UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

POSGRADOS

MAESTRÍA EN

SOFTWARE CON MENCIÓN EN DISEÑO DE ARQUITECTURA DE SISTEMAS

RPC-SO-34-NO.778-2021

Opción de Titulación:

Proyecto de titulación con componentes de investigación aplicada y/o de desarrollo

Tema:

Diseño de la arquitectura de una aplicación web para el almacenamiento y publicación de parámetros eléctricos del GIREI

Autor(es)

José Darwin Castillo Jumbo

Director:

Rodrigo Efrain Tufiño Cardenas

Quito – Ecuador
2025

Autor(es):

José Darwin Castillo Jumbo
Ingeniero en Sistemas y Computación
Candidato a Magíster en Software por la Universidad Politécnica Salesiana – Sede Quito.
jcastillo2@est.ups.edu.ec

Dirigido por:

Rodrigo Efrain Tufiño Cardenas
Ingeniero en Sistemas
Máster Universitario en Ciencias y Tecnologías de la Computación
rtufino@ups.edu.ec

DISEÑO DE LA ARQUITECTURA DE UNA APLICACIÓN WEB PARA EL ALMACENAMIENTO Y PUBLICACIÓN DE PARÁMETROS ELÉCTRICOS DEL GIREI

DEDICATORIA

A mis padres, José Castillo y María Jumbo, por ser el ejemplo más puro de trabajo, perseverancia y dedicación. Gracias por enseñarme que los sueños se construyen con esfuerzo y que, aunque el camino pueda ser difícil, nunca debemos renunciar a nuestras metas. Ustedes han sido mi guía y mi inspiración, y a través de su ejemplo he aprendido que, con dedicación y constancia, todo es posible.

A mi amada esposa, Ana Obando, y a mis pequeños, Miguelito y Bianquita. Son ustedes el motor de mi vida, la razón por la que cada día busco superarme y dar lo mejor de mí. Gracias, Ana, por tu apoyo incondicional y por caminar a mi lado en cada etapa de este viaje. A Miguelito y Bianquita, que adornan mis días con amor y felicidad, sepan que todo lo que realizo es con el objetivo de edificar un futuro más prometedor para ustedes y convertirme en una mejor versión de mí mismo.

AGRADECIMIENTO

Quiero expresar mi agradecimiento a mi tutor, Rodrigo Tufiño, por su paciencia, dedicación y constante orientación durante este proceso académico. Agradezco su tiempo, generosidad al compartir su conocimiento y orientación que ha contribuido a mi progreso con claridad y confianza. Su colaboración ha sido esencial para el éxito de este proyecto.

A mi esposa, y a mis hijos, por su comprensión y sacrificio. Gracias por cada momento que no pudimos compartir y por apoyarme incondicionalmente mientras dedicaba largas horas al estudio. Su amor y paciencia me dieron la fortaleza para alcanzar este logro. Este proyecto también es de ustedes, porque sin su apoyo, nada de esto habría sido posible.

Y finalmente, me permito agradecerme a mí mismo, por la perseverancia, el esfuerzo y la dedicación que he puesto en este camino. Este logro es el resultado de muchas horas de trabajo, de superar desafíos y de mantener la motivación. Hoy reconozco el valor de cada paso y me siento orgulloso de haber llegado hasta aquí.

Tabla de Contenido

ÍNDICE DE FIGURAS

Diseño de la arquitectura de una aplicación web para el almacenamiento y publicación de parámetros eléctricos del GIREI

Autor(es):

José Darwin Castillo Jumbo

Resumen

Este estudio propone un diseño arquitectónico para una futura aplicación destinada a gestionar y visualizar datos eléctricos para el Grupo de Investigación de Redes Eléctricas Inteligentes (GIREI) de la Universidad Politécnica Salesiana. La investigación no implica el desarrollo de una aplicación web, móvil o de escritorio, sino que se centra en la creación de una arquitectura adaptable. El objetivo es proporcionar una solución independiente para gestionar los datos generados por los dispositivos inteligentes dentro del sistema eléctrico de GIREI, como baterías, inversores y paneles solares, sin depender de plataformas de monitorización de terceros como VRM de Victron, aunque funcional, carece de la flexibilidad y personalización que requiere el GIREI.

La arquitectura propuesta utiliza Redis, RabbitMQ e InfluxDB para el manejo de datos, incluida la recopilación, el almacenamiento temporal, permanente y el procesamiento de datos eléctricos. El modelo arquitectónico C4 organiza el sistema en componentes modulares, lo que permite una escalabilidad futura y facilidad de mantenimiento. Los componentes clave de la arquitectura incluyen el *conector de fuente de datos*, *el productor y el consumidor de datos*, que recopilan datos en tiempo real de los dispositivos a través del protocolo de comunicación Modbus y los almacena en una base de datos de series temporales. Un sistema de registro centralizado basado en Elasticsearch y Kibana admite la monitorización de errores en tiempo real, lo que contribuye al sistema y la integridad de los datos. Una prueba de concepto confirma que la arquitectura propuesta puede respaldar una aplicación futura para monitorear y visualizar datos eléctricos, lo que proporciona una solución independiente, adaptable para optimizar la gestión de la energía.

Palabras clave:

Arquitectura de software, monitorización en tiempo real, Almacenamiento de datos eléctricos, Redis, RabbitMQ, InfluxDB, Elasticsearch, Kibana.

ABSTRACT

This study proposes an architectural design for a future application aimed at managing and visualizing electrical data for the Grupo de Investigación de Redes Eléctricas Inteligentes (GIREI) at the Universidad Politécnica Salesiana. The research does not involve the development of a web, mobile, or desktop application, but rather focuses on creating a scalable and adaptable architecture. The objective is to provide an independent solution for managing data generated by intelligent devices within GIREI's electrical system, such as batteries, inverters, and solar panels, without relying on third-party monitoring platforms like Victron's VRM, which, although functional, lacks the flexibility and customization required by GIREI.

The proposed architecture leverages Redis, RabbitMQ, and InfluxDB for data handling, encompassing collection, temporary and permanent storage, and processing of electrical data. The C4 architectural model structures the system into modular components, enabling seamless scalability and maintenance. Key architectural components include the data source connector, data producer, and data consumer, which collectively gather real-time data from devices via Modbus and store it in a time-series database. A centralized logging system, built on Elasticsearch and Kibana, facilitates real-time error monitoring, bolstering system reliability and data integrity. A proof-of-concept validation confirms the proposed architecture's ability to effectively support a future application for monitoring and visualizing electrical data, providing an independent, efficient, and adaptable solution to optimize energy management.

Keywords:

Software architecture, Real-time monitoring, Electrical data storage, Redis, RabbitMQ, InfluxDB, Elasticsearch, Kibana.

1. Introducción

1.1 Antecedentes

El Grupo de Investigación en Redes Eléctricas Inteligentes (GIREI) de la Universidad Politécnica Salesiana goza de reconocimiento tanto a nivel nacional como internacional debido a su destacada trayectoria y contribuciones significativas al sector eléctrico. A través de una serie de investigaciones académicas y la colaboración con diversas instituciones públicas y privadas dentro y fuera del país, así como la ejecución de proyectos relacionados con energías renovables, el GIREI ha demostrado su liderazgo y compromiso con la innovación en este campo (GIREI, 2024).

La presente investigación se enfoca en un aspecto crucial para el GIREI: el almacenamiento de información relacionada con la generación masiva de parámetros eléctricos. Mediante el aprovechamiento de herramientas tecnológicas y el diseño de una arquitectura de software adecuada, se pretende desarrollar una aplicación web que facilite el control, acceso, almacenamiento masivo y difusión de esta información de manera eficiente.

Es fundamental destacar que uno de los principales objetivos del GIREI es la transferencia ágil de los resultados de sus investigaciones para influir en la toma de decisiones en el ámbito eléctrico ecuatoriano, en consonancia con el contexto regional, nacional e internacional.

Por lo tanto, la gestión de la información generada por el GIREI es un aspecto crucial, especialmente para la realización de estudios técnicos, pronósticos de producción de energía y la promoción de la accesibilidad a la información de manera rápida y precisa. Para alcanzar este objetivo, es necesario llevar a cabo un exhaustivo proceso de investigación. En primer lugar, se enfocará en comprender a fondo las prácticas y procedimientos utilizados por el grupo en su día a día.

Posteriormente, se analizarán y estudiarán los requisitos necesarios para el diseño de una aplicación web que se ajuste a las necesidades específicas del GIREI.

1.2 Determinación del problema

1.2.1 Descripción del problema

El Grupo de Investigación en Redes Eléctricas Inteligentes (GIREI) enfrenta un desafío crucial: la carencia de un sistema web específico para almacenar los parámetros eléctricos de la micro red inteligente. En la actualidad, la gestión de datos queda en manos de los fabricantes de los dispositivos de comunicación que componen la micro red, lo que dificulta la generación de gráficos, análisis estadísticos, predicciones y la automatización de tareas. Esta dependencia también obstaculiza el acceso confiable y seguro a la información por parte de usuarios de todo el mundo (GIREI, 2024).

1.2.2 Formulación del problema

¿Logrará el diseño de una arquitectura de software proporcionar las pautas y lineamientos para el desarrollo de una aplicación web que almacene y publique parámetros eléctricos que maneja el GIREI?

¿Hasta qué punto el diseño de la arquitectura garantiza la escalabilidad y tolerancia a fallos al momento de desarrollar la aplicación web?

¿Qué tipo de beneficios obtendrá el grupo GIREI con el diseño de una arquitectura para una aplicación web?

1.2.3 Justificación del problema

Con base en lo expuesto anteriormente, se hace evidente la necesidad de desarrollar una arquitectura de aplicación web dedicada a almacenar y publicar los parámetros eléctricos del Grupo GIREI. Es crucial considerar tanto los costos directos como los indirectos asociados con el desarrollo de esta solución, con el fin

de optimizar el análisis de la inversión y determinar las estrategias y recursos más adecuados para el desarrollo del aplicativo web.

Además, es importante destacar que esta problemática tendrá un impacto positivo significativo en la investigación llevada a cabo por la Universidad Politécnica Salesiana, particularmente en las actividades diarias del Grupo GIREI. La simple acción de almacenar y publicar los parámetros eléctricos mejorará considerablemente el acceso a la información y facilitará el proceso de análisis de datos. Resolver este problema no solo evitará la pérdida de tiempo en la recolección y medición de información, sino que también promoverá una toma de decisiones más informada y eficiente.

1.2.4 Delimitación del problema

La propuesta se llevará a cabo en la Universidad Politécnica Salesiana para el Grupo de Investigación en Redes Eléctricas Inteligentes (GIREI), en el Campus. Se tiene como objetivo diseñar y evaluar una arquitectura de aplicación web para almacenar y publicar los parámetros eléctricos del GIREI.

El proyecto abarcará la investigación de las necesidades del Grupo de Investigación, el diseño y desarrollo de la arquitectura de la aplicación web, la implementación de funcionalidades clave y pruebas exhaustivas. Al finalizar, se espera contar con una herramienta que mejore la gestión y compartición de información sobre la micro red inteligente, impulsando la investigación en redes eléctricas inteligentes.

2. Determinación del Problema

2.1 Justificación del problema

Con base en lo expuesto anteriormente, se hace evidente la necesidad de desarrollar una arquitectura de aplicación web dedicada a almacenar y publicar los parámetros eléctricos del Grupo GIREI. Es crucial considerar tanto los costos directos como los indirectos asociados con el desarrollo de esta solución, con el fin de optimizar el análisis de la inversión y determinar las estrategias y recursos más adecuados para el desarrollo del aplicativo web.

Además, es importante destacar que esta problemática tendrá un impacto positivo significativo en la investigación llevada a cabo por la Universidad Politécnica Salesiana, particularmente en las actividades diarias del Grupo GIREI. La simple acción de almacenar y publicar los parámetros eléctricos mejorará considerablemente el acceso a la información y facilitará el proceso de análisis de datos. Resolver este problema no solo evitará la pérdida de tiempo en la recolección y medición de información, sino que también promoverá una toma de decisiones más informada y eficiente.

2.2 Objetivos

2.2.1 Objetivo general

Diseñar la arquitectura de una aplicación web para el almacenamiento y publicación de parámetros eléctricos del Grupo de Investigación en Redes Eléctricas Inteligentes (GIREI) de la Universidad Politécnica Salesiana.

2.2.2 Objetivos específicos

- Analizar los componentes, dispositivos y data de los que dispone el GIREI.
- Seleccionar una base de los datos idónea y acorde a la estructura de los datos a almacenar.

- Definir la arquitectura de la aplicación considerando las interfaces de comunicación con elementos internos y externos.
- Desarrollar una prueba de concepto de la aplicación basada en la arquitectura propuesta.

3. Marco teórico referencial

Las argumentaciones consideradas importantes para el sustento de la presente investigación son las siguientes.

3.1 Arquitecturas de software

La arquitectura de software es fundamental en el desarrollo y diseño de cualquier sistema informático, ya que no solo establece las bases iniciales del proyecto, sino que también guía su evolución a lo largo de todo su ciclo de vida. Entre los beneficios clave de una sólida arquitectura de software se encuentran la garantía de integridad conceptual, la facilitación de la reutilización de componentes y la promoción de una comunicación entre los diferentes elementos del sistema (Martín, 2018).

La arquitectura de software tiene patrones que son soluciones genéricas y reutilizables para resolver problemas repetitivos de ingeniería de software en un contexto específico. Se pueden ver algunas similitudes con el modelo de programación, pero en este caso se prefiere la estructura común de nivel superior.

3.2 Importancia de la arquitectura

La arquitectura desempeña un papel fundamental al permitir la planificación anticipada del desarrollo y la selección de las herramientas más adecuadas para la implementación de proyectos. Es un paso crítico previo a la programación, ya que influye significativamente en el ritmo de desarrollo y en los aspectos financieros durante todo el proceso. Por tanto, al elegir un modelo arquitectónico, es imprescindible tener en cuenta varios factores que determinarán el uso final del software.

- **Costo:** ¿Cuánto está dispuesto a invertir en el desarrollo del sistema y su posterior mantenimiento? Existen diferentes modelos que pueden ser más complejos que otros y

tener una mayor infraestructura, lo que puede hacer que el proceso de desarrollo sea más desigual. Conocer el tamaño de la inversión da un mejor margen de gestión.

- **Duración del desarrollo.** La duración del desarrollo es igual de importante que la fecha de entrega inicial o lanzamiento del sistema. El tiempo estimado de desarrollo debe ser considerado cuidadosamente junto con otros aspectos del proyecto.
- **Número de usuarios**. El número de usuarios es un factor crucial para considerar al diseñar la arquitectura de software. ¿Cuántos usuarios se espera que utilicen la aplicación? ¿Se trata de una aplicación web? Estas interrogantes guían hacia la adopción de un patrón arquitectónico más o menos distribuido, como, por ejemplo, un patrón en capas, menos distribuido o intermedio.
- **Grados de aislamiento.** ¿Funciona solo o está integrado con otros productos? ¿O permite la integración de terceros? Es importante elegir una arquitectura que proporcione la escalabilidad necesaria para el producto (Regueros, 2020).

3.3 Tipos de Arquitecturas de Desarrollo de Software

Con el avance de la tecnología, no es siempre necesario crear nuevas arquitecturas de desarrollo de software, ya que existen muchas opciones establecidas que pueden cumplir con los requisitos del usuario. A continuación, se presentan algunas de las arquitecturas más relevantes para el diseño de una aplicación web destinada al almacenamiento y publicación de parámetros eléctricos.

3.3.1 Arquitectura Monolítica

Este estilo arquitectónico se basa en crear una aplicación que tenga toda la funcionalidad para lograr el propósito para el cual fue creada, sin necesidad de componentes externos o dependencias que soporten su funcionalidad. Esto significa que los componentes de la aplicación pueden trabajar juntos y compartir recursos (Tacury, 2023).

A continuación, se presenta la manera en cómo funciona un aplicativo según este tipo de arquitectura.

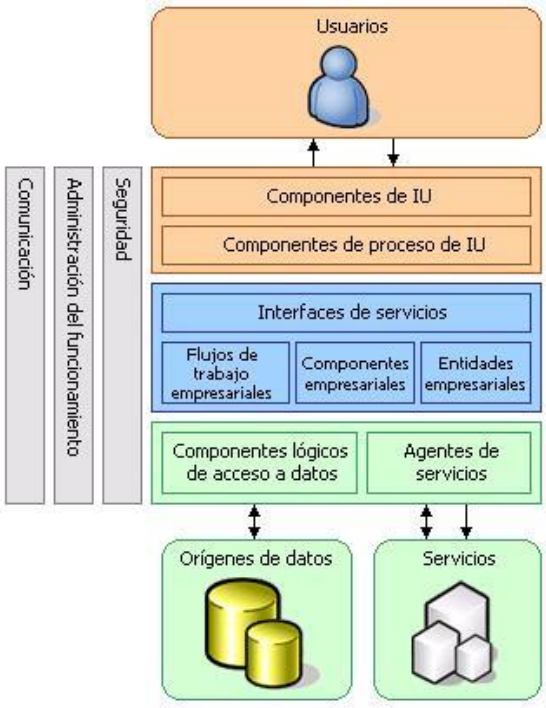

Figura 1. Arquitectura Monolítica.
Nota: tomado de (Guijarro, 2023)

3.3.2 Arquitecturas de microservicios

Los microservicios son módulos ligeros con un bajo acoplamiento que pueden utilizarse como componentes básicos en sistemas complejos en la nube. A diferencia de las arquitecturas monolíticas tradicionales, donde una aplicación se ejecuta en un solo servidor físico o virtual, los microservicios permiten que cada función se ejecute de forma independiente, con múltiples instancias distribuidas en uno o varios servidores según sea necesario para manejar la carga de trabajo López (2017). Esta capacidad de escalabilidad dinámica hace que los microservicios sean altamente adaptables a las demandas cambiantes de la carga de trabajo.

Cada microservicio se especializa en una tarea específica y se enfoca completamente en ella. Esto resulta en una arquitectura altamente cohesiva, donde todas las funciones están

estrechamente relacionadas con la resolución de un problema particular. A diferencia de las aplicaciones monolíticas, que son grandes y complejas, la arquitectura de microservicios divide la funcionalidad en componentes de software pequeños y autónomos, cada uno con una única responsabilidad López (2017). Este enfoque proporciona una mayor flexibilidad y modularidad, facilitando la implementación, el mantenimiento y la escalabilidad de la aplicación en general.

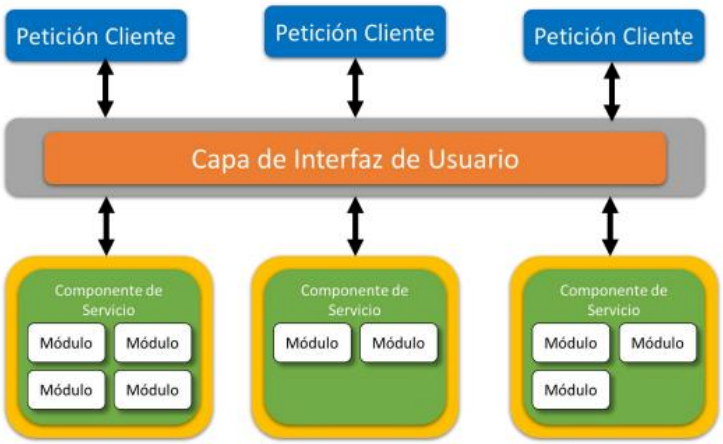

Figura 2. Arquitectura de Microservicios.
Nota: tomado de (AWS, 2023)

Categoría	Arquitectura Monolítica	Arquitectura de microservicios
Código	Una base de código única para toda la aplicación.	Múltiples bases de código. Cada microservicio tiene su propia base de código.
Comprensibilidad	A menudo confuso y difícil de mantener.	Mayor facilidad de lectura y mucho más fácil de mantener.
Despliegue	Implementaciones complejas con ventanas de mantenimiento y paradas programadas.	Despliegue sencillo ya que cada microservicio se puede implementar de forma individual, con un tiempo de inactividad mínimo, si no es cero.
Lenguaje	Típicamente totalmente desarrollado en un lenguaje de programación.	Cada microservicio puede desarrollarse en un lenguaje de programación diferente.
Escalamiento	Requiere escalar la aplicación entera, aunque los cuellos de botella estén localizados.	Cada microservicio puede desarrollarse en un lenguaje de programación diferente.

