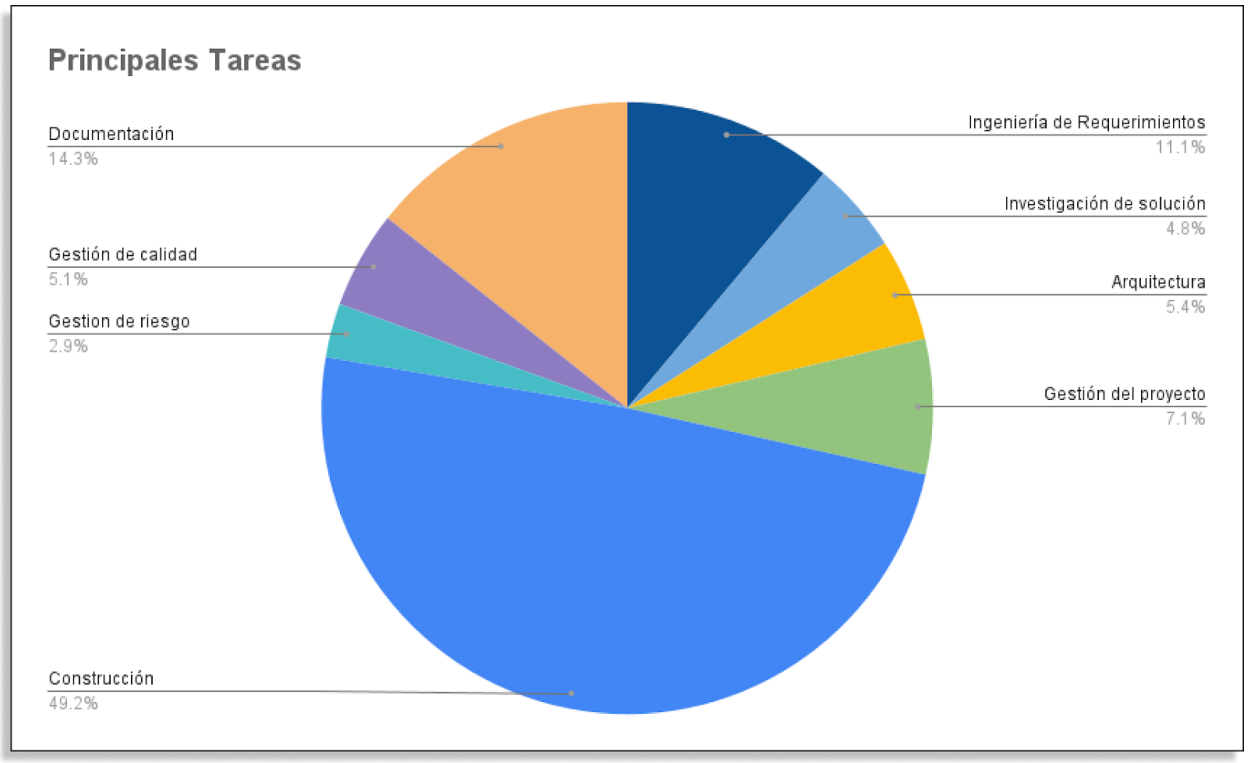


Figura 48: Distribución de horas por áreas

Área	Porcentaje
Ingeniería de Requerimientos	11.1%
Investigación de solución	4.8%
Arquitectura	5.4%
Gestión del proyecto	7.1%
Gestión de riesgo	2.9%
Gestión de calidad	5.1%
Construcción	49.2%
Documentación	14.3%

Tabla 24: Desglose porcentaje de horas por área.

Como se puede observar, la tarea que consumió más tiempo fue la construcción del producto, algo esperado dada la dimensión del mismo.

Por otra parte, tanto la ingeniería de requerimientos como la documentación insumieron un gran porcentaje de tiempo, ambas más del diez por ciento. La explicación yace tanto en la dificultad que ofreció el proyecto, ante la necesidad de una clara explicación de su funcionamiento, como también por lo difícil que fue establecer los requerimientos en conjunto con el cliente, en la etapa inicial.

Luego se destacan las áreas de gestión de proyecto y gestión de calidad. El equipo se esforzó por la entrega de un producto de calidad, dándole un gran énfasis a la gestión del proyecto para poder entregar el producto de manera satisfactoria.

Por último, otras áreas destacables son las de arquitectura e investigación de la solución, las cuales, si bien no implicaron un gran uso de tiempo en comparación con otras áreas, fueron fundamentales para la buena ejecución del producto, dado que proyectaron el rumbo del equipo desde un comienzo.

6.7 Gestión de riesgos

La gestión de riesgos implicó una parte significativa del proyecto, de forma implícita y crucial al ciclo de vida seleccionado. Los riesgos del proyecto fueron gestionados a lo largo de la duración del mismo.

El objetivo central de esta gestión fue poder lograr, en primer lugar, la identificación de riesgos existentes o de aquellos que pudieran ocurrir a futuro y en segundo lugar, actuar sobre los mismos acorde a los planes predefinidos. Al estar trabajando en un entorno ágil, fue posible tomar medidas sobre riesgos materializados durante el transcurso del proyecto, aprovechando los beneficios de estos marcos de trabajo, como por ejemplo, la posibilidad de tomar acciones casi inmediatas en las iteraciones.

Cada dos meses se realizaron análisis de evaluación de riesgos, donde se tomaba en cuenta cuáles existían en el momento, y cuáles eran los más probables de materializarse en el futuro cercano. De haber riesgos adicionales, en el corto plazo se identificaban y se realizaba el estudio correspondiente.

Los riesgos analizados bimensualmente no implicaban la desestimación de aquellos que se materializan en algún momento puntual. En caso de la ocurrencia de un evento imprevisto fuera del marco de gestión de riesgos, se pasaba a agregar el mismo y se procedía a tomar acción.

6.7.1 Marco desarrollado

Para la confección de este marco, se realizaron las siguientes actividades: identificación, categorización, planeación de la gestión y seguimiento del riesgo.

Estas actividades se realizan para cada riesgo identificado y se detallan meticulosamente en una **matriz** específica al proceso del equipo, detallada en la sección 6.7.5, basada en la metodología descrita en el libro Prince2 [35]. A continuación se puntualizan las actividades y se presenta su registro en la matriz, para luego dar lugar a un esquema a modo de tabla, fácil de entender para un lector que desee no incurrir en los anexos digitales donde se encuentran los riesgos en formato de hoja de cálculo (Anexo Digital 1 - Matriz de riesgos).

Actividad 1: Identificación del riesgo

La identificación del riesgo, como su nombre indica, se basa en establecer potenciales riesgos que puedan aparecer en el proceso. Se detalla para la matriz los siguientes campos que representan columnas en la misma:

- Identificador (ID): A fin de poder hacer mención precisa o buscar un riesgo particular sin ambigüedades.
- SI (riesgo identificado): Descripción del riesgo a tratar.
- ENTONCES (descripción del efecto): Descripción del efecto posible del riesgo.
- Fase/Etapa de posible materialización: Establecimiento de la fase/etapa en la cual el riesgo puede potencialmente materializarse. Cabe señalar que pese a haber superado una fase potencial, el riesgo ha de seguir considerándose.

Actividad 2: Categorización del riesgo

A fin de trabajar con los riesgos identificados se determinan los siguientes campos para cada uno:

- Fuente: Marca el origen del riesgo, ya sea interno o externo, al equipo.
- Categoría: Se detallan categorías particulares para los posibles riesgos. Las identificadas para el proyecto son las siguientes (Tabla 25):

Categoría de los Riesgos	Relación
Producto	Estabilidad del producto, arquitectura del producto, disponibilidad del producto (versionamiento), ambientes disponibles.
Gestión de proyecto	Negociación, planeación, contratación, seguimiento de proyectos, estimación, costos.
Cliente	Incumplimiento de compromisos, poca dedicación de tiempo al proyecto, falta de compromiso, desconocimiento del propio negocio.

Tecnología	Interfaces, *hardware*, herramientas de trabajo, recursos vinculados a la informática en general.
Humano	Incapacidades, traslados, renuncias, recursos no disponibles, falta de compromiso.
Capacitación/*Skills*	Falta de conocimiento/habilidades, capacitación inadecuada.
Desarrollo	Vinculado al equipo de desarrollo.
QA	Vinculado a estándares de calidad establecidos por el equipo.

Tabla 25: Relación de categorías de los riesgos

- Probabilidad de ocurrencia

Se considerarán los siguientes valores acordes a los valores estimados por el equipo (Tabla 26):

Probabilidad	Descripción
10%	Muy bajo
30%	Bajo
50%	Medio
70%	Alto
90%	Muy alto

Tabla 26: Probabilidades de ocurrencia y su descripción

- Impacto

Se considera a continuación la siguiente escala para el impacto potencial de la materialización de un riesgo (Tabla 27).

Impacto	Descripción	Valor numérico
MB	Muy bajo	1
B	Bajo	2

M	Medio	3
A	Alto	4
MA	Muy alto	5

Tabla 27: Descripción de impactos

- Prioridad

Métrica resultante del cálculo de Probabilidad x Impacto (Figura 49).

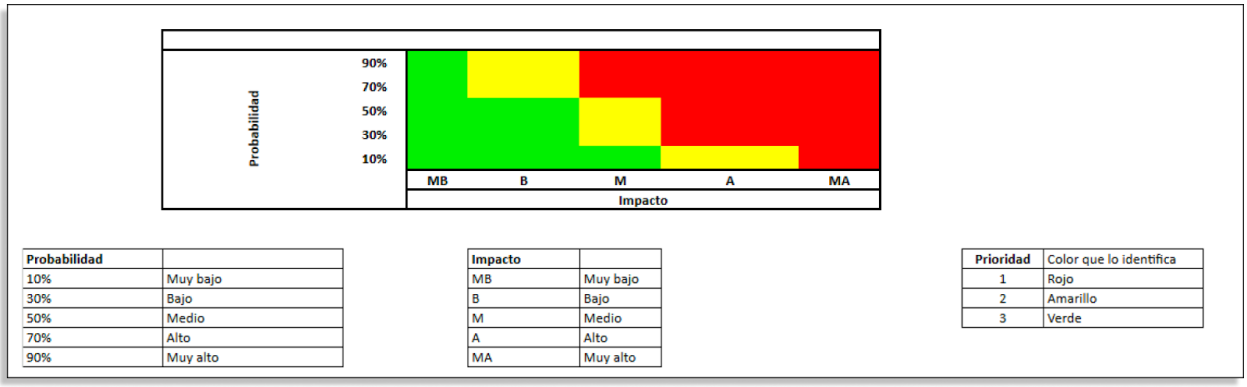


Figura 49: Tabla vacía de visualización de riesgos

La prioridad junto con la gráfica presentada precedentemente, y por consiguiente, el impacto junto a la probabilidad son indicadores que varían a lo largo del ciclo de vida del proyecto. Estas representaciones son determinadas para un punto en el tiempo.

Actividad 3: Planeación de la gestión del riesgo

- Estrategia principal

Para cada riesgo se define una estrategia a seguir como forma de tomar acciones sobre cada uno. Son categorizadas en positivas y negativas acorde a los resultados que puedan desprenderse de éstas. El hecho de tomar una estrategia no implica que no se puedan incorporar aspectos de otras a la hora de tomar acciones.

Pese a haber sido definidas y tomadas en consideración fue difícil para el equipo encontrar riesgos positivos para un proyecto cuyo resultado final sea un MVP, por lo tanto no se detalla ningún riesgo que siga esa estrategia, pero se destacan en la tabla a continuación junto a las "estrategias negativas" (Tabla 28).

Estrategia	Tipos de estrategias	Descripción
Negativos	N - Investigar	Cuando no hay suficiente información para definir una estrategia se ha de recabar información.
	N- Prevenir	Reducir la probabilidad y/o el impacto del riesgo hasta un umbral aceptable. Se debe actuar de manera temprana, antes de que el riesgo se materialice.
	N - Mitigar	Neutralizar o disminuir el efecto inmediato que genera la materialización de un riesgo, con el fin de evitarle a la compañía mayores pérdidas.
	N - Transferir	Involucrar a un tercero en su manejo, quien en algunas ocasiones puede absorber parte de las pérdidas ocasionadas por la ocurrencia e incluso responsabilizarse de la aplicación de las medidas de control para reducirlo. La entidad o persona que recibe el riesgo es la responsable de gestionarlo. Por ejemplo, seguros, garantías, cláusulas de contratos, etc.
	N - Aceptar	No es necesario desarrollar medidas adicionales de prevención o mitigación; porque su evaluación, desde el punto de vista de probabilidad de ocurrencia y de impacto, da como resultado un riesgo poco representativo, es decir, que su ocurrencia no tendría un efecto significativo en la estabilidad del proyecto.
	N - Evitar	Hacer todo lo necesario para que el riesgo NO se presente
	N - Observar	No hacer nada. Mantener en observación por si aumenta

		la probabilidad o el impacto
Positivos	P - Investigar	Cuando no hay suficiente información para definir una estrategia
	P - Explotar	Aumentar la probabilidad de que ocurra con todo lo que se pueda para que se logre el 100% de la oportunidad
	P - Compartir	Hacer *joint ventures*, asociaciones para poder enfrentar la situación
	P - Aumentar	Hacer lo posible para que se de la oportunidad
	P - Aceptar	No es necesario desarrollar medidas adicionales de prevención o mitigación; porque su evaluación, desde el punto de vista de probabilidad de ocurrencia y de impacto, da como resultado un riesgo poco representativo, es decir, que su ocurrencia no tendría un efecto significativo en la estabilidad de la empresa.
	P - Observar	No hacer nada. Mantener en observación por si aumenta la probabilidad o el impacto

Tabla 28: Estrategias principales

- Plan de mitigación y plan de contingencia

Para cada riesgo se determinan acciones de mitigación (disminución de la probabilidad del riesgo) y de contingencia (disminución del impacto del riesgo).

Actividad 4: Seguimiento del riesgo

- Seguimiento: de corresponder, se marca el estado anterior del riesgo obtenido en la última instancia de evaluación.
- Estado: tres posibilidades: materializado, cerrado (riesgo sin posibilidad de ocurrencia) y abierto (posibilidad de ocurrencia).

- Lecciones aprendidas: que se ha aprendido en el transcurso del proyecto sobre el riesgo en cuestión. No necesariamente implica que el riesgo se haya materializado.

Muestra de riesgos identificados

Como se mencionó anteriormente, los riesgos fueron identificados y detallados en una matriz que consta de las mismas columnas descritas en la sección anterior. Sin embargo, para mejorar la legibilidad de este documento, se opta por presentar los riesgos en una tabla que pueda ser fácilmente leída en papel.

En caso de necesitar más información acerca de un riesgo en particular, se puede trazar una conexión entre el identificador del riesgo marcado en la tabla de los riesgos identificados y el de la matriz original. Es importante destacar que, para mantener la claridad, algunos campos pueden variar ligeramente en su contenido, especialmente en la descripción donde se mezclan los campos "SI" y "ENTONCES".

Se presenta a continuación una tabla con los principales riesgos identificados hasta el final del proyecto (Tabla 29). La recopilación final se ubica en el Anexo 10.8.

ID	Descripción	Categoría	Estrategia Principal	Plan de mitigación	Plan de contingencia
R1	Estimaciones muy ambiciosas que derivan en falta de completitud en el desarrollo del producto o incumplimiento con fechas estipuladas	Gestión de proyecto	N-Prevenir	Realizar revisiones periódicas de las estimaciones y buscar re-estimar aquellas que no continúen siendo factibles. Establecer márgenes de tiempos adicionales en los cronogramas de proyecto para acomodar cambios. Realizar seguimiento periódico de los progresos y hacer	Establecer una línea de comunicación con dirección de Renovus para informarle de los cambios en las estimaciones o el cronograma. Considerar la posibilidad de aumentar el plazo de una historia o solicitar apoyo de expertos de Renovus.

				ajustes en función de los mismos.	
R2	Desconocimiento de tecnologías que desembocan en errores de desarrollo o incumplimiento de fechas establecidas.	Capacitación/*Skills*	N- Prevenir	Realización de cursos pertinentes y pruebas de concepto. Tener reuniones con expertos de Renovus. Realizar consultas en foros de expertos.	Utilizar días de licencia o estudio con el fin de obtener más tiempo del previsto para la capacitación y corregir los errores ocurridos.
R3	Enfermedad de alguno de los miembros del equipo que provoca retrasos en las tareas establecidas para el/los Sprint/s en que el miembro esté de baja o no pueda rendir en su plenitud.	Humano	N - Aceptar	Cada integrante ha de conocer el progreso del otro, particularmente a la hora de hacer revisiones de código. Esto implica que todos los integrantes conocen las áreas donde se ha de recuperar trabajo en futuras iteraciones. Se mantienen reuniones en modalidad virtual a fin de evitar contagios de todo el equipo.	Distribuir las tareas que se tendrían que hacer por el integrante enfermo. Ajustar tiempo y alcance en consecuencia. De ser necesario redoblar esfuerzo por parte del integrante cuando se recupere.
R4	Conflictos interpersonales entre los compañeros del equipo que llevan a bajo rendimiento del equipo afectando el cumplimiento de el/los Sprint/s durante los cuales persisten el/los	Humano	N-Evitar	Los miembros se comprometen a interactuar de forma educada y ordenada, respetando las opiniones del resto y favoreciendo la convivencia.	De materializarse el riesgo se toma acción mediante la utilización y aplicación del Método de Negociación de Harvard [36] para el que el equipo realizó una materia electiva en la Universidad ORT a

	conflicto/s				fin de poder renegociar los términos de trabajo entre las partes afectadas.
R5	Requerimientos existentes cambiantes por parte del cliente (no tienen claro lo que quieren) y conlleva a especificaciones de requerimientos erróneas y pérdida de tiempo.	Gestión de proyecto	N-Prevenir	Se realizan validaciones propias del proceso. Se validan con Product Owner las historias, se realizan prototipos a confirmar cambios.	De haberse implementado cierto aspecto se toma en cuenta la aprobación previa del cliente como justificación al ajuste de tiempo a realizar. Se ajusta el cronograma.
R6	Requerimientos existentes cambiantes por parte del cliente (cambios en la necesidad del negocio).	Cliente	N-Mitigar	Se busca llevar la discusión de los requerimientos a largo plazo con el cliente a fin de mitigar cualquier posible cambio predecible en el negocio.	Se vuelve a realizar el análisis del requerimiento afectado y se planifica el cambio.
R7	Problemas de comunicación con el cliente.	Cliente	N-Prevenir	Se buscan formas de comunicación con el cliente que se adecuen a sus horarios y tiempos. Se reafirma a modo de recordatorio que tienen un compromiso formal con el equipo en caso de no .	Se realiza un ajuste de los requerimientos acorde al entendimiento del negocio por el equipo. Se insiste al cliente hasta lograr su aparición.
R8	Demoras del cliente para brindar recursos/ambientes que demoren las	Cliente	N-Mitigar	Solicitar reiteradas veces los ambientes al responsable de	Levantar el ambiente en un entorno del equipo estudiante. Asumir

	pruebas en entornos reales			tecnología. Mencionar el tema en las reuniones.	los costos del mismo.
R9	Perdida de informacion causa de robo o falla informática.	Tecnología	N-Evitar	A medida que se avance en el desarrollo se ha de garantizar la persistencia en la nube de cada avance.	Se reemplaza la información perdida lo antes posible. Se repone el recurso dañado lo antes posible. En caso de no contar con el capital solicitar al cliente o a la Universidad.
R10	El proceso de integración de *software* a solución actual no se da de forma exitosa y resulta en interferencias accidentales con las operaciones de Renovus, retrasando el proyecto.	Tecnología	N-Prevenir	Definir bien la gestión de cambios y coordinación con el equipo cliente. Mantener una comunicación abierta con ellos sobre acciones a tomar.	Detectar fallas junto a la integración de cambios y actuar sobre los mismos.

Tabla 29: Riesgos identificados

6.7.2 Control y monitoreo

Las etapas mencionadas de control y monitoreo se realizan de forma iterativa cada dos meses, en una instancia definida puntualmente para esta tarea. Es aquí donde se revisan los planes generados y se vuelven a calcular la probabilidad y el impacto de cada riesgo para el momento puntual del proyecto.

Es importante destacar que a fin de respetar la metodología ágil el equipo opta por no esperar a estas instancias para tomar acciones y busca que los planes sean aplicables en el momento en que surja la manifestación del riesgo, en el caso del plan de contingencia.

6.7.3 Evolución de riesgos

De los riesgos presentados en la sección anterior y el Anexo 10.8 se seleccionan cinco para que el lector visualice la evolución de los mismos a lo largo del proyecto. Para realizar la proyección, se detalla la prioridad, siendo ésta el potencial impacto del riesgo multiplicado por probabilidad de ocurrencia, que se encuentra en el eje de las ordenadas. Por su parte, el tiempo en el que se realizó la revisión se ubica en el eje de las abscisas. Los riesgos seleccionados se presentan a continuación (Tabla 30) y la evolución de los mismos (Figura 50) y se detalla más ampliamente en el Anexo 10.9.

Identificador	Descripción
R1	Estimaciones muy ambiciosas que derivan en falta de completitud en el desarrollo del producto o incumplimiento con fechas estipuladas
R2	Desconocimiento de tecnologías que desembocan en errores de desarrollo o incumplimiento de fechas establecidas.
R13	Atraso en desarrollo que signifique un incumplimiento en tiempos acordados con cliente
R5	Requerimientos existentes cambiantes por parte del cliente (no tienen claro lo que quieren) y conlleva a especificaciones de requerimientos erróneas y pérdida de tiempo.
R7	Problemas de comunicación con el cliente.

Tabla 30: Riesgos seleccionados para visualizar su evolución

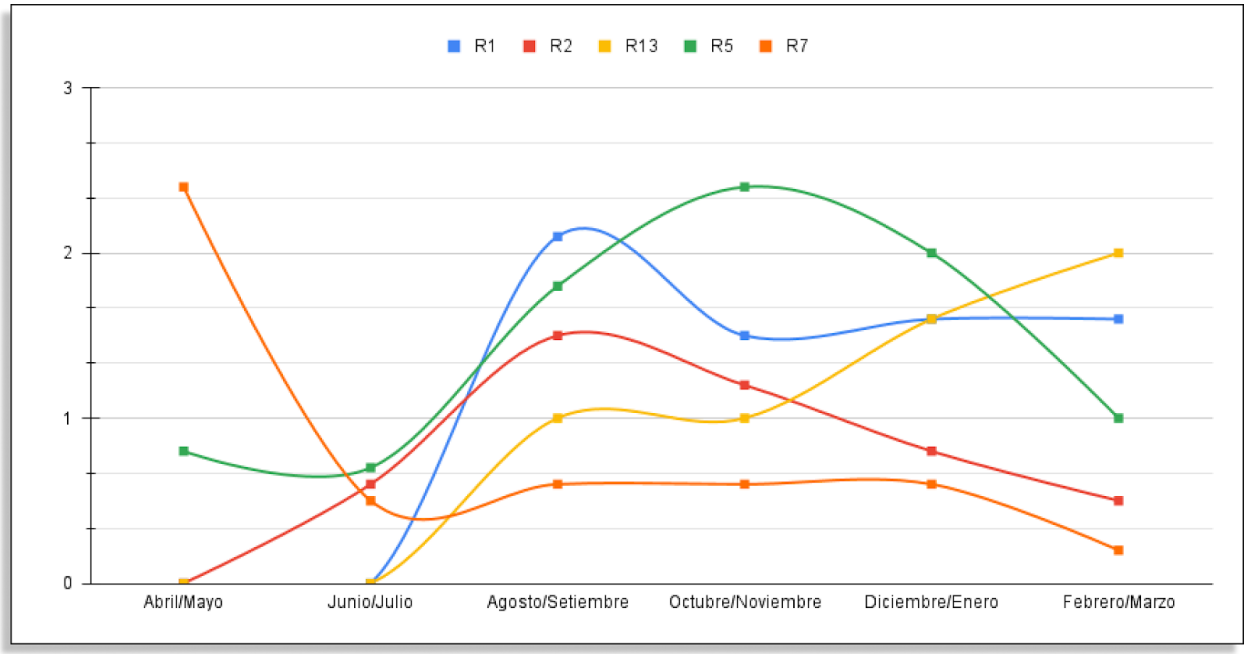


Figura 50: Gráfica de evolución de riesgos

- **R1:** Al comienzo las estimaciones ambiciosas son esperables, el equipo no está familiarizado con su propia velocidad, por consecuencia el impacto es bajo, a medida que avanza el proyecto la probabilidad del riesgo disminuye, pero el impacto de una mala estimación tardía es mucho más perjudicial.

- **R2:** Se observa una tendencia similar en cuanto a los efectos del desconocimiento de tecnologías, que aumenta al principio y luego disminuye. Esto se debe a que al empezar a codificar y diseñar la arquitectura, los errores relacionados con la falta de conocimiento son más comunes pero de menor impacto. A medida que avanza el proyecto, se espera que el conocimiento de la tecnología aumente y no sean comunes los imprevistos en el desarrollo.

- **R13:** A medida que progresa el desarrollo un atraso se convierte en un problema cada vez mayor debido al tiempo y esfuerzo que insume realizar un ajuste sobre la marcha.

- **R5:** Durante un proyecto, es común que los requerimientos sufran cambios. Este pico de cambios suele ocurrir a mitad del proyecto, debido a que es en ese momento cuando se tiene mayor probabilidad de visualizar y manipular el producto con las propias manos y

es aquí donde se pueden identificar diferencias entre lo esperado y lo que se ha desarrollado, lo que puede generar un impacto negativo en el calendario del proyecto.

- **R7:** Los problemas de comunicación pueden resultar especialmente perjudiciales al inicio de un proyecto, ya que es en ese momento cuando la presencia constante de los involucrados es crucial. Son ellos quienes validan los requerimientos y aprueban los cambios, por lo que la falta de comunicación puede tener un impacto negativo significativo en el proyecto.

6.7.4 Riesgos materializados

Durante el proyecto muchos riesgos se materializaron o alcanzaron puntos donde se debió hacer foco en los planes de mitigación a fin de evitar que estos ocurran.

Uno de los riesgos tempranos que ocurrieron fue establecer estimaciones demasiado ambiciosas (R1). Debido a que el equipo a cargo de la estimación era inexperto, se esperaba que pudieran cometer errores en la etapa inicial. Esta situación tuvo un impacto ligero al comienzo, lo que obligó a volver a estimar las historias y a realizar ajustes en el alcance y el cronograma del proyecto. Más adelante, sobre el tercer *release* el equipo vuelve a enfrentarse al problema y lo resuelve dedicando horas adicionales al proyecto para llegar a tiempo.

En las etapas tempranas, el cliente demoró en brindar los ambientes (R8) de AWS requeridos para comenzar a alojar los componentes que se iban construyendo en los mismos. Esto implicó que se tuviese que poner en marcha el plan de contingencia llevando al equipo a trabajar temporalmente con ambientes propios a fin de no retrasar el proyecto.

El desconocimiento de las tecnologías (R2) tuvo su impacto en la etapa de construcción, donde el equipo sobreestimaba la facilidad con la que se podrían utilizar ciertas herramientas de AWS. Particularmente se presentaron problemas al intentar conectar las funciones Lambda al Textract y Comprehend. La base de datos OpenSearch tuvo sus complicaciones imprevistas a causa del desconocimiento de este entorno y de su lenguaje de consultas.

Durante el transcurso del proyecto, otro de los riesgos que se presentaron fueron los cambios en los requerimientos existentes (R5). En la etapa final del proyecto, se realizó un ligero cambio en

la cantidad de documentos que podían vincularse a una tarea del sistema, lo que provocó retrasos temporales. Sin embargo, el equipo logró solucionar este riesgo mediante la duplicación de esfuerzos.

Se presentaron otros riesgos a lo largo del proyecto, como la enfermedad (R3) de algunos de los miembros del equipo en momentos particulares, la pérdida de información necesaria para el resto del equipo debido al robo de un equipo informático (R9) que imposibilitaron a un integrante a programar temporalmente y errores relacionados con la integración del *frontend* con el sistema existente (R10). Sin embargo, estos riesgos fueron solventados rápidamente, impactando levemente en el proyecto.

6.7.5 Matriz realizada

Como se mencionó anteriormente, el equipo definió una matriz que forma la base del marco de gestión de riesgos del proyecto. La misma funciona como un apoyo visual que permite identificar rápidamente los riesgos y examinar los planes de mitigación y contingencia en las instancias de control mencionadas anteriormente.

El artefacto está definido por el equipo y basado en la metodología Prince2 [35], un marco de gestión de proyectos ampliamente utilizado en la industria. A lo largo del proyecto, la matriz evolucionó y se actualizó para convertirse en la piedra angular del marco de gestión de riesgos propuesto.

El detalle completo de la matriz puede ser encontrado como Anexo Digital 1 - Matriz de Riesgos, junto a la entrega de este documento.

A continuación se muestra la matriz para los primeros cinco riesgos (Figura 51), en las actividades de identificación y categorización.

IDENTIFICACIÓN DEL RIESGO				CATEGORIZACIÓN DEL RIESGO					
ID	SI (descripción del riesgo, cuando)	ENTONCES (descripción del efecto)	Evento de posible materialización / Fase	Fuente	Categoría	Probabilidad de ocurrencia	Impacto	Prioridad (Prob x Impacto)	Estrategia Principal
R1	Estimaciones muy ambiciosas	Falta de completitud en el desarrollo del producto, o incumplimiento con las fechas establecidas	Construccion	Interno	Gestión de proyecto	30%	Alto	1	N- Prevenir
R2	Desconocimiento de las tecnologías	Errores en el desarrollo del producto, o incumplimiento con las fechas establecidas	Inicial / Construccion	Interno	Capacitación/Skills	10%	Muy alto	1	N- Prevenir
R3	Enfermedad de alguno de los miembros del equipo	Retrasos en las tareas establecidas para el/los sprint/s en que el miembro este de baja o no pueda rendir en su plenitud	Todo el proyecto	Externo	Humano	50%	Medio	2	N - Aceptar
R4	Conflictos interpersonales entre los compañeros del equipo	Bajo rendimiento del equipo afectando el cumplimiento de el/los sprint/s durante los cuales persistan el/los conflicto/s	Todo el proyecto	Interno	Humano	10%	Medio	3	N - Evitar
R5	Requerimientos existentes cambiantes por parte del cliente (no tienen claro lo que quieren)	Se pueden realizar especificaciones de requerimientos erroneas y por consecuente perder tiempo.	Construccion	Externo	Gestión de proyecto	30%	Muy alto	1	N- Prevenir

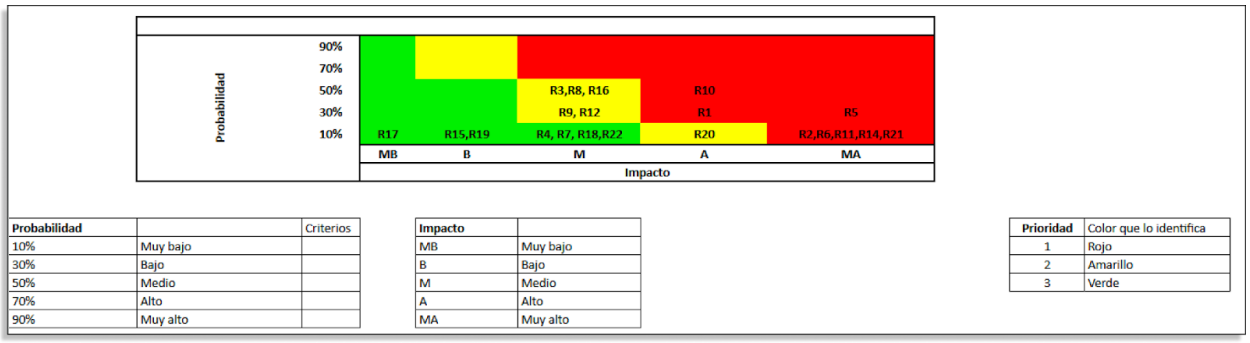
Figura 51: Matriz de riesgos, última evaluación - Identificación y categorización

Luego la matriz continua (Figura 52), marcando la planeación mencionada y el seguimiento a cada riesgo. Se puede apreciar que la primera fila corresponde al R1, la segunda al R2 y así hasta el R5.

PLANEACIÓN DE LA GESTIÓN DEL RIESGO		SEGUIMIENTO DEL RIESGO		
Plan de mitigacion (diminuir la PROBABILIDAD del riesgo)	Plan de contingencia (diminuir el IMPACTO del riesgo)	Seguimiento (ultimo estado)	Estado	Lecciones Aprendidas
Realizar revisiones periódicas de las estimaciones y buscar re-estimar aquellas que no continúen siendo factibles. Establecer márgenes de tiempos adicionales en los cronogramas de proyecto para acomodar cambios. Realizar seguimiento periodico de los progresos y hacer ajustes en función de los mismos.	Establecer una linea de comunicacion con direccion de Renovus para informarle de los cambios en las estimaciones o el cronograma. Considerar la posibilidad de aumentar el plazo de una historia o solicitar apoyo de expertos de Renovus.	Materializado	Cerrado	Al trabjar con tecnologias Cloud se ha de tener mucho cuidado con suponer que la integracion al sistema es sencilla. Muchas veces las tareas vinculadas a la interaccion de componentes deberian de ocupar el doble de SP de los supuestos.
Realización de cursos pertinentes y pruebas de concepto. Tener reuniones con expertos de Renovus. Realizar consultas en foros de expertos.	Utilizar días de licencia o estudio con el fin de obtener más tiempo del previsto para la capacitación y corregir los errores ocurridos.	Abierto	Cerrado	Muchas veces la mejor fuente de informacion para las tecnologias cloud resultan ser cursos introductorios gratuitos en Youtube. La ruta de estudio para AWS Certified Cloud Practitioner es la mejor para comprender las bases.
Cada integrante ha de conocer el progreso del otro, particularmente a la hora de hacer revisiones de código. Esto implica que todos los integrantes conocen las áreas donde se ha de recuperar trabajo en futuras iteraciones.	Distribuir las tareas que se tendrían que hacer por el integrante enfermo. Ajustar tiempo y alcance en consecuencia. De ser necesario redoblar esfuerzo por parte del integrante cuando se recupere.	Abierto	Cerrado	N/A
Los miembros se comprometen a interactuar de forma educada y ordenada, respetando las opiniones del resto y favoreciendo la convivencia.	De materializarse el riesgo se toma acción mediante la utilización y aplicación del Método de Negociación de Harvard a fin de poder renegociar los términos de trabajo entre las partes afectadas.	Abierto	Cerrado	Recordar antes de decir algo potencialmente ofensivo que el interes de todos los integrantes es el cumplimiento de todos los objetivos planteados al comienzo del proyecto. Es importante recordar lo logrado como equipo y lo lejos que se llegó junto a ellos.
Se realizan validaciones propias del proceso. Se validan con PO las historias, se realizan prototipos a confirmar cambios.	De haberse implementado cierto aspecto se toma en cuenta la aprobación previa del cliente como justificación al ajuste de tiempo a realizar. Se ajusta el cronograma.	Abierto	Cerrado	Uso de Design Thinking con foco en prototipacion resulta muy efectivo

Figura 52: Continuación de la matriz para los primeros 5 riesgos

Con los riesgos definidos se marca cada uno en su correspondiente coordenada basado en la probabilidad y el impacto a fin de poder visualizar rápidamente todos. (Figura 53)



Probabilidad		Criterios
10%	Muy bajo	
30%	Bajo	
50%	Medio	
70%	Alto	
90%	Muy alto	

Impacto	
MB	Muy bajo
B	Bajo
M	Medio
A	Alto
MA	Muy alto

Prioridad	Color que lo identifica
1	Rojo
2	Amarillo
3	Verde

Figura 53: Categorización al final del proyecto

6.8 Gestión de la configuración

Seguidamente se detalla la gestión de configuración llevada a cabo en el proyecto, haciendo hincapié en la identificación de elementos tanto a nivel de *software* como de documentación. También se describirán las tácticas implementadas para controlar los cambios y se explicará el flujo de trabajo establecido para integrar dichos cambios con el equipo de desarrollo de Renovus a la solución en uso. Cabe destacar que la integración de cambios resultó fundamental para garantizar la calidad del producto final entregado.

6.8.1 Elementos de configuración

Los elementos de configuración fueron separados en dos categorías, *software* y documentación. Los considerados son aquellos que se utilizaron durante la implementación del proyecto.

Software:

- Código FalconeerUI *web* (React.js).
- Código DocumentGateway (.NET).
- AWS (Textract, Comprehend, S3, OpenSearch, Lambda, Cloud Watch, SNS, SQS).
- Especificación Swagger endpoints REST API.

- Librerías externas.

Documentación:

- Documentación académica del proyecto.
- Informes académicos de avance.
- Documentos *README* en repositorios.
- Plan de Calidad.
- Plan de Riesgos.
- Product Backlog.
- Documento de Arquitectura.
- *Mockups* y prototipos.

6.8.2 Organización de repositorios

Software

Como sistema de control de versiones se utilizó Git apoyado por la plataforma GitHub [37] para alojar el mismo.

Al ser un proyecto cuya salida a producción no era requerida por el cliente, se generó una organización dentro de la plataforma llamada "renovus-ort" para gestionar los repositorios dedicados al proyecto. La misma se encontraba compartida con los desarrolladores y expertos de Renovus y estaba compuesta de 4 repositorios, siendo 3 de ellos creados desde cero y uno de ellos (renovus_web) copiado (*forked*) de su repositorio *web* en producción. El motivo de esto fue pensado no solo para permitir una mayor flexibilidad en el momento de realizar el proyecto, sino también para que, en caso de que el cliente quisiera integrar el código en sus repositorios, pudiera hacerlo fácilmente utilizando las herramientas que proporciona GitHub (Figura 54).

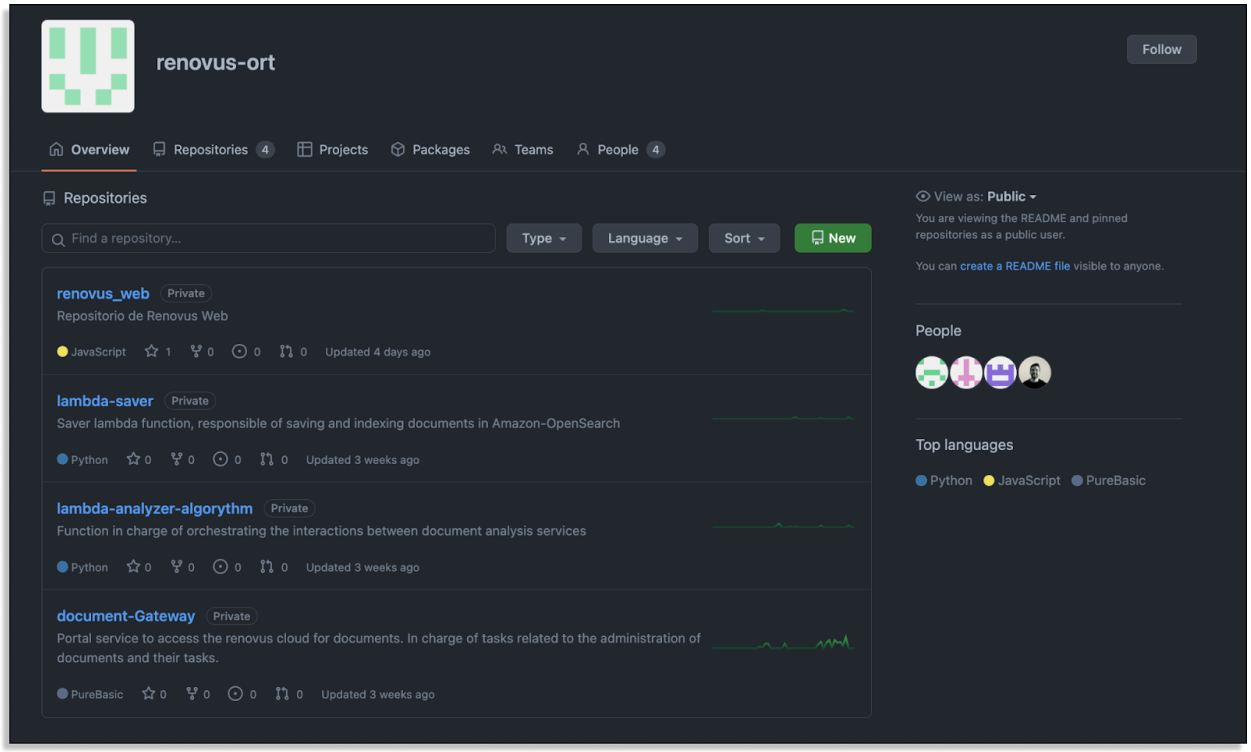


Figura 54: Organización renovus-ort y repositorios en GitHub

Branching

En cada uno de los repositorios, se empleó la técnica de *branching* GitFlow [38] para el trabajo colaborativo. La rama principal, denominada main, fue la encargada de contener la versión estable más reciente del sistema en producción. Para desarrollar cada nueva funcionalidad, se crearon ramas a partir de la rama develop, rama encargada de contener todo el código en desarrollo previo a estar en producción. Estas ramas se nombraron con el formato feature/REN-XX, donde REN correspondió al código identificador del proyecto en Jira y XX el identificador del ticket autogenerado por Jira. Una vez completado el desarrollo de una funcionalidad y realizadas las pruebas necesarias, se enviaba un PR (*pull request*) hacia la rama develop. Después de que al menos un integrante del equipo revisara y aprobara el código, se aceptaba la solicitud y se procedía a hacer un *merge* de la rama en develop (Figuras 55 y 56).

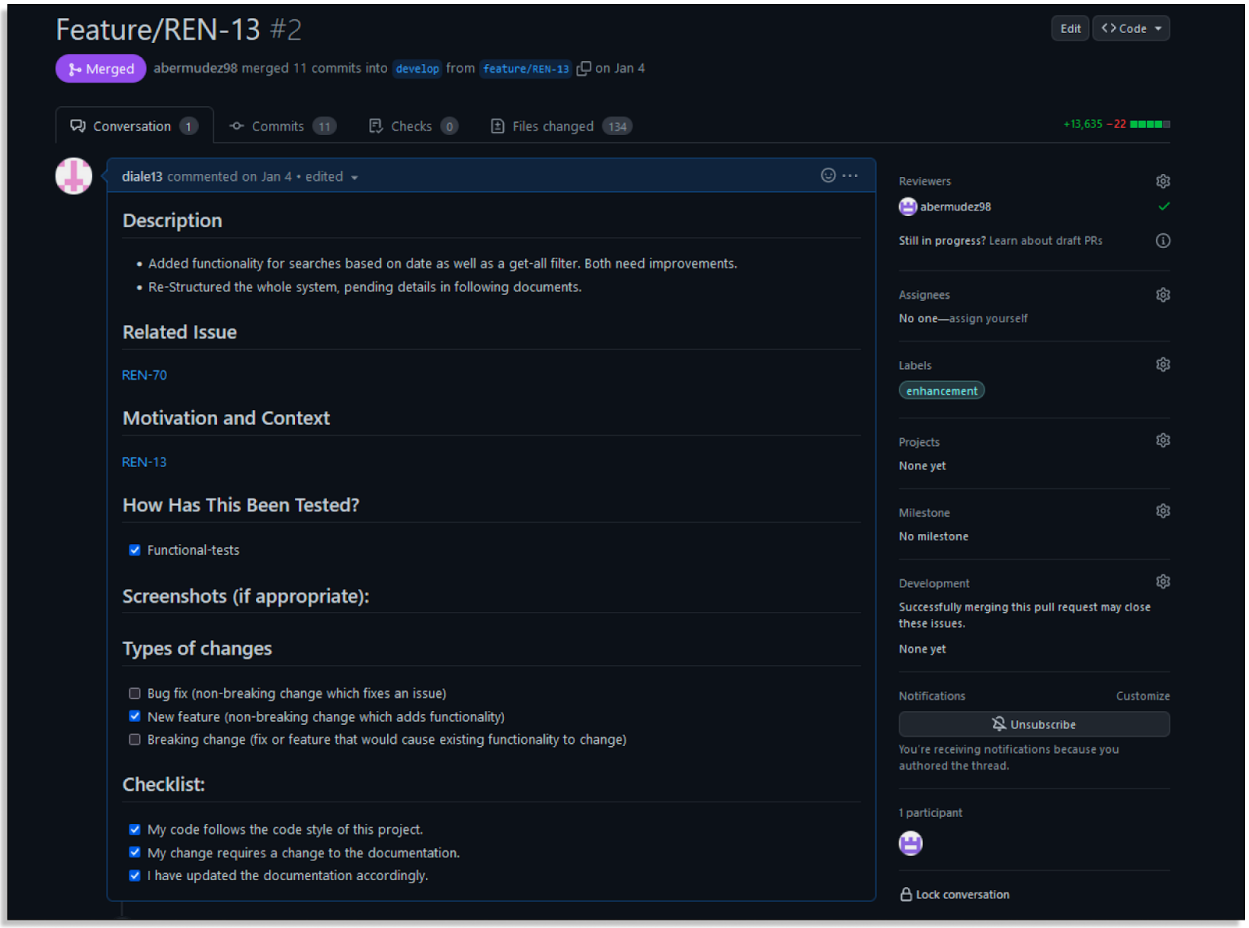


Figura 55: Ejemplo de *pull request*

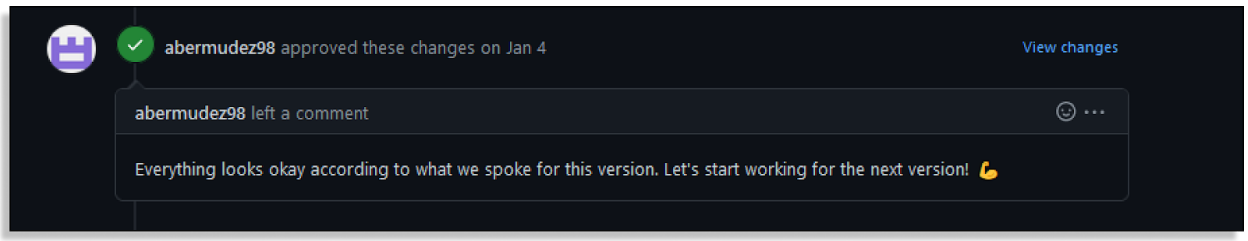


Figura 56: Ejemplo aprobación de *pull request*

Para gestionar los errores, se utilizaron dos tipos de ramas según su gravedad: *Hotfix* y *Bugfix*. *Hotfix* se utilizó para solucionar los errores que ocurrieran en versiones de producción que necesitaban atención inmediata. Para nombrar esta rama se utilizó una nomenclatura similar a la de una *feature*, pero con el prefijo "hotfix/" en lugar de "feature/". *Bugfix*, por su parte, se utilizó

para solucionar errores que no necesitaban ser atendidos de inmediato y podían solucionarse en la próxima liberación.

Luego de mergear una rama de funcionalidad (*feature*), las mismas eran borradas y pasadas las dos semanas (un Sprint), a modo de mantener la prolijidad del repositorio en caso de que el cambio haya sido exitoso.

Swagger

Swagger es una plataforma de *software* de código abierto que ofrece un conjunto completo de herramientas para diseñar, construir, documentar y utilizar servicios *web* RESTful. En el caso del equipo, se utilizó esta plataforma para generar la documentación de la API DocumentGateway, ya que los *endpoints* de la misma eran necesarios para el *frontend*.

La documentación generada consta de una lista detallada de *endpoints*, junto con su descripción, que incluye la URI del recurso, el verbo HTTP, los *headers*, el *payload* del *request*/*response* y los códigos de estado de la respuesta, entre otros detalles relevantes.

La ventaja de utilizar Swagger radica en que la documentación se genera automáticamente, lo que garantiza que se mantendrá actualizada en tiempo real a medida que se produzcan cambios en la API (Figuras 57 y 58).