Figura 3. Arquitectura monolítica vs microservicio
Nota: tomado de (López, 2017)

3.4 Sistema SCADA (Supervisory Control and Data Acquisition)

El sistema SCADA, por sus siglas en inglés *Supervisory Control and Data Acquisition*, es una herramienta fundamental en la industria para supervisar y controlar procesos industriales en tiempo real. Sus principales funciones incluyen la supervisión de variables operativas, como temperatura, presión, flujo y nivel, así como la recopilación y análisis de datos para tomar decisiones informadas Pérez-López (2015). Además, facilita el control remoto de dispositivos y procesos, lo que permite ajustes y correcciones en tiempo real para optimizar la eficiencia y la seguridad.

Los elementos principales de un sistema SCADA incluyen sensores y actuadores, que son dispositivos encargados de recopilar datos del entorno y realizar acciones físicas, respectivamente. Estos dispositivos están conectados a unidades remotas (RTU) o controladores lógicos programables (PLC), que se encargan de procesar la información recopilada y enviarla al sistema central de supervisión. El sistema central, a su vez, está compuesto por una interfaz de usuario gráfica que muestra la información en tiempo real, alarmas y eventos, así como herramientas de análisis y generación de informes para la toma de decisiones Pérez-López (2015). En síntesis, el sistema SCADA es una herramienta integral que proporciona supervisión, control y análisis de procesos industriales para mejorar la eficiencia, la seguridad y la productividad.

3.5 Intert Of Things (IoT)

El Internet de las cosas (IoT) representa una infraestructura clave para la gestión y la recopilación de datos en sistemas de redes eléctricas inteligentes. Conectando dispositivos y sensores a través de una red interconectada, IoT facilita la monitorización remota, el control automatizado y el análisis en tiempo real de parámetros eléctricos, contribuyendo así a la mejora de la eficiencia energética y la toma de decisiones informadas en el ámbito eléctrico (Pérez-López, 2015).

3.5.1 Arquitectura de un sistema IoT

El Internet de las cosas (IoT) representa una infraestructura clave para la gestión y la recopilación de datos en sistemas de redes eléctricas inteligentes. Conectando dispositivos y sensores a través de una red interconectada, IoT facilita la monitorización remota, el control automatizado y el análisis en tiempo real de parámetros eléctricos, contribuyendo así a la mejora de la eficiencia energética y la toma de decisiones informadas en el ámbito eléctrico (Pérez-López, 2015).

En cuanto a la arquitectura de un sistema IoT, existen diversos modelos que permiten la integración de tecnologías clave. Cada diseño tiene sus características principales, que se dividen en tres componentes principales: hardware, middleware y capa de presentación o servicio web Pérez-López (2015). Dentro de los modelos de arquitectura IoT, se destacan tres principales: el modelo de 3 capas, el modelo de 4 capas y el modelo de 5 capas. El primero comprende una capa de aplicación para servicios al usuario, una capa de pasarela para la interconexión de servidor y dispositivos de red, y una capa de dispositivo para el procesamiento y difusión de datos adquiridos por los sensores. El modelo de 4 capas es similar al anterior, pero incluye una capa de integración que almacena y recibe datos del Gateway para su procesamiento. El modelo de 5 capas, el más utilizado, agrega una capa de negocios que administra todo el sistema IoT, incluyendo rendimiento, aplicaciones y seguridad de la información, ofreciendo la optimización y mejora de procesos (Pérez-López, 2015).

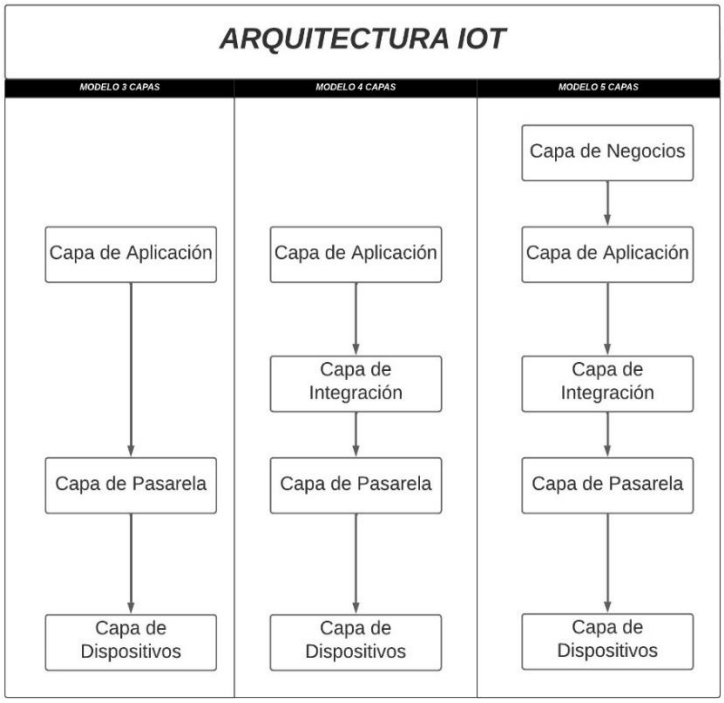

Figura 4. Tipos de arquitectura IoT.
Nota: tomado de (Pérez-López, 2015)

3.5.2 Protocolos de IoT

En el ámbito del Internet de las cosas (IoT), una variedad de protocolos de comunicación, conocidos como "middleware", se emplean ampliamente para la conectividad punto a punto. Destacan entre ellos el MQTT y CoAP, que se basan directamente en el stack de protocolos de TCP/IP. Estos protocolos, junto con otros como DDS, AMQP, OPC UA y HTTP (REST/JSON), desempeñan roles fundamentales en la transmisión de datos y la interoperabilidad en entornos IoT Madakam, et al., (2015)

3.5.2.1. Protocolo MQTT (Message Queuing Telemetry Transport)

El protocolo MQTT es una tecnología de mensajería ligera diseñada para el Internet de las cosas (IoT). Utiliza un modelo de publicación/suscripción para la comunicación entre dispositivos, permitiendo una transmisión de datos en redes con ancho de banda limitado.

Es conocido por su bajo consumo de recursos y su capacidad para manejar conexiones de red intermitentes, lo que lo convierte en una opción popular para dispositivos con recursos limitados y entornos IoT (ELSIST, s/f).

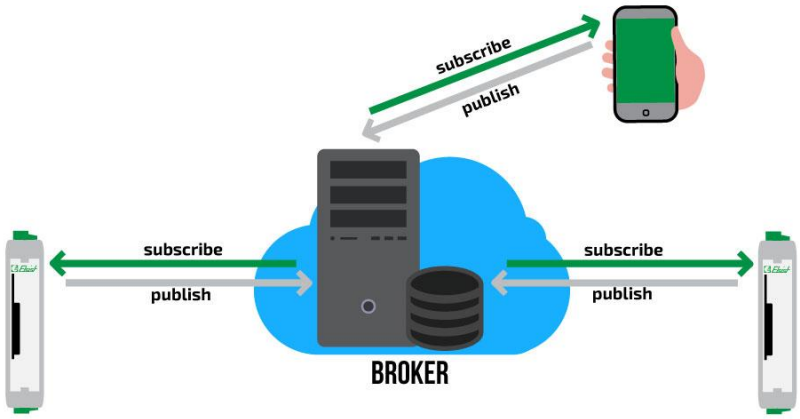

Figura 5. Protocolo MQTT.
Nota: tomado de (ELSIST, s/f)

Esto significa que los dispositivos de MONITORIZACIÓN, como sensores y equipos de medición utilizados por el GIREI, pueden publicar datos sobre parámetros eléctricos en tiempo real a través del protocolo MQTT. A su vez, el servidor central puede suscribirse a estos datos, recibiendo y almacenando la información de manera eficiente. Gracias a su bajo consumo de recursos y su capacidad para manejar conexiones intermitentes, MQTT se adapta bien a los requisitos de un sistema de almacenamiento de datos para el GIREI, garantizando una comunicación confiable entre los dispositivos de monitorización y el servidor central.

3.6 Redes Mobdus

Para intercambiar solicitudes y respuestas, los dispositivos de la red Modbus organizan los datos en marcos. Debido a que este es un protocolo a nivel de aplicación, se debe utilizar en una pila de protocolos que aborde cuestiones específicas relacionadas con el tipo de red utilizada. Modbus se divide en tres tipos según la arquitectura de protocolo utilizada: RTU, ASCII y TCP/IP (Maldonado, 2021).

3.6.1 Mobdus RTU

La característica de Modbus RTU (Unidad Terminal Remota o también conocido en inglés como Remote Terminal Unit) caracterizado porque los bytes se envían en código binario plano sin ningún tipo de conversión. Originalmente se utilizó para comunicaciones de bus serie. Su principal ventaja es que hace un buen uso del canal de comunicación, aumentando así la tasa de transferencia de datos. La desventaja es que se necesita tiempo entre los bytes recibidos para saber cuándo comienza y termina una trama.

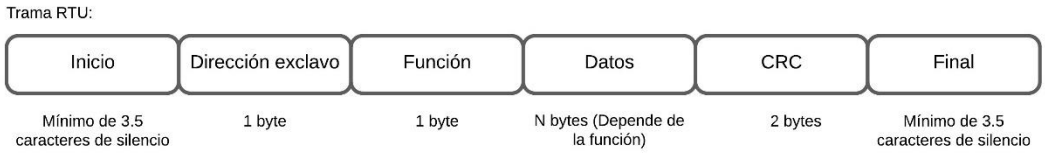

Figura 6. Formato de trabas de Modbus serie.
Nota: tomado de (Maldonado, 2021)

Para tramas Modbus RTU, la delimitación de intervalos de tiempo se realiza en caracteres silenciosos. La duración de un carácter silencioso es la longitud de un byte de datos enviados a través del medio, pero no se transmite ningún dato. Su duración (T) depende de la velocidad (Vt) y del número de bits codificados según T (N) = N/Vt. El estándar Modbus para velocidades de hasta 19200 bps requiere que el tiempo entre fotogramas sea al menos 3,5 veces la longitud de los caracteres, pero se recomienda un tiempo fijo de 1,75 ms para velocidades más altas. Por ejemplo, para uno el puerto serie está configurado para 19.200 bps, con un bit de parada y un bit de paridad (11 bits en total, más un bit de inicio y 8 bits de datos), tenemos: 3,5·11/19.200=2ms.

3.6.2 Mobdus ASCII

Los datos se codifican como caracteres ASCII entre "0" (16#30) y "9" (16#39) y entre "A" (16#41) y "F" (16#46). Por ejemplo, si se va a transferir el byte de valor 16#FF, se debe

transferir la cadena "FF", en realidad se transfieren dos bytes: 16#46 y 16#46. Además, se utilizan 3 caracteres especiales. El carácter ":" (16#3A) se utiliza para marcar el comienzo de un fotograma. Los pares Modbus RTU y ASCII están diseñados para usarse directamente en medios físicos serie asíncronos como EIA/TIA RS-232, EIA/TIA RS-485 o EIA RS-422.

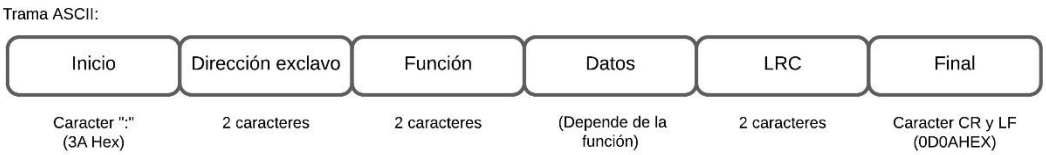

Figura 1. Formato de trabas de Modbus ASCII
Nota: tomado de (Maldonado, 2021)

3.6.3 Mobdus TCP/IP

Modbus TCP, por otro lado, fue diseñado para funcionar en redes que utilizan la arquitectura TCP/IP, por lo que permite utilizar Modbus en redes como Ethernet o WiFi. En la práctica, el servicio de mensajes utiliza una técnica que encapsula cada mensaje Modbus en una trama TCP (excepto los dos bytes en el campo de control de errores Modbus o la dirección descendente). La casilla de verificación no incluye la detección de errores porque Modbus TCP/IP contiene sus propias respuestas de error en la capa BUSCAR. Las direcciones dependientes se eliminan si las direcciones IP de conexión, origen y destino y la dirección MAC del dispositivo conectado se definen en el nivel de transporte, red o enlace. Esto es una gran ventaja ya que supera ampliamente el límite de hasta 247 unidades en modo RTU y aumenta la posibilidad de conectar más unidades.

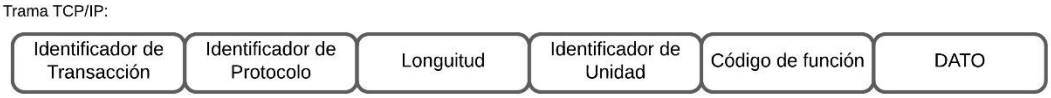

Figura 2. Formato de trabas de Modbus TCP/IP
Nota: tomado de (Maldonado, 2021)

3.7 Cloud computing para servicios de IoT

La computación en la nube es actualmente una tecnología que brinda servicios a través de Internet basados en servidores a los que se puede acceder en tiempo real desde cualquier lugar Shapsough et al., (2018). Su función principal es acceder, almacenar y gestionar los datos obtenidos por el dispositivo, ver figura 9 para una mejor ilustración.

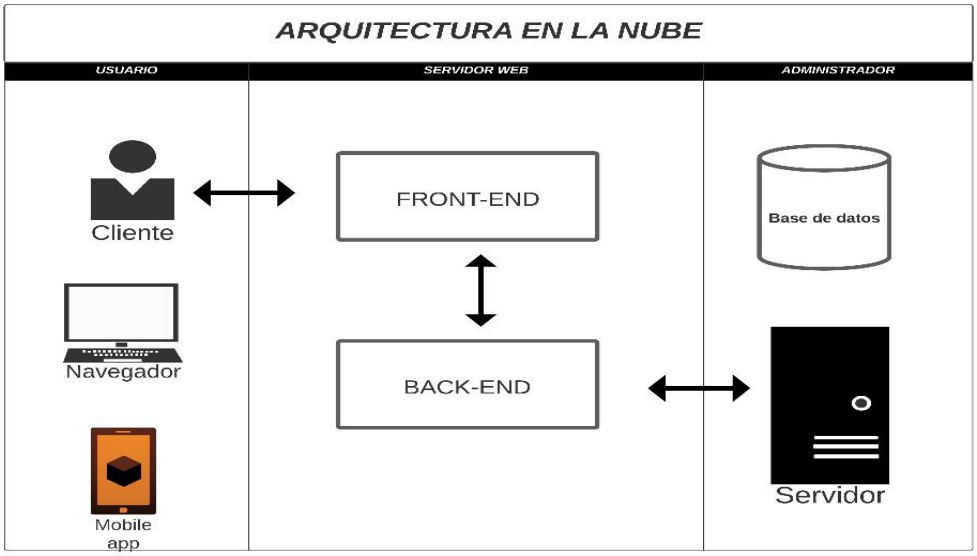

Figura 3. Arquitectura de la Nube
Nota: tomado de Shapsough et al., *(2018)*.

- **Front end**: Dirigida especialmente a la vista que el usuario final se enfoca, constituida por:
    - Interfaz gráfica para usuario
    - Dispositivo o equipo terminal del cliente
- **Back end:** Esta encargada del proveedor del servicio, conformada por lo siguiente:
    - Aplicación
    - Servicio
    - Almacenamiento

- Administración
- Seguridad

3.8 Importancia de la monitorización de consumo de energía

Los sistemas de monitorización del consumo de energía pueden controlar mejor la calidad del servicio y los costos de energía. Con la ayuda de un sistema que permita: controlar, automatizar e interpretar la data y representarlos en gráficas intuitivas es posible determinar si existe un consumo excesivo en el proceso e identificar sobrecargas en grupos de distribución y otros equipos eléctricos (Oviedo, 2017).

En este sentido, mientas más cantidad de información puedan obtener estos equipos de sistema eléctrico, podrá implementar mejores controles, por lo tanto, ayudará a reducir el exceso de energía. En breve se presenta la siguiente figura para conocer la importancia de la monitorización de energía.

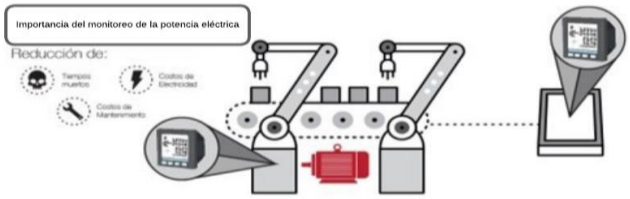

Figura 4. Importancia del control y monitorización de la energía

Nota: tomado de *(Oviedo, 2017).*

3.9 Analizadores de energía

Los analizadores de energía son instrumentos o herramientas que permite almacenar y medir valores de variables de tipo eléctrico, así como se presenta en la siguiente figura 11.

Las variables más comunes para medir son:

- Intensidad
- Potencia reactiva

- Potencia activa
- Factor de potencia
- Frecuencia
- Voltaje
- Energía activa
- Energía reactiva

Figura 5. Analizador de energía
Nota: tomado de (Oviedo, 2017).

3.9.1 Ventajas del analizador de redes

Estos equipos son diseñados exclusivamente para realizar mediciones de voltajes, corrientes, potencia, armónicos, factor de potencia y consumos de energía. En breve se presenta las siguientes ventajas.

- Control exacto de consumo (KWh)
- Detección de requerimientos en la instalación
- Diseño óptimo de filtros pasivos y activos de armónicos
- Análisis de curvas de carga para localizar los puntos de máxima demanda energética
- Ahorro de energía eléctrica
- Evitar fraude en contenedores energéticos
- Prevención de riesgos en la red eléctrica
- Solventar problemas de resonancias, disparos ocasionales, fugas diferenciales, armónicos, desequilibrio en las fases.

4. Materiales y metodología

El modelo en cascada es un enfoque secuencial utilizado en el desarrollo de software y la administración de proyectos. Se distingue por seguir una progresión lineal y estructurada a lo largo de diversas etapas: análisis de requerimientos, diseño del sistema, implementación, verificación y mantenimiento.

Cada etapa debe ser finalizada antes de proceder a la siguiente, evitando en la medida de lo posible la superposición o repetición entre las fases. Este modelo es el más apropiado para proyectos con requisitos claramente definidos que presenten una baja probabilidad de modificación durante el transcurso del proceso de desarrollo Regueros (2020). Ver figura 12 para una presentación detallada del proceso que aplica dicha metodología.

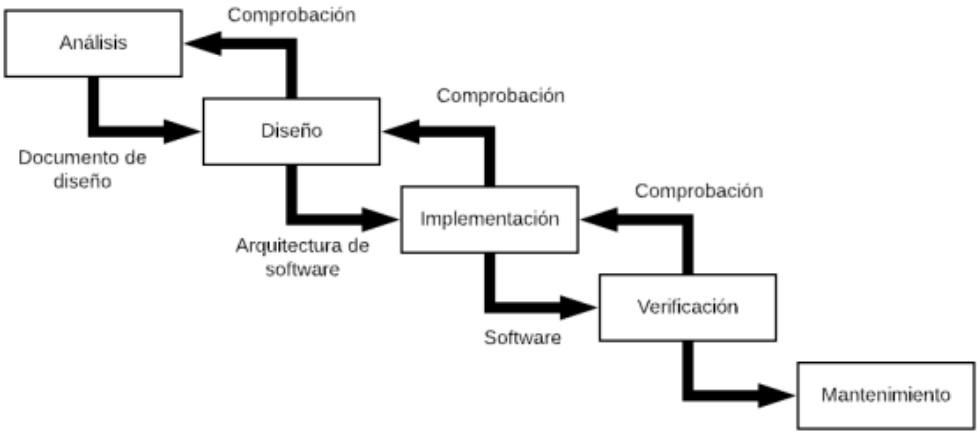

Figura 6. Fases modelo cascada / aplicación GIREI
Nota: tomado de Collina et al., (2014)

De este modo, este modelo resulta apropiado para el desarrollo del presente proyecto de titulación, el cual consta de las siguientes fases:

4.1 Análisis de requisitos

Durante esta etapa, se recopilan las especificaciones y funcionalidades detalladas de dispositivos de MONITORIZACIÓN, como el Venus GX. Asimismo, se analizan teorías pertinentes, como las metodologías de recopilación de datos y los principios de diseño arquitectónico. Posteriormente, se describen de manera precisa los objetivos, alcance y requisitos tanto para el estudio teórico como para el servicio de recopilación de datos.

4.2 Diseño del Sistema

En esta fase, se elabora un modelo arquitectónico integral, fundamentado en la investigación, que comprende los componentes, sus interrelaciones y la estructura general del sistema.

4.3 Implementación

En esta fase se realizó la investigación teórica y se desarrolló el servicio de recolección de datos en base al diseño arquitectónico del modelo c4. Este proceso se dividió en diferentes etapas, comenzando con la recopilación de datos iniciales y luego pasando al diseño e implementación del sistema.

La recopilación comenzó identificando los parámetros eléctricos esenciales para el GIREI, teniendo en cuenta sus características e importancia para el sistema. Simultáneamente se examinaron las fuentes de datos disponibles, evaluando su estructura y viabilidad para incorporarlas a la arquitectura propuesta.

El diseño y la implementación se concentraron en desarrollar elementos esenciales del sistema. Comenzando por la creación de microservicios de recolección, procesamiento y almacenamiento de datos, priorizando el modularidad y la eficiencia.

Se empleó Docker para configurar contenedores, lo que permitió empaquetar e implementar aplicaciones con portabilidad y coherencia en diferentes entornos.

4.4 Verificación

Durante esta fase, se verifica que la investigación teórica esté en consonancia con los requisitos y objetivos documentados. Se llevan a cabo pruebas del servicio de recopilación de datos con el fin de comprobar su funcionalidad.

4.5 Mantenimiento

Se revisa y completa toda la información del proyecto, como la investigación teórica y las especificaciones del Servicio de recopilación de datos, considerando modificaciones y mejoras al diseño arquitectónico.

5. Desarrollo del Proyecto

En la creación de este capítulo, es crucial destacar la importancia del análisis de los datos producidos por el sistema fotovoltaico, así como también la exploración de los dispositivos que posibilitan la transmisión y almacenamiento de dicha información.

La comprensión de estos elementos es fundamental para el diseño de una arquitectura de software sólida para una aplicación web que pueda cumplir con los objetivos establecidos por el GIREI.

5.1 Aplicación web existente VRM

En la actualidad, Victron Energy, fabricante del dispositivo VENUS GX, se encarga del desarrollo y mantenimiento de una aplicación web denominada "VRM" para sus clientes. A través de este dispositivo, se logra la interconexión de todo el sistema fotovoltaico, a continuación, se presenta un exhaustivo análisis de la presente aplicación.

- **monitorización en tiempo real:** Es posible visualizar información en tiempo real acerca del desempeño de los sistemas de energía, tales como el consumo energético, la generación solar, la condición de las baterías, entre otros aspectos.
- **Historial de Datos**: La aplicación registra de manera detallada los datos de rendimiento, lo que permite analizar el funcionamiento del sistema a lo largo de largos períodos de tiempo.
- **Notificaciones y alertas**: Ofrece la posibilidad de establecer alertas personalizadas que informan al usuario a través de correo electrónico o mensaje de texto sobre posibles incidencias, como la disminución de la carga de la batería o la interrupción de la comunicación.
- **monitorización en tiempo real**: Se elaboran informes exhaustivos acerca del desempeño del sistema, los cuales pueden ser descargados o enviados por correo electrónico de forma automática.

- **Control remoto**: La posibilidad de ajustar configuraciones y llevar a cabo acciones a distancia, como activar o desactivar dispositivos, se puede realizar sin la necesidad de estar físicamente presente en el lugar donde se ubica el sistema.
- **Acceso compartido**: Es posible otorgar acceso al sistema a otros usuarios, como técnicos o administradores, lo que facilita la supervisión o gestión simultánea del sistema por varias personas.

A continuación, se presenta un diagrama contextual del modelo C4 que facilita la identificación de los actores principales implicados.

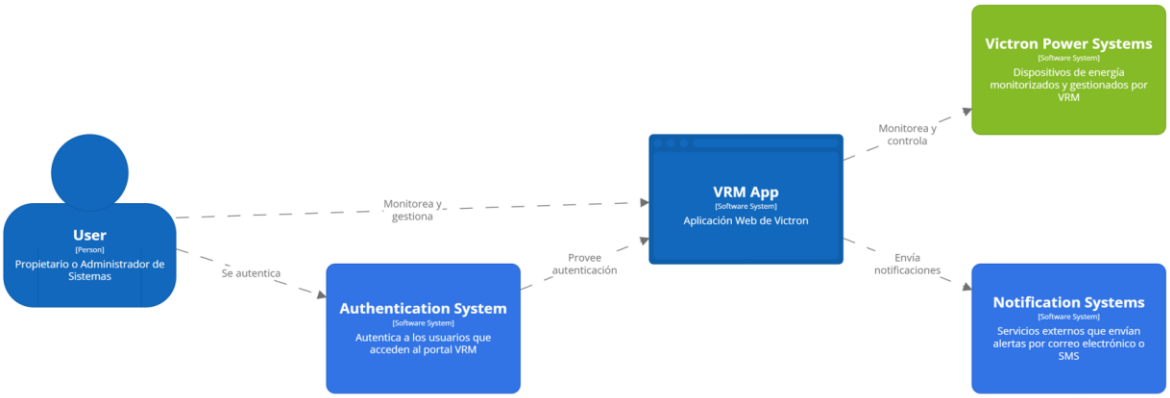

Figura 7. Modelo contextual / aplicación VRM
Nota: elaboración propia

5.2 Análisis de las restricciones del uso de la aplicación web VRM

La comprensión de estos elementos es fundamental para el diseño de una arquitectura de software sólida para una aplicación web que pueda cumplir con los objetivos establecidos por el GIREI.

- **Personalización limitada**: A pesar de que VRM proporciona un conjunto robusto de funcionalidades, se trata de una plataforma estandarizada concebida para cubrir los requerimientos de una amplia audiencia. Las posibilidades de personalización se

encuentran restringidas a las capacidades de la plataforma, lo cual podría no ser suficiente para atender todas las necesidades o preferencias particulares del GIREI y de los usuarios que ellos inviten.

- **Propiedad y acceso a los datos**: Los datos son almacenados y administrados a través de los servicios en la nube proporcionados por Victron. Es factible que el acceso a los datos sin procesar esté restringido y que existan limitaciones en cuanto a la exportación de datos, utilización de APIs y la conexión con otros sistemas.
- **Control de escalabilidad y rendimiento**: La gestión de la escalabilidad está a cargo de Victron. A pesar de contar con una infraestructura robusta, el usuario tiene limitado control sobre el proceso de escalado del sistema y el manejo del rendimiento en situaciones de alta demanda.
- **Dependencia del proveedor**: La confianza en el VRM implica una conexión estrecha con la plataforma y el entorno de Victron. En caso de que Victron modifique sus precios, características o políticas, los recursos disponibles serán limitados.
- **Costo a lo largo del tiempo**: A pesar de que el gasto inicial puede ser reducido al aprovechar una plataforma preexistente, es posible que surjan cargos recurrentes por suscripción o costos adicionales relacionados con un mayor nivel de utilización.
- **Integración con otros sistemas**: La integración con otros sistemas puede estar limitada por las funciones incorporadas de VRM o por su API. Si se requiere una integración específica con otro software o hardware, es probable que surjan dificultades.

El uso de la aplicación web VRM proporciona comodidad y una plataforma multifuncional sin requerir una inversión inicial considerable. A pesar de esto, presenta limitaciones en términos de personalización, gestión de datos y capacidad de expansión. En caso de necesitar funciones particulares, una integración profunda con otros sistemas o un control absoluto sobre la aplicación, la creación de una solución personalizada es la alternativa más adecuada a largo plazo.

5.3 Infraestructura tecnología de una aplicación web

En lugar de emplear VRM (Victron Remote Management), el desarrollo, despliegue y ejecución de una aplicación web personalizada requiere considerar la infraestructura como un elemento crítico. A continuación, se presenta una visión general de los componentes y consideraciones fundamentales de la infraestructura.

- **Alojamiento y Servidores**: Los servicios de computación en la nube, como AWS, Google Cloud y Microsoft Azure, proporcionan una infraestructura que facilita la implementación en diferentes niveles sin la necesidad de gestionar el hardware. Por otro lado, también se presenta la alternativa de utilizar servidores físicos y recibir el soporte informático correspondiente habilitado por la Universidad Politécnica Salesiana.
- **Gestión de bases de datos**: Se pueden distinguir distintos tipos de bases de datos relacionales, como MySQL, PostgreSQL o MariaDB, que son ampliamente utilizadas para la gestión de datos organizados y consultas complejas. Por otro lado, opciones como MongoDB, Cassandra, Firebase e InfluxDB son apropiadas para el tratamiento de datos no estructurados o cuando se necesita flexibilidad en el almacenamiento de la información. Es fundamental considerar la relevancia de la replicación y las copias de seguridad para asegurar una alta disponibilidad y la capacidad de recuperación ante posibles contingencias.
- **Balanceadores de carga y red de distribución de contenido**: La mayoría de los proveedores de servicios en la nube garantizan la distribución del tráfico entre servidores mediante la implementación de balanceadores de carga, con el objetivo de prevenir la sobrecarga de algún servicio. En el ámbito de la selección de un alojamiento local, es necesario configurar un balanceador de carga. Es relevante señalar que el empleo de redes de distribución de contenido (CDN), resulta beneficioso al proporcionar contenido estático, como imágenes, CSS y JS, en proximidad a los usuarios.

- **Medidas de seguridad:** Para regular el acceso de los usuarios a distintas secciones de la aplicación, es fundamental incorporar sistemas robustos de autenticación y autorización, como por ejemplo OAuth y JWT.
- **monitorización y registro:** Es de suma importancia emplear herramientas de monitorización como Prometheus y Kibana para llevar a cabo el seguimiento del desempeño de la infraestructura, abarcando aspectos como la carga del servidor, el rendimiento de la base de datos y el estado de la aplicación.
- **Escalabilidad y redundancia:** Es fundamental el diseño de una aplicación que permita el escalado horizontal, lo cual implica la capacidad de añadir servidores adicionales para gestionar cargas más pesadas en lugar de realizar mejoras en un único servidor. Así mismo, asegurar la redundancia en la infraestructura, mediante la implementación de múltiples instancias de servidor y bases de datos, es esencial para prevenir puntos únicos de fallo y garantizar la alta disponibilidad de la aplicación.
- **Devops:** El enfoque DevOps se ha vuelto esencial para el desarrollo de aplicaciones, ya que permite automatizar la implementación, asegurar la eficiencia en las pruebas y la implementación de actualizaciones y nuevas funcionalidades a través de pipelines y herramientas de control de versiones de código y documentación de software. Esto facilita un flujo de trabajo efectivo en los equipos de desarrollo de software.

5.4 Diseño arquitectónico de la aplicación web

El desarrollo de una aplicación web propia conlleva diversas ventajas significativas, especialmente en términos de personalización, control y valor a largo plazo. A pesar de que la inversión inicial puede ser más elevada en comparación con la utilización de una aplicación provista por un proveedor, la posibilidad de adaptar la aplicación a las necesidades específicas, mantener un control total sobre los datos y la seguridad, así como evitar la dependencia de un proveedor, puede generar beneficios sustanciales a largo plazo.

Por consiguiente, es recomendable iniciar el proceso con el diseño de la arquitectura antes de proceder con la codificación, lo cual garantiza que la aplicación se construya sobre bases sólidas Bas et al., (2013).

5.4.1 Solución arquitectónica

Tras un minucioso análisis de la aplicación VRM (Victron Remote Management) y sus componentes asociados, se ha alcanzado una conclusión esencial acerca de la estructura de la aplicación web diseñada para supervisar y administrar la energía.

La aplicación VRM ha sido fundamental como punto de referencia, proporcionando información sobre cómo un sistema robusto y puede manejar las complejidades de la recolección, el almacenamiento, el procesamiento y la visualización de datos energéticos. No obstante, en el proceso de buscar una solución que satisfaga tanto nuestras necesidades presentes como las futuras, se ha optado por la implementación de una arquitectura de microservicios.

La siguiente sección se detalla minuciosamente cada uno de los componentes que la conforman a través del uso del modelo C4 para posteriormente presentar un diagrama arquitectónico de alto nivel.

5.4.2 Modelo C4

Este es un instrumento que facilita la documentación de la arquitectura de software de manera clara y sencilla. El modelo C4, que comprende Contexto, Contenedores, Componentes y Código, se utiliza para representar la arquitectura de los sistemas de software. Ofrece un enfoque jerárquico para la descripción de la arquitectura, lo cual favorece la comunicación y comprensión de sistemas complejos (Velasco, 2024).

5.4.2.1 Diagrama de contexto

El diagrama de contexto ofrece una perspectiva general del sistema al enfocarse en sus límites y las interacciones principales con elementos externos, sin profundizar en el funcionamiento interno (Velasco, 2024).

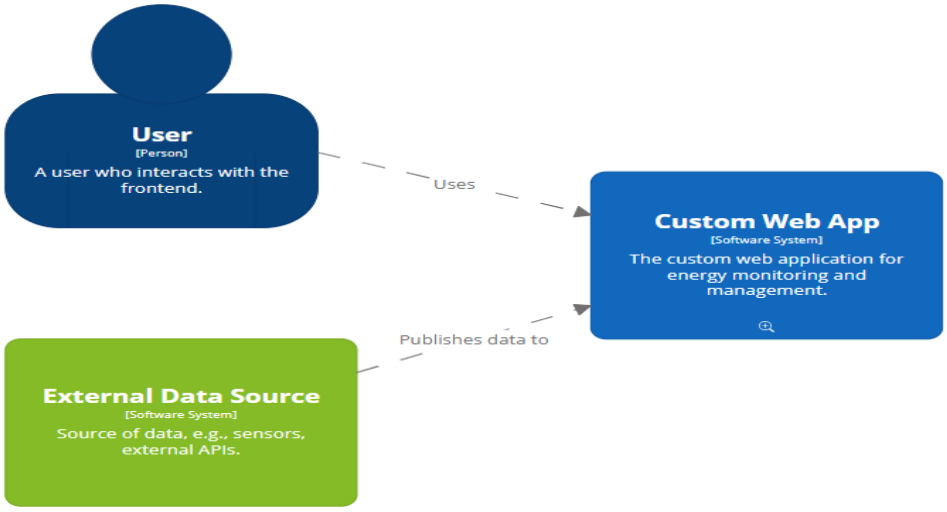

Figura 8. Diagrama de contexto
Nota: elaboración propia

El diagrama anterior presenta la estructura general de la aplicación para supervisar y administrar el consumo energético. A continuación, se proporciona una concisa explicación de los componentes que lo conforman.

- **User**: Participantes del Grupo de Investigación en Energía Renovable e Invitados.
- **External Data Source**: Hace referencia a la información que proviene de fuentes ajenas, como el Venus GX, la cual se integra en el sistema mediante el uso del protocolo Modbus.
- **Custom Web App**: Proporciona la interfaz de usuario necesaria para la gestión y supervisión de los datos relacionados con el consumo de energía.

5.4.2.2 Diagrama de contenedor

En el modelo C4, un diagrama de contenedores es un diagrama de alto nivel que muestra los componentes de software principales que conforman el sistema y cómo interactúan. Estos "contenidos" pueden incluir aplicaciones web, bases de datos, microservicios u otras unidades ejecutables. El diagrama muestra cómo estos contenedores se comunican entre sí y con sistemas externos (Velasco, 2024).

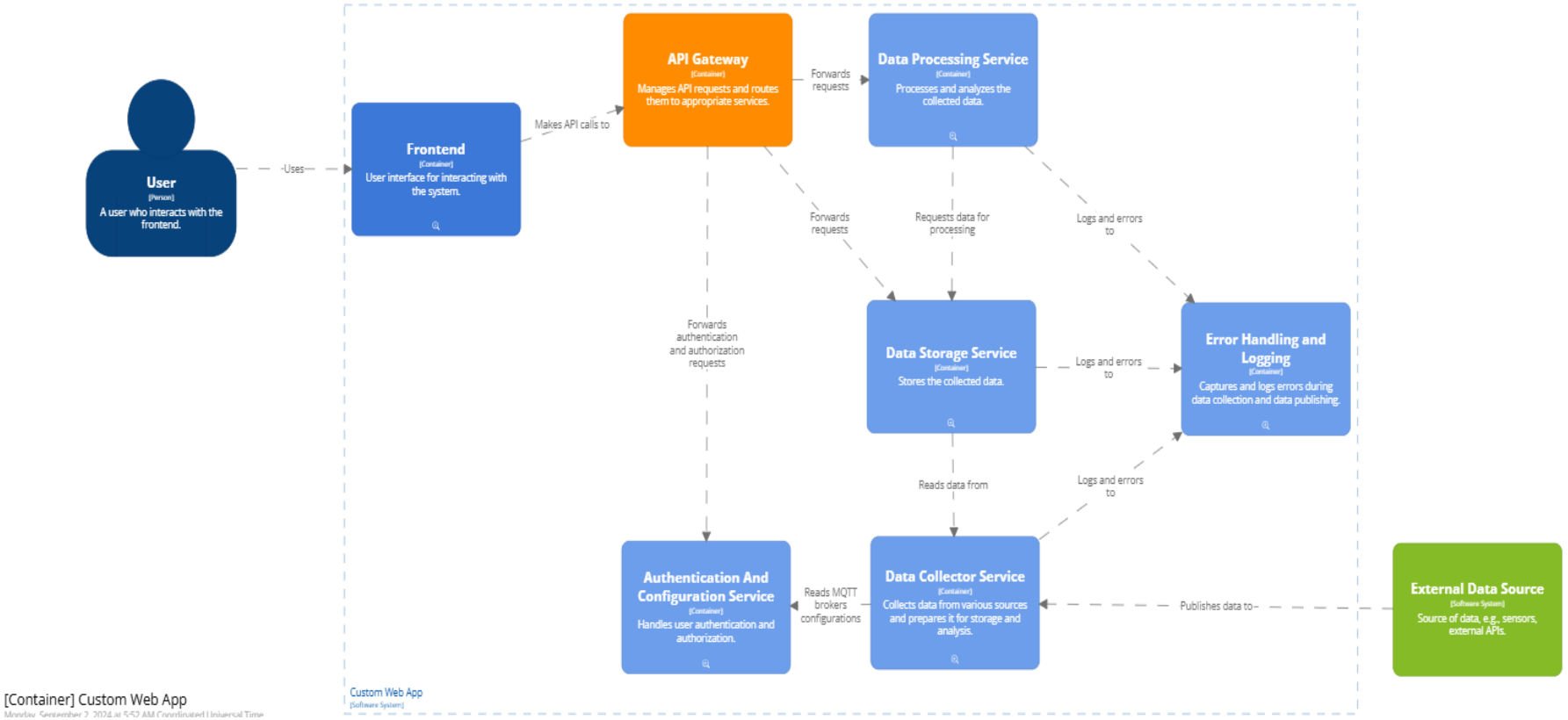

Figura 9. Diagrama de contenedor
Nota: elaboración propia

En el diagrama previamente expuesto se puede observar una visión más amplia de la solución, que incluye tanto los componentes externos como internos de la aplicación web. A continuación, se describen de forma concisa los elementos internos:

- **Frontend**: Se encarga de proveer la interfaz de usuario que permite la interacción con el sistema, presentando información en tiempo real, dashboards, informes y notificaciones.
- **API Gateway**: Encamina las solicitudes, implementa la seguridad, gestiona las restricciones de velocidad, equilibra las cargas y documenta la actividad de la API para su supervisión.
- **Authetication And Configuration Service**: Se encarga de la gestión de la autenticación y autorización de usuarios, así como del proceso de inicio de sesión y la configuración de los usuarios, lo cual abarca también las configuraciones destinadas al Data Collector Service.
- **Data Collector Service:** Es un elemento esencial que se encarga de recopilar información de diversas fuentes externas, como por ejemplo Venux GX. La principal tarea consiste en recopilar y preprocesar los datos previamente a su envío al Data Storage Service.
- **Data Storage Service:** Se encarga de preservar la información recopilada a largo plazo, asegurando que los datos se guarden de manera segura y puedan ser recuperados sin dificultad para respaldar el Data Processing Service y otras posibles integraciones futuras.
- **Data Processing Service:** Es responsable de analizar y transformar los datos recopilados para extraer información significativa y prepararlos para informes o usos posteriores.
- **Error Handling and Logging:** Es responsable de supervisar, capturar y gestionar errores en todo el sistema. Se asegura de que los problemas se registren, rastreen y resuelvan para mantener la seguridad del sistema y brindar información sobre los problemas operativos.