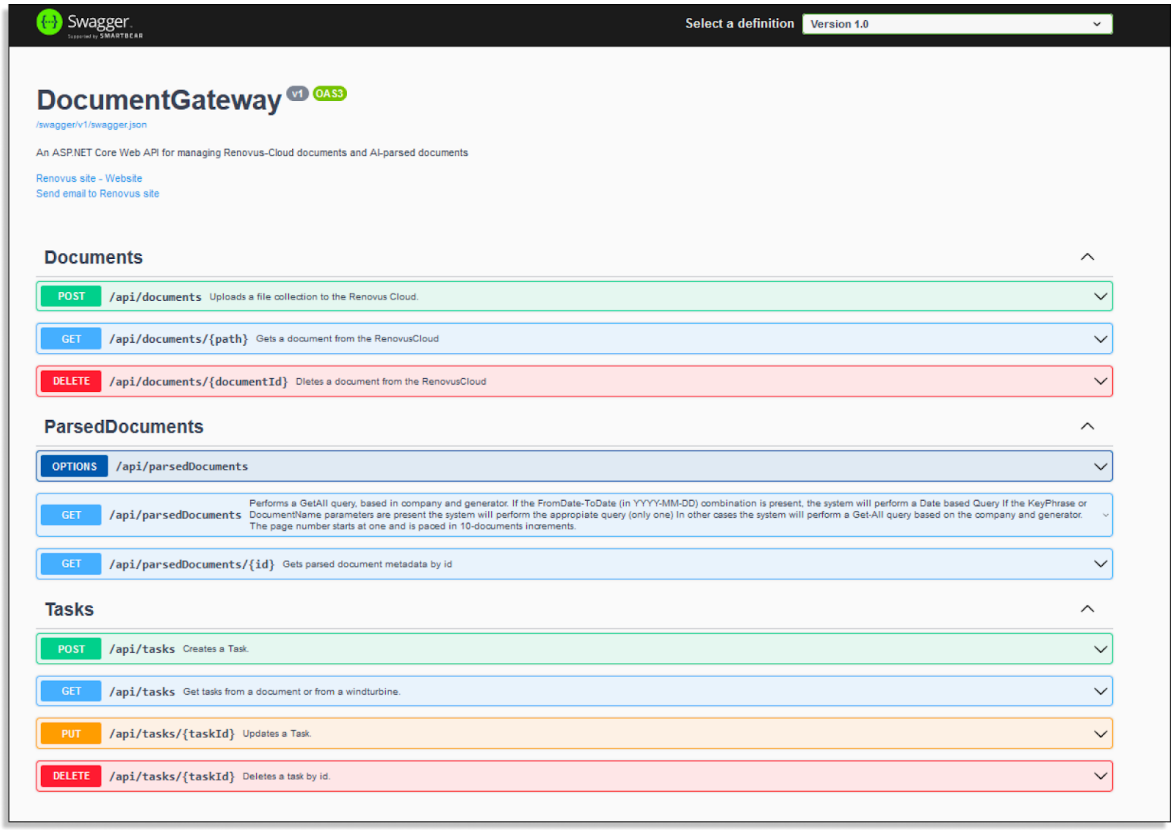


Figura 57: Vista completa de Swagger para DocumentGateway

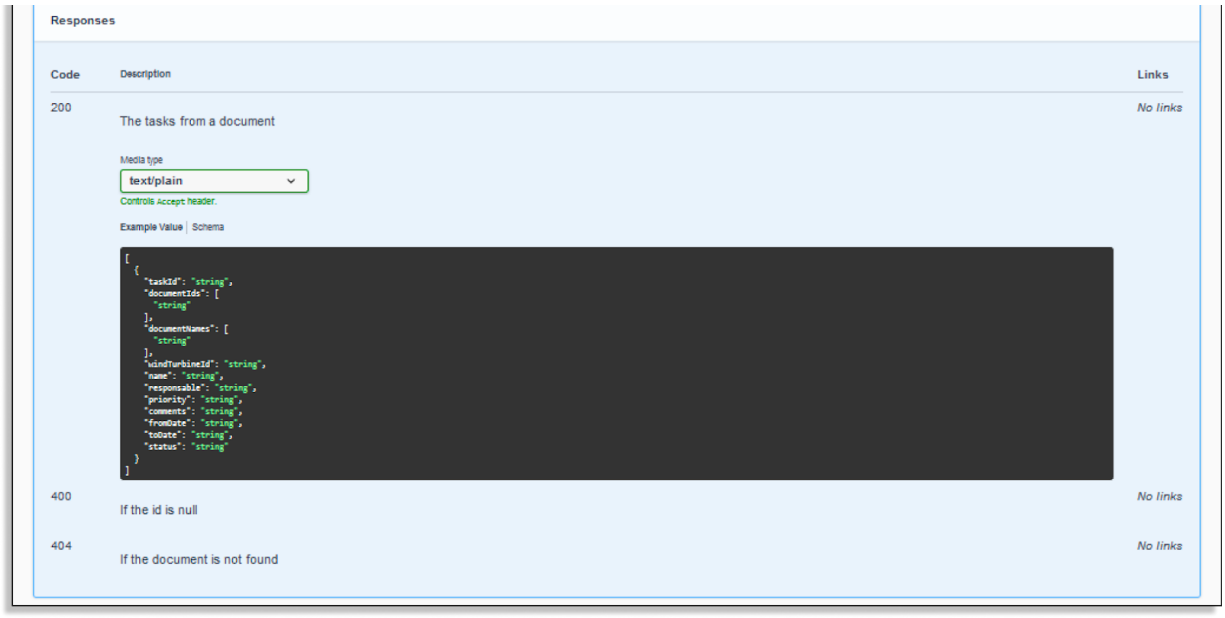


Figura 58: Vista expandida de una *response* de un *endpoint*

Documentación

Para todo lo referente a elementos de configuración del tipo documento, se utilizó Google Drive para los elementos académicos del equipo, así también como elementos de investigación, gestión y arquitectura del proyecto. Asimismo, también se utilizó Jira para la gestión del Product Backlog y los repositorios para los archivos README encargados de contener instructivos de instalación y ejecución (Figuras 59 y 60)

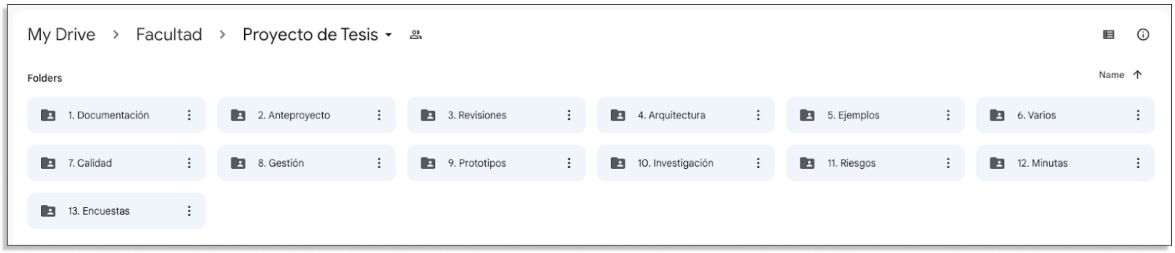


Figura 59: Carpeta dedicada a Proyecto de Tesis

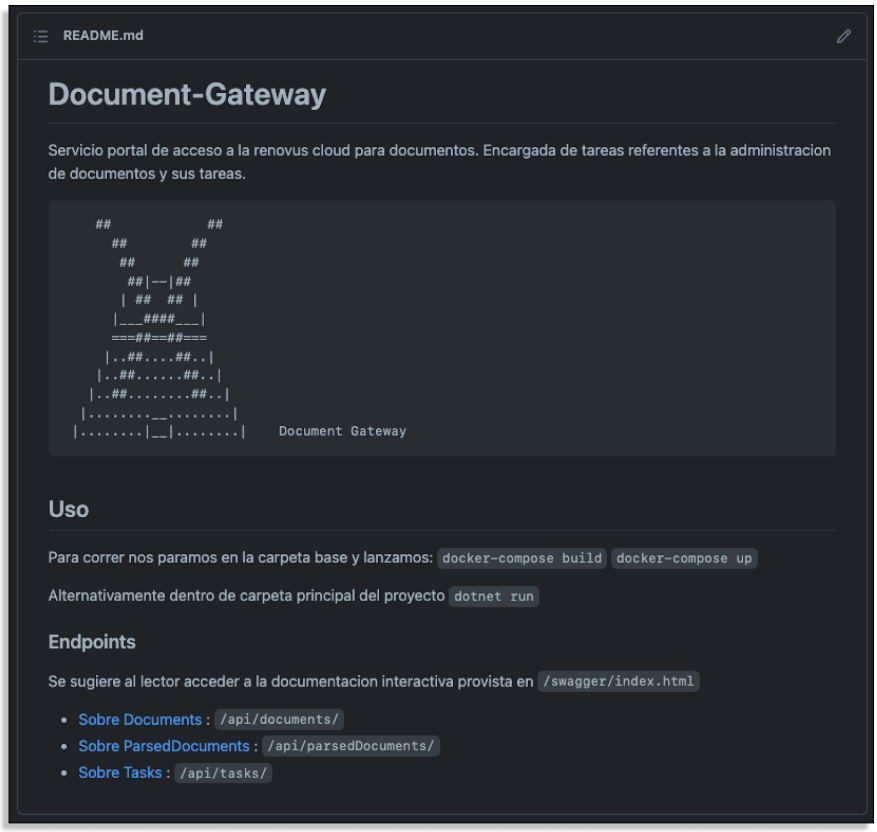


Figura 60: Documentación Readme dentro de repositorios de GitHub

6.8.3 Flujo de trabajo

Como fue mencionado anteriormente, para la gestión del flujo de trabajo se utilizó la metodología de *branching* GitFlow. El objetivo de la misma fue mantener un estándar a la hora de trabajar para así evitar errores y agilizar la producción.

El flujo de trabajo fue el siguiente:

1. Escoger la tarea de mayor prioridad del *board* de Jira y mover la misma a la columna "In Progress" para comenzar su trabajo.

2. Crear rama feature/REN-XX desde la rama de desarrollo, *develop*.

3. Una vez terminada la nueva *feature*, verificar que la nueva funcionalidad se encuentra funcionando de forma correcta.

4. Una vez confirmado el cambio, crear el *PR* (*Pull Request*) de la funcionalidad con intención de *merge* a la rama *develop*, la cual debe contener el título de la tarea de Jira, una descripción robusta, así como también los cambios realizados y una demo de la funcionalidad de ser necesario. Una vez realizado, notificar al resto del equipo y mover la tarea de Jira a la columna "Code Review".

5. Una vez en revisión, un desarrollador ajeno al *PR* realizado será el encargado de hacer la revisión total del *PR* y de ser necesario realizar comentarios o sugerir mejoras.

6. Una vez aprobado el *PR*, el creador del mismo es el encargado de mergear la rama a develop y mover la tarea realizada de la columna "Code Review" a "Done" en el *board* de Jira.

Este proceso fue fundamental para evitar errores y malentendidos. La intención principal fue hacerlo en conjunto con el tablero de Jira para así de esta manera generar métricas exactas y tener a todo el equipo actualizado sobre la situación del proyecto con tan solo mirarlo.

6.8.4 Uso de herramientas automatizadas en repositorio

El alcance establecido por el cliente no requería necesariamente de herramientas de aplicación automática al realizar algún cambio, por ejemplo un despliegue automatizado para todos sus componentes en simultáneo y la ejecución de distintos controles de código. De todas formas, el equipo encontró dos opciones de automatización, una siendo para la liberación del sitio *web* y otra para las pruebas de la plataforma *backend*.

En cuanto al sitio *web*, el objetivo fue automatizar las liberaciones en distintas ramas del repositorio de GitHub, de esta forma generando dos sitios *web* estáticos utilizando la plataforma Render que permite el *hosting* gratuito de sitios *web*, creando así uno de producción (rama *main*) y otro de desarrollo (rama *develop*). De esta forma, tanto el cliente como el equipo pudieron hacer uso de la página en todo momento para poder probar tanto la última versión del *release* como la última versión en desarrollo. La ejecución del despliegue se ejecutaba cada vez que se subía un nuevo cambio a las ramas *main* o *develop*. Así, cada vez que un *PR* fuese aceptado, al hacer *merge* a la rama de desarrollo se actualizaría en el sitio *web* automáticamente.

Por otra parte, en cuanto al *backend*, se realizó la implementación de pruebas automáticas para garantizar que el código cumpla con todas las pruebas unitarias establecidas. Para ello se utilizó la herramienta llamada GitHub Actions [39] (Figura 61) donde al momento de hacer un cambio en el repositorio, se corren las pruebas realizadas y se verifica que la construcción del sistema esté funcionando correctamente.

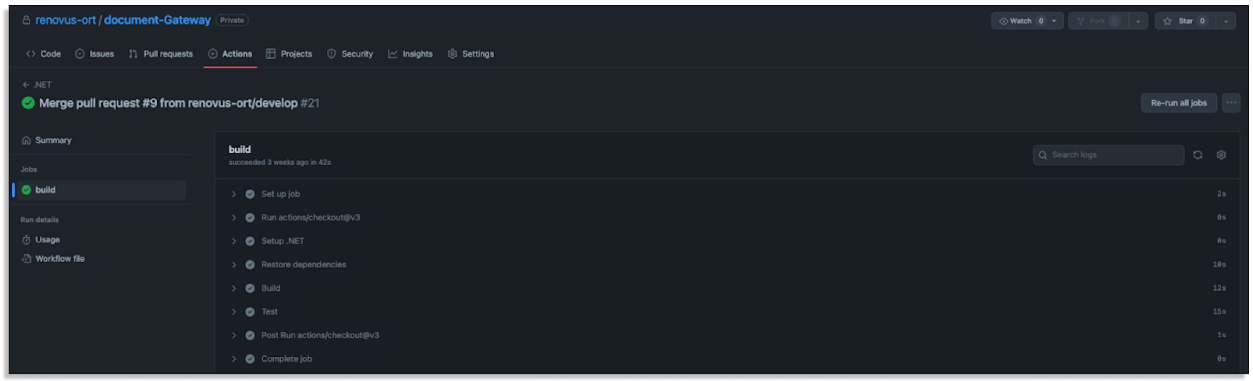


Figura 61: Vista GitHub Actions

6.8.5 Gestión de dependencias

Para la gestión de dependencias se utilizaron una variedad de gestores dependiendo de la plataforma y del lenguaje que se estaba trabajando.

En el caso del *frontend*, al estar codificado en JavaScript utilizando React.js [40] se utilizó el gestor de paquetes yarn [41], previamente utilizado por el cliente, el cual permite consumir módulos que se encuentran en el repositorio central de npm. Las dependencias del *frontend* se definen en el archivo package.json mientras que las versiones de las mismas en el archivo yarn.lock.

Por otra parte, en el *backend* se utilizó una serie de gestores de dependencias, entre ellos NuGet [42], el cual funciona como el gestor de paquetes para códigos realizados, utilizando el lenguaje .NET. [43]

Finalmente para la gestión de las dependencias del código desarrollado en Python [44], para las Lambda, se aprovechó de una herramienta de AWS Lambda [45] conocida como Layers. Estas permiten empaquetar las dependencias del código y subirlas a la nube, garantizando que todas aquellas funciones que dependan de una librería usen la misma versión y sea más sencillo realizar el mantenimiento cuando sea requerido.

6.9 Herramientas de apoyo

WhatsApp

Esta herramienta se utilizó como forma principal de comunicación tanto entre los miembros del equipo como con los miembros de Renovus y el tutor del proyecto. Con objetivo de mantener el orden se crearon 3 grupos, uno destinado únicamente a los miembros del equipo, otro para el equipo en conjunto con Renovus (utilizado para coordinar pequeñas tareas o planificar videollamadas) y por último otro grupo entre el equipo y el tutor del proyecto, destinado a la planificación y discusión de variedad de temas relacionados al proyecto.

Google Meet

Google Meet fue la principal herramienta utilizada para realizar videollamadas cuando era necesario reunirse con una persona ajena al equipo, como por ejemplo tutor, cliente, revisores y expertos, entre otros.

Discord

Discord fue la plataforma principal de comunicación interna entre los miembros del equipo. Sirviendo como herramienta para realizar Sprint reviews, Sprint planning, refinamientos y variedad de eventos de coordinación.

Si bien su valor principal fue el de realizar llamadas de audio, también se utilizó como gestor de notas del equipo, manteniendo canales de comunicación para compartir información de desarrollo relacionadas con el proyecto (Figura 62).

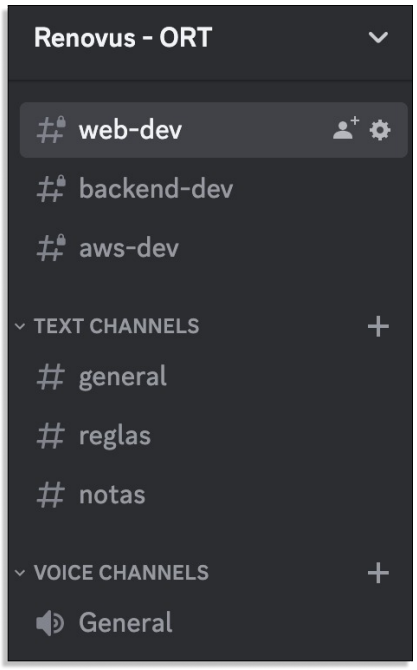


Figura 62: Vista Discord

Google Calendar

Con motivo de gestionar las reuniones tanto con el cliente como con el tutor y los eventos semanales, se utilizó Google Calendar para administrar reuniones y notificar a los integrantes de las mismas las reuniones agendadas. Este mecanismo aportó un gran valor al equipo dado que ayudó a planificar eventos semanales útiles para la gestión (Figura 63).

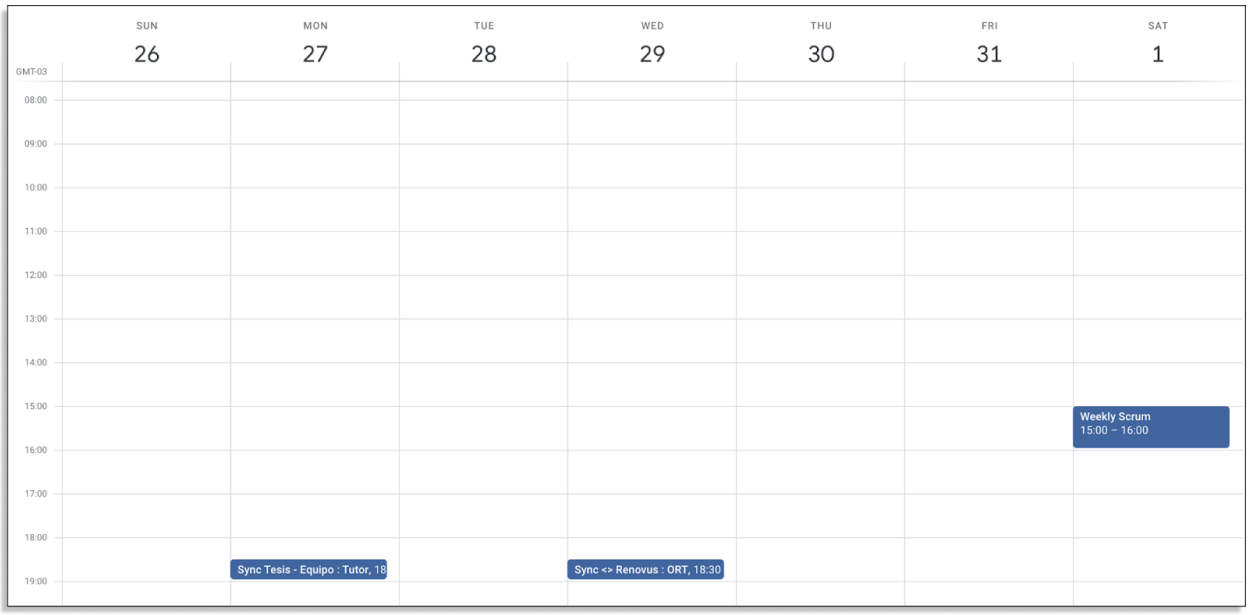


Figura 63: Vista Google Calendar

Jira

Durante el desarrollo del proyecto, la principal herramienta de gestión fue Jira, plataforma que le permitió al equipo administrar de manera efectiva y sencilla todos los artefactos de Scrum y otros componentes adicionales incluyendo el Product Backlog, Task Board, reporte de incidencias, historias de usuario, *bugs*, épicas, versiones y los Sprints desarrollados. Esta plataforma fue elegida debido a la experiencia previa de algunos de los integrantes y a la eficacia demostrada en la gestión de proyectos similares.

Además de la gestión de tareas, Jira también proporcionó una variedad de informes útiles, como el Burndown Chart que permitió visualizar el progreso del trabajo durante cada Sprint. El Sprint Report brindó información detallada sobre las tareas realizadas en cada iteración y el Velocity

Chart permitió evaluar la velocidad del equipo en cada Sprint, lo que ayudó a planificar futuros Sprints a medida que avanzaba el proyecto. (Figura 64)

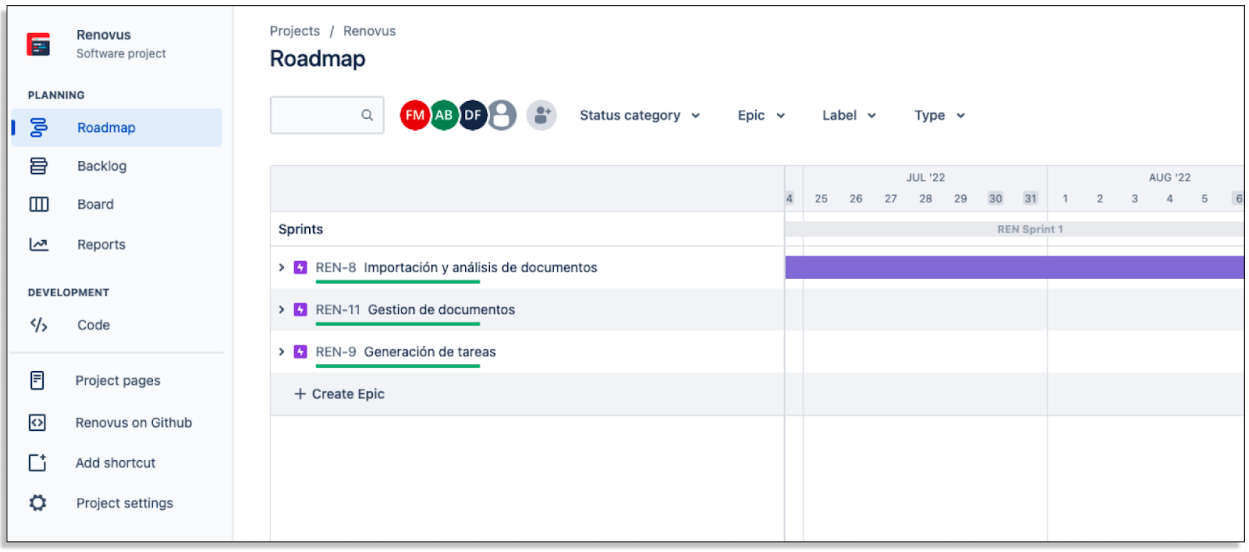


Figura 64: Vista Roadmap Jira

Google Drive

Como fue mencionado en la sección 6.8.2, Google Drive se utilizó como herramienta de gestión de documentos importantes del proyecto, así funcionando como repositorio de informes realizados y documentos importantes. Drive se utiliza tanto para compartir información entre miembros del equipo como también con el tutor y el cliente.

6.10 Traspaso de conocimiento

Una vez finalizado el proceso de desarrollo y concluida la última revisión de Sprint, el equipo inició la transferencia de conocimientos al cliente.

Para el traspaso de conocimiento se realizó un documento en el cual se puede apreciar todos los detalles del funcionamiento del sistema y cómo modificarlo y extenderlo a futuro. Este documento se encuentra en el Anexo 10.5.

Por otra parte, una vez finalizada la última demo del producto, se demostró cómo entrenar el sistema (detección de datos de documentos) y el equipo se puso a disposición para atender cualquier consulta o duda que pueda tener el cliente.

7. Gestión y aseguramiento de calidad

En esta sección se detalla cómo se llevó a cabo la Gestión de la Calidad (*Quality Management* o QM por sus siglas en inglés) del proyecto en cuestión. Aquí se describen los objetivos planteados, el plan establecido para cumplirlos, las tareas realizadas y las etapas identificadas. También se incluirán los resultados del proceso y las etapas para asegurar la calidad del producto en secciones y anexos debidamente identificados cuando corresponda.

En el Capítulo 3, sección 1 se detalló la solución funcional, lo que permitió describir el alcance final y la evolución que se dio desde el primer alcance acordado hasta el último. Este proceso de cambio no solo afectó el alcance, sino que también impactó en la gestión de calidad. Inicialmente, el proyecto iba a ser un complemento productivo de la solución actual de Renovus, lo que implicaba que el aseguramiento de la calidad del producto final sería más exhaustivo y completo, incluyendo pruebas automatizadas a nivel de interfaz de usuario (*User Interface* o UI por sus siglas en inglés) o de API, así como el diseño de casos de prueba, entre otras cosas.

Después de las últimas modificaciones, el producto a construir se convirtió, por decisión del cliente, en un MVP. Por esta razón, el equipo decidió dedicar más tiempo a otras tareas, considerando que trabajarían con tecnologías desconocidas y profundizarían en inteligencia artificial, un aspecto en el que tenían poca o ninguna experiencia. Por lo tanto, se decidió prescindir de algunas particularidades, como las pruebas automatizadas de UI, optando por trabajar bajo la modalidad de Agile Testing (concepto que se explicará más adelante). Estas decisiones se basaron en la premisa de mantener las ideas y prácticas originales, en el convencimiento de que aportaría un gran valor a largo plazo, pero al tratarse de un MVP, el valor añadido quizás no fuera el esperado por el cliente. Por lo tanto, se priorizaron otros aspectos que generarían un mayor valor para el proyecto y el cliente.

7.1 Objetivos

Para poder hablar de la gestión y aseguramiento de la calidad, es necesario establecer ciertos objetivos que permitan medir estos aspectos, permitiendo tanto, al equipo como al cliente, tener referencias de progresión y cumplimiento. Considerando el tipo de proyecto y los desafíos

presentados, se decide dividir los objetivos en aquellos vinculados al producto, a su desarrollo y demás aspectos por un lado y por otro, a los vinculados al proceso en sí mismo. Esta medida se debió a que el equipo se enfrentaba a varios desafíos nuevos, y se entendió necesario organizarlos adecuadamente para ayudar a clarificar y facilitar el desarrollo del proyecto.

7.1.1 Objetivos del Producto

Dentro de los objetivos del producto encontramos los siguientes:

- Cumplir con la totalidad de requerimientos funcionales y no funcionales, teniendo en cuenta las restricciones impuestas por el cliente.
- Prevenir la introducción de defectos al *software*.
- Lograr aumentar el grado de confianza en los algoritmos de Amazon Textract y Amazon Comprehend para los *inputs* utilizados por Renovus.
- Al momento de entregar el proyecto no poseer defectos clasificados cómo Críticos o Bloqueantes.

7.1.2 Objetivos del Proceso

Para el proceso, los objetivos definidos son los siguientes:

- Cumplir con las ceremonias establecidas en el marco del proceso que utilizó el equipo basado en Scrum, así como el cumplimiento de los artefactos que derivan de cada instancia.
- Asegurar el correcto entendimiento de las necesidades del cliente.
- Mantener a los interesados informados sobre los aspectos relevantes a lo largo del proyecto.

7.2 Plan de calidad

Ya con los objetivos definidos, se procedió con la creación de un plan de calidad. Este plan tendría ciertas contemplaciones y libertades que se irán explicando a lo largo de la sección, considerando el producto en cuestión y su objetivo final.

Los MVP suelen plantearse como proyectos de bajo costo que establecen las ideas principales del producto que se desea construir. Por esta razón, no suelen contar con planes de calidad ni con otros aspectos, tareas y artefactos vinculados a la calidad del producto, ya que no es el foco principal. En este caso, el producto en cuestión tiene muchas características de MVP, pero a la vez se profundiza más en el desarrollo, teniendo en cuenta el tiempo disponible para la realización del proyecto por parte del equipo. Por esta razón, se decidió implementar un plan de calidad que permita que este MVP se acerque lo más posible a convertirse en un complemento productivo de la solución actual. Asimismo, se busca asegurar que el producto tenga una buena base en cuanto a calidad respecta, para su integración en el futuro.

En la siguiente sección, se explicarán de manera general las actividades llevadas a cabo a lo largo del proyecto, detallando el objetivo y los resultados alcanzados en cada una de ellas. Todo esto se presenta en el contexto del plan de calidad, con el fin de comprender mejor los aspectos de calidad a considerar en cada etapa y actividad del proyecto.

7.2.1 Actividades transversales

Estas actividades son aquellas que se desarrollan a lo largo del proyecto y no se vinculan a ninguna etapa en particular. Las mismas no tienen como responsable a ninguno de los roles mencionados anteriormente, sino que, los responsables son todos los integrantes del equipo.

Actividades comunes a todas las etapas:

- Contacto semanal con el cliente: regularmente, el equipo mantiene contacto con el cliente a través de reuniones quincenales, así como también con el intercambio de mails e incluso interacciones por un grupo de WhatsApp creado con tal finalidad. Esto se realiza con el objetivo de intercambiar opiniones, visiones sobre el producto, realizar diferentes validaciones, presentación de avances y propuestas de nuevas ideas del producto.

- Seguimiento del Plan de Estándares: el equipo especificó un documento que recopila diferentes estándares que serán tenidos en cuenta para el desarrollo del proyecto. Este documento será detallado más adelante.

- Contacto semanal con el tutor: todas las semanas se intenta mantener contacto con el tutor para coordinar y validar aspectos de los avances realizados, principalmente volcado a aspectos del proceso, así como también del producto de *software*.

Resultados de la etapas comunes

Todas las actividades arrojaron resultados muy positivos para el equipo. El contacto semanal con el cliente resultó fundamental para el desarrollo del producto, ya que se generaron instancias de retroalimentación en las que se plantearon muchas ideas y aportes que permitieron recabar información valiosa para la elaboración del producto, entre otros aspectos. El seguimiento del plan de estándares que se estableció, permitió facilitar el proceso de construcción de algunos elementos cómo pueden ser los documentos o el código. El contacto con el tutor fue de gran ayuda, ya que proporcionó al equipo numerosos consejos y guías para llevar a cabo el proyecto. Además, el tutor ofreció ideas para el desarrollo y aportes valiosos en el proceso, lo que permitió sacar el máximo provecho de cada instancia, incluyendo las revisiones realizadas por la Universidad ORT.

7.2.2 Investigación y capacitación

Estas actividades principalmente ocurrieron al principio del proyecto, pero durante el desarrollo se presentaron situaciones puntuales en las que el equipo desconocía algunas tecnologías o aspectos del negocio en cuestión. En estos casos, se requirió nuevamente de instancias de investigación y capacitación para superarlas.

Actividades de la etapa

- Investigación de herramientas disponibles: el equipo en su conjunto realizó una investigación de las herramientas que se encontraban disponibles para el desarrollo del proyecto, centrándose, principalmente, en las herramientas vinculadas a la IA, dado que era el área en la cual el equipo contaba con menos experiencia y conocimientos.

- Elección de herramientas: después de analizar la información recopilada durante la investigación, el equipo seleccionó las herramientas consideradas útiles y presentó sus ventajas y desventajas al cliente, con el fin de acordar cuáles serían utilizadas para continuar con el proyecto.

- Capacitación en herramientas seleccionadas: una vez establecidas las herramientas que se utilizarán a lo largo del proyecto para la construcción del producto de *software*, el equipo decide capacitarse en las mismas, teniendo en consideración los puntos fuertes y débiles de cada integrante.

- Capacitación en el lenguaje de programación Python: el equipo decide capacitarse en Python, ya que no tenían un conocimiento profundo del lenguaje y considerando la elección previa de algunas herramientas, consideraron que era necesario para obtener mejores resultados con ellas.

Resultados de la etapa

Al igual que la etapa anterior, esta fue muy positiva, aunque bastante desafiante para el equipo, dado que se debió profundizar en tecnologías y aspectos del negocio que no conocían tanto, como por ejemplo, la Inteligencia Artificial, los servicios de AWS y los servicios de Azure, entre otros. La investigación de herramientas le brindó un gran abanico de posibilidades al equipo en lo que a la construcción del producto refiere, y principalmente con respecto a la IA (punto más débil del equipo inicialmente) dónde destacaron los servicios de Microsoft y de Amazon mencionados previamente. Finalmente y luego de intercambiar opiniones con el cliente, el equipo se apegó a la decisión de tener todo alojado en AWS optando por estos servicios.

Una vez definida las herramienta mencionadas, el equipo procedió a capacitarse en aquellas que más desconocía, cómo pueden ser los servicios de AWS, particularmente Amazon Textract y Amazon Comprehend. Finalmente, se realizó la capacitación en Python, lenguaje que parte del equipo conocía y que entendían vital para sacar el mayor rédito posible de algunas herramientas utilizadas cómo los Lambda de AWS.

A pesar de haber sido una etapa desafiante para el equipo, se considera positiva, ya que, gracias al buen trabajo que se realizó, se agilizaron las etapas de construcción y se pudieron obtener buenos resultados de cara a la construcción del producto de *software*.

7.2.3 Ingeniería requerimientos

Estas actividades están relacionadas con los requerimientos necesarios para construir el producto de *software* a fin de satisfacer las necesidades del cliente.

Actividades de la etapa

- Identificación de requerimientos: en esta actividad, el equipo se apoyó en las reuniones quincenales que tenía con el cliente, además de un mail enviado por ellos que contenía un listado con las funcionalidades pretendidas para el sistema, así como también del documento inicial presentado a la Universidad ORT para identificar en una primera instancia los requerimientos del producto de *software* a construir.

- Refinamiento de los requerimientos: posterior a la actividad anterior, el equipo trabajó para refinar los requerimientos identificados y poder construir una lista de requerimientos que luego transformarían en historias de usuario, como se establece en la sección 4.5.

- Validación de la lista de requerimientos y las historias de usuario: luego del refinamiento realizado por el equipo, se vuelve a iterar con el cliente para validar los requerimientos y las historias de usuario construidas a partir este listado para de esta manera obtener *feedback* sobre el trabajo realizado por el equipo y poder realizar los ajustes necesarios.

- Correcciones finales: con los comentarios y sugerencias recibidos por parte del cliente, el equipo procede a realizar las correcciones y ajustes necesarios para presentar una versión final de los requerimientos. Posteriormente, se comparte con el cliente a fin de obtener la validación final y la aprobación para continuar con el proyecto.

- Estimación: al contar con la lista de requerimientos, el equipo puede proceder a la estimación de las historias de usuario, actividad que resulta vital para las proyecciones del equipo.

- Priorización de los requerimientos: una vez que la lista de requerimientos fue validada y aprobada por el cliente, se procede a la priorización de los mismos con el Product Owner.

Resultados de la etapa

La etapa de Ingeniería de Requerimientos resultó clave para el desarrollo del proyecto, dado que una mala definición y priorización de los requerimientos puede conducir a grandes dificultades a futuro e incluso al fracaso del proyecto. Al igual que en etapas anteriores, se considera la misma muy positiva, con un gran apoyo del cliente para la construcción de la lista de requerimientos y de historias de usuario así como los criterios de aceptación de cada una. Estas últimas conformaron el Product Backlog que fue vital para establecer los Sprints y los Releases del proyecto.

7.2.4 Arquitectura

Esta etapa contempló todas aquellas actividades vinculadas a la definición de la arquitectura del sistema.

Actividades de la etapa

- Análisis de requerimientos no funcionales: en esta actividad el equipo se encargó de realizar un análisis en profundidad de los requerimientos no funcionales para definir los aspectos más importantes de la arquitectura, con el objetivo de cumplir con los atributos de calidad derivados de estos (sección 4.5.6).

- Bosquejo y análisis de posibles soluciones: a raíz de la actividad anterior, y los comentarios obtenidos para el diseño de la arquitectura, el equipo comienza a delinear ideas para la misma, e inicia un proceso de iteración con el objetivo de alcanzar la mejor solución posible. Para consultar la evolución de la arquitectura y el resultado de estos bosquejos se refiere al lector al Anexo 10.10.

- Validaciones: el equipo optó por validar en diferentes ocasiones el diseño de la arquitectura tanto con la contraparte técnica del cliente, quien participaba de las

reuniones quincenales, así como también con el tutor en alguna ocasión, y con expertos de la Universidad durante las evaluaciones requeridas.

Resultados de la etapa

La etapa fue rigurosa, pero altamente beneficiosa. Después de analizar los requerimientos no funcionales, se identificaron los atributos de calidad necesarios. A través de una combinación de la información obtenida durante el análisis, diferentes tácticas y el Product Backlog previamente creado, el equipo comenzó a diseñar y a iterar sobre la arquitectura. Cada iteración se basó en validaciones y en lo que el equipo aprendió en cada fase del proceso, lo que permitió la evolución constante de la arquitectura hasta lograr un producto satisfactorio tanto para el cliente, para el equipo, como para el proyecto en general. El resultado final fue una solución que cumplió con los requisitos del proyecto y dejó satisfechos a todos los miembros del equipo.

7.2.5 Construcción y testing

Como se detalló anteriormente, este proyecto se basa en los marcos de trabajo *Agile* y el ciclo de vida incremental iterativo, por lo cual se está constantemente iterando, motivo por el cual estas actividades se vieron repetidas a lo largo del proceso de construcción del producto de *software*.

Actividades de la etapa

- Reevaluación de arquitectura: a medida que el proyecto avanza, y se va construyendo la solución, el equipo reevalúa la arquitectura con el fin de buscar puntos de mejora y optimización en la misma.

- Adaptación/Modificación de la arquitectura: en caso de que la actividad anterior dejara puntos de mejora u optimización, el equipo realizaba los cambios que entendía necesarios para la mejora de la arquitectura.

- Desarrollo: el equipo comienza el desarrollo de las tareas acordadas para el Sprint, con el objetivo de incrementar las funcionalidades existentes en el sistema. En esta tarea se tienen en cuenta los estándares de codificación y las buenas prácticas de diseño para lograr el mejor resultado posible.

- Refactoring: durante el proceso de desarrollo, surgen oportunidades para mejorar y optimizar el código existente mediante un proceso conocido como *refactoring*. Este proceso implica revisar y ajustar los elementos del código existente con el objetivo de mejorar la calidad y la eficiencia de desarrollos previos, si es posible.

- Pair Programming: para ciertos módulos, según el equipo entendiera necesario, se aplicó esta técnica para la validación del código en construcción, ya que se busca brindar mayor seguridad en la calidad del código al tener múltiples perspectivas y habilidades enfocadas en un solo módulo.

- Peer Validation: esta técnica se aplicó en todos los módulos, tanto para el *backend* como para el *frontend* mediante el uso de *pull-requests* en GitHub. El uso de esta funcionalidad implicó que todos los integrantes del equipo participaran de la aprobación y validación del trabajo efectuado por el resto, buscando aportar mayor confiabilidad al código.

- Validación mediante prototipos: el equipo elabora diferentes prototipos según la funcionalidad que se esté trabajando para obtener *feedback* del cliente, estas presentaciones se efectúan en las reuniones quincenales.

- Seguimiento del Plan de Testing: durante la construcción el equipo se asegura de hacer el seguimiento y cumplimiento del plan de *testing*, desarrollando las distintas prácticas según esté indicado para la etapa.

Resultados de la etapa

Todas las actividades llevadas a cabo en esa etapa y sus resultados fueron fundamentales para el resultado final. En esta se materializó una gran parte del trabajo y se obtuvo el *software* que se había comenzado a planificar en las etapas anteriores. Durante esa etapa, se llevó a cabo un proceso riguroso que involucró múltiples actividades diseñadas para asegurar la máxima calidad posible en el producto final. Se controlaron todos los aspectos posibles para evitar defectos que pudieran deteriorar el valor que se buscaba brindar al cliente y se aseguró que se cumplieran todos los requisitos y especificaciones previamente definidos.

7.2.6 Documentación

Por último se presentan las actividades referentes a la documentación del presente documento:

- Revisiones de ortografía y gramática: durante la documentación, el equipo se dividirá la redacción de las diferentes secciones de la misma, y luego irán intercambiando y rotando entre sí para revisar y asegurar la calidad de los textos.

- Revisiones de redacción y completitud: en esta actividad, el equipo tendrá cómo objetivo garantizar la correcta redacción del documento, buscando que éste mantenga el sentido y la coherencia a lo largo de todas las secciones, así como también intentará garantizar la completitud del mismo asegurándose de que no queden elementos clave por fuera del mismo.

- Revisión de citas y referencias: durante la redacción de la documentación, se deben revisar las citas y referencias bibliográficas para que cumplan con el estándar establecido.

Resultados de la etapa

Fue esencial garantizar la calidad de la documentación por múltiples razones, entre ellas, la facilidad de mantenimiento del producto a largo plazo y el registro del proyecto por parte del equipo estudiante. Un desempeño sobresaliente en esta fase fue crucial para asegurar un nivel mínimo de calidad y declarar el proyecto como satisfactorio.

7.3 Aseguramiento de la calidad

En esta sección, se procederá a detallar las distintas tareas, actividades, y documentos realizados por el equipo para lograr asegurar la calidad del producto de *software*.

7.3.1 Estándares

En cuanto a los estándares, el equipo ha establecido una distinción entre los estándares de codificación y los de documentación. Los estándares de codificación son pautas establecidas por el equipo para guiar la construcción y desarrollo del *software* con el fin de producir un código de alta calidad y fácilmente mantenible en el futuro. Por otro lado, los estándares de documentación

se basaron en los documentos proporcionados por la Universidad ORT Uruguay para la entrega formal del proyecto final de carrera.

Estándares de codificación

En este proyecto, se aplicaron los conocimientos y estándares obtenidos de diferentes libros y textos de referencia en la industria del desarrollo de *software*. En este apartado, se enumeran los estándares de codificación utilizados por el equipo, junto con una breve explicación del aporte que brindó cada uno. De esta manera, se buscó asegurar que el código generado fuera de alta calidad, fácilmente mantenible y escalable a futuro.

- **Clean Code:** es un estándar ampliamente utilizado en la industria del desarrollo de *software*, basado en el libro homónimo de Robert Martin [46]. Este ha sido y sigue siendo una referencia clave para los desarrolladores que buscan mejorar la calidad de su código y hacerlo más mantenible en el tiempo. El libro está construido sobre la base de Java, pero los principios y conceptos establecidos pueden extrapolarse, por lo general, a los demás lenguajes, dado que su objetivo principal es aportar en el desarrollo de un código de calidad sin importar la base utilizada. Este es el motivo por el cual el equipo optó por su utilización para los lenguajes utilizados, principalmente C# y JavaScript.

- **Refactoring:** el libro "Refactoring: Improving the Design of Existing Code" escrito por Martin Fowler y Kent Beck [47] ha sido publicado en dos versiones: una primera edición en 1999 y una segunda edición más reciente, lanzada a finales de 2018. Aunque ambas ediciones comparten muchos conceptos, la segunda edición incluye numerosas actualizaciones y nuevas técnicas, fruto de la experiencia acumulada a lo largo de los años. Si bien la primera edición del libro se centraba en el lenguaje de programación Java, la segunda edición se enfoca en JavaScript debido a su gran popularidad en la actualidad. Sin embargo, muchas de las técnicas presentadas en ambos libros son aplicables a diferentes lenguajes de programación, ya que el objetivo principal es transmitir técnicas para mejorar la calidad del código, independientemente del lenguaje utilizado.

- **JavaScript ES6:** se adoptó la versión ES6 [48] de JavaScript como estándar de codificación para el *frontend* desarrollado en React JS. Esta elección se basó en varios factores, entre los que se incluye la experiencia previa de uno de los miembros del equipo con este estándar, así como en su amplia aceptación en la industria del desarrollo de *software*. Además, dado que ES6 es una versión más moderna y actualizada de JavaScript, el equipo consideró que sería beneficioso utilizarlo para aprovechar sus funcionalidades y mejoras en términos de calidad de código y mantenibilidad.

- **REST:** Para las API desarrolladas, se adoptó el estilo de arquitectura REST como estándar de codificación. Esta elección se basó en su amplia aceptación en la industria del desarrollo de *software* y en su uso previo por parte del cliente en sus propias API y de los estudiantes al ser un estilo obligatorio en los proyectos de la Universidad. Al seguir este estándar, se buscó asegurar que las API fueran fáciles de entender, utilizar y mantener, lo que en última instancia, contribuiría a la satisfacción del usuario final.

- **SOLID:** El equipo consideró que los principios SOLID [25] eran fundamentales para el éxito del proyecto. Estos principios habían sido aplicados por varios años tanto en desarrollos académicos como laborales por todos los miembros del equipo, y además eran ampliamente reconocidos en la industria por su capacidad de mejorar la calidad del código y, en particular, fomentar su mantenibilidad. Al seguir estos principios, se buscó asegurar que el código fuera fácilmente comprensible, extensible y escalable, y que pudiera adaptarse fácilmente a futuros cambios o mejoras en el sistema. En última instancia, esto contribuiría a garantizar la satisfacción del usuario final y la viabilidad a largo plazo del producto de *software*.

Además de lo anteriormente mencionado, el equipo estableció otros dos aspectos cruciales para el desarrollo del código. En primer lugar, se acordó que todo el código desarrollado, ya sea *backend* o *frontend* y en cualquier lenguaje utilizado, debía estar escrito en inglés, debido a que actualmente es prácticamente un estándar en la industria y a que el cliente sería quien mantendría el código. En segundo lugar, se decidió utilizar herramientas de análisis como SonarQube [49] y ESLint [50], configuradas de tal manera que ayudaron en la revisión y validación del

cumplimiento de los estándares. Esta decisión demostró ser muy útil en la detección temprana de problemas y en la garantía de la calidad del código.

Ambas herramientas, SonarQube y ESLint, se utilizaron en el proceso de revisión de código para garantizar el cumplimiento de los estándares establecidos. SonarQube (Figuras 65 y 66) se encargó de analizar el código en busca de posibles problemas de calidad y seguridad, mientras que ESLint se enfocó en el cumplimiento de los estándares de codificación específicos del lenguaje utilizado. De esta forma, se logró una revisión rigurosa y eficiente del código desarrollado.

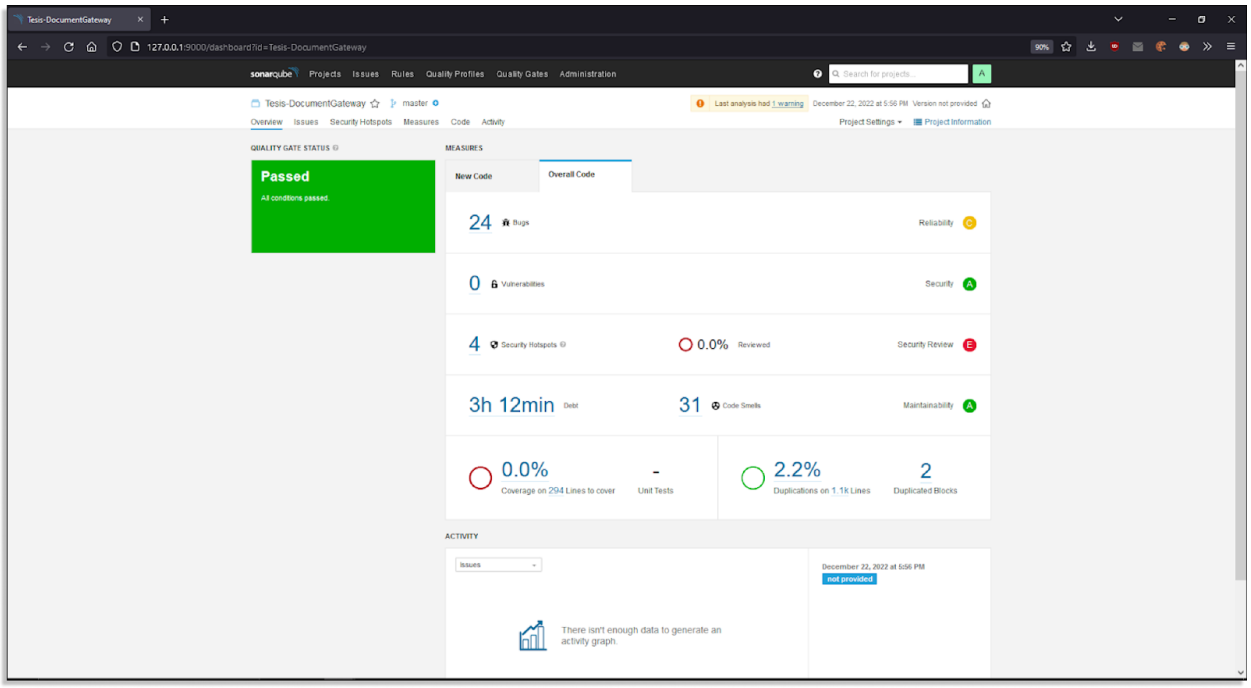


Figura 65: Vista SonarQube: Dashboard Release 2

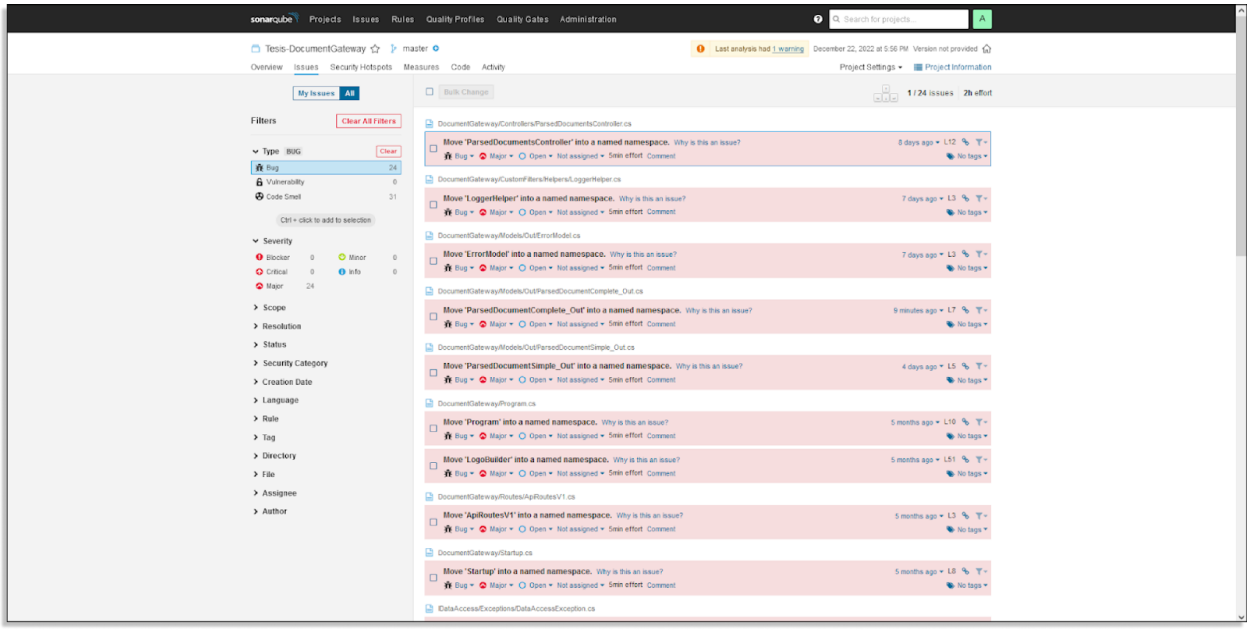


Figura 66: Vista SonarQube: Bugs

Estándares de documentación

Como se mencionó anteriormente, para los estándares de la documentación, tanto los académicos cómo cualquier otra documentación generada, el equipo optó por utilizar aquellos brindados por la Universidad ORT Uruguay para los proyectos finales de carrera.

- **Documento 302:** este documento corresponde a las normas específicas para la presentación de trabajos finales de carrera de la Facultad de Ingeniería (TFDC) excepto Biotecnología. [51]

- **Documento 303:** este documento corresponde a la hoja de verificación de pautas de presentación de trabajos finales de carreras de la Facultad de Ingeniería. [52]

- **Documento 304:** en este documento se encuentran las normas para el desarrollo de trabajos finales de carrera. [53]

- **Documento 306:** este documento corresponde a la orientación para títulos, resúmenes o abstracts e informes de corrección de trabajos finales de carrera. [54]

7.3.2 Revisiones

Durante el proyecto, el equipo llevó a cabo varias actividades de revisión para detectar áreas de mejora, problemas y oportunidades de optimización en el proceso y el producto. Estas revisiones se realizaron internamente, con el cliente y formalmente con revisores de la Universidad ORT Uruguay. Cada tipo de revisión se enfocó en diferentes aspectos, desde aquellos enfocados en los procesos hasta los de carácter más académicos, con el objetivo de mejorar en todos los ámbitos posibles.

Revisiones internas

En cuanto a las revisiones internas, en lo referente a la programación y el código generado por el equipo, cómo se mencionó anteriormente, se aplicó la técnica de *peer review*, o revisión de pares, la cual era utilizada bajo *code reviews* al momento de evaluar los *pull-requests* de otros integrantes del equipo.

Uno de los objetivos de estas revisiones era fomentar la colaboración entre los integrantes del equipo, alentándolos a estar atentos al código desarrollado por otros miembros. Sin embargo, el propósito principal de estas revisiones era detectar aspectos de mejora y garantizar el cumplimiento de los estándares establecidos. De esta manera, se buscaba mejorar la calidad del código y asegurar su mantenibilidad a largo plazo. Además de estas instancias, el equipo utilizaba sus reuniones semanales para presentar avances, y en estas instancias también se aprovechaba el tiempo para hacer pequeñas revisiones, ajustes e incluso intercambios de opiniones que permitieran detectar o prevenir problemas a futuro.

De cara a revisiones internas del proceso, el equipo aprovechaba las ceremonias establecidas en el marco de trabajo adoptado, sobre todo en la Sprint Retrospective, para discutir y poner sobre la mesa diferentes aspectos que cada uno consideraba del proceso, con el objetivo de identificar puntos de mejora. Durante estas discusiones, se evaluaban prácticas que debían mantenerse y otras que debían ser sustituidas o removidas del proceso.

También, para lo que respecta al proceso, se recurría a las encuestas internas del equipo (Anexo 10.7), como otra fuente donde cada integrante pudiera llegar a aportar elementos vinculados al proceso.

Para las revisiones internas de la documentación, el equipo se valió de la *suite* de Google, especialmente Docs, Sheets y Slides. Esto permitió que los integrantes pudieran comentar en las secciones que consideraban necesarias, sugerir ediciones para su posterior revisión y trabajar de forma colaborativa y sincrónica en tiempo real, todo ello con un respaldo automático en la nube.

El proceso de revisión de documentación era sencillo: una vez que cada integrante había completado su parte del documento, lo notificaba al resto del equipo. Los dos integrantes restantes realizaban, de forma secuencial, la corrección del documento. En la primera revisión se enfocaban en las correcciones más importantes, tales como coherencia, redacción y gramática. En una segunda instancia, se contemplaban con mayor precisión el cumplimiento de los estándares y otros aspectos más detallados.