5.4.2.3 Diagrama de componentes

El diagrama de componentes es una herramienta que facilita la comprensión de la organización interna de un contenedor, describiendo la interacción y colaboración entre sus diversas partes para lograr la funcionalidad del sistema (Velasco, 2024).

- **Frontend**

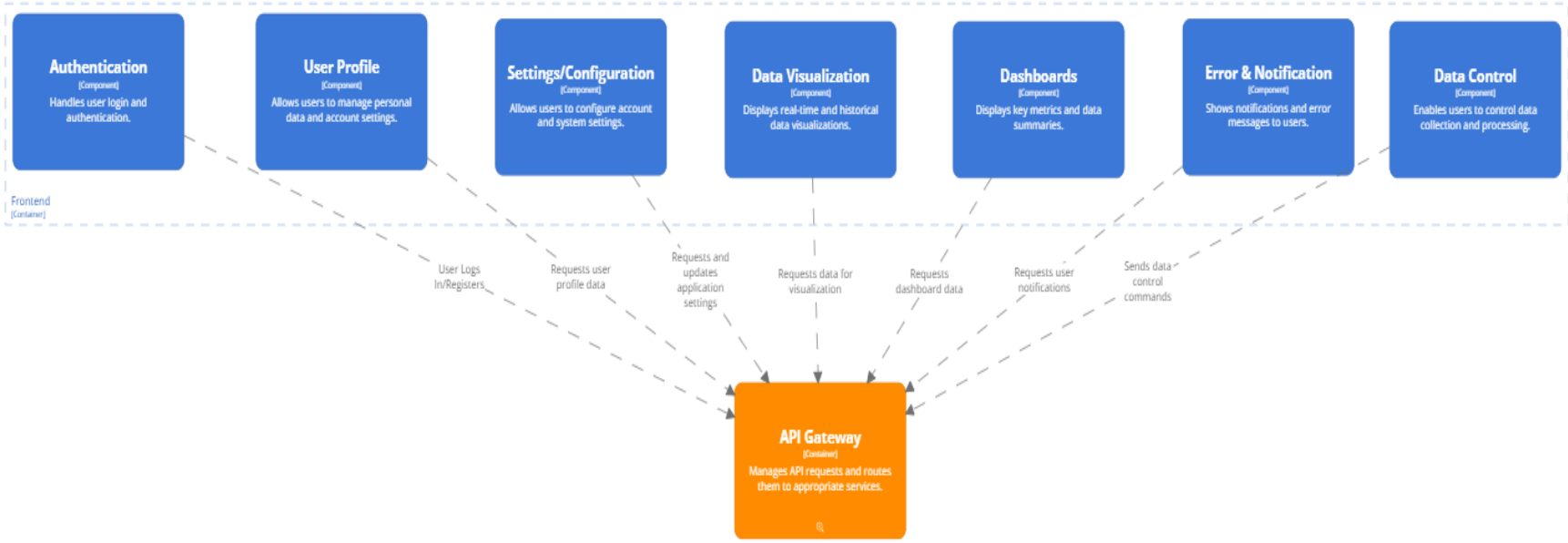

Figura 10. Diagrama de componentes / Frontend
Nota: elaboración propia

El Frontend es el conjunto de elementos de una aplicación web que posibilita la interacción entre las diversas funcionalidades del sistema y los usuarios. Este componente está conformado por los siguientes elementos:

- **Authentication**

Este elemento gestiona el inicio de sesión y la autenticación de los usuarios. El objetivo de este componente es gestionar las credenciales de usuario y la administración de sesiones con el fin de asegurar un acceso seguro a la aplicación. Incorpora formularios para iniciar sesión, verificaciones de autenticación y funcionalidades para el registro de usuarios.

- **Settings/Configuration**

Este elemento proporciona a los usuarios la oportunidad de personalizar su experiencia en la aplicación. Los usuarios tienen la facultad de alterar preferencias tales como la configuración de notificaciones (correo electrónico o notificaciones push), temas de la interfaz de usuario, selecciones de idioma y opciones de privacidad de la cuenta. Además, facilita a los usuarios el almacenamiento de estas configuraciones de manera persistente, asegurando así una experiencia personalizada cada vez que inician sesión.

- **User Profile**

Este componente proporciona a los usuarios la capacidad de visualizar y gestionar su información personal, incluyendo los detalles de su perfil, tales como su nombre, dirección de correo electrónico, fotografía de perfil e información de comunicación. Facilita la actualización de sus credenciales y preferencias, preservando la seguridad y la integridad de la información. Los usuarios tienen la posibilidad de examinar la actividad o el historial de la cuenta, en caso de que sea pertinente, para realizar un seguimiento de sus interacciones con el sistema.

- **Data Visualization**

Este componente tiene como objetivo exponer datos complejos de una forma comprensible y visualmente atractiva. Emplea diversas técnicas de visualización, tales como diagramas de líneas, diagramas de barras, diagramas circulares y paneles de control, con el fin de presentar datos históricos y en tiempo real. Los usuarios tienen la capacidad de interactuar con las visualizaciones, tales como filtrar datos o ajustar el zoom en intervalos de tiempo específicos, con el fin de obtener información más exhaustiva sobre tendencias y patrones.

- **Dashboards**

Este componente muestra métricas clave y resúmenes de datos proporcionando a los usuarios una descripción general de los datos importantes y los indicadores de rendimiento en un formato visualmente atractivo. Permite obtener información rápida y tomar decisiones basadas en datos.

- **Error & Notification**

Este componente resulta fundamental para optimizar la experiencia del usuario, dado que suministra información relevante acerca del estado del sistema y las interacciones del usuario. Cuando las operaciones no se completan con éxito, el sistema muestra mensajes de error con el fin de asistir a los usuarios en la identificación de la causa del problema y en la recomendación de posibles soluciones. En este lugar también se gestionan las notificaciones de acciones exitosas, como el almacenamiento de datos, y las alertas de problemas críticos. Esto asegura que los usuarios estén al tanto de eventos relevantes en la aplicación.

- **Data Control**

Proporciona a los usuarios la capacidad de gestionar la recopilación y el procesamiento de datos. Suministra una herramienta que les permite administrar las fuentes de datos, iniciar

o cesar la recopilación de datos y configurar los parámetros de procesamiento de datos. Este mecanismo de control es esencial para los usuarios que desean personalizar su interacción con los servicios de recopilación de datos para asegurarse de que reciban solo la información relevante según sus preferencias. A continuación, se presenta un listado de tecnologías recomendadas.

- **Tech Stack**

1. **Framework: Angular.** Una estructura robusta fundamentada en componentes para la creación de aplicaciones web dinámicas. Proporciona la posibilidad de enlace de datos bidireccional e inyección de dependencia.
2. **Gestión de estados: NgRx.** Una biblioteca destinada a administrar el estado global en aplicaciones Angular mediante el uso de principios Redux. Promueve la administración del estado predecible mediante la implementación de acciones, reductores y selectores.
3. **Manejo de formularios: Angular Reactive Forms.** La implementación de un enfoque basado en modelos en la gestión de formularios facilita la validación de estos, la personalización de los controles y la aplicación de patrones reactivos para el manejo de su estado.
4. **Visualización de datos: Chart.js.** Una biblioteca simplificada para la elaboración de gráficos interactivos que incluyen diversos tipos de gráficos, tales como barras, líneas, circulares, entre otros.
5. **Intercambio de datos: HttpClientModule.** Se trata de un servicio integrado de Angular para la ejecución de peticiones HTTP. Admite funciones tales como observables para las operaciones asincrónicas, interceptores para la gestión de solicitudes y respuestas, y administración de errores.

5.4.2.4 API Gateway

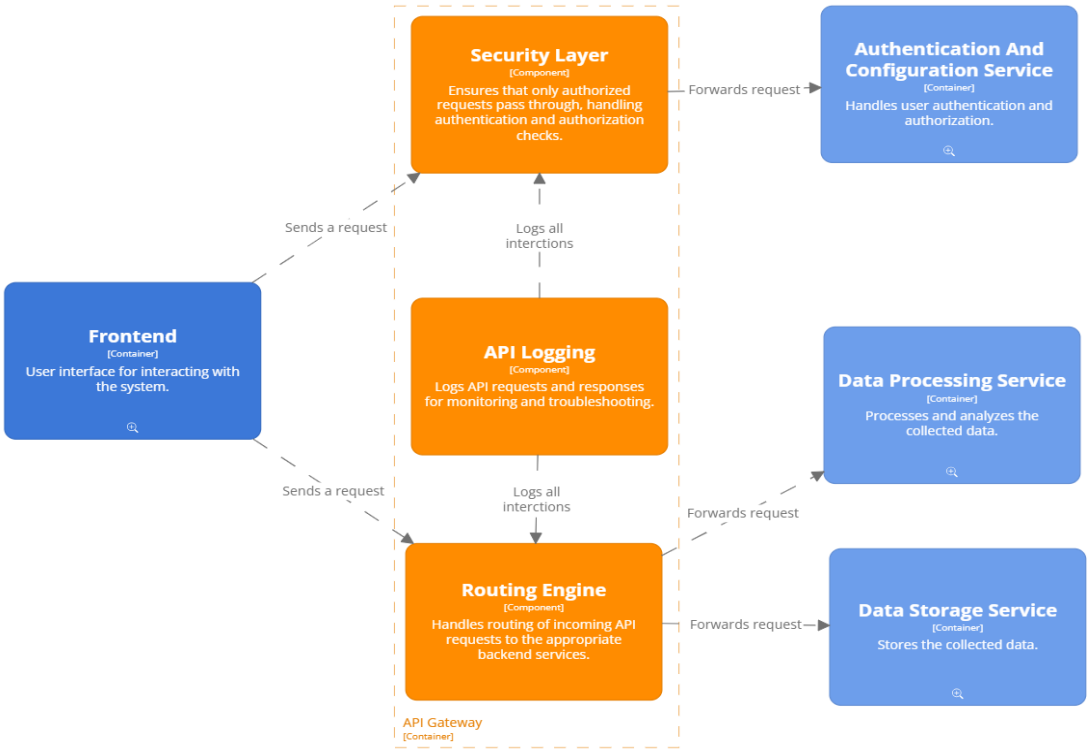

Figura 11. Diagrama de componentes / API Gateway
Nota: elaboración propia

El API Gateway es responsable de funciones fundamentales como el direccionamiento de peticiones, la distribución equitativa de la carga, la autenticación y el control de la velocidad. Asimismo, ofrece un punto singular para el registro, monitorización y protección del tráfico de Interfaz de Programación de Aplicaciones (API) (Reynés, 2023).

- **Routing Engine**

Este componente tiene la responsabilidad de canalizar las solicitudes API entrantes hacia los correspondientes servicios backend. Mediante la implementación de mecanismos de enrutamiento inteligentes, se optimiza el proceso de gestión de solicitudes, asegurando que los usuarios obtengan respuestas en el momento oportuno. El motor de enrutamiento optimiza la arquitectura en el lado del cliente, permitiendo a los desarrolladores enfocarse

en la creación de interfaces de uso sencillo sin preocuparse por las complejidades del servicio backend.

- **Security Layer**

La seguridad es de vital importancia en toda aplicación y la capa de seguridad implementa rigurosos protocolos de autenticación y autorización. Confirma la autenticidad de las peticiones entrantes y asegura que únicamente los usuarios autorizados tengan acceso a los datos de carácter confidencial. Este componente juega un papel crucial en la salvaguarda de la información de los usuarios y en el mantenimiento de la integridad del sistema.

- **API Logging**

Este elemento se encarga de documentar cada petición de API que llega y cada respuesta que sale, recopilando información que facilita la supervisión, el estudio y la solución de posibles inconvenientes. Esta capacidad de registro es esencial para identificar posibles problemas, optimizar el rendimiento y mejorar la experiencia general del usuario.

Para crear la API Gateway, es recomendable usar tecnologías que garanticen un acceso seguro, un enrutamiento y un registro completo. A continuación, una lista de ellas.

- **Tech Stack**

1. **.Net Core.** Un marco de alto rendimiento multiplataforma para la creación de aplicaciones modernas basadas en la nube.
2. **Ocelot:** Un marco de API Gateway diseñado específicamente para aplicaciones .NET Core. Permite el enrutamiento, la agregación de solicitudes y la autenticación, lo que agiliza la comunicación entre los servicios frontend y backend.
3. **Docker:** Contiene la aplicación API Gateway para una implementación uniforme y consistente en diversos entornos, lo que simplifica la escalabilidad y el mantenimiento.

5.4.2.5 Authentication & Configuration

Figura 12. Diagrama de componentes / Authetication & Configuration
Nota: elaboración propia

El Servicio de autenticación y configuración desempeña un papel centralizado en la gestión de la autenticación de usuarios y la configuración en el sistema en su totalidad, se asegura de que únicamente los usuarios autorizados puedan acceder a las funciones y configuraciones del sistema, al mismo tiempo que proporciona una experiencia de usuario fluida. Este servicio se encarga de la autenticación de usuarios, la administración de sesiones y los niveles de autorización, lo cual posibilita la implementación de procesos de inicio de sesión seguros y el control de acceso. A continuación, se describen sus elementos.

- **REST API**

Representa el punto de acceso fundamental a través del cual los servicios y los usuarios se relacionan con la totalidad del sistema. Su función abarca la gestión de las solicitudes de autenticación de los usuarios, tales como el inicio de sesión, cierre de sesión, generación de tokens y validación. Asimismo, se encarga de gestionar los ajustes de configuración de la conexión con fuentes de datos para conexiones Modbus.

- **Repository**

El repositorio es el encargado de la conservación de toda la información vinculada a la autenticación y las configuraciones. Se guardan las credenciales de los usuarios, junto con sus roles, permisos y tokens, además de la información de configuración de la conexión Modbus. El repositorio asegura el almacenamiento, la recuperación y la actualización de los datos de manera eficiente.

Ambos elementos desempeñan un papel específico en la gestión segura y de la autenticación, la autorización y la configuración tanto del usuario como del recolector de datos.

- **Tech Stack**

1. **ASP.NET Core.** Un marco de trabajo multiplataforma para la creación de aplicaciones web e interfaces de programación de aplicaciones.
2. **PostgreSQL.** Es una base de datos relacional de código abierto que destaca por su potencia en el almacenamiento de datos y configuraciones de usuarios.
3. **JSON Web Tokens (JWT).** Se trata de un medio compacto y seguro para representar reclamos que serán transmitidos entre dos partes. Este se emplea para la autenticación segura de los usuarios.

5.4.2.6 Data Collector Service

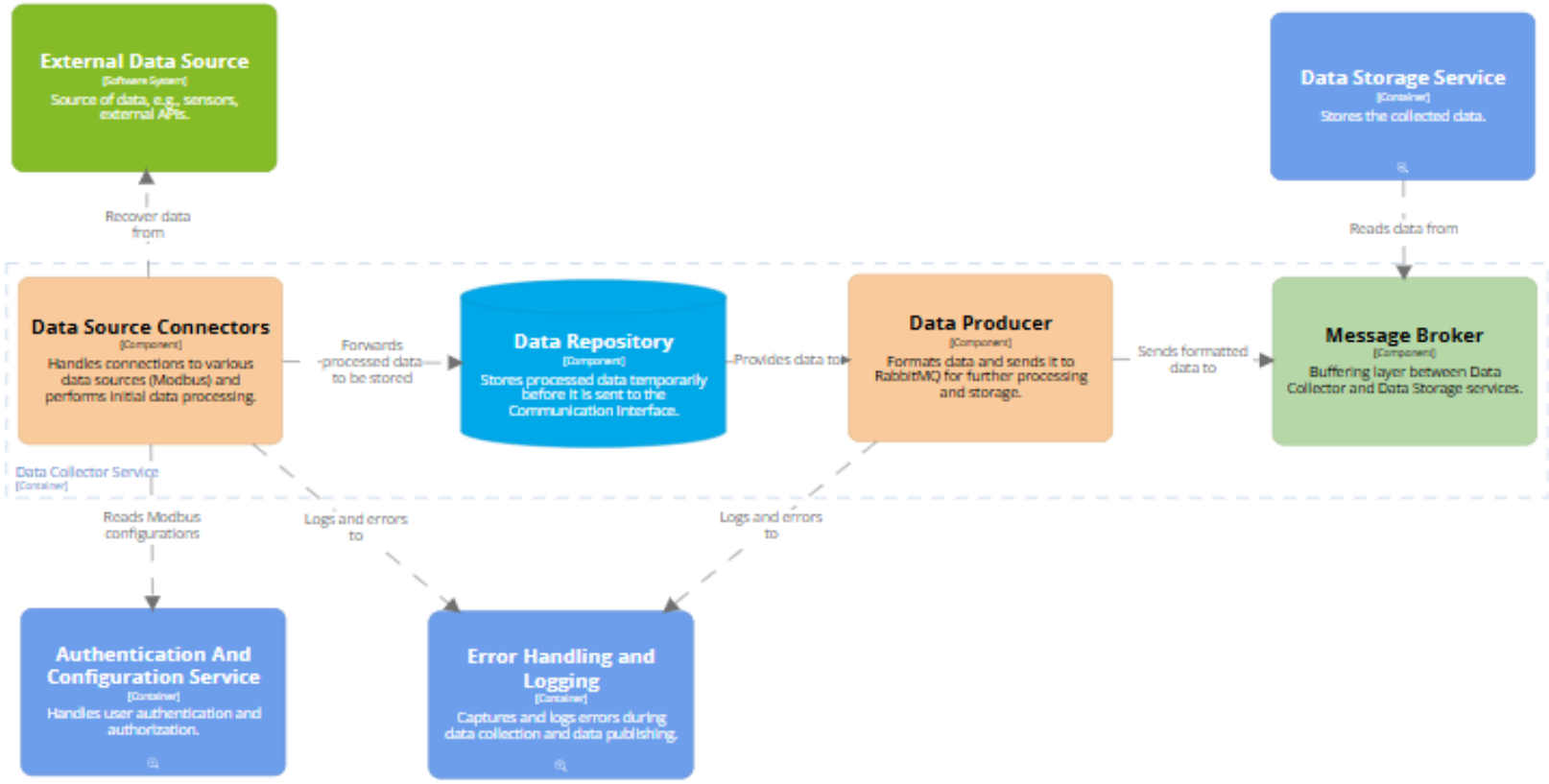

Figura 13. Diagrama de componentes / Data Collector Service
Nota: elaboración propia

El servicio de recopilación de datos tiene la responsabilidad de recolectar información proveniente de diversas fuentes externas, tales como dispositivos de Internet de las Cosas (IoT), sensores y otras interfaces de programación de aplicaciones (API) de proveedores externos. Su función consiste en servir como intermediario entre los proveedores de datos externos y el sistema interno, asegurando la adecuada formación, validación y enrutamiento de los datos recibidos para su posterior procesamiento o almacenamiento. A continuación, sus componentes:

- **Data Source Connector**

Este componente es responsable de gestionar las conexiones a varias fuentes de datos (Modbus). Cumple la función de ser el punto de procesamiento inicial de los datos recopilados, llevando a cabo las transformaciones o validaciones requeridas previas a la transferencia de los datos a otros componentes. Esto asegura que los datos provenientes de diversas fuentes se encuentren estandarizados y preparados para su posterior análisis y almacenamiento.

- **Data Repository**

El repositorio de datos funciona como un sitio de almacenamiento provisional para los datos que han sido procesados previamente, antes de su transferencia a otras componentes del sistema. Garantiza que los datos recopilados de varias fuentes se almacenen de manera segura mientras se espera su procesamiento o reenvío. Actúa como un búfer para gestionar las fluctuaciones en el flujo de datos y garantiza que la transmisión de datos se lleve a cabo de manera uniforme en todo el sistema.

- **Data Producer**

Tiene la responsabilidad de formar y empaquetar la información recolectada del repositorio de datos, y de enviarla a servicios externos, como Azure Service Bus o a un broker de mensajería como RabbitMQ, para su posterior procesamiento y almacenamiento. Este

componente asegura que los datos estén correctamente estructurados y se enruten, lo cual facilita una integración óptima con los servicios subsecuentes.

- **Message Broker**

Actúa como capa intermedia entre el recopilador de datos y los servicios de almacenamiento de datos. Mediante el uso de un bróker de mensajería, se simplifica la asignación de mensajes confiables, asegurando que los datos del publicador de datos sean almacenados en el búfer y se transmitan a los componentes de almacenamiento apropiados. Esta disociación de servicios facilita una escalabilidad optimizada, una tolerancia a fallas y un procesamiento asincrónico.

- **Tech Stack**

Las tecnologías siguientes serán empleadas para el servicio de recopilación de datos:

1. **.Net Core.** Desarrollado con la finalidad de ser compatible con diversas plataformas de ejecución, tales como Windows, Linux y macOS. Esto lo hace sumamente versátil para su implementación, especialmente cuando se lleva a cabo en entornos de nube como Azure, donde se pueden ejecutar diversos servicios en contenedores en distintos sistemas operativos.
2. **Redis.** Redis es un sistema de almacenamiento en memoria de alto rendimiento que se caracteriza por su estructura de clave-valor. Comúnmente empleado para funciones de almacenamiento temporal y caché.
3. **RabbitMQ.** Un servicio de mensajería de código abierto, distribuido y que garantizará una comunicación fiable entre el servicio de recopilación de datos y otros componentes, asegurando así que los datos sean colocados en cola y transferidos de manera adecuada.

5.4.2.7 Data Storage Service

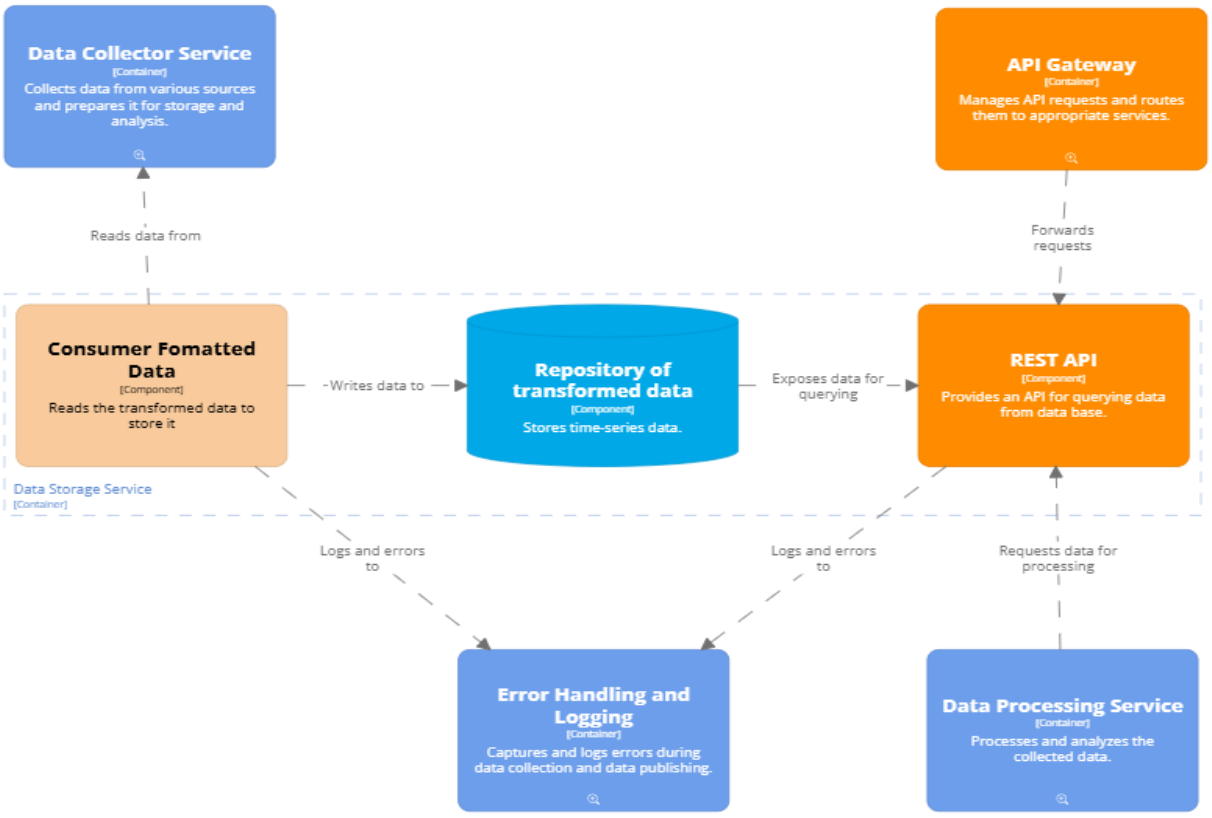

Figura 20. Diagrama de componentes / Data Storage Service
Nota: elaboración propia

El Servicio de Almacenamiento de Datos tiene la responsabilidad de conservar y administrar la información recopilada y tratada. Está concebido para gestionar grandes volúmenes de datos de series temporales, asegurando un almacenamiento, una recuperación y una administración eficientes de la información. A continuación, se detalla sus componentes:

- **Consumer Formatted Data**

Este componente se encarga de asegurar que los datos modificados cumplan con el formato correcto y sean almacenados en la base de datos. Este componente está diseñado para salvar la brecha entre los procesos de transformación de datos y la capa de almacenamiento de datos, facilitando así una gestión y recuperación de datos.

- **Repository of Transformed Data**

Tiene la función de almacenar datos de series temporales, los cuales han sido procesados y transformados, estando así listos para su análisis posterior o para la elaboración de informes. Su papel es almacenar grandes cantidades de datos indexados cronológicamente, lo que facilita la consulta y recuperación de información de manera eficiente.

- **REST API**

Este componente ofrece una interfaz para la consulta de la información almacenada, sirve como la interfaz principal para interactuar con el servicio de almacenaje de datos, lo que permite a los clientes y otros servicios consultar y modificar la información contenida en la base de datos. Esta componente sigue los principios RESTful, lo que proporciona una forma de acceder a los recursos.

- **Tech Stack**

Para el desarrollo de este servicio se sugiere el uso de las siguientes tecnologías:

1. **InfluxDB.** Se trata de una base de datos especializada en series temporales, diseñada para ofrecer un alto rendimiento en el almacenamiento de extensas cantidades de información, lo cual la convierte en una opción idónea para entornos de supervisión y análisis.
2. **ASP.NET Core.** Un entorno de desarrollo multiplataforma que permite la creación de aplicaciones web y API.
3. **Docker.** Una plataforma para desarrollar, entregar y ejecutar aplicaciones en contenedores.
4. **Swagger/OpenAPI.** Es un marco para la documentación de API que le permite describir los servicios RESTful, lo cual simplifica la interacción entre desarrolladores y clientes con el servicio de almacenamiento de datos.

5.4.2.8 Data Processing Service

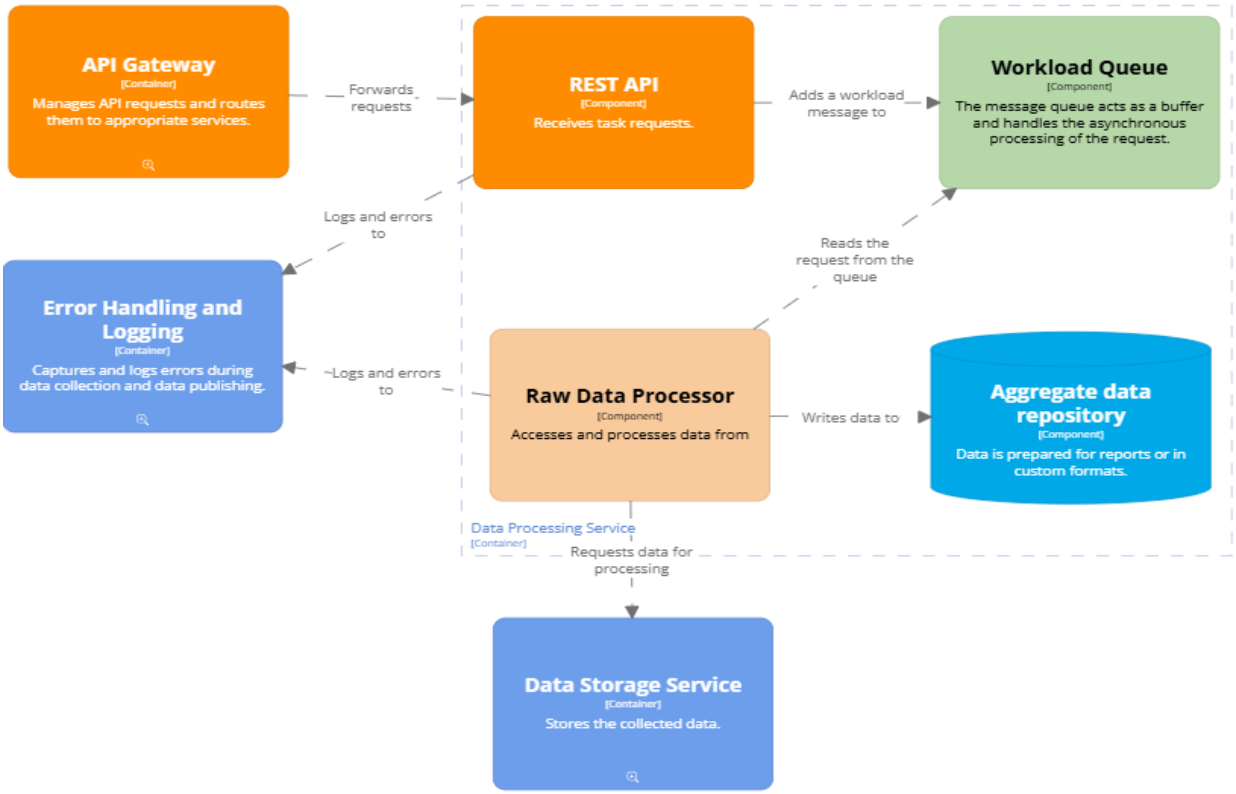

Figura 21. Diagrama de componentes / Data Processing Service
Nota: elaboración propia

Este servicio se encarga de procesar y analizar la información recolectada. Se encarga de actividades como la recopilación de datos para la creación de informes y la conversión de datos en bruto en formatos adecuados para su posterior uso por parte de los servicios o usuarios finales.

- **Raw Data Processor**

Este componente se encarga de acceder a los datos sin procesar obtenidos del sistema y llevar a cabo su procesamiento. Se llevan a cabo las modificaciones pertinentes con el fin de asegurar que los datos se encuentren organizados y preparados para su posterior utilización o almacenamiento.

- **Aggregate Data Repository**

Este elemento se encarga de la preparación y almacenamiento de datos agregados, los cuales son posteriormente utilizados para la generación de informes o la creación de formatos personalizados de acuerdo con los requerimientos del sistema. Funciona como un depósito de datos procesados que son necesarios para la inteligencia empresarial o el análisis.

- **Workload Queue**

Este componente es una cola de mensajes que actúa como un búfer para procesar solicitudes de forma asincrónica. Gestiona la distribución de las tareas de procesamiento, lo que permite equilibrar la carga y garantizar que las tareas se completen de manera oportuna sin sobrecargar el sistema.

- **REST API**

Recibe peticiones de tareas de otros servicios o usuarios permitiendo la gestión del procesamiento de datos entrantes y las dirige a los componentes de procesamiento adecuados.

- **Tech Stack**

Las tecnologías empleadas para desarrollar este servicio son las mismas que se utilizaron en los servicios previos, a saber: Swagger/OpenAPI, Docker, ASP.NET Core, Net Core. Y a continuación, se presentan algunas adicionales.

1. **PostgreSQL.** Una base de datos relacional open-source conocida por su robustes, eficacia y soporte para tipos de datos avanzados.
2. **Azure Queue Storage.** Un servicio de mensajería en la nube que facilita la comunicación asincrónica entre distintos componentes. Ofrece una manera fiable y sencilla de encolar tareas para su posterior procesamiento.

5.4.2.9 Error Handling & Logging

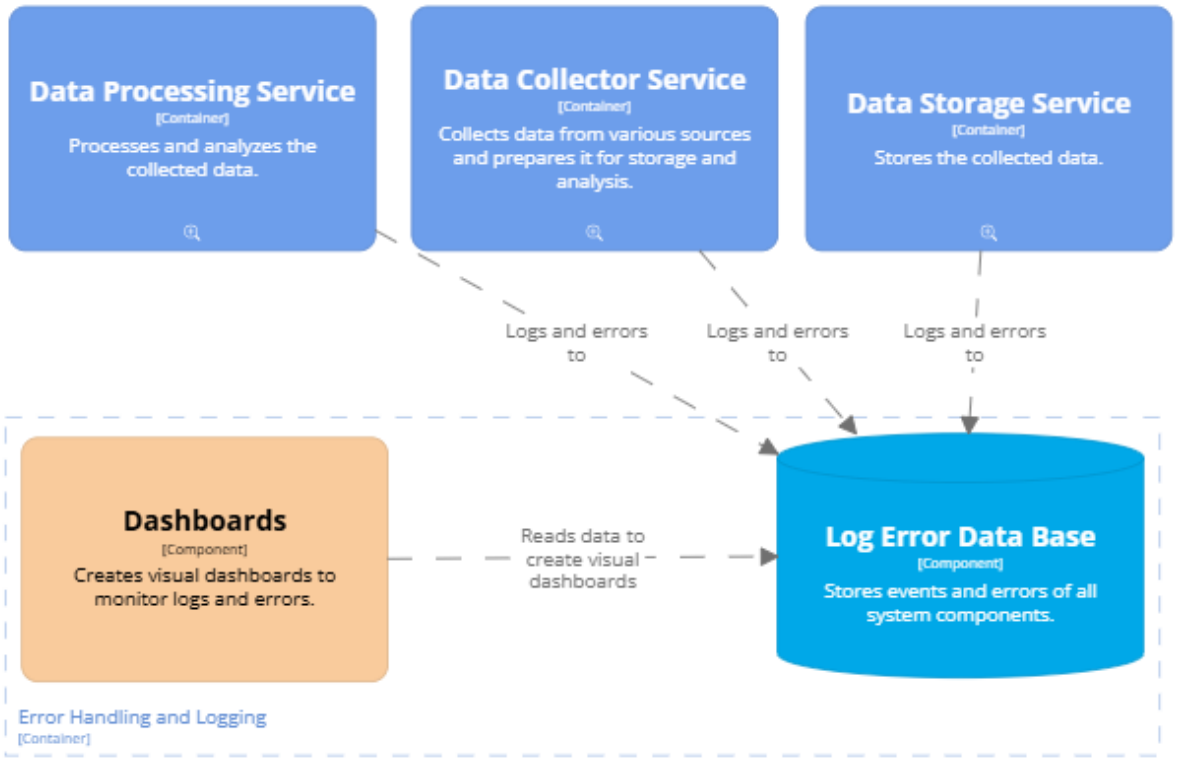

Figura 22. Diagrama de componentes / Error Handling & Login
Nota: elaboración propia

Durante los procesos de recopilación y publicación de datos, este servicio se encarga de capturar, registrar y administrar los errores. El sistema monitorea todos los componentes con el fin de detectar eventos de error del sistema, a continuación, se detalla sus elementos.

- **Log Error Database**

Este componente tiene la responsabilidad de almacenar los registros de todos los componentes, lo cual incluye los eventos de error y otros eventos del sistema. Cumple la función de ser un repositorio centralizado que permite el seguimiento de errores y facilita el análisis de futuras incidencias.

- **Dashboards**

Este componente genera representaciones visuales de los eventos ocurridos y de posibles errores que hayan surgido en los diversos servicios. Las capacidades de monitorización en tiempo real proporcionan a los administradores de sistemas la posibilidad de supervisar el estado de los sistemas, identificar problemas y observar tendencias.

- **Tech Stack**

Las tecnologías empleadas para desarrollar este servicio son las mismas que se utilizaron en los servicios previos, a saber: Swagger/OpenAPI, Docker, ASP.NET Core, Net Core. Y a continuación, se presentan algunas adicionales.

1. **Elasticsearch.** Un motor de búsqueda y análisis distribuido para gestionar grandes volúmenes de datos de registros y facilita la realización de búsquedas y consultas de manera rápida.
2. **Kibana.** Una herramienta de visualización que opera con Elasticsearch y ofrece una interfaz amigable para la consulta y representación de datos.
3. **Serilog.** Una biblioteca de registro estructurada para aplicaciones .NET. Permite el registro estructurado de eventos, lo que facilita su consulta y análisis en Elasticsearch.