Revisiones con el cliente

Estas instancias no eran revisiones formales, sino reuniones quincenales con el cliente establecidas en el proceso. Sin embargo, según la etapa del proyecto y los temas a tratar, estas reuniones se convertían en la alternativa a revisiones al solicitar de forma ocasional retroalimentación más detallada con expertos técnicos, quienes aportaron valiosos conocimientos sobre el proceso.

Estas instancias resultaron especialmente útiles en términos de arquitectura, ya que los expertos técnicos del cliente eran quienes tenían un conocimiento más profundo del sistema y podían ofrecer sugerencias cruciales sobre cómo la arquitectura del equipo podría interactuar con la suya. Estos aportes fueron vitales para la creación y evolución de las soluciones propuestas por los estudiantes.

También se recibieron aportes de proceso durante el segundo y tercer *release*, cuando el foco estaba en aumentar la frecuencia de los avances para obtener retroalimentación más rápidamente. En este sentido, los comentarios del cliente buscaron impulsar aún más las interacciones entre nuestro equipo y ellos, con el objetivo de dinamizar el proceso y acelerar la obtención de resultados.

Revisiones académicas de Universidad ORT Uruguay

Durante el transcurso del proyecto, se realizaron tres revisiones en distintos momentos clave. La primera revisión tuvo lugar al inicio del proyecto, el 1° de agosto de 2022, cuando el equipo se encontraba en las primeras etapas de codificación después de haber completado la capacitación. En este punto, el Máster en Computación Rafael Bentancur realizó varios aportes importantes que el equipo no había considerado, incluyendo aspectos de la arquitectura del proyecto y sugerencias para mejorar la presentación final y defensa de la tesis.

La segunda revisión se llevó a cabo el 29 de septiembre de 2022 con el Máster en Ciencias de la Computación Gastón Mousques. En esta instancia, se enfocó en aspectos de documentación, procesos y presentación, particularmente sobre cómo presentar el problema, realizando aportes muy significativos.

Finalmente, la última revisión académica se realizó el 22 de diciembre con el Máster en Computación y Sistemas de Información Álvaro Ortas, con un enfoque en la presentación final y defensa del proyecto. En esta ocasión, el profesional proporcionó numerosos aportes relacionados con la explicación de los procesos y las justificaciones, así como otros aspectos de la documentación y la presentación final.

El equipo consideró muy positivas estas instancias. En todas ellas se lograron identificar puntos fuertes del proyecto, pero también se pudieron reconocer muchos otros aspectos en los que se debían trabajar para lograr mejoras y optimizaciones, hecho que permitió alcanzar tanto los objetivos del equipo como los personales. El detalle de estas instancias se encuentran en el Anexo 10.11 donde se podrán encontrar los principales puntos destacados por el equipo en cada instancia.

7.3.3 Validaciones

Durante el proyecto, el equipo mantenía instancias quincenales con el cliente para validar los requerimientos. En la primera instancia, se validaron las historias de usuario desarrolladas por el equipo, tomando en cuenta los diferentes *inputs* proporcionados por el cliente para su elaboración. El objetivo era asegurarse de que la idea fuera clara y cumpliera con la

funcionalidad esperada por el cliente, así como evitar ambigüedades o especificaciones poco claras.

Después de completar esta etapa, el equipo comenzaba a trabajar en el desarrollo de cada *release*. En esta fase, se priorizaba el avance en los conceptos y funcionalidades para que estuvieran completamente claros, especialmente en lo relacionado con el *backend*. Al mismo tiempo, se desarrollaban prototipos en Figma para el *release* correspondiente, los cuales se presentaban al cliente para validar principalmente los aspectos visuales de la interfaz de usuario y la experiencia de usuario (UI y UX, respectivamente por sus siglas en inglés), así como cualquier otro pequeño detalle de comportamiento esperado.

Los prototipos tenían una consigna implícita: el desarrollo de nuevas pantallas debía adaptarse al diseño actual del producto y contemplar no solo su aspecto visual, sino también algunos comportamientos deseados por el cliente. Como se explica en las secciones 4.5.2 y 4.5.7, los prototipos fueron de vital importancia para validar que el equipo iba por buen camino en la construcción del producto, y que los requerimientos habían sido correctamente trabajados.

Respecto a las validaciones funcionales, cuando el equipo entendía que el avance era lo suficientemente significativo, se planteaba una demo en las reuniones quincenales, para que el cliente pudiera ver los avances, evaluarlos e ir brindando *feedback* acorde a lo que iban viendo y las pruebas que se realizaban con ellos en el momento. Estas validaciones también resultaron importantes para que tanto el equipo como el cliente estuvieran alineados y de acuerdo en que el producto en construcción satisfacía sus necesidades y aportaba valor a su solución actual.

Finalmente, el equipo planteó instancias de Pruebas de Aceptación de Usuario (*User Acceptance Testing* o UAT por sus siglas en inglés), dado que el equipo lo entendía como necesario por diferentes motivos, por ejemplo el alto entendimiento del *Product Owner* (persona que formaba parte del equipo cliente) sobre el negocio, y cuya visión, experiencia y pruebas podrían aportar alguna retroalimentación particular que el equipo no hubiese tenido en consideración al momento del desarrollo. Lamentablemente, y por motivos que se explicarán más adelante en el documento, estás pruebas no pudieron llevarse a cabo dentro del plan de testing.

7.3.4 Plan de testing

Al igual que con el plan de calidad, la naturaleza del proyecto como MVP llevó a modificaciones en el plan de testing. Aunque ambos planes están relacionados, no necesariamente deben verse afectados por los mismos factores, ya que el testing es solo uno de los módulos contemplados en el plan de calidad. Dado el tipo de proyecto y el tiempo disponible, el equipo de desarrollo decidió enfocarse en pruebas unitarias para la API, pruebas exploratorias funcionales bajo el concepto de Agile Testing, así como pruebas de rendimiento y de aceptación del usuario. Desafortunadamente, no se pudieron ejecutar las últimas dos pruebas durante el proceso. A continuación, se describirán los distintos tipos de pruebas, la estrategia seleccionada para abordarlas y los resultados obtenidos.

Pruebas unitarias

Con el objetivo de garantizar una mayor calidad en el código, especialmente en el módulo donde se esperaban cambios significativos en la lógica, refactorizaciones y nuevas funcionalidades, se llevaron a cabo pruebas unitarias. Estas pruebas se centraron en el módulo DocumentGateway utilizando la técnica de desarrollo conocida como TDD (*Test Driven Development*), la cual implica que el desarrollo del código sea guiado por las pruebas. Para llevar a cabo estas pruebas, se utilizó el marco de pruebas Xunit y se incorporaron los paquetes Nsubstitute y Bogus, los cuales se explican detalladamente en el Anexo 10.5 del documento de mantenimiento. Estas pruebas permitieron guiar el desarrollo de la lógica y asegurar su calidad desde una etapa temprana del proyecto.

Inicialmente, las pruebas unitarias se ejecutaban manualmente por parte del desarrollador que estaba trabajando y realizando cambios, con el objetivo de identificar cualquier impacto negativo que pudiera tener sobre el código existente. Sin embargo, posteriormente se configuraron las mismas a través de GitHub Actions en el repositorio, para que pudieran ejecutarse automáticamente al solicitar un *pull request* (PR) hacia la rama Develop o Main. Esta configuración se realizó con la finalidad de contar con una instancia adicional de revisión antes de la aprobación del PR, evitando posibles detalles que pudieran pasar desapercibidos por el equipo en estos chequeos. En la sección 6.8.4 del documento se proporciona más información

sobre esta configuración y su importancia en el proceso de aseguramiento de la calidad del código (Figura 67).

```
public async Task Delete_File_Valid()
{
    string dummyId = Guid.NewGuid().ToString();
    documentDataAccess.DeleteDocumentAsync(Arg.Any<string>())
        .Returns(true);
    var response = await documentLogic.DeleteDocumentAsync(dummyId);
    Assert.True(response);
}
```

Figura 67: Ejemplo de prueba unitaria (lógica para borrado de documento)

El equipo estableció que no todos los módulos tendrían cobertura de pruebas, y que algunas clases tampoco. Aquellos módulos que se contemplaron bajo el desarrollo de TDD fueron los siguientes:

- **API:** módulo que contiene los controllers de las API, los modelos de entrada y salida de datos se desarrolló aplicando TDD, exceptuando las clases LogoBuilder utilizado para imprimir un logo en la inicialización del proyecto y ErrorModel, clase que establecía un modelo particular para un mensaje de error.

- **Dominio:** este módulo contiene las clases que definen los objetos referentes a la lógica de negocio propio de la solución, por lo que su correcta implementación es fundamental para el éxito del proyecto.

- **Logics:** este paquete contiene las clases de la lógica de negocio, encargadas de realizar las operaciones más complejas de transformación de datos. Al igual que el módulo Dominio, el desarrollo del paquete Logics se llevó a cabo aplicando TDD. De esta manera, se garantizó que las operaciones realizadas por las clases de este paquete cumplieran con los requerimientos establecidos y que se mantuvieran libres de errores.

Los módulos que quedaron por fuera de la cobertura de pruebas se deben a que no es necesario desarrollarlos siguiendo la metodología TDD. Esto se debe a que se trata de módulos o clases que contienen conexiones a bases de datos o a servicios externos como AWS, para los cuales se

recurre a librerías externas ya probadas. Desarrollar pruebas unitarias para estas clases sería una inversión de tiempo y esfuerzo no redituable para el equipo, especialmente considerando que estos módulos y clases serán probados más adelante a nivel funcional y de integración. Por lo tanto, el equipo decidió enfocar sus esfuerzos de pruebas en aquellos módulos que son críticos para la funcionalidad de la aplicación y que contienen lógica de negocio compleja.

Al desarrollar utilizando TDD, se logró alcanzar una cobertura de aproximadamente del 100% en la mayoría de los módulos, lo que significa que cada línea de código fue probada exhaustivamente. Sin embargo, en aquellos módulos donde la metodología TDD no se utilizó, la cobertura fue menor debido a que no se dedicó tanto esfuerzo en la creación de pruebas unitarias.

El equipo estableció un mínimo aceptable de cobertura del 85% para considerar el código de calidad. En etapas más avanzadas del proyecto, si el tiempo lo permitía, se buscó aumentar la cobertura de las pruebas unitarias para alcanzar el 100% (Figuras 68 y 69).

Name	Covered	Uncovered	Coverable	Total	Line coverage	Covered	Total	Branch coverage
+ Api	282	23	305	715	92.4%	74	74	100%
+ DataAccess	0	130	130	195	0%	0	24	0%
+ Domain	61	0	61	83	100%	4	4	100%
+ IDataAccess	12	0	12	25	100%	0	0	
+ ILogics	173	0	173	246	100%	62	68	91.1%
+ Logics	177	0	177	275	100%	27	30	90%
+ Utility	19	162	181	428	10.4%	6	22	27.2%
+ WebApiServiceFactory	0	52	52	145	0%	0	4	0%

Figura 68: Captura del estado final de las pruebas unitarias

− ILogics	173	0	173	246	100%	62	68	91.1%
− ILogics	173	0	173	246	100%	62	68	91.1%
ILogics.BusinessLogicException	5	0	5	12	100%	0	0	
ILogics.DocumentDTO	16	0	16	27	100%	7	8	87.5%
ILogics.ErrorCode	11	0	11	24	100%	0	0	
ILogics.ParsedDocumentDTO	72	0	72	89	100%	22	22	100%
ILogics.QueryDTO	20	0	20	31	100%	15	20	75%
ILogics.TaskDTO	49	0	49	63	100%	18	18	100%
− Logics	177	0	177	275	100%	27	30	90%
− Logics	177	0	177	275	100%	27	30	90%
Logics.DocumentLogic	46	0	46	72	100%	6	6	100%
Logics.ParsedDocumentLogic	59	0	59	90	100%	9	12	75%
Logics.TaskLogic	72	0	72	113	100%	12	12	100%

Figura 69: Desglose de ejemplo de cobertura en ILogics y Logics

Pruebas funcionales - exploratorias

A lo largo del capítulo se ha mencionado el concepto de Agile Testing en varias ocasiones. Si bien esta metodología se puede aplicar a varios tipos de pruebas, su enfoque principal es en las pruebas funcionales y las diferentes formas de realizarlas.

Agile Testing es una técnica que permite adaptar el proceso de pruebas a los marcos ágiles de trabajo, fomentando una colaboración continua entre los distintos equipos que trabajan en el producto. El objetivo es poder adaptarse rápidamente a los cambios, identificando los requisitos de prueba en etapas tempranas. De esta manera, se busca establecer y definir los marcos sobre los cuales el equipo trabajará para probar el producto. El foco está en asegurar la calidad y evitar la inclusión de defectos.

Por estos motivos, el equipo llegó a la conclusión de que aplicar Agile Testing era la mejor opción para el proyecto, a pesar de ser un equipo pequeño. Con esta decisión en mente, se comenzó la planificación para implementar esta metodología. Teniendo en cuenta que el producto final sería un MVP, se optó por utilizar pruebas exploratorias, una técnica comúnmente utilizada en esta metodología, en lugar de un enfoque formal.

Para estas pruebas, el equipo realizó documentos de exploración como los ejemplos del Anexo 10.12 en donde se establecía quién era el probador (*tester*) de la sesión, la fecha de la prueba, hora de inicio y fin, el sistema operativo utilizado y navegador en caso de ser necesario (por ejemplo, no era requerido si se iba a probar únicamente algún aspecto lógico de AWS).

Además de los aspectos mencionados, se estableció una misión general y objetivos específicos para cada sesión de prueba exploratoria. Al final de cada sesión, los miembros del equipo registraban los defectos encontrados y comportamientos inesperados en una planilla para discutirlos en la siguiente reunión y definir los próximos pasos a seguir. Si se identificaba un *bug*, se registraba en Jira de acuerdo con la gestión de errores detallada en la sección 7.3.5.

Además de estas sesiones, el equipo elaboró una *suite* de pruebas en Postman (Figura 70), para realizar las pruebas funcionales de la API, en donde se encontraban divididas por carpetas las diferentes funcionalidades, y dentro de cada una de ellas, se encontraban las consultas a la API.

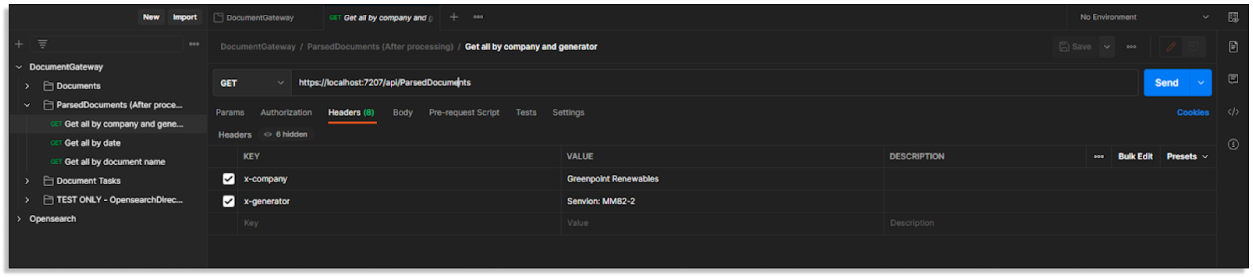


Figura 70: Ejemplo de colección de pruebas en Postman

Es importante señalar que durante las sesiones exploratorias también se realizaron pruebas en el *backend* y las API, pero se enfocaron en la integración del *frontend* con el *backend* y los servicios de AWS.

Pruebas performance

Como se mencionó al inicio de la sección, el equipo contempló estas pruebas teniendo en cuenta un requisito no funcional establecido por el cliente, que se centraba en el atributo de calidad mencionado (performance, RNF 2). Si bien el equipo cumplió con el RNF establecido por el cliente, que especificaba que un documento no debía demorar más de 24 horas en ser procesado, y que se logró que tardara menos de 60 segundos, lamentablemente no se pudieron realizar las pruebas adicionales correspondientes, debido a limitaciones de tiempo. El equipo consideraba que estas pruebas eran de gran valor, ya que proporcionarían datos precisos sobre los tiempos de procesamiento de los documentos, así como sobre los tiempos de respuesta de otros módulos (como la búsqueda de documentos y la creación de tareas) en situaciones de alta demanda por parte de los usuarios.

Para estas pruebas, el equipo había definido la utilización de Apache JMeter, herramienta con la cual un integrante del equipo estaba familiarizado, y consideraba que era la apropiada para el proyecto. En esta herramienta se desarrollarían los *scripts* que permitirían simular las distintas cargas y elaborar pruebas de estrés y de picos, entendiendo que podía contemplar alguna situación puntual, posible de materializarse en el futuro.

Después de analizar la situación con el cliente, se llegó a un acuerdo en el que se determinó que, debido a otras prioridades del proyecto, no era necesario realizar estas pruebas adicionales. Todo

quedó debidamente pactado con Renovus y se aseguró que el equipo cumplió con los requisitos no funcionales establecidos por el cliente en cuanto al tiempo de procesamiento de documentos y sus búsquedas. A pesar de no haber llevado a cabo estas pruebas adicionales, el equipo confía en que el proyecto se ha desarrollado de manera sólida y cumpliendo con los estándares de calidad esperados.

Pruebas de Aceptación de Usuario (UAT)

Durante el proyecto, por cuestiones de tiempo y priorización, no se llevaron a cabo las pruebas de aceptación de usuario, al igual que las pruebas de performance. Si bien el equipo tenía acceso a los usuarios finales de la solución, no se pudo aprovechar la valiosa experiencia que podrían haber proporcionado al no haber ejecutado estas pruebas.

La metodología de pruebas estipulada era Agile Testing, y se pretendía obtener información a partir de escenarios presentados a los usuarios para ejecutar. Sin embargo, debido a que los usuarios podrían tener una amplia experiencia en estos sistemas, se decidió que además de los escenarios, cada usuario tendría un tiempo limitado de alrededor de 60 minutos para probar libremente el sistema y anotar cualquier hallazgo encontrado.

Para registrar los hallazgos, se había contemplado la utilización de la herramienta Jira, que el equipo ya utilizaba para la gestión del proyecto. De esta manera, el equipo podría trabajar sobre los hallazgos reportados por los usuarios y realizar optimizaciones y correcciones de errores en el producto.

A pesar de la decisión de no ejecutar las pruebas de aceptación de usuario y las pruebas de performance, se acordó con el cliente que, debido al plazo del proyecto y a las exigencias del mismo, estas pruebas no se llevarían a cabo.

7.3.5 Gestión de errores

En esta sección se describe el proceso de gestión de errores que se llevó a cabo durante el proyecto, desde la identificación y registro hasta la resolución de los mismos. El control y monitoreo de estos errores fue crucial para permitir al equipo entregar un producto robusto en el tiempo acordado.

Inicialmente, el equipo tenía previsto utilizar Jira únicamente para registrar los hallazgos propios. Sin embargo, tras analizar la falta de experiencia en testing y prácticas relacionadas por parte de algunos miembros del equipo, se decidió agregar un paso intermedio antes del registro en Jira para evitar falsos positivos. Este paso consistía en completar una planilla elaborada por el equipo para discutir en una reunión y corroborar el hallazgo antes de pasarlo a Jira, priorizándolo para el siguiente Sprint.

Para asegurar una gestión efectiva de errores, se instruyó al Product Owner y a todos los usuarios del sistema a utilizar Jira para registrar los hallazgos. De esta manera, se pudo centralizar la gestión de errores en una única herramienta y mantener un seguimiento más eficiente del estado de los mismos. Estas medidas permitieron que el equipo gestionara eficazmente los errores y pudiera entregar un producto de calidad en el plazo acordado.

Registro de errores

Para un registro más detallado de los defectos, el equipo estableció una escala de severidad y otra de prioridad. De esta forma, se logró una doble filtración que permitió priorizar las tareas en cada Sprint. Las escalas utilizadas por el equipo fueron las siguientes:

Severidad

- **Bloqueante:** dentro de esta categoría se incluyen aquellos defectos que no permitieran continuar con el uso del sistema.
- **Alto:** se refiere a aquellos defectos que están relacionados con una funcionalidad y/o característica que no cumple con los requisitos establecidos, lo que hace que su comportamiento se aleje significativamente de lo esperado.
- **Medio:** dentro de esta categoría se incluyen aquellos defectos donde una funcionalidad y/o característica no cumple completamente con los requisitos establecidos y puede presentar desviaciones de lo esperado pero con un impacto mínimo.
- **Bajo:** dentro de esta categoría se incluyen por ejemplo los defectos cosméticos cómo errores ortográficos, problemas de alineación, fuente, entre otros.

Prioridad

- **Crítica:** dentro de esta categoría se incluyen aquellos defectos más severos que requerían una corrección rápida, dado que tienen un impacto grande sobre el producto en desarrollo.

- **Alta:** dentro de esta categoría se incluyen aquellos defectos que el equipo entendiera vital su corrección previo a una salida o *release*.

- **Media:** dentro de esta categoría se incluyen los defectos que de ser posible deberían corregirse previo a una salida o *release*, pero que están en discusión, ya que su impacto no es tan severo.

- **Baja:** dentro de esta categoría se incluyen aquellos defectos que no es necesario corregir previo a una liberación debido a que su impacto y severidad es extremadamente bajo, pero que sí deben estar corregidos previo a la salida final del producto.

Además de la severidad y la prioridad, quien reportase el error debía aportar, un título para el error, una descripción clara de que estaba ocurriendo, las precondiciones de la ejecución en caso de aplicar, los pasos a seguir para su reproducción, el resultado actual, el esperado, y la evidencia del defecto. Adicionalmente, el defecto registrado se vinculó a la historia de usuario relacionada con el error, y luego el equipo designó a un encargado para su corrección. Por último, se estipuló el Sprint en el cual se debía solucionar el error.

Seguimiento de errores

Para facilitar el seguimiento de los errores, el equipo definió las etapas por las cuales puede pasar, siendo esta indicada en la tarjeta de Jira. El proceso definido para los defectos es el siguiente (Figuras 71 y 72):

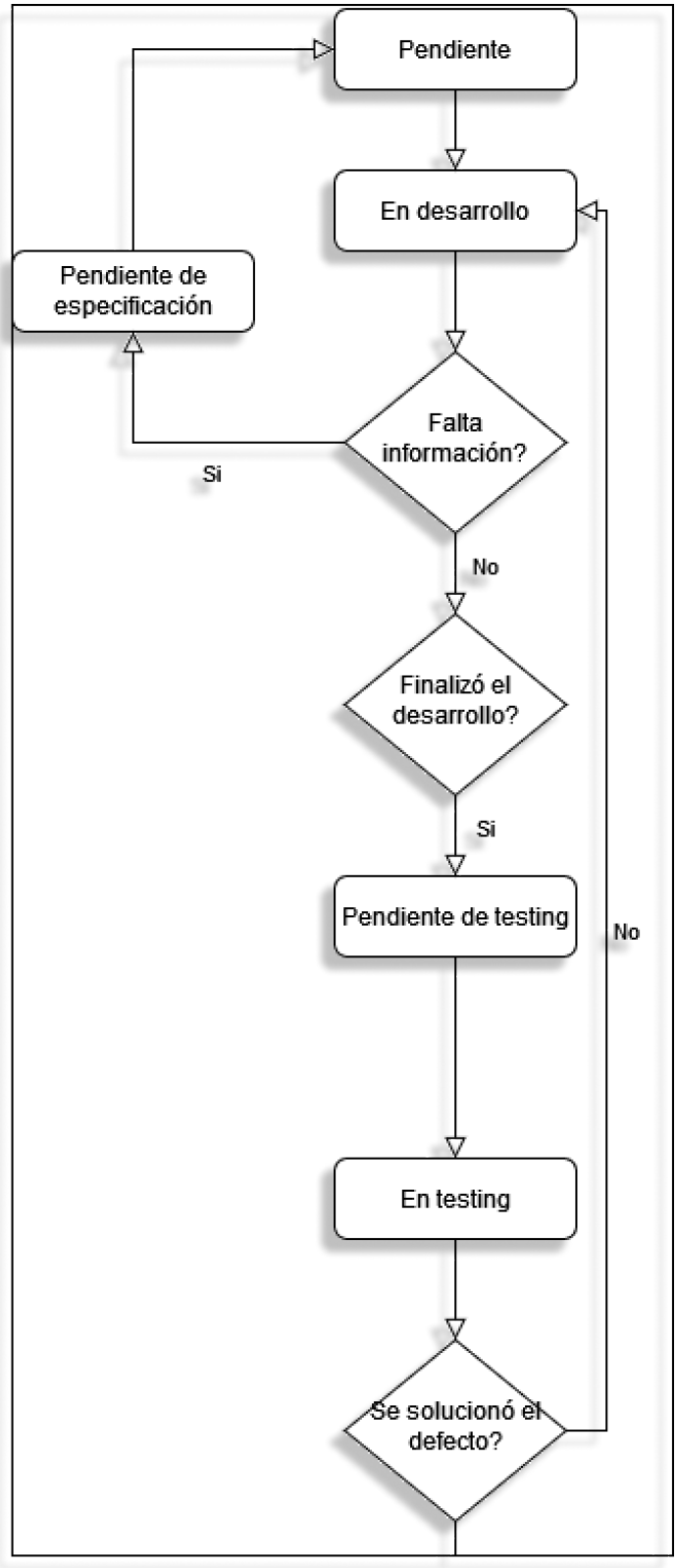


Figura 71: Primera parte del proceso de seguimiento de errores

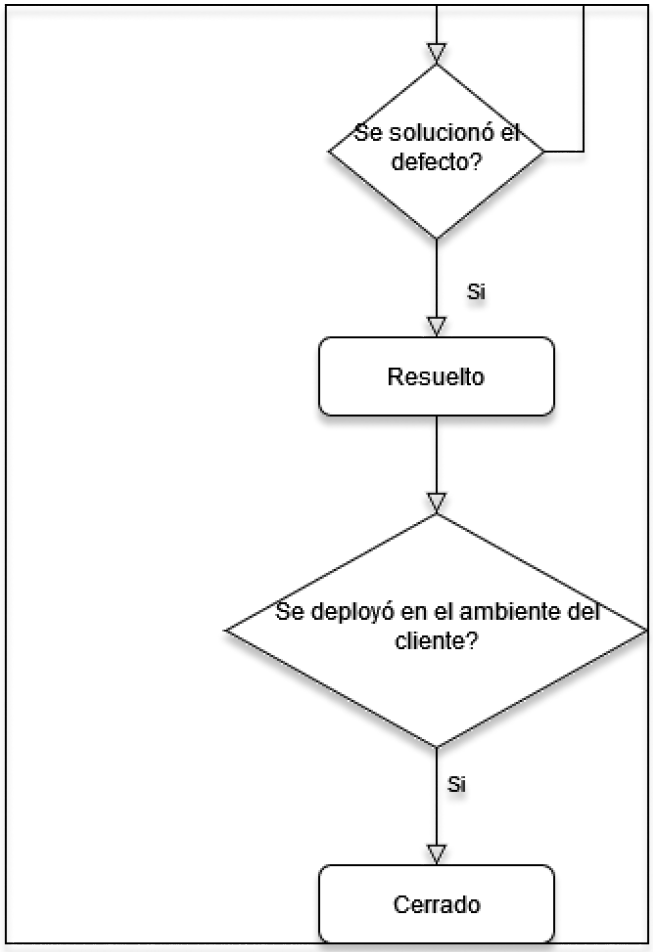


Figura 72: Segunda parte del proceso de seguimiento de errores

Con la implementación de este proceso, el equipo logró una gestión efectiva de los defectos hallados y su tratamiento a lo largo del proyecto. La utilización de Jira resultó fundamental en este sentido, ya que ante cualquier cambio de estado en los *tickets* de defectos reportados, los integrantes del equipo involucrados eran notificados y podían visualizar los estados dentro del tablero de la herramienta, lo que ayudó aún más al seguimiento de los mismos.

Una vez que el defecto era solucionado y alcanzaba el estado de "resuelto", el equipo lo mantenía hasta realizarse el *merge* de la rama en la que se estaba trabajando a la rama de desarrollo (Develop) y se efectuara el despliegue correspondiente en los entornos del cliente. En este momento, el estado del *ticket* se cambiaba a "cerrado" y se daba por finalizado todo lo relacionado con dicha tarea

7.4 Métricas

En esta sección, se tratará el tema de las métricas, un aspecto crucial en el ámbito de la calidad del *software*. Las métricas son una herramienta fundamental para evaluar el cumplimiento de aspectos relacionados tanto con el proceso como con el producto en cuestión. A lo largo del documento, se han mencionado diferentes métricas, tanto de proceso como de producto, explicando su valor para el equipo y su impacto en el trabajo realizado.

Se proporcionará una breve explicación de los aspectos de calidad relacionados con cada una de estas métricas, y se explicará el objetivo por el cual fueron incluidas en el documento.

7.4.1 Métricas del proceso

Estas métricas se establecen bajo la necesidad de comprender si el proceso que se está implementando es correcto, si necesita ajustes, dónde están los puntos débiles y los que requieren de una optimización.

Cantidad de defectos por Sprint

El control de la cantidad de defectos por Sprint es un monitoreo importante para poder evaluar la eficiencia del equipo a lo largo del tiempo (Figura 73). Esta gráfica se puede comparar con el detalle de los Sprints del Anexo 10.6, para comprender mejor la situación Sprint a Sprint y obtener aún más información de la eficiencia del equipo.

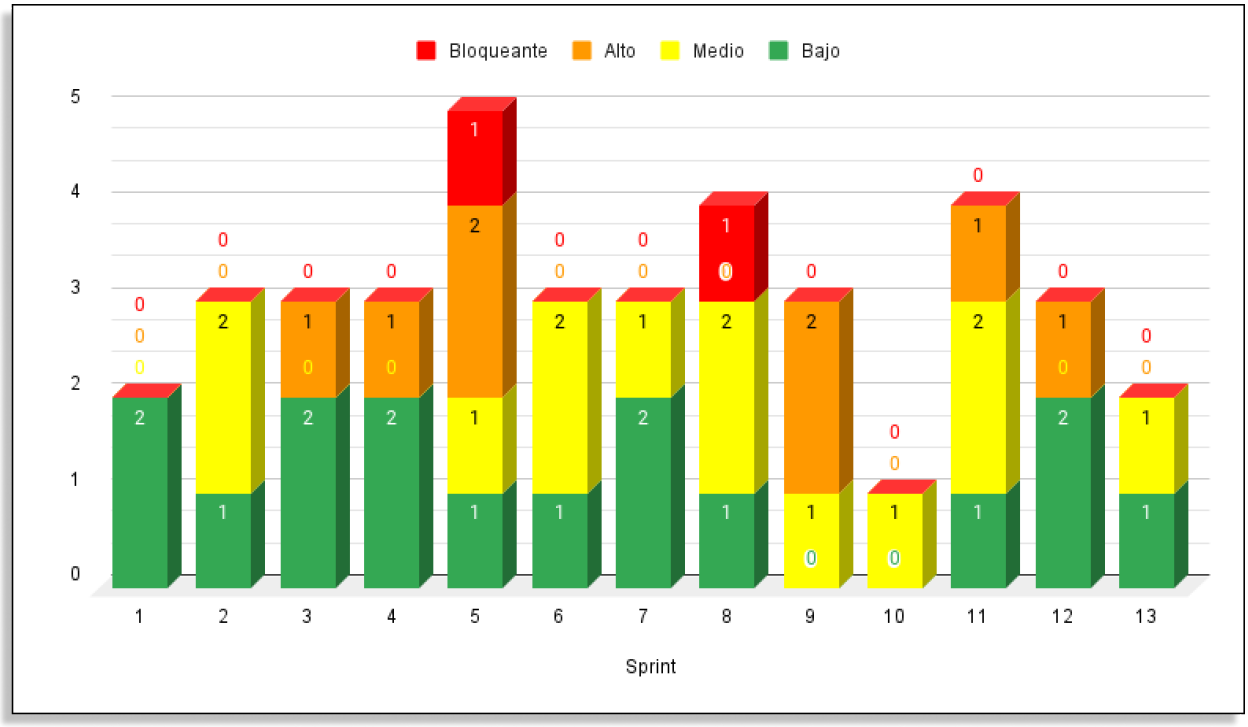


Figura 73: Gráfica de defectos por Sprint

Se puede observar cómo los Sprints de mayor complejidad son aquellos que reflejan mayor cantidad de defectos, e incluso donde se pueden visualizar los únicos dos defectos bloqueantes del proyecto. Como se comentó en la sección 7.3.5, el equipo priorizaba siempre la corrección de los defectos de severidad alta y bloqueante, pudiendo ocurrir que algún defecto de severidad media o baja pudiera "arrastrarse" entre Sprints, pero si esto ocurría la idea era que el defecto se corrigiera, como tarde, un Sprint de corrido luego de su identificación y registro.

Desviación de story points realizados vs. estimados

La métrica de desviación de puntos realizados contra estimados es ampliamente utilizada en equipos que trabajan bajo marcos ágiles. Su objetivo es ayudar al equipo a comprender mejor el proceso y detectar si se han asumido compromisos que no pueden cumplirse, si se han materializado riesgos, o si no se han tenido en cuenta otros aspectos que podrían afectar el compromiso del equipo, como por ejemplo una tendencia a estimaciones erróneas. En la sección 6.6.3 se explicó esta métrica en detalle, presentando gráficos y un análisis más detallado que destaca sus beneficios y aportes a la gestión del proyecto.

Velocidad ágil

Al igual que la métrica anterior, ésta es bastante estándar para los equipos de proyectos que trabajan bajo marcos ágiles. La velocidad ágil le permite al equipo, luego de algunos Sprint, comenzar a entender mejor su velocidad por iteración, en otras palabras, la cantidad de *story points* aproximados que pueden cumplir por Sprint.

Este valor es fundamental para que el equipo pueda conocer sus capacidades y comprometerse de manera más efectiva con los hitos de cada iteración, asegurando el cumplimiento de los estándares de calidad establecidos en el tiempo acordado. Es importante destacar que no debe haber una exigencia excesiva que pueda generar inconvenientes tanto en el equipo como en el producto a desarrollar. La métrica correspondiente se detalla en la sección 6.6.2, donde se presenta la gráfica de evolución de la velocidad del equipo.

7.4.2 Métricas del producto

Estas métricas son esenciales para que el equipo pueda verificar que la construcción del producto se ajuste a los objetivos de calidad establecidos y garantice el cumplimiento de los atributos de calidad definidos para la solución.

Complejidad ciclomática

Se decidió realizar el cálculo de la complejidad ciclomática del proyecto para las funciones Lambda de AWS, debido a que el equipo tenía menos experiencia en estos módulos en comparación con otros de la solución. El cálculo de esta métrica resultó crucial para comprender el trabajo del equipo y evaluar el grado de dificultad de mantenimiento. Además, esta métrica permitió tener en cuenta la posibilidad de agregar nuevas funcionalidades en el futuro, como la interpretación de otros formatos de documentos.

Para evaluar esta métrica, el equipo se basó en el estándar definido por Microsoft [55]. Según este estándar, un proyecto se considera exitoso y mantenible cuando la complejidad ciclomática se sitúa en un valor entre 1 y 10, siendo mejor cuanto más bajo sea el valor. El cálculo de la complejidad ciclomática se realizó utilizando la herramienta Pygenie [56], que permite su cálculo específicamente para proyectos elaborados en Python.

A modo de ejemplo se presenta la siguiente gráfica que refiere a los resultados de la complejidad ciclomática presente en las funciones Lambda: Analyzer y Saver (Figura 74).

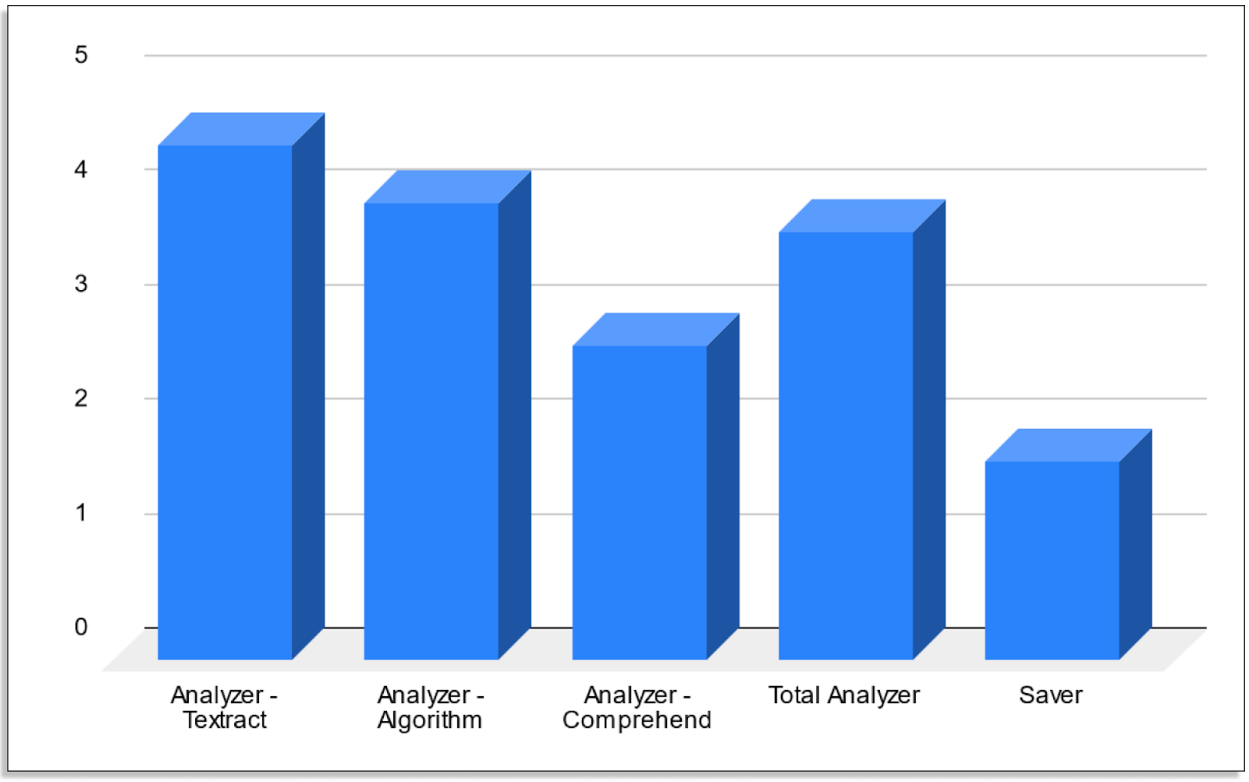


Figura 74: Gráfica de complejidad ciclomática módulos AWS

Como se puede observar, el valor de la complejidad ciclomática no superó el umbral de 5, lo que se considera altamente satisfactorio. Este resultado es importante en términos del atributo de calidad de mantenibilidad, ya que, como se mencionó anteriormente, a menor valor de complejidad ciclomática, menor complejidad tiene el módulo y, por lo tanto, resulta más fácil de mantener en el futuro.

Índice de mantenibilidad

El índice de mantenibilidad es una métrica que se vincula al atributo de calidad de mantenibilidad, el cual indica, bajo las pautas establecidas por Microsoft [57], que a mayor valor, el código construido es más mantenible. La escala se divide en tres partes: de 0% a 9%, el código se considera imposible de mantener, y se necesita una gran inversión de tiempo y esfuerzo para transformarlo en mantenible; del 10% al 19%, el código se considera difícil de mantener, y se

necesita una mayor inversión de tiempo para hacerlo, pero alcanzando un mínimo para que sea mantenible en el tiempo; del 20% al 100%, cuanto mayor sea el valor, más mantenible va a ser el código, y menor el esfuerzo requerido para lograrlo.

Para garantizar la mantenibilidad del sistema y su facilidad, el equipo decidió aplicar este índice sobre el proyecto de la API y lógica del sistema, estableciendo que el mínimo valor aceptable por proyecto era de 65%.

Se utilizó el Visual Studio 2022 y el módulo que trae incluido para el cálculo de métricas sobre el código. Con esta métrica, se busca medir la capacidad del código para ser mantenido y mejorado con el menor esfuerzo posible (Figura 75).

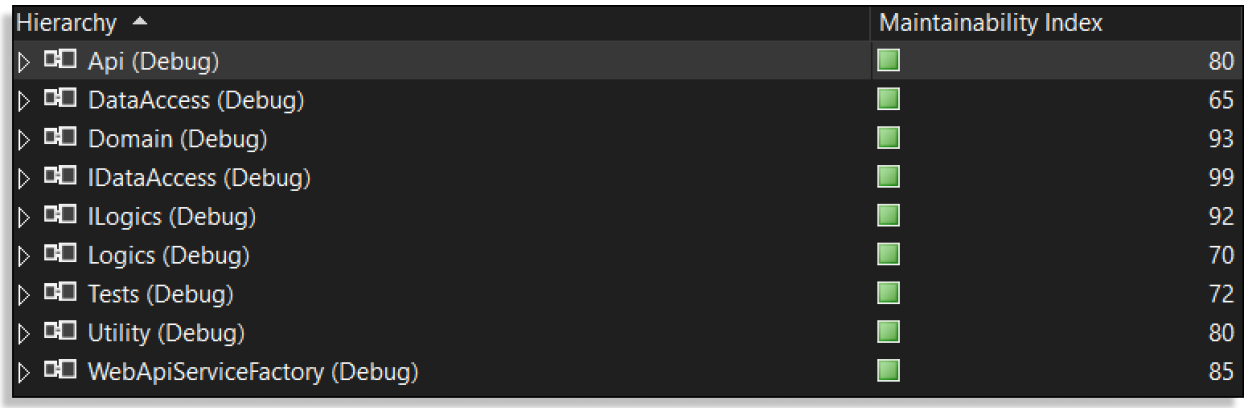


Figura 75: Índice de mantenibilidad

Como se puede observar, el equipo logró cumplir con los objetivos establecidos en cuanto al índice de mantenibilidad, asegurando así la construcción de un código que sea fácilmente mantenible. Se puede apreciar en la imagen que el paquete con menor índice es el del DataAccess, lo cual era esperable debido a sus dependencias con librerías externas. No obstante, el equipo logró alcanzar el mínimo establecido, lo que demuestra un esfuerzo para mantener la calidad del código en todos los proyectos.

Cobertura de pruebas unitarias

Como se mencionó anteriormente, el equipo adoptó la metodología de desarrollo guiado por pruebas (TDD) para el módulo de la API y la lógica del sistema, con el objetivo de mejorar la

calidad del código y aumentar la eficiencia del proceso de desarrollo. Se estableció un porcentaje mínimo de cobertura del 85% para considerar satisfactorio al producto.

Gracias a la aplicación de la técnica de TDD, el equipo logró cumplir con la meta establecida y, en una etapa de refactorización, incluso aumentar aún más el porcentaje de cobertura, alcanzando valores cercanos al 100%. Se detalla todo lo relacionado con las pruebas unitarias y su cobertura en la sección 7.3.4 del informe.

Satisfacción del cliente

Finalmente, la satisfacción del cliente también se consideró una métrica clave. Esta métrica se midió a través de encuestas realizadas al final de cada *release*, donde el cliente brindaba su opinión sobre diferentes aspectos del producto.

El equipo analizó cuidadosamente las respuestas y las utilizó para encontrar oportunidades de mejora. En total se realizaron tres encuestas, todas con buenos resultados. Sin embargo, la velocidad de entrega fue identificada como un área de mejora potencial, con una valoración de 4 sobre 5.

El equipo trabajó para hacer mejoras en este aspecto, basándose no solo en los comentarios de las encuestas, sino también en las conversaciones con el cliente. Para obtener más detalles sobre las encuestas y sus resultados, consulte el Capítulo 6 y el Anexo 10.7.

8. Conclusiones y lecciones aprendidas

Este capítulo tiene como finalidad dar un cierre al proyecto y al documento, analizar los objetivos establecidos en relación a su cumplimiento y realizar una reflexión sobre el proyecto.

8.1 Análisis de objetivos planteados

En el primer capítulo, en la sección 1.2, el equipo presenta un conjunto de objetivos que se deseaban cumplir al finalizar el proyecto. A continuación se encuentran listados según sus categorías.

8.1.1 Académicos

Lograr la aprobación del proyecto final de carrera.

El equipo no se encuentra en condiciones de determinar si este objetivo fue logrado ya que al momento de la producción de este documento se desconoce cuál será el resultado que tendrá la evaluación final. Sin embargo, el equipo concluye que dió todo de sí para lograr este objetivo, encontrándose satisfechos con el resultado obtenido.

Aplicación de conocimientos relacionados con gestión y arquitectura de *software* adquiridos durante la carrera en un contexto real.

El equipo pudo aplicar los conocimientos de manera práctica y pudo experimentar las complejidades y desafíos de aplicar estos conocimientos en un entorno real. El hecho de que se haya tenido que consultar apuntes y bibliografía manejada a lo largo de la carrera en las diferentes asignaturas de la Universidad, así como recurrir a materiales de referencia durante todo el proceso realizado, demuestra un esfuerzo consciente para aplicar conocimientos teóricos en un entorno práctico, así como alcanzar una adecuada integración de saberes.

El proceso de aplicación de los conocimientos ayudó al equipo a consolidar su comprensión de los mismos y a desarrollar habilidades en gestión y en arquitectura de *software*. La experiencia práctica también permitió al equipo discernir mejor los conceptos y enfoques que habían sido limitados a la teoría en el ámbito académico.

8.1.2 Personales

Aprender a utilizar nuevas tecnologías y expandir nuestro conocimiento.

Para cumplir con lo esperado por el cliente el equipo se propuso aprender a usar tecnologías *cloud* que lo ayudarán a lograr esto. Desde un comienzo buscando el entendimiento de Azure a una profundización en el portafolio de servicios de AWS, el equipo invirtió buena parte del tiempo abocado al proyecto, en expandir su conocimiento en diversas áreas del mundo "en la nube".

Lenguajes nuevos como Python se incorporaron al proceso de aprendizaje y se utilizaron para la construcción de soluciones específicas del proyecto. Además, se estudió el lenguaje de consultas a la base de datos de OpenSearch para poder manejar eficientemente los datos.

Durante el proceso de aprendizaje, el equipo se dedicó a estudiar a fondo la documentación técnica de cada tecnología y servicio utilizado, y a aplicar las mejores prácticas para garantizar la calidad del *software*. Asimismo, se exploraron y se realizaron estudios de otras tecnologías que no fueron finalmente utilizadas en el proyecto, pero que aportaron al entendimiento de las posibilidades tecnológicas y a la toma de decisiones estratégicas.

En todo momento, el equipo estableció parámetros claros para medir el éxito del aprendizaje y la utilización de nuevas tecnologías. Estos parámetros se evaluaron de manera constante para asegurar que el trabajo esté alineado con los objetivos del proyecto y se mantenga un alto nivel de calidad en todo momento. Ello permitió al equipo construir una solución que cumpla con las expectativas del cliente y que se encuentre en línea con los estándares definidos en el plan de SQA.

Lograr una dinámica de trabajo en la que se obtenga un promedio mayor o igual a 3 en la evaluación interna del trabajo en equipo por cada integrante.

Dados los buenos resultados de las encuestas Team Pulse se da como logrado el objetivo. Como se detalla en el Capítulo 4, sección 4.2.2, el equipo está compuesto por miembros que tienen una relación que va más allá de lo académico, por lo que conocen y confían plenamente en sus

habilidades. Aunque surgieron algunas instancias de tensión menores durante el proyecto, el promedio de evaluación en las encuestas fue de 4,6 de un máximo de 5.

Si bien este valor sugiere que hubo algunos momentos donde el relacionamiento no fue el ideal, estos pueden atribuirse a la ansiedad que genera un proyecto académico de este porte. Por otro lado, dado este promedio tan positivo, puede concluirse que las instancias de tensión fueron mínimas y que el equipo logró una excelente performance para lograr sus objetivos a nivel personal y académico.

Aportar a una empresa cuyo negocio es el de mejorar la gestión de parques de energías renovables.

A través de la ejecución del proyecto, el equipo ha logrado desarrollar una solución que permite mejorar la eficiencia en la gestión de parques de energías renovables, lo que ha sido altamente valorado por el cliente, como se puede apreciar en el Anexo 10.7.

Para el equipo, ha sido un gran logro poder contribuir al desarrollo de un negocio que busca promover la sustentabilidad en el mundo de la energía, lo que ha generado una gran satisfacción y sentido de propósito en el mismo.

8.1.3 Producto

Construir un producto que satisfaga las necesidades del cliente establecidas en los requerimientos.

A través del proceso de Ingeniería de requerimientos, el equipo ha logrado identificar y comprender de manera efectiva las necesidades del cliente, lo que ha permitido desarrollar soluciones que se ajustan a sus requerimientos y expectativas. Esto último se puede apreciar en las encuestas de satisfacción que denotan que el cliente ha quedado satisfecho con el trabajo realizado.

Si bien es cierto que se podrían haber mejorado los tiempos de entrega, el equipo se siente satisfecho con los resultados obtenidos y toma esta satisfacción del cliente como un indicador de éxito del proyecto.

Generar los entregables acordados con el cliente

El equipo identificó los entregables, creó un plan detallado y estableció plazos junto a responsables para cada tarea. Asimismo, mantuvo una comunicación efectiva con el cliente, implementó un proceso de control de calidad y monitoreó constantemente el progreso del proyecto para cumplir con el objetivo y entregar todo lo acordado con el cliente en tiempo y forma, asegurándose de cumplir con todas las expectativas y requerimientos establecidos en el proyecto.

Desarrollar una solución que cumpla los parámetros establecidos en QA

Al desarrollar la solución, el equipo se aseguró de cumplir con los objetivos establecidos en cada etapa del proceso, desde la ingeniería de requisitos hasta la investigación y la arquitectura del sistema. Se verificaron los estándares de calidad en cada paso del proceso, lo que garantizó que se cumplieran los parámetros establecidos en los procesos de control de calidad.

La solución final se evaluó de acuerdo con los estándares de calidad y se concluyó que cumple con los requisitos establecidos en el plan de calidad. El equipo logró desarrollar una solución de alta calidad que satisface los criterios y estándares establecidos.

8.1.4 Cliente

Adicionalmente el cliente tenía un conjunto de expectativas y objetivos, planteados en la sección 2.4, que buscaban resolver el problema planteado al comienzo del proyecto y que fueron ajustándose a medida que se fue avanzando en el desarrollo del producto. A continuación se realiza el análisis de los objetivos:

Solucionar el problema de la gestión de grandes documentos generados por los parques eólicos

Gracias a la implementación de un sistema robusto, se logró solucionar el problema planteado. El sistema fue diseñado para soportar grandes volúmenes de datos, alojándolo en herramientas escalables como ElasticBeanstalk y otras del portafolio de AWS (Capítulo 5) con políticas de

autoescalado, lo que permitió la gestión eficiente de los documentos de los generadores en tiempo real.

Además, se brindó la posibilidad de escalar el sistema para buscar sobre miles de documentos procesados en segundos, lo que aumentó significativamente la velocidad y la precisión en la búsqueda y análisis de los datos. En caso de aumentar la demanda, el sistema se escalará automáticamente para asegurar una respuesta rápida y eficaz.

Desarrollar un sistema que permita cargar y organizar los archivos relacionados con los generadores de los parques eólicos

Para lograr el objetivo se diseñó una solución que permite organizar y agrupar los documentos al subirlos y mantener el orden en el *backend,* para que un administrador pueda consultarlos de forma sencilla. Además, se brindó al usuario la posibilidad de visualizar rápidamente sus documentos relacionados con sus aerogeneradores. Esta solución permitió mejorar la gestión de los documentos generados por los parques eólicos, facilitando su acceso y consulta para el personal.

Posibilitar la extracción de valores clave de los documentos para facilitar su análisis y búsqueda por parte de los responsables de los parques.