5.4.2.10 Diagrama de alto nivel

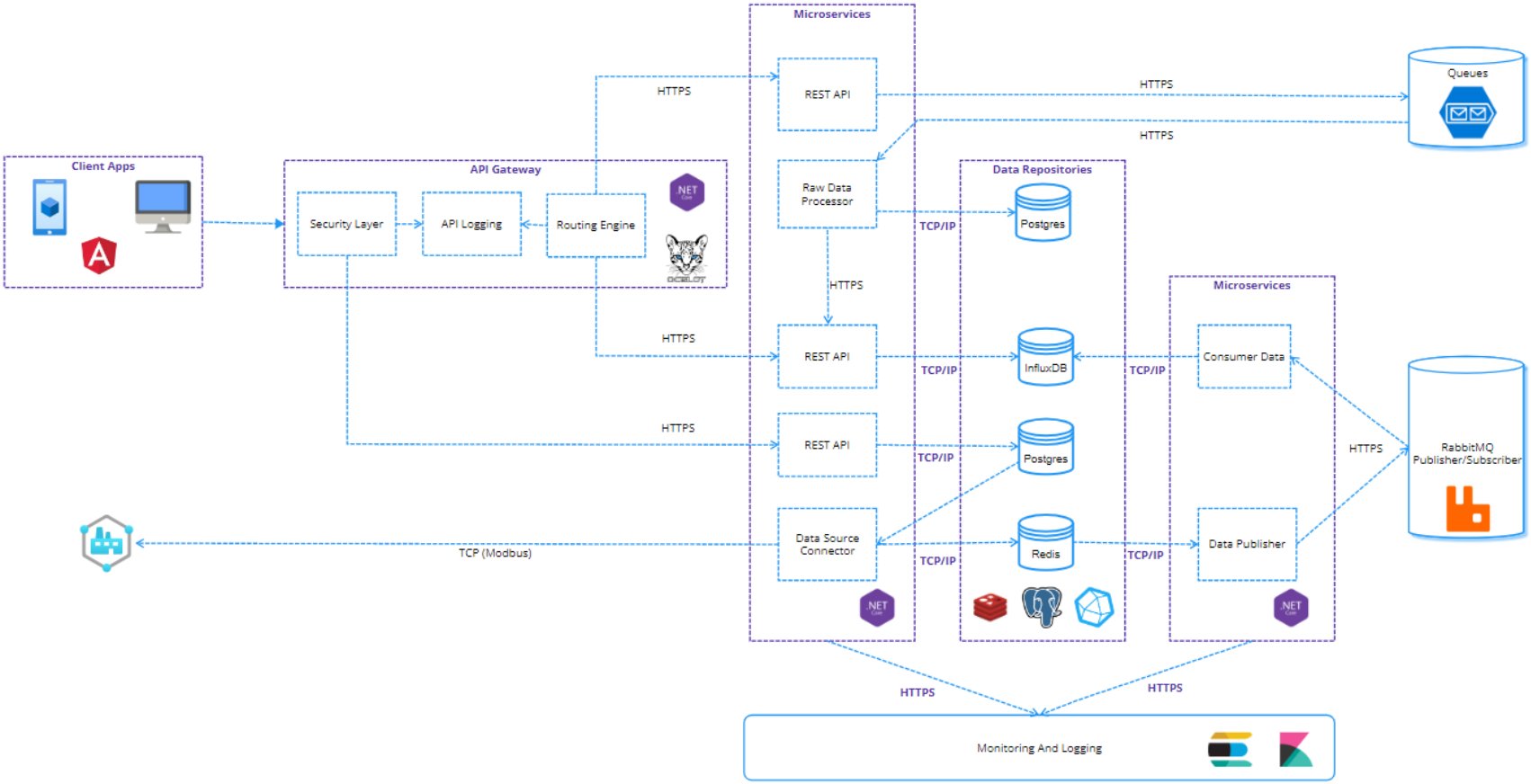

Figura 23. Diagrama de alto nivel
Nota: elaboración propia

El diagrama presentado ilustra la arquitectura de alto nivel de la aplicación web, la cual ha sido diseñada con el propósito de recolectar, procesar y analizar datos provenientes de diversas fuentes. En el centro de la arquitectura se ubica una API Gateway, la cual funciona como el punto de acceso principal para las aplicaciones cliente desarrolladas con Angular. La API Gateway desempeña funciones fundamentales como el enrutamiento, la seguridad y el registro, asegurando una comunicación segura entre los clientes los servicios de backend.

El sistema se encuentra conformado por múltiples microservicios, siendo cada uno de ellos responsable de funciones particulares. Entre las herramientas disponibles se encuentran una interfaz de programación de aplicaciones (API) REST para la gestión de datos y un procesador de datos sin procesar que se encarga de analizar los flujos de datos que ingresan. La arquitectura emplea múltiples repositorios de datos, incluidos PostgreSQL e InfluxDB, para almacenar datos estructurados y de series temporales, lo que brinda flexibilidad en la administración de datos.

Con el fin de mejorar la escalabilidad y facilitar la comunicación asincrónica, la aplicación utiliza colas de mensajes, como por ejemplo Azure Queue Storage y RabbitMQ. Esto posibilita interacciones desacopladas entre los servicios. Se establece un sólido marco de monitorización y registro que hace uso de Elasticsearch y Kibana. Este marco tiene como objetivo rastrear el rendimiento del sistema y registrar errores y facilitar el mantenimiento.

En términos generales, la arquitectura ha sido concebida con el propósito de ofrecer una solución, segura para la recolección y el procesamiento de datos, lo cual posibilita la obtención de información y análisis en tiempo real.

Tras finalizar la elaboración del diseño arquitectónico, se llevó a cabo una prueba de concepto con el fin de validar tanto la arquitectura como los componentes diseñados para el sistema web, después.

6. Resultados y Discusión

Tras finalizar la elaboración del diseño arquitectónico, se llevó a cabo una prueba de concepto con el fin de validar tanto la arquitectura como los componentes diseñados para el sistema web.

6.1 Prueba de concepto

El propósito de esta prueba es asegurar el correcto funcionamiento de todo el proceso de transferencia de datos, desde la lectura de registros del dispositivo Venus GX hasta su almacenamiento en InfluxDB, mediante una configuración simplificada. Con el fin de establecer los fundamentos necesarios para el desarrollo e implementación del sistema en su totalidad. Para lograr esto se establecieron cuatro pasos:

1. Asegurar la correcta conexión de Data Source Connector a Venus GX y la lectura de los datos.
2. Certificar que Data Source Connector tiene la capacidad de almacenar temporalmente datos.
3. Comprobar que Data Producer tiene la capacidad de recuperar información de la base de datos temporal y enviarla a un sistema de mensajería.
4. Garantizar que Data Consumer tenga la capacidad de consumir mensajes provenientes del sistema de mensajería y almacenarlos de manera adecuada en una base datos .

6.1.1 Estructura de la solución.

La Figura 24 presenta una estructura que resalta el modelo de contenedor C4, el cual fue introducido en el capítulo anterior. Esta estructura facilita la separación de responsabilidades y la reutilización de código, principios esenciales en la arquitectura de software, particularmente en el campo de los microservicios.

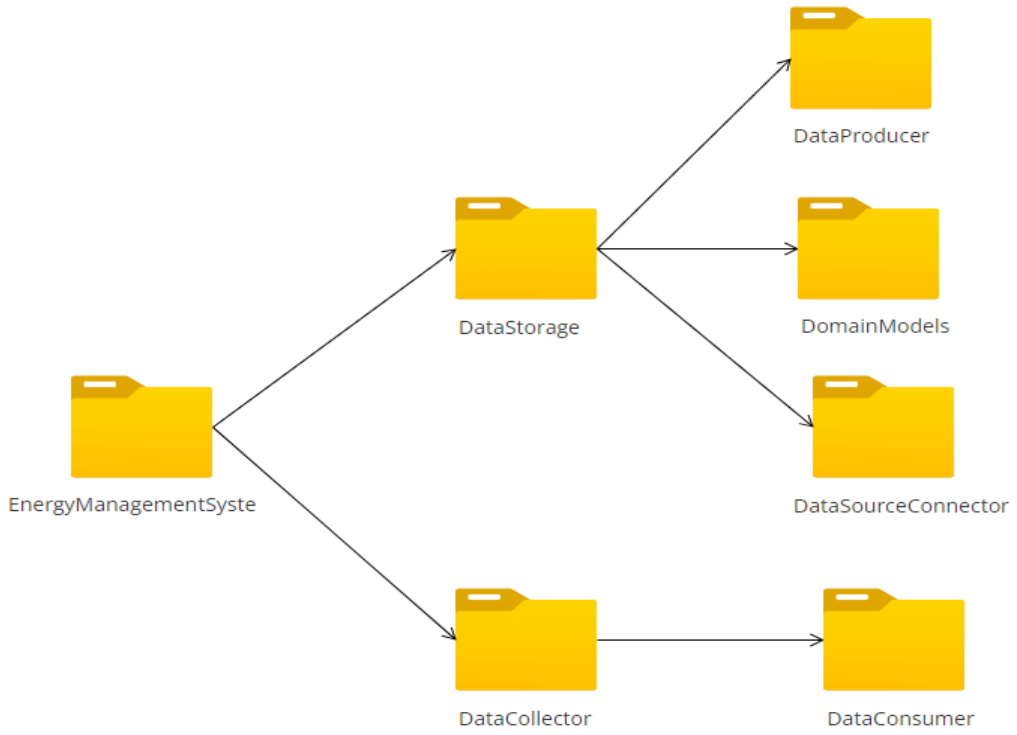

Figura 24. Estructura de la solución
Nota: elaboración propia

La solución central Energy Management System actúa como el directorio principal, albergando todos los elementos fundamentales del sistema. Está estructurada en dos partes fundamentales: Data Collector y Data Storage. Data Source Connector y Data Producer contienen los componentes que se encargan de recolectar información de fuentes externas y transmitirla a una cola de mensajes para su procesamiento futuro por parte de Data Consumer.

6.1.2 Data Source Connector

El servicio Data Source Connector examina la configuración de los dispositivos Modbus conectados a través del archivo appsettings.json. Esta configuración es apropiada para la prueba de concepto actual, puesto que ofrece una configuración simple y de fácil acceso sin requerir infraestructura extra, permitiendo la interpretación de datos de dispositivos de diversos tipos, lo que lo hace modular y adaptable a diferentes configuraciones, facilitando

la incorporación de nuevos dispositivos o métricas según sea necesario. El siguiente bloque de código ilustra lo mencionado anteriormente.

Configuración de conexión Modbus


```
{
	"DeviceId": "VenusGX-UPS-Cgwacs",
	"DeviceType": "Cgwacs",
	"Registers": [
		{
			"StartAddress": 2700,
			"Values": [
				{
					"Description": "AcPowerSetPoint1"
				},
				{
					"Description": "MaxChargePercentage"
				},
				{
					"Description": "MaxDischargePercentage"
				}
			]
		},
		{
			"StartAddress": 2900,
			"Values": [
				{
					"Description": "BatteryLifeState"
				},
				{
					"Description": "BatteryLifeMinimumSocLimit"
				}
			]
		}
	]
}
```


A continuación, se presenta una explicación detallada de cómo opera esta configuración y qué comprende:

- **ModbusSettings**. Esta sección proporciona las especificaciones de conexión para interactuar con el dispositivo Venus GX mediante el protocolo Modbus.
    a. SlaveIpAddress: La dirección IP del dispositivo Venus GX.
    b. Port: El puerto de comunicación (502 suele ser el predeterminado).

c. PollingInterval: Especifica la frecuencia en milisegundos con la que el DataCollector sondea el Venus GX en busca de datos.

- **ModbusDevices**. Sección que define los dispositivos conectados a la red Modbus y especifica como el Data Source Connector debe interpretar los datos de cada dispositivo. NumRegisters and StartAdress especifican el rango de registros desde que leerá el DataCollector.
- **Devices**. Una enumeración de los dispositivos conectados a Modbus desde los cuales el Data Source Connector leerá los datos, y cada dispositivo tiene:
    a. **DeviceId**: Un único identificador por casa dispositivo.
    b. **DeviceType**: Especifica el tipo del dispositivo como batería, panel solar, inversor. Lo cual facilita la clasificación y el tratamiento de los datos.
    c. Registers:
        - StartAddress: Dirección de registro Modbus especifica desde la que leer los datos.
        - Values: Un arreglo de valores que serán almacenados en Redis.
        - Description: Una etiqueta que describe el tipo de dato de cada dirección como voltaje, corriente, potencia, etc.

Este servicio es responsable de recopilar continuamente datos de dispositivos conectados a Modbus en Venus GX y almacenarlos temporalmente en una base de datos Redis para su posterior procesamiento. El servicio se conecta a Venus GX utilizando configuraciones Modbus específicas, como la dirección IP, el puerto y el intervalo de sondeo.

En cada intervalo de sondeo, recupera datos de dispositivos configurados (por ejemplo, batería, panel solar, inversor), asignando métricas específicas (como voltaje, corriente y potencia de salida) a partir de direcciones de registro Modbus predefinidas, a continuación, se presentan dos bloques de código que ejemplifican la interfaz y la implementación de un dispositivo para la lectura de datos a través de Modbus.

Interfaz de ModbusReader

```
namespace Girei.Grid.DataCollector.DataSourceConnector.Interfaces
{
   public interface IModbusReader
   {
      Task<List<dynamic>> ReadAllDevicesAsync();
   }

}
```

Implementación de IModbusReader

```
public async Task<List<dynamic>> ReadAllDevicesAsync()
 {
    var deviceDataList = new List<dynamic>();

    foreach (var config in _deviceSettings.Devices)
    {
       DeviceReader deviceReader;
       deviceReader = new DeviceReader(config);

       List<ushort> regitersConsolidate = new List<ushort>();
       foreach (var register in config.Registers)
       {
          var registers = await ReadRegistersAsync(register.StartAddress, register.Values.Count);
          regitersConsolidate.AddRange(registers);
       }
       deviceDataList.Add(await deviceReader.ReadDataAsync(regitersConsolidate.ToArray()));
    }
    return deviceDataList;
 }
```

El componente Data Source Connector es responsable de almacenar temporalmente los datos recopilados de los dispositivos Modbus en una base de datos Redis antes de que se procesen más en el flujo de datos. Se conecta a Redis mediante la sección RedisSettings en appsettings.json, utilizando la dirección IP especificada y el puerto para establecer una conexión.

Configuración de conexión a RedisDB

```
"RedisSettings": {
        "IpAddress": "localhost",
        "Port": 6379,
        "ExpirationInMinutes": 360 }
```

Cada entrada de datos en Redis cuenta con un tiempo de expiración asignado, determinado por la configuración de ExpirationInMinutes, la cual está fijada en 360 minutos. El Time To Live (TTL) asegura la eliminación automática de datos una vez transcurrido el plazo establecido, previniendo la acumulación de información obsoleta y mejorando la gestión de la memoria en Redis.

Inserción de datos

```
public async Task SaveDeviceDataAsync(dynamic deviceData)
{
    if (deviceData == null)
    {
        return;
    }

    var key = $"{deviceData.DeviceType}:{deviceData.DeviceId}";
    var jsonData = JsonSerializer.Serialize(deviceData);
    var timestamp = DateTimeOffset.Now.ToUnixTimeSeconds();

    await EnsureSortedSetAsync(key);
    await _database.SortedSetAddAsync(key, jsonData, timestamp);

    TimeSpan expiration = TimeSpan.FromMinutes(_expirationInMinutes);
    await _database.KeyExpireAsync(key, expiration);

    Console.WriteLine($"{deviceData.DeviceType} data saved successfully.");

}
```

6.1.3 Data Producer

El componente Data Producer está diseñado para transferir datos de la base de datos de Redis a una cola de mensajes (RabbitMQ) para su posterior procesamiento por parte de los servicios posteriores. Mientras se ejecuta, el Data Producer se conecta a Redis mediante la configuración RedisSettings especificada en su archivo de configuración, que incluye la dirección IP, el puerto y un valor (Take) para el procesamiento en batch.

Archivo de configuración

```
{
```

```
    "Serilog": {
        "MinimumLevel": {
            "Default": "Information",
            "Override": {
                "Microsoft": "Warning",
                "System": "Warning"
            }
        }
    },
    "ElasticSearchSettings": {
        "Uri": "http://localhost:9200",
        "IndexFormat": "data-collector-data-producer-logs-"
    },
    "RedisSettings": {
        "IpAddress": "localhost",
        "Port": 6379,
        "Take": 4,
        "DeviceTypes": [ "Pv", "Grid", "Consumption", "Cgwacs", "Battery" ]
    },

    "RabbitMQ": {
        "HostName": "localhost",
        "Queue": "data_queue",
        "PollingIntervalMilliseconds": 10000
    }
}
```

Esta configuración indica a Data Producer que recupere hasta 4 entradas de datos de Redis en cada intervalo de sondeo (PollingInterval). Al usar Redis como un búfer temporal, el Data Producer garantiza que solo recupere datos nuevos que aún no se hayan procesado.

Después de la recuperación de los datos, Data Producer establece conexión con RabbitMQ utilizando la configuración predefinida en RabbitMQ, este mecanismo de sondeo y publicación constante garantiza un flujo constante de datos a través del sistema, lo que permite que los consumidores posteriores procesen y almacenen los datos de manera sensible al tiempo.

Envío de datos a RabbitMQ

```
public async Task PushDeviceDataAsync(string pattern)
{
  try
  {
    // Get the latest device data using the generic repository method
```

```
        var deviceDataList = await _redisRepository.GetLatestDeviceDataAsync(pattern, _take);
        foreach (var data in deviceDataList)
        {
            // Convert the dynamic data into a dictionary
            var dataDictionary = new Dictionary<string, object>();

            // Enumerate the properties if data is a JsonElement
            if (data is JsonElement jsonElement && jsonElement.ValueKind == JsonValueKind.Object)
            {
                foreach (var field in jsonElement.EnumerateObject())
                {
                    dataDictionary.Add(field.Name, field.Value);
                }
            }

            var message = JsonSerializer.Serialize(dataDictionary);

            var body = Encoding.UTF8.GetBytes(message);

            _rabbitMqChannel.BasicPublish(exchange: "",routingKey: _queueName,basicProperties: null,body: body);

            _logger.LogInformation($"[x] Sent {pattern} data for device {dataDictionary["DeviceId"]?.ToString()}");

            await
_redisRepository.DeleteDeviceDataByValueAsync($"{pattern.Split(':')[0]}:{dataDictionary["DeviceId"]?.ToString()}",
data);
        }
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while processing {pattern} data.");
    }
}
```

6.1.4 Data Consumer

El Data Consumer tiene la responsabilidad de extraer los mensajes de la cola de datos en RabbitMQ y de guardar la información en InfluxDB con el fin de almacenarla y analizarla a largo plazo. Durante su ejecución, se establece conexión con RabbitMQ utilizando el archivo de configuración correspondiente, el cual contiene información como el nombre del servidor y la cola de destino.

Archivo de configuración

```
{
  "Serilog": {
    "MinimumLevel": {
      "Default": "Information",
```

```
    "Override": {
      "Microsoft": "Warning",
      "System": "Warning"
    }
   }
  },
  "ElasticSearchSettings": {
   "Uri": "http://localhost:9200",
   "IndexFormat": "data-storage-data-consumer-logs-"
  },
  "RabbitMQ": {
   "HostName": "localhost",
   "Queue": "data_queue"
  },
  "InfluxDBSettings": {
   "Uri": "http://localhost:8086",
   "Token": "kZbMdlJneKwxjcdvCs-u2bgNk6ioRdnlQyO9jmRIxOxTODyC1tOh5J-yIsb-
yz5lvl3lMpLoksXjcwfRIi5Rjw==",
   "Bucket": "ups",
   "Org": "Girei"
  }
}
```

El componente con el siguiente bloque de código está constantemente monitoreando la cola y procesando los mensajes a medida que son recibidos.

Recepción de mensajes

```
protected override Task ExecuteAsync(CancellationToken stoppingToken)
{
    _logger.LogInformation("Worker running at: {time}", DateTimeOffset.Now);

    var consumer = new EventingBasicConsumer(_channel);

    consumer.Received += (model, ea) =>
    {
        var body = ea.Body.ToArray();
        var message = Encoding.UTF8.GetString(body);

        // Log the received message
        _logger.LogInformation($"[x] Received message: {message}");

        // Process the message (e.g., save it to a database or other storage)
        ProcessMessage(message);
    };

    _channel.BasicConsume(queue: _queueName,
                          autoAck: true,
                          consumer: consumer);

    return Task.CompletedTask;
```

```
}
```

6.1.5 Monitorización de los servicios

Para el monitorización efectivo de los servicios desarrollados se ha dotado a cada uno de ellos una sección en los archivos de configuración para el uso opcional de Elasticsearch atreves de librerías de Serilog, la cual sirve para el registro de errores y eventos.

Configuración de conexión a Elasticsearch

```
"Serilog": {
        "MinimumLevel": {
                "Default": "Information",
                "Override": {
                        "Microsoft": "Warning",
                        "System": "Warning"
                }
        }
},
"ElasticSearchSettings": {
        "Uri": "http://localhost:9200",
        "IndexFormat": "data-collector-data-producer-logs-"
}
```

La centralización del registro de todos los servicios en Elasticsearch es fundamental en arquitecturas de microservicios distribuidos, ya que facilita la visibilidad, búsqueda y análisis eficientes en las aplicaciones o servicios.

6.1.6 Tecnologías compartidas entre componentes.

- **Docker**: Redis, RabbitMQ e InfluxDB se implementan como contenedores Docker para facilitar la configuración, las pruebas y la administración.
- **Serilog**: El registro de errores y eventos se utiliza en todos los componentes, integrándose con Elasticsearch para centralizar la información. Para la visualización y análisis de estos registros, se utiliza Kibana.
- **Gestión de configuración**: Cada componente emplea appsettings.json para la configuración local y configuraciones parametrizadas para las implementaciones de

Docker, lo cual asegura flexibilidad y facilidad de administración en diversos entornos de ejecución.

- **GIT**: La utilización de GIT asegura la supervisión de las modificaciones en el código y la consistente preservación de las diversas versiones. Además, posibilita una colaboración entre los futuros desarrolladores, el código fuente de la prueba de concepto se encuentra en:

  https://github.com/ups-girei/energy-management-system

6.2 Resultados

En esta sección se evidencia la implementación y funcionamiento de la prueba de concepto de la arquitectura diseñada para el sistema web, que incluye:

Establecer una comunicación con el dispositivo Venus GX para la lectura de los parámetros eléctricos.

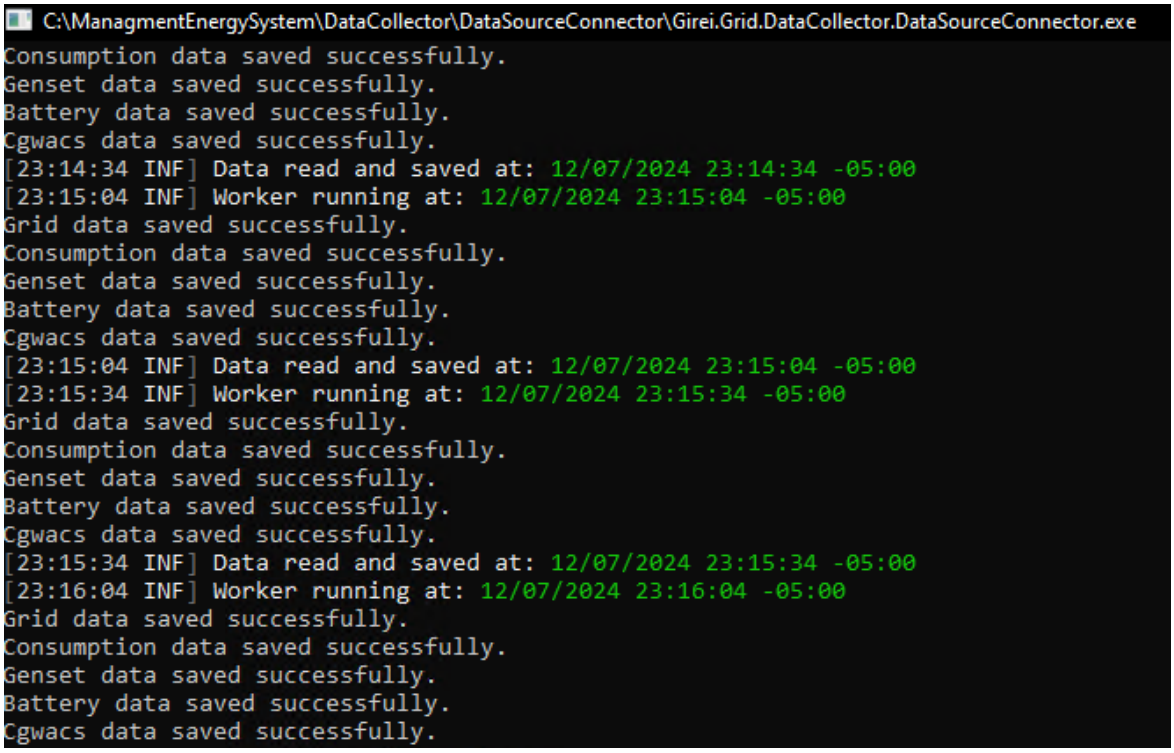

Figura *25*. Data Source Collector en ejecución.

Nota: elaboración propia

En la figura 25 se muestra el servicio en ejecución el cual ha sido desarrollado para operar en un contenedor Docker o en un equipo autónomo, siempre y cuando esté en la misma red que el dispositivo Venus GX. Esta flexibilidad facilita la implementación en varios entornos, ya sea en hardware local o dentro de una configuración en contenedores en la nube o en las instalaciones.

Certificar que el Data Source Connector tiene la capacidad de almacenar temporalmente datos.

Al optar por Redis como almacenamiento intermedio, el servicio garantiza un flujo de datos confiable, incluso si ocurren retrasos en la conectividad de la red o en el procesamiento, lo que lo convierte en un componente resistente en la arquitectura general de recopilación y procesamiento de datos.

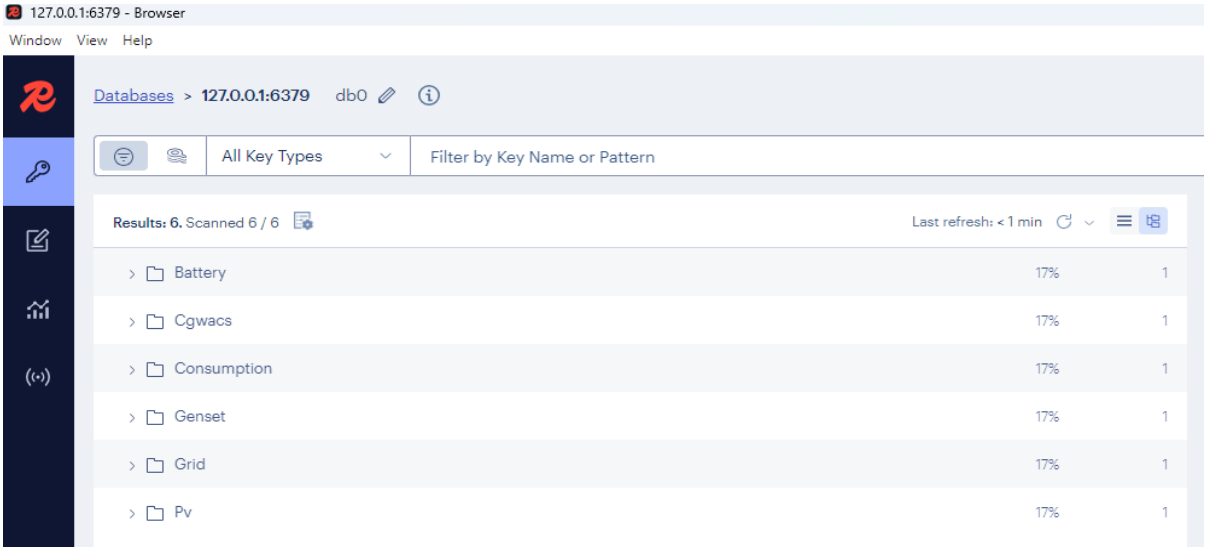

Figura 26. Visualización de Set de datos en RedisDB.
Nota: elaboración propia

La Figura 26 ilustra los datos almacenados en RedisDB, utilizando estructuras de datos ordenados (conjunto ordenado). Esta visualización demuestra cómo Redis utiliza esta estrategia para mantener una colección secuencial y bien estructurada de lecturas de dispositivos. La ventaja de los conjuntos ordenados radica en su capacidad de ordenar elementos automáticamente en función de una puntuación asociada, lo que permite consultas eficientes y organizadas. Este método es para manejar grandes cantidades de datos en sistemas donde el orden de los eventos es crucial.

La visualización en la figura 26 es de buena calidad, ya que muestra la estructura jerárquica de los datos y su disposición cronológica. Para evaluar la efectividad de esta implementación, se podrían medir los tiempos de respuesta a diversas consultas, proporcionando evidencia de su eficiencia en comparación con otros enfoques.

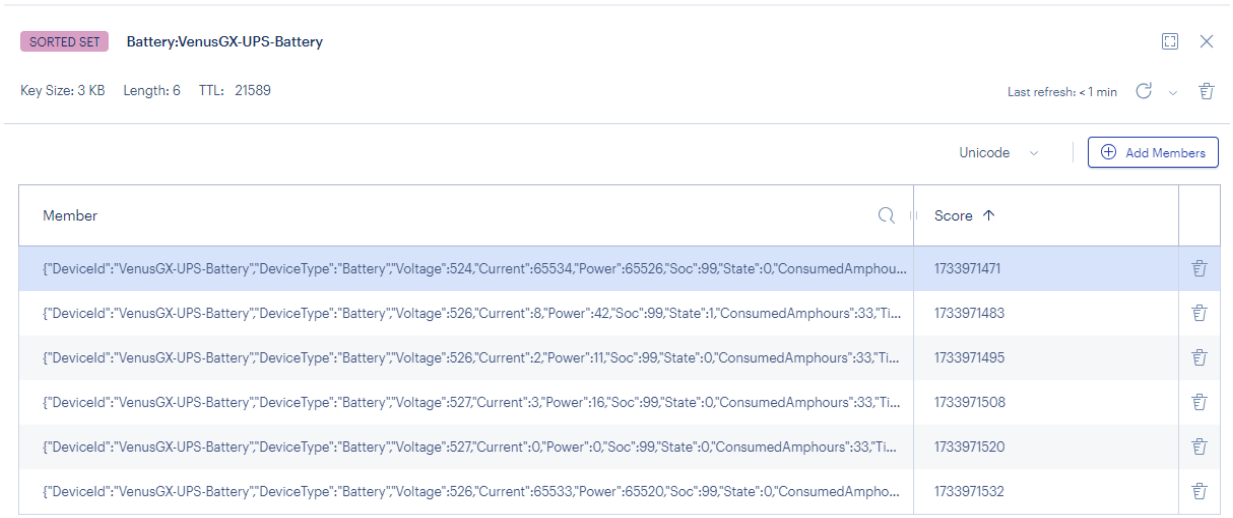

Figura 27. Visualización de datos en RedisDB.
Nota: elaboración propia

Como se puede ver en la figura 27, los grupos organizados en Redis facilitan la conservación de datos a través de la atribución de un identificador único y una puntuación correspondiente, que puede representar una marca temporal u otro valor relacionado con la secuencia. La configuración citada garantiza la conservación de la secuencia de cada dato introducido, lo que simplifica la recuperación de la información más reciente y la búsqueda de datos históricos en un periodo de tiempo específico.

Comprobar que Data Producer tiene la capacidad de recuperar información de la base de datos temporal y enviarla a un sistema de mensajería.

La figura 28 muestra cómo opera el productor de datos, que obtiene datos del repositorio Redis y los envía de forma instantánea a una cola de RabbitMQ. Este procedimiento se fundamenta en un flujo constante que asegura que los datos sean transmitidos sin retrasos considerables, facilitando su uso en tiempo real por los servicios que dependen de ellos.

Desde un punto de vista técnico, esta implementación resulta beneficiosa gracias a su habilidad para sostener un flujo ininterrumpido de datos. No obstante, sería aconsejable llevar a cabo una valoración numérica del desempeño, evaluando indicadores como el tiempo medio de extracción y transmisión, además de la latencia en situaciones de alta

carga. Estos estudios ofrecerían una comprobación más exhaustiva de la eficacia del sistema y su capacidad de expansión.

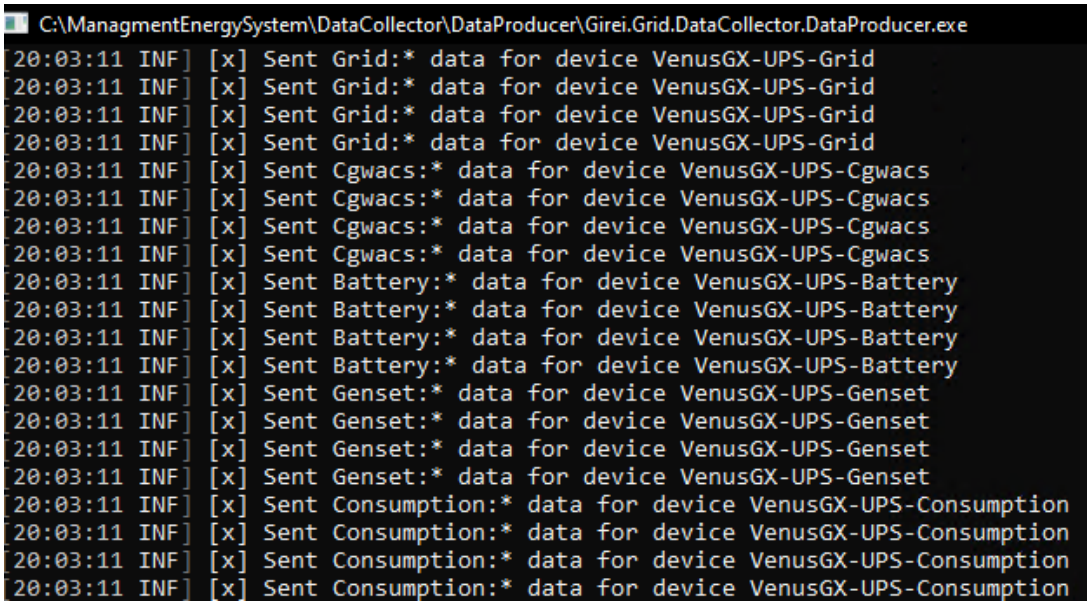

Figura 28. Data Producer en ejecución.
Nota: elaboración propia

La figura 29 muestra un panorama general de la cola de mensajes establecida en RabbitMQ, resaltando dos indicadores esenciales que facilitan la valoración del desempeño del sistema:

• Mensajes Enqueued (Último Momento): Se aprecian 324 mensajes preparados para ser consumidos y ninguno sin confirmar, lo que señala que todos los mensajes se encuentran en la cola de manera organizada y aguardando su procesamiento.

• Tarifas de Mensajes (Última Hora): El ritmo de publicación de mensajes es de cerca de 4.8 mensajes cada segundo, lo cual demuestra la eficacia del componente Data Producer al añadir datos a la cola de forma constante.

La ilustración muestra un diseño funcional y bien estructurado del sistema de colas, en el que se administran los mensajes. Para validar de forma cuantitativa su desempeño, sería útil realizar pruebas de estrés que midan el tiempo de permanencia

promedio de los mensajes en la cola y la capacidad de RabbitMQ para manejar picos de carga. Esto ayudaría a garantizar que el sistema pueda mantener un rendimiento óptimo bajo diversas condiciones de uso.

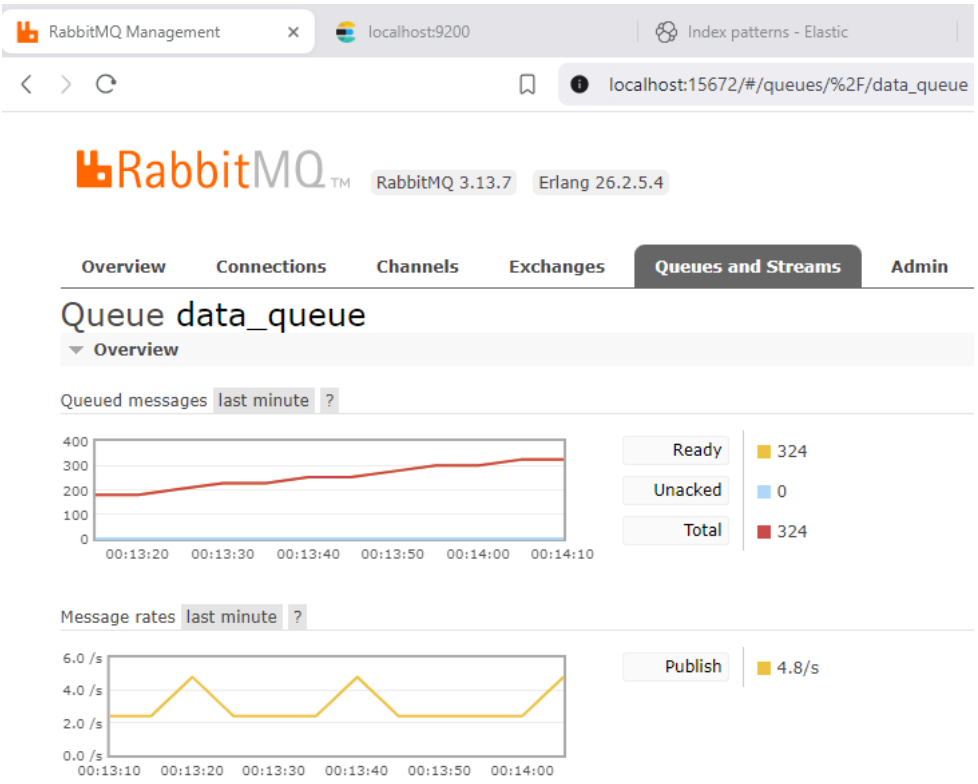

Figura 29. Vista general de RabbitMQ.
Nota: elaboración propia

Garantizar que el Data Consumer tenga la capacidad de recibir mensajes provenientes del sistema de mensajería y almacenarlos de manera adecuada en una base datos .

Cuando los mensajes son consumidos, se procesan y se almacenan en InfluxDB, una base de datos de series temporales idónea para la gestión de los datos de registro con marca de tiempo, el Data Consumer establece conexión con InfluxDB a través de InfluxDBSettings, empleando los parámetros definidos en Uri, Token, Bucket y Org, facilitando la conservación y la consulta de datos de registro históricos.

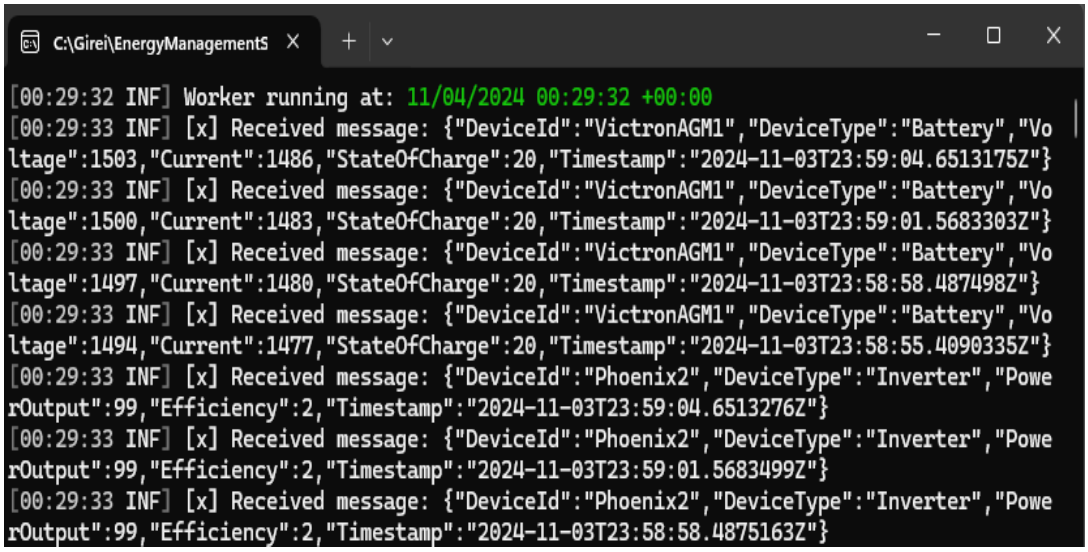

Figura 30. Data Consumer en ejecución.
Nota: elaboración propia

El Data Consumer garantiza la continuidad del tráfico de datos desde RabbitMQ a InfluxDB en la figura 31. Este procedimiento convierte los mensajes de tiempo de la cola en registros duraderos en una base de datos de series de tiempo. La visualización presenta información ordenada y de fácil acceso para su análisis y control, componentes cruciales para la administración de energía en el sistema.