Mediante el uso de técnicas avanzadas de procesamiento de lenguaje natural y análisis de datos, apoyados por Textract y Comprehend se ha logrado la extracción exitosa de información relevante de los documentos. Los valores clave que se pueden extraer incluyen la fecha, el número de aerogenerador, las conclusiones y el texto plano.

Además, se ha brindado la opción de búsqueda sobre estos campos, tal como se describe en el Capítulo 3. Cada campo clave es considerado para un filtro, lo que significa que no hay información desaprovechada y que los responsables de los parques pueden realizar búsquedas precisas y eficientes.

Posibilitar la generación de tareas específicas para facilitar el estudio y seguimiento de los generadores de los clientes de Renovus.

Después de evaluar el objetivo referente a la generación de tareas, se determina que el sistema desarrollado permite a los responsables de los parques eólicos generar tareas específicas para cada aerogenerador en base a la información relevante extraída de los documentos. Esto, a su vez, permite un seguimiento más efectivo del estado de los generadores.

Además, el sistema ofrece la posibilidad de que los responsables puedan ver y realizar seguimientos de tareas tanto para un documento en particular como para un generador específico, lo que facilita aún más el seguimiento y la gestión de las actividades.

8.2 Lecciones aprendidas y reflexiones

Después de completar el proyecto, el equipo reflexionó sobre las lecciones aprendidas durante el mismo. Aunque todos estaban satisfechos con el resultado, coincidieron que el camino fue más difícil de lo que esperaban, particularmente en los procesos menos esperados, como fueron la gestión del proyecto y el control de calidad.

Los aspectos técnicos fueron los que inicialmente parecían más intimidantes, con un portafolio de servicios de Amazon de cientos de herramientas a considerar y un desafío programático que parecía muy difícil de atacar al principio, pero, cuando llegó la hora de la verdad hubo otros aspectos que probaron ser más trabajosos.

En retrospectiva, esto se debió a que el equipo se enfrentaba con *software* en su día a día, constantemente buscando soluciones creativas a desafíos de programación, pero no estaban acostumbrados a un proyecto real donde los retos se escapan de aspectos limitados a la codificación y se adentran en la gestión del proyecto en sí.

Trabajar con un cliente implica un cambio de mentalidad. No es sencillo transmitir ideas complejas y es difícil comprender las necesidades del cliente, por eso hay que adaptarse a usar herramientas como Design Thinking a fin de poder bajar a tierra los problemas. Además, es importante aprender a comunicarse de manera efectiva con ellos para entender sus expectativas.

Muchos de los riesgos planteados llegaron a manifestarse, pero gracias al plan de contingencia previamente establecido, el equipo pudo responder a los impactos negativos que fueron surgiendo en el desarrollo del proyecto. Esto resalta la importancia de la planificación y el seguimiento constante del plan de gestión de riesgos.

No se quiere desestimar lo difícil que fue aprender IA y AWS, ya que implicó muchas horas de estudio y dedicación. La documentación no siempre era clara y contenía errores, lo que obligó al equipo a improvisar en varios momentos del proyecto. Muchas de las herramientas seleccionadas por ser "fáciles de usar" resultaron ser más complejas de lo esperado, lo que subraya la necesidad de hacer una evaluación cuidadosa y exhaustiva de las herramientas antes de seleccionarlas y plantear escenarios alternativos, para situaciones inesperadas.

Los tiempos del proyecto fuera de lo habitual para los integrantes del equipo fueron otro gran desafío, ya que ninguno estaba acostumbrado a proyectos mayores a 6 meses. Adicionalmente a diferencia de los proyectos de la Universidad, se debió definir muchos de los procesos desde cero. Esta práctica sin embargo, se concluye como una de las más valiosas en cuanto a los aprendizajes realizados, ya que es algo que ningún integrante del equipo se había enfrentado hasta el momento.

Se aprendió mucho durante el proyecto, así como también preparar a los integrantes del equipo para enfrentarse en el futuro con nuevos desafíos. Uno de los mayores aprendizajes radicó en la comprensión de las fortalezas de cada uno de sus integrantes, así como de la sinergia que se podía generar en el trabajo colaborativo. Este aprendizaje, sin dudas acompañará a cada uno a lo largo de la vida.

En definitiva, todos los miembros del equipo quedaron muy conformes con lo logrado y consideraron que fue un gran espacio de aprendizaje y puesta en práctica de las herramientas adquiridas en la carrera de Ingeniería en Sistemas de la Universidad ORT Uruguay.

9. Referencias bibliográficas

[1] Renovus. “RENOVUS Energy Tech”. 2023. [Online]. Available: https://www.renovus.tech/.

[2] Centro de innovación y emprendimientos. “RENOVUS”. 2021. [Online] Available: https://cie.ort.edu.uy/emprendimientos/renovus.

[3] Agencia Nacional de Investigación e Innovación - Proyectos. “Renovus Energy Tech”. 2022. [Online] Available: https://anii.org.uy/proyectos/EI_X_2021_1_170319/renovus-energy-tech/.

[4] Danish Wind Industry Association. "Curva de potencia de un aerogenerador", 2003. [Online] Available: http://xn--drmstrre-64ad.dk/wp-content/wind/miller/windpower%20web/es/tour/wres/pwr.htm.

[5] Universidad ORT Uruguay. "Renovus - análisis de datos de parques eólicos", 2023. [Online]. Available: https://fi365.sharepoint.com/:w:/s/ORTsf2022s1/EQ1ubd996-JGszXf6m9QmLkBG3zRCrWZCpK59j2NxgyVTA?e=5hfB4l.

[6] I. Sommerville, I. Alfonso, and E. Al, *Ingeniería del software*. Madrid: Pearson Educación, 2005, ch.4, secs. 4.1-4.2.

[7] Agile Alliance. "Manifesto for Agile Software Development", 2001. [Online] Available: https://agilemanifesto.org/.

[8] Scrum.org. "What is Scrum?", 2023. [Online] Available: https://www.scrum.org/resources/what-is-scrum/ .

[9] Agile Alliance, “Scrumban”, julio 2021. [Online] Available: https://www.agilealliance.org/scrumban/

[10] Atlassian, “What is kanban?”, 2022. [Online] Available: https://www.atlassian.com/agile/kanban

[11] Atlassian, "Jira Software Overview", 2023. [Online]. Available: https://www.atlassian.com/software/jira .

[12] Scrum.org, "Professional Scrum Master I Certification", 2023. [Online]. Available: https://www.scrum.org/assessments/professional-scrum-master-i-certification .

[13] Scrum.org, "The Scrum Guide", 2020. [Online]. Available: https://www.scrum.org/resources/scrum-guide .

[14] Universidad ORT Uruguay, "Aulas - Universidad ORT Uruguay", 2023. [Online]. Available: https://aulas.ort.edu.uy .

[15] MIT Sloan School of Management, "Design Thinking Explained", 2017. [Online]. Available: https://mitsloan.mit.edu/ideas-made-to-matter/design-thinking-explained .

[16] Figma, Inc, "Figma: the collaborative interface design tool", 2022. [Online]. Available: https://www.figma.com/ .

[17] Agile Alliance, "BDD - Behavior Driven Development" 2015. [Online]. Available: https://www.agilealliance.org/glossary/bdd/.

[18] Agile Alliance, "What does INVEST Stand For", 2015. [Online] Available: https://www.agilealliance.org/glossary/invest/.

[19] Atlassian, "A Brief Overview of Planning Poker", 2021. [Online] Available: https://www.atlassian.com/blog/platform/a-brief-overview-of-planning-poker .

[20] Amazon Web Services, “Ejemplos y prácticas recomendadas de arquitecturas de referencia” , 2023. [Online] Available: https://aws.amazon.com/es/architecture/

[21] R. C. Martin, “Component Cohesion,” in *Clean architecture a craftsman's guide to software structure and Design*, Prentice Hall, 2018, pp. 119–146.

[22] W3Schools, “RESTful Web Services”, RESTful web services 2023. [Online]. Available: https://www.w3schools.in/restful-web-services/intro.

[23] D. Copeland , “Patterns of SOA: Background job”, 2017 [Online]. Available: https://multithreaded.stitchfix.com/blog/2017/05/31/patterns-of-soa-background-job/.

[24] J. Nielsen, “10 usability heuristics for user interface design.”, 1994 [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/.

[25] R. C. Martin, “Getting a SOLID start”, diciembre 2009. [Online]. Available: https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start.

[26] K. Larkin, S. Smith, and B. Dahler, “Dependency injection in ASP.NET Core.”, enero 2023. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-7.0.

[27] International Organization for Standardization, “ISO/IEC 25010”, 2023. [Online]. Available: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010.

[28] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley, 2013.

[29] Docker, “What is a container?”, febrero 2023. [Online]. Available: https://www.docker.com/resources/what-container/.

[30] Microsoft, “Language-integrated query (LINQ) (C#)”, agosto 2023. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/.

[31] Microsoft, “Nuget and .net libraries”, marzo 2022. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/standard/library-guidance/nuget.

[32] B. Chavez, “Bogus for .NET: C#, F#, and VB.NET”, 2023. [Online]. Available: https://github.com/bchavez/Bogus.

[33] Amazon Web Services, “.Net clients”, noviembre 2022. [Online]. Available: https://opensearch.org/docs/2.4/clients/dot-net/.

[34] SmartBear Software, “REST API Documentation Tool | Swagger UI”, 2023. [Online]. Available: https://swagger.io/tools/swagger-ui/.

[35] C. Bentley, “Risk” in Prince2 - A Practical Handbook, 2002, pp. 197–216.

[36] K. Shonk, “Principled negotiation: Focus on interests to create value”, marzo 2023. [Online].Available:https://www.pon.harvard.edu/daily/negotiation-skills-daily/principled-negotiation-focus-interests-create-value/.

[37] Microsoft, "GitHub", 2023 [Online]. Available: https://github.com/

[38] Atlassian, "Gitflow Workflow - Atlassian Git Tutorial", noviembre 2021. [Online]. Available: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

[39] GitHub, “Automate your workflow from idea to production”, 2023 [Online]. Available: https://github.com/features/actions.

[40] Meta Open Source, “React:La biblioteca para interfaces de usuario web y nativas”, 2023 [Online]. Available: https://es.reactjs.org.

[41] Yarn, “Yarnpkg”, 2023. [Online]. Available: https://yarnpkg.com/.

[42] Microsoft, "NuGet Gallery | Home”, 2023 [Online]. Available: https://www.nuget.org/

[43] Microsoft, “Create .NET apps faster with NuGet”, 2023. [Online]. Available: https://www.nuget.org/.

[44] Python Software Foundation, “Python 3.11.3 documentation”, 2023. [Online]. Available: https://docs.python.org/3/.

[45] Amazon Web Services, “AWS Lambda”, 2023. [Online]. Available: https://aws.amazon.com/lambda/.

[46] Robert C. M, *Clean Code*, Boston, MA, USA: Pearson Education 2008

[47] M. Fowler, K. Beck, *Refactoring: Improving the Design of Existing Code*, 2nd Ed, New York, NY, USA: Addison-Wesly 2018

[48] OpenJS Foundation, “ECMAScript 2015 (ES6) and beyond”, 2023. [Online]. Available: https://nodejs.org/gl/docs/es6.

[49] SonarSource S.A, “SonarQube documentation”, 2023. [Online]. Available: https://docs.sonarqube.org/latest.

[50] OpenJS Foundation, “Find and fix problems in your JavaScript code - eslint - pluggable JavaScript linter”, 2023. [Online]. Available: https://eslint.org/.

[51] Universidad ORT Uruguay, "Normas específicas para la presentación de trabajos finales de carrera (TFDC) Facultad de Ingeniería - Escuela de Tecnología", octubre 2019. [Online]. Available: https://www.ort.edu.uy/innovaportal/file/95484/1/normas-especificas-para-la-presentacion-de-trabajos-finales-de-carrera-facultad-de-ingenieria-escuela-de-tecnologia__documento-302.pdf

[52] Universidad ORT Uruguay, "Hoja de verificación de pautas de presentación de trabajos finales de carreras (excepto Biotecnología)", septiembre 2019. [Online]. Available: https://www.ort.edu.uy/innovaportal/file/95484/1/hoja-de-verificacion-de-pautas-de-presentacion-de-trabajos-finales-de-carreras-excepto-biotecnologia__documento-303.pdf

[53] Universidad ORT Uruguay, "Normas para el desarrollo de trabajos finales de carrera", agosto 2019. [Online]. Available: https://www.ort.edu.uy/innovaportal/file/95484/1/normas-para-el-desarrollo-de-trabajos-finales-de-carrera__documento-304.pdf.

[54] Universidad ORT Uruguay, "Orientación para títulos, resúmenes o abstracts e informes de corrección de trabajos finales de carrera", marzo 2012. [Online]. Available: https://www.ort.edu.uy/innovaportal/file/95484/1/orientacion-para-titulos-resumenes-o-abstracts-e-informes-de-correccion-de-trabajos-finales-de-carrera__documento-306.pdf

[55] M. Jones and G. Hogenson, “Code metrics - cyclomatic complexity - visual studio (windows)”, noviembre 2022. [Online]. Available: https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-cyclomatic-complexity?view=vs-2022.

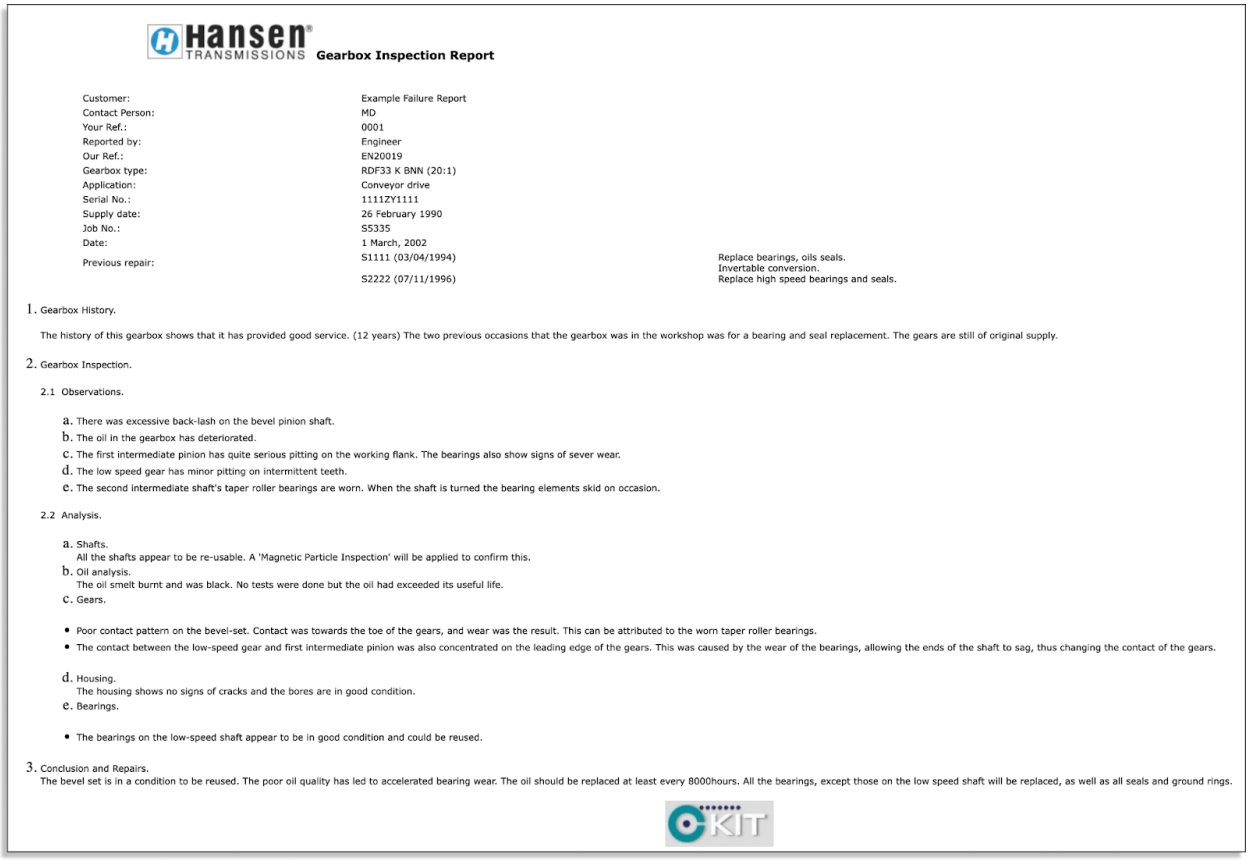
[56] M. Morrison, "Python-Cyclomatic-Complexity", 2011. [Online]. Available: https://github.com/mattjmorrison/Python-Cyclomatic-Complexity.

[57] M. Jones and G. Hogenson. "Code metrics - Maintainability index range and meaning", abril 2022. [Online]. Available: https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-maintainability-index-range-and-meaning?view=vs-2022.

10. Anexos

10.1 Flujo de procesado de documento

A continuación se podrá observar una imagen de un documento ejemplo otorgado por el personal de Renovus. A diferencia de la mayoría de los documentos de mantenimiento que están cubiertos por NDAs, este es de acceso público y fue utilizado como prueba para analizar y extraer datos del sistema (Figura 76).



Hansen TRANSMISSIONS **Gearbox Inspection Report**

Customer:	Example Failure Report	
Contact Person:	MD	
Your Ref.:	0001	
Reported by:	Engineer	
Our Ref.:	EN20019	
Gearbox type:	RDF33 K BNN (20:1)	
Application:	Conveyor drive	
Serial No.:	1111ZY1111	
Supply date:	26 February 1990	
Job No.:	S5335	
Date:	1 March, 2002	
Previous repair:	S1111 (03/04/1994)	Replace bearings, oils seals.
	S2222 (07/11/1996)	Invertable conversion. Replace high speed bearings and seals.

1. Gearbox History.

The history of this gearbox shows that it has provided good service. (12 years) The two previous occasions that the gearbox was in the workshop was for a bearing and seal replacement. The gears are still of original supply.

2. Gearbox Inspection.

2.1 Observations.

a. There was excessive back-lash on the bevel pinion shaft.
b. The oil in the gearbox has deteriorated.
c. The first intermediate pinion has quite serious pitting on the working flank. The bearings also show signs of sever wear.
d. The low speed gear has minor pitting on intermittent teeth.
e. The second intermediate shaft's taper roller bearings are worn. When the shaft is turned the bearing elements skid on occasion.

2.2 Analysis.

a. Shafts.
All the shafts appear to be re-usable. A 'Magnetic Particle Inspection' will be applied to confirm this.
b. Oil analysis.
The oil smelt burnt and was black. No tests were done but the oil had exceeded its useful life.
c. Gears.

- Poor contact pattern on the bevel-set. Contact was towards the toe of the gears, and wear was the result. This can be attributed to the worn taper roller bearings.
- The contact between the low-speed gear and first intermediate pinion was also concentrated on the leading edge of the gears. This was caused by the wear of the bearings, allowing the ends of the shaft to sag, thus changing the contact of the gears.

d. Housing.
The housing shows no signs of cracks and the bores are in good condition.
e. Bearings.

- The bearings on the low-speed shaft appear to be in good condition and could be reused.

3. Conclusion and Repairs.
The bevel set is in a condition to be reused. The poor oil quality has led to accelerated bearing wear. The oil should be replaced at least every 8000hours. All the bearings, except those on the low speed shaft will be replaced, as well as all seals and ground rings.

CKIT

Figura 76: Primera página de documento de ejemplo

- Este documento se puede consultar en el sitio *web*: https://www.ckit.co.za/secure/conveyor/troughed/drive_units_complete/hansen/10-repair-report.htm

Utilizando el ejemplo del documento anterior, se presentará en este anexo, el funcionamiento de la lógica de procesamiento de un documento, pero a alto nivel para que se pueda visualizar los diferentes estados por los que pasa un documento. En la etapa inicial, este documento se carga desde el *frontend* para ser almacenado en el *bucket de* S3 y disparar el *trigger* que comienza el procesado del documento por el módulo denominado Analyzer (Figuras 77 y 78)

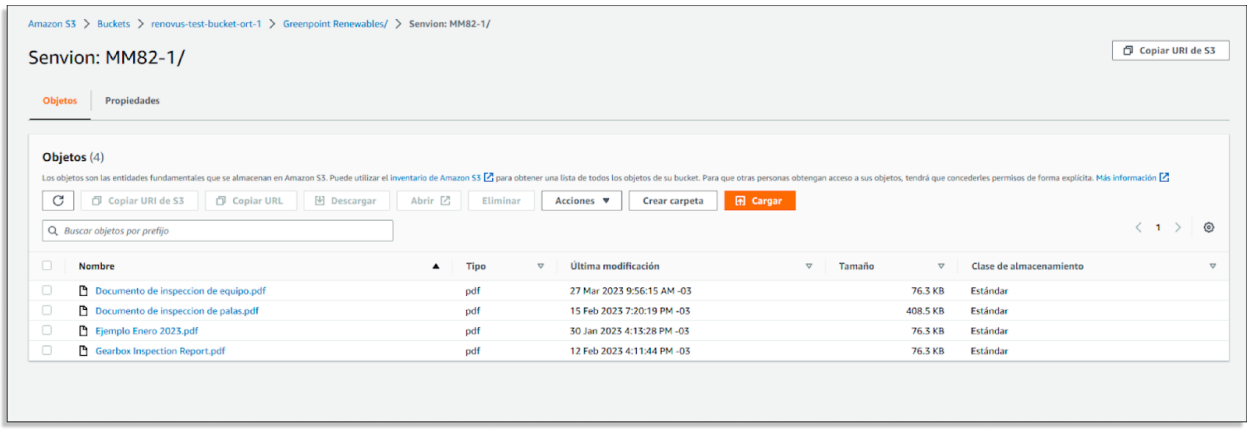


Figura 77: *Bucket* S3 con documentos

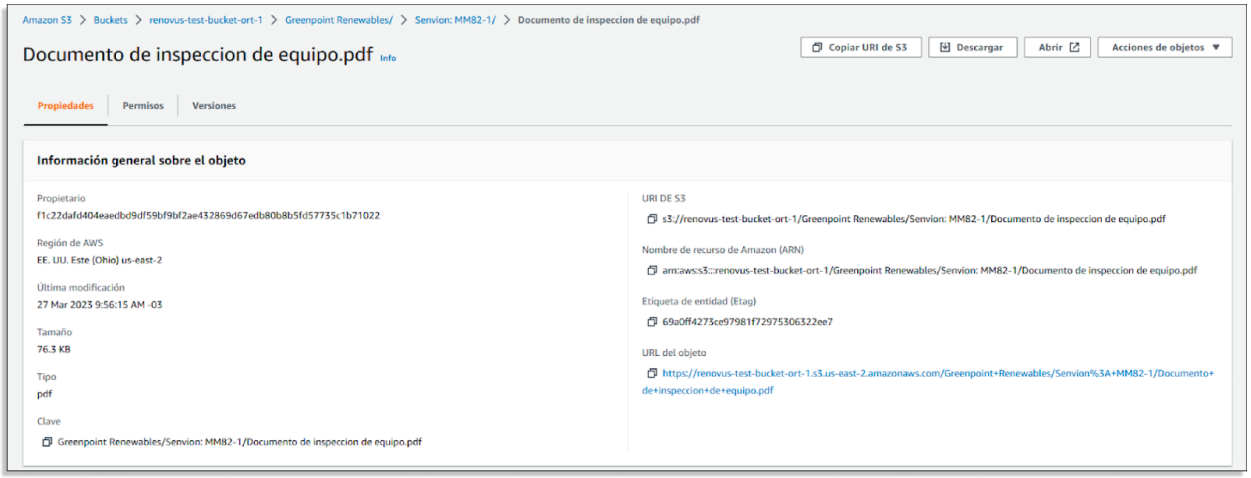


Figura 78: Documento en el *Bucket*

Luego de que el documento ha sido cargado, se dispara el trigger y comienza la ejecución del módulo Analyzer, donde se realizan los análisis correspondientes por Amazon Textract y Amazon Comprehend, para luego transformar esa información y ser enviados al Saver para su guardado final en la base de datos OpenSearch (Figuras 79, 80, 81, 82 y 83).

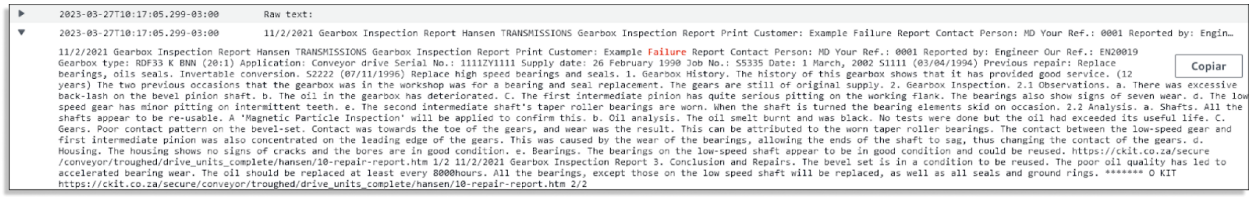


Figura 79: Bitácora en CloudWatch de comienzo de procesado en Analyzer, texto plano

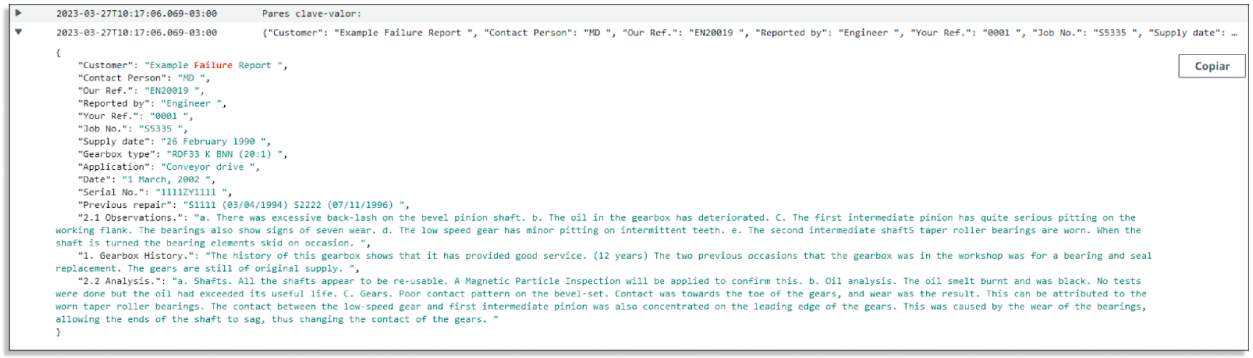


Figura 80: Bitácora en CloudWatch de procesado en Analyzer, datos extraídos

Figura 81: Bitácora en CloudWatch de procesado en Analyzer, entidades encontradas

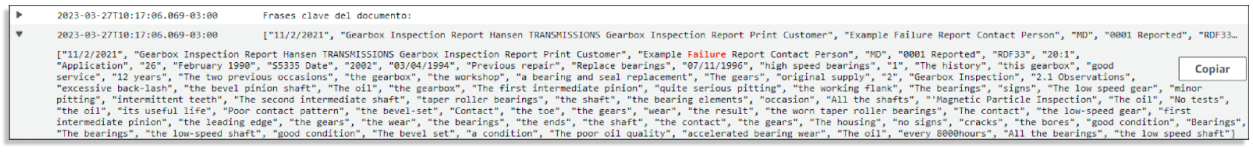


Figura 82: Bitácora en CloudWatch de procesado en Analyzer, frases clave encontradas

Figura 83: Bitácora en CloudWatch de procesado en Analyzer, metadata definida

Una vez finalizada la obtención de datos y su transformación, el objeto es enviado al Saver, donde tendrán una última transformación, removiendo prefijos, acomodando formatos y demás, para que esta información pueda ser enviada y guardada en la base de datos (Figuras 84, 85, 86 y 87).

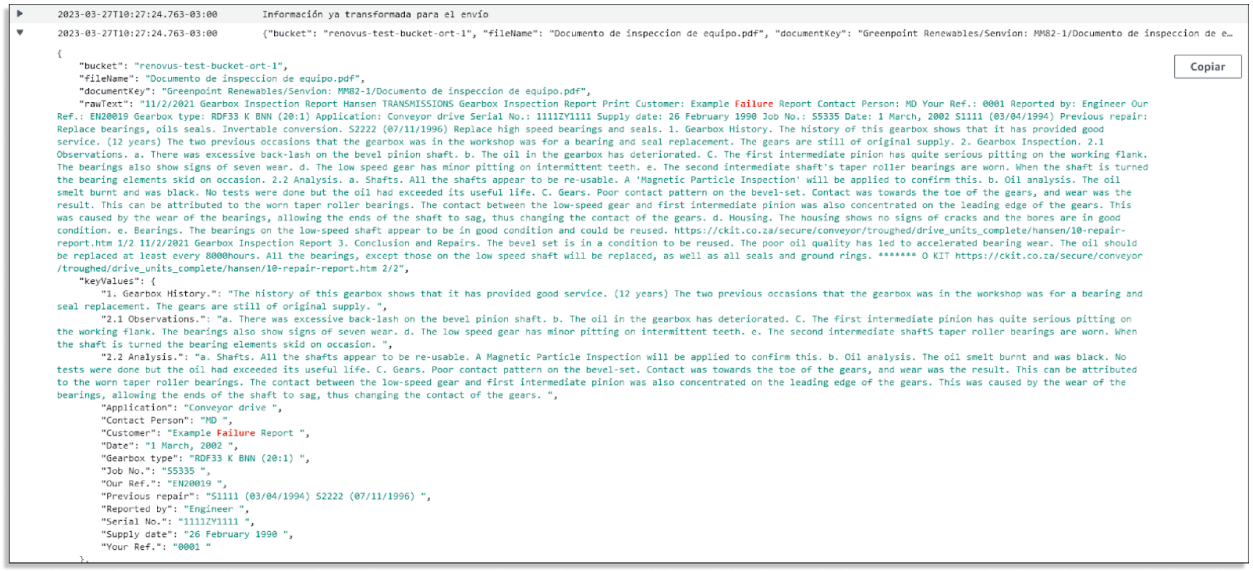


Figura 84: Bitácora en CloudWatch de los datos transformados por el Saver para almacenar en la base de datos, parte 1

Figura 85: Bitácora en CloudWatch de los datos transformados por el Saver para almacenar en la base de datos, parte 2

Figura 86: Bitácora en CloudWatch de los datos transformados por el Saver para almacenar en la base de datos, parte 3

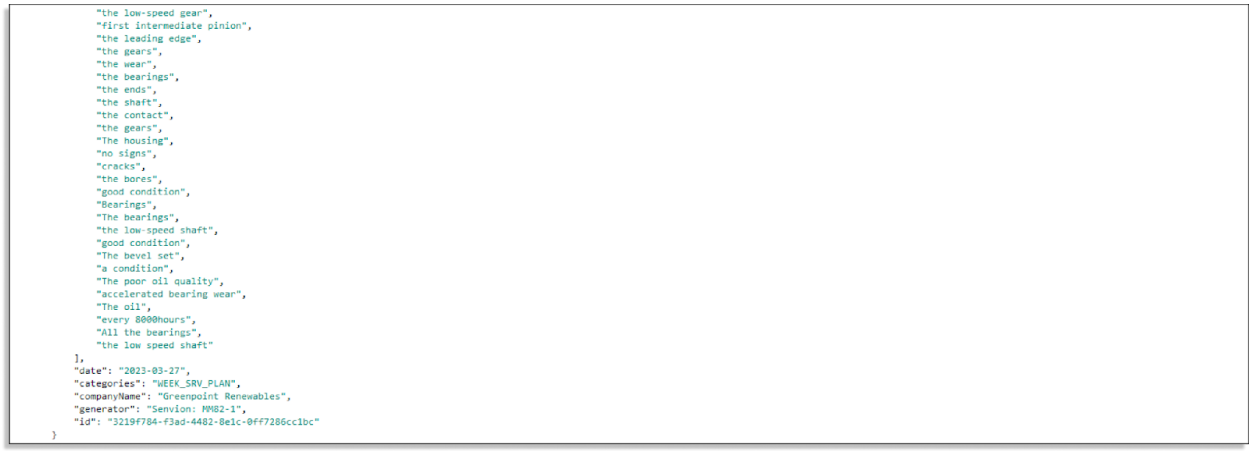


Figura 87: Bitácora en CloudWatch de los datos transformados por el Saver para almacenar en la base de datos, parte 4

Toda esta información luego de ser enviada se almacena en la base de datos, la cual se visualiza de la siguiente manera (Figuras 88, 89 y 90).

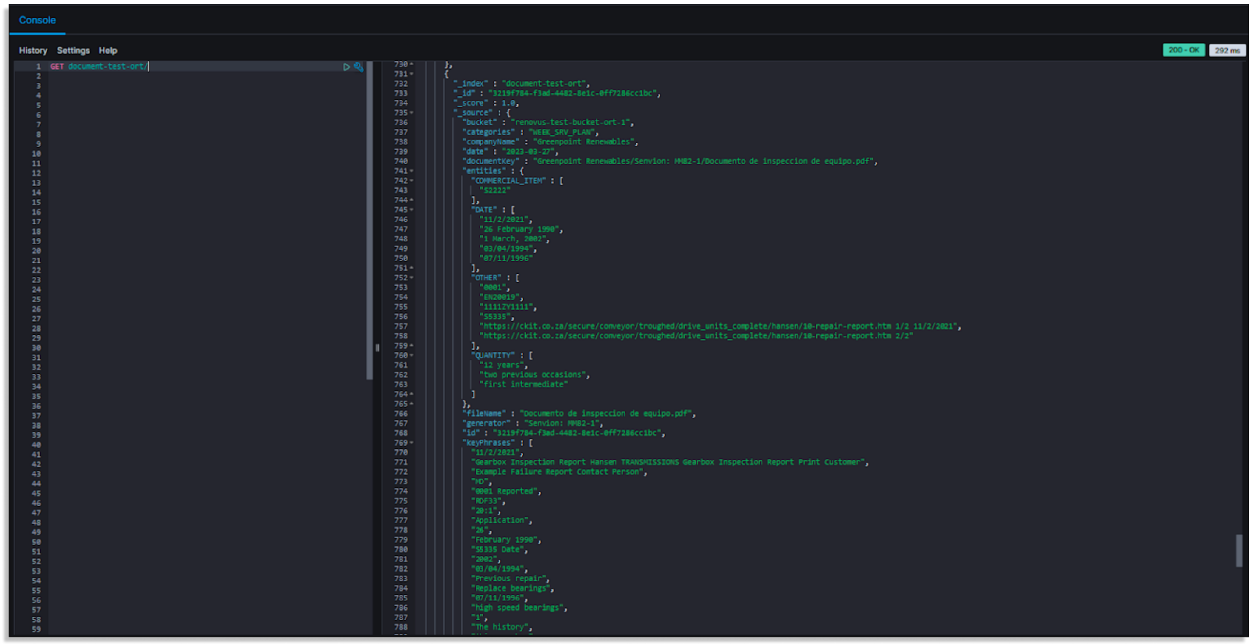


Figura 88: Visualización del documento en base de datos, parte 1

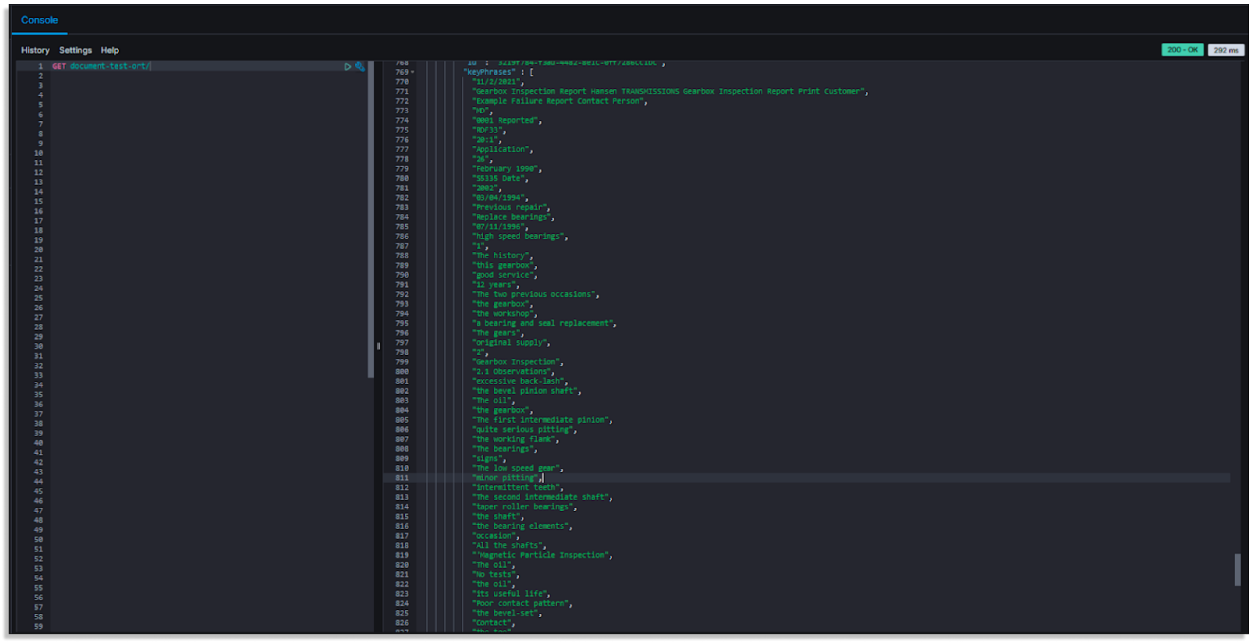


Figura 89: Visualización del documento en base de datos, parte 2

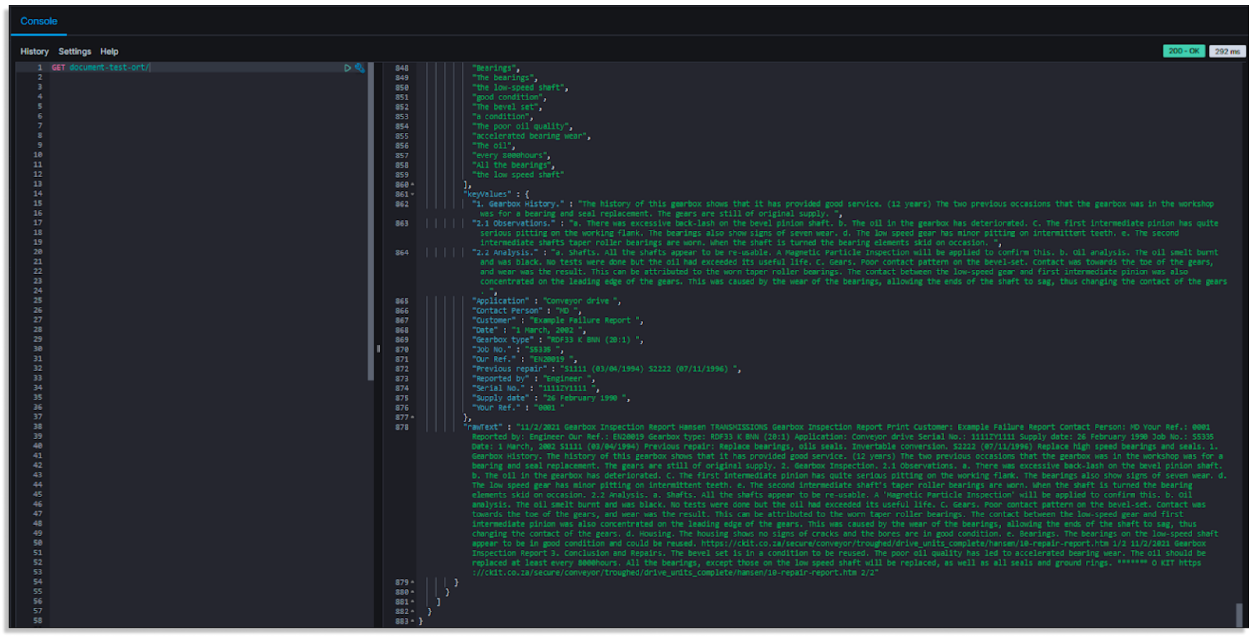


Figura 90: Visualización del documento en base de datos, parte 3

10.2 Documento presentación del proyecto

A continuación se muestran capturas del documento de presentación del proyecto de tesis (Figuras 91, 92, 93 y 94).

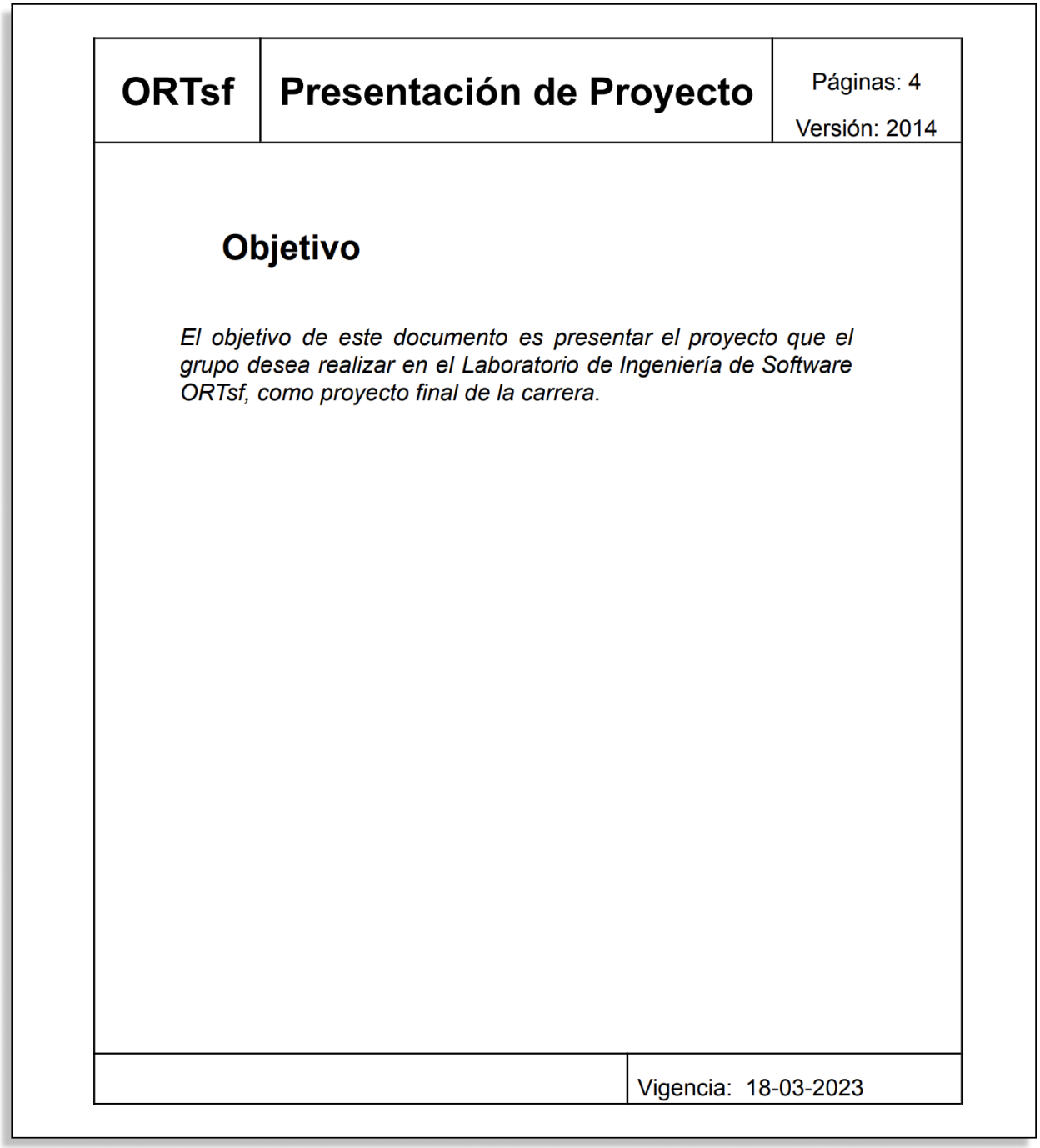


Figura 91: Página 1 de documento de presentación de proyecto

INTEGRANTES DEL EQUIPO (nombre y número de estudiante):

222313 - Bermúdez, Alejandro
210434 - Franggi, Diego
210072 - Martínez, Federico

CARRERA:

Ingeniería: SI Licenciatura: NO

MARQUE CON UNA CRUZ EL CASILLERO QUE CORRESPONDA:

EMPRENDIMIENTO PROPIO		**PROYECTO PARA UNA PERSONA O UNA EMPRESA**	X

NOMBRE DEL CLIENTE O EXPERTO DEL DOMINIO:

Diego Blixen

CONTACTO EN EL CLIENTE O EXPERTO DEL DOMINIO

diegoblixen@gmail.com

DESCRIPCIÓN DEL CLIENTE O EXPERTO DEL DOMINIO

Renovus es una solución tecnológica que utiliza ciencia de datos e inteligencia artificial, para ofrecer herramientas de mejora a empresas de la industria de las energías renovables. El emprendimiento se inició en marzo 2021 y en diciembre 2021 y fue impulsado por la ANII

TIPO DE PROYECTO:

Desarrollo

- [x] Desarrollo de solución a medida (solicitado por el cliente)
- [ ] Desarrollo de producto o solución
- [ ] Diseño de servicio con soporte tecnológico

Investigación

- [ ] Metodologías y procedimientos
- [ ] Nuevas tecnologías
- [ ] Desarrollo de prototipos

- [ ] Otros (especificar):

DESCRIPCIÓN

El proceso de mantenimiento y gestión de los parques eólicos en nuestro país conlleva un desafío enorme en lo que respecta al seguimiento. Todo desde un chequeo de rutina a una reparación de emergencia genera un documento particular el cual actualmente es administrado en forma manual. Renovus busca implementar un sistema de gestión de archivos que busque simplificar esta administración, con funcionalidades desde la generación de documentos y la búsqueda de los mismos en base a prioridades.

Figura 92: Página 2 de documento de presentación de proyecto

REQUERIMIENTOS

En base a lo planteado originalmente por el cliente, entendimos que los requerimientos son los siguientes:

- Implementación de un sistema **basado en microservicios** que permita la generación de documentos apropiados para el negocio. Los mismos varían desde eventos rutinarios, automáticos o de situaciones críticas (clasificación de daños, alarmas diarias). Por ejemplo, cada reparación de un molino genera información sobre el tipo de daño, la severidad, fecha, entre otros muchos campos que han de ser registrados y organizados acorde a las necesidades de negocio.

- Búsqueda y gestión sobre esos archivos en base a diversos parámetros (fecha, criticidad, tipo de documento). Estas búsquedas han de ser performantes al trabajar con miles de archivos.

- Generación de reportes estadísticos en base a fecha, criticidad, tipo de documento. Los mismos buscarán detectar posibles patrones recurrentes entre los reportes generados.

- Prevención de documentos repetidos, organización de documentos en base a diversas categorías.

- Generación de "tickets" o tareas para el encargado requerido en base a la configuración deseada. Envío y seguimiento de estas tareas. Mantenimiento de un "to-do-list".

- Brindar API para la generación de reportes o alertas por sistemas automáticos. Por ejemplo una alarma puede generar un reporte al invocar un endpoint.

- Posible App móvil tanto para plataformas IOS y Android.

Requerimientos no funcionales tentativos:

- Despliegue en la nube de Amazon.

- Uso de metodologías ágiles como Scrum por solicitud del cliente.

- Mantener configuraciones básicas de seguridad para la web.

- Autenticación, autorización y tenancy: solo subir archivos de usuarios o dispositivos autorizados.

- Confiabilidad y disponibilidad: brindar chequeo de salud de nuestros microservicios.

- Observabilidad: posible uso de new relic para visualizar el estado de la aplicación.

- Identificación de fallas con seguimiento de Logs a fin de lograr un proyecto profesional.

- Uso de flujos de GitHub e Integración continua.

IMPACTO DEL PROYECTO PARA EL CLIENTE

Con la implementación de este proyecto, el cliente se asegura una mejor performance y una mejor operativa a nivel global. Con la solución se busca optimizar el tiempo de análisis de archivos y un mejor seguimiento de los mismos.

El cliente se verá beneficiado ya que podrá desarrollar un análisis más exhaustivo y organizado de los datos extraídos de los molinos de viento y así generar una mejor sinergia entre los que se encarga de obtener los datos y entre los que se encarga de analizarlos (mejor gestión). De esta forma se logrará una mayor ganancia tanto a niveles energéticos como económicos.

Figura 93: Página 3 de documento de presentación de proyecto

TECNOLOGÍAS a UTILIZAR (En caso de ya estar definida)

En base a las primeras tomas de contacto se definen como potenciales tecnologías:

- .Net Core
- React.js
- React Native
- Amazon Web Services
- Selenium

EXPERIENCIA DEL EQUIPO CON ESTAS TECNOLOGÍAS

El equipo posee experiencia en estas tecnologías tanto de forma académica como laboral. Las tecnologías correspondientes a frontend (desarrollo mobile y web) son más conocidas y manejadas por Federico Martínez dada su experiencia laboral. En cuanto a lo relacionado a backend y la nube, es manejado por todo el equipo, principalmente de forma académica, pero con buenos conocimientos y experiencia en el uso de las mismas. En cuanto a Selenium como framework de pruebas automatizadas, es una tecnología manejada por Alejandro Bermúdez, que adquirió en base a su experiencia laboral en el área de QA. En lo relacionado a seguridad, si bien no mencionamos tecnologías puntuales al respecto ya que están a definir, Diego Franggi es nuestro experto por su experiencia laboral con dichas tecnologías.

DIFICULTADES PREVISIBLES DEL PROYECTO

- Retrasos por parte del equipo por problemas personales, de enfermedades, etc.
- Dificultades de comunicación con el cliente.
- Complejidad a la hora de realizar la integración con sus sistemas

MOTIVACIÓN DEL EQUIPO PARA ESTE PROYECTO

Participar de un proyecto que permita a la empresa optimizar sus tiempos, con una plataforma de gestión con gran potencial, nos resultó super interesante dada la amplia gama de posibilidades que esto nos brinda, si bien partimos con un horizonte de funcionalidades bastante definido, creemos que puede ser un sistema altamente escalable en base a las necesidades de Renovus.
Todo esto nos resultó sumamente interesante, porque entendemos que el valor agregado que le podemos aportar al cliente es muy relevante, y sumado a esto, la libertad a nivel tecnológico que se nos brindó, resultó primordial para la selección del proyecto, ya que nos da la libertad de no solo poner en práctica los conocimientos adquiridos a través de los años, no solo de forma académica sino que también de forma laboral en los últimos años, sino que también nos permite investigar tecnologías que resulten útiles para conseguir la mejor solución.

Figura 94: Página 4 de documento de presentación de proyecto

10.3 Proceso de Design Thinking

A continuación se muestran imágenes relacionadas al proceso de Design Thinking (Figuras 95, 96. 97, 98, 99. 100, 101 y 102).

Figura 95: Primera parte del Proceso de Design Thinking

Figura 96: Segunda parte del Proceso de Design Thinking

Reuniones de empatización y definición

Figura 97: Empatización con cliente

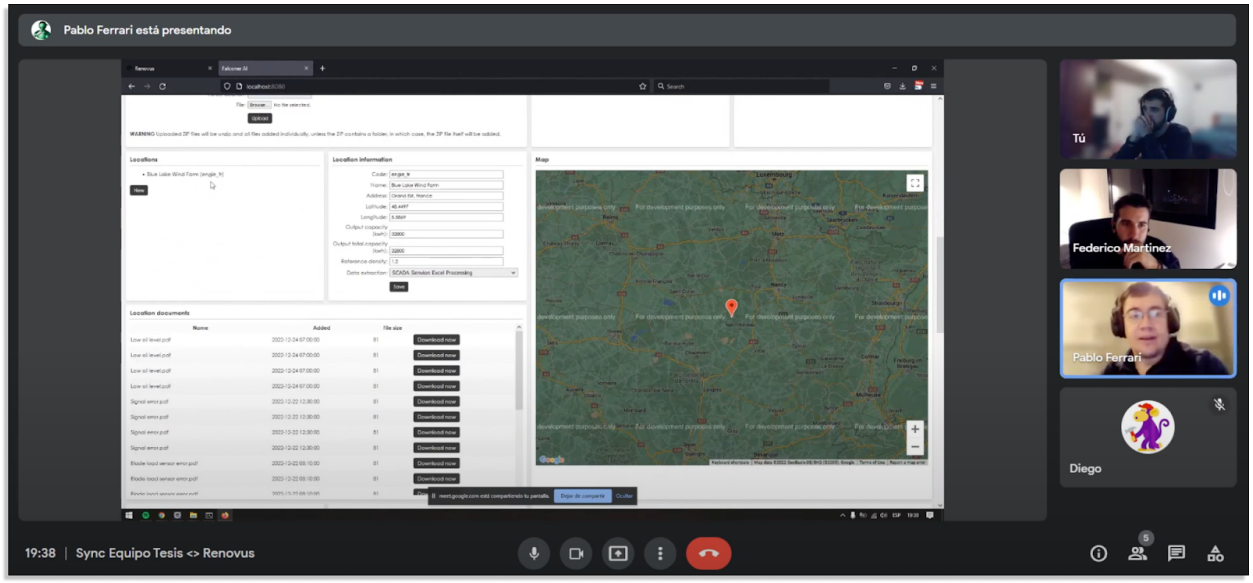


Figura 98: Definir el problema

Idear, prototipar y testear

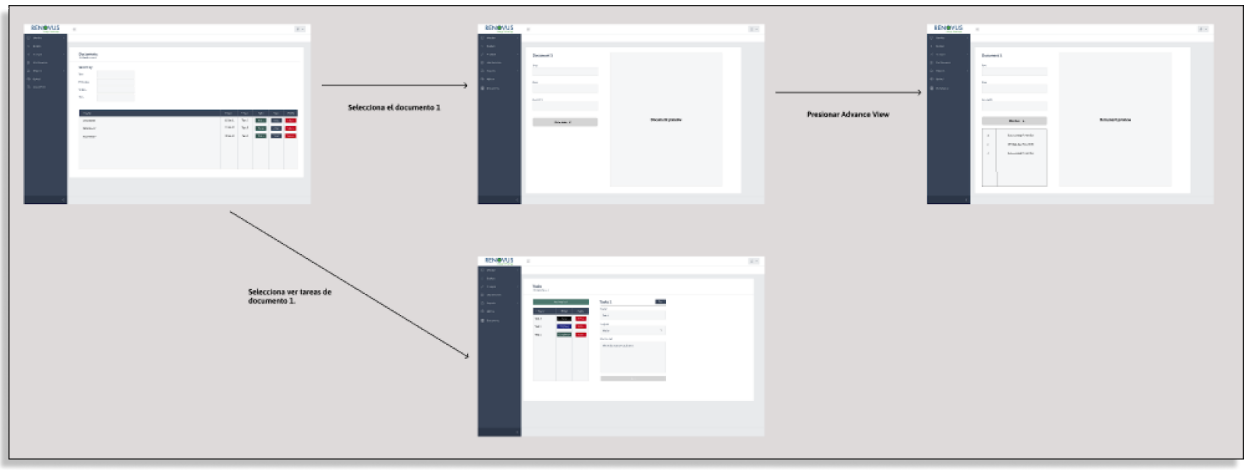


Figura 99: Primer prototipo

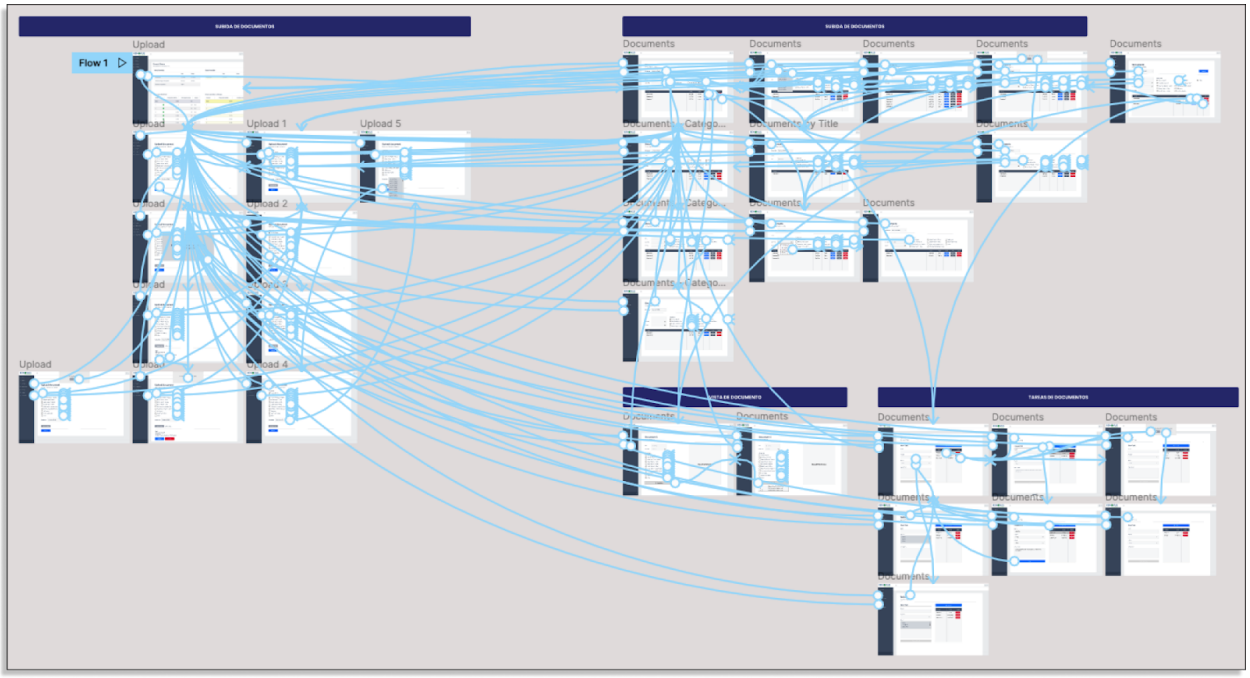


Figura 100: Prototipo final

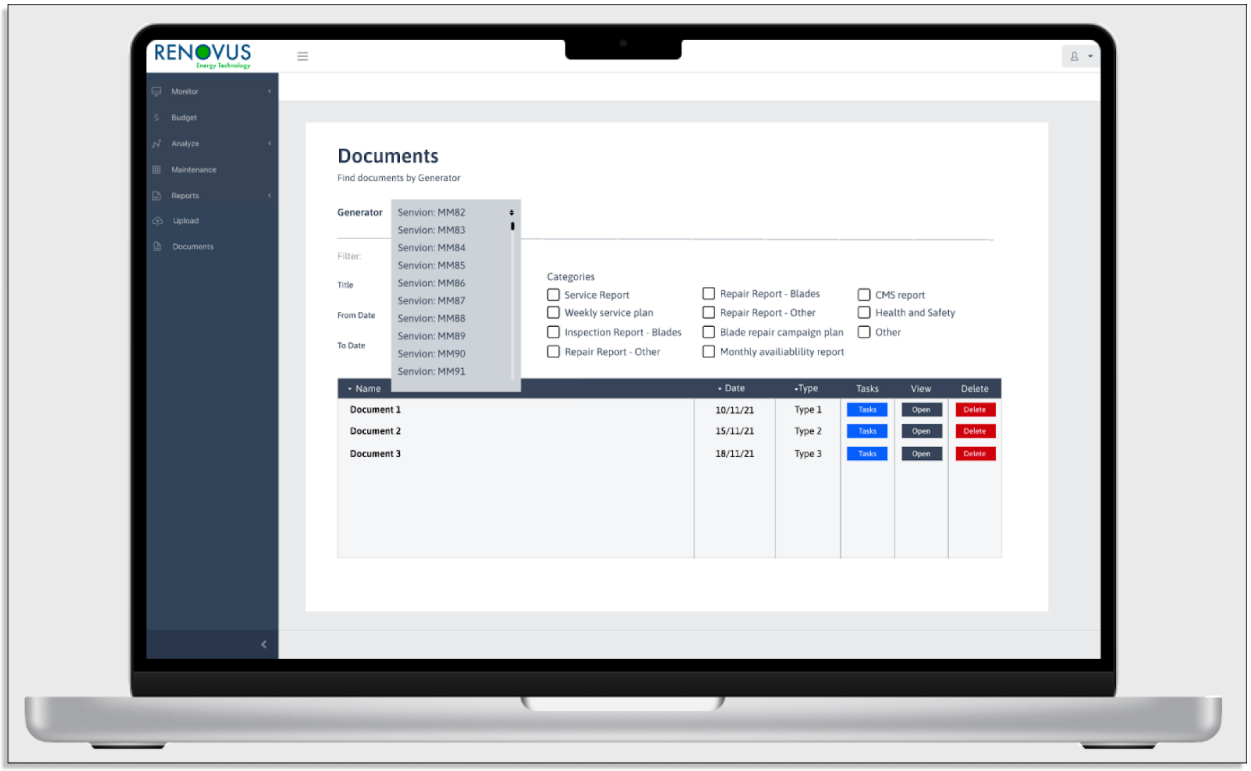


Figura 101: Prototipo en acción

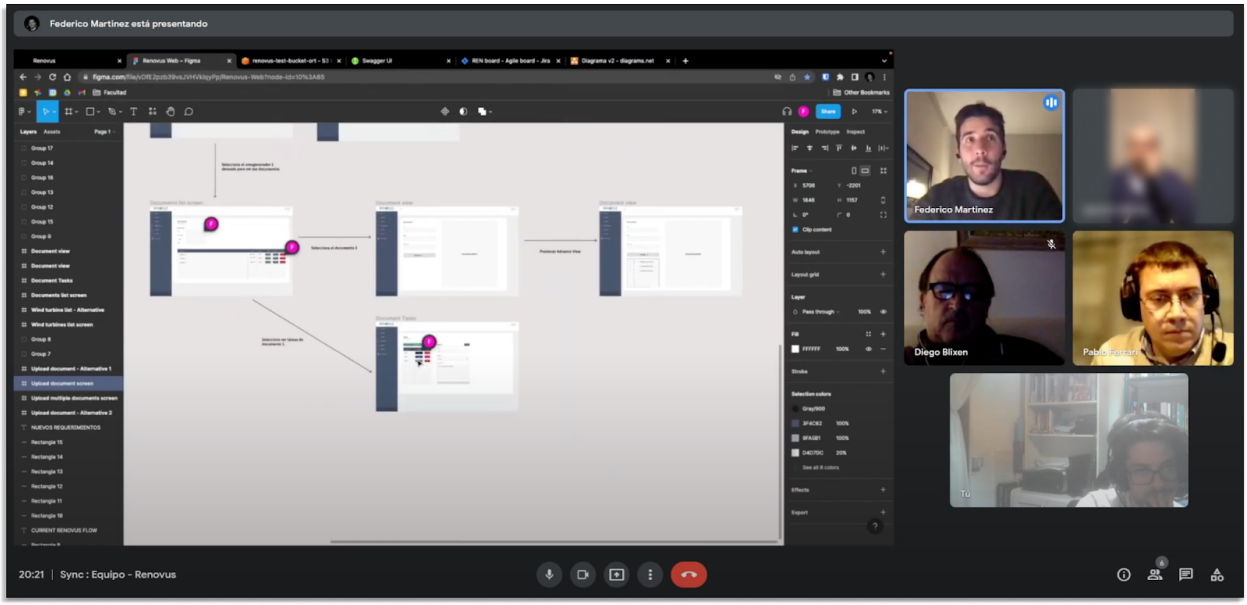


Figura 102: Demostración de prototipo

10.4 Especificación no detallada

Nota: Este documento es una transcripción de la versión original del documento de Especificación no detallada.

Abstract

El presente documento detalla lo que se espera del sistema de administración de archivos para parques eólicos a ser desarrollado por el equipo de proyecto final de carrera de Universidad ORT Uruguay.

Este sistema debe ser un componente que pueda ser instanciado por el producto comercial SaaS llamado Renovus. El resultado del sistema permitirá extraer datos de documentos, pudiendo asignar tareas y organizarlos según criterio del usuario.

Equipo de proyecto

Por Renovus:

- Diego Blixen (Co-fundador)
- Pablo Ferrari (Co-fundador)
- Anónimo (Experto / Usuario final)

Por ORT:

- Alejandro Bermúdez
- Diego Franggi
- Federico Martínez

Limitaciones de arquitectura

Repudio a plataformas fuera de AWS	El sistema debe correr en servicios o infraestructura de Amazon Web Services.
Performance en subida de documentos	El sistema debe permitir la carga de documentos al sistema en un tiempo no superior a un día.
Performance en búsqueda de documentos	El sistema debe permitir la búsqueda de documentos en el sistema en un tiempo menor a cinco segundos.
Idioma del sistema	El sistema debe ser elaborado en su totalidad en inglés, incluyendo sus interfaces de usuario y código.
Crecimiento vertical y horizontal	El sistema debe ser escalable tanto vertical como horizontalmente.
Confiabilidad en análisis de documentos	Se requiere alcanzar un buen análisis de la información presente en los documentos.
Usabilidad del sistema	El sistema debe ser fácil e intuitivo de utilizar, manteniendo el diseño y el flujo de la página de Renovus
Bitácoras en Cloudwatch	El sistema debe tener un registro de bitácoras (logs) utilizando Cloudwatch
Región AWS	El sistema debe utilizar la región us-east2 de Amazon Web Services cuando sea posible.

Tabla 31: Limitaciones de arquitectura

Componentes

- **Frontend:** Componente encargado de manejar el sitio *web* de Renovus donde se agregaran las nuevas funcionalidades al sistema actual.

- **Backend:** Componente Web API encargada de manejar una variedad de tareas relacionadas con la administración de documentos.
- **Base de datos:** Componente encargado de almacenar los documentos del sistema
- **Textract:** Componente encargado de realizar el reconocimiento óptico de caracteres (OCR) dentro de documentos.
- **Comprehend**: Componente encargado de analizar los documentos mediante inteligencia artificial.

Funcionalidades

1. Subida y análisis de archivos

El objetivo de esta funcionalidad es que el sistema permita la subida de documentos para poder analizar y luego extraer valores de interés para el cliente.

- Subida de documentos a sistema
- Registro de documentos en base de datos
- Análisis y extracción de datos de documentos subidos
- Resultado de análisis visible para cliente.

2. Búsqueda rápida sobre archivos analizados

Para cada documento analizado, el sistema permitirá la búsqueda de los mismos mediante distintos métodos de filtrado para que el cliente pueda ver el resultado de los documentos subidos.

- Página de filtrado de documentos
- Filtrado de documentos por distintos parámetros y paginado.
- Vista de detalles de documentos (datos post análisis).

- Eliminación de documentos

3. Administración de tareas

El sistema dispondrá de asignación de tareas por cada documento con motivo de asignar responsabilidades a personal de mantenimiento de distintos generadores.

- Página de administración de tareas para documentos

- Generación, edición y eliminación de tareas

10.5 Documento de mantenimiento

Nota: Este documento es una transcripción de la versión original del documento de mantenimiento entregado a Renovus.

1. **Propósito**

El propósito de este documento es facilitar primero el entendimiento del sistema como un todo y luego profundizar en detalle las decisiones de diseño más importantes que se han tomado durante su desarrollo. Además, se busca garantizar que el lector pueda introducir cambios en el sistema sin comprometer su funcionamiento.

El mantenimiento del sistema es crucial para asegurar que este siga funcionando correctamente y de manera eficiente a lo largo del tiempo. Con este documento, los desarrolladores pueden tener una guía detallada sobre los módulos del sistema, lo que les permitirá comprender su funcionamiento y detectar posibles problemas o mejoras que se puedan realizar, así como facilitar la introducción de futuros cambios en los mismos.

2. **Descripción de la arquitectura general**

Se comienza realizando una vista general del sistema previo al detalle de los módulos particulares del mismo, a modo de transmitir el conocimiento del contexto en el cual se ejecutan los servicios a estudiar.

A continuación se detalla un diagrama de referencia de arquitectura de Amazon Web Services como un primer acercamiento a la solución. Este diagrama sustituye en parte a la vista de despliegue, ya que posee en cierta forma la misma información. Se advierte al lector que el siguiente diagrama no contiene el *backend* del sistema actual de Renovus, es decir, solamente refiere al aplicativo realizado por el equipo estudiante, a excepción del Frontend que representa un trabajo conjunto entre ambas partes (Figura 103).

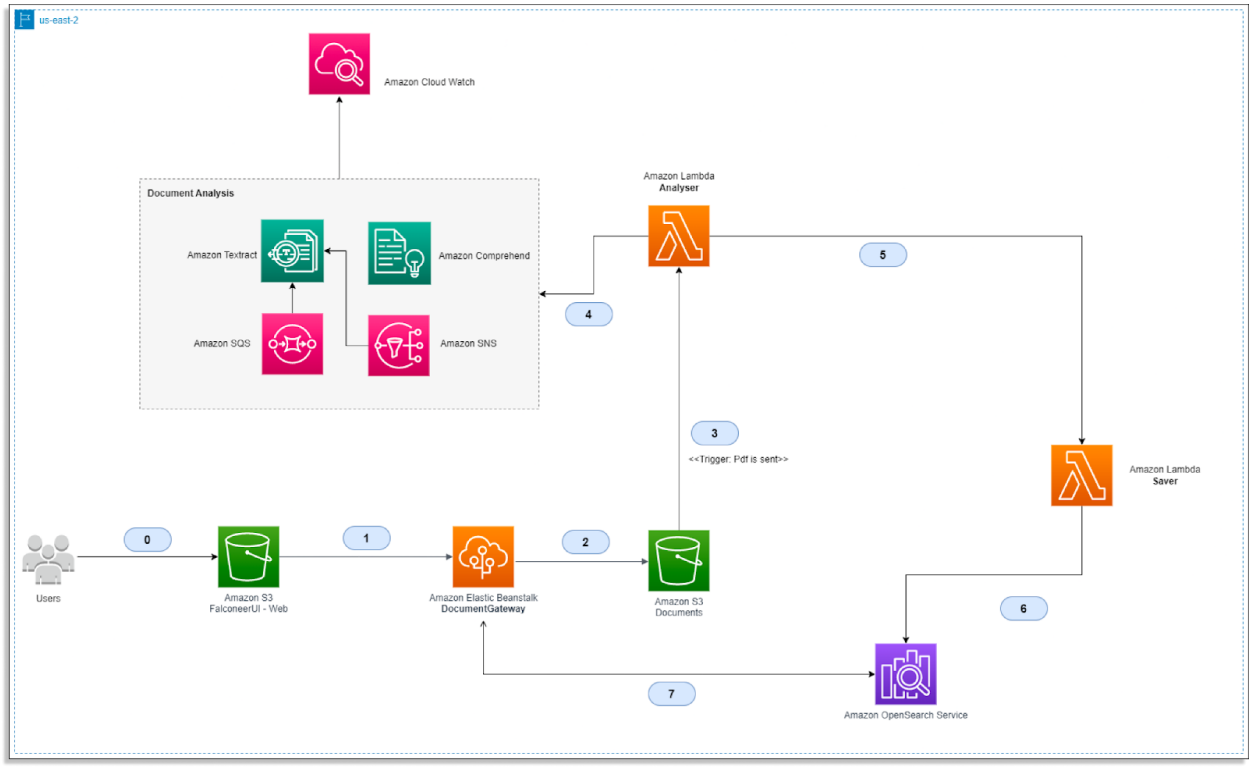


Figura 103: Vista de arquitectura general

3. Breve Explicación de los servicios involucrados

FalconeerUI

Este servicio representa el *frontend web* de la aplicación y se trata del resultado de un esfuerzo conjunto entre el equipo estudiante y Renovus. Los estudiantes trabajaron sobre la solución existente a modo de generar los apartados correspondientes a las funcionalidades planteadas en los requerimientos del proyecto.

Las responsabilidades del servicio recaen en las de ser el punto de contacto de los usuarios con las funcionalidades del proyecto, permitiendo la subida de archivos, gestión y búsqueda de documentos, así como las tareas asociadas a los mismos.

DocumentGateway

Se trata de una WebAPI construida en .NET Core que sirve como portal de acceso a la nube de Renovus para la gestión de documentos. Esta WebAPI se encarga de manejar una variedad de tareas relacionadas con la administración de documentos.

Se presenta entonces como la principal interfaz de búsqueda para que servicios como FalconeerUI puedan buscar documentos en el sistema mediante consultas *web* y adicionalmente ofrece el punto de contacto con las funcionalidades vinculadas a la administración de tareas de los documentos y aerogeneradores del sistema.

Documents Bucket (S3)

Documents Bucket es un contenedor de objetos en AWS, también conocido como "bucket", que se utiliza para almacenar y acceder a los documentos del sistema de forma rápida y escalable. Aunque el nombre real en el sistema es "renovus-test-bucket-ort-1", se le denomina "Documents Bucket" o “Documents” por simplicidad.

El *bucket* está cuidadosamente controlado para asegurar que los documentos de una compañía no se mezclen con los de otra, y para facilitar el acceso a los documentos específicos de una compañía o generador dentro de la misma.

Analyzer

Se trata de una función Lambda compleja, diseñada como un “orquestador” a los servicios encargados de analizar los documentos. Su invocación es desencadenada por un disparador interno de Amazon al subir un archivo a “Documents Bucket”.

Document Analysis no es un servicio per se, se trata de una agrupación de servicios relacionados con el análisis de documentos. En primer lugar, tenemos Amazon Textract, encargado de la conversión del documento a texto mediante técnicas de OCR y luego Amazon Comprehend,

servicio configurado por el equipo para el correcto estudio y obtención de datos claves de documentos. Adicionalmente, se hace uso de distintos servicios de apoyo, como lo es una cola de mensajes para manejar la carga de múltiples documentos que podrían entorpecer las operaciones; un servicio de notificaciones llamado SNS que informa sobre tareas de larga duración completadas (puntualmente el análisis de Textract) y un servicio de Amazon llamado CloudWatch destinado al monitoreo de los logs propios de este conjunto de servicios.

Saver

La función Lambda "Saver" tiene como principal tarea guardar la información procesada por el servicio previo en el formato correspondiente dentro de la base de datos. Como su nombre sugiere, su función es asegurar que el almacenamiento se realice correctamente y en el formato adecuado. Además, esta función también encapsula las responsabilidades de cambiar el formato en el futuro, si fuese necesario.

Open Search Service

Este servicio es la base de datos principal del proyecto, donde se almacenan los documentos procesados junto con metadatos que permiten un filtrado rápido y eficiente en el sistema, ya sea por nombre, palabras clave, fecha o aerogenerador. Además, esta base de datos también se encarga de almacenar las tareas relacionadas con dichos documentos.

4. Explicación de los pasos relevantes

Si volvemos a examinar el diagrama de referencia, podremos ver una secuencia numérica del 0 al 7. Los mismos representan pasos relevantes a la funcionalidad de carga y procesado de archivos, la cual fue dictaminada como la más relevante para el entendimiento del sistema y elegida para la numeración requerida por el diagrama.

0. El usuario interactúa con la *web*, actualmente alojada en un bucket de S3.

1. Se efectúa la carga del documento al servicio DocumentGateway, quien valida y procesa la solicitud

2. De estar todo correcto se procede a subir el archivo a Documents Bucket.

3. Se lanza un disparador interno de amazon invocando a Analyzer.

4. La función Analyzer coordina las llamadas a los diferentes servicios. Primero, encola el documento y se subscribe a la notificación de procesado de Textract en un tópico SNS. Una vez que el documento se ha convertido en texto plano, lo envía a Comprehend para su análisis. Este servicio extrae información importante como palabras clave, datos relevantes, entidades y una comprensión general del texto, ya que ha sido entrenado para procesar este tipo de datos. Todas las operaciones se registran en las bitácoras (logs) de CloudWatch.

5. El documento es enviado a Saver.

6. Saver optimiza el formato de los documentos para mejorar la búsqueda en OpenSearch Service y facilitar su lectura, garantizando un almacenamiento adecuado en esta base de datos.

Nota: aunque la funcionalidad de tareas no se tiene en cuenta para este flujo, no se agregan servicios adicionales ni relaciones imprevistas al diagrama, por lo que no fueron consideradas para no causar confusiones.

5. Vista de componentes y conectores

A la hora de adentrarse en un sistema complejo de *software*, una de las tareas más importantes es identificar y definir la arquitectura del sistema. La arquitectura describe la estructura del sistema y las relaciones entre sus componentes y conectores, puede ayudar a los desarrolladores a comprender mejor cómo funciona el sistema, cómo se comunican sus componentes y cómo se gestionan los datos.

La vista de Componentes y Conectores es una de las vistas más comunes de la arquitectura de un sistema, y se centra en los componentes que tienen una importancia en **tiempo de ejecución**, como los servicios, procesos y bases de datos, así como en los conectores y caminos por donde circula la información (Figura 104 y Tabla 32).

Representación primaria de la aplicación

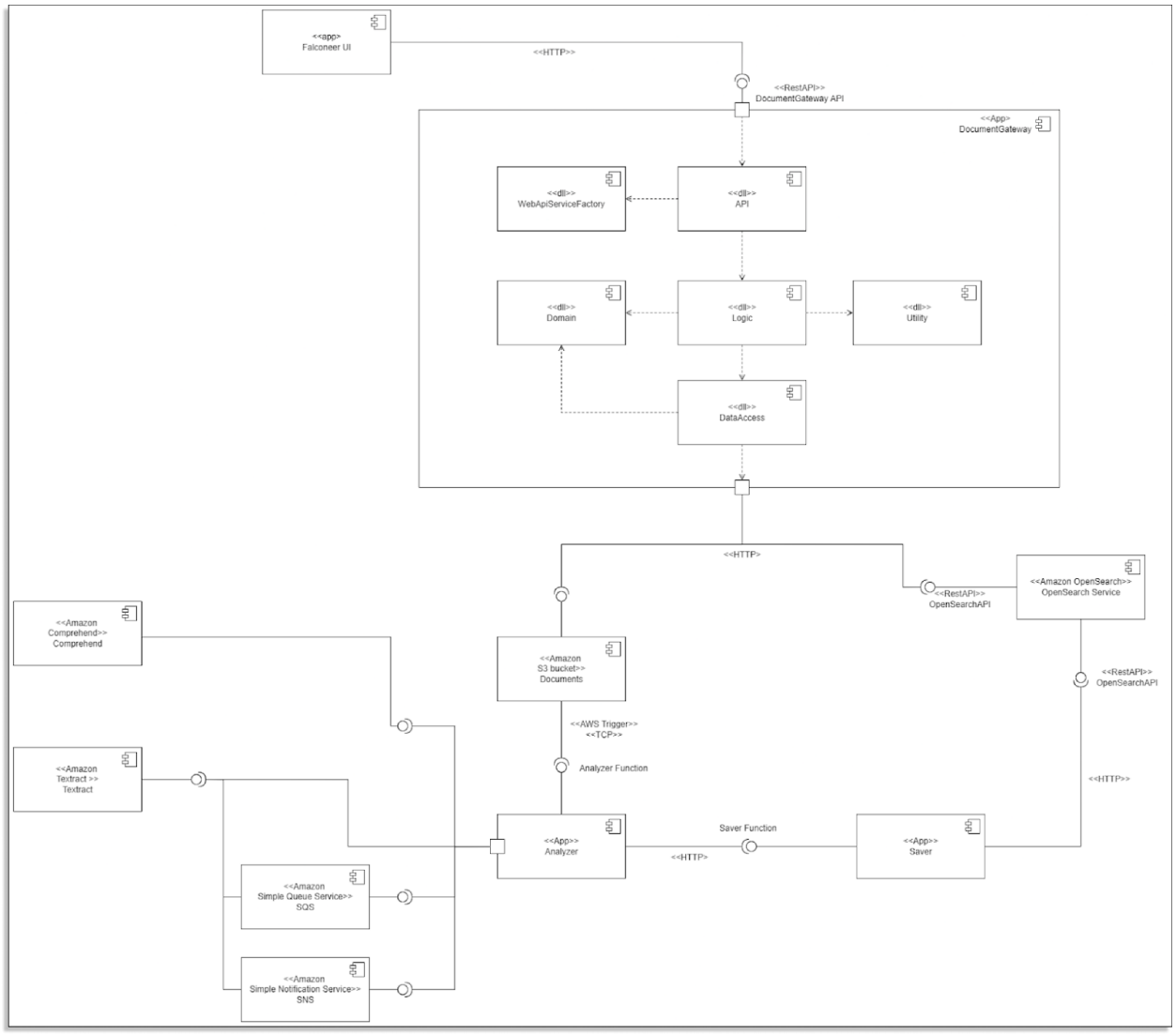


Figura 104: Vista de componentes y conectores

Catálogo de elementos

Elemento	Tipo	Responsabilidad
Analyzer	APP	Orquestador del análisis de documentos. Posee un algoritmo de apoyo a los componentes a los que llama.

Api	DLL	Interfaz que provee el *backend* como medio de interacción entre este, la *web* y el resto del sistema siguiendo el estilo arquitectónico REST.
Comprehend	Amazon Comprehend Service	Componente encargado de analizar los documentos mediante inteligencia artificial. El mismo fue entrenado por el equipo para mejores resultados.
DataAccess	DLL	Componente donde se realizan las comunicaciones pertinentes al acceso a datos del sistema.
DocumentGateway	APP	API encargada de manejar una variedad de tareas relacionadas con la administración de documentos.
Documents	Amazon S3 Bucket	Contenedor de documentos del sistema.
Falconeer UI	APP	Servicio *frontend* de la aplicación, encargada de presentar la misma al usuario de forma sencilla. No se detalla por completo sus componentes individuales a modo de simplificar el diagrama.
Logic	DLL	Componente encargado de ejecutar la lógica de la aplicación.
OpenSearch	Amazon OpenSearch Service	Base de datos no relacional especializada para el almacenaje y búsqueda de texto plano. Motor de las búsquedas de documentos en el sistema.
Saver	APP	Componente encargado de guardar la información procesada sobre los documentos con cierto formato en la base de datos.

SNS	Amazon Simple Notification Service	Servicio encargado de controlar y notificar sobre la completitud del procesamiento del texto en Textract a la función Lambda Analyzer.
SQS	Amazon Simple Queue Service	Cola de mensajes encargada de limitar las llamadas al componente de Textract, permitiendo procesar cada documento de forma apropiada
Textract	Amazon Textract Service	Servicio de Amazon que representa el componente encargado de realizar el reconocimiento óptico de caracteres (OCR) a los documentos suministrados
Utility	DLL	Servicio de apoyo a Logic encargado de ejecutar funciones comunes o reusables.
WebApiServiceFactory	DLL	Servicio que mediante inyección de dependencias elimina relaciones entre los módulos inferiores del sistema y la API.

Tabla 32: Catálogo de elementos

Diagramas de comportamiento

A continuación, se presentan los diagramas de comportamiento más relevantes de la aplicación como una guía ilustrativa. Cabe destacar que estos diagramas representan flujos correctos sin errores y están destinados a ayudar al lector a comprender mejor el caso que se va a probar, omitiendo pasos que no contribuyan al correcto entendimiento del flujo (Figuras 105 y 106).

Diagrama de secuencia para subida de archivo

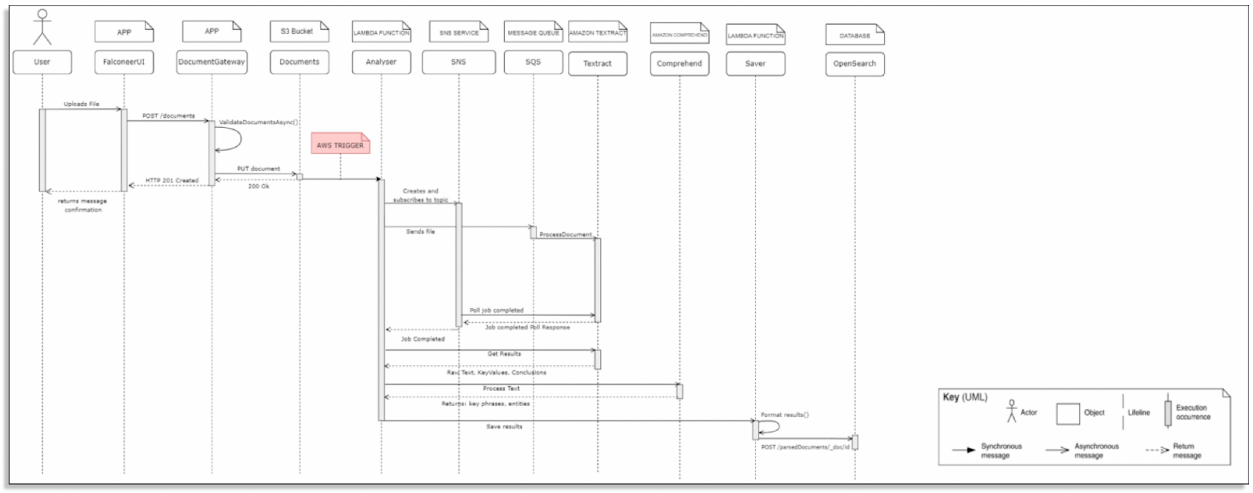


Figura 105: Diagrama de secuencia de subida de archivos

Diagrama de secuencia para búsqueda compleja de documentos según palabra clave

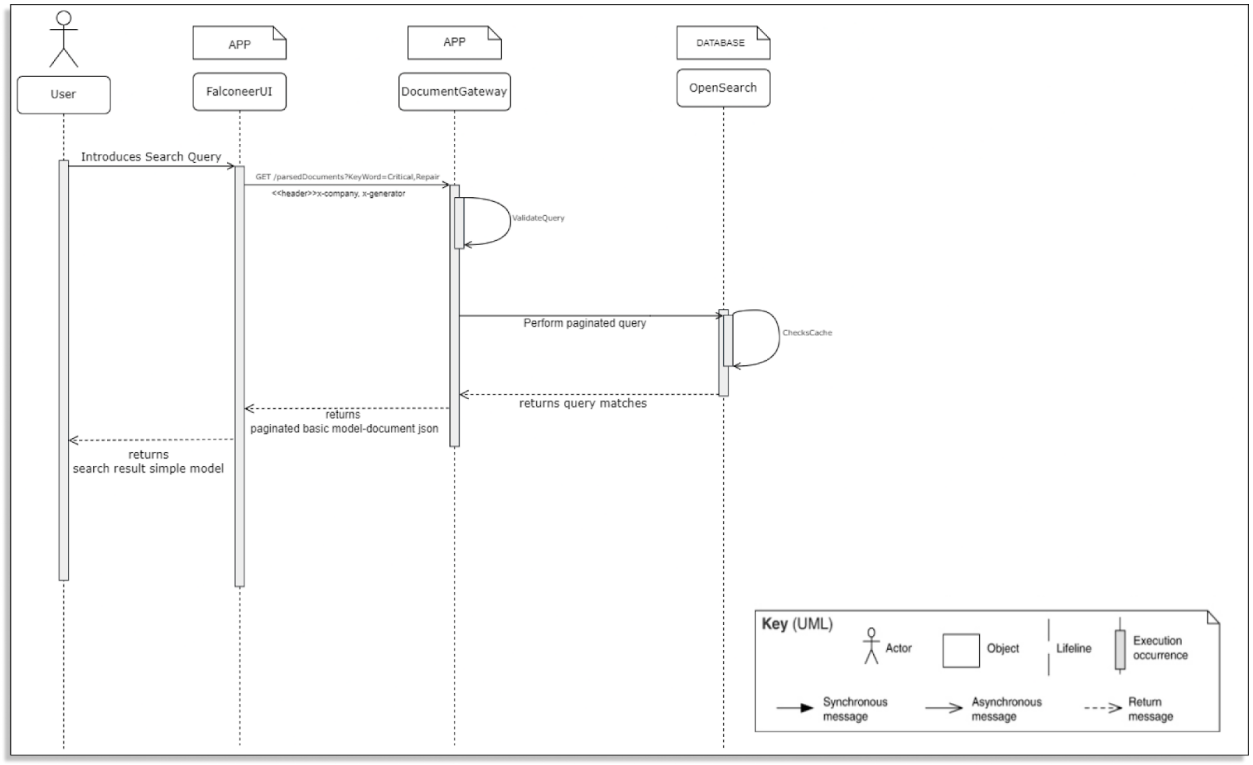


Figura 106: Diagrama de secuencia de subida de archivos

Con esto el lector ya posee un conocimiento básico sobre el funcionamiento de la aplicación como un todo. En los capítulos siguientes se realizará la profundización en módulos particulares.

6. Document Gateway

DocumentGateway es una poderosa API diseñada para manejar una amplia variedad de tareas relacionadas con la administración de documentos. Esta API ha sido construida en .NET Core 6, lo que garantiza un alto rendimiento y escalabilidad.

Con DocumentGateway, los usuarios pueden realizar tareas como la gestión de archivos, la búsqueda y recuperación de información de documentos procesados y la administración de tareas.

Esta API puede ser consumida por otros componentes del sistema al seguir el estilo arquitectónico REST del que se hará detalle junto a sus interfaces más adelante.

7. Vista de módulos

Vista de layers

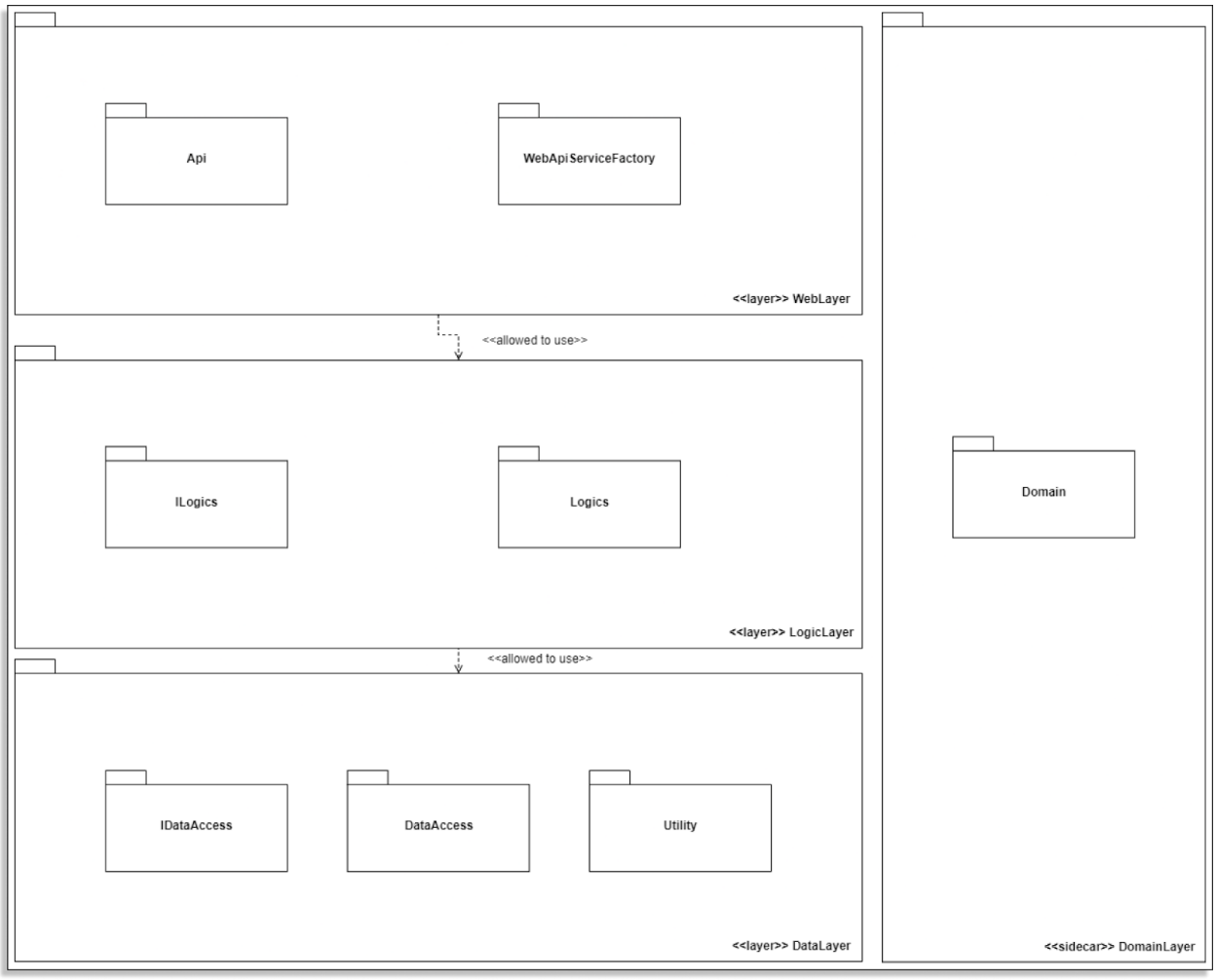


Figura 107: Vista de layers

El patrón de capas (*layered*) divide el sistema de manera que los módulos puedan desarrollarse y evolucionar por separado con poca interacción entre las partes, lo que fomenta la modificabilidad y la reutilización. Para lograr esta separación de responsabilidades, el patrón de capas divide el *software* en unidades llamadas capas. Cada una de estas es un grupo de módulos que ofrece un conjunto cohesivo de elementos. La relación permitida entre las capas debe ser unidireccional a excepción de las colocadas en las capas especiales llamadas *sidecar* que han de ser sumamente consideradas previo a su adicción.

Las capas dividen completamente los módulos, y cada partición se expone a través de una interfaz pública. En el diagrama podemos intuir que ILogic es el módulo que aloja las clases de interfaces para acceder a la lógica por ejemplo, de igual modo para IDataAccess para el acceso a la capa de datos.

Los beneficios de utilizar este patrón han de ser entendidos previo a la realización de algún cambio para garantizar los mismos y son los siguientes:

- La capa de acceso a datos se puede cambiar sin afectar a las capas superiores siempre que la interfaz no cambie, lo mismo para la capa de lógica.

- Las capas inferiores se pueden reutilizar en diferentes aplicaciones, en caso de querer migrar la API.

- Las relaciones permitidas reducen el número de interfaces que cualquier equipo debe comprender. Por ejemplo, con un desarrollador enfocado en integrar un cambio a la lógica no se necesita más que la interfaz para trabajar con el acceso a datos.

Se pueden considerar alguno de los tradeoffs de la aplicación del patrón a la hora de estudiar la performance o el tedio de agregar nuevos servicios que requieran múltiples adiciones de interfaces y cuidados de selección.

Vista de usos

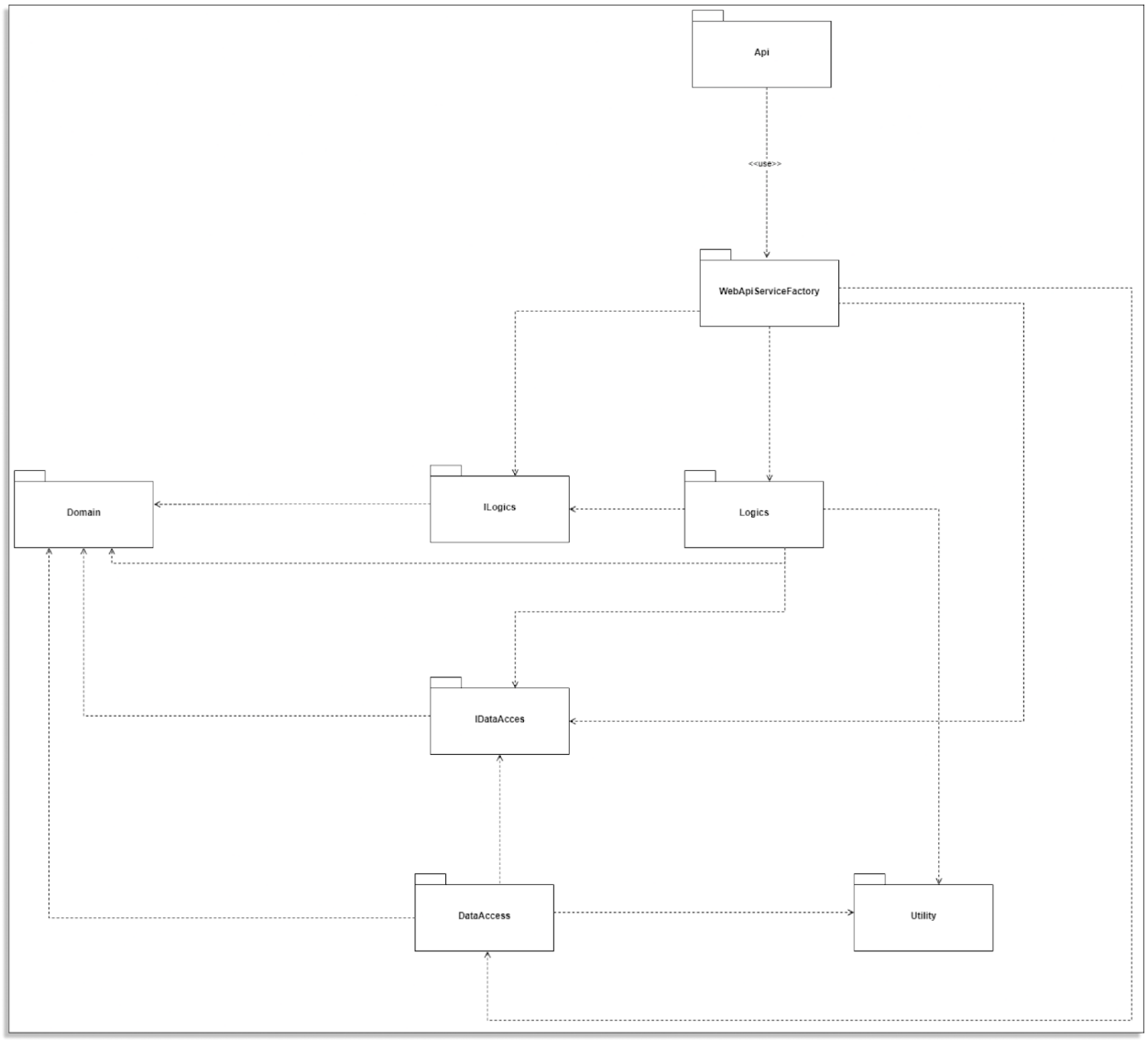


Figura 108: Vista de usos

Uno de los aspectos más llamativos de este diagrama presentado es el alto acoplamiento que tiene “WebApiServiceFactory” con varios módulos de la solución. Este usa y depende de varios de los proyectos del sistema, pero esto es una decisión consciente.

Para quien no esté familiarizado con las últimas versiones de .Net core y C# el marco emplea un sistema inyector de dependencias donde se puede especificar la utilización de una clase para una interfaz dada. El equipo toma este sistema y lo lleva un paso aún más lejos, extrayendo por

completo esta funcionalidad a un paquete externo, exponiendo una única fachada cuyo objetivo es “instalar” todas las clases (que respondan a un tipo abstracto llamado “IInstaller”) del paquete las cuales se toman en base a la técnica de reflection.

Esto es una clara aplicación de la táctica de defer binding, dada la posibilidad de instalar paquetes nuevos o cambiarlos rápidamente (incluso durante tiempo de ejecución) al trabajar únicamente con abstracciones que luego el sistema sabrá inyectar de forma apropiada. Como compensación al ganar esta amplia modificabilidad se pierden unos pequeños milisegundos de performance al llamar a este servicio dada esta búsqueda en tiempo de ejecución.

Retomando la idea de las capas mencionadas previamente podemos ver como los elementos de capas superiores dependen de abstracciones para comunicarse con los módulos de capas inferiores, manteniendo una abstracción de estos servicios a fin de mantener la mantenibilidad del código, en caso que este haya de cambiar en un futuro.

Para más información sobre el sistema de inyección de dependencias o la técnica de reflection referirse a los siguientes enlaces (con contenido en inglés):

- https://learn.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/reflection
- https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection

Principales paquetes NuGet utilizados

NuGet es el gestor de paquetes para .NET que permite la importación de módulos creados por la comunidad en forma de dependencias que podemos sumar a nuestro código. Es importante mantener vista de nuestras dependencias y tenerlas siempre actualizadas (Tabla 33).

NuGet	Responsabilidad	Paquete con dependencia	Versión
AWSSDK.Extensions	Extensiones varias para AWS que permiten integrarse con la configuración de .NET Core y los marcos de inyección de dependencia.	Utility	3.7.2

AWSSDK.S3	Proporciona a los desarrolladores acceso a los servicios de Amazon S3 de forma sencilla.	Utility	3.7.9.31
Bogus	Generador de datos para prueba. Permite la creación de objetos llenos de datos falsos siguiendo estándares definidos por el programador.	Tests	34.0.2
Coverlet.collector	Colector de cobertura de pruebas unitarias	Tests	3.1.2
Microsoft.AspNetCore.Cors	Middleware para la aplicación de políticas de tipo CORS.	Api	2.2.0
Microsoft.AspNetCore.Http.Features	Permite la realización de peticiones HTTP.	Utility	5.0.17
Microsoft.Extensions.Configuration	Facilita la lectura de archivos de configuración como App.Json	Utility	6.01
Microsoft.Extensions.DependencyInjection.Abstract	Librería de inyección de dependencias.	WebApiServiceFactory	6.0.0
Newtonsoft.Json	Paquete que permite la serialización de objetos a formato JSON	Logics, Utility	13.0.2
NSubstitute	Librería para la generación de objetos MOCK en las pruebas unitarias.	Tests	4.4.0
OpenSearch.Client	Librería que permite la realización de consultas a OpenSearch de forma clara y concisa siguiendo el estándar de LINQ.	DataAccess	1.2.0
Serilog.AspNetCore	Soporte para herramienta de logs Serilog.	Api	6.0.1

Swashbuckle.AspNetCore	Soporte para herramienta de generación de documentación para *web* apis llamada Swagger.	Api	6.2.3
Xunit	Marco de pruebas de .net	Tests	2.4.1

Tabla 33: Principales paquetes NuGet utilizados

8. Descripción de principales módulos del sistema

Api

Comenzando por la capa *web*, este módulo (Api) es el que expone nuestros recursos al mundo, al ser una WebAPI representa el patrón fachada para nuestra aplicación. Busca cuidar el uso de mensajes de error apropiados y maneja las excepciones lanzadas por capas inferiores acorde a ellas.

Api es un paquete de nivel superior que contiene controladores, modelos, filtros propios de la solución y rutas, que trabajan juntos para proporcionar una interfaz de programación de aplicaciones clara y coherente para los clientes. Cada uno de estos componentes es esencial para el éxito de la solución, y juntos forman la columna vertebral de la entrada al módulo.