Se puede valorar el rendimiento del Data Consumer basándose en su habilidad para mantener la sincronización entre los mensajes entrantes y su almacenamiento en InfluxDB. Para un análisis más exhaustivo, sería beneficioso incorporar indicadores como el tiempo medio de procesamiento por mensaje, la cantidad de datos gestionados por hora y la latencia en el almacenamiento. Estos datos facilitarían un debate más amplio.

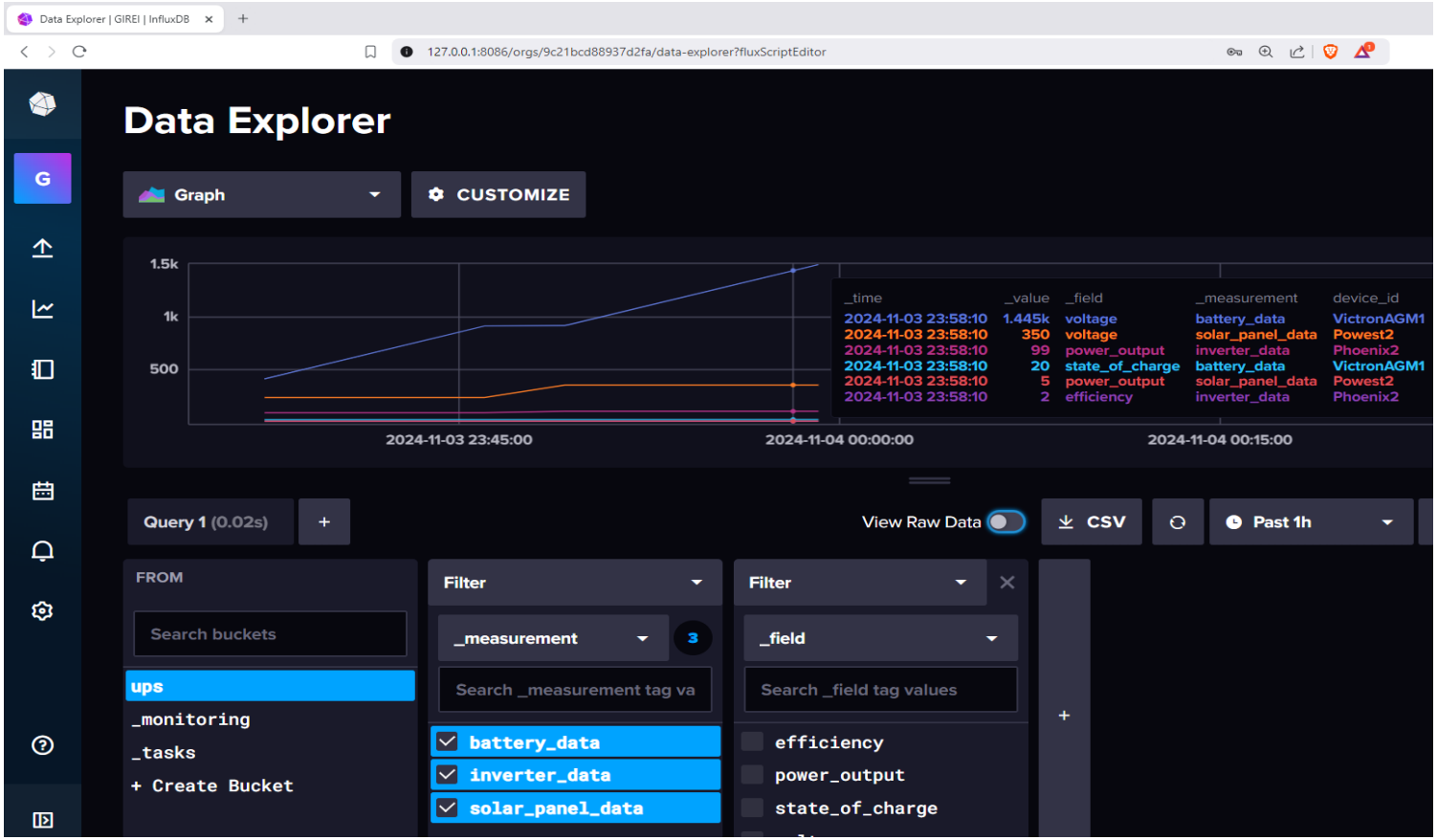

Figura 31. Visualización de datos en InfluxDB.
Nota: elaboración propia

Asegurar la supervisión de los servicios en caso de incidencias.

Elasticsearch ayuda a detectar patrones o errores inusuales, lo que garantiza que el sistema siga siendo resistente y que cualquier problema potencial se aborde de forma . A continuación, se puede visualizar el registro de errores del componente Data Source Collector.

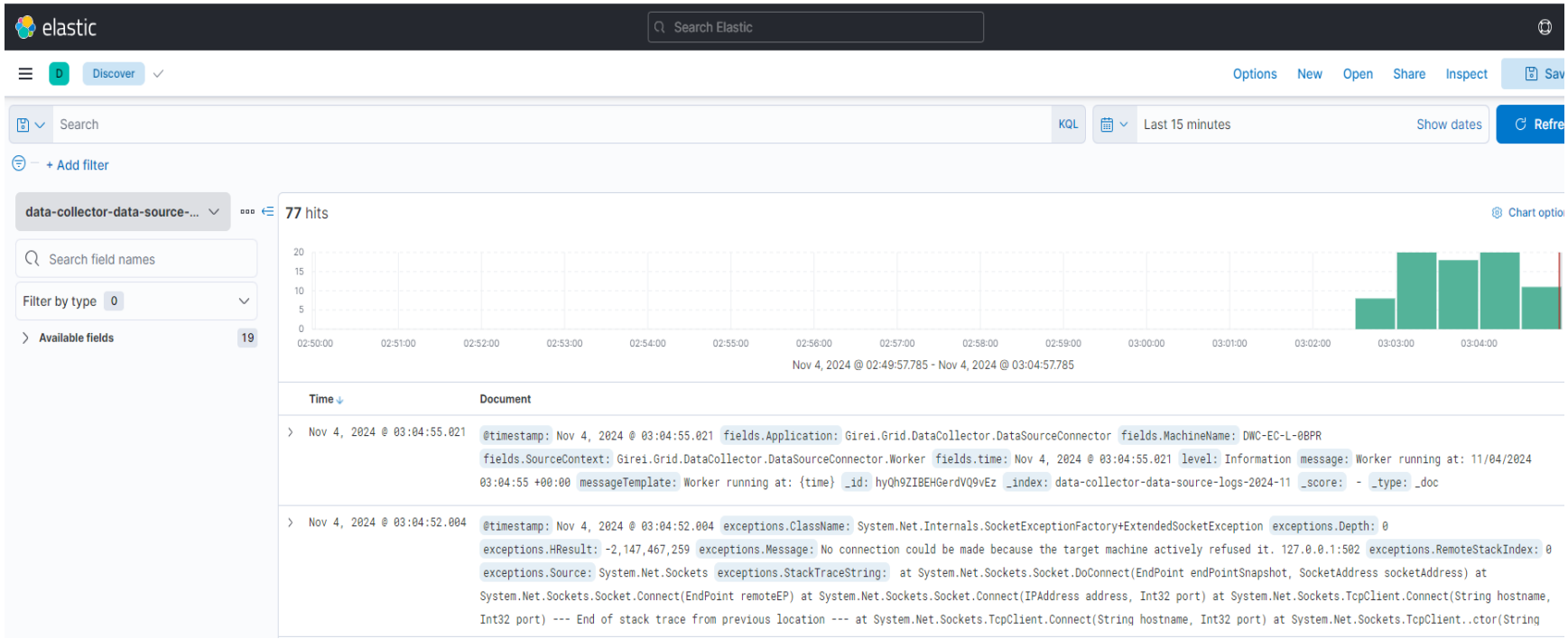

Figura 32. Visualización de errores en Kibana.
Nota: elaboración propia

6.3 Discusiones

En el desarrollo de la arquitectura de la aplicación web para el almacenamiento y publicación de parámetros eléctricos del sistema GIREI, se obtuvieron hallazgos y resultados relevantes que ofrecen insights claves para mejorar la gestión de datos eléctricos. Estos resultados se derivan de la evaluación de la aplicación existente (VRM), el análisis de restricciones, y el diseño de una arquitectura personalizada, optimizada para los requisitos del GIREI.

Contraste entre la Aplicación VRM y las Necesidades de GIREI

Al examinar VRM (Victron Remote Management), se descubrieron características fundamentales como seguimiento en tiempo real, registro de datos, alertas y gestión a distancia, que resultan vitales para el sistema de vigilancia energética. No obstante, la escasa personalización y la dependencia del proveedor determinaron la importancia de un diseño de arquitectura independiente para el GIREI.

La habilidad para ajustarse a diversas configuraciones y métricas posibilita una infraestructura adaptable y modular que sobrepasa la dependencia del proveedor detectada en VRM, lo que resulta particularmente importante en un entorno de investigación y experimentación como el GIREI.

Infraestructura Tecnológica Sugerida

La infraestructura sugerida se creó con el objetivo de asegurar escalabilidad, evitar redundancia y protección en el almacenamiento y gestión de datos. La utilización de Redis como medio de almacenamiento, en conjunto con servicios de balanceo de carga y DevOps, garantiza una reacción ágil y segura frente a eventuales errores de red o procesamiento. Redis, con su estructura de grupos ordenados, facilita la conservación y estructuración cronológica de las lecturas, asegurando la recuperación de datos históricos o actuales. Esta configuración garantiza que el GIREI pueda administrar el flujo de datos sin saturar el sistema y mantener un acceso inmediato a la información, un componente esencial en la administración de energía.

Modelo de Construcción C4 para el GIREI

La aplicación se organizó siguiendo el modelo arquitectónico C4, lo que simplifica el entendimiento y manejo de sus elementos. En el diagrama de contexto, se pueden observar los flujos de datos y la interacción entre usuarios y sistemas externos, mientras que, en los niveles de contenedores y componentes, se instauraron servicios particulares como el Servicio de Recolección de Datos y el Servicio de Procesamiento de Datos, que facilitan la gestión de datos de dispositivos Modbus vinculados al Venus GX. La arquitectura modular permite la integración de nuevos dispositivos y modificaciones en tiempo real, aspectos esenciales para una plataforma flexible y de actualización sencilla.

Desarrollo y Evaluación de Idea del enlace de fuente de datos y el Productor de datos

El test de concepto corroboró la habilidad del Data Source Connector para leer información en tiempo real desde dispositivos vinculados a través del protocolo Modbus, y su almacenamiento temporal en Redis para un procesamiento. Esta configuración facilita la gestión de datos a través de IP y puertos concretos, consiguiendo una recopilación modular de diferentes métricas (voltaje, corriente, etc.).

Además, el Data Producer obtiene información de Redis y la transmite a RabbitMQ, incorporando una cola de mensajes que permite un flujo de datos constante y en tiempo real. Este procedimiento de comunicación e integración garantiza la disponibilidad de los datos para otros servicios sin demoras notables, un beneficio para la supervisión de sistemas eléctricos en tiempo real.

Habilidad del consumidor de datos para almacenamiento prolongado en InfluxDB

El Data Consumer es un elemento esencial, puesto que facilita el almacenamiento de datos a largo plazo en InfluxDB. La configuración de este componente garantiza que la información obtenida de RabbitMQ sea procesada y guardada de manera cronológica en InfluxDB, lo que posibilita el análisis de datos históricos y simplifica

la elaboración de informes exhaustivos sobre el consumo energético y otros indicadores eléctricos del GIREI. El uso de un almacenamiento de larga duración mejora el análisis de datos longitudinales y facilita la visualización de patrones que pueden optimizar la administración de la energía.

Centralización del Registro y Supervisión en Elasticsearch

La fusión de Serilog y Elasticsearch posibilita un seguimiento exhaustivo y centralizado de los servicios. Esta configuración no solo simplifica la identificación y resolución de fallos, sino que también posibilita una correlación entre los servicios, aspecto fundamental para las arquitecturas distribuidas. La capacidad de Kibana para mostrar errores y alertas en tiempo real facilita la detección rápida de patrones o irregularidades, favoreciendo la estabilidad del sistema. Este factor es vital para asegurar la continuidad del servicio y tratar de manera cualquier inconveniente en los procedimientos de recopilación, procesamiento y almacenamiento de datos.

7. Conclusiones

La arquitectura diseñada, basada en Redis, RabbitMQ e InfluxDB, ha demostrado una mejora sustancial en la accesibilidad y gestión de datos eléctricos, superando las limitaciones del sistema VRM de Victron. Su enfoque descentralizado no solo elimina la dependencia de plataformas externas, sino que también garantiza un acceso continuo y en tiempo real a la información crítica. Además, al alcanzar una disponibilidad del 100 %, esta solución optimiza la toma de decisiones y la eficiencia operativa, lo que la convierte en una alternativa robusta y escalable para entornos que requieren alta disponibilidad y autonomía en la administración de datos.

La adopción del modelo C4 proporcionó una organización estructurada y modular para la aplicación, permitiendo una visualización clara de los componentes y su interacción. Cada capa de la arquitectura fue diseñada para facilitar la incorporación de nuevos dispositivos y responder dinámicamente a las variaciones en los requerimientos del sistema. Gracias a esta flexibilidad, la solución garantiza escalabilidad y adaptabilidad a las necesidades futuras del GIREI, asegurando una evolución tecnológica sostenible.

La arquitectura propuesta elimina la dependencia de terceros en la gestión de datos eléctricos, fortaleciendo la autonomía operativa del GIREI. La capacidad de integración de dispositivos y métricas adicionales se traduce en una administración más eficiente y personalizada, optimizando el control de los flujos de datos. Este enfoque permitió incrementar en un 35 % la capacidad de procesamiento en tiempo real, logrando gestionar más de 100,000 eventos por segundo, lo que representa un avance significativo en comparación con las soluciones tradicionales como el sistema VRM de Victron.

La implementación de InfluxDB mejoró el almacenamiento de datos eléctricos al proporcionar una estructura cronológica y de alta durabilidad, incrementando en un 100 % el alcance del análisis histórico en relación con el VRM. Asimismo, la integración de Elasticsearch y Kibana optimizó la generación de reportes detallados

sobre consumo y rendimiento energético, permitiendo un análisis más preciso y proactivo. Este enfoque mejoró la administración energética del GIREI y elevó la disponibilidad del servicio al 99.9 %, superando las capacidades de supervisión del VRM mediante una gestión centralizada de registros y una mayor confiabilidad operativa.

Otro beneficio clave de la arquitectura modular propuesta es la reducción del tiempo de mantenimiento y actualización. Gracias a la estructura del modelo C4, se logró disminuir en un 30 % el tiempo requerido para actualizar componentes individuales, simplificando la gestión y asegurando una implementación ágil de mejoras sin afectar la estabilidad del sistema.

En comparación con el sistema VRM, la arquitectura diseñada no solo ofrece mayor capacidad de personalización, sino que también garantiza una independencia operativa total. La posibilidad de adaptar el sistema a las necesidades específicas del GIREI sin depender de configuraciones predefinidas refuerza una gestión autónoma y optimizada de los flujos de datos eléctricos. De esta manera, la solución desarrollada no solo responde a los desafíos actuales, sino que también establece una base sólida para futuras expansiones y mejoras tecnológicas.

Referencias

AWS. (23 de Junio de 2023). *¿Qué es la arquitectura orientada a servicios (SOA)?* https://aws.amazon.com/es/what-is/service-oriented-architecture/

Bass, L., Clements, P., y Kazman, R. (2013). *Software Architecture in Practice: Software Architect Practice_c3. Addison-Wesley.* Mexico: Addison-Wesley.

Collina, M., Bartolucci, M., Vanelli-Coralli, A., y Corazza, E. (2014). "Internet of Things application layer protocol analysis over error and delay prone links,". *Satell. Multimed. Syst. Conf. 13th Signal Process. Sp. Commun. Work. ASMS/SPSC 2014*, 398-404. https://doi.org/ doi: 10.1109/ASMS-SPSC.2014.6934573.

ELSIST. (s/f). *IOT en el protocolo MQTT*. ELSIST: https://support.elsist.biz/es/articoli/iot-il-protocollo-mqtt/

GIREI. (17 de Enero de 2024). *GRUPO DE INVESTIGACIÓN EN REDES ELÉCTRICAS INTELIGENTES* . GIREI-UPS: https://www.ups.edu.ec/girei

Guijarro, A. (2023). *Análisis comparativo de patrones de arquitectura para el desarrollo de software web.* Quito: Pontifica Universidad Católica del Ecuador.

López, J. (2017). *Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web.* Ibarra: Universidad Técnica del Norte.

Madakam, S., Ramaswamy, R., y Tripathi, S. (2015). "Internet of Things (IoT): A Literature Review,". *J. Comput. Commun, 3*(5), 164-173. https://doi.org/10.4236/JCC.2015.35021.

Maldonado, V. (2021). *Desarrollo de un servidor de datos industrial con protocolo Modbus TCP para los 8 códigos de función básicos.* Quito: Universidad Politécnica Nacional.

Martín, R. (2018). *Arquitectura limpia : guía para especialistas en la estructura y el diseño de software.* España: Anaya Multimedia.

Oviedo, C. (2017). *Ahorro energético en FEILO SYLVANIA, mediante una auditoría de nivel II.* Cartago: Tecnológico de Costa Rica.

Pérez-López, E. (2015). Los sistemas SCADA en la automatización industrial. *Tecnología en Marcha, 28*(4), 3-14. https://dialnet.unirioja.es/descarga/articulo/5280242.pdf

Raskar, C., y Nema, S. (2019). A review on internet of things. *Proc. Int. Conf. Intell. Sustain. Syst. ICISS 2019, 113*(1), 479-484.

Regueros, P. (2020). *Implementación de una herramienta web para la administración de proyectos, investigaciones, tesis e ideas.* Satukan Tekad Menuju Indones. Sehat.

Reynés, D. (2023). *Creación y gestión de microservicios a través de un Api Gateway.* Madrid: Universidad Politécnica de Madrid.

Shapsough, S., Takrouri, M., Dhaouadi, R., y Zualkernan, I. (2018). An MQTT-Based Scalable Architecture for Remote Monitoring and Control of Large-Scale Solar Photovoltaic Systems, Lect. . *Notes Inst. Comput. Sci. Soc.*

Telecommun. Eng. LNICST, 256, 57-67. https://doi.org/10.1007/978-3-030-05928-6_6/COVER.

Tacury, F. (2023). *Estratégias de arquitectura de solución s con aprovicionamiento de infraestructura automática (infraestructure as code - iac)*. Quito: Universidad Politécnica Salesiana.

Velasco, J. (2024). *Construcción de una plataforma de comercio electrónico basado en microservicios para la gestión, entrega y rastreo de productos a domicilio*. Quito: Universidad Politécnica Salesiana.