Diagrama de composición

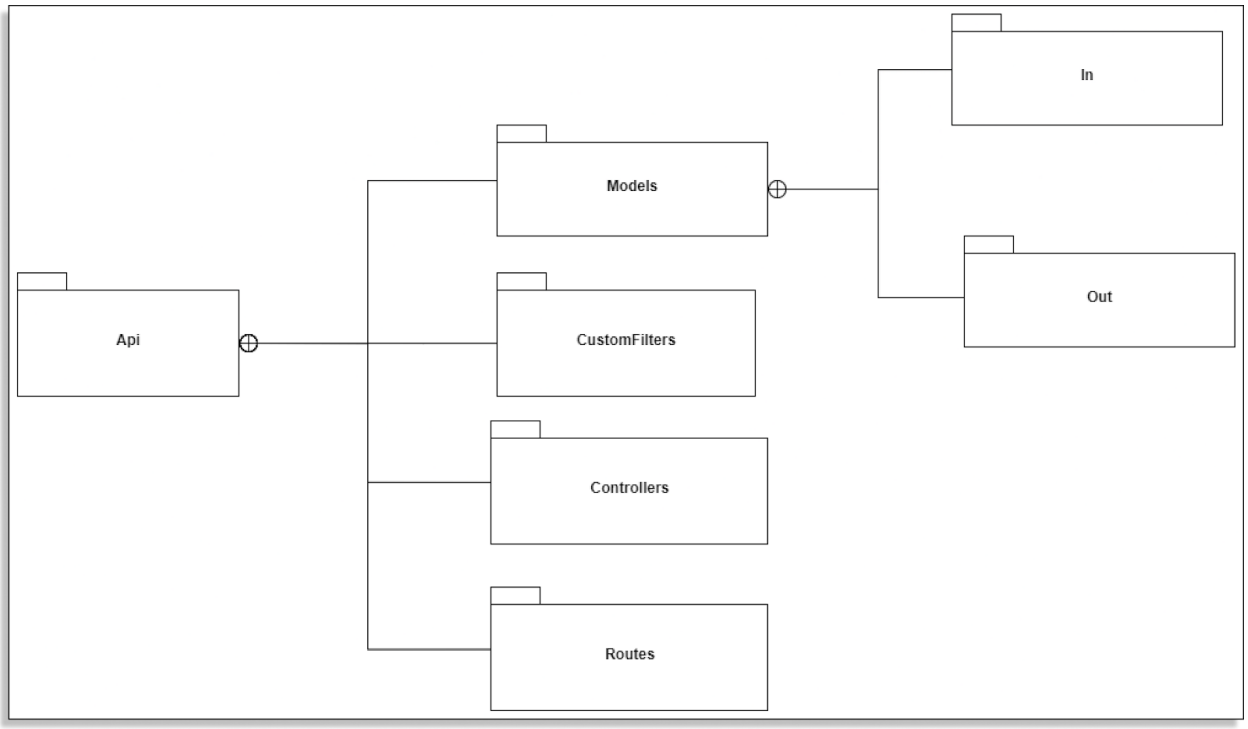


Figura 109: Diagrama de composición

Controllers

Dentro de este paquete tenemos precisamente nuestros controladores, encargados de ser endpoints singulares de la aplicación. Cada uno es una clase que se inicializa con su correspondiente dependencia a una interfaz abstracta de lógica, por ejemplo el controlador de documentos se inicializa requiriendo un IDocumentsLogic.

Cada controlador tiene definidas sus funciones con los correspondientes verbos HTTP que han de ser invocados cuando se los requiera. Encima de cada verbo HTTP encontramos una serie de comentarios con contenido en formato XML que serán utilizados por Swagger para generar la documentación apropiada.

En siguientes secciones se detallarán explícitamente estos *endpoints* y sus usos.

CustomFilters

.Net nos permite el uso de filtros de ejecución propios del lenguaje o directamente los creados por el equipo propios.

Dentro de este paquete encontramos tres filtros relevantes a la aplicación. Primero un filtro de apoyo a la documentación llamado “CompanyAndGeneratorFilter”, este le indica a Swagger ciertos endpoints donde se han de agregar los encabezados “x-company” y “x-generator” a su documentación. El motivo del uso de cabezales propios del sistema se aclara más adelante pero el uso del filtro se limita únicamente a ser un apoyo a la documentación.

LoggerHelper es el segundo filtro y como su nombre lo indica ayuda al formato de los logs del sistema. Aplicando acorde etiquetas y ayudando a hacer el rastreo del origen de la excepción, todo apoyado de la herramienta Serilog.

Finalmente ExceptionFilter es el filtro de más interés para la mayoría de los desarrolladores del sistema. Hereda del filtro nativo de .Net llamado IExceptionFilter y permite la administración de los tipos de excepciones desarrolladas en el sistema, particularmente las “BusinessLogicException”.

Esta excepción posee un código interno de error correspondiente a los códigos HTTP que la API ha de retornar para cada caso, esto es, por ejemplo, un código 404 para una excepción causada por una petición de búsqueda de un elemento inexistente.

La simplicidad de este filtro permite que capas inferiores lancen sus excepciones de error y este haga el trabajo pesado de responder al usuario. En caso de una falla fuera de las controladas, es decir, un error interno, el sistema sabe tratarla para no cortar la ejecución pero informar al usuario con un mensaje de error detallado.

Models

Este paquete contiene vistas en forma de clases que representan, valga la redundancia, modelos de entrada y salida de datos. A destacar algunos objetos como los documentos procesados pueden tener dos vistas diferentes, una simple para cuando se agrupan en listas (como las que

vemos con un GET a todos los documentos de un generador) o complejas cuando se busca en singular y se espera obtener un listado de sus datos procesados.

Algunos modelos poseen validaciones mediante DataAnnotations, que son requeridas o controladas por algún motivo en particular.

Routes

El menos complejo de los paquetes presentados para la API contiene únicamente una clase estática que administra las diversas rutas del sistema. Esto implica que todos los *endpoints* responden a información concentrada aquí. Esto hace que de introducirse una nueva versión de la API se pueda modificar la ruta desde un único lugar o si se desea agregar nuevas también sea sencillo.

Se recuerda al lector que el dominio (*web*) donde se despliega la solución y su puerto no es configurado desde una clase interna, si desde las propiedades de lanzamiento del sistema o desde el *webserver* donde se aloje.

API

Finalmente tenemos el paquete “madre” llamado API, el mismo engloba a todos los anteriores pero además contiene clases de vital importancia como Startup.cs y Program.cs.

Estas clases son de vital importancia para cualquier WebApi escrita en .Net y su funcionamiento es bastante estándar para ambos.

Se notan como excepcionales para Startup.cs:

- Aquí se configura el uso de la política de CORS y la política de uso de cabezales HTTP propios al sistema.
- Se invoca la llamada a WebApiServiceFactory que instala todas las inyecciones de dependencia necesarias.
- Se llama a Swagger y se determina versión, título, descripción y contacto con Renovus.

- Aquí se invoca el uso de ExceptionFilter para toda la Api

ILogics & Logics

ILogics

El paquete ILogics contiene dentro una serie de abstracciones y contenedores pensados para ser reusados o simplificar los pasajes de información. Dentro del mismo encontramos una carpeta con los contenedores de datos a transferir llamados DTO, diseñados para evitar manejar demasiados parámetros en las funciones y para mantener un código limpio y legible acorde a los estándares presentados en el libro Clean Code.

Adicionalmente contiene las interfaces de la lógica que sirven como contratos abstractos para las capas superiores a fin de seguir el principio de inversión de las dependencias de SOLID.

Finalmente el paquete contiene las excepciones que se lanzarán desde las capas inferiores y han de ser conocidas por la capa de presentación para su correcta manipulación.

Logics

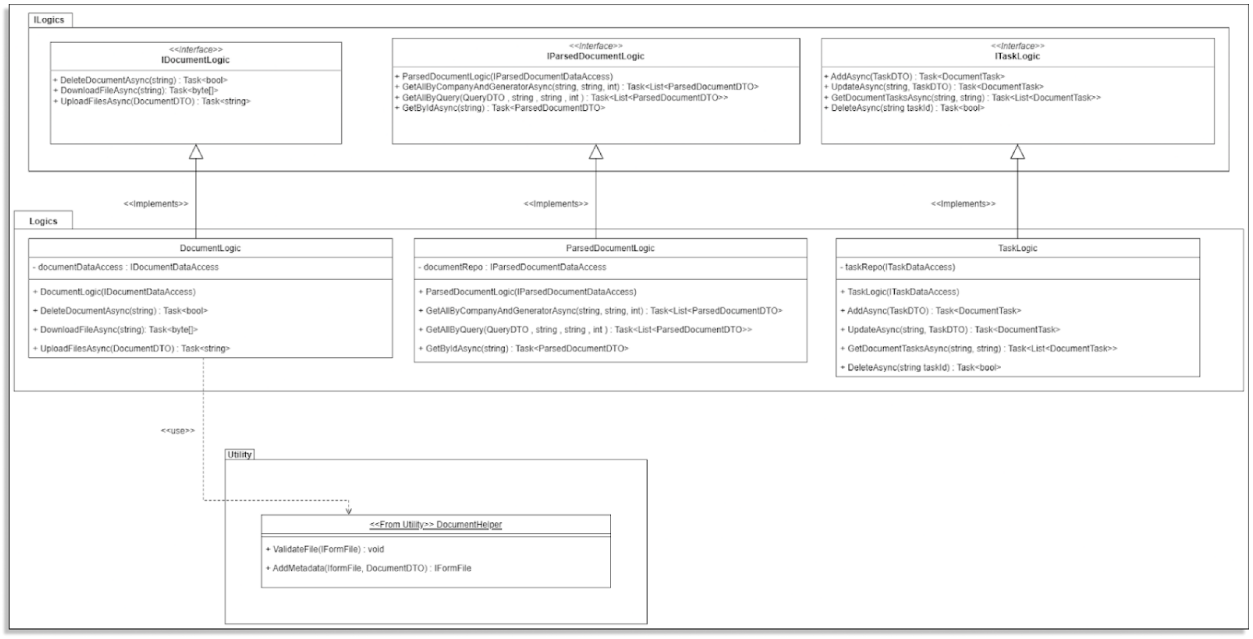


Figura 110: Diagrama UML de clases para Logics

Nota: el diagrama solo abarca aquellas clases necesarias para transmitir el conocimiento básico del paquete. Otras clases y módulos no representados existen en el sistema.

La lógica presenta tres clases principales, correspondientes a documentos, documentos procesados y tareas a nivel de dichos documentos. Cada una implementa su correspondiente interfaz del diagrama y todas reciben por inyección de dependencia (configurada a nivel de WebApiServiceFactory) su correspondiente interfaz de acceso a datos para realizar sus tareas.

Aquí se encuentran todas las responsabilidades relacionadas a las validaciones, modificaciones y manipulaciones de datos propias de la lógica de negocios.

DataAccess

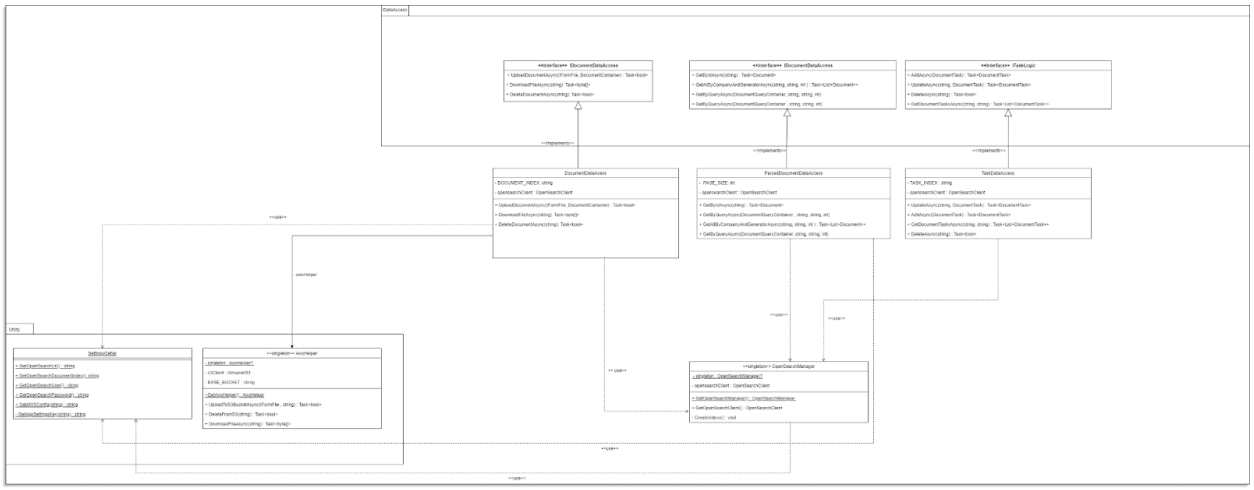


Figura 111: Diagrama UML de clases para DataAccess

Nota: el diagrama solo abarca aquellas clases necesarias para transmitir el conocimiento básico del paquete. Otras clases y módulos no representados existen en el sistema.

El paquete DataAccess es uno de los más importantes en el proyecto, pero también uno de los más complicados de entender. Su importancia radica en que se encarga de la gestión de acceso a datos de la aplicación. Contiene consultas en lenguaje LINQ que se apoyan en la biblioteca OpenSearchClient, lo que le permite una mayor eficiencia y rapidez en la gestión de tareas en la base de datos.

Una de las características más interesantes de DataAccess es el uso de la Clase OpenSearchManager como aplicación del patrón singleton. Esto significa que, a través de ella, se simplifica el código necesario para obtener una instancia del OpenSearchClient (cliente de la librería homónima). Se cuenta con un caché interno a este cliente, lo que agiliza el acceso a los datos de la aplicación.

Otro beneficio del OpenSearchManager creado por el equipo es la generación de los esquemas del OpenSearch de la solución, que se disparan como chequeo de índice al realizar una consulta. Esto garantiza la efectividad de las consultas y se asegura de que el índice está correctamente creado previo a su uso.

Como se aprecia en el diagrama se hace uso de dos clases de Utility, una clase estática llamada SettingsGetter que, como indica su nombre, le permite obtener configuraciones como el nombre de usuario o los índices en los que se va a trabajar y AwsHelper, otro singleton que apoya en la subida de archivos al bucket S3.

9. Manejo de excepciones

Cada paquete de interfaz (contratos) tiene consigo un conjunto de excepciones que han de ser conocidas por aquel módulo que requiera de usarlo. Por ejemplo ILogics tiene las excepciones llamadas BusinessLogicException. El objetivo de estas excepciones es que cuando sean lanzadas las capas superiores tengan la responsabilidad de capturarlas y actuar acorde.

Las excepciones lanzadas por la capa de lógica presentan una entidad propia que representa un código de error. Estos códigos se corresponden con los HTTP que ha de lanzar la aplicación. En la WebApi se incluye un filtro que ataja dichas excepciones con código propio y las retorna como respuesta. El uso y aplicación de estas excepciones capturables por filtros mejora enormemente la prolijidad del código de los controladores de la *web* api y además aumenta el performance al cortar la ejecución de un hilo que eventualmente retornaría un error. Aquellas excepciones que no hayan sido lanzadas por un desarrollador del sistema (un error no controlado) son registradas en la consola a modo de log y responden al cliente un error de código 500 “Internal server error” sin cortar la ejecución del programa.

10. Vista de componentes y conectores

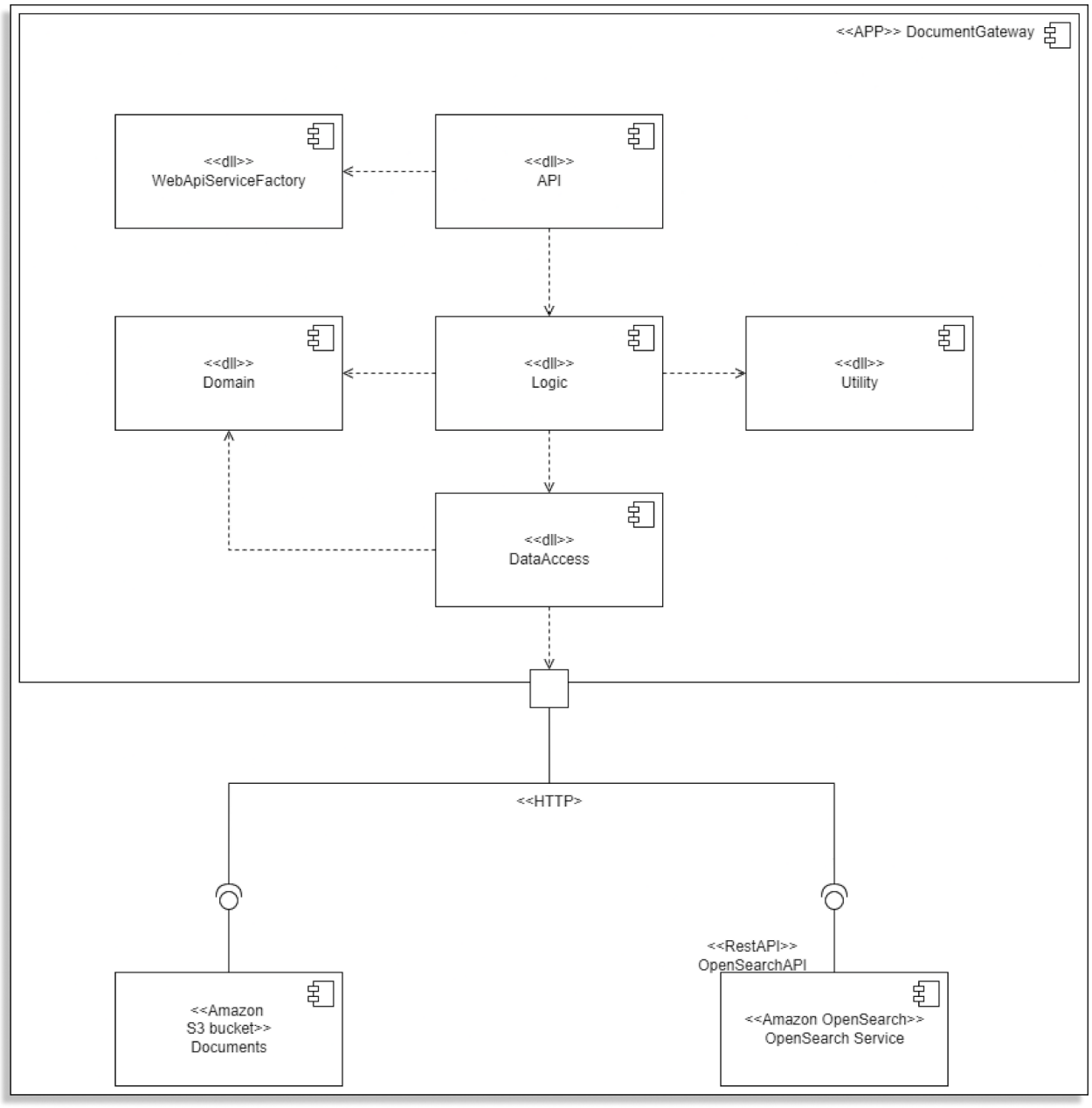


Figura 112: Vista de componentes y conectores DocumentGateway

11. Especificación de la API

Al dirigirnos a "/swagger/index.html" dentro de la API nos encontramos con una pantalla interactiva de documentación de los endpoints de la misma. Se encuentran divididos en "Documents" para aquellos documentos en formato PDF, "ParsedDocuments" para documentos procesados por el sistema y finalmente "Tasks" para las tareas. Dentro de esta sección también se documenta el tipo de respuesta que hemos de esperar (Figuras 113 y 114).

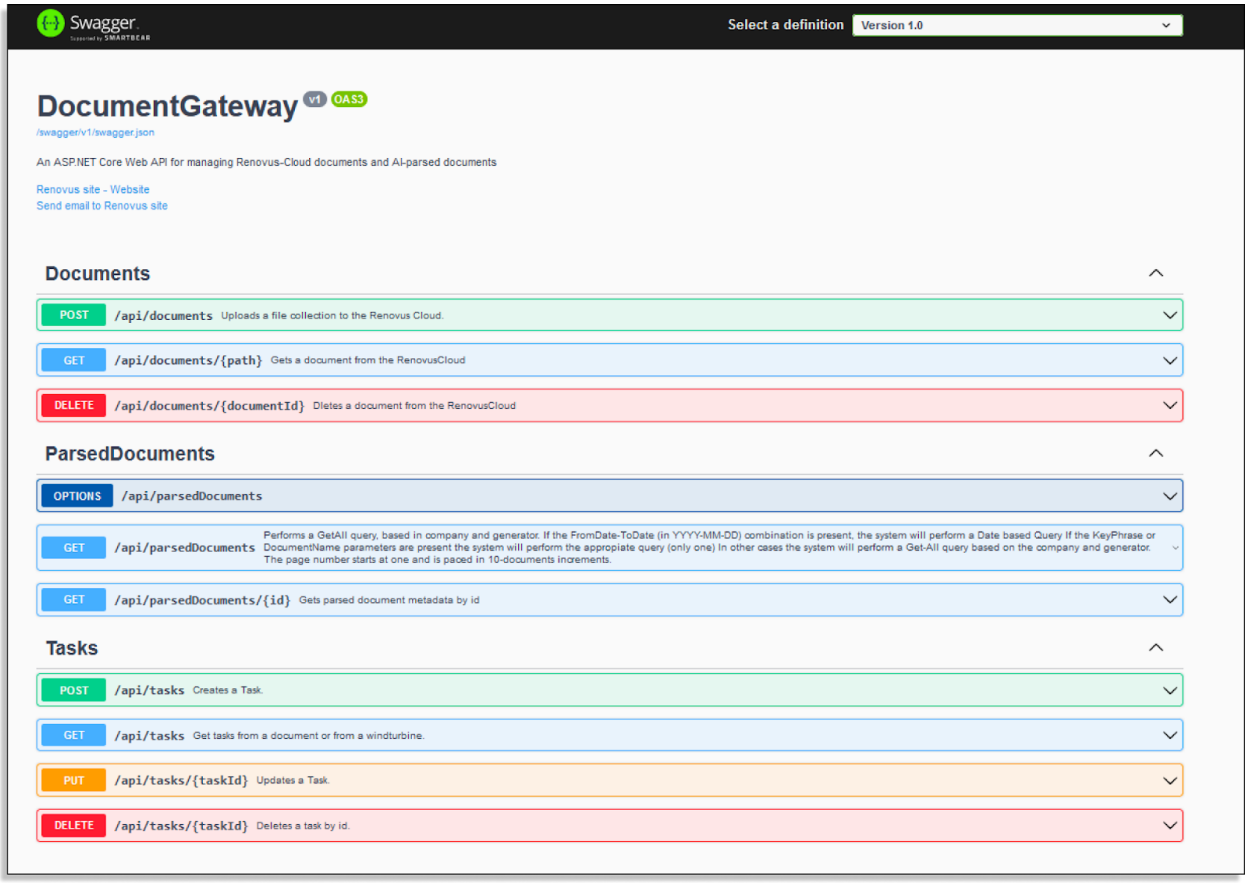


Figura 113: Vista de Swagger para DocumentGateway

Ejemplo de especificación de endpoint:

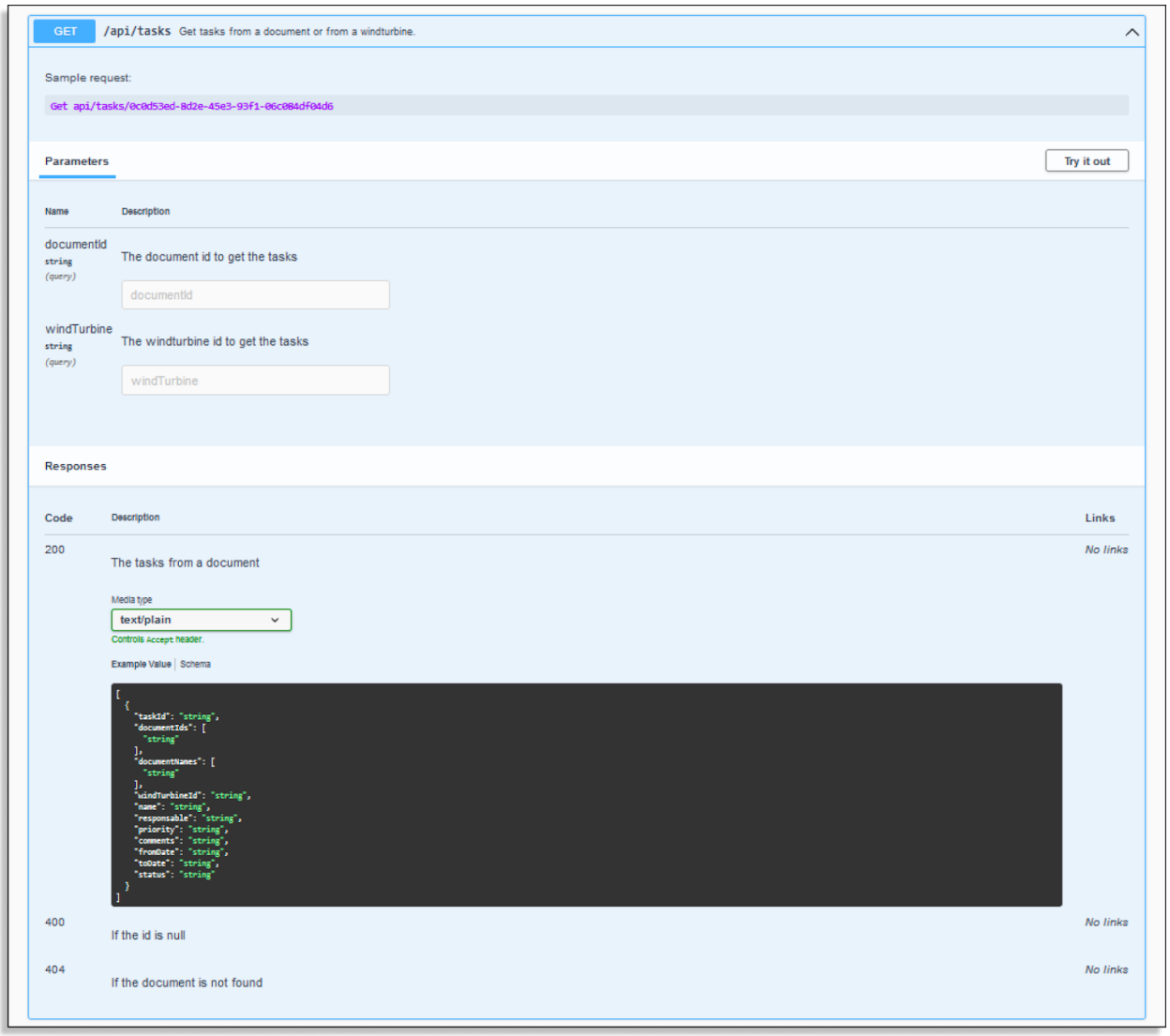


Figura 114: Vista de *endpoint* para obtención de tareas en Swagger

12. Sobre pruebas

Para la creación de pruebas unitarias se utilizó Nsubstitute, una biblioteca de .NET que permite a los desarrolladores crear objetos simulados. Estos objetos simulados se utilizan en lugar de los objetos reales en el código de prueba, lo que permite a los desarrolladores aislar el código y probarlo de manera efectiva sin depender de otras partes del sistema (Figura 115).

```
1 reference
public class ParsedDocumentControllerGetParsedDocument
{
    private readonly IParsedDocumentLogic parsedDocumentLogic = Substitute.For<IParsedDocumentLogic>();
    private readonly ParsedDocumentsController parsedDocumentController;
    private const int OK_REQUEST_RESULT = 200;


    0 references
    public ParsedDocumentControllerGetParsedDocument()
    {
        parsedDocumentController = new ParsedDocumentsController(parsedDocumentLogic);
    }

    [Fact]
    0 references
    public async Task GetDocumentOkCase_CompleteModel()
    {
        var mockedParsedDocument = DocumentTestingUtils.GenerateDummyParsedDocument();
        parsedDocumentLogic.GetByIdAsync(Arg.Any<string>())
            .Returns(mockedParsedDocument);

        var result = await parsedDocumentController.Get("fileTest.pdf", false);
        var okResult = result as ObjectResult;

        Assert.NotNull(okResult);
        if (okResult != null)
            Assert.Equal(OK_REQUEST_RESULT, okResult.StatusCode);
    }
```

Figura 115: Ejemplo de pruebas unitarias

En la captura observamos cómo al comenzar se sustituye la clase de la lógica por un sustituto, permitiendo la simulación de respuestas controladas. Adicionalmente vemos en la prueba el uso de una clase estática llamada DocumentTestingUtils. Esta clase, junto a otras sirven de apoyos para la generación de elementos "realistas" en base a un conjunto de reglas mediante la herramienta Bogus (Figura 116).

```
24 references
public static class DocumentTestingUtils
{
    1 reference
    public static Document GenerateDummyDocumentAllSameCompanyAndGenerator(string company, string generator)
    {
        Dictionary<string, List<string>> bogusEntities = GenerateMockedEntities();
        Dictionary<string, string> mockedKeyValuePairs = GenerateMockedKeyValues();
        List<string> mockedKeyPhrases = new() { "0001 Reported", "RDF33", "20:1", "Application" };

        return new Faker<Document>()
                            .RuleFor(x => x.RawText, x => x.Lorem.Text())
                            .RuleFor(x => x.Bucket, x => x.Internet.Url())
                            .RuleFor(x => x.Categories, x => x.Lorem.Word())
                            .RuleFor(x => x.CompanyName, x => company)
                            .RuleFor(x => x.Date, x => x.Date.Weekday())
                            .RuleFor(x => x.Entities, bogusEntities)
                            .RuleFor(x => x.FileName, x => x.System.FileName("pdf"))
                            .RuleFor(x => x.Generator, x => generator)
                            .RuleFor(x => x.Id, x => Guid.NewGuid().ToString())
                            .RuleFor(x => x.KeyValues, x => mockedKeyValuePairs)
                            .RuleFor(x => x.KeyPhrases, x => mockedKeyPhrases)
                            .Generate();
    }
```

Figura 116: Ejemplo de generación de documentos "mock" mediante Bogus.

Es importante conocer el funcionamiento de las dos bibliotecas mencionadas si se desea realizar alguna modificación sobre las pruebas unitarias existentes.

13. Análisis de métricas

A modo de registro se agrega en este documento los resultados obtenidos en las métricas de abstracción e inestabilidad previo a la fecha de entrega.

- Si un *assembly* tiene muchos tipos abstractos (interfaces y clases abstractas) y pocas concretas, es considerado abstracto.

- Luego, si un *assembly* es usado por otros se considera inestable y el problema de lo mismo es que se convierte en algo difícil de modificar. El equipo está conforme con los resultados obtenidos en el análisis métrico de abstracción y estabilidad.

El único paquete que rompe la norma es la de dominio, aunque dada su importancia en el desarrollo guiado por este es esperable que se encuentre en una zona donde si se realiza algún cambio tenga un impacto considerable. La siguiente gráfica fue realizada con la herramienta NDepend (Figura 117):

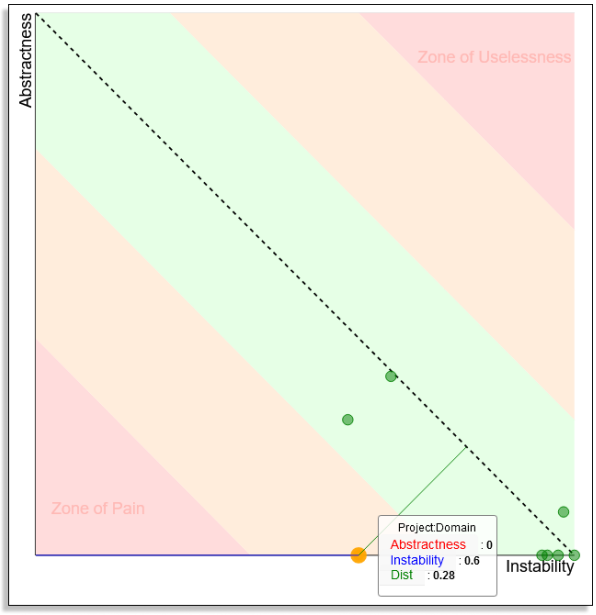


Figura 117: Análisis de métricas de abstracción e inestabilidad

Es importante para un desarrollador que mantiene un código conocer el estado de sus métricas de abstracción e inestabilidad porque estas métricas pueden ayudar a evaluar la calidad del código y guiar la toma de decisiones en el proceso de mantenimiento y evolución del *software*. Al monitorear estas métricas, el desarrollador puede identificar problemas potenciales en el código, como clases o módulos que pueden requerir refactorización para mejorar su abstracción o dependencias externas que pueden ser demasiado inestables. Con esta información, el desarrollador puede tomar medidas proactivas para mejorar la calidad del código y evitar problemas futuros durante el mantenimiento del *software*.

14. Renovus *Web* - Falconeer UI

Dado que el sitio *web* ya contenía un flujo de trabajo claro, solo se hará mención de lo agregado, las nuevas páginas creadas, distribución de carpetas, librerías utilizadas y estilo de codificación.

El objetivo fue preservar el estilo de programación ya utilizado, manteniendo las librerías y la distribución de las carpetas ya definidas.

Responsabilidades

Con motivo de agregar las nuevas funcionalidades, se crearon 3 nuevas carpetas, cada una asociada a la funcionalidad requerida, estás siendo **documents** (vista de documentos, búsqueda, filtrado y detalles de documentos), **upload** (vista de subida de documentos) y por último **tasks** (vista de tareas de un documento y de un generador). Se mantuvo el formato de agregar todo lo asociado a nuevas pantallas en subcarpetas separadas dentro de la carpeta **pages** (Figura 118).

Figura 118: Distribución de vistas por carpetas

Distribución por carpetas

Dentro de cada carpeta asociada a una pantalla se crearon dos subcarpetas dedicadas a los componentes y funciones auxiliares. A continuación la utilidad de cada una:

- **Components:** Esta carpeta es la encargada de contener componentes que podrían ser reutilizados o que su implementación opacaría demasiado espacio en el código de la pantalla principal.

- **Utils:** Es una abreviación de *utilities* lo cual refiere a utilidades que se pueden utilizar en el código fuente, esto es por ejemplo constantes, textos o estilos. El objetivo de esta subcarpeta fue el de mantener un código más legible en el archivo de la pantalla. Al usar esta metodología, el cambiar estilos o textos se vuelve mucho más simple y accesible, también aporta la oportunidad de reutilizar valores que se utilizan en varias partes del código.

En último lugar, se hallan los archivos de las pantallas, que contienen el código correspondiente a la interfaz gráfica de usuario. A diferencia de los archivos de las carpetas mencionadas anteriormente, estos están definidos en la carpeta de rutas del proyecto y sirven como punto de entrada a la vista (Figura 119).

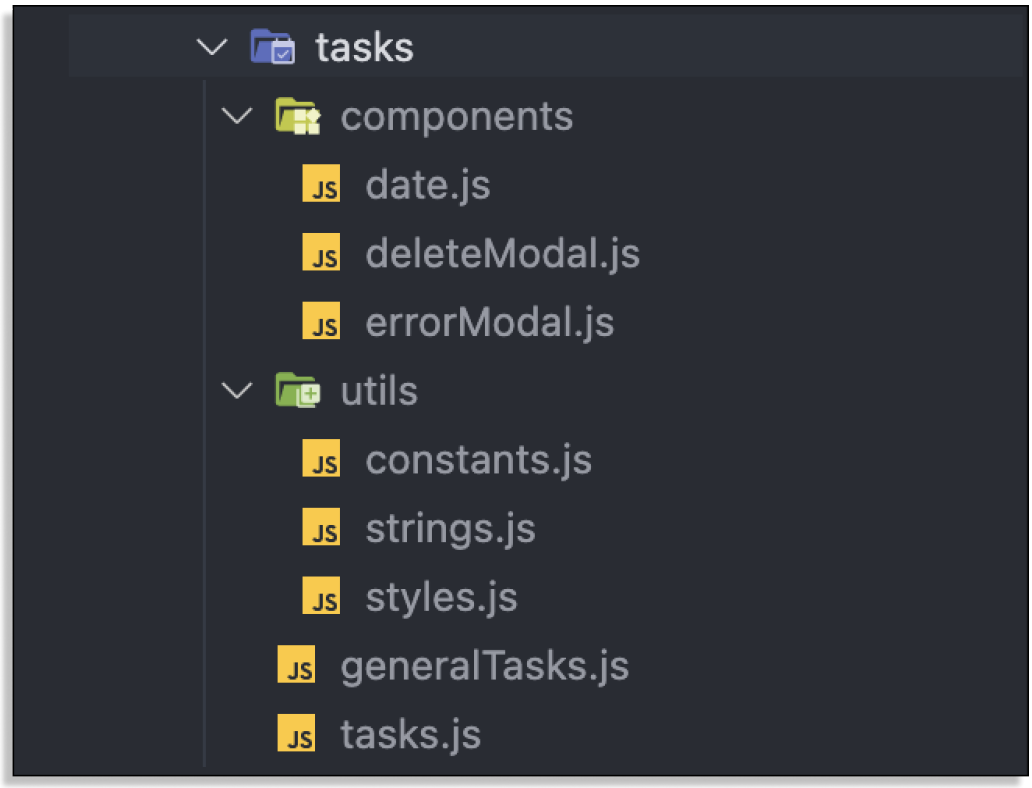


Figura 119: Distribución de carpetas dentro de cada funcionalidad

Librerías utilizadas

Se mantuvo la utilización de la librería **@coreui/react** para las vistas de pantallas. Ya el proyecto contenía todo lo necesario para poder realizar las pantallas siguiendo el formato de la páginas anteriores, por ende no fue necesario agregar nuevas librerías asociadas a estilos.

Codificación

Se mantuvo el mismo estilo de codificación utilizado en la gran mayoría del proyecto, utilizando estados para diferentes variables, manteniendo el código legible, prolijo y ordenado por responsabilidades.

Con motivo de mantener el orden se delegaron los estilos a una carpeta externa, así como también los textos y las constantes. El objetivo es que el JSX se vea prolijo para la detección rápida de errores, así como también para poder reutilizar estilos y textos. El proporcionar una carpeta ajena de textos ayuda a que a futuro, de ser necesario integrar un nuevo idioma al proyecto, se pueda realizar de manera rápida y ordenada (Figura 120).

Figura 120: Ejemplo de codificación

15. Analyzer

Esta instancia de Lambda es una de las dos que se utiliza en el sistema, y es la más compleja debido a que contiene la mayor responsabilidad en cuanto al procesamiento de los documentos y la manipulación inicial de los resultados obtenidos. Analyzer es invocado mediante un disparador (*trigger* de AWS) previamente configurado en el S3, que se ejecuta cada vez que se sube un documento a este *bucket*.

Esta instancia tiene una alta complejidad ya que es responsable de diversas tareas críticas. En primer lugar, debe buscar el documento a manipular, lo que sucede en el momento mencionado anteriormente. Luego, establece la conexión con Amazon Textract y Amazon Comprehend, además de otros módulos que se utilizan durante el proceso, como una cola de notificaciones y una cola para almacenar las respuestas de Amazon Textract que indican que el procesamiento ha finalizado. Después de los procesamientos, esta instancia busca la metadata del documento, la cual es utilizada por la instancia del Saver para unificar toda la información y procesarla, dejando

solo los datos relevantes para el sistema. Por último, se encarga de enviar los datos al Saver para que continúe su procesamiento.

A grandes rasgos, este módulo se puede dividir en cuatro partes: la conexión y procesamiento de Amazon Textract, la conexión y procesamiento de Amazon Comprehend, procesamiento y transformación de los resultados obtenidos, preparación y envío de información al Saver.

Antes de analizar cada parte de esta instancia Lambda, es importante hacer una aclaración sobre la organización del proyecto que se encuentra en GitHub. El proyecto se divide en carpetas según el módulo correspondiente, como Textract y Comprehend, y fuera de ellas se encuentra la clase de la lógica, llamada "lambda_function", y el archivo de configuración. Sin embargo, no fue posible replicar esta estructura en la instancia de Lambda dentro de AWS debido a errores que surgieron y que, según la investigación del equipo, podrían deberse a aspectos del editor de texto o a la implementación del soporte de Python en los Lambda. Por esta razón, al pasar el proyecto a la instancia de Lambda dentro del entorno de Renovus, se optó por no utilizar carpetas y crear los archivos juntos en el mismo directorio del Lambda.

Comenzamos por la sección relacionada con Amazon Textract en este submódulo de la solución. En esta sección, encontramos dos clases. La primera clase es responsable de la conexión y comunicación con Amazon Textract. Al analizar esta clase, notamos que se divide la comunicación en partes más pequeñas para facilitar el mantenimiento en el futuro. En primer lugar, se encuentran los módulos necesarios para trabajar con Textract que deben importarse. Luego, se presenta la lógica necesaria para instanciar la clase, que incluye el constructor con los parámetros requeridos y la instanciación de los clientes del servicio, la cola de notificaciones y la cola de mensajes. Después de esto, se presentan dos métodos relacionados con las colas de mensajes y de notificaciones. El primer método se encarga de crear cada instancia utilizando los clientes previamente instanciados en el constructor, definiendo las políticas y suscripciones necesarias. En el segundo método, se eliminan estas instancias para evitar costos innecesarios y solo mantener las instancias requeridas. Debajo de estos métodos, encontramos dos funciones auxiliares, creadas para separar las responsabilidades y mejorar la organización. La primera función se encarga de registrar la respuesta de Amazon Textract en los logs, lo que permite

explorar los resultados en los casos necesarios. La segunda función convierte el texto crudo (*raw-text*) en una respuesta que puede ser analizada con Amazon Comprehend.

Uno de los aspectos más importantes de esta clase es la lógica encargada de obtener los resultados del análisis realizado. Esta función realiza una petición a la instancia de Amazon Textract para obtener la información sobre el trabajo realizado. Para esto, se envía el identificador del trabajo, junto con dos valores correspondientes a un token de paginación (en caso de ser necesario) y una cantidad máxima de resultados. A medida que se reciben los bloques de respuesta, se procesan utilizando un algoritmo en la otra clase de este submódulo que se describirá más adelante. Este algoritmo arma los pares clave-valor basándose en el procesamiento realizado. A continuación, se ensambla el texto crudo y se continúa el procesamiento hasta que no hay más respuestas que procesar. En este punto, se debería haber ensamblado el texto crudo y los pares de clave-valor.

Por último, encontramos la función que envía el documento para su procesamiento. Después de enviar el documento, se espera la respuesta de Textract en las colas. Cuando se recibe la notificación de que el trabajo ha finalizado, se busca el identificador en la cola de mensajes. Este identificador es necesario para la función que busca y procesa la respuesta, como se mencionó anteriormente. Después de obtener el identificador, se llama a la función mencionada anteriormente para procesar la respuesta y devolver los valores finales.

Dentro de este submódulo, y cómo se adelantó en la parte anterior, hay otra clase, la cual está compuesta únicamente por la lógica correspondiente al algoritmo necesario para transformar la respuesta de Amazon Textract vinculada a los pares clave-valor obtenidos del análisis del documento. Esta transformación es necesaria para convertir la respuesta en un objeto que se pueda trabajar dentro de la instancia de Lambda para su posterior envío y almacenamiento en la base de datos.

El siguiente submódulo que se presenta es el de Amazon Comprehend, el cual es una clase que maneja la lógica de conexión y llamados a los diferentes servicios que ofrece este módulo. Esta clase tiene una estructura similar a la lógica de Amazon Textract que se detalló anteriormente. Al comienzo de la misma se encuentran las importaciones necesarias para el correcto funcionamiento de la lógica y para establecer la conexión con Amazon Comprehend. Luego, se

encuentra el constructor que inicializa los aspectos necesarios y se instancia el cliente para la conexión con Comprehend.

Después del constructor, se realizan llamados a las diferentes funcionalidades de Comprehend, que incluyen la identificación del idioma, las entidades relevantes del sistema, las frases clave, la identificación de información personal y la detección/clasificación del sentimiento del documento (ya sea positivo, negativo, etc.). Excepto para la detección del lenguaje, donde solo se necesita pasar el texto a analizar, los demás llamados incluyen tanto el texto a analizar como el idioma correspondiente. Por lo tanto, es necesario realizar primero un análisis del idioma del texto en cuestión para luego utilizar ese idioma en los demás servicios.

A diferencia del módulo de Amazon Textract, Comprehend devuelve los análisis de manera rápida, por lo que no es necesario utilizar una cola de mensajes ni notificaciones. Una vez que se realiza el llamado y procesamiento por parte de este módulo, cada función devuelve la respuesta de Comprehend para ser tratada en el último submódulo de este componente de la solución.

Finalmente, hemos llegado al último submódulo que se compone de dos componentes. El primer componente es el archivo de configuración que contiene la región de los servidores de AWS que se está utilizando, un identificador determinado como rol ARN utilizado para llamar a la instancia del Saver, y el porcentaje mínimo aceptado para los valores analizados por el módulo de Amazon Comprehend. El segundo componente es la función Lambda, donde se integran todos los submódulos anteriores para que esta instancia de Lambda pueda cumplir con su responsabilidad.

Al analizar este último componente, se puede ver que, al igual que con el resto de las clases de lógica, al principio, contendrá la importación de aquellos módulos necesarios para el correcto funcionamiento del componente. También se instancian los dos clientes necesarios para este módulo: el cliente para conectar con el *bucket* de S3 para obtener los documentos y el cliente para hacer la conexión con el Saver.

En la siguiente parte de la clase, se establece la conexión con S3 para obtener el documento y poder comenzar con los análisis correspondientes. Después de esto, se inicia el análisis y procesamiento del documento, comenzando por Textract. Se utiliza el constructor para

instanciarlo, creando las colas de mensajes y notificaciones, llamando al método encargado de procesar los documentos y finalmente borrando las colas.

Posteriormente, se comienza el análisis del Comprehend. Aquí, la división es parecida, comenzando por la utilización del constructor para instanciar el módulo. Luego, se procede con el análisis de lenguaje, para finalizar con las entidades y las frases clave. Una vez recabada toda la información, se procede a transformar los datos obtenidos, los pares clave-valor, las frases clave y las entidades, adaptando dicha información al formato necesario y utilizando también el filtro para los resultados de Comprehend.

Completada la transformación, se ingresa en la última parte de esta lógica. Aquí, se utiliza el cliente de S3 para la obtención de la metadata del documento, se define el objeto que será enviado al módulo Saver, y posteriormente se envía a este para continuar su procesamiento y el guardado en la base de datos.

16. Saver

Cómo se adelantó anteriormente, este módulo de la solución corresponde a una instancia Lambda de AWS, donde se encuentra el código responsable de formatear toda la información procesada, establecer la conexión con la base de datos utilizada, y realizar el envío de la información. A diferencia del Analyzer, este módulo se compone únicamente de una clase, y de un archivo de configuración que contiene los parámetros necesarios para la conexión a la base de datos, y donde se deben realizar las modificaciones en caso de ser necesario.

Respecto a la lógica de este módulo, podemos dividirla en 4 partes para analizarlo de mejor manera. En la parte inicial del código, encontramos todas las importaciones necesarias para la implementación realizada. Estás, mayormente, corresponden a módulos para la conexión con la base de datos, pero también existen módulos necesarios en Python para trabajar con diferentes elementos cómo los JSON. En la segunda parte del código se puede encontrar todo lo relacionado con el establecimiento de la conexión a la base de datos. Aquí encontraremos el código que va al archivo de configuración para obtener los parámetros cómo son el host, la región, el *index* de la base de datos, y las credenciales de acceso, para luego unificarlos todo en una instancia de cliente.

Ingresando en la tercera parte de la lógica, encontramos el inicio de la función, donde si analizamos detenidamente veremos que es donde se trabaja la información obtenida por el módulo anterior para convertirla al modelo definido para la base de datos. Primero se trabaja en la remoción de los prefijos que AWS incluye en algunos atributos de la metadata, además de eliminar los valores que no serán tenidos en cuenta para este procesamiento. Luego, se procede a convertir toda la información que el módulo anterior compartió a un JSON, que luego será asignado a un nuevo objeto donde encontraremos todos los atributos que contendrá cada elemento en la base de datos con su formato correspondiente. Finalmente, en el apartado final de este código, podremos encontrar el establecimiento de la conexión de la instancia Lambda con la base de datos, y el envío de la información ya procesada y convertida al formato final que se definió.

10.6 Ejecución de las iteraciones

Logros obtenidos a lo largo de cada Sprint. A continuación se podrá ver un infograma realizado para representar hitos del proyecto en cada Sprint (Figuras 121, 122 y 123).

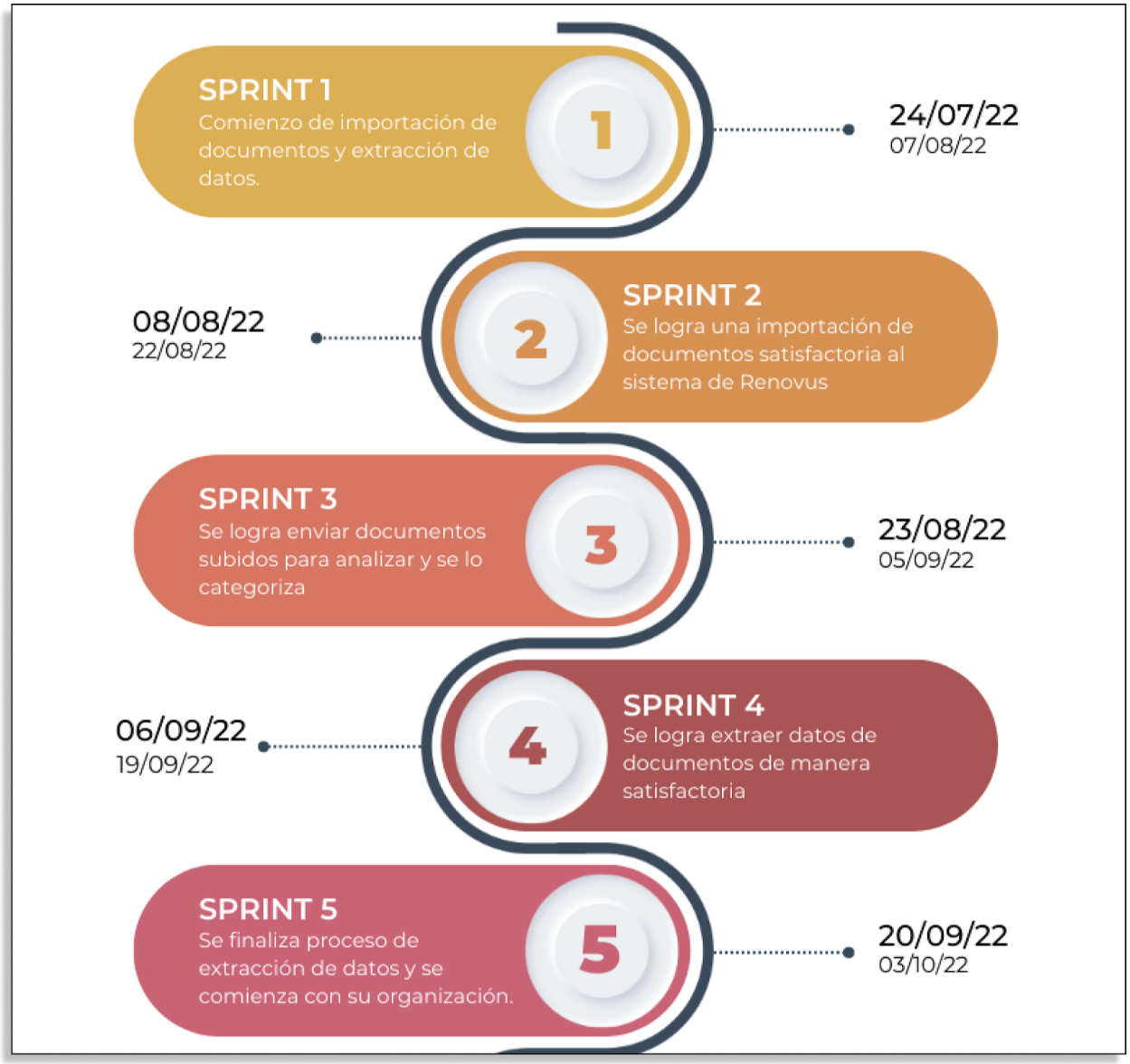


Figura 121: Primera parte infograma de Sprints

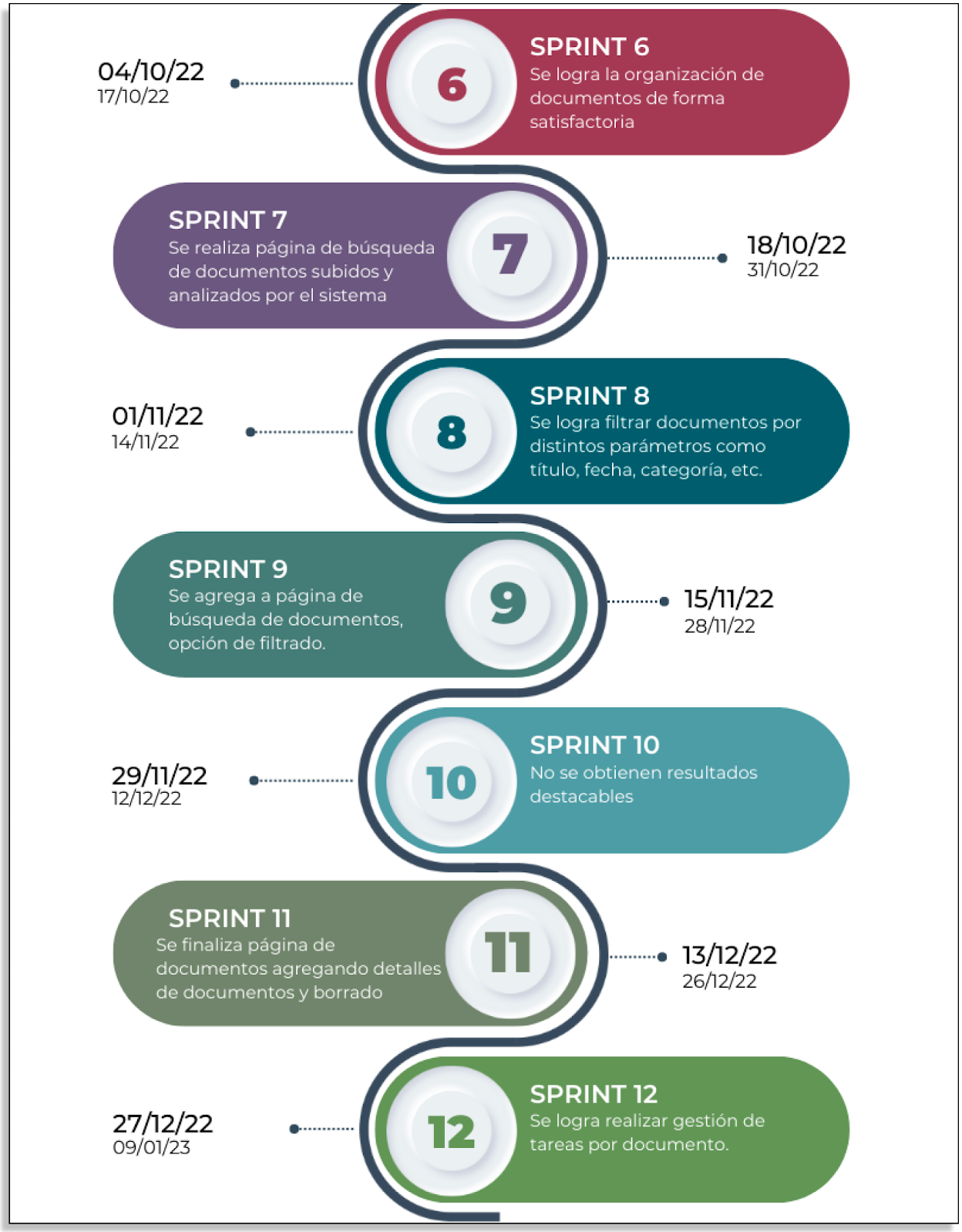


Figura 122: Segunda parte infograma de Sprints

Figura 123: Tercera parte infograma de Sprints

A continuación, se exponen de manera detallada los periodos, objetivos y resultados más destacados correspondientes a cada Sprint.

Sprint 1

- Periodo: 24/07/22 a 07/08/22
- Objetivos: Desarrollar la importación de documentos y comienzo de extracción de datos de documentos.
- Resultados destacados: El equipo logró comenzar con la importación de documentos, pero fue muy optimista al momento de estimar la cantidad de puntos totales del Sprint con un total de 21 puntos esperados y 10 obtenidos. Este suceso ayudó a establecer la velocidad del equipo para los próximos Sprints.

Sprint 2

- Periodo: 08/08/22 a 22/08/22
- Objetivos: Desarrollar la importación de documentos y extracción de datos de documentos.
- Resultados destacados: Se logró importar documentos desde el sistema de Renovus y luego enviarlos al analizador de documentos. Por otra parte, la extracción de datos de documentos comenzó a dar sus primeros pasos hacia una extracción completa.

Sprint 3

- Periodo: 23/08/22 a 05/09/22
- Objetivos: Extracción de datos de documentos y categorización de documentos
- Resultados destacados: El equipo logra la exportación de pequeños datos de un documento completo de manera satisfactoria y logrando así parcialmente la exportación de datos. Se realiza la categorización de documentos tanto en *frontend* como *backend* permitiendo la selección de categorías particulares especificadas por el cliente.

Sprint 4

- Periodo: 06/09/22 a 19/09/22
- Objetivos: Extracción de datos de documentos y organización de documentos.
- Resultados destacados: Se logra extraer todos los datos necesarios de un documento de manera satisfactoria, así logrando uno de los grandes hitos del proyecto. También se comienza con la gestión de documento

Sprint 5

- Periodo: 20/09/22 a 03/10/22
- Objetivos: Finalizar extracción de datos de documentos y organización de documentos
- Resultados destacados: El equipo logra finalizar con la importación y extracción de datos de documentos, así generando un sistema capaz de subir documentos, los cuales son analizados para así poder exportar datos importantes. Por otra parte, al finalizar el Sprint se realiza el primer *release* pudiendo así mostrar al cliente cómo funciona la importación de documentos al sistema y la ejecución del análisis y exportación de datos.

Sprint 6

- Periodo: 04/10/22 a 17/10/22

- Objetivos: Organización de documentos y comienzo de búsqueda rápida de documentos
- Resultados destacados: El equipo finaliza la organización de documentos, permitiendo así que el sistema organice los documentos importados y con datos extraídos, para luego poder implementar la búsqueda rápida de los mismos.

Sprint 7

- Periodo: 18/10/22 a 31/10/22
- Objetivos: Búsqueda rápida de documentos
- Resultados destacados: Se logra realizar una página de documentos que muestre todos los documentos analizados en una tabla, así como también su búsqueda por generador.

Sprint 8

- Periodo: 01/11/22 a 14/11/22
- Objetivos: Búsqueda rápida de documentos
- Resultados destacados: El equipo logra agregar a la página de documentos la opción de filtrado por nombre y fecha, así como la opción de ordenar los documentos de la tabla por columna (nombre, fecha, categoría).

Sprint 9

- Periodo: 15/11/22 a 28/11/22
- Objetivos: Búsqueda rápida de documentos y detalles de documento.
- Resultados destacados: Se logra agregar al sitio *web* el filtrado por título, fecha, categoría y palabras claves de un documento, así permitiendo el filtrado exacto y preciso de documentos claves. Por otra parte, se comienza a implementar la opción de ver detalles de documento.

Sprint 10

- Periodo: 29/11/22 a 12/12/22
- Objetivos: Búsqueda rápida de documentos y detalles de documento.
- Resultados destacados: El equipo no consigue resultados destacados debido a una pausa requerida por entregas y evaluaciones de la Universidad.

Sprint 11

- Periodo: 13/12/22 a 26/12/22
- Objetivos: Búsqueda rápida de documentos, detalles de documento y generación de tareas.
- Resultados destacados: Se realiza la paginación de documentos, así finalizando la página de búsqueda de documentos. Por otra parte, se logró mostrar los detalles de un documento, de esta manera permitiendo al usuario visualizar los datos extraídos de cada documento de manera ordenada y concisa, dejando todo pronto para el segundo *release*. Por último se comienza con la página de generación de tareas para un único documento.

Sprint 12

- Periodo: 27/12/22 a 09/01/23
- Objetivos: Generación de tareas
- Resultados destacados: El equipo logra generar tareas para cada documento, permitiendo la creación, edición y borrado de tareas. De esta forma logrando visualizar las tareas específicas para cada documento.

Sprint 13

- Periodo: 10/01/23 a 23/01/23
- Objetivos: Generación de tareas y gestión de tareas.

- Resultados destacados: El equipo logra finalizar la generación de tareas e implementa la página de gestión de tareas donde se permite al usuario ver todas las tareas de cada generador y también poder ver documentos asociados a cada tarea. De esta forma finalizando el *release* 3 y la entrega final del producto.

10.7 Encuestas internas y externas

Con motivo de gestionar el proyecto, sus avances tanto con el cliente como entre los miembros del equipo se realizaron encuestas.

En lo que respecta al cliente Renovus y siguiente la metodología aplicada basada en Scrum se realizaron encuestas de satisfacción una vez terminado cada *release*. Estas encuestas aportaron un gran valor y fueron claves para el progreso del equipo hacia una mejor atención al cliente y a sus intereses. La encuesta de satisfacción estuvo conformada por dos secciones y siete preguntas en total con opción a realizar comentarios adicionales. Esta encuesta se realizó utilizando la plataforma Google Forms y fue enviada luego de cada demo y *release* realizado. Al haber 3 *releases* en todo el proyecto, esta encuesta se realizó 3 veces al product *owner* del Proyecto.

A continuación se podrán ver capturas de la encuesta realizada y sus resultados a través del proyecto (Figuras 124, 125, 126, 127, 128, 129, 130, 131, 132, 133 y 134).

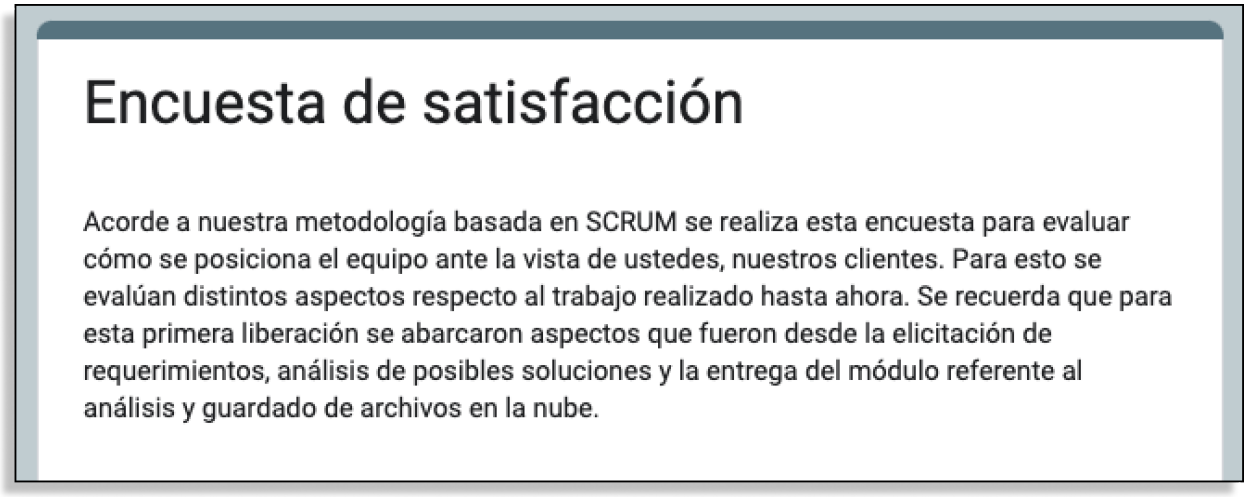


Figura 124: Vista portada de encuesta

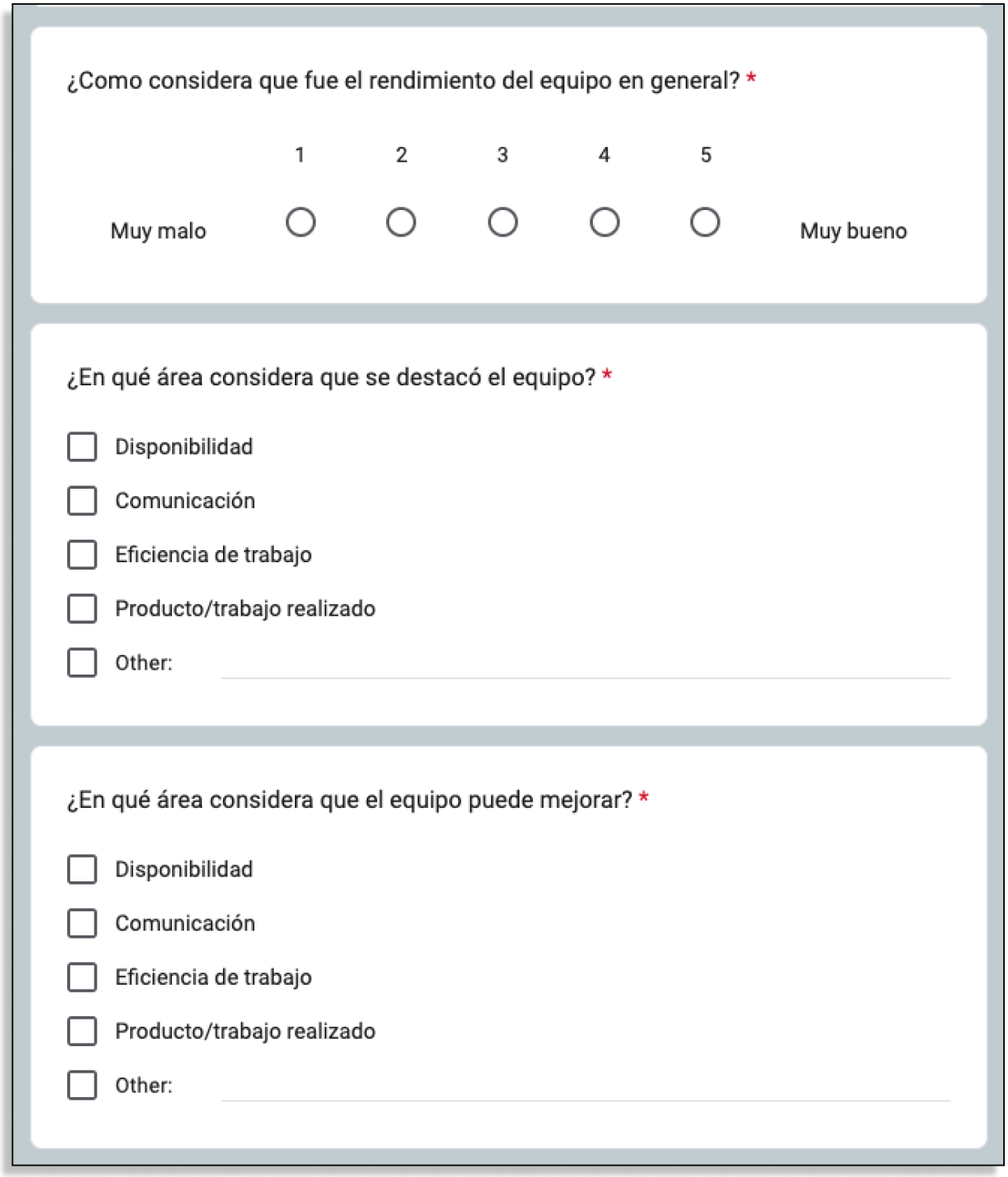


Figura 125: Primera sección de encuesta

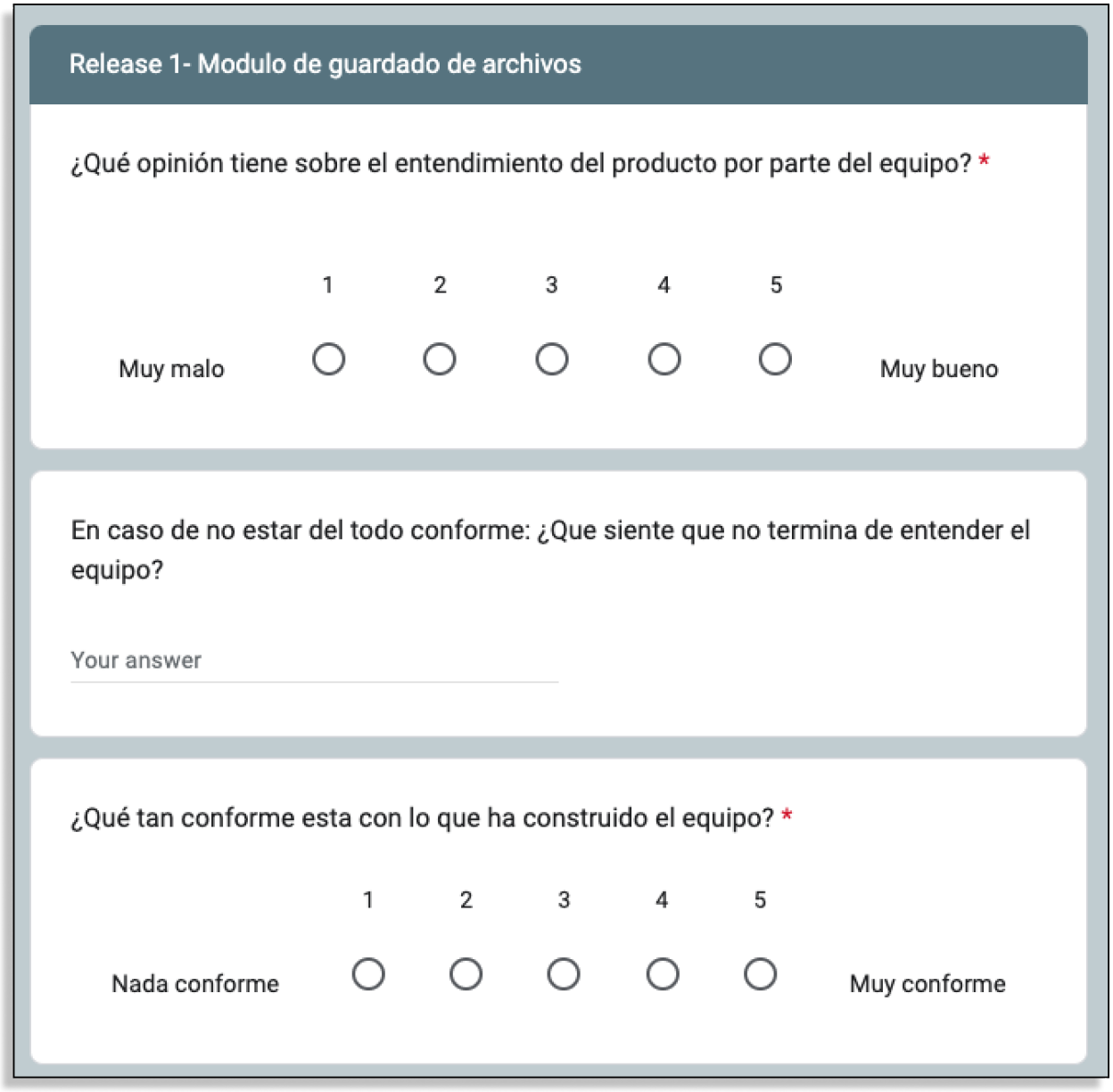


Figura 126: Segunda sección de encuesta primera parte

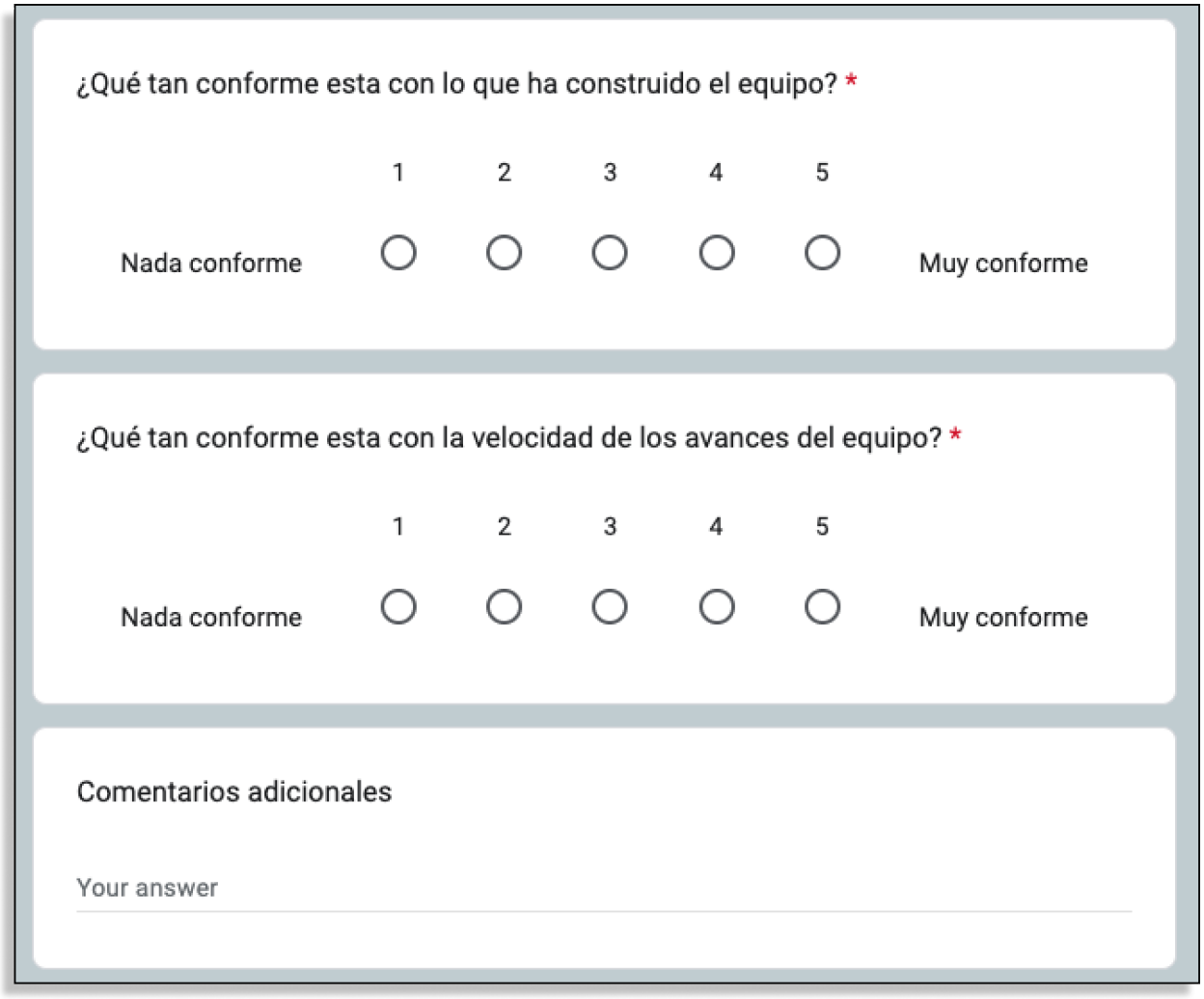


Figura 127: Segunda sección de encuesta segunda parte

Resultados de encuestas:

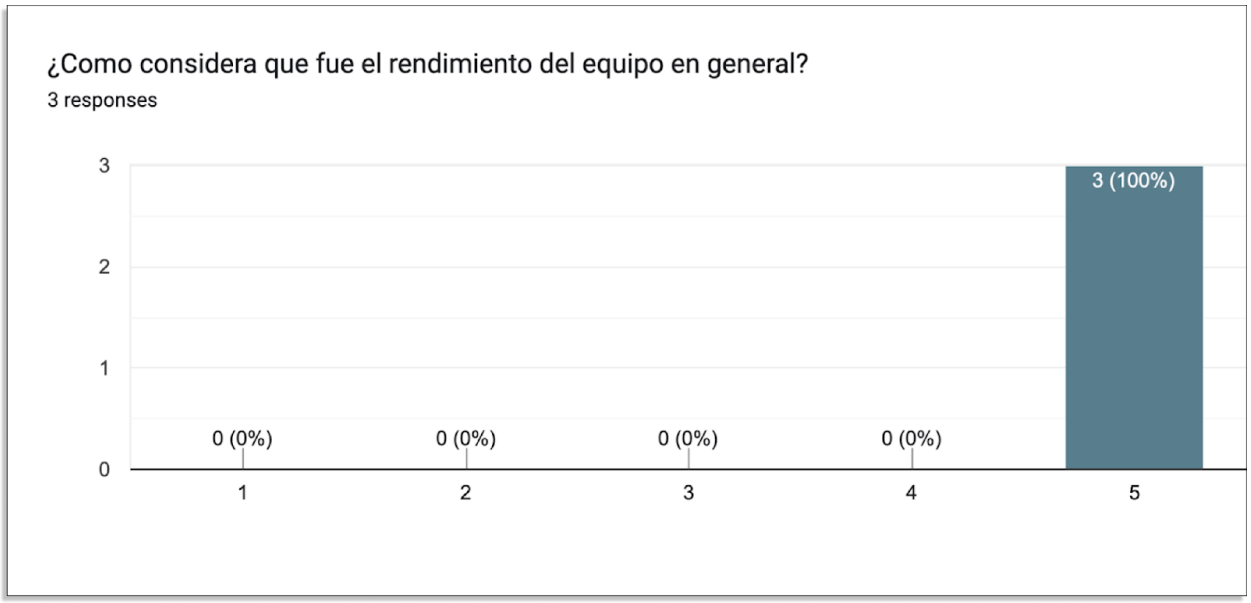


Figura 128: Resultado encuesta cliente primera pregunta

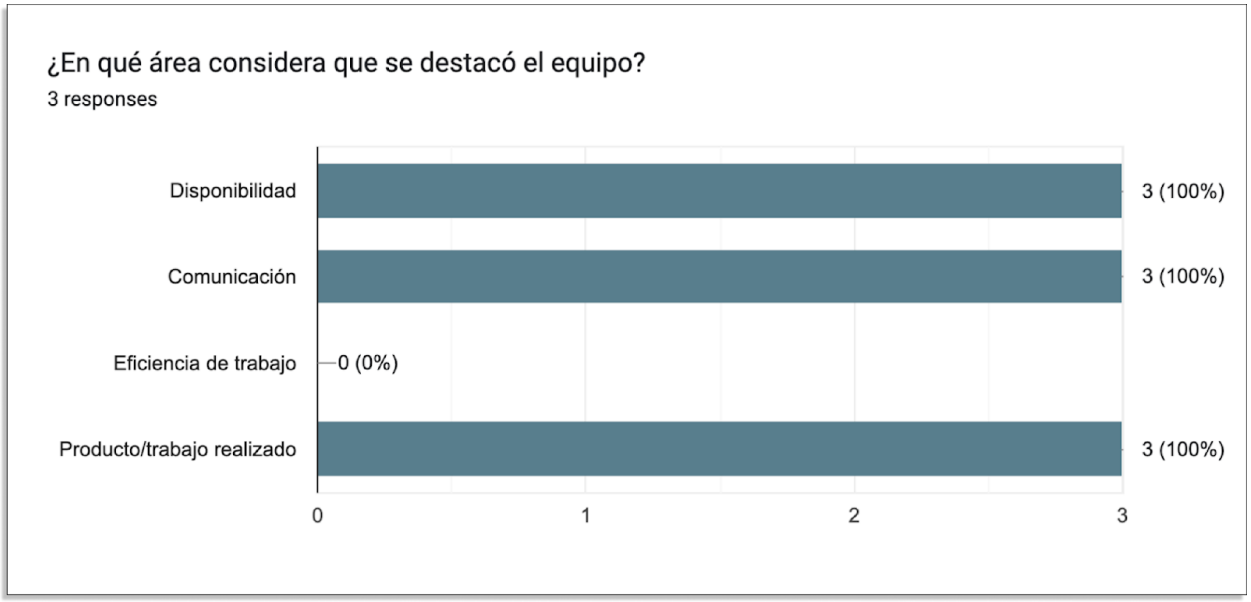


Figura 129: Resultado encuesta cliente segunda pregunta

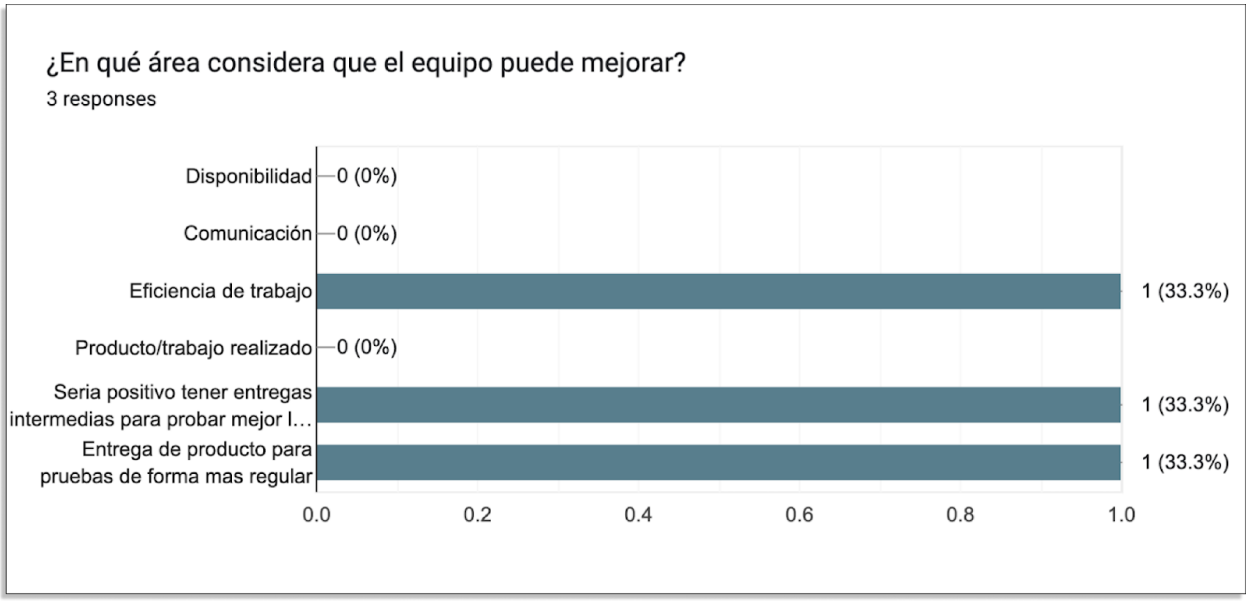


Figura 130: Resultado encuesta cliente tercera pregunta

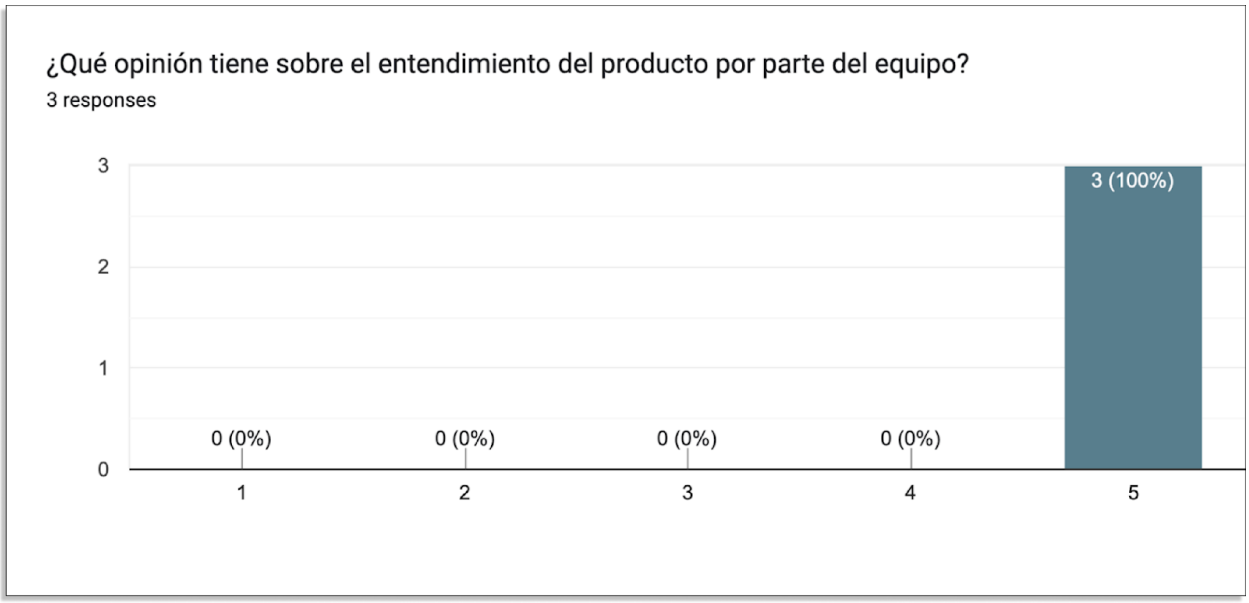


Figura 131: Resultado encuesta cliente cuarta pregunta

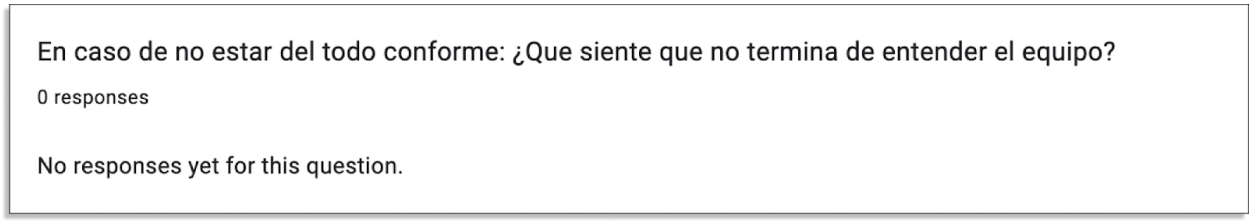


Figura 132: Resultado encuesta cliente quinta pregunta

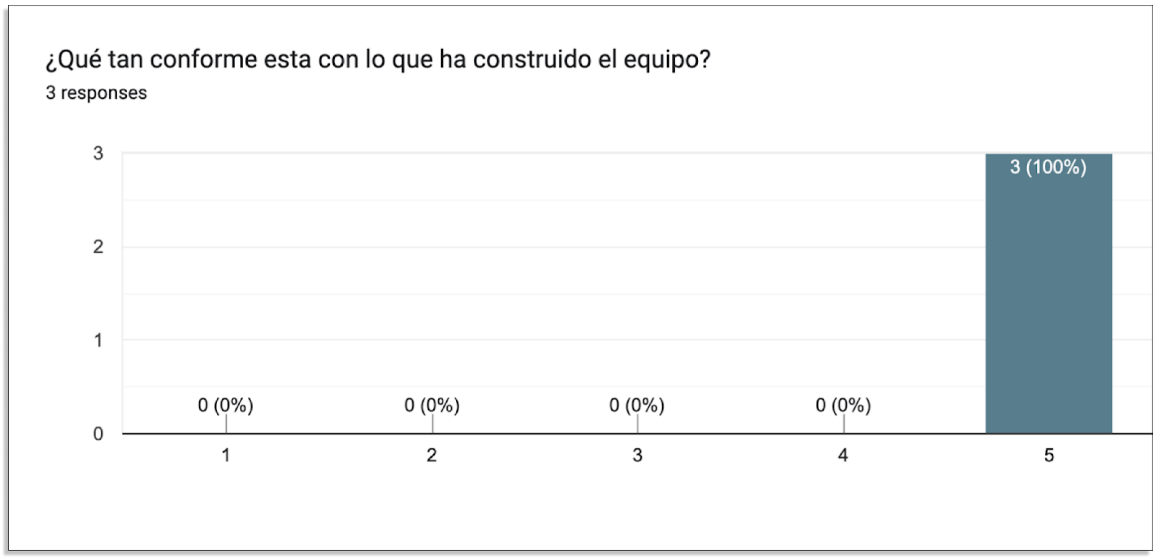


Figura 133: Resultado encuesta cliente sexta pregunta

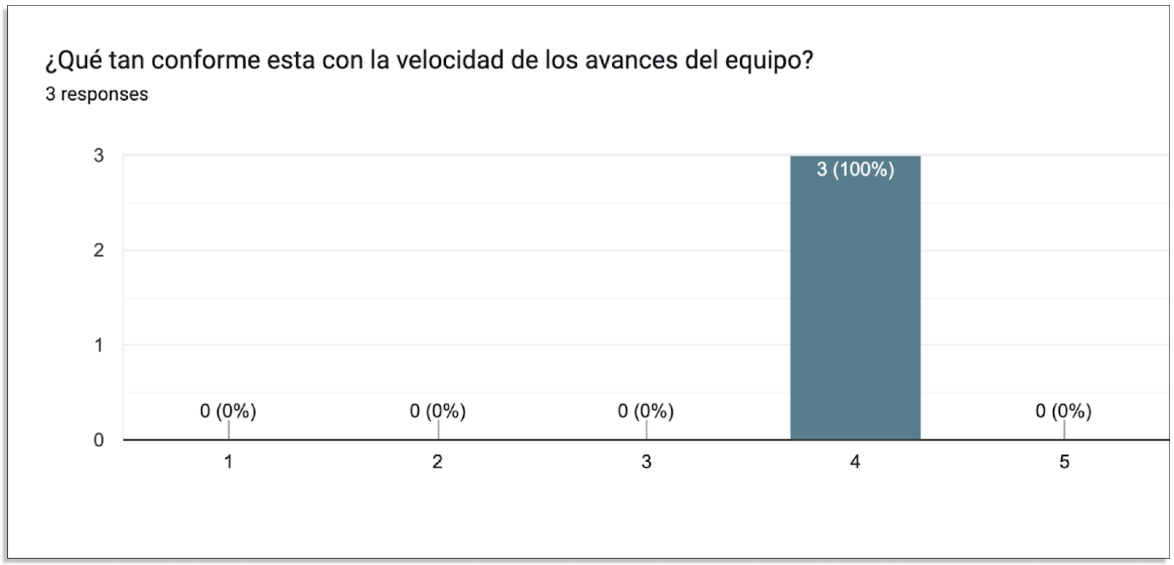


Figura 134: Resultado encuesta cliente séptima pregunta

Comentarios adicionales:

- **Release 1:** Sería bueno poder tener hitos intermedios del producto para ir probando funcionalidades a medida que se dan los avances

- **Release 2:** El equipo está haciendo un trabajo de buena calidad, con avances consistentes y sustanciales.

- **Release 3:** El equipo ha hecho un muy buen trabajo hasta ahora. Ha sido muy claro y profesional en las comunicaciones, trabajando con regularidad y avances consistentes. Además aportan ideas y sugerencias de valor agregado, para seguir mejorando el desarrollo.

En lo que respecta a encuestas internas el equipo realizó dos encuestas:

- ORTsf - Clima de trabajo en el equipo

- Evaluación interna de trabajo del equipo.

El objetivo de las mismas fue el de analizar el clima de trabajo en equipo, a continuación ejemplo de las dos encuestas (Figuras 135 y 136).

ORTsf - Clima de trabajo en el equipo

Responda cada pregunta con un número entre 1 y 5 de acuerdo a la siguiente escala:

1. Totalmente en desacuerdo
2. En desacuerdo
3. Ni en desacuerdo ni de acuerdo
4. De acuerdo
5. Totalmente de acuerdo

Relaciones interpersonales

Se siente cómodo trabajando con los recursos humanos de este proyecto	
Le resulta sencillo contactarse con el resto del equipo / tutor.	
Usted considera que está implicado en el proyecto.	
Usted se considera valorado	
Sus sugerencias son escuchadas	

Sobre su trabajo

Su trabajo está siendo valorado.	
Conoce claramente sus responsabilidades y funciones	
Resulta fácil trabajar con esta metodología.	
La dedicación horaria exigida es apropiada	

Aspectos tecnológicos

Las herramientas de comunicación en esta etapa/fase (e mail; egroups; etc) han funcionado correctamente	

El cliente y usted

El cliente acompaña y define el desarrollo de la etapa en forma satisfactoria.	
El cliente se interesa por la evolución del proyecto	
El cliente manifiesta empatía con el grupo	

Conclusión

Si tuviera la posibilidad, cambiaría de grupo	

Figura 135: Encuesta ORTsf - Clima de trabajo en el equipo

Evaluación interna del trabajo del equipo

1. Nuestra discusión es caótica; utilizamos mucho tiempo en asuntos no esenciales.				1. Todos seguimos la agenda o el propósito de la reunión. Nos movemos lógicamente de un tema a otro.
1	2	3	4	5
2. Las reglas de funcionamiento (puntualidad, asistencia, participación, etc.) no fueron respetadas por la mayoría del grupo.				2. Todos respetamos las reglas de funcionamiento del grupo (puntualidad, asistencia, participación, etc.)
1	2	3	4	5
3. Las reuniones son dominadas por un número pequeño de personas. Algunos integrantes del grupo no participan.				3. Todos toman parte al tomar decisiones. Todos pueden expresar sus opiniones.
1	2	3	4	5
4. Cuando alguien habla es constantemente interrumpido o no se le permite finalizar.				4. Todas las opiniones son escuchadas por igual. Las personas prestan atención a los problemas del momento.
1	2	3	4	5
5. Los miembros del grupo no hacen nada para mantener la reunión focalizada en los temas del día.				5. Todos ayudan a mantener la discusión focalizada en los temas del día.
1	2	3	4	5
6. Se podría obtener más si trabajáramos por separado. Juntos logramos resultados insignificantes o cero.				6. Hay mucha sinergia presente en nuestro trabajo. Logramos muchos resultados trabajando juntos.
1	2	3	4	5
7. Nadie (excepto quizá uno) realmente asume la responsabilidad de realizar el trabajo asignado.				7. Cada uno de los miembros del grupo asume su responsabilidad personal de realizar el trabajo asignado.
1	2	3	4	5
8. Me siento frustrado. Fue una pérdida de tiempo y de energía.				8. Me siento realmente bien sobre el trabajo realizado.
1	2	3	4	5

Figura 136: Evaluación interna de trabajo del equipo

Encuesta Team Pulse

La encuesta team pulse se genera en base a una combinación entre la encuesta de clima de trabajo en el equipo y la evaluación interna de trabajo en equipo. A continuación preguntas y resultados (Figura 137, 138, 139, 140, 141, 142, 143 y 144 y Tabla 34):

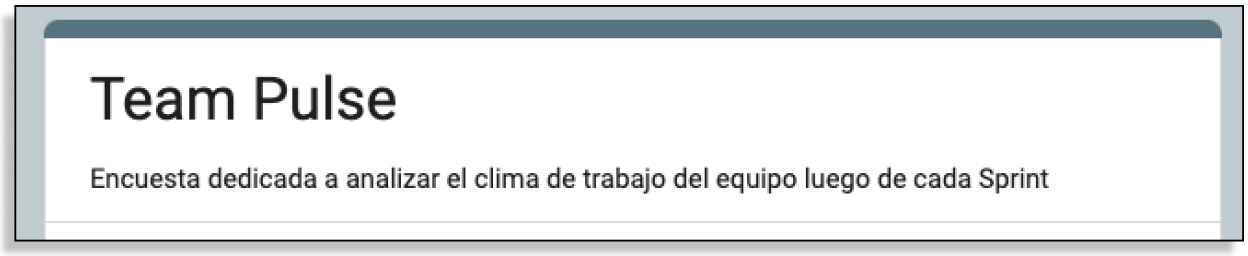


Figura 137: Encuesta team pulse descripción

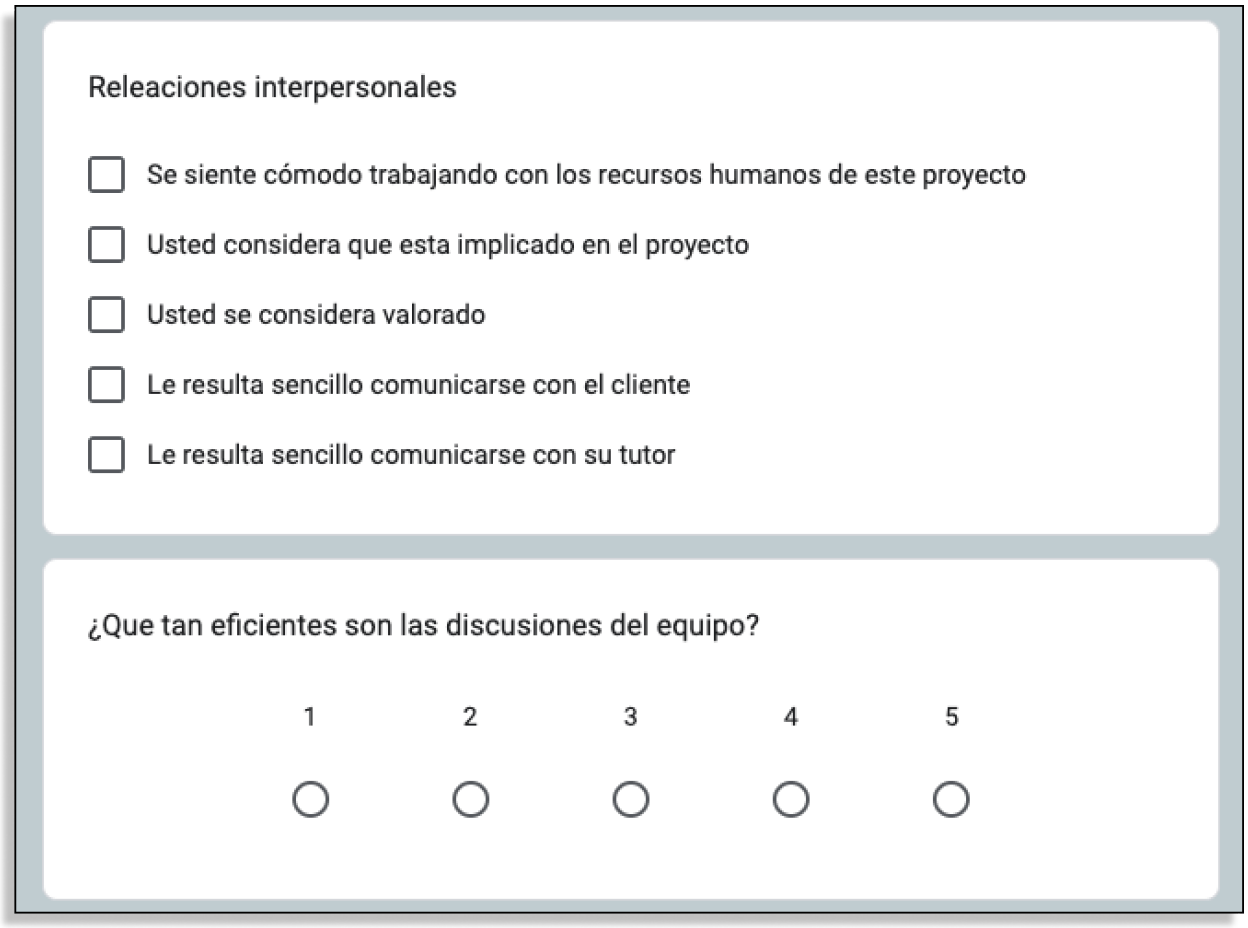


Figura 138: Encuesta team pulse parte uno

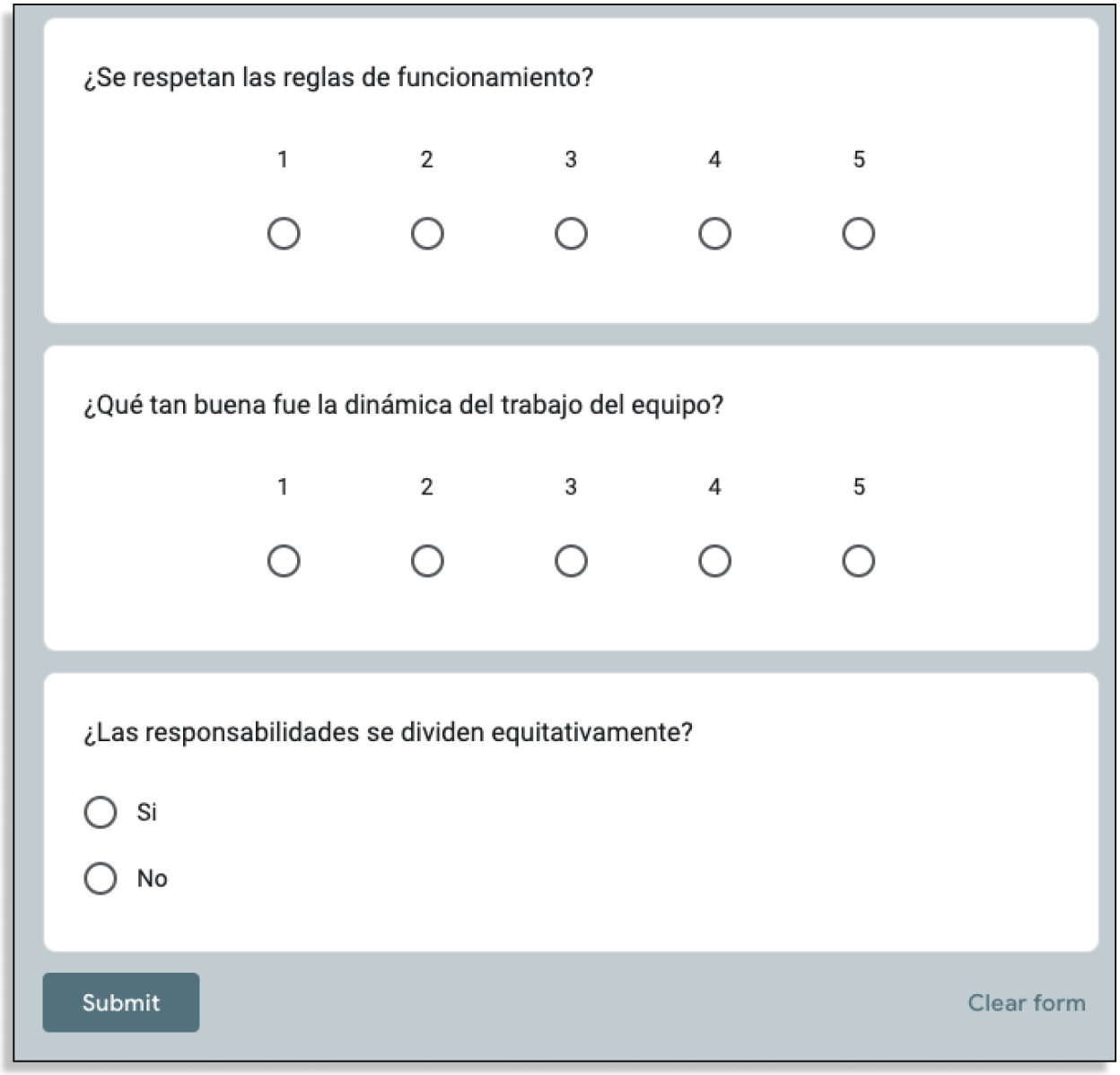


Figura 139: Encuesta team pulse parte dos

A continuación se podrán ver las gráficas resultantes de todas las encuestas realizadas (39 en total) a lo largo del proyecto.

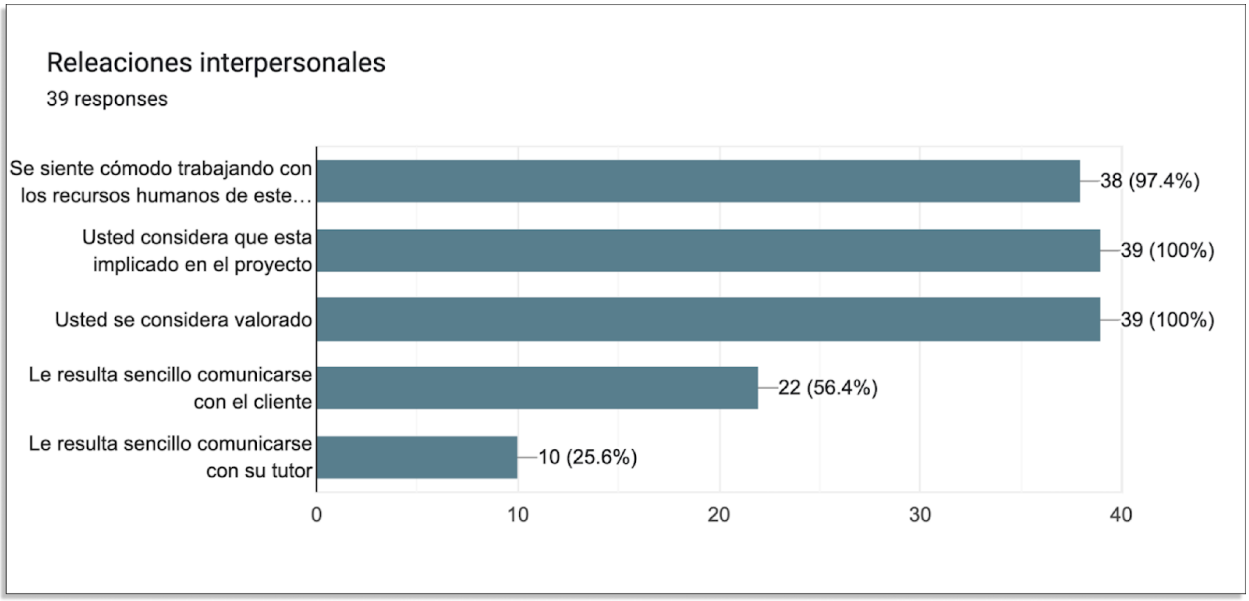


Figura 140: Resultado de encuestas, primer pregunta

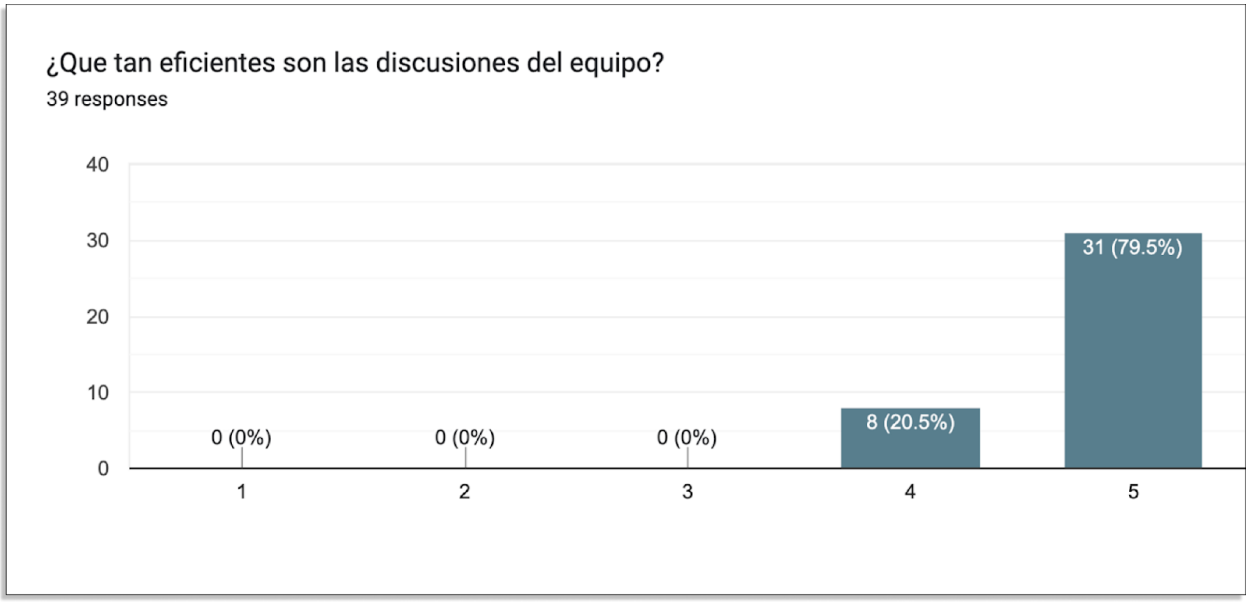


Figura 141: Resultado de encuestas, segunda pregunta

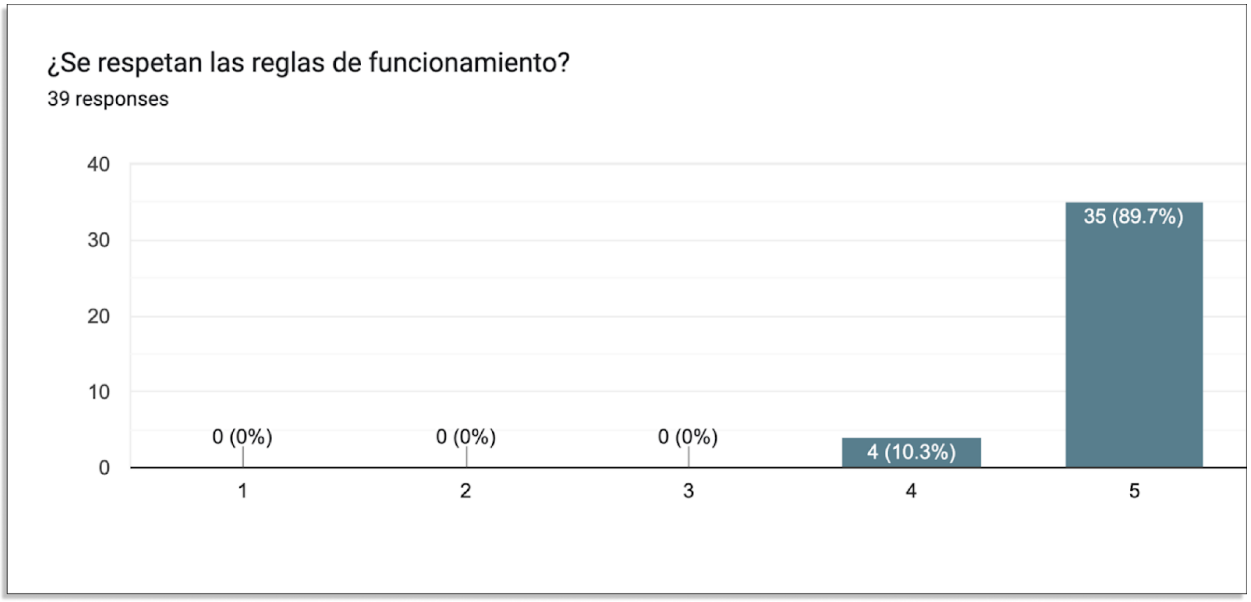


Figura 142: Resultado de encuestas, tercera pregunta

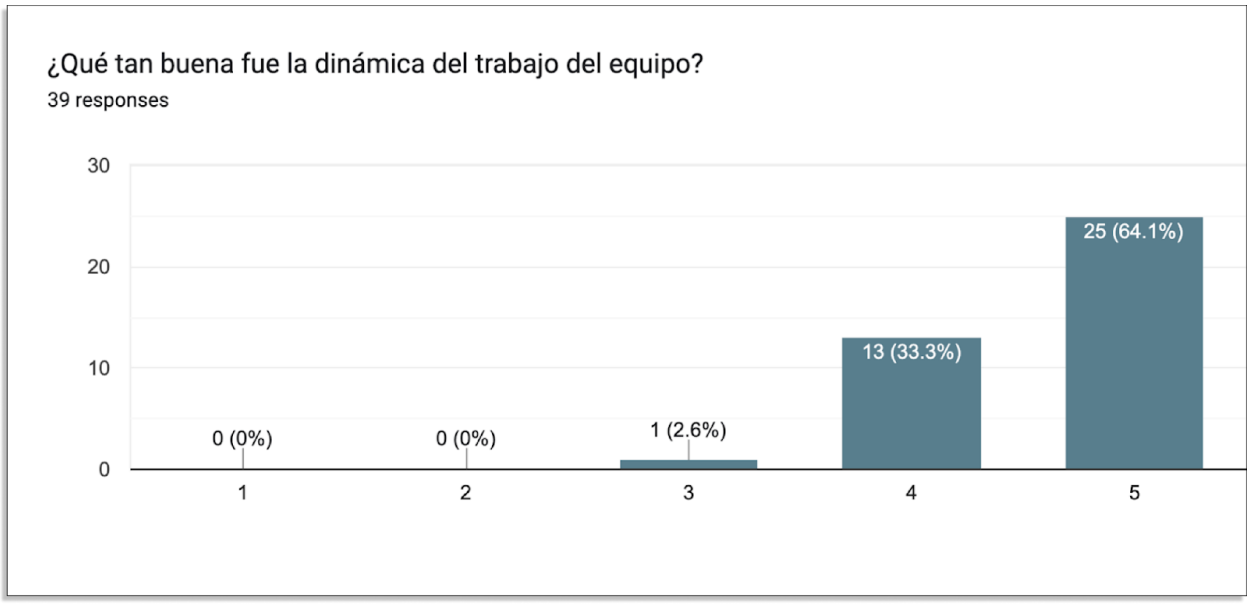


Figura 143: Resultado de encuestas, cuarta pregunta

Promedio de calificación de dinámica de trabajo del equipo	**4.61**

Tabla 34: Promedio de calificación de dinámicas de trabajo en equipo alcanzado.

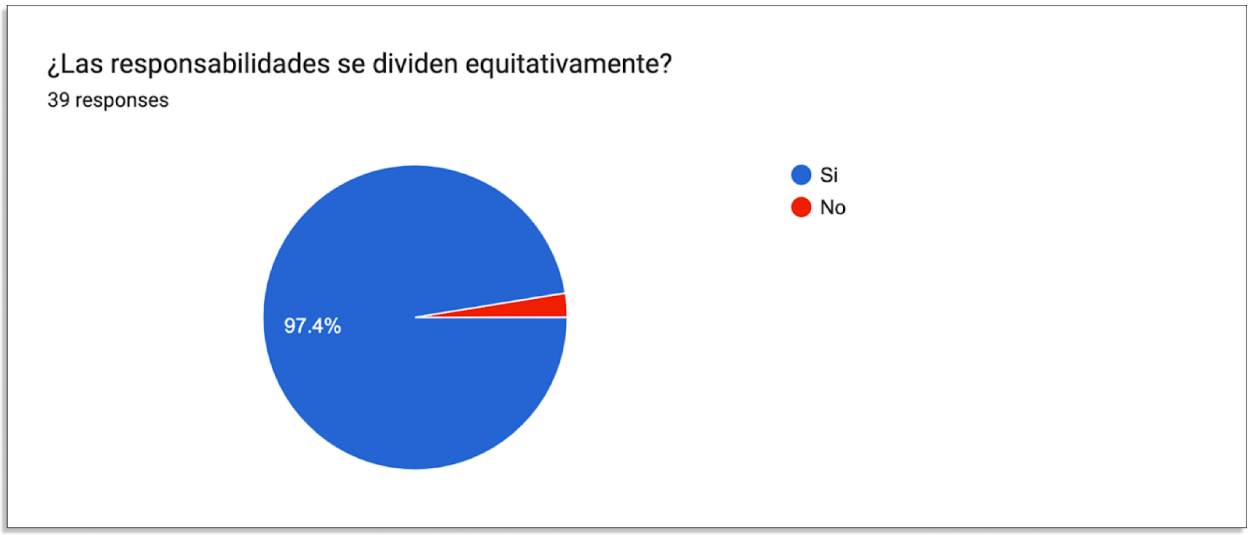


Figura 144: Resultado de encuestas, cuarta pregunta

10.8 Recopilación de riesgos

Recopilación de riesgos encontrados (Tabla 35):

ID	Descripción	Categoría	Estrategia Principal	Plan de mitigación	Plan de contingencia
R1	Estimaciones muy ambiciosas que derivan en falta de completitud en el desarrollo del producto o incumplimiento con fechas estipuladas	Gestión de proyecto	N-Prevenir	Realizar revisiones periódicas de las estimaciones y buscar re-estimar aquellas que no continúen siendo factibles. Establecer márgenes de tiempos adicionales en los cronogramas de proyecto para acomodar cambios. Realizar seguimiento periódico de los progresos y hacer ajustes en función de los mismos.	Establecer una línea de comunicación con dirección de Renovus para informarle de los cambios en las estimaciones o el cronograma. Considerar la posibilidad de aumentar el plazo de una historia o solicitar apoyo de expertos de Renovus.
R2	Desconocimiento de tecnologías que desembocan en errores de desarrollo o incumplimiento de fechas establecidas.	Capacitación/Skills	N-Prevenir	Realización de cursos pertinentes y pruebas de concepto. Tener reuniones con expertos de Renovus. Realizar consultas en foros de expertos.	Utilizar días de licencia o estudio con el fin de obtener más tiempo del previsto para la capacitación y corregir los errores ocurridos.
R3	Enfermedad de alguno de los miembros del equipo que provoca retrasos en las tareas establecidas	Humano	N - Aceptar	Cada integrante ha de conocer el progreso del otro, particularmente a la hora de hacer revisiones de código. Esto implica que	Distribuir las tareas que se tendrían que hacer por el integrante enfermo. Ajustar tiempo y alcance en consecuencia. De ser

	para el/los Sprint/s en que el miembro esté de baja o no pueda rendir en su plenitud.			todos los integrantes conocen las áreas donde se ha de recuperar trabajo en futuras iteraciones. Se mantienen reuniones en modalidad virtual a fin de evitar contagios de todo el equipo.	necesario redoblar esfuerzo por parte del integrante cuando se recupere.
R4	Conflictos interpersonales entre los compañeros del equipo que llevan a bajo rendimiento del equipo afectando el cumplimiento de el/los Sprint/s durante los cuales persisten el/los conflicto/s	Humano	N-Evitar	Los miembros se comprometen a interactuar de forma educada y ordenada, respetando las opiniones del resto y favoreciendo la convivencia.	De materializarse el riesgo se toma acción mediante la utilización y aplicación del Método de Negociación de Harvard [36] para el que el equipo realizó una materia electiva en la Universidad ORT a fin de poder renegociar los términos de trabajo entre las partes afectadas.
R5	Requerimientos existentes cambiantes por parte del cliente (no tienen claro lo que quieren) y conlleva a especificaciones de requerimientos erróneas y pérdida de tiempo.	Gestión de proyecto	N-Prevenir	Se realizan validaciones propias del proceso. Se validan con PO las historias, se realizan prototipos a confirmar cambios.	De haberse implementado cierto aspecto se toma en cuenta la aprobación previa del cliente como justificación al ajuste de tiempo a realizar. Se ajusta el cronograma.

R6	Requerimientos existentes cambiantes por parte del cliente (cambios en la necesidad del negocio).	Cliente	N-Mitigar	Se busca llevar la discusión de los requerimientos a largo plazo con el cliente a fin de mitigar cualquier posible cambio predecible en el negocio.	Se vuelve a realizar el análisis del requerimiento afectado y se planifica el cambio.
R7	Problemas de comunicación con el cliente.	Cliente	N-Prevenir	Se buscan formas de comunicación con el cliente que se adecuen a sus horarios y tiempos. Se reafirma a modo de recordatorio que tienen un compromiso formal con el equipo en caso de no .	Se realiza un ajuste de los requerimientos acorde al entendimiento del negocio por el equipo. Se insiste al cliente hasta lograr su aparición.
R8	Demoras del cliente para brindar recursos/ambientes que demoren las pruebas en entornos reales	Cliente	N-Mitigar	Solicitar reiteradas veces los ambientes al responsable de tecnología. Mencionar el tema en las reuniones.	Levantar el ambiente en un entorno del equipo estudiante. Asumir los costos del mismo.
R9	Perdida de informacion causa de robo o falla informática.	Tecnología	N-Evitar	A medida que se avance en el desarrollo se ha de garantizar la persistencia en la nube de cada avance.	Se reemplaza la información perdida lo antes posible. Se repone el recurso dañado lo antes posible. En caso de no contar con el capital solicitar al cliente o a la Universidad.
R10	El proceso de integración de *software* a solución actual no se da de	Tecnología	N-Prevenir	Definir bien la gestión de cambios y coordinación con el equipo cliente. Mantener una	Detectar fallas junto a la integración de cambios y actuar sobre los mismos.

	forma exitosa y resulta en interferencias accidentales con las operaciones de Renovus, retrasando el proyecto.			comunicación abierta con ellos sobre acciones a tomar.	
R11	No se logra llegar a solución con funcionalidades pactadas incumpliendo acuerdos formales sobre alcance	Gestión de proyecto	N-Prevenir	Monitorear durante el ciclo de vida, comparar estado actual contra planificado y ver de tomar acciones correctivas acorde a los retrasos.	Ajustar tiempos para corregir demoras, duplicar esfuerzo. Utilizar días de licencia por parte de los integrantes del equipo a fin de llegar a tiempo.
R12	Requerimientos nuevos no planificados que afectan el cronograma previsto	Cliente	N-Prevenir	Hacer que instancias de refinamiento sean lo más efectivas posibles a fin de tener bien definido el alcance del proyecto.	El cambio es analizado y de lograrse en tiempo esperable se acepta. De otra forma se rechaza la solicitud respaldados por alcance definido previamente.
R13	Atraso en desarrollo que signifique un incumplimiento en tiempos acordados con cliente	Gestión de proyecto	N-Prevenir	Duplicar esfuerzos al ver que no se llega a lo pactado para *Release*. Sacrificar horas de sueño.	Tomar acciones necesarias, incluido la toma de licencias y días de estudios para lograr llegar a lo pactado.
R14	Definición poco clara de requerimientos que conlleven a entregables erróneos	QA	N-Evitar	Realizar instancias de refinamiento y prestar atención al proceso de validación de historias de usuario.	Analizar el impacto de la falta de definición y volver evaluar todas las historias presentes a fin de confirmar si no hay otra. Re-estimar el alcance.

R15	Falta de comunicación entre miembros del equipo que provoquen errores en el proyecto	Humano	N-Evitar	La realización de reuniones semanales en formato de llamada de voz buscan coordinar al equipo y evitar errores a causa de la mala comunicación.	Cada miembro debe buscar entender el problema existente y coordinar de forma clara cómo se va a resolver.
R16	Errores de *software* que eviten o dificulten el uso de alguna funcionalidad del sistema.	Producto	N-Mitigar	Asegurarse de comprender correctamente las funcionalidades y realizar pruebas establecidas en parámetros de QA.	De aparecer errores se ha de trabajar en ellos acorde a su criticidad.
R17	Problemas con el repositorio Git que lleven a sobreescritura y pérdida de información	Desarrollo	N-Mitigar	Se tiene cuidado con el uso de ramas, manejo de GitFlow.	La herramienta Git permite la restauración de versiones en el peor caso.
R18	Ataque ransomware que derive en el secuestro información relevante al proyecto	Tecnologia	N-Evitar	A medida que se avance en el desarrollo se ha de garantizar la persistencia en la nube de cada avance. Adicionalmente se evita descargar *software* de fuentes cuestionables y evitar clicar en enlaces maliciosos.	No se paga el rescate y se reinstala el sistema operativo.
R19	Incumplimento con estandares de codificacion que disminuyan la calidad del proyecto	QA	N-Evitar	Utilizar herramientas de chequeo estático en el IDE, externas como SonarQube.	Realizar refactor en caso de detectar algún problema de calidad vinculado. Se vuelven a aplicar pruebas y chequeos de estándares.

R20	Baja de un integrante	Humano	N-Evitar	Resolver conflictos internos a medida que aparezcan a fin de que no se agraven resultando en una salida.	Reestimar el alcance propuesto a la Universidad y a Renovus.
R21	Quiebre de empresa Renovus.	Cliente	N-Aceptar	Se escapa del control del equipo la mitigación de este riesgo.	Liberar el código como *OpenSource* a fin de que pueda ser aprovechado en la industria de las energías renovables.

Tabla 35: Recopilación de riesgos

10.9 Evaluaciones de riesgos

Nota: las probabilidades están expresadas en notación decimal a diferencia de los porcentajes detallados en la matriz.

Evaluación Abril/Mayo (Tabla 36)

Riesgo	Impacto	Probabilidad	Prioridad (Impacto x Probabilidad)
R1	1	0	0
R2	1	0	0
R13	1	0	0
R5	1	0.8	0.8
R7	4	0.6	2.4

Tabla 36: Recopilación de riesgos Abril/Mayo

Evaluación Junio/Julio (Tabla 37)

Riesgo	Impacto	Probabilidad	Prioridad (Impacto x Probabilidad)
R1	2	0	0
R2	3	0.6	1.8
R13	1	0	0
R5	2	0.7	1.4
R7	4	0.5	2

Tabla 37: Recopilación de riesgos Abril/Mayo

Evaluación Agosto/Setiembre (Tabla 38)

Riesgo	Impacto	Probabilidad	Prioridad (Impacto x Probabilidad)
R1	3	0.7	2.1

R2	3	0.5	1.5
R13	2	0.5	1
R5	3	0.6	1.8
R7	2	0.3	0.6

Tabla 38: Recopilación de riesgos Agosto/Setiembre

Evaluación Octubre/Noviembre (Tabla 39)

Riesgo	Impacto	Probabilidad	Prioridad (Impacto x Probabilidad)
R1	3	0.5	1.5
R2	3	0.4	1.2
R13	2	0.5	1
R5	4	0.6	2.4
R7	2	0.3	0.6

Tabla 39: Recopilación de riesgos Octubre/Noviembre

Evaluación Diciembre/Enero (Tabla 40)

Riesgo	Impacto	Probabilidad	Prioridad (Impacto x Probabilidad)
R1	4	0.4	1.6
R2	4	0.2	0.8
R13	4	0.4	1.6
R5	5	0.4	2
R7	3	0.2	0.6

Tabla 40: Recopilación de riesgos Diciembre/Enero

Evaluación Febrero/Marzo (Tabla 41)

Riesgo	Impacto	Probabilidad	Prioridad (Impacto x Probabilidad)
R1	4	0.4	1.6
R2	5	0.1	0.5
R13	5	0.4	2
R5	5	0.2	1
R7	2	0.1	0.2

Tabla 41: Recopilación de riesgos Febrero/Marzo

10.10 Evolución de Arquitectura

Tal como se mencionó en diferentes partes del documento, la arquitectura fue uno de los módulos que experimentó mayores transformaciones. En esta sección, se presentará la evolución de la arquitectura y se comentarán los aspectos relevantes de cada versión.

Es importante tener en cuenta que algunos de los diagramas pueden contener algunas simplificaciones notacionales o abusos de notación, ya que fueron creados como bocetos para que el equipo pudiera entenderlos, y no como diagramas formales.

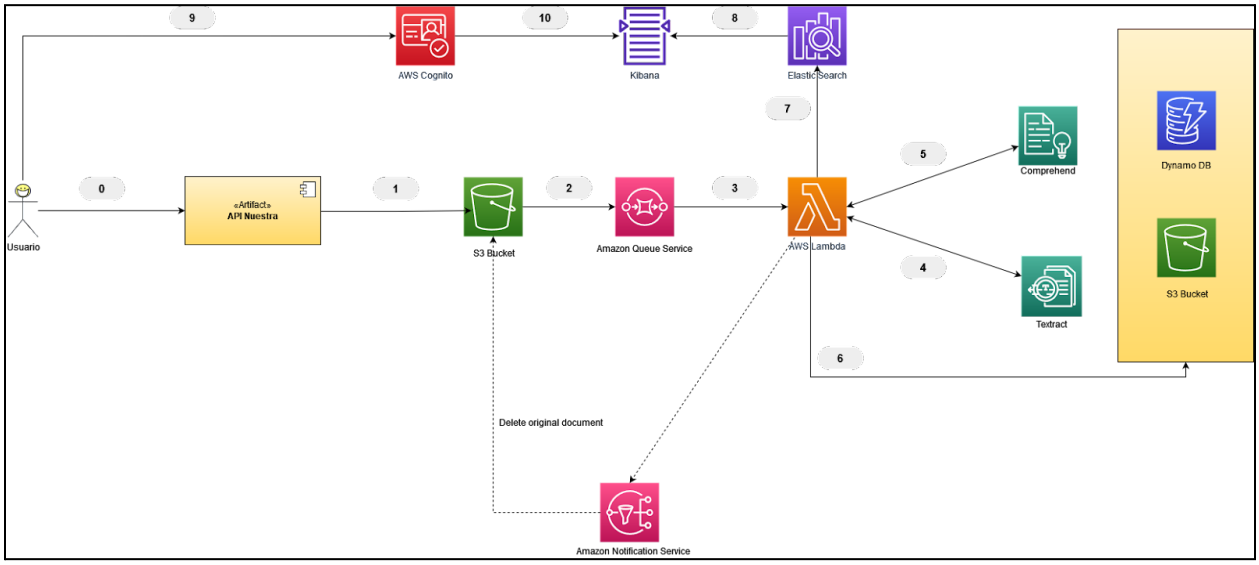


Figura 145: Arquitectura inicial

Inicialmente, cómo se puede apreciar en la Figura 145, la arquitectura se seguía trabajando bajo la premisa de que el sistema iba a ser una solución externa a la actual, por esto es que en la parte superior se pueden observar los pasos 9 y 10, donde se accedería a través de AWS Cognito a Kibana para el filtrado y búsqueda de los documentos además de la visualización de estos y otros datos resultantes del análisis.

Los puntos a destacar de esta arquitectura, además de lo ya mencionado, es que, se utilizaba una sola instancia de Lambda para todo desde la comunicación con Amazon Textract y Comprehend, el envío de datos a una base llamada ElasticSearch y el envío de una notificación para el borrado

de los documentos del primer S3 para luego hacer un respaldo de información en DynamoDB y un segundo Bucket.

A modo de evitar que un solo Lambda tuviese todas las responsabilidades el equipo plantea una mejora a la arquitectura.

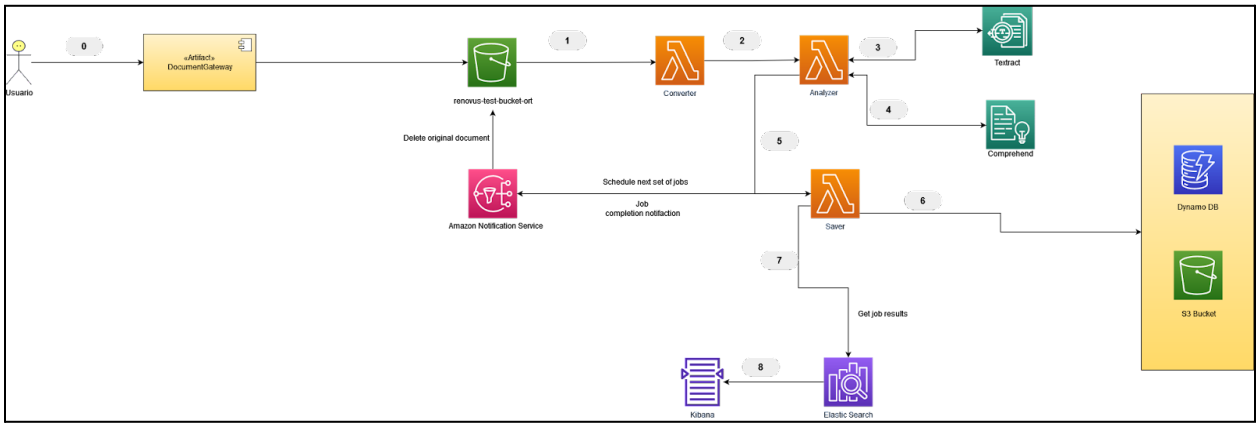


Figura 146: Segunda versión de arquitectura

En esta segunda versión Figura 146 se observan algunos cambios que comenzaron a surgir debido a la evolución del alcance del proyecto. Como se puede apreciar, se pasó de una sola instancia de Lambda a tres, con los conceptos de Analyzer y Saver, y se añadió el módulo llamado Converter, que se encargaba de la conversión de archivos Excel a formato PDF, dado que Amazon Textract no soporta archivos Excel en la actualidad. Estos cambios se hicieron necesarios para manejar mejor las tareas y para ajustar la arquitectura a las necesidades del proyecto en evolución.

El módulo del Analyzer mantendría las responsabilidades del análisis como bien indica su nombre, y luego el Saver sería el encargado de enviar la notificación para el borrado de los documentos en el primer S3, almacenar la información en el respaldo que se planificaba en DynamoDB y otro *Bucket* de S3, y finalmente el envío a la base de datos, que hasta ese momento era ElasticSearch.

Otro cambio presente en este diagrama, es que desaparece la autenticación con el módulo de AWS Cognito, ya que se empezaba a conversar que la solución sería integrada a la actual, por lo

que el control de acceso, manejo de usuarios y demás, se desprendería de las responsabilidades del equipo.

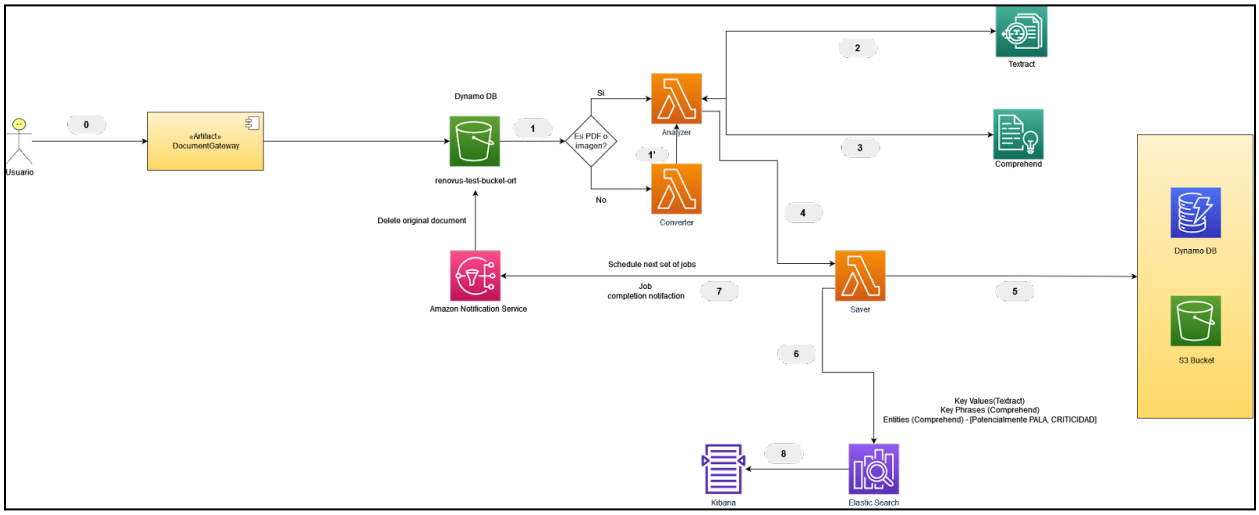


Figura 147: Tercera versión de la arquitectura del sistema

En este diagrama (Figura 147), no se aprecian grandes diferencias con respecto al anterior. La evolución en esta etapa fue leve, pero crucial, especialmente en lo que se refiere a la organización y definición de responsabilidades. El cambio más significativo consistió en la inclusión de un cuadro de decisión entre el S3, Converter y Analyzer, que establecía que solo se pasaría por el módulo Converter si el documento no era un PDF. Este ajuste fue importante para optimizar el flujo de tareas y asegurar un mejor rendimiento del sistema.

Anteriormente, en la segunda versión de arquitectura, se procesaban todos los documentos a través de una instancia específica. Aunque el consumo de una instancia de Lambda no era excesivo, el equipo decidió optimizar los costos y reducir los pasos de ejecución e instancias de procesamiento innecesarias. Por lo tanto, se implementó un cambio en el proceso para evitar pasar todos los documentos por la misma instancia. Las demás responsabilidades y aspectos de la arquitectura se mantienen de acuerdo con la versión 2.

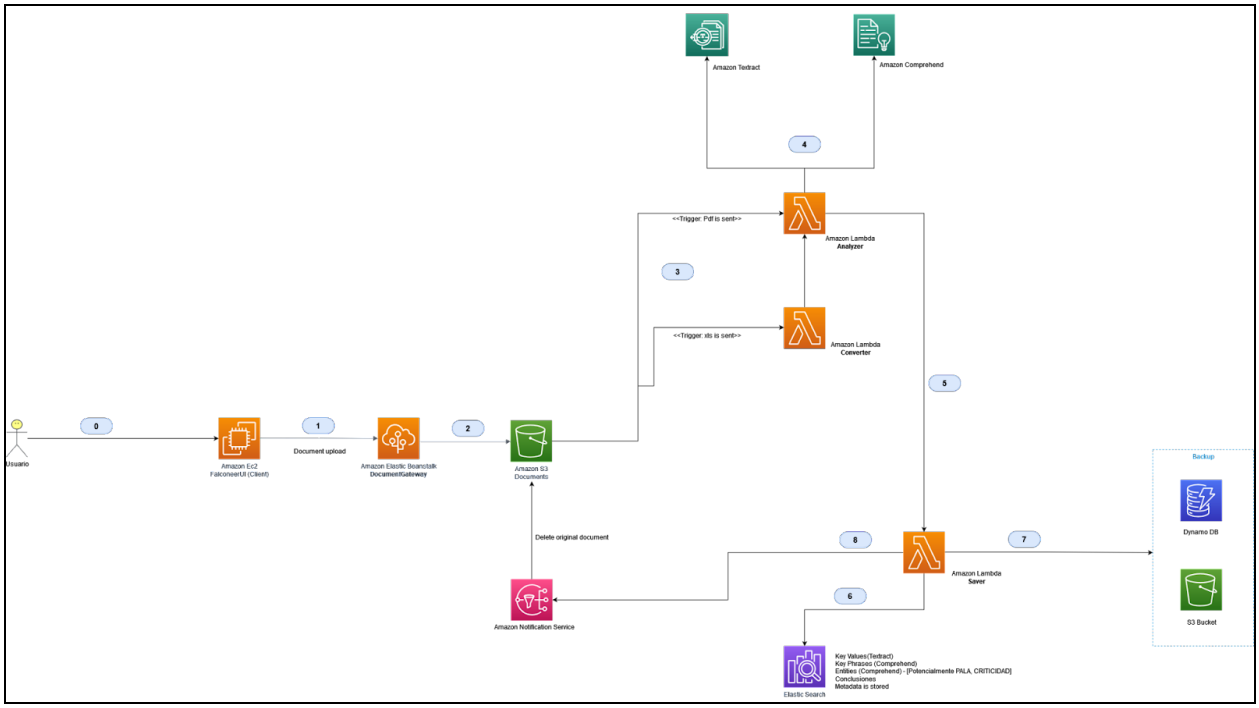


Figura 148: Cuarta versión de la arquitectura del sistema

Esta versión (Figura 148) fue la penúltima realizada por el equipo, mantiene un estructura similar a los dos diagramas anteriores desde el primer *Bucket* en adelante. Aquí se introducen cambios previos a que el documento sea almacenado en el primer S3, y es que se agrega la instancia de Amazon EC2 Falconeer UI, donde se integrará el *frontend* desarrollado en este proyecto junto al del cliente, dado que desde este punto se trabajó la solución como parte de la solución actual, y también se incluye Amazon Elastic Beanstalk donde se ubicaba la API desarrollada.

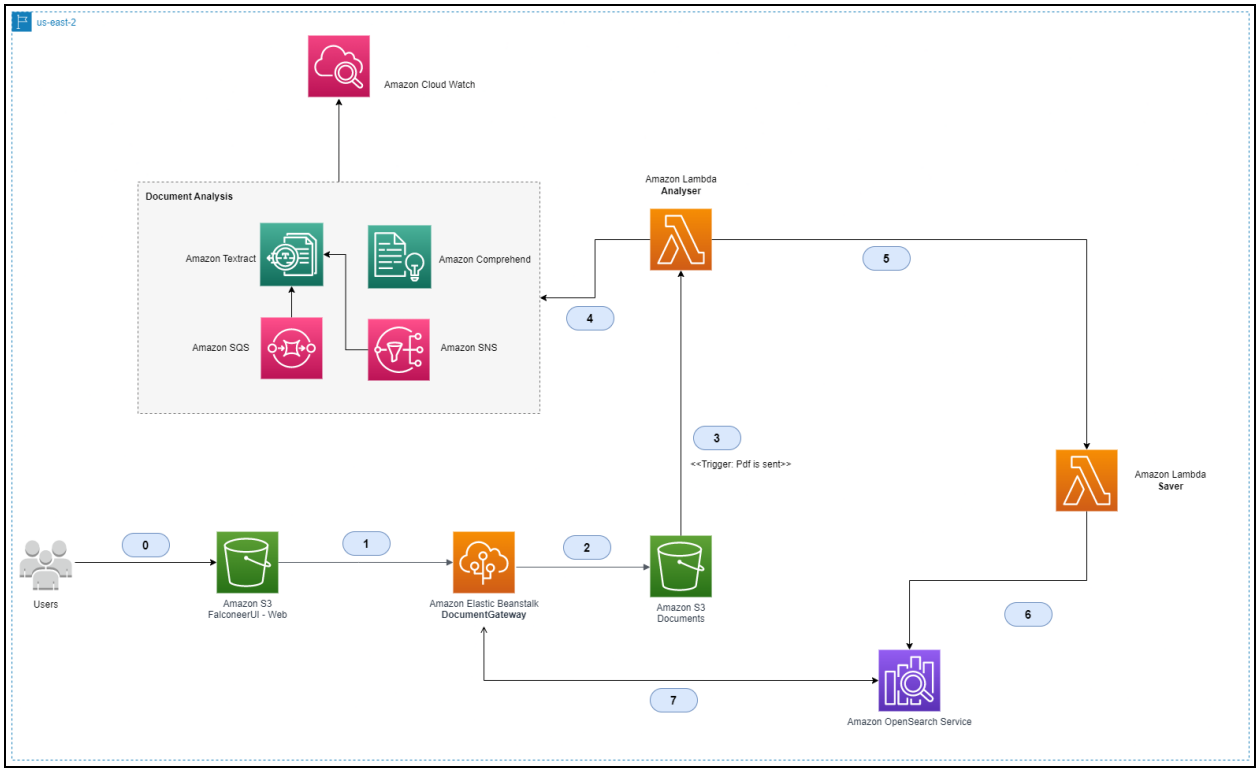


Figura 149: Arquitectura final

Finalmente, llegamos al último diagrama (Figura 149), y a lo que fue la versión final de la arquitectura del sistema. Como se puede observar existen varios cambios, algunos con justificación en el alcance, otros en aspectos de optimización y costos.

Por un lado, se remueve la instancia de Lambda denominada Converter, dado que para esta versión se manejarían solo documentos en formato PDF y deberían ser los usuarios quienes transforman los Excel a dicho formato para utilizar el sistema. Luego, en el módulo de análisis de documentos, donde se encuentran Amazon Textract y Comprehend, se suman las colas de mensajes y de notificaciones utilizadas durante el proceso para buscar los resultados obtenidos una vez finalizado el análisis de los documentos; así como también Amazon CloudWatch, módulo utilizado para los logs y registros del procesamiento de los documentos.

Posteriormente, se llevaron a cabo cambios con el objetivo de optimizar los costos, entre los cuales se encontraba la eliminación del módulo de DynamoDB y del segundo S3. El equipo consideró que mantener un respaldo adicional en dichos servicios era costoso, especialmente

porque los datos ya estaban almacenados en una base de datos con capacidad de replicación de datos. Después de analizar en detalle la solución, se llegó a la conclusión de que no era necesario trasladar los documentos de un S3 a otro, ya que no aportaba ningún beneficio a la solución. En consecuencia, se decidió remover dichos componentes.

A su vez, se modifica la base de datos a utilizar, dejando ElasticSearch por OpenSearch principalmente por consideraciones a nivel de costos, ya que OpenSearch brindaba los mismos beneficios que ElasticSearch, pero que al ser un módulo propio de AWS y no de un tercero como, el costo por utilizarlo era mucho menor, lo cual era conveniente para el cliente con el objetivo de mantener los mismos lo más bajo posible.

10.11 Notas de revisiones

Primera revisión - 1° de agosto de 2022

Revisor: Ing. Rafael Bentancour

Informe

El día 1° de agosto , se llevó a cabo una revisión con el Ing. Rafael Bentancour, con el fin de evaluar el progreso del proyecto en cuestión. A continuación, se detallarán los puntos más relevantes tratados durante la reunión.

En primer lugar, se planteó la importancia de establecer objetivos claros y medibles desde un punto de vista académico. Para ello, se sugirió utilizar el enfoque SMART, que contempla la creación de objetivos simples, medibles y alcanzables. Además, se recomendó detallar los objetivos en documentos verificables y reconocibles para asegurar la trazabilidad del proceso.

Asimismo, se señaló la necesidad de corregir el manejo de los tiempos en el proyecto, con el fin de lograr una mejor organización en el ciclo de vida del mismo. En este sentido, se destacó la importancia de generar soluciones de forma incremental y considerar si estas serán utilizadas por los usuarios. Respecto a las adaptaciones, se mencionó la necesidad de incluir en la presentación final el uso de *burndown charts* y JIRA, para mostrar información puntual y relevante.

En cuanto a la ingeniería de requerimientos, se hizo hincapié en la identificación, especificación y validación de los mismos. Además, se sugirió incluir técnicas de identificación de requerimientos como reuniones y *mockups*, aunque no todo tiene que ser incluido en la presentación final. También se recomendó concentrarse en la metodología Scrum, sin profundizar en detalles ya conocidos por el equipo. A su vez, se sugiere adaptar la presentación y no entrar en demasiados detalles que ya han sido leídos por los correctores.

En cuanto a la arquitectura, se sugiere evaluar la posibilidad de dividir en varias funciones Lambda y verificar cómo cambia la cantidad de tiempo de procesamiento. Se menciona la importancia de establecer varios escenarios en distintos volúmenes y tiempos de respuesta, a fin de obtener una evaluación repetible y justificable de los tiempos de respuesta.

En cuanto a la mejora de la estimación, se recomendó incluir un "colchón" y partir las US de forma más detallada. También se sugirió explicar con más detalle lo aprendido en las retrospectives.

Respecto a los riesgos, se planteó la necesidad de establecer estrategias para prevenir y mitigar los mismos. Se mencionó que, según el PMBOK, existen tres estrategias para riesgos positivos y tres para riesgos negativos, que incluyen evitar, mitigar o aceptar los riesgos.

Por último, se destacó la importancia del plan de calidad, que incluye tanto la calidad del producto para el cliente como la calidad para el proceso de ingeniería y los de apoyo. Se enfatizó la necesidad de incluir un plan de comunicación y un plan de configuración, y justificar cómo estos mejorarán el proceso de mejora continua.

En conclusión, la revisión con Rafael Bentancour permitió identificar áreas de mejora en el proyecto y establecer estrategias para optimizarlo. Se destacó la importancia de establecer objetivos claros y medibles, corregir el manejo de los tiempos, incluir técnicas de identificación de requerimientos y establecer estrategias de mitigación de riesgos.

Segunda revisión - 29 de septiembre de 2022

Revisor: MSc. Gastón Mousques

Informe

En esta revisión, el profesional Gastón Mousques hace varias recomendaciones para mejorar la presentación del trabajo. En primer lugar, sugiere que se detalle mejor la funcionalidad del sistema y se especifique porque aportan los requerimientos funcionales al cliente. También indica que es necesario detallar más el volumen de los requerimientos y especificar mejor el problema que se quiere solucionar. Además, sugiere destacar la complejidad del problema debido a la existencia de distintos proveedores y formatos.

En cuanto a la presentación, Mousques sugiere que se destaquen mejor los puntos mencionados anteriormente, así como los objetivos y los beneficios del servicio. También se debe agregar mayor información sobre las validaciones de atributos de calidad y la correlación entre el usuario y el beneficio.

En términos de gestión, se recomienda incluir más información sobre el Product Backlog, el tamaño de Sprint y la velocidad del equipo. También es importante definir todas las actividades que ayuden a asegurar la calidad del proceso, incluyendo los estándares de programación y los *checklist*.

Finalmente, Mousques sugiere justificar más la elección de tecnologías, hablar de análisis de costos y mostrar prototipos de los requerimientos funcionales atractivos en la presentación. También recomienda que el equipo unifique el QA con la gestión y se defina el proceso formalmente para asegurar la calidad del trabajo.

Tercera revisión - 22 de diciembre de 2022

Revisor: MSc. Álvaro Ortas

Informe

En la revisión del proyecto, se discutieron varias ideas y se hicieron comentarios sobre la gestión del mismo. Se elogió la buena gestión del proyecto y la excelente selección de tecnologías. Además, Álvaro destacó el proyecto como desafiante, y sugiere al equipo intentar explotar eso en la exposición del mismo.

Entre los comentarios realizados, se mencionaron algunos problemas en la comunicación y presentación de la demo. En particular, la demo no resultó tan interesante como se esperaba. Se sugiere que se cambie el orden de presentación de la información, comenzando por los requerimientos, luego la demo y finalmente el proceso a fin de que el espectador pueda visualizar el producto primero. También es importante mejorar la comunicación de los requerimientos, ya que no quedó claro el alcance de las tareas.

Los objetivos planteados en el proyecto son buenos, pero se sugiere agregar más información sobre cómo se medirán los resultados. Por ejemplo, el primer objetivo podría medirse con encuestas de satisfacción, mientras que el segundo objetivo podría medirse con parámetros de QA. Estas métricas deberían ser presentadas en las conclusiones. También se sugirió que se podrían incluir objetivos de calidad.

En cuanto al proceso y ciclo de vida, se sugiere explayarse más sobre la información presentada en la imagen, y considerar el modelo de 4+1. También se recomienda analizar si es necesario incluir el ciclo de vida iterativo e incremental en el proyecto, y hablar más sobre la metodología Scrum. Es importante mencionar que el equipo del proyecto está compuesto por tres personas, lo que limita la capacidad de trabajo y puede generar horarios disjuntos, por lo cual se sugiere explicar por qué se usa Weekly Scrum y también explicar que la frecuencia de las reuniones con el Product Owner se basa en que su tiempo es limitado.

En cuanto a la ingeniería de requerimientos, se sugiere ampliar la información presentada en el proyecto y hablar más sobre el proceso de Design Thinking. Luego, comentó que es necesario aclarar que no se utilizó la IEEE para realizar la definición de restricciones justificando que se clasificó de esa manera porque el equipo se basó la arquitectura en un libro recomendado.

Para mejorar el proyecto, se sugiere que se tenga en cuenta que se trata de una prueba de concepto y que, por lo tanto, no es necesario tener un alto grado de aseguramiento de calidad. También es importante definir qué hacer con cada métrica, y que valor aporta en los diferentes aspectos del proyecto.

10.12 Ejemplos Sesiones exploratorias

En este anexo se exponen tres ejemplos de los documentos de sesiones exploratorias, que corresponden a la búsqueda de documentos, subida de los mismos y creación de tareas de estos.

Ejemplo 1

Sesión exploratoria: búsqueda de documentos

- **Tester**: Diego Franggi
- **Fecha de Inicio**: 17/12/2022
- **Hora de inicio**: 8:53
- **Hora de finalización**: 11:23
- **Duración total**: 2 horas y 30 minutos
- **Sistema Operativo**: ParrotOS 5.1 (Debian)
- **Navegador**: Mozilla Firefox

Misión:

La misión de la prueba es encontrar y reportar posibles problemas y defectos en la funcionalidad de búsqueda rápida de documentos.

Objetivos:

El objetivo principal es identificar cualquier comportamiento inesperado o no deseado en la funcionalidad de búsqueda, que pueda afectar la usabilidad o la calidad general del sistema. Además, se busca asegurarse de que la búsqueda devuelva resultados precisos y relevantes para las consultas realizadas por los usuarios.

Los objetivos específicos de la prueba de búsqueda de documentos son los siguientes:

- Evaluar la capacidad del sistema para recuperar los documentos relevantes en función de las palabras clave y otros criterios de búsqueda.
- Probar diferentes tipos de consultas de búsqueda para asegurarse de que el sistema pueda manejarlas correctamente.
- Evaluar la usabilidad de la interfaz de búsqueda, asegurándose de que los usuarios puedan comprender y usar la funcionalidad de búsqueda de manera efectiva.
- Identificar y reportar problemas como errores de búsqueda, resultados irrelevantes, tiempos de respuesta lentos, problemas de paginación, entre otros.
- Asegurar que la funcionalidad de búsqueda cumpla con los requisitos del cliente y las expectativas de los usuarios.

Ejemplo 2

Sesión exploratoria: creación de tareas de un documento

- **Tester**: Alejandro Bermúdez
- **Fecha de Inicio**: 04/01/2023
- **Hora de inicio**: 19:29
- **Hora de finalización**: 21:33
- **Duración total**: 2 horas y 4 minutos
- **Sistema Operativo**: Windows 11
- **Navegador**: Google Chrome

Misión:

La misión de la prueba para la funcionalidad de creación de tareas vinculadas a un documento es encontrar y reportar posibles problemas y defectos en la capacidad del sistema para crear y gestionar tareas relacionadas con documentos específicos.

Objetivos:

El objetivo principal de la prueba es identificar cualquier comportamiento inesperado o no deseado en la funcionalidad, que pueda afectar la usabilidad o la calidad general del sistema. Además, se busca asegurarse de que las tareas se creen y asignen correctamente, y que los usuarios puedan realizar un seguimiento efectivo de las tareas pendientes y completadas.

Los objetivos específicos de la prueba de creación de tareas para un documento son los siguientes:

- Evaluar la capacidad del sistema para crear tareas y asignarlas a documentos específicos.
- Probar la funcionalidad de actualización de estado para asegurarse de que los usuarios puedan ver de forma rápida y efectiva los cambios de estado que puedan sufrir las tareas.
- Evaluar la capacidad del sistema para permitir la asignación de tareas a diferentes personas.
- Identificar y reportar problemas como la creación de tareas duplicadas, asignación incorrecta de tareas o seguimiento inadecuado de tareas pendientes y completadas.
- Asegurarse de que la funcionalidad de creación de tareas vinculadas a un documento cumpla con los requisitos del cliente y las expectativas de los usuarios.

Ejemplo 3

Sesión exploratoria: carga de documentos

- **Tester:** Federico Martínez
- **Fecha de Inicio:** 26/08/2022

- **Hora de inicio:** 17:20
- **Hora de finalización:** 19:07
- **Duración total:** 1 hora y 47 minutos
- **Sistema Operativo:** macOS Ventura 13.2.1
- **Navegador:** Google Chrome y Safari

Misión:

La misión de la prueba es encontrar y reportar posibles problemas y defectos en la capacidad del sistema para cargar, y almacenar los documentos.

Objetivos:

El objetivo principal de la prueba es identificar cualquier comportamiento inesperado o no deseado en la funcionalidad de carga de documentos, que pueda afectar la usabilidad o la calidad general del sistema. Además, se busca asegurarse de que los documentos cargados sean almacenados y queden disponibles para el procesamiento y análisis de los mismos.

Los objetivos específicos de la prueba de carga de documentos son los siguientes:

- Evaluar la capacidad del sistema para cargar y almacenar documentos .pdf de diferentes tamaños y formatos, asegurándose de que el proceso de carga sea rápido y confiable.
- Identificar y reportar problemas como errores de carga, documentos dañados o ilegibles, problemas de visualización, entre otros.
- Asegurar que los documentos cargados estén protegidos y sean accesibles solo para usuarios autorizados.
- Garantizar que la funcionalidad de carga de documentos cumpla con los requisitos del cliente y las expectativas de los usuarios.